Vital Signs AGC

API Documentation
revision v0.1.1

Generated by Doxygen

Contents

1 Vital Signs AGC

1.1 Overview

The Automatic Gain Compensation (AGC) algorithm keeps the PPG signal within a configured ADC range with optional amplitude control. The PPG signal range is controlled via the PD offset and the PPG amplitude is controlled via the LED current.

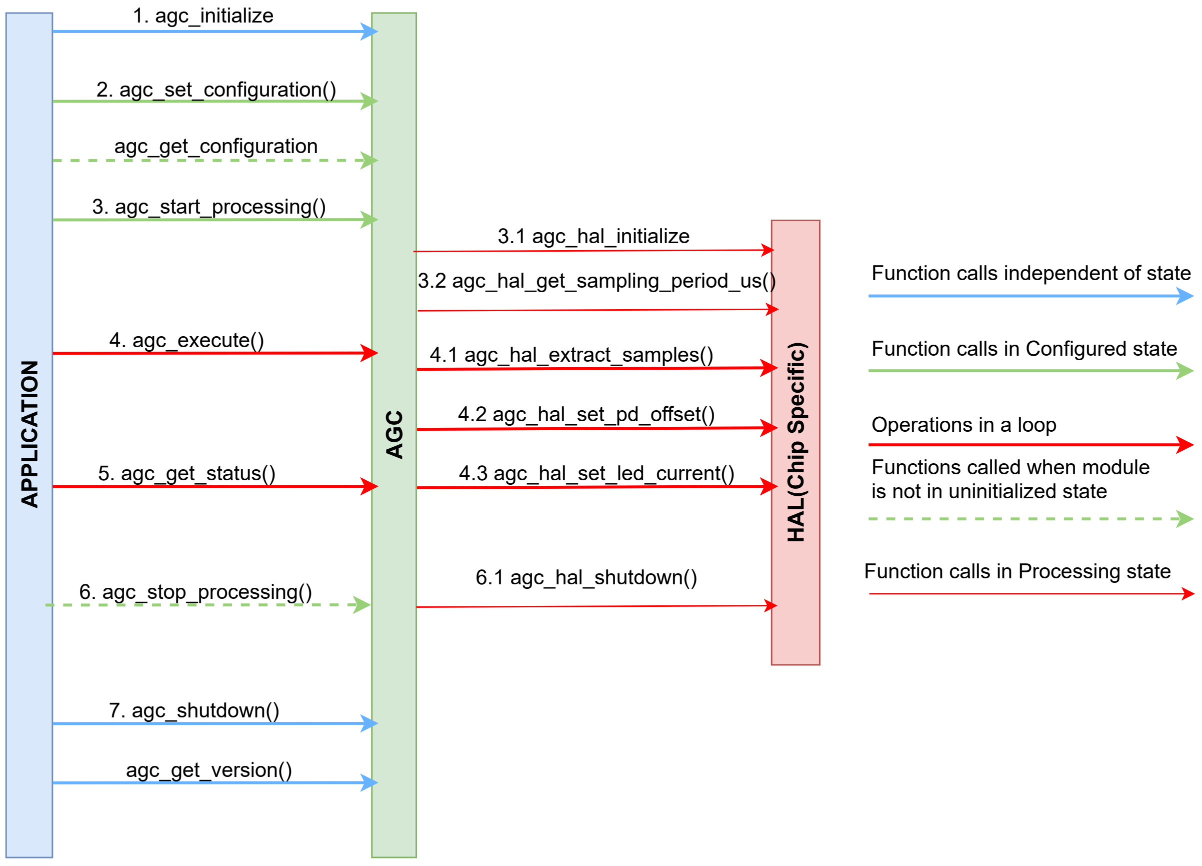
APPLICATION
AGC
HAL(Chip Specific)
1. agc_initialize
2. agc_set_configuration()
agc_get_configuration
3. agc_start_processing()
3.1 agc_hal_initialize
3.2 agc_hal_get_sampling_period_us()
4. agc_execute()
4.1 agc_hal_extract_samples()
4.2 agc_hal_set_pd_offset()
5. agc_get_status()
4.3 agc_hal_set_led_current()
6. agc_stop_processing()
6.1 agc_hal_shutdown()
7. agc_shutdown()
agc_get_version()
Function calls independent of state
Function calls in Configured state
Operations in a loop
Functions called when module is not in uninitialized state
Function calls in Processing state

Figure 1 Block Diagram

PD offset control is self-adjusting regarding the impact of PD offset changes and ADC sensitivity. The integrated LED current control algorithm follows a simple statistical approach to identify phases of a steady PPG signal, free of heavy motion artifacts. A proper configuration is required for good algorithm performance. The module also supports that the LED current is controlled externally by a Vital Signs algorithm.

