Function Reference Manual - **HAL API**

HAL-API for software development with **m:explore** ultra-wideband sensors

Ilmsens GmbH
Ehrenbergstr. 11
98693 Ilmenau
Germany

Tel: +49 3677-76130-30
Fax: +49 3677-76130-39
Web: www.ilmsens.com
Email: hal-api@ilmsens.com

Contents

1 Function Reference Manual - HAL API

HAL-API for software development with **m:explore** ultra-wideband sensors

1.1 Introduction

As a service to our customers who want to integrate Ilmsens UWB sensors into their software environment, Ilmsens offers a hardware abstraction layer (**HAL**) application programming interface (**API**) for the **m:explore** ultra-wideband (UWB) sensors ("the software"). The HAL API comes in form of a dynamic library working on top of the device drivers with corresponding C header files. This is the function reference for the API.

The HAL API is available for different operating systems and allows device management, sensor configuration, acquisition configuration, and performing measurements. It abstracts from device- or digital interface-specific details as much as possible to enable portability of the application software. Future generations of the HAL API and Ilmsens UWB sensors will be developed with backward compatibility in mind. Product-specific extensions will extend the API rather than changing existing functions.

By default, the HAL API is provided as software in binary (compiled) form for many popular operating systems. Should the provided functionality be insufficient for the customer's purposes, please contact Ilmsens for other options (see below). We provide limited software support for the HAL API. To report problems, ask for help, or suggest improvements, contact Ilmsens (see below).

1.2 Copyright and Disclaimer

1.3 Further documentation

This is a function reference only. Ilmsens provides additional documentation in various guides.

- **HAL API Programming Guide**

  A software developer can find design and basic background information in the programming guide to ease application development. Data format, buffer layout, and typical data processing steps are explained as well.

- **HAL API Setup Guide**

  The setup guide explains how to install the HAL library, the dependencies, and how to test successful installation.

1.4 Third party software

The HAL library makes use of great software developed by open source communities. These modules and libraries allow us to make the HAL library available on many platforms and operating systems. We appriciate this work very much and would like to thank the contributors of the following projects:

1.4.1 The libusb project

Copyright

We are using libusb-1.0 for communication with our sensors. The library is available on many Linux flavours and most modern Windows platfoms (where it makes use of the built-in WinUSB drivers thus avoiding the need for proprietary driver development).

Check out these ressources:

- Project website: http://libusb.info
- License: check the latest link on the project website. Currently the license is the GNU Lesser General Public License, version 2.1

1.4.2 The POCO project

Copyright

POCO provides a very powerful multi-threaded framework for logging messages, errors, etc. It can be setup in a hierarchical manner and allows fine grain control of what gets logged to where. We are using libpocofoundation-1.7 for logs from the HAL library.

Check out these ressources:

- Project website: pocoproject.org
- License: The Boost Software License 1.0

1.5 Third party tools

The development of the HAL library is done with the following great and free tools. They help a lot in keeping the library consitent, easy to build, and the documentation up to date. We appriciate these tools very much and would like to thank their contributors.

Check out these tools:

- Version control system: Git
- Building and Packaging: CMake
- Code documentation: Doxygen

1.6 Contact Ilmsens

Ilmsens GmbH
Ehrenbergstr. 11
98693 Ilmenau
Germany

Tel.: +49 3677 76130-30
Fax: +49 3677 76130-39
Email: hal-api@ilmsens.com

2 Measurement Data Format

This section briefly explains the format and buffer layout for measured data returned by the HAL API.

More detailed information can be found in the **HAL API Programming Guide**.

2.1 Data format of raw measured data

The data retrieved from the sensors is returned as 32 bit integers (**int**) to avoid any loss of precision while keeping the amount of data as small as possible. Besides the actual signal samples, additional status information is contained in the output buffer. How to convert raw integers into physical units is explained in detail in the **HAL API Programming Guide**.

2.2 Calculating the output-buffer size

This section describes how to calculate the minimum size of the output buffer for retrieving measured data for a given sensor configuration.

- In case of sensors with different configurations included in the measurement run

$$\begin{aligned} BufferSize_{min} &= \sum_i ChannelSize(i) \times NumberOfRx(i) \\ &= \sum_i 2^{MLBS-order(i)} \times NumberOfOversampling(i) \times NumberOfRx(i) \end{aligned}$$

- In case of common configuration for all sensors included in the measurement run

$$\begin{aligned} BufferSize_{min} &= NumberOfSensors \times ChannelSize \times NumberOfRx \\ &= NumberOfSensors \times 2^{MLBS-order} \times NumberOfOversampling \times NumberOfRx \end{aligned}$$

2.3 Layout example of output buffer

The following layout assumes a measurement with two indentically configures 9th order **m:explore** sensors.

