Acceleration Logging

Concept, Implementation, GATT Interface (SS2021)

Fachhochschule
Südwestfalen
University of Applied Sciences

Agenda

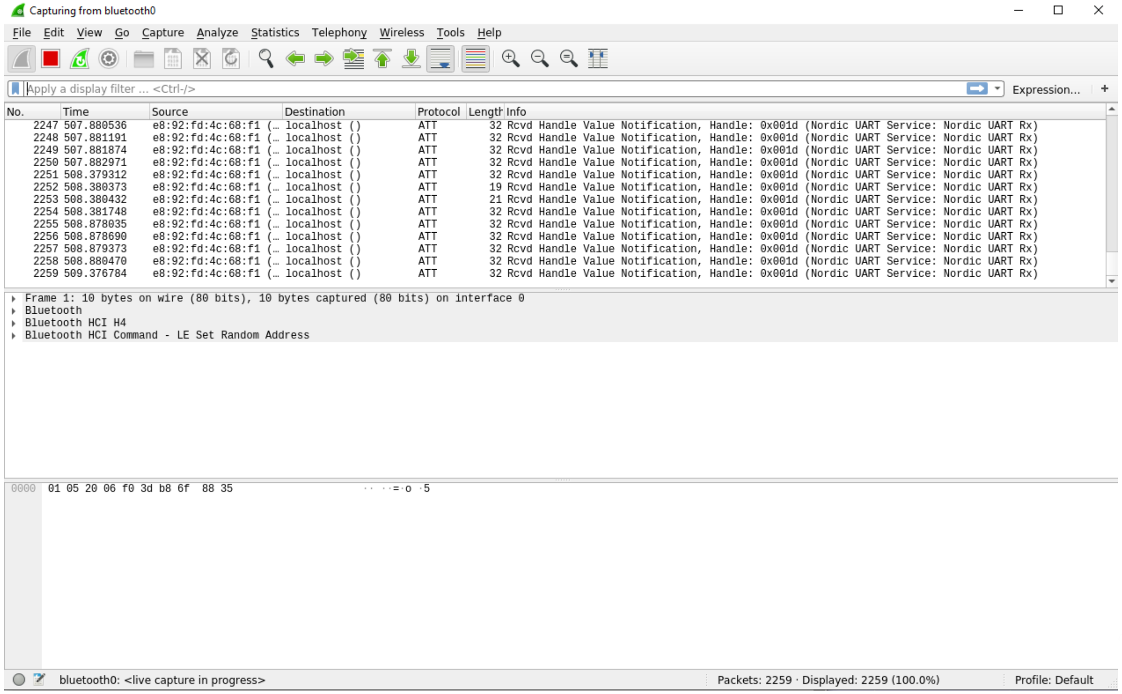
4th Semester

- Bluetooth performance
- Bluetooth device information service
- Issue 230, Firmware crash after short time Bluetooth connection
- New / removed Features
- Implementation
  - 2020: Winter Semester 2020/21
  - 2021: Sommer Semester 2021
- Sample application (separate presentation)

Bluetooth performance

Starting position

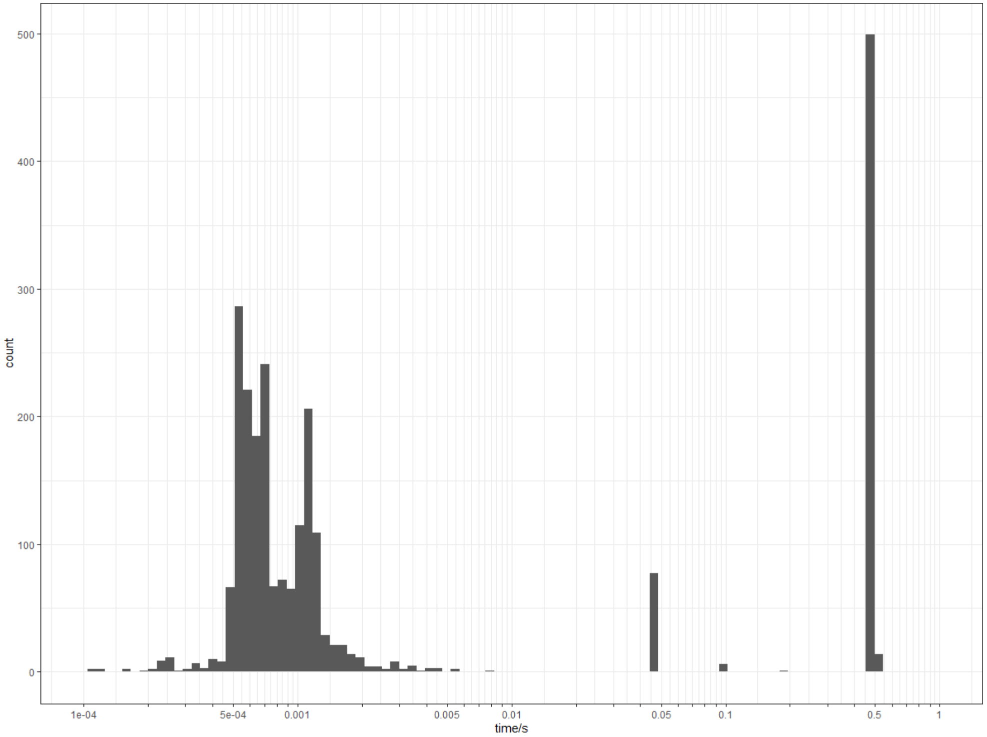
- Very low bandwidth when transferring data via Bluetooth service Nordic NUS.
- Ringbuffer of 48.000 Byte (12 flash pages) size transferring with less than 300 bytes per second takes more than 160 seconds.
- Our goal for this semester is to integrate a flash chip with 8 Mbyte memory. Transferring this would take more than 7 hours.
- Analyze with Wireshark.


Capturing from bluetooth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No.   Time       Source               Destination   Protocol Length Info
 2247 507.880536 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2248 507.881191 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2249 507.881874 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2250 507.882971 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2251 508.379312 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2252 508.380373 e8:92:fd:4c:68:f1 (… localhost ()  ATT      19 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2253 508.380432 e8:92:fd:4c:68:f1 (… localhost ()  ATT      21 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2254 508.381748 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2255 508.878035 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2256 508.878690 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2257 508.879373 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2258 508.880470 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)
 2259 509.376784 e8:92:fd:4c:68:f1 (… localhost ()  ATT      32 Rcvd Handle Value Notification, Handle: 0x001d (Nordic UART Service: Nordic UART Rx)

▸ Frame 1: 10 bytes on wire (80 bits), 10 bytes captured (80 bits) on interface 0
▸ Bluetooth
▸ Bluetooth HCI H4
▸ Bluetooth HCI Command - LE Set Random Address

