AS7050 Application Manager

API Documentation
revision v3.0.2

Generated by Doxygen

Contents

1 Module Documentation

1.1 Application Manager

The Application Manager is a component of the AS7050 Software Support Package that handles and executes all Vital Signs Applications in a platform independent way. It manages the execution of Vital Signs Applications and passes Chip Library FIFO data, accelerometer measurement data, and AGC status information to the individual Vital Signs Applications.

Macros

- #define AS7050_APPMGR_VER_MAJOR 3
- #define AS7050_APPMGR_VER_MINOR 0
- #define AS7050_APPMGR_VER_PATCH 2

Functions

- err_code_t as7050_appmgr_initialize (void)

  Initializes the Application Manager.
- err_code_t as7050_appmgr_enable_apps (uint32_t enabled_apps)

  Sets the enabled Vital Signs Applications.
- err_code_t as7050_appmgr_set_signal_routing (as7050_appmgr_app_id_t app, const as7050_appmgr_↩ channel_id_t ∗p_channels, uint8_t channels_num)

  Sets the signal routing for a Vital Signs Application.
- err_code_t as7050_appmgr_configure_app (as7050_appmgr_app_id_t app, const void ∗p_config, uint8_↩ t size)

  Configures a Vital Signs Application.
- err_code_t as7050_appmgr_start_processing (as7050_meas_config_t measurement_config, uint32_t acc↩ _sample_period_us, const as7050_appmgr_channel_id_t ∗p_agc_mappings, uint8_t agc_mappings_num)

  Starts processing.
- err_code_t as7050_appmgr_set_input (const uint8_t ∗p_fifo_data, uint16_t fifo_data_size, as7050_appmgr_chip_status_t chip_status, const bio_agc_status_t ∗p_agc_statuses, uint8_t agc_statuses_num, const vs_acc_data_t ∗p↩ _acc_samples, uint16_t acc_samples_num, uint32_t ∗p_ready_for_execution)

  Provides measurement data to the Application Manager.
- err_code_t as7050_appmgr_set_ext_event_occurred (void)

  Informs the Application Manager that an external event occurred.
- err_code_t as7050_appmgr_execute (uint32_t ∗p_data_available)

  Executes enabled Vital Signs Applications.
- err_code_t as7050_appmgr_get_output (as7050_appmgr_app_id_t app, void ∗p_dest, uint16_t ∗p_size)

  Writes output of a Vital Signs Application to a buffer provided by the caller.
- err_code_t as7050_appmgr_stop_processing (void)

  Stops processing.
- err_code_t as7050_appmgr_shutdown (void)

  De-initializes the Application Manager.
- err_code_t as7050_appmgr_get_version (as7050_appmgr_version_t ∗p_version)

  Gets the version of the Application Manager.

1.1.1 Detailed Description

The Application Manager is a component of the AS7050 Software Support Package that handles and executes all Vital Signs Applications in a platform independent way. It manages the execution of Vital Signs Applications and passes Chip Library FIFO data, accelerometer measurement data, and AGC status information to the individual Vital Signs Applications.

The Application Manager routes measurement data to Vital Signs Applications and provides a unified interface for application configuration and obtaining resulting output data.

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

- After initialization, the component is configured based on sensor configuration and based on the Vital Signs Applications that shall be executed.
- After the configuration has been set, the component enters the Processing state.
- While in the Processing state, measurement data from the Chip Library, accelerometer measurement data, and AGC status information can be provided to the component at any time. To each enabled Vital Signs Application, the required subset of this data is forwarded. Vital Signs Applications queue the data internally for processing and indicate whether enough data has been received for execution.
- A function to execute enabled Vital Signs Applications can be called anytime while in the Processing state. The Vital Signs Applications process the queued input data and generate output data, that is also for output internally by the corresponding Vital Signs Application. It is indicated to the caller of the function whether output has been generated.
- Queued output data can be collected via separate function, that can also be called anytime while in the Processing state.
- After the measurement has been stopped, the component exits the Processing state. The configuration of the component can now be updated again.

This design allows for efficient execution of the Vital Signs Applications on any platform. Basic embedded systems can call the functions of the Application Manager in a simple super loop, while more advanced systems can call use the Application Manager from multiple threads.