The AGC algorithm is a single PPG channel solution that can be applied independently to multiple PPG channels, if the sensor supports setting PD offset and LED current per PPG channel. This means that this solution is suitable for single-channel HRM, multi-channel HRM, and dual-channel SpO2 applications when used with a compatible sensor such as AS7050 and AS7056/57. AS703x sensors are not suited for multi-channel applications with this AGC algorithm.

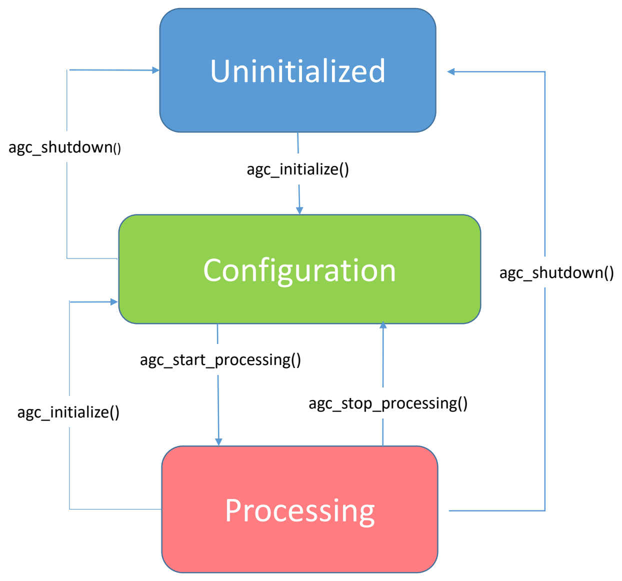

Figure 2 State Diagram

On a high-level overview, the AGC module is used as follows:

1. After initialization, the AGC module enters Configuration state and can be configured based on the sensor configuration and the application.

2. After the configuration has been set, the module enters the Processing state. When entering the Processing state, the HAL layer and the internal parameters of the module are initialized and the AGC algorithm is ready for execution.

3. The function to execute the algorithm can be called anytime while in the Processing state. The algorithm processes the provided input data and generates the resulting PD offset and LED current values. The resulting PD offset and LED current values are applied to the chip via HAL functions.

4. The AGC status can be read anytime while in the Processing state.

5. When calling the function to stop processing, the AGC module enters the Configuration state. Internally, this function de-initializes the HAL layer.