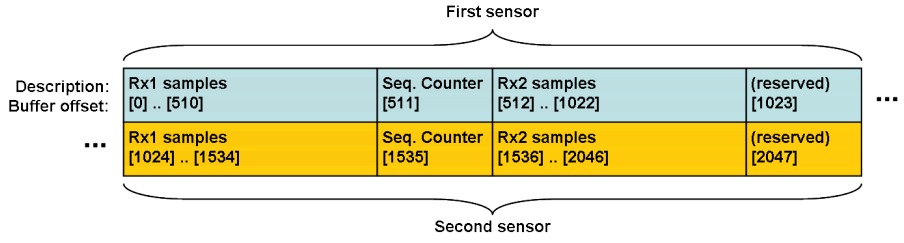

Figure 1 Buffer layout of example setup with two measuring 9th order m:explore sensors

3 Module Index

3.1 Modules

Here is a list of all modules:

4 Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

5 File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

6 Module Documentation

6.1 Ilmsens Error Codes

Ilmsens error codes provided as return values.

Enumerations

- enum ilmsens_error {
  ILMSENS_SUCCESS = 0, ILMSENS_ERROR_INVALID_PARAM = -1, ILMSENS_ERROR_STATE = -2, I↩
  LMSENS_ERROR_BUSY = -3,
  ILMSENS_ERROR_ACCESS = -4, ILMSENS_ERROR_IO = -5, ILMSENS_ERROR_NO_MEMORY = -6, I↩
  LMSENS_ERROR_AGAIN = -7,
  ILMSENS_ERROR_TIMEOUT = -8, ILMSENS_ERROR_NOT_SUPPORTED = -9, ILMSENS_ERROR_U↩
  NKNOWN = -99 }

  Common error codes.

6.1.1 Detailed Description

Ilmsens error codes provided as return values.

6.1.2 Enumeration Type Documentation

6.1.2.1 ilmsens_error

`enum ilmsens_error`

Common error codes.

Enumerator

ILMSENS_SUCCESS	No error.
ILMSENS_ERROR_INVALID_PARAM	Invalid parameter.
ILMSENS_ERROR_STATE	Invalid state (e.g. on reading data, when no measurement is running).
ILMSENS_ERROR_BUSY	Ressource is busy.
ILMSENS_ERROR_ACCESS	Access denied.
ILMSENS_ERROR_IO	Input/Output error (e.g. transmission error or device disconnected).
ILMSENS_ERROR_NO_MEMORY	Ressource allocation failed.
ILMSENS_ERROR_AGAIN	Not enough data, try again later (e.g. when a non-blocking operation would need to block the caller).
ILMSENS_ERROR_TIMEOUT	Timeout expired.
ILMSENS_ERROR_NOT_SUPPORTED	Operation not supported.
ILMSENS_ERROR_UNKNOWN	Unspecified error.

6.2 Library initialization/deinitialization

Modules

- Library version and revision
- Diagnostic interface

Functions

- int ilmsens_hal_initHAL (void)

  Initializes the library.
- void ilmsens_hal_deinitHAL (void)

  Deinitializes the library.

6.2.1 Detailed Description

6.2.2 Function Documentation

6.2.2.1 ilmsens_hal_initHAL()

```
int ilmsens_hal_initHAL (
             void  )
```

Initializes the library.

This function must be called before calling any other library function.

Returns

number of connected devices
negative error-code

6.2.2.2 ilmsens_hal_deinitHAL()

```
void ilmsens_hal_deinitHAL (
             void  )
```

Deinitializes the library.

Should be called after closing all open devices and before your application terminates. This function always succeeds. Errors that appeared, will only be visible when calling ilmsens_hal_initHAL() again (i.e. if the library is not unloaded but a new session is started).

6.3 Library version and revision

Data Structures

- struct ilmsens_hal_Version

  Represents the version identifier.

Macros

- #define ILMSENS_HAL_API_VER_MAJOR 1

  major version number of HAL API
- #define ILMSENS_HAL_API_VER_MINOR 1

  minor version number of HAL API
- #define ILMSENS_HAL_API_VER_BUILD 1

  build number of HAL API
- #define ILMSENS_HAL_API_VER (ILMSENS_HAL_API_VER_MAJOR ∗ 100000 + ILMSENS_HAL_API↩ _VER_MINOR ∗ 1000 + ILMSENS_HAL_API_VER_BUILD)

  single version number for preprocessors

Functions

- int ilmsens_hal_getVersion (struct ilmsens_hal_Version ∗pVersion)

  Return the HAL version.

6.3.1 Detailed Description

6.3.2 Function Documentation

6.3.2.1 ilmsens_hal_getVersion()

```
int ilmsens_hal_getVersion (
            struct ilmsens_hal_Version * pVersion )
```

Return the HAL version.

Parameters

pVersion	pointer to version information structure

Returns

ILMSENS_SUCCESS on success
negative error-code

6.4 Diagnostic interface

Macros

- #define ILMSENS_DEB_NO 0U

  do not output debug/info messages
- #define ILMSENS_DEB_INFO 1U

  only output errors and very important information
- #define ILMSENS_DEB_MORE 2U

  output errors, warnings, and important information
- #define ILMSENS_DEB_MOST 3U

  output errors, warnings, and debug information
- #define ILMSENS_DEB_ALL 4U

  output errors, warnings, and trace information

Functions

- int ilmsens_hal_setDEBLevel (unsigned int pLevel)

  Sets the verbosity of the diagnostics output Diagnostics are sent to std.

