TEKTELIC Communications Inc.
7657 10th Street NE Calgary, Alberta
Canada, T2E 8X2

Armband (Bronze) Sensor

Technical Reference Manual

DOCUMENT TYPE:	Technical Reference Manual
DOCUMENT NUMBER:	T0006922_TRM_Armband_Bronze
DOCUMENT VERSION:	0.1
PRODUCT NAME:	Armband (Bronze) Sensor
PRODUCT CODE:	T0007405
RELEASE DATE:	August 10, 2021

Revision History

Version	Date	Editor	Comments
0.1	Aug 10, 2021	Reza Nikjah	Initial draft.

Table of Contents

List of Tables

List of Figures

Acronyms and Glossary

ABP ..... activation by personalization
ADC ..... analog-to-digital converter
ADR ..... adaptive data rate
CRC ..... cyclic redundancy check
DL ..... downlink
DR ..... data rate
EIRP ..... effective isotropic radiated power
Flash memory ..... non-volatile memory on the Armband Sensor, which contains application and configuration settings
g ..... gravity (unit of acceleration ≈ 9.8 m/s^2)
ID ..... identity
IoT ..... Internet of Things
LoRa ..... a patented "long-range" IoT technology acquired by Semtech
LoRAMAC ..... LoRaWAN MAC
LoRaWAN ..... LoRa wide area network (a network protocol based on LoRa)
LoRaWAN Commissioning ..... the unique device identifiers and encryption keys used for LoRaWAN communication (see LoRaWAN Specification [1] for more details)
LSB ..... least significant bit
MAC ..... medium access control
MCU ..... microcontroller unit
min ..... minute(s)
MSB ..... most significant bit
NS ..... network server
OTA ..... over-the-air
OTAA ..... OTA activation
RH ..... relative humidity
RO ..... read-only
R/W ..... read/write
Rx ..... receiver
sec ..... second(s)
Sensor ..... Armband (Bronze) Sensor
transducer ..... the sensing element on the Armband Sensor (e.g. temperature transducer)
TRM ..... technical reference manual (this document)
Tx ..... transmitter
UL ..... uplink
WO ..... write-only

1 Overview

This TRM describes the user accessible configuration settings (pseudo registers) supported by the Lora IoT Armband (Bronze) Sensor, referred to as the Armband Sensor or the Sensor henceforth. This document is intended for a technical audience, such as application developers, with an understanding of the NS and its command interfaces.

The Armband Sensor is a LoRaWAN IoT sensor packed into a very small form factor. The Armband Sensor is ideal for monitoring and reporting skin temperature and activity factor. Table 1-1 presents the different components of the Armband Sensor. Also, Table 1-2 lists the Armband Sensor variants for regions identified by the LoRa Alliance [2]—see [2] for the Tx and Rx bands in each LoRaWAN region.

Table 1-1: Armband (Bronze) Sensor Components

Part Number			Description
Level 1	Level 2	Level 3	
T0007405			Armband (Bronze) Sensor Module, LoRa IoT
	T0007119		Armband (Bronze) Sensor PCBA, LoRa IoT
		T0007120	Armband (Bronze) Sensor PCB, LoRa IoT

Table 1-2: Armband (Bronze) Sensor Region Specific Variants

LoRaWAN RF Variant	Order Code
EU868	MEDBBEU868
US915	MEDBBUS915
AS923	MEDBBAS923
AU915	MEDBBAU915
IN865	MEDBBIN865
KR920	MEDBBKR920
RU864	MEDBBRU864

Regarding communication direction (UL or DL) and LoRaWAN ports, all information streams currently supported by the SW are as follows:

- Readings obtained from on-board transducers (***sent in UL, LoRaWAN port 10***)
- Configuration and control commands from the NS used to change the Sensor's behavior or inquire the Sensor for the values of registers (***sent in DL, LoRaWAN port 100***)
- Response to configuration and control commands from the NS (***sent in UL, LoRaWAN port 100***)

The default configuration of the Sensor for reporting transducer readings includes the following:

- Report the battery voltage every minute.
- Report the on-board temperature every minute.