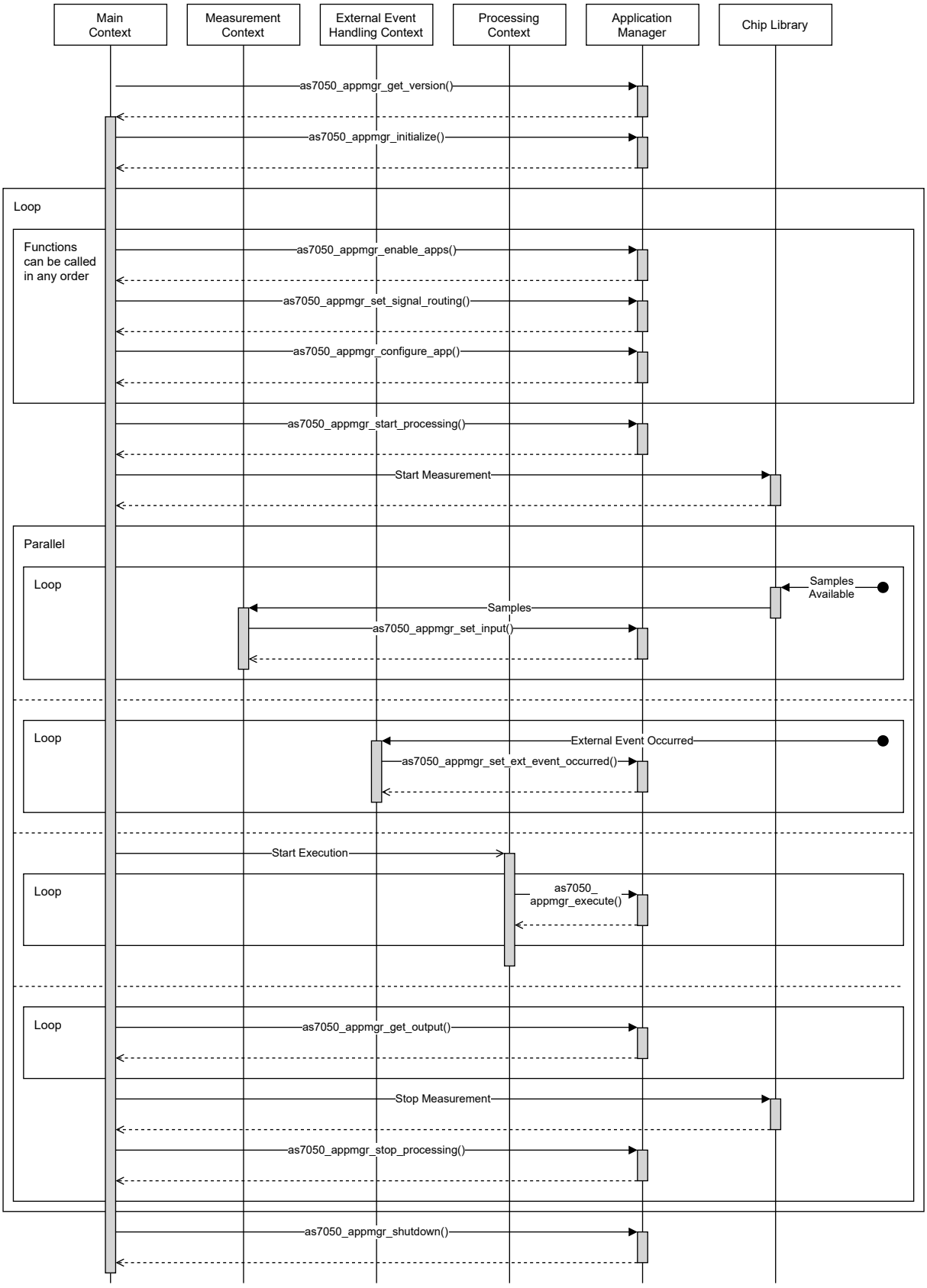

Figure 1 Sequence Diagram

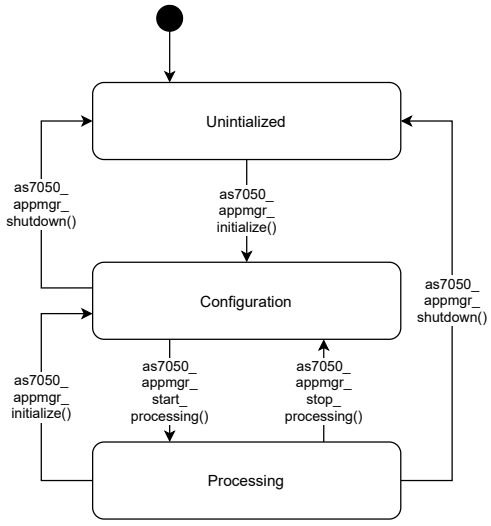

Figure 2 State Diagram

1.1.2 Macro Definition Documentation

1.1.2.1 AS7050_APPMGR_VER_MAJOR `#define AS7050_APPMGR_VER_MAJOR 3`

Major version of the Application Manager.