6. The configuration of the AGC module can be updated while in the Configuration state.

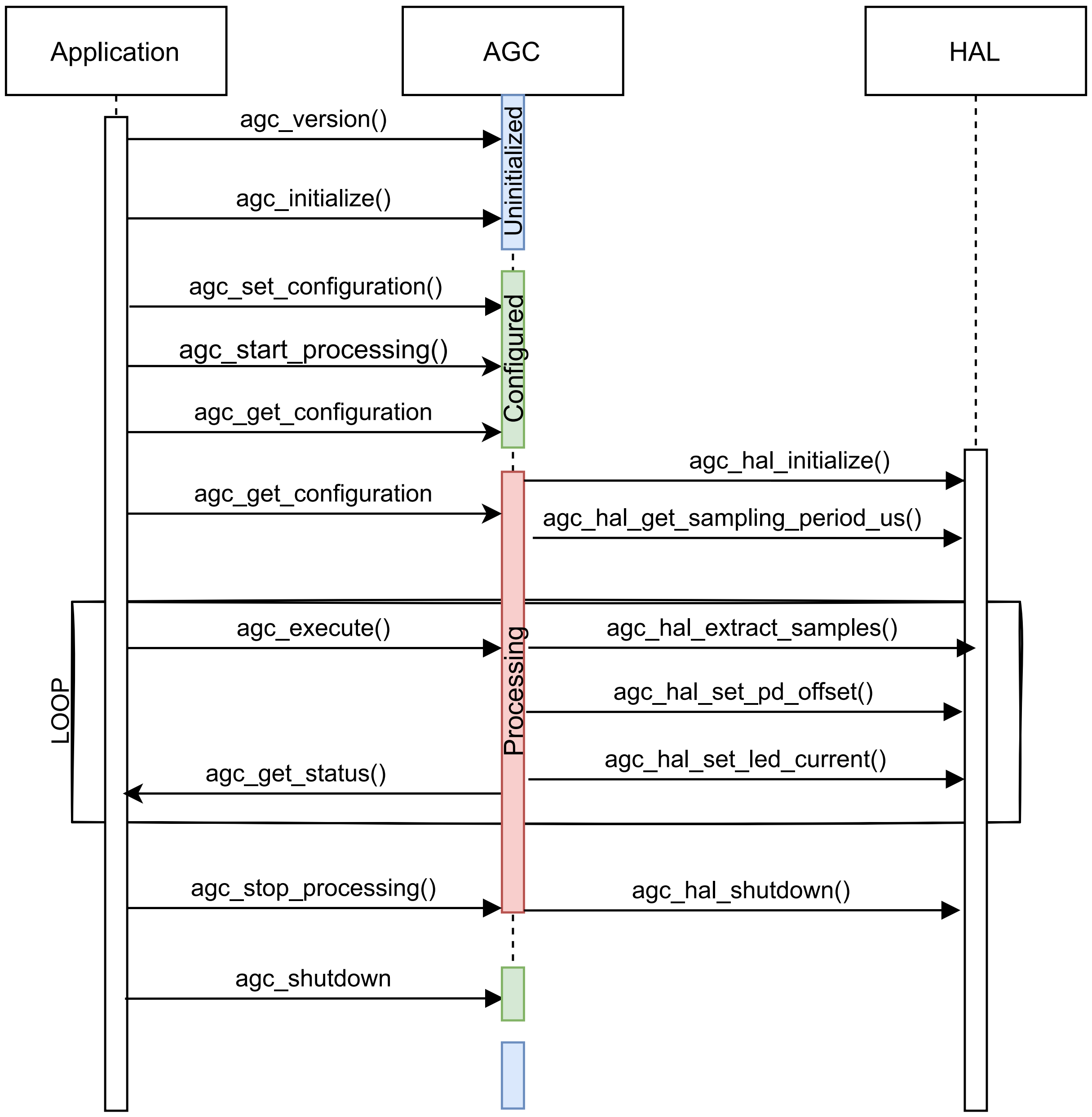

Figure 3 Sequence Diagram

2 Module Documentation

2.1 AGC Module

Data Structures

- struct agc_status_t
- struct agc_configuration_t
- struct agc_version_t

Macros

- #define AGC_MAX_CHANNEL_CNT 4

Typedefs

- typedef uint8_t agc_change_state_t
- typedef uint8_t agc_mode_t
- typedef uint8_t agc_ampl_cntl_mode_t
- typedef uint8_t agc_channel_id_t

Enumerations

- enum agc_change_state {
  AGC_STATE_UNCHANGED = 0,
  AGC_STATE_INCREASED = 1,
  AGC_STATE_DECREASED = 2,
  AGC_STATE_ATMIN = 3,
  AGC_STATE_ATMAX = 4,
  AGC_STATE_NOT_CONTROLLED = 5 }
- enum agc_mode { AGC_MODE_DEFAULT = 0 }
- enum agc_ampl_cntl_mode {
  AGC_AMPL_CNTL_MODE_DISABLED = 0,
  AGC_AMPL_CNTL_MODE_AUTO = 1,
  AGC_AMPL_CNTL_MODE_EXTERNAL = 2 }

