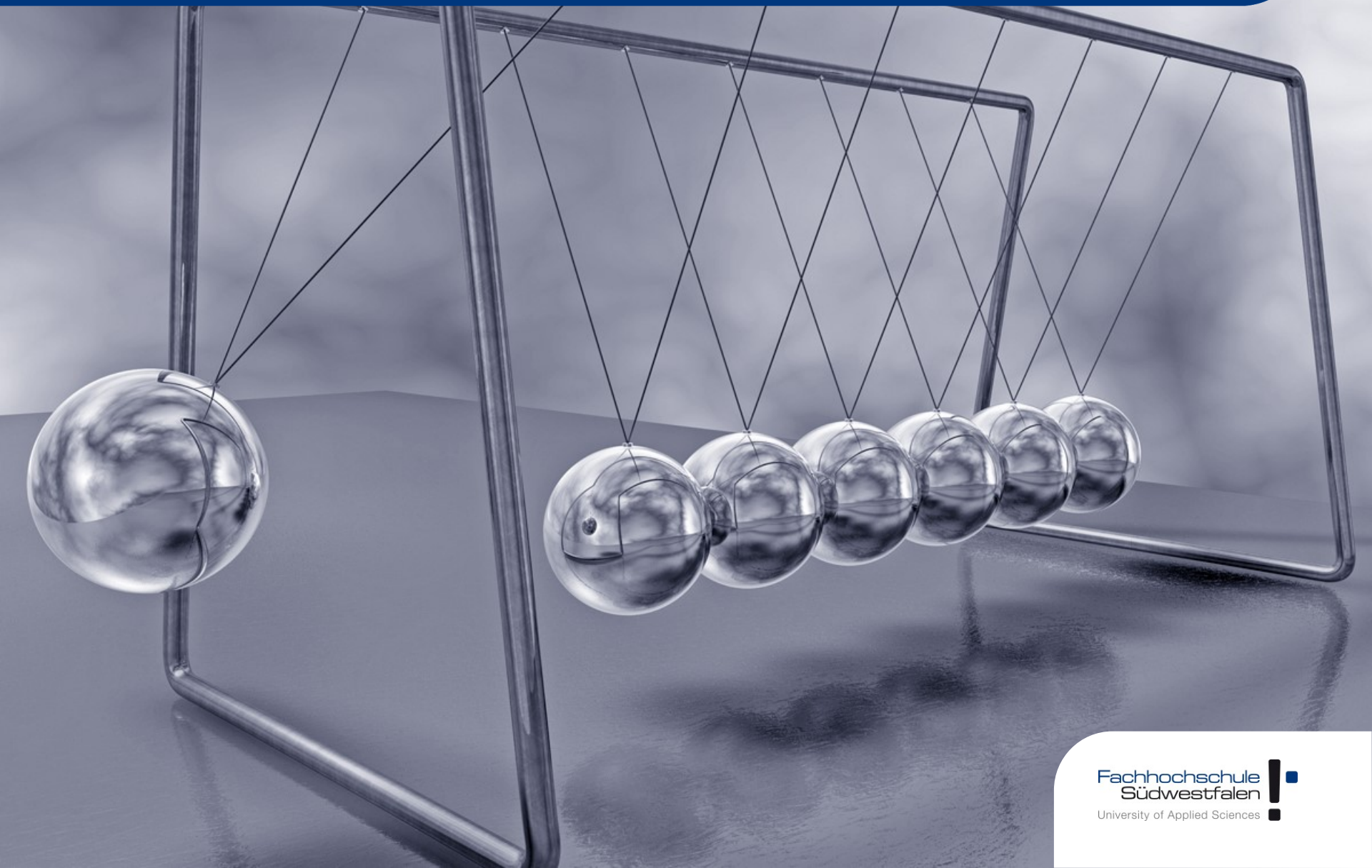
Acceleration Logging

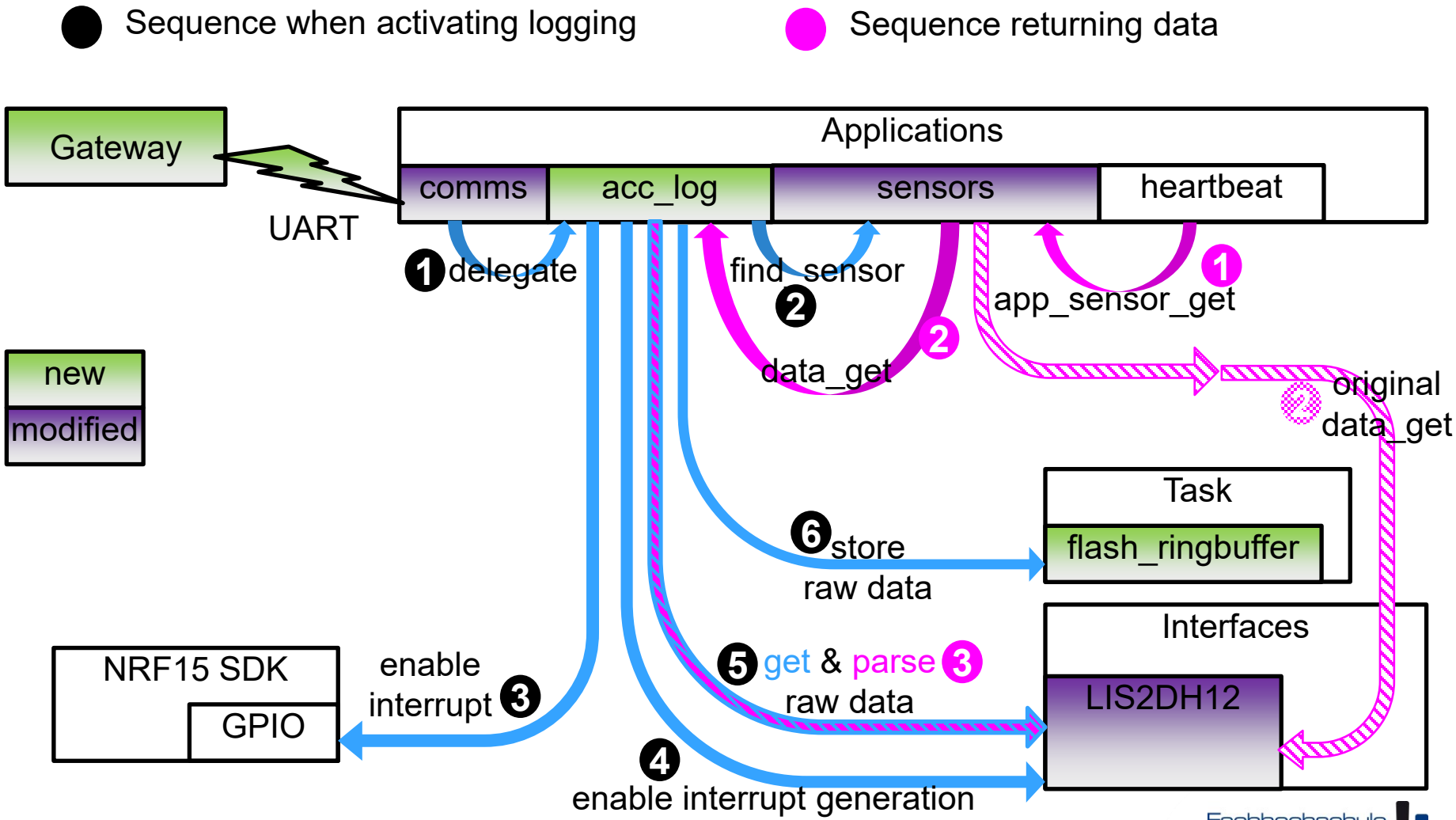
Concept, Implementation, GATT Interface



Fachhochschule Südwestfalen
University of Applied Sciences

Concept

Wrapping of data_get



Files (1)

Name	Status	
app_accelerometer_logging.*	new	Main part for acceleration logging.
app_comms.c	modified	GATT message added.
app_config.h	modified	Added macro for conditional compiling.
app_heartbeat.c	modified	Disable environment logging when acceleration logging.
app_sensor.*	modified	Function for finding sensor context added.
main.c	modified	Initialization of acceleration logging added.