1.1.2.2 AS7050_APPMGR_VER_MINOR `#define AS7050_APPMGR_VER_MINOR 0`

Minor version of the Application Manager.

1.1.2.3 AS7050_APPMGR_VER_PATCH `#define AS7050_APPMGR_VER_PATCH 2`

Patch version of the Application Manager.

1.1.3 Function Documentation

1.1.3.1 as7050_appmgr_initialize() `err_code_t as7050_appmgr_initialize ( void )`

Initializes the Application Manager.

The Application Manager transitions to Configuration state after initialization.

Return values

ERR_SUCCESS	Initialized successfully.

1.1.3.2 as7050_appmgr_enable_apps()

```
err_code_t as7050_appmgr_enable_apps (
    uint32_t enabled_apps )
```

Sets the enabled Vital Signs Applications.

This function can only be called when the Application Manager is in Configuration state.

Parameters

in	*enabled_apps*	Flags of enabled Vital Signs Applications, see Application Flags. A Vital Signs Application is enabled when the corresponding bit is set and disabled when the bit is not set. At least one app must be enabled. AS7050_APPMGR_APP_FLAG_HRM_A0 and AS7050_APPMGR_APP_FLAG_SPO2_A0 cannot be enabled at the same time.

Return values

ERR_SUCCESS	Updated successfully.
ERR_ARGUMENT	Invalid selection of application.
ERR_PERMISSION	Invalid state.

1.1.3.3 as7050_appmgr_set_signal_routing()

```
err_code_t as7050_appmgr_set_signal_routing (
    as7050_appmgr_app_id_t app,
    const as7050_appmgr_channel_id_t * p_channels,
    uint8_t channels_num )
```

Sets the signal routing for a Vital Signs Application.

Each Vital Signs Application has an ordered list of signals assigned, which can be found in Supported Vital Signs Applications. When as7050_appmgr_set_input is called, the Application Manager extracts the samples of each signal of each enabled Vital Signs Application. In order to extract samples from the provided sensor FIFO data, the Application Manager requires a mapping between sensor channels and Vital Signs Application signals. This mapping is provided to the Application Manager for each Vital Signs Application using this function.

No signal routing is required for AS7050_APPMGR_APP_ID_RAW.

This function can only be called when the Application Manager is in Configuration state.

Parameters

in	*app*	Identifier of the Vital Signs Application for which the signal routing shall be set.

Parameters

in	*p_channels*	Pointer to the start of an array containing the channel identifiers used for each signal of the given Vital Signs Application, in the order specified in the subsections of Supported Vital Signs Applications. Can be NULL if channels_num is zero.
in	*channels_num*	Number of items contained in the p_channels array. This number must be equal to the signal count of the given Vital Signs Application, which can be found in the subsections of Supported Vital Signs Applications.

Return values

ERR_SUCCESS	Updated successfully.
ERR_ARGUMENT	Invalid application identifier or invalid channel identifiers.
ERR_SIZE	Mismatching number of channels.
ERR_POINTER	Invalid pointer argument value.
ERR_PERMISSION	Invalid state.

1.1.3.4 as7050_appmgr_configure_app()

```
err_code_t as7050_appmgr_configure_app (
    as7050_appmgr_app_id_t app,
    const void * p_config,
    uint8_t size )
```

Configures a Vital Signs Application.

This function can only be called when the Application Manager is in Configuration state.

Parameters

in	*app*	Identifier of the Vital Signs Application to configure.
in	*p_config*	Pointer to the configuration structure for the given Vital Signs Application, which can found in the subsections of Supported Vital Signs Applications.
in	*size*	Size of the configuration structure.

Return values

ERR_SUCCESS	Updated successfully.
ERR_ARGUMENT	Invalid application identifier or mismatching configuration structure size.
ERR_POINTER	Invalid pointer argument value.
ERR_PERMISSION	Invalid state.

1.1.3.5 as7050_appmgr_start_processing()

```
err_code_t as7050_appmgr_start_processing (
    as7050_meas_config_t measurement_config,
```

```
uint32_t acc_sample_period_us,
const as7050_appmgr_channel_id_t * p_agc_mappings,
uint8_t agc_mappings_num )
```

Starts processing.

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

Parameters

in	*measurement_config*	Measurement configuration used to acquire the data that will be provided to the Application Manager. This value is typically obtained from the Chip Library.
in	*acc_sample_period_us*	Sample period of the accelerometer in microseconds.
in	*p_agc_mappings*	Pointer to the start of an array containing the identifiers of the channels controlled by Automatic Gain Control (AGC). When providing AGC status information via as7050_appmgr_set_input, the status information items must be ordered identically. For example, if p_agc_mappings[0] contains the identifer of the first channel, then the AGC status information for the first channel must be provided to as7050_appmgr_set_input via the first item of the AGC status information array. Can be NULL if agc_mappings_num is zero.
in	*agc_mappings_num*	Number of items in the p_agc_mappings array.