6.4.1 Detailed Description

6.4.2 Function Documentation

6.4.2.1 ilmsens_hal_setDEBLevel()

```
int ilmsens_hal_setDEBLevel (
            unsigned int pLevel )
```

Sets the verbosity of the diagnostics output Diagnostics are sent to std.

error output by default. In case of an error, the current level is not changed.

Parameters

pLevel	verbosity level from ILMSENS_DEB_NO to ILMSENS_DEB_ALL

Returns

current debug level (on success and error)

6.5 Device handling

Modules

- Device ID & properties

  Functions for identification and access to hardware-parameters of connected devices.

Functions

- int ilmsens_hal_openSensors (unsigned int *pDevNums, unsigned int pNum)

  Allocates specified devices for a measurement session.
- void ilmsens_hal_closeSensors (unsigned int *pDevNums, unsigned int pNum)

  Releases specified devices.

6.5.1 Detailed Description

6.5.2 Function Documentation

6.5.2.1 ilmsens_hal_openSensors()

```
int ilmsens_hal_openSensors (
            unsigned int * pDevNums,
            unsigned int pNum )
```

Allocates specified devices for a measurement session.

This function must be called before doing the configuration or starting a measurement session.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements

Returns

ILMSENS_SUCCESS on success
negative error-code

6.5.2.2 ilmsens_hal_closeSensors()

```
void ilmsens_hal_closeSensors (
            unsigned int * pDevNums,
            unsigned int pNum )
```

Releases specified devices.

Should be called after the last function call to any of the specified devices.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements

6.6 Device ID & properties

Functions for identification and access to hardware-parameters of connected devices.

Data Structures

- struct ilmsens_hal_ModInfo

  Represents the device configuration & runtime parameters and limits.

Macros

- #define ILMSENS_HAL_MOD_ID_BUF_SIZE 1024

  maximum length buffer size for retrieving the unique ID string.

Functions

- int ilmsens_hal_getModId (unsigned int pDevNum, char ∗pID, size_t pBufferSize)

  Gets unique device-identifier.
- int ilmsens_hal_getModInfo (unsigned int pDevNum, struct ilmsens_hal_ModInfo ∗pInfo)

  Gets device hardware-configuration.

6.6.1 Detailed Description

Functions for identification and access to hardware-parameters of connected devices.

6.6.2 Function Documentation

6.6.2.1 ilmsens_hal_getModId()

```
int ilmsens_hal_getModId (
            unsigned int pDevNum,
            char * pID,
            size_t pBufferSize )
```

Gets unique device-identifier.

Parameters

pDevNum	device-index
pID	pointer to an output buffer
pBufferSize	Size of buffer pointed to by `pID` in bytes. Maximum ID length is given via ILMSENS_HAL_MOD_ID_BUF_SIZE.

Returns

number of characters written to the buffer
negative error-code

6.6.2.2 ilmsens_hal_getModInfo()

```
int ilmsens_hal_getModInfo (
            unsigned int pDevNum,
            struct ilmsens_hal_ModInfo * pInfo )
```

Gets device hardware-configuration.

Parameters

pDevNum	device-index
pInfo	pointer to the ilmsens_hal_ModInfo structure

Returns

ILMSENS_SUCCESS on success (and populates pInfo)
negative error-code

6.7 Device configuration

Functions for configuring specified devices.

Data Structures

- struct ilmsens_hal_ModConfig

  Represents the basic device configuration.

Macros

- #define ILMSENS_HAL_SLAVE_SENSOR 0

  setup the sensor(s) to be slave devices
- #define ILMSENS_HAL_MASTER_SENSOR 1

  setup the sensor(s) to be master devices

Functions

- int ilmsens_hal_setupSensors (unsigned int ∗pDevNums, unsigned int pNum, const struct ilmsens_hal_↩ ModConfig ∗pConfig)

  Performs the initial setup of specified devices.
- int ilmsens_hal_setMaster (unsigned int ∗pDevNums, unsigned int pNum, int pMode)

  Sets master/slave operational mode.

6.7.1 Detailed Description

Functions for configuring specified devices.

6.7.2 Function Documentation

6.7.2.1 ilmsens_hal_setupSensors()

```
int ilmsens_hal_setupSensors (
            unsigned int * pDevNums,
            unsigned int pNum,
            const struct ilmsens_hal_ModConfig * pConfig )
```

Performs the initial setup of specified devices.

This function must be called before starting a measurement session.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pConfig	pointer to configuration parameters in a ilmsens_hal_ModConfig structure

Returns

ILMSENS_SUCCESS on success
negative error-code

6.7.2.2 ilmsens_hal_setMaster()

```
int ilmsens_hal_setMaster (
            unsigned int * pDevNums,
            unsigned int pNum,
            int pMode )
```

Sets master/slave operational mode.