Functions

- err_code_t agc_initialize (void)
- err_code_t agc_set_configuration (const agc_configuration_t ∗p_agc_configs, uint8_t agc_config_num)
- err_code_t agc_get_configuration (agc_configuration_t ∗p_agc_configs, uint8_t ∗p_agc_config_num)
- err_code_t agc_start_processing (void ∗p_config, uint16_t size)
- err_code_t agc_execute (const uint8_t ∗p_fifo_data, uint16_t fifo_data_size)
- err_code_t agc_get_status (agc_status_t ∗p_agc_status, uint8_t ∗p_agc_status_num)
- err_code_t agc_stop_processing (void)
- err_code_t agc_shutdown (void)
- err_code_t agc_get_version (agc_version_t ∗p_agc_version)

2.1.1 Detailed Description

The Automatic Gain Compensation (AGC) module implements an algorithm to keep PPG signals within a configured range by continiously adjusting the photodiode (PD) offset current and the LED current.

2.1.2 Macro Definition Documentation

2.1.2.1 AGC_MAX_CHANNEL_CNT `#define AGC_MAX_CHANNEL_CNT 4`

The maximum number of enabled AGC instances.

2.1.3 Typedef Documentation

2.1.3.1 agc_change_state_t `typedef uint8_t agc_change_state_t`

Type for agc_change_state

2.1.3.2 agc_mode_t `typedef uint8_t agc_mode_t`

Type for agc_mode.

2.1.3.3 agc_ampl_cntl_mode_t `typedef uint8_t agc_ampl_cntl_mode_t`

Type for agc_ampl_cntl_mode.

2.1.3.4 agc_channel_id_t `typedef uint8_t agc_channel_id_t`

Represents a PPG channel. A value of 0 represents a disabled channels, all other values are chip-specific.

2.1.4 Enumeration Type Documentation

2.1.4.1 agc_change_state `enum agc_change_state`

Describes how a photodiode or LED current changed.

Enumerator

AGC_STATE_UNCHANGED	Current not changed.
AGC_STATE_INCREASED	Current increased.
AGC_STATE_DECREASED	Current decreased.
AGC_STATE_ATMIN	Current needs to be decreased, but is at lower limit.
AGC_STATE_ATMAX	Current needs to be increased, but is at upper limit.
AGC_STATE_NOT_CONTROLLED	current is not controlled as LED control mode is DISABLED.

2.1.4.2 agc_mode `enum agc_mode`

Operating mode of the AGC algorithm.

Enumerator

AGC_MODE_DEFAULT	Default mode.

2.1.4.3 agc_ampl_cntl_mode `enum agc_ampl_cntl_mode`

States of the amplitude control mode.

Enumerator

AGC_AMPL_CNTL_MODE_DISABLED	No amplitude control enabled.
AGC_AMPL_CNTL_MODE_AUTO	Internal amplitude control enabled.
AGC_AMPL_CNTL_MODE_EXTERNAL	Amplitude is controlled by an external algorithm.

2.1.5 Function Documentation

2.1.5.1 agc_initialize()

```
err_code_t agc_initialize (
        void  )
```

Initializes the AGC module.

AGC module transitions to Configuration state after initialization.

Return values

ERR_SUCCESS	Function returns without error.

2.1.5.2 agc_set_configuration()

```
err_code_t agc_set_configuration (
        const agc_configuration_t * p_agc_configs,
        uint8_t agc_config_num )
```

Sets the configuration of the AGC module.

All previous AGC configurations are discarded when this function is called. This function can only be called when the AGC module is in Configuration state.

Parameters

in	*p_agc_configs*	Pointer to an array of AGC configurations. A configuration needs to provided for each AGC instance that shall be enabled.
in	*agc_config_num*	Number of AGC configurations in the p_agc_configs array.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	At least one parameter is wrong.
ERR_POINTER	Detected NULL pointer.
ERR_PERMISSION	Invalid state.

2.1.5.3 agc_get_configuration()

```
err_code_t agc_get_configuration (
        agc_configuration_t * p_agc_configs,
        uint8_t * p_agc_config_num )
```

Gets the configuration of the AGC module.

This function can only be called when the AGC module is not in Uninitialized state.

Parameters

out	*p_agc_configs*	Pointer to where the AGC configurations are written to. The configuration for each enabled AGC instance is provided.
in,out	*p_agc_config_num*	Pointer to a variable containing the maximum number of AGC configurations that can be written to the p_agc_configs array. After writing the AGC configurations, the variable is updated to contain the number of written AGC configurations.