Return values

ERR_SUCCESS	Updated successfully.
ERR_CONFIG	Measurement configuration incompatible with application configurations.
ERR_ARGUMENT	Invalid accelerometer sample period.
ERR_PERMISSION	Invalid state.

1.1.3.6 as7050_appmgr_set_input()

```
err_code_t as7050_appmgr_set_input (
const uint8_t * p_fifo_data,
uint16_t fifo_data_size,
as7050_appmgr_chip_status_t chip_status,
const bio_agc_status_t * p_agc_statuses,
uint8_t agc_statuses_num,
const vs_acc_data_t * p_acc_samples,
uint16_t acc_samples_num,
uint32_t * p_ready_for_execution )
```

Provides measurement data to the Application Manager.

This function can only be called when the Application Manager is in Processing state.

Parameters

in	*p_fifo_data*	Pointer to the start of the sensor FIFO data. This data is typically obtained from the Chip Library.
in	*fifo_data_size*	Size of the FIFO data.
in	*chip_status*	Chip status. This value is typically obtained from the Chip Library.
in	*p_agc_statuses*	Pointer to the start of an array containing Automatic Gain Control (AGC) status information for each channel with enabled AGC. The order of the items must be identical to the order of items in the AGC mapping array provided to as7050_appmgr_start_processing. Can be NULL if agc_statuses_num is zero.
in	*agc_statuses_num*	Number of AGC status information items. Must be equivalent to the number of AGC mappings provided to as7050_appmgr_start_processing.
in	*p_acc_samples*	Pointer to the start of an array containing accelerometer samples.
in	*acc_samples_num*	Number of accelerometer samples.
out	*p_ready_for_execution*	Flags of Vital Signs Applications that indicated that they are ready for execution, see Application Flags. An app is ready for execution when the corresponding bit is set.

Return values

ERR_SUCCESS	Measurement data accepted.
ERR_SIZE	Too many samples in the FIFO data or invalid number of AGC status information items.
ERR_OVERFLOW	Signals of an app have sample counts which are too different to be handled.
ERR_POINTER	Invalid pointer argument value.
ERR_PERMISSION	Invalid state.

1.1.3.7 as7050_appmgr_set_ext_event_occurred()

```
err_code_t as7050_appmgr_set_ext_event_occurred
(
            void  )
```

Informs the Application Manager that an external event occurred.

The function counts how many times it has been called and provides the count to applications on the next invocation of as7050_appmgr_set_input. This function can only be called when the Application Manager is in Processing state.

Return values

ERR_SUCCESS	Information accepted.
ERR_OVERFLOW	Too many events occurred since last as7050_appmgr_set_input call.
ERR_PERMISSION	Invalid state.

1.1.3.8 as7050_appmgr_execute()

```
err_code_t as7050_appmgr_execute (
        uint32_t * p_data_available )
```

Executes enabled Vital Signs Applications.

This function can only be called when the Application Manager is in Processing state.

Parameters

out	*p_data_available*	Flags of Vital Signs Applications that indicated that their execution generated output data, see Application Flags. An app has output data available when the corresponding bit is set.

Return values

ERR_SUCCESS	Execution successful.
ERR_POINTER	Invalid pointer argument value.
ERR_PERMISSION	Invalid state.

1.1.3.9 as7050_appmgr_get_output()

```
err_code_t as7050_appmgr_get_output (
        as7050_appmgr_app_id_t app,
        void * p_dest,
        uint16_t * p_size )
```

Writes output of a Vital Signs Application to a buffer provided by the caller.

This function can only be called when the Application Manager is in Processing state.

Parameters

in	*app*	Identifier of the Vital Signs Application to get output from.
out	*p_dest*	Pointer to the buffer where the output shall be written to. The output is application-specific and is described in the subsections of Supported Vital Signs Applications.
in,out	*p_size*	Pointer to the amount of memory allocated for the output data. The function updates the value pointed to with the actual size of the written output data.

Return values

ERR_SUCCESS	Output data write successful.
ERR_ARGUMENT	Invalid app identifier or disabled application.
ERR_POINTER	Invalid pointer argument value.
ERR_PERMISSION	Invalid state.

1.1.3.10 as7050_appmgr_stop_processing() err_code_t as7050_appmgr_stop_processing (void)

Stops processing.