- Report the thermistor temperature every minute.
    - The on-board and thermistor temperatures are combined in the UL data converter (see Section 4) to yield the body temperature every minute.
- Report the ambient RH every minute.
- Report actuation (an open-to-close or close-to-open event) of the reed switch every 1 (one) actuation

2 UL Payload Formats

The UL streams (from the Sensor to the NS) include the following:

- The readings obtained from on-board transducers (***sent on LoRaWAN port 10***)
- Response to configuration and control commands from the NS (***sent on LoRaWAN port 100***)

These topics are explained in Sections 2.1 and 2.2, respectively.

2.1 Frame Payload to Report Transducers Data

Each data field from the Sensor is encoded in a frame format shown in Figure 2-1. A big-endian format (MSB first) is always followed.

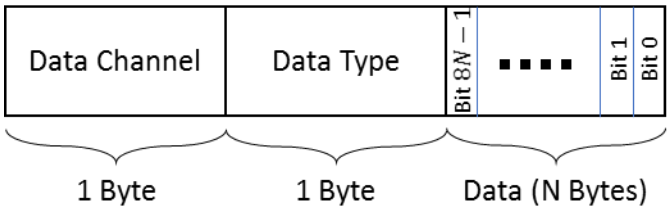


Figure 2-1: The UL frame payload format.

A Sensor message payload can include multiple transducer data frames. The ordering of frames is not guaranteed (they can be in any order). A single payload may include data from any given transducer. The Armband Sensor payload frame values are shown in Table 2-1. In this table, the bit indexing scheme is as shown in Figure 2-1. Payload frame values in Table 2-1 has been grouped by bolded boundaries. This grouping is only to indicate which payloads are related to the same physical transducer. The grouping *does not imply* that the payloads within the same group are uplinked together.

Transducer data in the UL are sent through ***LoRaWAN port 10***.

Table 2-1: UL Frame Payload Values for Transducer Data

Information Type	Channel ID	Type ID	Size	Data Type	Data Format	JSON Variable (Type/Unit)
Battery Voltage	0x00	0xFF	2 B	Analog	• 10 mV / LSB (signed)	*battery_voltage: <value> (unsigned/volt)*
Reed Switch State	0x01	0x00	1 B	Digital	• 0x00 = Low—magnet present • 0xFF = High—magnet absent	*reed_switch_state: <value> (unsigned/no unit)*
Reed Switch Count	0x08	0x04	2 B	Counter	• Number	*reed_switch_count: <value> (unsigned/no unit)*

Impact Alarm	0x0C	0x00	1 B	Digital	• 0x00 = Impact alarm inactive • 0xFF = Impact alarm active	*impact_alarm: <value> (unsigned/no unit)*
Acceleration Magnitude	0x05	0x02	2 B	Analog	• 1 milli-*g*/LSB (unsigned)	*acceleration_magnitude: <value> (unsigned/g)*
Acceleration Vector	0x07	0x71	6 B	Acceleration	• 1 milli-*g*/LSB (signed) • Bits 32-47: X-axis acceleration • Bits 16-31: Y-axis acceleration • Bits 0-15: Z-axis acceleration	*acceleration_x: <value> (signed/g)* *acceleration_y: <value> (signed/g)* *acceleration_z: <value> (signed/g)*
Thermistor Temperature	0x11	0x02	2 B	Analog	• 1 mV/LSB (unsigned)[1]	*temp_therm: <value> (unsigned/°C)*
MCU Temperature	0x0B	0x67	2 B	Temperature	• 0.1°C / LSB (signed)	*temp_mcu: <value> (signed/°C)*
On-board Temperature	0x03	0x67	2 B	Temperature	• 0.1°C / LSB (signed)	*temp_shtc3: <value> (signed/°C)*
Ambient RH	0x04	0x68	1 B	RH	• 0.5% / LSB	*relative_humidity: <value> (unsigned/%)*
Body Temperature	—	—	—	Temperature	• Derived in the UL data converter (see Section 4) from the on-board and thermistor temperatures	*temp_body: <value> (unsigned/°C)*

2.2 Response to Configuration and Control Commands

Sensor responses to DL configuration and control commands (which are sent on ***LoRaWAN port 100***; see Section 3) are sent in the UL on ***LoRaWAN port 100***. These responses include the following:

- Returning the value of a configuration register in response to an inquiry from the NS.
- Writing to a configuration register.