Application needs to make sure, that any independent sensor is set to master mode.

For hardware-synchronized operation of multiple sensors, exactly one sensor of a connected group has to be master the others in teh group must be configured as slaves.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pMode	ILMSENS_HAL_SLAVE_SENSOR = slave | ILMSENS_HAL_MASTER_SENSOR = master

Returns

ILMSENS_SUCCESS on success
negative error-code

6.8 Measurement configuration

Macros

- #define ILMSENS_HAL_SYNCH_OFF 0

  retract digital synchronisation, i.e. sensors will be unsynch'ed
- #define ILMSENS_HAL_SYNCH_ON 1

  perform digital synchronisation, i.e. sensors will be synch'ed
- #define ILMSENS_HAL_TX_OFF 1

  transmitter is in power down mode, i.e. output amplifier is switch off
- #define ILMSENS_HAL_TX_ON 0

  transmitter is working, i.e. output amplifier is switched on

Functions

- int ilmsens_hal_setAvg (unsigned int ∗pDevNums, unsigned int pNum, unsigned int pAvg, unsigned int p↩ WaitCyc)

  Sets software averages and wait cycles.
- int ilmsens_hal_setMLBS (unsigned int ∗pDevNums, unsigned int pNum)

  Resets the M-sequence generator (transmitter) of each device.
- int ilmsens_hal_setPD (unsigned int ∗pDevNums, unsigned int pNum, int pPD)

  Controls the power-status of the transmitter.
- int ilmsens_hal_synchMS (unsigned int ∗pDevNums, unsigned int pNum, int pMode)

  Performs digital synchronisation.

6.8.1 Detailed Description

6.8.2 Function Documentation

6.8.2.1 ilmsens_hal_setAvg()

```
int ilmsens_hal_setAvg (
            unsigned int * pDevNums,
            unsigned int pNum,
            unsigned int pAvg,
            unsigned int pWaitCyc )
```

Sets software averages and wait cycles.

Software averages define the acquisition aperture duration. If wait cycles are set to 0, devices measure in continuous mode. If wait cycles are non-zero, devices measure in snapshot mode.

Note: May only be called when no measurement is running!

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pAvg	number of Sw averages
pWaitCyc	number of wait cycles

Returns

ILMSENS_SUCCESS on success
negative error-code

6.8.2.2 ilmsens_hal_setMLBS()

```
int ilmsens_hal_setMLBS (
            unsigned int * pDevNums,
            unsigned int pNum )
```

Resets the M-sequence generator (transmitter) of each device.

After digital synchronisation, the M-sequence generator (transmitter) of each device should be reset to ensure repeatable alignment of the transmitters and receivers. The reset typically takes a few ms to complete and cannot be used to mute the transmitter(s).

Note: May only be called when no measurement is running!

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements

Returns

ILMSENS_SUCCESS on success
negative error-code

6.8.2.3 ilmsens_hal_setPD()

```
int ilmsens_hal_setPD (
            unsigned int * pDevNums,
            unsigned int pNum,
            int pPD )
```

Controls the power-status of the transmitter.

Note: May only be called when no measurement is running!

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pPD	ILMSENS_HAL_TX_ON = power-up the transmitter | ILMSENS_HAL_TX_OFF = power-down (mute) the transmitter

Returns

ILMSENS_SUCCESS on success
negative error-code

6.8.2.4 ilmsens_hal_synchMS()

```
int ilmsens_hal_synchMS (
            unsigned int * pDevNums,
            unsigned int pNum,
            int pMode )
```

Performs digital synchronisation.

Must be used at least once before a measurement is started.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pMode	ILMSENS_HAL_SYNCH_OFF = de-synchronise sensors (revoke) | ILMSENS_HAL_SYNCH_ON = trigger digital synchronisation

Returns

ILMSENS_SUCCESS on success
negative error-code

6.9 Measurement run

Functions to start/stop measurements with one or more devices.

Macros

- #define ILMSENS_HAL_RUN_OFF 0

  sensor is currently not measuring
- #define ILMSENS_HAL_RUN_RAW 1

  sensor is measuring and data is not buffered by API
- #define ILMSENS_HAL_RUN_BUF 2

  sensor is measuring and data is buffered by API in separate thread

Functions

- int ilmsens_hal_measRun (unsigned int *pDevNums, unsigned int pNum, int pMode)

  Starts a measurement run with specified devices.
- int ilmsens_hal_measStop (unsigned int *pDevNums, unsigned int pNum)

  Stops running measurement.

6.9.1 Detailed Description

Functions to start/stop measurements with one or more devices.

The library supports two modes of operation for transfering measured data from the device-buffer memory to the host-system memory (HAL or application).

- In case the measurement session was started in the **raw mode**, measured data is transferred from the device to the host-system memory whenever a complete measurement is available and the function ilmsens_hal_measRdy() is called by the user application. This way the data transfer is performed synchronously with the user application's thread. However, the user application must ensure fast enough polling of the sensor to prevent ring buffer overflow by itself
- In case the measurement session is started in **buffered mode**, a separate thread is created by the library. This thread implements periodic polling of the devices' ring-buffer status and transfer of measured data to the host". Therefore, the data transfer is performed asynchronously with respect to the progress of the user application thread. This way high jitter in processing latency will not cause a discontinuity in the measurement progress and revent potential overflow of the devices' ring-buffer memory.