This function can only be called when the Application Manager is not in Uninitialized state.

Return values

ERR_SUCCESS	Stop successful.
ERR_PERMISSION	Invalid state.

1.1.3.11 as7050_appmgr_shutdown() err_code_t as7050_appmgr_shutdown (void)

De-initializes the Application Manager.

Return values

ERR_SUCCESS	De-initialization successful.

1.1.3.12 as7050_appmgr_get_version() err_code_t as7050_appmgr_get_version (as7050_appmgr_version_t * *p_version*)

Gets the version of the Application Manager.

Parameters

out	*p_version*	Pointer to the location where the version shall be written to.

Return values

ERR_SUCCESS	Version write successful.
ERR_POINTER	Invalid pointer argument value.

1.2 Supported Vital Signs Applications

Modules

- Raw Data Streaming
- ams HRM Algorithm
- ams SpO2 Algorithm
- Galvanic Skin Resistance
- Application Flags

  These definitions are used by the as7050_appmgr_enable_apps, as7050_appmgr_set_input, and as7050_appmgr_execute functions. Use these definitions with bitwise operators to set, clear, or read the flag bit corresponding to a given Vital Signs Application.

Typedefs

- typedef uint8_t as7050_appmgr_app_id_t

Enumerations

- enum as7050_appmgr_app_id {
  AS7050_APPMGR_APP_ID_RAW = 0,
  AS7050_APPMGR_APP_ID_HRM_A0,
  AS7050_APPMGR_APP_ID_SPO2_A0,
  AS7050_APPMGR_APP_ID_GSR,
  AS7050_APPMGR_APP_ID_NUM }

1.2.1 Detailed Description

The Application Manager supports four Vital Signs Applications:

- Raw Data Streaming
- ams HRM Algorithm
- ams SpO2 Algorithm
- Galvanic Skin Resistance

The Application Manager provides a common API to all supported Vital Signs Applications. Each application is assigned an as7050_appmgr_app_id identifier, which is used in the Application Manager APIs.

1.2.2 Typedef Documentation

1.2.2.1 as7050_appmgr_app_id_t `typedef uint8_t as7050_appmgr_app_id_t`

Type for as7050_appmgr_app_id.

1.2.3 Enumeration Type Documentation

1.2.3.1 as7050_appmgr_app_id `enum as7050_appmgr_app_id`

Identifiers of Vital Signs Applications.

Enumerator

AS7050_APPMGR_APP_ID_RAW	Identifies the Raw Data Streaming application.
AS7050_APPMGR_APP_ID_HRM_A0	Identifies the ams HRM Algorithm application.
AS7050_APPMGR_APP_ID_SPO2_A0	Identifies the ams SpO2 Algorithm application.
AS7050_APPMGR_APP_ID_GSR	Identifies the Galvanic Skin Resistance application.
AS7050_APPMGR_APP_ID_NUM	Number of Vital Signs Applications.

1.3 Application Flags

These definitions are used by the as7050_appmgr_enable_apps, as7050_appmgr_set_input, and as7050_appmgr_execute functions. Use these definitions with bitwise operators to set, clear, or read the flag bit corresponding to a given Vital Signs Application.

Macros

- #define M_AS7050_APPMGR_APP_FLAG(app) (1 << (app))
- #define AS7050_APPMGR_APP_FLAG_RAW M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_RAW)
- #define AS7050_APPMGR_APP_FLAG_HRM_A0 M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_HRM_A0)
- #define AS7050_APPMGR_APP_FLAG_SPO2_A0 M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_SPO2_A
- #define AS7050_APPMGR_APP_FLAG_GSR M_AS7050_APPMGR_APP_FLAG(AS7050_APPMGR_APP_ID_GSR)

1.3.1 Detailed Description

These definitions are used by the as7050_appmgr_enable_apps, as7050_appmgr_set_input, and as7050_appmgr_execute functions. Use these definitions with bitwise operators to set, clear, or read the flag bit corresponding to a given Vital Signs Application.

1.3.2 Macro Definition Documentation

1.3.2.1 M_AS7050_APPMGR_APP_FLAG `#define M_AS7050_APPMGR_APP_FLAG( app ) (1 << (app))`

Macro to create a application flag bitmask value with the bit for the given as7050_appmgr_app_id_t set.

1.3.2.2 AS7050_APPMGR_APP_FLAG_RAW `#define AS7050_APPMGR_APP_FLAG_RAW M_AS7050_APPMGR_APP_FLAG(AS7050_A`

Application flag bitmask value with the bit for AS7050_APPMGR_APP_ID_RAW set.