Return values

ERR_SUCCESS	Function returns without error.
ERR_SIZE	Destination buffer is too small.
ERR_POINTER	Detected NULL pointer.
ERR_PERMISSION	Invalid state.

2.1.5.4 agc_start_processing()

```
err_code_t agc_start_processing (
        void * p_config,
        uint16_t size )
```

Prepares the AGC algorithm for execution.

This function can only be called when the AGC module is in Configuration state. The AGC module transitions to Processing state when this function executes successfully.

Parameters

in	*p_config*	Chip-specific configuration information, which is forwarded to the HAL.
in	*size*	Size of the p_config configuration information.

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	Detected NULL pointer.
ERR_SYSTEM_CONFIG	Invalid sample period provided by HAL.
ERR_PERMISSION	Invalid state.

2.1.5.5 agc_execute()

```
err_code_t agc_execute (
        const uint8_t * p_fifo_data,
        uint16_t fifo_data_size )
```

Executes the AGC algorithm by processing the provided FIFO data.

This function can only be called when the AGC module is in Processing state.

Parameters

in	*p_fifo_data*	FIFO data.
in	*fifo_data_size*	Size of FIFO data.

Return values

ERR_SUCCESS	Function returns without error.
ERR_SIZE	Size of FIFO data is zero.
ERR_POINTER	Detected NULL pointer.
ERR_PERMISSION	Invalid state.

2.1.5.6 agc_get_status()

```
err_code_t agc_get_status (
        agc_status_t * p_agc_status,
        uint8_t * p_agc_status_num )
```

Gets the AGC status information for all enabled AGC instances.

This function can only be called when the AGC module is in Processing state.

Parameters

out	*p_agc_status*	Pointer to where the AGC status informations are written to. The AGC status information for each enabled AGC instance is provided. The order of the AGC status informations is identical to the order used for the configuration of the AGC instances.
in,out	*p_agc_status_num*	Pointer to a variable containing the maximum number of AGC status informations that can be written to the p_agc_status array. After writing the AGC status informations, the variable is updated to contain the number of written AGC status informations.

Return values

ERR_SUCCESS	Function returns without error.
ERR_SIZE	Destination buffer is too small.
ERR_POINTER	Detected NULL pointer.
ERR_PERMISSION	Invalid state.

2.1.5.7 agc_stop_processing()

```
err_code_t agc_stop_processing (
        void )
```

Exits the Processing state of the AGC algorithm.

This function can only be called when the AGC module is not in Uninitialized state. The AGC module transitions to Configuration state when this function executes successfully.

Return values

ERR_SUCCESS	Function returns without error.
ERR_PERMISSION	Invalid state.

2.1.5.8 agc_shutdown()

```
err_code_t agc_shutdown (
        void )
```

De-initializes the AGC module.

Return values

ERR_SUCCESS	Function returns without error.

2.1.5.9 agc_get_version()

```
err_code_t agc_get_version (
        agc_version_t * p_agc_version )
```

Gets the version of AGC module.

Parameters

out	*p_agc_version*	Pointer to where the version is written to.

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	Detected NULL pointer.

2.2 HAL Functions

Functions

- err_code_t agc_hal_initialize (void ∗p_config, uint16_t size)
- err_code_t agc_hal_set_led_current (agc_channel_id_t channel_id, uint8_t led_current)
- err_code_t agc_hal_set_pd_offset (agc_channel_id_t channel_id, uint8_t pd_offset)
- err_code_t agc_hal_extract_samples (agc_channel_id_t channel_id, const uint8_t ∗p_fifo_data, uint16_↩ t fifo_data_size, int32_t ∗p_chan_data, uint16_t ∗p_chan_data_num)
- err_code_t agc_hal_get_sampling_period_us (agc_channel_id_t channel_id, uint32_t ∗p_sampling_period)
- err_code_t agc_hal_shutdown (void)

2.2.1 Detailed Description

This file consists of platform-dependent interface functions and definitions for AGC module. The HAL needs to be implemented for every chip using the AGC algorithm.