Note: The **buffered mode** is the preferred mode of operation of the library.

6.9.2 Function Documentation

6.9.2.1 ilmsens_hal_measRun()

```
int ilmsens_hal_measRun (
            unsigned int * pDevNums,
            unsigned int pNum,
            int pMode )
```

Starts a measurement run with specified devices.

One measurement run may be pending at a time.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pMode	specifies the mode of operation (ILMSENS_HAL_RUN_RAW for **raw mode** of operation, ILMSENS_HAL_RUN_BUF for **buffered mode** of operation)

Returns

ILMSENS_SUCCESS on success
negative error-code

See also

ilmsens_hal_measStop()

6.9.2.2 ilmsens_hal_measStop()

```
int ilmsens_hal_measStop (
              unsigned int * pDevNums,
              unsigned int pNum )
```

Stops running measurement.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements

Returns

ILMSENS_SUCCESS on success
negative error-code

6.10 Raw measured data retrieval

Functions to read measured data out of internal ring-buffer.

Typedefs

- typedef int32_t ilmsens_hal_SampleType

  Represents the data type of raw measured data.

Functions

- int ilmsens_hal_measRdy (unsigned int *pDevNums, unsigned int pNum)

  Reads the fill-level of the internal ring-buffer for all specified devices.
- int ilmsens_hal_measRead (unsigned int *pDevNums, unsigned int pNum, ilmsens_hal_SampleType *p↩ Buffer, size_t pBufSizeBytes)

  Reads the measurement data for all specified devices in non-blocking way.
- int ilmsens_hal_measGet (unsigned int *pDevNums, unsigned int pNum, ilmsens_hal_SampleType *pBuffer, size_t pBufSizeBytes, unsigned int pTimeoutMillis)

  Blocks and reads the measurement data for all specified devices when it becomes available.

6.10.1 Detailed Description

Functions to read measured data out of internal ring-buffer.

- There is a simple blocking interface implemented by ilmsens_hal_measGet().
- Furthermode there is a non-blocking interface, implemented by ilmsens_hal_measRdy() for polling buffer-level of the ring-buffer(s) and ilmsens_hal_measRead() for actually reading the next measurement to the user application's memory.

6.10.2 Function Documentation

6.10.2.1 ilmsens_hal_measRdy()

```
int ilmsens_hal_measRdy (
            unsigned int * pDevNums,
            unsigned int pNum )
```

Reads the fill-level of the internal ring-buffer for all specified devices.

This functions is not blocking and returns immediately. However, in raw mode, it will transfer data from a device to the libraries' ring-buffer on the host, if it discovers that a complete dataset is available.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements

Returns

minimum number of complete datasets available for all specified devices
0 if at least one device has no new data available
negative error-code

6.10.2.2 ilmsens_hal_measRead()

```
int ilmsens_hal_measRead (
            unsigned int * pDevNums,
            unsigned int pNum,
            ilmsens_hal_SampleType * pBuffer,
            size_t pBufSizeBytes )
```

Reads the measurement data for all specified devices in non-blocking way.

This functions is not blocking and returns immediately with the next measurement data or an error-code if no data are available.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pBuffer	pointer to output buffer for measured data
pBufSizeBytes	Size of buffer pointed to by `pBuffer` in bytes

Returns

number of elements (samples) copied to the buffer
ILMSENS_ERROR_AGAIN if a completed dataset is not available for every device
negative error-code

See also

Measurement Data Format
Calculating the output-buffer size

6.10.2.3 ilmsens_hal_measGet()

```
int ilmsens_hal_measGet (
            unsigned int * pDevNums,
            unsigned int pNum,
            ilmsens_hal_SampleType * pBuffer,
            size_t pBufSizeBytes,
            unsigned int pTimeoutMillis )
```

Blocks and reads the measurement data for all specified devices when it becomes available.

This functions blocks the caller until at least one complete measurement is available for every device or a specified timeout expired. The buffer `pBuffer` must be large enough to hold one complete dataset for each device, i.e. it must be able to hold at least `pNum` complete datasets.

Note: if `pTimeoutMillis` is 0, this function will block forever...

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pBuffer	pointer to output buffer for measured data
pBufSizeBytes	Size of buffer pointed to by `pBuffer` in bytes
pTimeoutMillis	timeout [ms] (0 = forever)

Returns

number of elements (samples) copied to the buffer
ILMSENS_ERROR_TIMEOUT when the timeout expired before enough data was received
negative error-code

See also

Measurement Data Format
Calculating the output-buffer size

6.11 Low-level access to sensor registers & memory

Typedefs

- typedef uint32_t ilmsens_hal_MemoryType

  Represents data type for register and memory access.