Files (2)

Name	Status	
ruuvi_endpoint.h	todo	New constants for messages must be defined.
ruuvi_interface_lis2dh12.*	modified	Access to raw acceleration data added. Split up data_get() into getting data and parsing data.
ruuvi_interface_rtc.h ruuvi_nrf5_sdk15_rtc_mcu.c	modified	Function for setting RTC added.

Implementation

app_accelerometer_logging.c

new

`rd_status_t app_enable_sensor_logging(void)`
`rd_status_t app_disable_sensor_logging(void)`

Enables/Disables the logging of acceleration data by executing the following steps.

1. Enable/disable GPIO interrupt.
2. Enable/disable FIFO on sensor.
3. Enable/disable generating interrupt on sensor.
4. Save/restore original `data_get()` function in sensor context and replace it by new function `lis2dh12_logged_data_get()`.

returns	Status code of executing the function.

Special error codes	
RD_ERROR_INVALID_STATE	When logging is already enabled/disabled.
RD_ERROR_NOT_FOUND	When LIS2DH12 is not available.

Implementation

app_accelerometer_logging.c

new

```
void on_fifo_full (const ri_gpio_evt_t evt)
void fifo_full_handler (void * p_event_data,
    uint16_t event_size)
```

The two functions together form the interrupt handler. When FIFO in LIS2DH12 is full the interrupt triggers `on_fifo_full()`. This function schedules the execution of `fifo_full_handler()` outside interrupt context.

The function `fifo_full_handler()` reads the FIFO and stores the data inside the ringbuffer.

See ruuvi_interface_scheduler.h and ruuvi_interface_gpio_interrupt.h for parameters used in these functions.

Implementation

app_accelerometer_logging.c

new

```
void pack8/10/12(const uint16_t sizeData,
    const uint8_t* const data,
    uint8_t* const packeddata)
```

These functions store raw accelerometer values in 8/10/12 Bit format in compact form (without unused bits).

`sizeData`	in	Size of input data.
`data`	in	Input data.
`packeddata`	in/out	Memory for storing packed data.

Implementation

app_accelerometer_logging.c

new

```
rd_status_t lis2dh12_logged_data_get (
    rd_sensor_data_t * const data)
```

This function retrieves raw accelerometer values from RAM. The values are parsed and returned inside data. It is called by `app_sensor_get()` inside app_sensor.c when accelerometer logging is active.

`raw_data`	in/out	Memory for storing accelerometer values.
returns	Status code of executing this function.	

Implementation

app_accelerometer_logging.c

new

`rd_status_t app_acc_logging_send_last_sample( const ri_comm_xfer_fp_t reply_fp)`		
This function is called when a request to send the last sample is received by GATT/UART. It retrieves the last sample from RAM, does the compacting of the bits by calling `pack8/10/12()` and sends the data to the requestor.		
`reply_fp`	in	Function pointer to reply function.
returns	Status code of processing the message.	

Special error codes	
RD_ERROR_INVALID_STATE	When logging is not active.

Implementation

app_accelerometer_logging.c

new

`rd_status_t app_acc_logging_state(void)`	
This function is used to query the state of accelerometer logging. It is called when a control message is received by GATT/UART to return this state to the caller.	
returns	Status code regarding the state of accelerometer logging.

Special error codes	
RD_SUCCESS	When logging is active.
RD_ERROR_INVALID_STATE	When logging is not active.

Implementation

app_accelerometer_logging.c

new

```
rd_status_t app_acc_logging_configuration_set (
    rt_sensor_ctx_t* sensor,
    rd_sensor_configuration_t* new_config)
```

This function is called when a request to update the sensor configuration is received by GATT/UART. It checks every configuration parameter if it should be changed. It also checks if the value is different than actual value. If a change is detected it clears the ringbuffer, updates the configuration and stores the configuration in flash.

`sensor`	in	Sensor context of the sensor which configuration should be changed.
`new_config`	in	Structure containing the new configuration values.
returns	Status code of processing the message.	