0000  01 05 20 06 f0 3d b8 6f  88 35                      ·· ··=·o ·5

bluetooth0: <live capture in progress>        Packets: 2259 · Displayed: 2259 (100.0%)        Profile: Default
```

Bluetooth performance

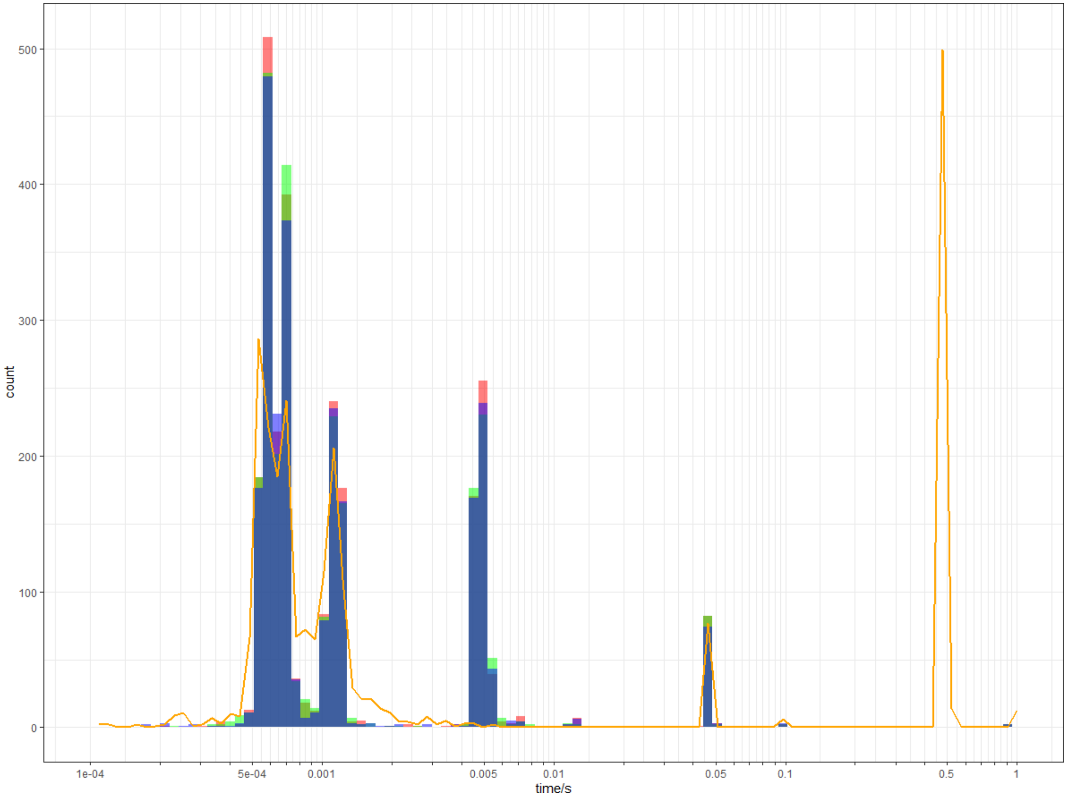
- Three Bluetooth packets are send in a burst.
- Default setup of Nordic Softdevice is to handle three Bluetooth Notifications in parallel.
- After every burst there is a time delay of ~500ms.
- This is configured in ruuvi_nrf5_sdk15_communication_ble_gatt.c by setting gap_conn_params.min_conn_interval and gap_conn_params.max_conn_interval.



Histogram of time between Bluetooth packets.

Bluetooth performance

- Changing gap_conn_params.min_conn_interval and gap_conn_params.max_conn_interval in ruuvi_nrf5_sdk15_communication_ble_gatt.c.
- Test if Bluetooth performance is dependent of the sampling frequency.
- No long delays after changing the parameters.

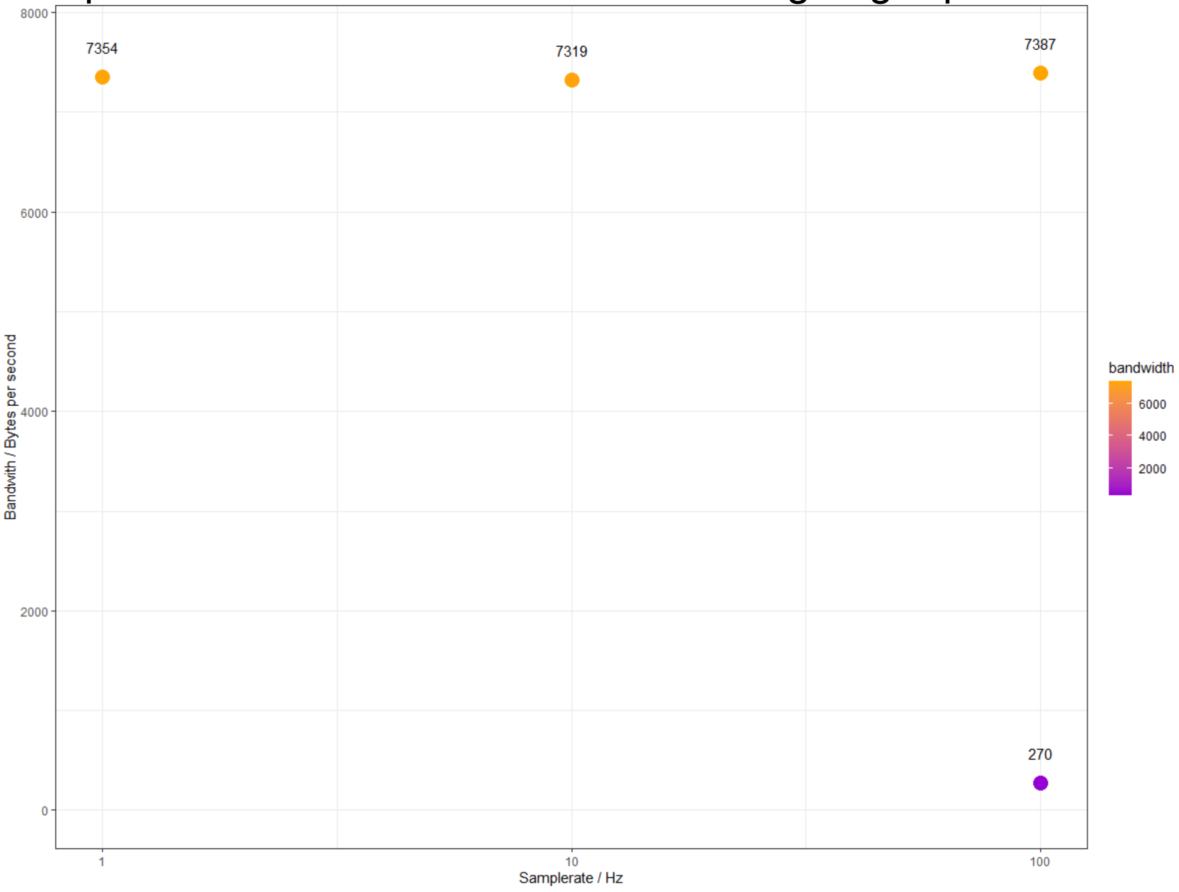


Histogram of time between Bluetooth packets. In contrast to the original setup.

Bluetooth performance

Compare

- Significant improvement of bandwidth 2700%
- No dependency from sampling frequency to bandwidth.
- No further improvement of bandwidth after enabling larger packet size.

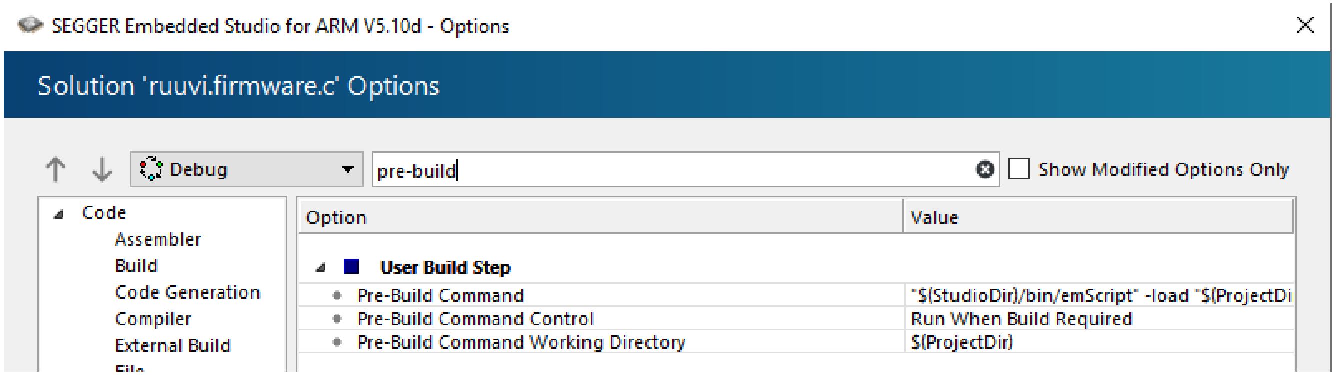


Fachhochschule Südwestfalen
University of Applied Sciences

Bluetooth device information service

Build number

- Add automatic build number to former unused property „Software revision string“ of DIS Service.
- Build number is generated by Pre-Build script of Segger Studio: "$(StudioDir)/bin/emScript" -load "$(ProjectDir)/buildnum.js“
- Script generates buildnum.h which is included by app_comms.c



SEGGER Embedded Studio for ARM V5.10d - Options

Solution 'ruuvi.firmware.c' Options

Debug | pre-build | □ Show Modified Options Only

Option	Value
User Build Step	
Pre-Build Command	"$(StudioDir)/bin/emScript" -load "$(ProjectDi
Pre-Build Command Control	Run When Build Required
Pre-Build Command Working Directory	$(ProjectDir)

Bluetooth device information service

List of available sensors

- Add information about available sensors to „Hardware revision string“ of DIS service to distinguish between different models.
- If function `app_sensor_ctx_get()` from app_sensor.c is available, enumerate Sensors and replace String „Check PCB“ by list of sensors.

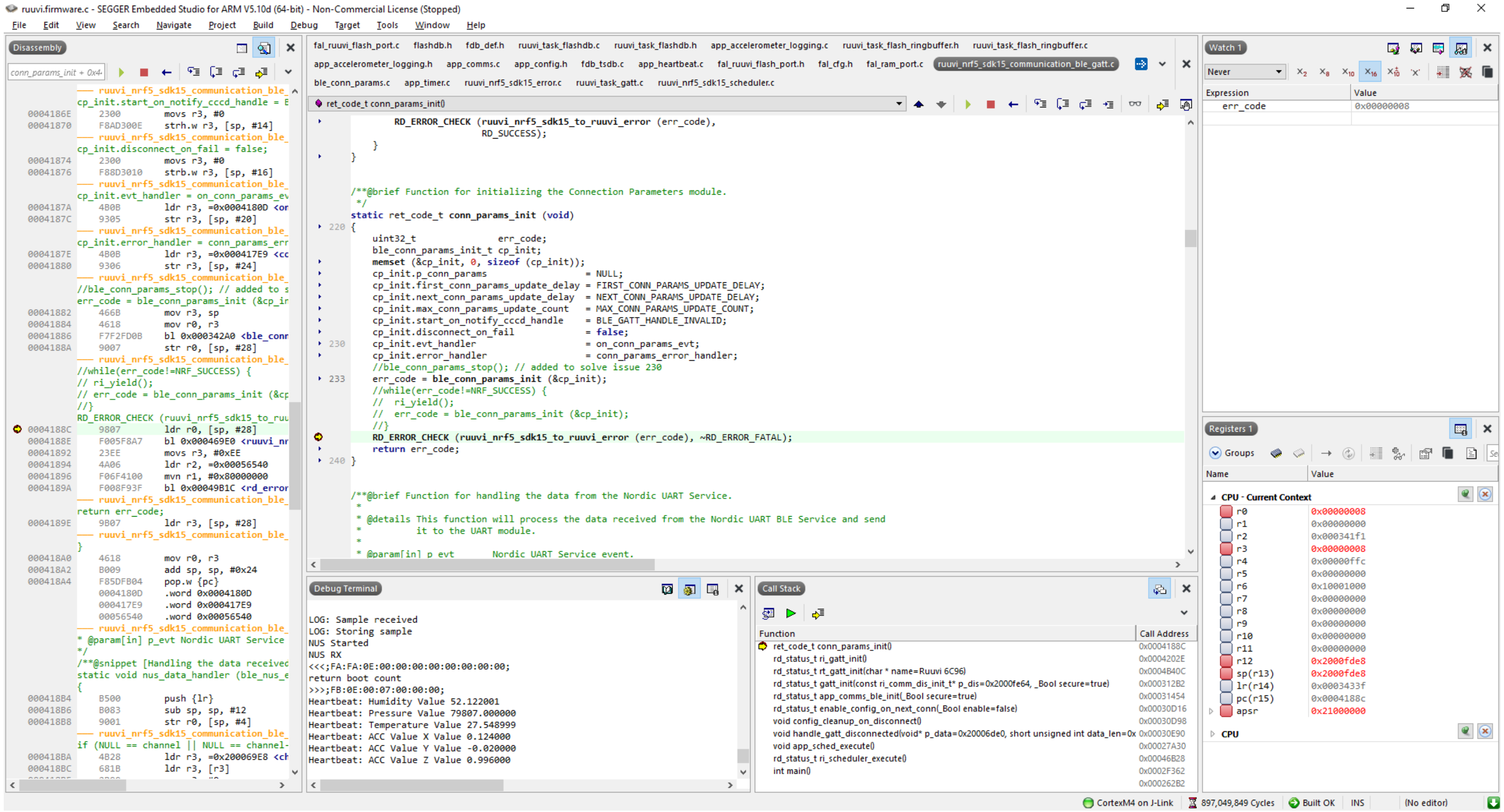
Example of new DIS values