1.3.2.3 AS7050_APPMGR_APP_FLAG_HRM_A0 `#define AS7050_APPMGR_APP_FLAG_HRM_A0 M_AS7050_APPMGR_APP_FLAG(AS`

Application flag bitmask value with the bit for AS7050_APPMGR_APP_ID_HRM_A0 set.

1.3.2.4 AS7050_APPMGR_APP_FLAG_SPO2_A0 `#define AS7050_APPMGR_APP_FLAG_SPO2_A0 M_AS7050_APPMGR_APP_FLAG(`

Application flag bitmask value with the bit for AS7050_APPMGR_APP_ID_SPO2_A0 set.

1.3.2.5 AS7050_APPMGR_APP_FLAG_GSR `#define AS7050_APPMGR_APP_FLAG_GSR M_AS7050_APPMGR_APP_FLAG(AS7050_A`

Application flag bitmask value with the bit for AS7050_APPMGR_APP_ID_GSR set.

1.4 Raw Data Streaming

Data Structures

- struct raw_app_configuration_t

1.4.1 Detailed Description

The output of the raw app includes FIFO data as received from the AS7050 device, accelerometer samples, and Automatic Gain Correction (AGC) status information. Accelerometer samples are only included in the output when enabled in the raw_app_configuration_t. AGC status information is only transmitted when the status information has changed since the last transmission.

The output has the following format:

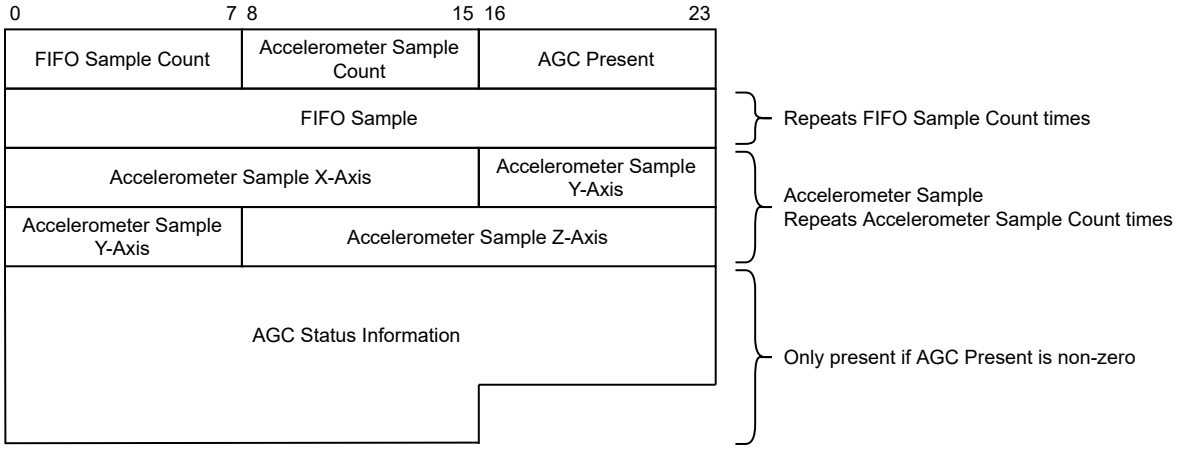

Figure 3 Raw App Output

The header fields (FIFO Sample Count, Accelerometer Sample Count, AGC Present) each have a size of 1 byte. Each FIFO Sample has a size of 3 bytes. The FIFO Sample field repeats FIFO Sample Count times. An Accelerometer Sample has a total size of 6 bytes. The Accelerometer Sample field repeats Accelerometer Sample Count times. The AGC status information has a size of 8 bytes and is only present if the AGC Present field has a non-zero value.

1.5 ams HRM Algorithm

Data Structures

- struct bio_hrm_a0_configuration_t
- struct bio_hrm_a0_output_t

Macros

- #define PRV_DATA_NUM 5

Enumerations

- enum bio_hrm_a0_signal {
  BIO_HRM_A0_SIGNAL_PPG = 0,
  BIO_HRM_A0_SIGNAL_NUM }

1.5.1 Detailed Description

1.5.2 Macro Definition Documentation

1.5.2.1 PRV_DATA_NUM `#define PRV_DATA_NUM 5`

Maximum number of PRV data items in a single bio_hrm_a0_output_t structure.

1.5.3 Enumeration Type Documentation

1.5.3.1 bio_hrm_a0_signal `enum bio_hrm_a0_signal`

Sensor signals provided to the HRM bio app.

Enumerator

BIO_HRM_A0_SIGNAL_PPG	PPG signal.
BIO_HRM_A0_SIGNAL_NUM	Number of sensor signals provided to the bio app.