Implementation

app_accelerometer_logging.c

new

`rd_status_t app_acc_logging_init(void)`

Initialize acceleration logging during boot. When logging was active before reboot it will be activated.

When logging was not active before reboot this function return RD_SUCCESS without activating acceleration logging.

The state if logging was active before reboot is detected by looking for the ringbuffer. When the ringbuffer exists, the logging must be active before reboot.

This function is called from main.c / `setup()`.

returns	Status code of processing the message.

Implementation

app_accelerometer_logging.c

new

`rd_status_t app_acc_logging_uninit(void)`	
The uninitialization of acceleration logging disables the logging when it is actually active. When logging is not active this function return RD_SUCCESS without doing anything.	
returns	Status code of processing the message.

Implementation

app_comms.c

modified

```
void handle_comms (
    const ri_comm_xfer_fp_t reply_fp,
    const uint8_t * const raw_message,
    size_t data_len)
```

Added new switch/case which forwards messages regarding configuration and control of acceleration logging to the function `handle_lis2dh12_comms()`.

Implementation

app_comms.c

new

```
rd_status_t handle_lis2dh12_comms (
    const ri_comm_xfer_fp_t reply_fp,
    const uint8_t * const raw_message,
    size_t data_len)
```

This function handles the GATT/UART communication needed to control the functionality of acceleration logging.

`reply_fp`	in	Function pointer to reply function.
`raw_message`	in	Message received.
`data_len`	in	Length of the received message.
returns	Status code of processing the message.	

Implememtation

app_config.h

- Added macro `APP_SENSOR_LOGGING` to control compilation of app_accelerometer_logging.*
- When `APP_SENSOR_LOGGING` is not defined or is defined as 0 the functionality of logging of acceleration data is not available in the application.

Implementation

app_heartbeat.c

modified

```
void heartbeat (void * p_event, uint16_t event_size)
```

When acceleration logging is activated, than logging of environmental data is disabled to avoid extreme fragmentation of flash memory.

Implementation

app_sensor.c

new

```
rt_sensor_ctx_t* app_sensor_find (
    const char *name)
```

Find sensor by it's name. Works only with initialized sensors, will not return a sensor which is supported in firmware but not initialized.

This function is called by `app_enable_sensor_logging()` / `app_disable_sensor_logging()` to retrieve the sensor context.

`name`	in	Name of the sensor.
returns	When sensor is found return it's sensor context structure.	

Implementation

main.c

modified

`void setup (void)`

Added call to `app_acc_logging_init()` to initialize acceleration logging when desired.

Implementation

ruuvi_endpoint.h

- This file is not changed yet. But it contains the constants used for the message ID‘s in GATT/UART messages.
- These ID‘s are referenced in app_comms.c
- The ID‘s and the name of the constants must be defined by Otso Jousimaa from Ruuvi.
- The ID‘s which are used when writing this document are temporary and will be changed when permanent ID‘s are assigned by Ruuvi.

Implementation

ruuvi_interface_lis2dh12.c

new

```
rd_status_t ri_lis2dh12_acceleration_raw_get (
    uint8_t * const raw_data)
```

This functions read raw acceleration values from the registers of LIS2DH12. It is called from the interrupt handler inside app_accelerometer_logging and from `ri_lis2dh12_data_get()` inside this module.

`raw_data`	in/out	Memory for storing raw accelerometer values.
returns	RD_SUCCESS: When data could be retrieved from LIS2DH12. RD_ERROR_INTERNAL: In case of error.	

Implementation

ruuvi_interface_lis2dh12.c

modified

```
rd_status_t ri_lis2dh12_data_get (
    rd_sensor_data_t * const data)
```

The original function `ri_lis2dh12_data_get()` is split into retrieving raw values from the sensor and parsing these data. Parsing is done by `ri_lis2dh12_raw_data_parse()`.

This function is used when the acceleration logging is not active. If acceleration logging is active this function is replaced by `lis2dh12_logged_data_get()` inside app_accelerometer_logging.c.