```
[Service] 0000180a-0000-1000-8000-00805f9b34fb: Device Information
          [Characteristic] 00002a28-0000-1000-8000-00805f9b34fb: Software Revision
String: 'Build 20210819_180233'
          [Characteristic] 00002a26-0000-1000-8000-00805f9b34fb: Firmware Revision
String: 'Ruuvi FW v0.0.1+debug'
          [Characteristic] 00002a27-0000-1000-8000-00805f9b34fb: Hardware Revision
String: 'With SHTCX DPS310 LIS2DH12'
          [Characteristic] 00002a24-0000-1000-8000-00805f9b34fb: Model Number String:
'RuuviTag B'
          [Characteristic] 00002a29-0000-1000-8000-00805f9b34fb: Manufacturer Name
String: 'Ruuvi Innovations Ltd'
```

Issue 230

Firmware crashes after short time Bluetooth connection

NRF_ERROR_INVALID_STATE after stop notify on Bluetooth GATT connection.



ruuvi.firmware.c - SEGGER Embedded Studio for ARM V5.10d (64-bit) - Non-Commercial License (Stopped)

File Edit View Search Navigate Project Build Debug Target Tools Window Help
Disassembly
conn_params_init + 0x4
fal_ruuvi_flash_port.c flashdb.h fdb_def.h ruuvi_task_flashdb.c ruuvi_task_flashdb.h app_accelerometer_logging.c ruuvi_task_flash_ringbuffer.h ruuvi_task_flash_ringbuffer.c
app_accelerometer_logging.h app_comms.c app_config.h fdb_tsdb.c app_heartbeat.c fal_ruuvi_flash_port.h fal_cfg.h fal_ram_port.c ruuvi_nrf5_sdk15_communication_ble_gatt.c
ble_conn_params.c app_timer.c ruuvi_nrf5_sdk15_error.c ruuvi_task_gatt.c ruuvi_nrf5_sdk15_scheduler.c
ret_code_t conn_params_init()
Watch 1
Never
Debug Terminal
Call Stack
Registers 1
Groups