1.6 ams SpO2 Algorithm

Data Structures

- struct bio_spo2_a0_configuration_t
- struct bio_spo2_a0_output_t

Enumerations

- enum bio_spo2_a0_signal {
  BIO_SPO2_A0_SIGNAL_PPG_RED = 0,
  BIO_SPO2_A0_SIGNAL_PPG_IR = 1,
  BIO_SPO2_A0_SIGNAL_AMBIENT = 2,
  BIO_SPO2_A0_SIGNAL_NUM }

1.6.1 Detailed Description

1.6.2 Enumeration Type Documentation

1.6.2.1 bio_spo2_a0_signal `enum bio_spo2_a0_signal`

Sensor signals provided to the SpO2 bio app.

Enumerator

BIO_SPO2_A0_SIGNAL_PPG_RED	Red PPG signal.
BIO_SPO2_A0_SIGNAL_PPG_IR	Infrared PPG signal.
BIO_SPO2_A0_SIGNAL_AMBIENT	Ambient light signal.
BIO_SPO2_A0_SIGNAL_NUM	Number of sensor signals provided to the bio app.

1.7 Galvanic Skin Resistance

Data Structures

- struct gsr_app_configuration_t
- struct gsr_app_output_t

Enumerations

- enum gsr_app_signal {
  GSR_APP_SIGNAL_ECG = 0,
  GSR_APP_SIGNAL_NUM }

1.7.1 Detailed Description

The output of the GSR app includes the measured Galvanic Skin Resistance value. For correct GSR measurement, the DAC reference value of the used AS7050 device needs to be determined using the corresponding function of the AS7050 Chip Library. The acquired DAC reference value needs to be provided to the GSR app via the app configuration.

1.7.2 Enumeration Type Documentation

1.7.2.1 gsr_app_signal `enum gsr_app_signal`

Sensor signals provided to the GSR app.

Enumerator

GSR_APP_SIGNAL_ECG	ECG signal.
GSR_APP_SIGNAL_NUM	Number of sensor signals provided to the GSR app.

1.8 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 }

1.8.1 Detailed Description

Generic error codes used by ams libraries.

1.8.2 Typedef Documentation

1.8.2.1 err_code_t `typedef enum error_codes err_code_t`

This definition will be used for function return values.

1.8.3 Enumeration Type Documentation

1.8.3.1 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 • ...
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.

Enumerator

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

2 Data Structure Documentation

2.1 as7050_appmgr_chip_status_t Struct Reference

Chip-specific chip status information type, which may include information such as interrupt status. This chip status is provided to as7050_appmgr_set_input and is intended to be forwarded to the raw data application.

2.1.1 Detailed Description

Chip-specific chip status information type, which may include information such as interrupt status. This chip status is provided to as7050_appmgr_set_input and is intended to be forwarded to the raw data application.

This structure is unused as the output of the raw data application does not include such fields.

2.2 as7050_appmgr_version_t Struct Reference

Data Fields

- uint8_t major
- uint8_t minor
- uint8_t patch

2.2.1 Detailed Description

Describes the version of the Application Manager.

2.2.2 Field Documentation

2.2.2.1 major `uint8_t as7050_appmgr_version_t::major`

Major version

2.2.2.2 minor `uint8_t as7050_appmgr_version_t::minor`

Minor version

2.2.2.3 patch `uint8_t as7050_appmgr_version_t::patch`

Patch version

2.3 bio_hrm_a0_configuration_t Struct Reference

Data Fields

- uint8_t enable_prv

2.3.1 Detailed Description

Contains the configuration options of the HRM bio app.

2.3.2 Field Documentation

2.3.2.1 enable_prv `uint8_t bio_hrm_a0_configuration_t::enable_prv`

PRV data is included in the output when set to TRUE.

2.4 bio_hrm_a0_output_t Struct Reference

Data Fields

- uint16_t heart_rate
- uint8_t quality
- uint8_t motion_frequency
- uint16_t prv_ms [PRV_DATA_NUM]
- uint8_t prv_ms_num
- uint8_t reserved

2.4.1 Detailed Description

Describes the output of the HRM bio app.

2.4.2 Field Documentation

2.4.2.1 heart_rate `uint16_t bio_hrm_a0_output_t::heart_rate`

Contains the heart rate. The unit is 0.1 bpm.

2.4.2.2 quality `uint8_t bio_hrm_a0_output_t::quality`

Contains information about the quality of the heart rate signal. A value of zero means best quality.

2.4.2.3 motion_frequency `uint8_t bio_hrm_a0_output_t::motion_frequency`

Contains the detected motion frequency. The unit is bpm. A value of zero means that no motion has been detected.