In the former case, the Sensor responds by the address and value of each of the registers under inquiry (this can be in one or more consecutive UL packets depending on the maximum frame payload size allowed). In the latter case, the Sensor responds with a CRC32 of the entire DL payload (which may be a combination of read

[1] Voltage value which is converted to °C in the UL data converter (see Section 4).

and write commands) as the first 4 bytes of the UL frame. If the DL payload has also had read commands, the 4 CRC32 bytes are followed by the address and value of each of the registers under inquiry (similar to the Sensor response in the former case).

3 DL Payload Formats

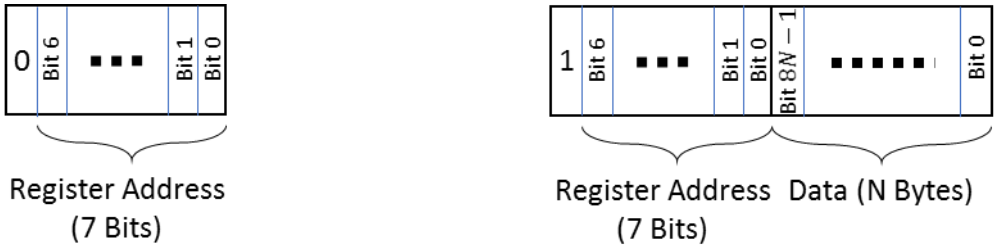
The only DL message (from the NS to the Sensor) supported by the SW includes the following:

- Configuration and control commands used to change the Sensor's behavior or inquire the Sensor for the values of registers (***sent on LoRaWAN port 100***).

A single DL configuration and control message can contain multiple command blocks, with a possible mix of read and write commands. Each message block is formatted as shown in Figure 3-1. A big-endian format (MSB first) is always followed.

The Register Address is used to access various configuration parameters. These addresses are bound between 0x00 and 0x7F.

Bit 7 of the first byte determines whether a read or write action is being performed, as shown in Figure 3-1. All read commands are one-byte long. Data following a read access command will be interpreted as a new command block. Read commands are processed last. For example, in a single DL message, if there is a read command from a register and a write command to the same register, the write command is executed first.



(a) The read command. (b) The write command.

Figure 3-1: The format of a DL configuration and control message block.

All DL configuration and control commands are sent on ***LoRaWAN port 100***.

When a write command is sent to the Sensor, the Sensor immediately responds with a CRC32 of the entire DL payload as the first 4 bytes of the UL frame on ***LoRaWAN port 100*** (also see Section 2.2).

DL configuration and control commands fall into one of the following four categories and are discussed in Sections 3.1, 3.2, and 3.3, respectively:

- LoRaMAC Configuration
- Application Configuration
- Command and Control

3.1 LoRaMAC Configuration

LoRaMAC options can be configured using DL commands. These configuration options change the default MAC configuration that the Sensor loads on start-up. They can also change certain run-time parameters. Table 3-1

shows the MAC configuration registers. All the registers have R/W access. In this table, the bit indexing scheme is as shown in Figure 3-1.