2.2.2 Function Documentation

2.2.2.1 agc_hal_initialize()

```
err_code_t agc_hal_initialize (
        void * p_config,
        uint16_t size )
```

Initializes the HAL interface for the AGC.

Parameters

in	*p_config*	Application-specific configuration.
in	*size*	Size of the configuration.

Return values

ERR_SUCCESS	Function returns without error.
ERR_SIZE	The size of the configuration does not match.
ERR_POINTER	Detected NULL pointer for data.

2.2.2.2 agc_hal_set_led_current()

```
err_code_t agc_hal_set_led_current (
        agc_channel_id_t channel_id,
        uint8_t led_current )
```

Sets the LED current.

Parameters

in	*channel↩_id*	Channel number
in	*led_current*	New LED current value.

Return values

ERR_SUCCESS	Function returns without error.
ERR_PERMISSION	HAL/OSAL Library is not initialized.
ERR_ARGUMENT	LED current is wrong or channel_id is not supported.

2.2.2.3 agc_hal_set_pd_offset()

```
err_code_t agc_hal_set_pd_offset (
            agc_channel_id_t channel_id,
            uint8_t pd_offset )
```

Sets the PD offset currents.

Parameters

in	*channel↩_id*	Channel number
in	*pd_offset*	New PD offset value.

Return values

ERR_SUCCESS	Function returns without error.
ERR_DATA_TRANSFER	Error during communication with sensor.
ERR_ARGUMENT	PD offset configuration is wrong or channel_id is not supported.
ERR_PERMISSION	HAL/OSAL Library is not initialized.

2.2.2.4 agc_hal_extract_samples()

```
err_code_t agc_hal_extract_samples (
            agc_channel_id_t channel_id,
            const uint8_t * p_fifo_data,
            uint16_t fifo_data_size,
            int32_t * p_chan_data,
            uint16_t * p_chan_data_num )
```

Extract the requested samples from FIFO data stream.

Parameters

in	*channel_id*	Channel number .
in	*p_fifo_data*	FIFO data.
in	*fifo_data_size*	Size of FIFO data.
out	*p_chan_data*	Pointer to buffer where the sub-sample data can be saved.
in,out	*p_chan_data_num*	Input: Number of provided p_chan_data elements; Output: Number of used p_chan_data elements.

Return values

ERR_SUCCESS	Function returns without error.
ERR_ARGUMENT	At least one parameter is wrong.
ERR_POINTER	Detected NULL pointer for data.
ERR_PERMISSION	HAL/OSAL Library is not initialized.

2.2.2.5 agc_hal_get_sampling_period_us()

```
err_code_t agc_hal_get_sampling_period_us (
        agc_channel_id_t channel_id,
        uint32_t * p_sampling_period )
```

Requests the sampling period for the given channel in microseconds.

Parameters

in	*channel_id*	Channel number .
out	*p_sampling_period*	Sampling period of the PPG signal in microseconds.

Return values

ERR_SUCCESS	Function returns without error.
ERR_POINTER	If the argument is NULL.
ERR_PERMISSION	HAL/OSAL Library is not initialized.

2.2.2.6 agc_hal_shutdown()

```
err_code_t agc_hal_shutdown (
        void )
```

Disables this library and block the functions calls.

Return values

ERR_SUCCESS	Function returns without error.