2.4.2.4 prv_ms `uint16_t bio_hrm_a0_output_t::prv_ms[PRV_DATA_NUM]`

Contains PRV data. The unit is milliseconds.

2.4.2.5 prv_ms_num `uint8_t bio_hrm_a0_output_t::prv_ms_num`

Contains the number of prv_ms fields that are used.

2.4.2.6 reserved `uint8_t bio_hrm_a0_output_t::reserved`

Padding byte.

2.5 bio_spo2_a0_configuration_t Struct Reference

Data Fields

- uint16_t a
- uint16_t b
- uint16_t c
- uint16_t dc_comp_red
- uint16_t dc_comp_ir

2.5.1 Detailed Description

Contains the configuration options of the SpO2 bio app.

2.5.2 Field Documentation

2.5.2.1 a `uint16_t bio_spo2_a0_configuration_t::a`

Quadratic correction coefficient a.

2.5.2.2 b `uint16_t bio_spo2_a0_configuration_t::b`

Quadratic correction coefficient b.

2.5.2.3 c `uint16_t bio_spo2_a0_configuration_t::c`

Quadratic correction coefficient c.

2.5.2.4 dc_comp_red `uint16_t bio_spo2_a0_configuration_t::dc_comp_red`

DC compensation for the red signal.

2.5.2.5 dc_comp_ir `uint16_t bio_spo2_a0_configuration_t::dc_comp_ir`

DC compensation for the infrared signal.

2.6 bio_spo2_a0_output_t Struct Reference

Data Fields

- uint8_t status
- uint8_t quality
- uint16_t spo2
- uint16_t heart_rate
- uint16_t pi
- uint16_t average_r
- uint16_t ac_comp_red
- uint16_t dc_comp_red
- uint16_t ac_comp_ir
- uint16_t dc_comp_ir

2.6.1 Detailed Description

Describes the output of the SpO2 bio app.

2.6.2 Field Documentation

2.6.2.1 status `uint8_t bio_spo2_a0_output_t::status`

The value of the field is zero when the structure contains valid SpO2 data. It is one when no result is present in the data.

2.6.2.2 quality `uint8_t bio_spo2_a0_output_t::quality`

Contains the quality of the signal. The unit is 1%.

2.6.2.3 spo2 `uint16_t bio_spo2_a0_output_t::spo2`

Contains the SpO2 measurement. The unit is 0.01%.

2.6.2.4 heart_rate `uint16_t bio_spo2_a0_output_t::heart_rate`

Contains the heart rate. The unit is 0.1 bpm.

2.6.2.5 pi `uint16_t bio_spo2_a0_output_t::pi`

Contains the Perfusion Index measurement. The unit is 0.01%.

2.6.2.6 average_r `uint16_t bio_spo2_a0_output_t::average_r`

Contains the Average R value. The unit is 10000.

2.6.2.7 ac_comp_red `uint16_t bio_spo2_a0_output_t::ac_comp_red`

Contains the AC component of the red PPG signal. It is used for calibration purposes.

2.6.2.8 dc_comp_red `uint16_t bio_spo2_a0_output_t::dc_comp_red`

Contains the DC component of the red PPG signal. It is used for calibration purposes.

2.6.2.9 ac_comp_ir `uint16_t bio_spo2_a0_output_t::ac_comp_ir`

Contains the AC component of the infrared PPG signal. It is used for calibration purposes.

2.6.2.10 dc_comp_ir `uint16_t bio_spo2_a0_output_t::dc_comp_ir`

Contains the DC component of the infrared PPG signal. It is used for calibration purposes.

2.7 gsr_app_configuration_t Struct Reference

Data Fields

- uint32_t dac_ref

2.7.1 Detailed Description

Configuration structure of the GSR app.

2.7.2 Field Documentation

2.7.2.1 dac_ref `uint32_t gsr_app_configuration_t::dac_ref`

DAC reference value which must be determined before.

2.8 gsr_app_output_t Struct Reference

Data Fields

- uint32_t resistor

2.8.1 Detailed Description

Describes the output of the GSR app.

2.8.2 Field Documentation

2.8.2.1 resistor `uint32_t gsr_app_output_t::resistor`

Contains the resistance value in Ohm. The output is invalid if the value of this field is 0xFFFFFFFF.

2.9 raw_app_configuration_t Struct Reference

Data Fields

- uint8_t include_acc

2.9.1 Detailed Description

Configuration structure of the raw app.

2.9.2 Field Documentation

2.9.2.1 include_acc `uint8_t raw_app_configuration_t::include_acc`

A non-zero value indicates that accelerometer data shall be included in the output data.