```
—— ruuvi_nrf5_sdk15_communication_ble_
cp_init.start_on_notify_cccd_handle = E
0004186E    2300        movs r3, #0
00041870    F8AD300E    strh.w r3, [sp, #14]
—— ruuvi_nrf5_sdk15_communication_ble_
cp_init.disconnect_on_fail = false;
00041874    2300        movs r3, #0
00041876    F88D3010    strb.w r3, [sp, #16]
—— ruuvi_nrf5_sdk15_communication_ble_
cp_init.evt_handler = on_conn_params_ev
0004187A    4B0B        ldr r3, =0x0004180D <on
0004187C    9305        str r3, [sp, #20]
—— ruuvi_nrf5_sdk15_communication_ble_
cp_init.error_handler = conn_params_err
0004187E    4B0B        ldr r3, =0x000417E9 <cc
00041880    9306        str r3, [sp, #24]
—— ruuvi_nrf5_sdk15_communication_ble_
//ble_conn_params_stop(); // added to s
err_code = ble_conn_params_init (&cp_in
00041882    466B        mov r3, sp
00041884    4618        mov r0, r3
00041886    F7F2FD0B    bl 0x000342A0 <ble_conn
0004188A    9007        str r0, [sp, #28]
—— ruuvi_nrf5_sdk15_communication_ble_
//while(err_code!=NRF_SUCCESS) {
// ri_yield();
// err_code = ble_conn_params_init (&cp
//}
RD_ERROR_CHECK (ruuvi_nrf5_sdk15_to_ruu
0004188C    9807        ldr r0, [sp, #28]
0004188E    F005F8A7    bl 0x000469E0 <ruuvi_nr
00041892    23EE        movs r3, #0xEE
00041894    4A06        ldr r2, =0x00056540
00041896    F06F4100    mvn r1, #0x80000000
0004189A    F008F93F    bl 0x00049B1C <rd_error
—— ruuvi_nrf5_sdk15_communication_ble_
return err_code;
0004189E    9B07        ldr r3, [sp, #28]
—— ruuvi_nrf5_sdk15_communication_ble_
}
000418A0    4618        mov r0, r3
000418A2    B009        add sp, sp, #0x24
000418A4    F85DFB04    pop.w {pc}
            0004180D    .word 0x0004180D
            000417E9    .word 0x000417E9
            00056540    .word 0x00056540
—— ruuvi_nrf5_sdk15_communication_ble_
* @param[in] p_evt Nordic UART Service
*/
/**@snippet [Handling the data received
static void nus_data_handler (ble_nus_e
{
000418B4    B500        push {lr}
000418B6    B083        sub sp, sp, #12
000418B8    9001        str r0, [sp, #4]
—— ruuvi_nrf5_sdk15_communication_ble_
if (NULL == channel || NULL == channel-
000418BA    4B28        ldr r3, =0x200069E8 <ch
000418BC    681B        ldr r3, [r3]
```

```c
            RD_ERROR_CHECK (ruuvi_nrf5_sdk15_to_ruuvi_error (err_code),
                            RD_SUCCESS);
        }
}


/**@brief Function for initializing the Connection Parameters module.
 */
static ret_code_t conn_params_init (void)
{
    uint32_t               err_code;
    ble_conn_params_init_t cp_init;
    memset (&cp_init, 0, sizeof (cp_init));
    cp_init.p_conn_params                  = NULL;
    cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay  = NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle    = BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail             = false;
    cp_init.evt_handler                    = on_conn_params_evt;
    cp_init.error_handler                  = conn_params_error_handler;
    //ble_conn_params_stop(); // added to solve issue 230
    err_code = ble_conn_params_init (&cp_init);
    //while(err_code!=NRF_SUCCESS) {
    //  ri_yield();
    //  err_code = ble_conn_params_init (&cp_init);
    //}
    RD_ERROR_CHECK (ruuvi_nrf5_sdk15_to_ruuvi_error (err_code), ~RD_ERROR_FATAL);
    return err_code;
}


/**@brief Function for handling the data from the Nordic UART Service.
 *
 * @details This function will process the data received from the Nordic UART BLE Service and send
 *          it to the UART module.
 *
 * @param[in] p_evt       Nordic UART Service event.
```

Expression	Value
err_code	0x00000008