2.3 Error Codes

Typedefs

- typedef enum error_codes err_code_t

Enumerations

- enum error_codes {
  ERR_SUCCESS = 0,
  ERR_PERMISSION = 1,
  ERR_MESSAGE = 2,
  ERR_MESSAGE_SIZE = 3,
  ERR_POINTER = 4,
  ERR_ACCESS = 5,
  ERR_ARGUMENT = 6,
  ERR_SIZE = 7,
  ERR_NOT_SUPPORTED = 8,
  ERR_TIMEOUT = 9,
  ERR_CHECKSUM = 10,
  ERR_OVERFLOW = 11,
  ERR_EVENT = 12,
  ERR_INTERRUPT = 13,
  ERR_TIMER_ACCESS = 14,
  ERR_LED_ACCESS = 15,
  ERR_TEMP_SENSOR_ACCESS = 16,
  ERR_DATA_TRANSFER = 17,
  ERR_FIFO = 18,
  ERR_OVER_TEMP = 19,
  ERR_IDENTIFICATION = 20,
  ERR_COM_INTERFACE = 21,
  ERR_SYNCHRONISATION = 22,
  ERR_PROTOCOL = 23,
  ERR_MEMORY = 24,
  ERR_THREAD = 25,
  ERR_SPI = 26,
  ERR_DAC_ACCESS = 27,
  ERR_I2C = 28,
  ERR_NO_DATA = 29,
  ERR_SYSTEM_CONFIG = 30,
  ERR_USB_ACCESS = 31,
  ERR_ADC_ACCESS = 32,
  ERR_SENSOR_CONFIG = 33,
  ERR_SATURATION = 34,
  ERR_MUTEX = 35,
  ERR_ACCELEROMETER = 36,
  ERR_CONFIG = 37,
  ERR_BLE = 38,
  ERR_FILE = 39,
  ERR_DATA = 40 }

2.3.1 Detailed Description

Generic error codes used by ams libraries.

2.3.2 Typedef Documentation

2.3.2.1 err_code_t `typedef enum error_codes err_code_t`

This definition will be used for function return values.

2.3.3 Enumeration Type Documentation

2.3.3.1 error_codes `enum error_codes`

Values represent the error codes.

Enumerator

ERR_SUCCESS	Normal return code if everything was successful executed.
ERR_PERMISSION	Operation not permitted
ERR_MESSAGE	Message is invalid. For example: • Message type is not supported • incorrect crc • ...
ERR_MESSAGE_SIZE	Message has the wrong size.
ERR_POINTER	Pointer is invalid. Can be a NULL Pointer or point to a wrong memory area.
ERR_ACCESS	Access denied
ERR_ARGUMENT	Invalid argument
ERR_SIZE	Argument size is too long or too short.
ERR_NOT_SUPPORTED	Function is not supported/implemented.
ERR_TIMEOUT	Got timeout while waiting for answer.
ERR_CHECKSUM	Checksum comparision failed.
ERR_OVERFLOW	Data overflow detected.
ERR_EVENT	Error to get or set an event. For example: • event queue is full or empty • receive an unexpected event • ...

Enumerator

ERR_INTERRUPT	Error to get or set an interrupt. For example a interrupt resource is not available.
ERR_TIMER_ACCESS	Error while accessing timer periphery.
ERR_LED_ACCESS	Error while accessing LED periphery.
ERR_TEMP_SENSOR_ACCESS	Error while accessing temperature sensor.
ERR_DATA_TRANSFER	Communication error
ERR_FIFO	Faulty FIFO handling
ERR_OVER_TEMP	Overtemperature detected.
ERR_IDENTIFICATION	Sensor identification failed.
ERR_COM_INTERFACE	Generic communication interface error. For example: • communication interface is not available • error during open or close an communication interface • ...
ERR_SYNCHRONISATION	Synchronisation error, e.g. on protocol
ERR_PROTOCOL	Generic protocol error
ERR_MEMORY	Memory allocation error
ERR_THREAD	Thread can not created.
ERR_SPI	Error while accessing SPI periphery
ERR_DAC_ACCESS	Error while accessing DAC periphery.
ERR_I2C	Error while accessing I2C periphery.
ERR_NO_DATA	No data available.
ERR_SYSTEM_CONFIG	Error during system configuration. When a system resource is not available or generates an error for example.
ERR_USB_ACCESS	USB error
ERR_ADC_ACCESS	Error while accessing ADC periphery.
ERR_SENSOR_CONFIG	Error during sensor configuration.
ERR_SATURATION	Saturation detected
ERR_MUTEX	Error while mutex handling
ERR_ACCELEROMETER	Error while reading accelerometer data
ERR_CONFIG	Software component is not fully or correctly configured
ERR_BLE	Error while executing BLE stack function
ERR_FILE	Error during file access
ERR_DATA	Internal data is faulty

3 Data Structure Documentation

3.1 agc_configuration_t Struct Reference

Data Fields

- agc_mode_t mode
- agc_ampl_cntl_mode_t led_control_mode
- agc_channel_id_t channel
- uint8_t led_current_min
- uint8_t led_current_max
- uint8_t rel_amplitude_min_x100
- uint8_t rel_amplitude_max_x100
- uint8_t rel_amplitude_motion_x100
- uint8_t num_led_steps
- uint8_t reserved [3]
- int32_t threshold_min
- int32_t threshold_max

3.1.1 Detailed Description

Configuration of an instance of the AGC algorithm.