`data`	in/out	Structure for storing parsed accelerometer values.
returns	Status code of executing this function.	

Implementation

ruuvi_interface_lis2dh12.c

new

```
rd_status_t ri_lis2dh12_raw_data_parse (
    rd_sensor_data_t * const data,
    axis3bit16_t *raw_acceleration,
    uint8_t *raw_temperature)
```

This function parses raw values from the sensor and stores the values inside data. It is called from `ri_lis2dh12_data_get()` and from `lis2dh12_logged_data_get()`.

`data`	in/out	Structure for storing parsed accelerometer values.
`raw_acceration`	in	Raw acceleration values.
`raw_temperature`	in	Raw temperature value. When used from app_accelerometer_logging.c this parameter is NULL.
returns	Status code of executing this function.	

Implementation

ruuvi_nrf5_sdk_rtc_mcu.c / ruuvi_interface_rtc.h

new

`rd_status_t ri_set_rtc_millis(uint64_t millis)`		
Set system time by external source. Set RTC to zero.		
`millis`	in	External time.
returns	RD_SUCCESS when success. RD_ERROR_NOT_INITIALIZED when RTC is not initialized.	

GATT Interface

General usage

Communication between Gateway and Sensor is done via Bluetooth Low Energy. It uses the Nordic UART service which itself uses the Bluetooth GATT protocol.

Three types of messages are used.

1. The Gateway sends control messages to the sensor to set or read configuration or start transmission of logged data.
2. The Sensor responds to control messages via response messages. This message type transports status information or configuration data.
3. If the Gateway requests the Sensor to send logged data, this data is sent by data messages. To transport all data many data messages are used in sequence. The end of data is signaled by a response message.

The messages are differentiated by the first byte which is noted in the table on the next slide in the first line.

In case of a fatal error there may be no response by the sensor.

Control messages must be padded by nullbytes to a minimum length of 11 bytes. This requirement is introduced by the Ruuvi firmware. The padding bytes are not shown in the following description of the messages.

GATT Interface

Control message 0xFA 0xFA + Type		Response message 0xFB + Type				Data Message 0xFC
Type	Message	Status (0x00)	Time (0x09)	Config (0x07)	End of data	
0x01	Testdata	Error			Succes	Succes
0x03	Last Sample	Error			Succes	Succes
0x05	Logged Data	Error			Succes	Succes
0x06	Set Config	Always				
0x07	Get Config			Always		
0x08	Set Time	Always				
0x09	Get Time		Always			
0x0A	Set Logging	Always				
0x0B	Get Logging	Always				

GATT Interface

Status response

A status response is used by the sensor when there are no data to return. This message is used as response to several control messages.

The concrete content of a status response is as follows `0xFB 0x00 SS`.

The byte `SS` contains the information about the status. It must be interpreted as a bitfield. The bits represent the different error conditions.

See file ruuvi_driver_error.h for an explanation of the bits from the status byte.

GATT Interface

Transmit test data

This message starts transmitting 4096 byte of test data.

The purpose of this message is for testing the reliability of transporting data via Bluetooth under different circumstances. The transmission of the data is done via data messages. It is followed by an end of data message which signals the end of the data.

This message takes no parameters. It's concrete content is: `0xFA 0xFA 0x01`.

This message will be removed in a future version.

GATT Interface

Transmit last sample

This message starts transmitting the last sample of acceleration data. This sample consists of 32 tuples of (X,Y,Z) values. It is taken from RAM and transmitted to the gateway by using data messages. The transmission of the data is done via data messages. It is followed by an end of data message which signals the end of the data.

This message takes no parameters. It's concrete content is: `0xFA 0xFA 0x03`.

If acceleration logging is not active, the Sensor responds a status response containing error code `RD_ERROR_INVALID_STATE`.

GATT Interface

Transmit logged data

This message starts transmitting the logged acceleration data from the ringbuffer. The data includes the last page which is not complete full and is not yet written to flash. The transmission of the data is done via data messages. It is followed by an end of data message which signals the end of the data. After downloading the logged data, the ringbuffer is empty.

This message takes no parameters. It's concrete content is: `0xFA 0xFA 0x05`.