```
LOG: Sample received
LOG: Storing sample
NUS Started
NUS RX
<<<;FA:FA:0E:00:00:00:00:00:00:00:00;
return boot count
>>>;FB:0E:00:07:00:00:00;
Heartbeat: Humidity Value 52.122001
Heartbeat: Pressure Value 79807.000000
Heartbeat: Temperature Value 27.548999
Heartbeat: ACC Value X Value 0.124000
Heartbeat: ACC Value Y Value -0.020000
Heartbeat: ACC Value Z Value 0.996000
```

Function	Call Address
ret_code_t conn_params_init()	0x0004188C
rd_status_t ri_gatt_init()	0x0004202E
rd_status_t rt_gatt_init(char * name=Ruuvi 6C96)	0x0004B40C
rd_status_t gatt_init(const ri_comm_dis_init_t* p_dis=0x2000fe64, _Bool secure=true)	0x000312B2
rd_status_t app_comms_ble_init(_Bool secure=true)	0x00031454
rd_status_t enable_config_on_next_conn(_Bool enable=false)	0x00030D16
void config_cleanup_on_disconnect()	0x00030D98
void handle_gatt_disconnected(void* p_data=0x20006de0, short unsigned int data_len=0x	0x00030E90
void app_sched_execute()	0x00027A30
rd_status_t ri_scheduler_execute()	0x00046B28
int main()	0x0002F362
	0x000262B2

Name	Value
CPU - Current Context	
r0	0x00000008
r1	0x00000000
r2	0x000341f1
r3	0x00000008
r4	0x00000ffc
r5	0x00000000
r6	0x10001000
r7	0x00000000
r8	0x00000000
r9	0x00000000
r10	0x00000000
r11	0x00000000
r12	0x2000fde8
sp(r13)	0x2000fde8
lr(r14)	0x0003433f
pc(r15)	0x0004188c
apsr	0x21000000
CPU	

CortexM4 on J-Link   897,049,849 Cycles   Built OK   INS   (No editor)

New / removed Features

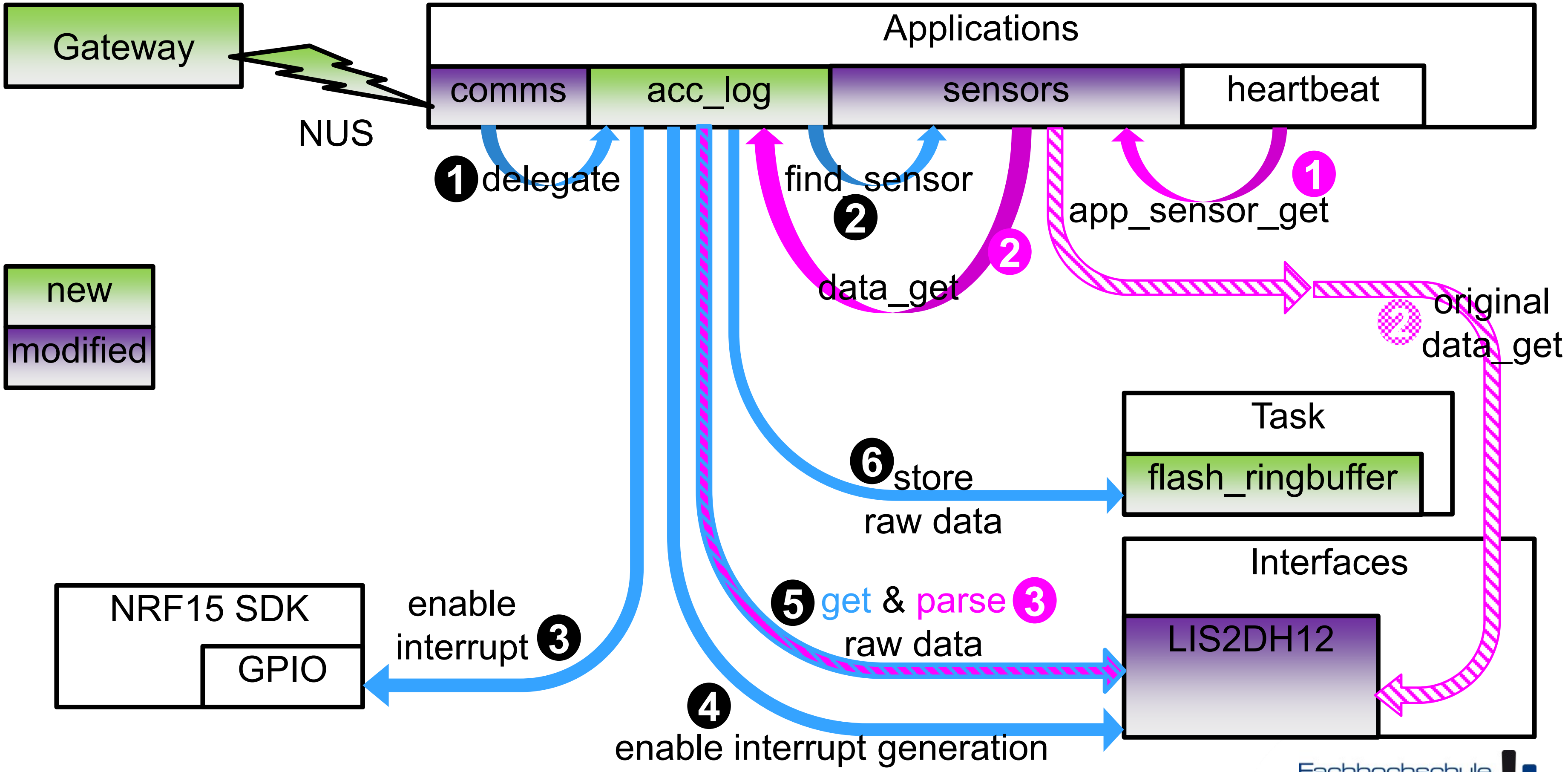
New Features	Removed Features
Connecting Macronix Flash	Download last sample
Replace Ringbuffer by FlashDB	Proprietary GATT messages
FAL devices using Nordic Flash or RAM as backend*	
Frequency divider	
Query Flash statistic	
Query Boot count	
Streaming of acceleration data	

*) See presentation of Jendrik and Jenny for architecture of FlashDB and description of the API needed to implement a FAL device.

Implementation

Wrapping of data_get

● Sequence when activating logging ● Sequence returning data



Gateway

NUS

Applications

comms

acc_log

sensors

heartbeat

1 delegate

find_sensor 2

app_sensor_get 1

data_get 2

original data_get

new

modified

Task

flash_ringbuffer

6 store raw data

Interfaces

LIS2DH12

NRF15 SDK

GPIO

enable interrupt 3

5 get & parse 3 raw data

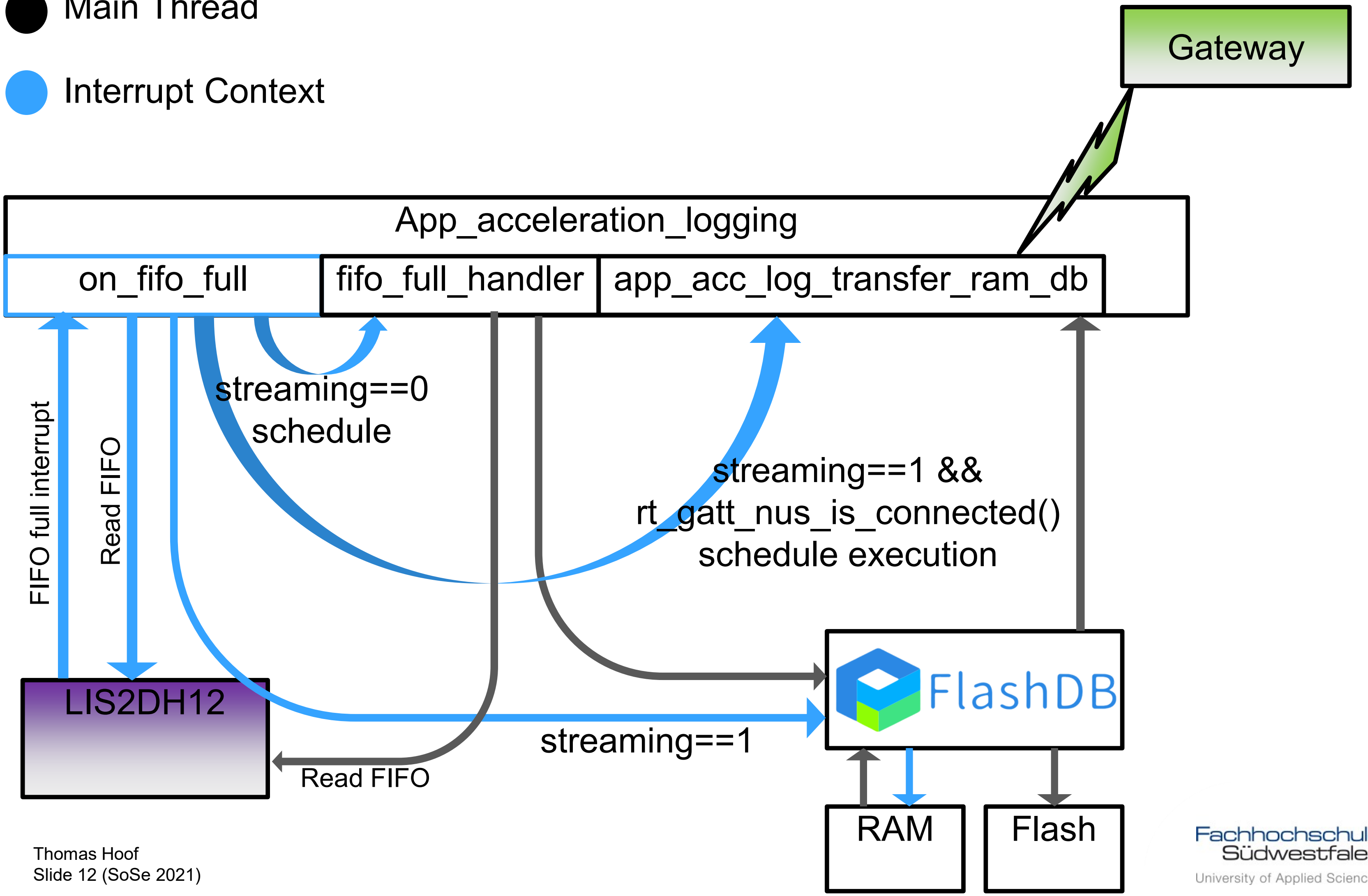
4 enable interrupt generation

Implementation

“Streaming”

● Main Thread

● Interrupt Context



Fachhochschule Südwestfalen
University of Applied Sciences

Files (1)

Name	Status	
app_accelerometer_logging.*	new2020	Main part for acceleration logging.
app_comms.c	modified	GATT message added.
app_config.h	modified	Macro for conditional compiling added. Configuration for memory management of Ringbuffer/FlashDB added.
app_sensor.*	modified	Function for finding sensor context added.
main.c	modified	Initialization of acceleration logging added.
ruuvi_nrf5_sdk15_power.c	modified	Function which return boot count added.

Files (2)

Name	Status	
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.
ruuvi_nrf5_sdk15_communication_ble_gatt.c	modified	Buildnumber added to Bluetooth DIS service.
ruuvi_task_flashdb.*	new2021	Supporting functions needed to integrate FlashDB into Ruuvi Firmware. Also contains functions authored by Jenny and Jendrik.

Files (3)

Name	Status	
ruuvi_task_flash_ringbuffer.*	new2020 rewrite 2021	This module provides a frontend to FlashDB for persisting acceleration data.
ruuvi.firmware.c/src/ruuvi.libraries.c/src/libs/flashdb/	new2021	Source files of FlashDB. Forked from https://github.com/armink/FlashDB

Implementation

app_accelerometer_logging.c

new2020