3.1.2 Field Documentation

3.1.2.1 mode `agc_mode_t agc_configuration_t::mode`

Selects the AGC algorithm mode.

3.1.2.2 led_control_mode `agc_ampl_cntl_mode_t agc_configuration_t::led_control_mode`

Selects the LED amplitude control mode.

3.1.2.3 channel `agc_channel_id_t agc_configuration_t::channel`

Selects the PPG channel that is controlled.

3.1.2.4 led_current_min `uint8_t agc_configuration_t::led_current_min`

Lower bound of the LED current range as a register value.

3.1.2.5 led_current_max `uint8_t agc_configuration_t::led_current_max`

Upper bound of the LED current range as a register value.

3.1.2.6 rel_amplitude_min_x100 `uint8_t agc_configuration_t::rel_amplitude_min_x100`

Lower bound of the targeted PPG signal amplitude, relative to the size of the target PGG signal range. The unit of this field is percent. The minimum valid value is 0, the maximum valid value is 100.

3.1.2.7 rel_amplitude_max_x100 `uint8_t agc_configuration_t::rel_amplitude_max_x100`

Upper bound of the targeted PPG signal amplitude, relative to the size of the target PGG signal range. The unit of this field is percent. This value must not be less than rel_amplitude_min_x100. The maximum valid value is 100.

3.1.2.8 rel_amplitude_motion_x100 `uint8_t agc_configuration_t::rel_amplitude_motion_x100`

Minimum PPG signal amplitude at which the PPG signal is considered a motion artifact. The threshold is relative to the size of the target PGG signal range. The unit of this field is percent. This value must not be less than rel_amplitude_max_x100. The maximum valid value is 100.

3.1.2.9 num_led_steps `uint8_t agc_configuration_t::num_led_steps`

Number of steps the LED current range is partitioned in. When the AGC algorithm determines that the LED current needs to be increased or decreased and the bounds of the LED current range are not yet reached, the LED current is adjusted by one step. If led_control_mode is set to AGC_AMPL_CNTL_MODE_AUTO, this value must not be zero and must not be greater than the size of the LED current range.

3.1.2.10 reserved `uint8_t agc_configuration_t::reserved[3]`

Padding bytes.

3.1.2.11 threshold_min `int32_t agc_configuration_t::threshold_min`

Lower bound of the target PPG signal range. The unit of this field is ADC counts.

3.1.2.12 threshold_max `int32_t agc_configuration_t::threshold_max`

Upper bound of the target PPG signal range. The unit of this field is ADC counts. This value must be greater than threshold_min.

3.2 agc_status_t Struct Reference

Data Fields

- agc_change_state_t pd_offset_change
- uint8_t pd_offset_current
- agc_change_state_t led_change
- uint8_t led_current

3.2.1 Detailed Description

Status information of an instance of the AGC algorithm.

3.2.2 Field Documentation

3.2.2.1 pd_offset_change `agc_change_state_t agc_status_t::pd_offset_change`

Change information for the photodiode offset current.

3.2.2.2 pd_offset_current `uint8_t agc_status_t::pd_offset_current`

Current value of the photodiode offset current as a register value.

3.2.2.3 led_change `agc_change_state_t agc_status_t::led_change`

Change information for the LED current.

3.2.2.4 led_current `uint8_t agc_status_t::led_current`

Current value of the LED current as a register value.

3.3 agc_version_t Struct Reference

Data Fields

- uint8_t major
- uint8_t minor
- uint8_t patch

3.3.1 Detailed Description

Version information of the AGC module.

3.3.2 Field Documentation

3.3.2.1 major `uint8_t agc_version_t::major`

Major version.

3.3.2.2 minor `uint8_t agc_version_t::minor`

Minor version.

3.3.2.3 patch `uint8_t agc_version_t::patch`

Patch version.