If acceleration logging is not active, the Sensor responds a status response containing error code `RD_ERROR_INVALID_STATE`.

GATT Interface

End of data message

This message is returned by the sensor after returning data. It signals the end of the transmission. The type byte (`0x01`, `0x03`, `0x05`) maps to the type of data requested. This message contains nine parameters. The current configuration of the acceleration sensor are the first eight parameter. The structure is the same as shown in the set configuration message. The CRC16 value of the transmitted data is the 9th parameter.

To compute the CRC16 value the polynom `0x11021` with the initial value `0xFFFF` is used. The output bytes are not reversed and not XOR'd. The CRC value is of size 2 bytes. It is transferred in little-endian byte sequence.

The concrete content of this message is
`0xFB 0x03/0x05 SS P1 P2 P3 P4 P5 P6 P7 P8 CRC1 CRC2.`

See "Set configuration" on next slide for description of the parameters P1 to P8.
SS is the status code, see "Status response".

GATT Interface

Set configuration of acceleration sensor

This message is used to set the configuration of the acceleration sensor (LIS2DH12). The message takes 8 Parameters.

It's concrete content is: `0xFA 0xFA 0x06 P1 P2 P3 P4 P5 P6 P7 P8`

Parameter	Description
P1	Rate of sampling in samples per second. Allowed values are 1Hz, 10Hz, 25Hz, 50Hz, 100Hz, 200Hz, 400Hz.
P2	Resolution in bits. Allowed values are 8, 10, 12.
P3	Measuring range. Allowed values are 2G, 4G, 8G, 16G.
P4	DSP function. See datasheet of LIS2DH12.
P5	DSP parameter. See datasheet of LIS2DH12.
P6	Mode of operation. Allowed values are `0xF2`, `0xF3`, `0xF4`.
P7	Reserved. Set to `0x00`.
P8	Reserved. Set to `0x00`.

GATT Interface

Read configuration

This message is used to read the configuration of the acceleration sensor (LIS2DH12). The message takes no parameters. The Sensor responds to this message either by a status response containing an error code or by a response message which transmits the configuration. If the status code signals an error the transmitted values are undefined.

The concrete content of this message is: `0xFA 0xFA 0x07`.

The concrete content of the message which returns the configuration is

`0xFB 0x07 SS P1 P2 P3 P4 P5 P6 P7 P8`.

See “Set configuration” for description of return parameters. `SS` is the status code see “Status response” for explanation.

GATT Interface

Set system time

This message is used to set the RTC of the sensor to a timestamp which is part of the message. The time is expressed in milliseconds. The value must be transmitted in little-endian byte sequence. The sensor responds to this message with a status response.

The concrete content of this message is

```
0xFA 0xFA 0x08 XX XX XX XX XX XX XX XX
```

GATT Interface

Get system time

This message is used to read the RTC of the sensor. The sensor responds to this message with a timestamp response. If the status code signals an error the transmitted value is undefined.

The concrete content of this message is `0xFA 0xFA 0x09`.

The concrete content of a timestamp response is the following `0xFB 0x08 SS XX XX XX XX XX XX XX XX`. Where `SS` is the status code.

The timestamp value is transmitted in little-endian byte sequence.

GATT Interface

Activate/Deactivate logging

This message is used to activate or deactivate acceleration logging. It takes one parameter. The parameter is interpreted as a Boolean value. If it maps to true, acceleration logging is activated. If it maps to false, acceleration logging is deactivated.

The sensor responds to this message using a status response. Activating acceleration logging when it is already active results in an error. Deactivating acceleration logging when it is not active results in an error.

The concrete content of this message is `0xFA 0xFA 0x0A XX`.

Where `XX` can be `0x00` or `0x01`.

GATT Interface

Query state of logging

This message is used to query the status of acceleration logging. The sensor responds to this message with a status response. If logging is active, the response contains the status `RD_SUCCESS` if logging is not active the status is `RD_ERROR_NOT_INITIALIZED`.

The concrete content of this message is `0xFA 0xFA 0x0B`.

GATT Interface

Communication example