```
rd_status_t app_enable_sensor_logging(
    const bool use_ram_db,
    const bool format_db)
```

Enables the logging of acceleration data.		
`use_ram_db`	in	Database in RAM is used in case of "streaming" of acceleration data.
`format_db`	in	This parameter is set to true if logging is enabled by gateway to ensure the database is cleared. When re-enabling logging after reboot it is set to false.
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

new2020

`rd_status_t app_disable_sensor_logging(void)`	
Disables the logging of acceleration data by executing the following steps. 1. Disable GPIO interrupt. 2. Disable FIFO on sensor. 3. Disable generating interrupt on sensor. 4. 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

new2020
rewrite2021

```
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()`. If using streaming this function reads the FIFO and writes the values to RAMDB. Without streaming it 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

new2021

```
void pack(const uint8_t resolution,
    const uint16_t sizeData,
    const uint8_t* const data,
    uint8_t* const packeddata)
```

This function stores raw accelerometer values in 8/10/12 Bit format in compact form (without unused bits). It is a frontend to the functions pack8/10/12().

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

Implementation

app_accelerometer_logging.c

new2020

```
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

new2020

`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

new2020

```
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

new2020
rewrite2021

`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 saved into FlashDB. Key-Value-Database.

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

returns	Status code of processing the message.

Implementation

app_accelerometer_logging.c

new2020

`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_accelerometer_logging.c

new2020
rewrite2021

```
rd_status_t app_acc_logging_send_logged_data(
    const ri_comm_xfer_fp_t reply_fp)
```

This function is called from `handle_lis2dh12_comms_v2()` if the gateway requests sending of logged acceleration data. The function triggers FlashDB to read data. Data from the database is read via the callback function `bool callback_send_data_block()`. Inside the callback function the data is send to the requestor.

`reply_fp`	in	Callback to function which actually sends the bytes to the gateway.
returns	Status code of processing the message.	

Implementation

app_accelerometer_logging.c

new2021

```
rd_status_t app_acc_logging_send_eof_v2(
    const ri_comm_xfer_fp_t reply_fp,
    const rd_status_t status_code,
    const uint16_t crc)
```

This function is called from `app_acc_logging_send_logged_data`() after all data is send to the gateway. It reads the configuration of the sensor and sends this to the gateway. This function generates the “end of data” message.

`reply_fp`	in	Callback to function which actually sends the bytes to the gateway.
`status_code`	in	Status code to send to the gateway
`crc`	in	CRC to send to the gateway.
returns	Status code of processing the message.	

Implementation

app_accelerometer_logging.c

new2021

```
void app_acc_log_transfer_ram_db (
    void * p_event_data,
    uint16_t event_size)
```

Execution of this function is scheduled if “streaming” is active and `rt_gatt_nus_is_connected()` returns true.

It starts reading currently logged data from the FlashDB and transferring the data to the gateway.

Scheduling of this function is done inside `on_fifo_full()` using `ri_scheduler_event_put()`.

Implementation

app_accelerometer_logging.c

new2021

`int64_t fdb_timestamp_get (void)`	
Callback for use by FlashDB to retrieve the timestamp of the current new entry.	
returns	Timestamp of actual acceleration sample.