Functions

- int ilmsens_hal_readReg (unsigned int *pDevNums, unsigned int pNum, unsigned int pReg, ilmsens_hal_↩ MemoryType *pVal, size_t pBufSizeBytes)

  Reads value from register at `pReg` address from all specified devices to buffer.
- int ilmsens_hal_writeReg (unsigned int *pDevNums, unsigned int pNum, unsigned int pReg, ilmsens_hal_↩ MemoryType pVal)

  Writes `pVal` value to register at `pReg` address to all specified devices.
- int ilmsens_hal_readBlk (unsigned int *pDevNums, unsigned int pNum, unsigned int pAdr, unsigned int p↩ NumEl, ilmsens_hal_MemoryType *pVal, size_t pBufSizeBytes)

  Reads `pNumEl` elements (32-bit words) starting at address `pAdr` from the internal memory of specified devices into a buffer `pVal`.
- int ilmsens_hal_writeBlk (unsigned int *pDevNums, unsigned int pNum, unsigned int pAdr, unsigned int p↩ NumEl, ilmsens_hal_MemoryType *pVal, size_t pBufSizeBytes)

  Writes `pNumEl` elements (32-bit words) from buffer `pVal` to internal memory starting at address `pAdr` of specified devices.

6.11.1 Detailed Description

6.11.2 Function Documentation

6.11.2.1 ilmsens_hal_readReg()

```
int ilmsens_hal_readReg (
            unsigned int * pDevNums,
            unsigned int pNum,
            unsigned int pReg,
            ilmsens_hal_MemoryType * pVal,
            size_t pBufSizeBytes )
```

Reads value from register at `pReg` address from all specified devices to buffer.

The buffer `pVal` must be large enough to hold one word for each device, i.e. it must be at least `pNum` words in size.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pReg	register address
pVal	pointer to buffer for register value(s)
pBufSizeBytes	Size of buffer pointed to by `pVal` in bytes

Returns

number of words (elements) copied to the buffer
negative error-code

6.11.2.2 ilmsens_hal_writeReg()

```
int ilmsens_hal_writeReg (
              unsigned int * pDevNums,
              unsigned int pNum,
              unsigned int pReg,
              ilmsens_hal_MemoryType pVal )
```

Writes `pVal` value to register at `pReg` address to all specified devices.

The same value is written to each device.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pReg	register address
pVal	new register value

Returns

ILMSENS_SUCCESS on success
negative error-code

6.11.2.3 ilmsens_hal_readBlk()

```
int ilmsens_hal_readBlk (
              unsigned int * pDevNums,
              unsigned int pNum,
              unsigned int pAdr,
              unsigned int pNumEl,
              ilmsens_hal_MemoryType * pVal,
              size_t pBufSizeBytes )
```

Reads `pNumEl` elements (32-bit words) starting at address `pAdr` from the internal memory of specified devices into a buffer `pVal`.

The buffer must be large enough to hold `pNumEl` words for all specified devices, i.e. it provide space for at least `pNumEl` x `pNum` words.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pAdr	word-aligned start memory address
pNumEl	number of words (elements) to read
pVal	pointer to buffer for the transferred memory content
pBufSizeBytes	Size of buffer pointed to by `pVal` in bytes

Returns

number of words (elements) copied to the buffer
negative error-code

6.11.2.4 ilmsens_hal_writeBlk()

```
int ilmsens_hal_writeBlk (
              unsigned int * pDevNums,
              unsigned int pNum,
              unsigned int pAdr,
              unsigned int pNumEl,
              ilmsens_hal_MemoryType * pVal,
              size_t pBufSizeBytes )
```

Writes `pNumEl` elements (32-bit words) from buffer `pVal` to internal memory starting at address `pAdr` of specified devices.

The same content is written to each device, i.e. the buffer `pVal` must hold `pNumEl` words regardless of `pNum`.

Parameters

pDevNums	pointer to a first element or an array of device-indexes
pNum	number of array-elements
pAdr	word-aligned start memory address
pNumEl	number of words (elements) to write
pVal	pointer to buffer with data to write
pBufSizeBytes	Size of buffer pointed to by `pVal` in bytes

Returns

ILMSENS_SUCCESS on success
negative error-code

7 Data Structure Documentation

7.1 ilmsens_hal_ModConfig Struct Reference

Represents the basic device configuration.

Data Fields

- unsigned int mOrder
  MLBS order.
- unsigned int mSub
  subsampling factor
- double mClk
  master clock
- unsigned int mOV
  number of oversampling
- unsigned int mTx
  number of transmitters
- unsigned int mRx
  number of receivers

7.1.1 Detailed Description

Represents the basic device configuration.

Hint: In case useless data is returned or memory access exceptions occur, make sure your compiler does not use wrong memory alignement for structure members.

7.2 ilmsens_hal_ModInfo Struct Reference

Represents the device configuration & runtime parameters and limits.