Table 3-1: LoRaMAC Configuration Registers

Address	Value	Size	Description	JSON Variable (Type/Unit)
0x10	Join Mode	2 B	• Bit 15: 0/1 = ABP/OTAA mode • Bits 0-14: Ignored	*loramac_join_mode: <value> (unsigned/no unit)*
0x11	Options	2 B	• Bit 0: 0/1 = Unconfirmed/Confirmed UL • Bit 1 = 1 (RO): 0/1 = Private/Public Sync Word • Bit 2: 0/1 = Disable/Enable Duty Cycle • Bit 3: 0/1 = Disable/Enable ADR • Bits 4-15: Ignored	*loramac_opts {* *confirm_mode: <value>, (unsigned/no unit)* *sync_word: <value>, (unsigned/no unit)* *duty_cycle: <value>, (unsigned/no unit)* *adr: <value> (unsigned/no unit)* *}*
0x12	DR and Tx Power[2]	2 B	• Bits 8-11: Default DR number • Bits 0-3: Default Tx power number • Bits 4-7, 12-15: Ignored	*loramac_dr_tx {* *dr_number: <value>, (unsigned/no unit)* *tx_power_number: <value>, (unsigned/no unit)* *}*
0x13	Rx2 Window	5 B	• Bits 8-39: Channel frequency in Hz for Rx2 • Bits 0-7: DR for Rx2	*loramac_rx2 {* *frequency: <value>, (unsigned/Hz)* *dr_number: <value> (unsigned/no unit)* *}*

Note: Modifying these values only changes them in the Sensor. Options for the Sensor in the NS also need to be changed in order to not strand a Sensor. Modifying configuration parameters in the NS is outside the scope of this document.

Examples:

- Switch Sensor to ABP Mode:

[2] Tx power number *m* translates to the maximum Tx power, which is a function of the LoRaWAN RF region, minus $2 \times m$ dB.

- DL payload: { 0x **90** 00 00 }
- Set ADR enabled, no duty cycle, and confirmed UL payloads:
  - DL payload: { 0x **91** 00 0B }
- Set default DR number to 1 and default Tx power number to 2:
  - DL payload: { 0x **92** 01 02 }

3.1.1 Default Configuration

Table 3-2 and Table 3-4 show the default values for the LoRaMAC configuration registers (cf. [1]).

Table 3-2: Default Values of LoRaMAC Configuration Registers

Address	Default Value
0x10	• OTAA mode
0x11	• Unconfirmed UL • Duty cycle enabled[3] • ADR enabled
0x12	• DR0 • Tx Power 0 (max power; see Table 3-3)
0x13	• As per Table 3-4

Table 3-3: Default Maximum Tx Power in Different Regions

RF Region	Max Tx EIRP [dBm]
EU868	16
US915	30
AS923	16
AU915	30
IN865	30
KR920	14
RU864	16

Table 3-4: Default Values of Rx2 Channel Frequency and DR Number in Different Regions

RF Region	Channel Frequency [Hz]	DR Number
EU868	869525000	0
US915	923300000	8
AS923	923200000	2
AU915	923300000	8
IN865	866550000	2

[3] In the LoRa RF regions where there is no duty cycle limitation, such as US915, the "enabled duty cycle" configuration of the Sensor is ignored.

KR920	921900000	0
RU864	869100000	0

3.2 Application Configuration

This section lists all possible application configurations (as part of DL configuration and control commands), including periodic Tx configuration and configurations of the different transducers.

3.2.1 Periodic Tx Configuration

All periodic transducer reporting is synchronized around *ticks*. A *tick* is simply a user configurable time-base that is used to schedule transducer measurements. For each transducer, the number of elapsed *ticks* before transmitting can be defined as shown in Table 3-5. All the registers in this table have R/W access.

Note: Certain transducer types, such as accelerometer and light, need to be enabled for periodic reporting. Details are available in each transducer's respective section.

Table 3-5: Periodic Transmission Configuration Registers

Address	Value	Size	Description	JSON Variable (Type/Unit)
0x20	Seconds per Core *Tick*	4 B	• *Tick* value for periodic events • Acceptable values: 0, 60, 61, …, 86400 • 0 disables all periodic transmissions • Other values: Invalid and ignored	*seconds_per_core_tick: <value> (unsigned/sec)*
0x21	*Ticks* per Battery	2 B	• *Ticks* between battery reports • 0 disables periodic battery reports	*ticks_per_battery: <value> (unsigned/no unit)*
0x22	*Ticks* per On-board Temperature	2 B	• *Ticks* between on-board temperature reports • 0 disables periodic on-board temperature reports	*ticks_per_onboard_temperature: <value> (unsigned/no unit)*
0x23	*Ticks* per Ambient RH	2 B	• *Ticks* between ambient RH reports • 0 disabled periodic ambient RH reports	*ticks_per_relative_humidity: <value> (unsigned/no unit)*
0x24	*Ticks* per Reed Switch	2 B	• *Ticks* between reed switch reports • 0 disables periodic reed switch reports	*ticks_per_reed_switch: <value> (unsigned/no unit)*
0x26	*Ticks* per Accelerometer	2 B	• *Ticks* between accelerometer reports • 0 disables periodic accelerometer reports	*ticks_per_accelerometer: <value> (unsigned/no unit)*
0x27	*Ticks* per MCU Temperature	2 B	• *Ticks* between MCU temperature reports • 0 disables periodic MCU temperature reports	*ticks_per_mcu_temperature: <value> (unsigned/no unit)*