Implementation

app_accelerometer_logging.c

new2021

```
bool callback_send_data_block(fdb_tsl_t tsl,
     void *arg)
```

Callback function for use with FlashDB. When reading the database this function will be called for every entry.

This function calls `app_comms_blocking_send()` to send the data to the gateway.

returns	If true, processing of entries will stop.

Implementation

app_accelerometer_logging.c

new2021

```
rd_status_t app_acc_logging_statistic (
    uint8_t* const statistic)
```

This function is called from `handle_lis2dh12_comms`() after the gateway sends the message to retrieve flash statistics. It calls `rt_flash_ringbuffer_statistic()` to retrieve flash statics.

`statistic`	In/ out	Memory to store the statistic values. See description of “Flash static response” for memory layout.
returns	Status code of processing the message.	

Implementation

app_comms.c

modified2020

```
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

new2020
rewrite2021

```
rd_status_t handle_lis2dh12_comms/
            handle_lis2dh12_comms_v2/
            handle_rtc_comms_v2
    (const ri_comm_xfer_fp_t reply_fp,
    const uint8_t * const raw_message,
    size_t data_len)
```

These three functions handles the GATT/UART communication needed to control the functionality of acceleration logging and RTC. The functions are grouped into proprietary messages regarding LIS2DH12 and standard messages regarding LIS2DH12 or real time clock.

`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.	

Implementation

app_comms.c

modified2021

```
rd_status_t dis_init (
    ri_comm_dis_init_t * const p_dis,
    const bool secure)
```

If `app_sensor_ctx_get()` from app_sensors.c is available replace hardware revision string by list of available sensors.

Implementation

ruuvi_nrf5_sdk15_communication_ble_gatt.c

modified2021

```
rd_status_t ri_gatt_dis_init (
    const ri_comm_dis_init_t * const p_dis)
```

Add buildnumber to former unused property `sw_rev_str` of `ble_dis_init_t`.

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.
- This module also contains macros for memory separation between acceleration logging and environmental logging. The relevant macros are `APP_FLASH_LOG_DATA_RECORDS_NUM` and `RT_FLASH_RINGBUFFER_MAXSIZE`.

Implementation

app_heartbeat.c

modified2020

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

Debug output is added to this function to watch the functionality.

Implementation

app_sensor.c

new2020

```
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

modified2020

`void setup (void)`

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

Implementation

ruuvi_interface_lis2dh12.c

new2020

```
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

modified2020

```
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.

<table>
<tr><td><code>data</code></td><td>in/out</td><td>Structure for storing parsed accelerometer values.</td></tr>
<tr><td>returns</td><td colspan="2">Status code of executing this function.</td></tr>
</table>

Implementation

ruuvi_interface_lis2dh12.c

new2020

```
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

new2020

`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.	

Implementation

ruuvi_nrf5_sdk15_power.c / ruuvi_nrf5_sdk15_power.h

new2021

```
rd_status_t ri_power_read_boot_count
    (uint32_t *boot_count)
```

Return boot count.

`boot_count`	out	Return boot count.
returns	RD_SUCCESS when success, otherwise any error code.	

Implementation

ruuvi_task_flashdb.c

new2021

`rd_status_t rt_flashdb_to_ruuvi_error (fdb_err_t fdb_err)`		
This function converts an error code of FlashDB to an Ruuvi error code.		
`Fdb_err`	in	Error code of FlashDB.
returns	Ruuvi error code which represents the state of FlashDB.	

Implementation

ruuvi_task_flash_ringbuffer.c

new2020
rewrite2021

```
rd_status_t rt_flash_ringbuffer_create(
    const char *partition,
    fdb_get_time get_time,
    const bool format_db)
```

This function initializes an instance of timeseries database. It is called during boot to open an existing database or during activation of acceleration logging to create a new, empty database.

`partition`	in	Name of the partition of the FAL device where the database will be stored.
`get_time`	in	Function pointer to callback function. This function is used by timeseries database to retrieve the current timestamp.
`format_db`	in	Whether to force to create a empty database.
returns	Ruuvi error code	

Implementation

ruuvi_task_flash_ringbuffer.c

new2020
rewrite2021

```
rd_status_t rt_flash_ringbuffer_write(
    const uint16_t size, const void* data)
```

This function writes data to FlashDB.		
`size`	in	Size of data to write.
`data`	in	Pointer to data.
returns	Ruuvi error code	

Implementation

ruuvi_task_flash_ringbuffer.c

new2020
rewrite2021

```
rd_status_t rt_flash_ringbuffer_read(
    const fdb_tsl_cb callback,
    const ri_comm_xfer_fp_t reply_fp,
    uint16_t* crc)
```

This function starts reading the timeseries database. Reading data from timeseries database is done by iterating all entries. For every entry a callback function is called.

`callback`	in	Callback function which would be called for every entry.
`reply_fp`	in	Function pointer to callback function which sends the data to the requestor using BLE.
`crc`	In/out	CRC16 value which gets calculated over all data send to the requestor.
returns	Ruuvi error code	

Implementation

ruuvi_task_flash_ringbuffer.c

new2020
rewrite2021

```
rd_status_t rt_flash_ringbuffer_clear (void)
rd_status_t rt_flash_ringbuffer_drop (void)
```

The function rt_flash_ringbuffer_clear clears the content of the ringbuffer.

The function rt_flash_ringbuffer_drop deinitializes the timeseries database. It does not free nor erase the flash memory used by the database.

returns	Ruuvi error code

Implementation

ruuvi_task_flash_ringbuffer.c

new2021

```
rd_status_t rt_flash_ringbuffer_statistic (
    uint8_t* const statistic)
```

This function reads some statistics about the usage of the internal Nordic Flash memory and returns them.		
`statistic`	In/out	Memory for storing statistics.
returns	Ruuvi error code	

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

Standard messages

Control message 0x4A 0x4A + Type		Response message 0x4A 0x4A + Type				Data Message 0x11
Type	Message	Status	Time	Config	End of data	
0x11	Logged Data	Error			Succes	Succes
0x02	Set Config	Always				
0x03	Get Config			Always		
0x08	Set Logging	Always				
0x08	Get Logging	Always				
Control message 0x21 0x21 + Type		**Response message 0x21 0x21 + Type**				
0x08	Set Time	Always				
0x09	Get Time		Always			

GATT Interface

Proprietary messages

Control message 0xFA 0xFA + Type		Response message 0xFB + Type				
Type	Message	Status (0x00)	Statistic (0x0D)	Boot Count (0x0E)		
0x0D	Flash statistic	Error	Success			
0x0E	Boot count	Always		Success		

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 `0x4a 0x4A Type SS`.

The `Type` byte reflects the same value as transmitted to the sensor in the command message.

The byte `SS` contains the information about the status. The value is equal to the bit position of the error code plus one. See file ruuvi_driver_error.h for an explanation of the bits from the status value.

As an example: RD_ERROR_NOT_INITIALIZED is defined as 2^19. If this condition would be returned as error state using a status response the value 20 = 19 + 1 would be returned.

GATT Interface

Transmit logged data

This message starts transmitting the logged acceleration data from the ringbuffer. 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 one parameters. After removing the possibility to download the last sample this parameter must have the value `0x01`.

The concrete content of this message is: `0x4A 0x4A 0x11 0x01`.

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. 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
`0x4A 0x4A 0x11 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: `0x4A 0x4A 0x02 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	Frequency divider. Divide the sample frequency by this value.
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: `0x4A 0x4A 0x03`.

The concrete content of the message which returns the configuration is

`0x4A 0x4A 0x03 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

```
0x21 0x21 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 `0x21 0x21 0x09`.