Data Fields

- ilmsens_hal_ModConfig mConfig
  basic configuration of sensor
- double mTB_Fc
  corner frequency for timebase calibration
- double mTemp
  device temperature
- double mLSB_Volt
  ADC LSB voltage (for calculating physical values out of raw data)
- double mFSR [2]
  ADC full scale range (minimum and maximum value)
- unsigned int mHWAvg
  number of hardware averages (FPGA)
- unsigned int mAvg

number of averages

- unsigned int mAvgLim [2]

  limits for averages (minimum and maximum value)
- unsigned int mWait

  number of wait cycles
- unsigned int mWaitLim [2]

  limits for wait cycles (minimum and maximum value)
- unsigned int mNumSamp

  number of samples per channel

7.2.1 Detailed Description

Represents the device configuration & runtime parameters and limits.

Hint: In case useless data is returned or memory access exceptions occur, make sure your compiler does not use wrong memory alignement for structure members.

7.3 ilmsens_hal_Version Struct Reference

Represents the version identifier.

Data Fields

- unsigned int mMajor

  major version number
- unsigned int mMinor

  minor version number
- unsigned int mBuild

  build number

7.3.1 Detailed Description

Represents the version identifier.

Hint: In case useless data is returned or memory access exceptions occur, make sure your compiler does not use wrong memory alignement for structure members.

8 File Documentation

8.1 ilmsens_error.h File Reference

Ilmsens error code definitions.

Enumerations

- enum ilmsens_error {
  ILMSENS_SUCCESS = 0, ILMSENS_ERROR_INVALID_PARAM = -1, ILMSENS_ERROR_STATE = -2, I↩
  LMSENS_ERROR_BUSY = -3,
  ILMSENS_ERROR_ACCESS = -4, ILMSENS_ERROR_IO = -5, ILMSENS_ERROR_NO_MEMORY = -6, I↩
  LMSENS_ERROR_AGAIN = -7,
  ILMSENS_ERROR_TIMEOUT = -8, ILMSENS_ERROR_NOT_SUPPORTED = -9, ILMSENS_ERROR_U↩
  NKNOWN = -99 }

  Common error codes.

8.1.1 Detailed Description

Ilmsens error code definitions.

The error codes defined in this file are used in various Ilmsens APIs as return values.

Author

Ralf Herrmann ralf.herrmann@ilmsens.com

8.2 ilmsens_hal.h File Reference

Function definitions for Ilmsens HAL API.

Functions

- int ilmsens_hal_getVersion (struct ilmsens_hal_Version *pVersion)

  Return the HAL version.
- int ilmsens_hal_setDEBLevel (unsigned int pLevel)

  Sets the verbosity of the diagnostics output Diagnostics are sent to std.
- int ilmsens_hal_initHAL (void)

  Initializes the library.
- void ilmsens_hal_deinitHAL (void)

  Deinitializes the library.
- int ilmsens_hal_openSensors (unsigned int *pDevNums, unsigned int pNum)

  Allocates specified devices for a measurement session.
- void ilmsens_hal_closeSensors (unsigned int *pDevNums, unsigned int pNum)

  Releases specified devices.
- int ilmsens_hal_getModId (unsigned int pDevNum, char *pID, size_t pBufferSize)

  Gets unique device-identifier.
- int ilmsens_hal_getModInfo (unsigned int pDevNum, struct ilmsens_hal_ModInfo *pInfo)

Gets device hardware-configuration.

- int ilmsens_hal_setupSensors (unsigned int *pDevNums, unsigned int pNum, const struct ilmsens_hal_ModConfig *pConfig)

  Performs the initial setup of specified devices.
- int ilmsens_hal_setMaster (unsigned int *pDevNums, unsigned int pNum, int pMode)

  Sets master/slave operational mode.
- int ilmsens_hal_setAvg (unsigned int *pDevNums, unsigned int pNum, unsigned int pAvg, unsigned int pWaitCyc)

  Sets software averages and wait cycles.
- int ilmsens_hal_setMLBS (unsigned int *pDevNums, unsigned int pNum)

  Resets the M-sequence generator (transmitter) of each device.
- int ilmsens_hal_setPD (unsigned int *pDevNums, unsigned int pNum, int pPD)

  Controls the power-status of the transmitter.
- int ilmsens_hal_synchMS (unsigned int *pDevNums, unsigned int pNum, int pMode)

  Performs digital synchronisation.
- int ilmsens_hal_measRun (unsigned int *pDevNums, unsigned int pNum, int pMode)

  Starts a measurement run with specified devices.
- int ilmsens_hal_measStop (unsigned int *pDevNums, unsigned int pNum)

  Stops running measurement.
- int ilmsens_hal_measRdy (unsigned int *pDevNums, unsigned int pNum)

  Reads the fill-level of the internal ring-buffer for all specified devices.
- int ilmsens_hal_measRead (unsigned int *pDevNums, unsigned int pNum, ilmsens_hal_SampleType *pBuffer, size_t pBufSizeBytes)

  Reads the measurement data for all specified devices in non-blocking way.
- int ilmsens_hal_measGet (unsigned int *pDevNums, unsigned int pNum, ilmsens_hal_SampleType *pBuffer, size_t pBufSizeBytes, unsigned int pTimeoutMillis)