0x29	*Ticks* per Thermistor Temperature	2 B	• *Ticks* between thermistor reports • A value of 0 disables periodic thermistor reports	*ticks_per_thermistor: <value> (unsigned/no unit)*
—	*Ticks* per Body Temperature	—	• Least common multiple (lcm) of *Ticks* per On-board Temperature and *Ticks* per Thermistor Temperature[4]	—

3.2.1.1 Seconds per Core Tick

All periodic Tx events are scheduled in *ticks*. This allows for transducer reads to be synchronized, reducing the total number of ULs required to transmit Sensor data. The minimum seconds per *tick* is 60 sec, and the maximum is 86,400 sec (one day). Values from 1 sec to 59 sec and values above 86,400 sec are invalid and ignored. A value of 0 (zero) disables all periodic reporting.

3.2.1.2 Ticks per <Transducer>

This register sets the reporting period for a transducer in terms of *ticks*. Once the configured number of *ticks* has expired, the Sensor polls the specified transducer and reports the data in an UL message. A setting of 0 (zero) disables periodic reporting for the specified transducer.

3.2.1.3 Default Configuration

Table 3-6 shows the default values for the periodic transmission configuration registers.

Table 3-6: Default Values of Periodic Transmission Configuration Registers

Seconds per Core *tick*	60 (1 min)[5]
***Ticks* per Battery**	1 (thus 1-min period)
***Ticks* per On-board Temperature**	1 (thus 1-min period)
***Ticks* per Ambient RH**	1 (thus 1-min period)
***Ticks* per Reed Switch**	0 (periodic Tx disabled)
***Ticks* per Accelerometer**	0 (periodic Tx disabled)
***Ticks* per MCU Temperature**	0 (periodic Tx disabled)
***Ticks* per Thermistor Temperature**	1 (thus 1-min period)[6]

[4] The body temperature is calculated as a function of the on-board temperature and thermistor temperature. Therefore, it is obtained from the UL data converter (see Section 4) whenever both on-board and thermistor temperatures exist. For example, if *Ticks* per On-board Temperature = 1 and *Ticks* per Thermistor Temperature = 1, then *Ticks* per Body Temperature = 1. But if *Ticks* per On-board Temperature = 2 and *Ticks* per Thermistor Temperature = 3, then *Ticks* per Body Temperature = lcm(2, 3) = 6.

[5] SW default value is 3600 (1 hour). Default values are restored using register 0x72 (see Section 3.3).

[6] SW default value is 0. Default values are restored using register 0x72 (see Section 3.3).

Examples:

- Disable all periodic events:
  - DL payload: { 0x **A0** 00 00 00 00 }
    - Register 0x20 with the write bit set to true
    - Seconds per *Tick* set to 0 (zero)—i.e. disable periodic transmissions
- Read current value of Seconds per *Tick*:
  - DL payload: { 0x **20** }
    - Register 0x20 with the write bit set to false
- Report Temperature every *tick* and RH every two *ticks*:
  - DL payload: { 0x **A2** 00 01 **A3** 00 02 }
    - Registers 0x22 and 0x23 with their write bits set to true
    - Temperature *Ticks* set to 1 (one)
    - RH *Ticks* set to 2 (two)

3.2.1.4 Anti-Bricking Strategy

Care has been taken to avoid stranding (hard or soft bricking) the Sensor during reconfiguration. Hard bricking refers to the condition that the Sensor does not transmit anymore as all periodic and event-based reporting (see subsequent sections) have been disabled and the configuration has been saved to the Flash memory. Soft bricking refers to the condition where the Sensor has been configured such that all event-based reporting is disabled and any periodic reporting is either disabled or has a period of larger than a week.