The concrete content of a timestamp response is the following `0x21 0x21 0x09 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 `0x4A 0x4A 0x08 XX`. The parameter XX may have one of the following values.

Parameter	Description
0x00	No logging
0x01	Logging of acceleration data to flash.
0x02	Logging of acceleration data to RAM. Used for logging of high frequency sampling. If someone is connected to NUS, the data from RAM is immediately transferred.

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 `0x4A 0x4A 0x09`.

GATT Interface (proprietary)

Query flash statistic

This message is used to query flash usage. The concrete content of this message is `0xFA 0xFA 0x0D`. The sensor responds several values which were retrieved from the Nordic softdevice by calling `fds_stat()`.

The following table shows the response Message to this command.

Byte	+0	+1
0	`0xFB`	`0x0D`
2	`SS`	Logging status
4	Unused (`0xFF`)	Unused (`0xFF`)
6	Unused (`0xFF`)	Valid records (L)
8	Valid records (H)	Dirty records (L)
10	Dirty records (H)	Words reserved (L)
12	Words reserved (H)	Words used (L)
14	Words used (H)	Largest contig (L)
16	Largest contig (H)	Freeable words (L)
18	Freeable words (H)	

GATT Interface (proprietary)

Flash statistic response

Value	Description
SS	Status of executing this command. If this indicates an error, the following values are unpredicted.
Logging status	Background error code of logging thread.
Unused bytes	Reserved for retrieving DB usage.
Valid records	The number of valid records.
Dirty records	The number of deleted ("dirty") records.
Words reserved	The number of words reserved.
Words used	The number of words written to flash, including those reserved for future writes.
Largest contig	The largest number of free contiguous words in the file system.
Freeable words	The largest number of words that can be reclaimed by garbage collection.

GATT Interface (proprietary)

Query boot count

This message is used to query the boot count. The purpose is to check the quality of our implementation.

The concrete content of this message is `0xFA 0xFA 0x0E`.

The Sensor responds with returning the boot count. The value is stored inside flash memory.

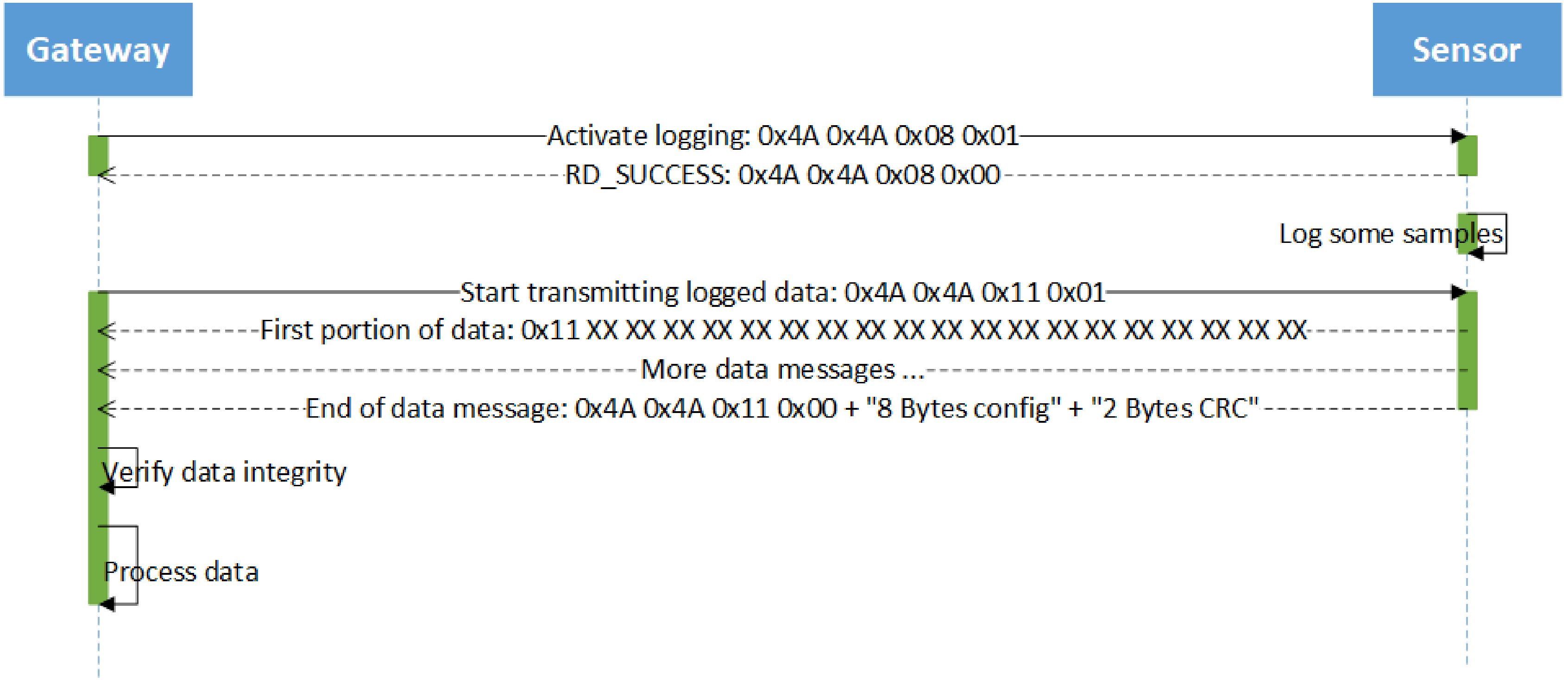
The concrete content of the response message is:
`0xFB 0x0E SS BC1 BC2 BC3 BC4`.

Where `SS` is the status code of executing the command. If there is an error, the content of the following bytes is unpredictable.

`BC1 BC2 BC3 BC4` are the bytes from the boot counter. This value is of type 32 Bit unsigned integer. The bytes are transferred in little-endian byte sequence.

GATT Interface

Communication example



Gateway

Sensor

Activate logging: 0x4A 0x4A 0x08 0x01

RD_SUCCESS: 0x4A 0x4A 0x08 0x00

Log some samples

Start transmitting logged data: 0x4A 0x4A 0x11 0x01

First portion of data: 0x11 XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX

More data messages ...

End of data message: 0x4A 0x4A 0x11 0x00 + "8 Bytes config" + "2 Bytes CRC"

Verify data integrity

Process data

Fachhochschule Südwestfalen
University of Applied Sciences