  Blocks and reads the measurement data for all specified devices when it becomes available.
- int ilmsens_hal_readReg (unsigned int *pDevNums, unsigned int pNum, unsigned int pReg, ilmsens_hal_MemoryType *pVal, size_t pBufSizeBytes)

  Reads value from register at `pReg` address from all specified devices to buffer.
- int ilmsens_hal_writeReg (unsigned int *pDevNums, unsigned int pNum, unsigned int pReg, ilmsens_hal_MemoryType pVal)

  Writes `pVal` value to register at `pReg` address to all specified devices.
- int ilmsens_hal_readBlk (unsigned int *pDevNums, unsigned int pNum, unsigned int pAdr, unsigned int pNumEl, ilmsens_hal_MemoryType *pVal, size_t pBufSizeBytes)

  Reads `pNumEl` elements (32-bit words) starting at address `pAdr` from the internal memory of specified devices into a buffer `pVal`.
- int ilmsens_hal_writeBlk (unsigned int *pDevNums, unsigned int pNum, unsigned int pAdr, unsigned int pNumEl, ilmsens_hal_MemoryType *pVal, size_t pBufSizeBytes)

  Writes `pNumEl` elements (32-bit words) from buffer `pVal` to internal memory starting at address `pAdr` of specified devices.

8.2.1 Detailed Description

Function definitions for Ilmsens HAL API.

This header contains prototypes for the exported functions of the HAL library.

Author

Ralf Herrmann `ralf.herrmann@ilmsens.com`

8.3 ilmsens_hal_defn.h File Reference

Macro definitions for Ilmsens HAL.

Macros

- #define ILMSENS_HAL_RUN_OFF 0
  sensor is currently not measuring
- #define ILMSENS_HAL_RUN_RAW 1
  sensor is measuring and data is not buffered by API
- #define ILMSENS_HAL_RUN_BUF 2
  sensor is measuring and data is buffered by API in separate thread
- #define ILMSENS_HAL_MOD_ID_BUF_SIZE 1024
  maximum length buffer size for retrieving the unique ID string.
- #define ILMSENS_HAL_SLAVE_SENSOR 0
  setup the sensor(s) to be slave devices
- #define ILMSENS_HAL_MASTER_SENSOR 1
  setup the sensor(s) to be master devices
- #define ILMSENS_HAL_SYNCH_OFF 0
  retract digital synchronisation, i.e. sensors will be unsynch'ed
- #define ILMSENS_HAL_SYNCH_ON 1
  perform digital synchronisation, i.e. sensors will be synch'ed
- #define ILMSENS_HAL_TX_OFF 1
  transmitter is in power down mode, i.e. output amplifier is switch off
- #define ILMSENS_HAL_TX_ON 0
  transmitter is working, i.e. output amplifier is switched on
- #define ILMSENS_DEB_NO 0U
  do not output debug/info messages
- #define ILMSENS_DEB_INFO 1U
  only output errors and very important information
- #define ILMSENS_DEB_MORE 2U
  output errors, warnings, and important information
- #define ILMSENS_DEB_MOST 3U
  output errors, warnings, and debug information
- #define ILMSENS_DEB_ALL 4U
  output errors, warnings, and trace information

8.3.1 Detailed Description

Macro definitions for Ilmsens HAL.

This header contains definitions for various values used in different HAL functions.

Author

Ralf Herrmann ralf.herrmann@ilmsens.com

8.4 ilmsens_hal_types.h File Reference

Type definitions for Ilmsens HAL.

Data Structures

- struct ilmsens_hal_Version

  Represents the version identifier.
- struct ilmsens_hal_ModConfig

  Represents the basic device configuration.
- struct ilmsens_hal_ModInfo

  Represents the device configuration & runtime parameters and limits.

Typedefs

- typedef int32_t ilmsens_hal_SampleType

  Represents the data type of raw measured data.
- typedef uint32_t ilmsens_hal_MemoryType

  Represents data type for register and memory access.

8.4.1 Detailed Description

Type definitions for Ilmsens HAL.

This header defines structure types used by the various setup and information retrieval HAL functions. Furthermore, it defines the data types for measured samples and register/memory access.

Author

Ralf Herrmann ralf.herrmann@ilmsens.com

8.5 ilmsens_hal_version.h File Reference

Version definitions.

Macros

- #define ILMSENS_HAL_API_VER_MAJOR 1

  major version number of HAL API
- #define ILMSENS_HAL_API_VER_MINOR 1

  minor version number of HAL API
- #define ILMSENS_HAL_API_VER_BUILD 1

  build number of HAL API
- #define ILMSENS_HAL_API_VER (ILMSENS_HAL_API_VER_MAJOR ∗ 100000 + ILMSENS_HAL_API_↩ VER_MINOR ∗ 1000 + ILMSENS_HAL_API_VER_BUILD)

  single version number for preprocessors

8.5.1 Detailed Description

Version definitions.

Author

Ralf Herrmann ralf.herrmann@ilmsens.com

Index