To avoid these situations, for any reconfiguration command sent to the Sensor, the following algorithm is executed:

After the reconfiguration is applied, if all event-based reporting (as explained in subsequent sections) is disabled, then periodic reporting is checked. If all periodic reporting is disabled or the minimum non-zero period is greater than a week, then to avoid bricking the Sensor, the core *tick* is set to 86,400 (i.e. one day), and the battery *tick* is set to 1 (one).

3.2.2 Reed Switch Configuration

Table 3-7 shows a list of Reed Switch configuration registers. All registers have R/W access. In this table, the bit indexing scheme is as shown in Figure 3-1.

Table 3-7: Reed Switch Configuration Registers

Address	Name	Size	Description	JSON Variable (Type/Unit)
0x2A	Mode	1 B	• Bit 0: 0/1 = Rising edge disabled/enabled • Bit 1: 0/1 = Falling edge disabled/enabled	*reed_switch_mode {* *rising_edge_enabled: <value>, (unsigned/no unit)* *falling_edge_enabled: <value>*

			• Both bits 0 and 1 set to 0: Invalid and ignored Bits 2-7: Ignored	*(unsigned/no unit)* *}*
0x2B	Count Threshold	2 B	• Number of triggers for event transmission • 0 disables event transmission	*reed_switch_count_threshold: <value>* *(unsigned/no unit)*
0x2C	Report Options	1 B	• Bit 0: 0/1 = Input state not reported/reported • Bit 1: 0/1 = Counter value not reported/reported • Both bits 0 and 1 set to 0: Invalid and ignored • Bits 2-7: Ignored	*reed_switch_report_options {* *report_state_enabled: <value>, (unsigned/no unit)* *report_count_enabled: <value>, (unsigned/no unit)* *}*

3.2.2.1 Mode

The Reed Switch is edge-triggered and can be set to trigger to rising-edge trigger (Low or Closed to High or Open), falling-edge triggered (High or Open to Closed or Low) or both. An attempt to set the Mode to 0x00 (i.e., to disable both rising and falling edges) is ignored by the Sensor.

3.2.2.2 Count Threshold

The Count Threshold determines when the Sensor transmits after seeing an event on the Reed Switch. A value of 0 (zero) disables the event driven transmission, while a value of 1 (one) or greater triggers an event-based transmission after the configured number of events has occurred, which is when the event "counter" reaches the value of the Count Threshold. Whenever such event-based transmission occurs, the event counter is automatically reset to 0 and starts incrementing as events occur until the counter reaches the threshold again and another event-based transmission occurs.

3.2.2.3 Report Options

The Report Options determines what information is transmitted whenever an event or periodic digital transmission is required. If the value is "Counter Value", the transmission contains the number of times the Reed Switch was triggered since the last transmission, while the value of "Input State" causes a transmission of the current input state of the switch (i.e., Open or Closed).

3.2.2.4 Default Configuration

Table 3-8 shows the default values for the Reed Switch configuration registers.

Table 3-8: Default Values of Reed Switch Configuration Registers

Mode	Rising and falling edges enabled

Threshold	1 (one)
Report Options	State and count reported

Examples:

- Have Reed Switch be triggered only on rising edges:
  - DL payload: { 0x **AA** 01 }
    - Register 0x2A with write bit set to true
    - "Rising Edge" enabled, "Falling Edge" disabled
- Read current value of Count Threshold:
  - DL payload: { 0x **2B** }
    - Register 0x2B with write bit set to false
- Transmit the Reed Switch "state" as soon as the Reed Switch is tripped 10 times:
  - DL payload: { 0x **AB** 00 0A **AC** 01 }
    - Registers 0x2B and 0x2C with their write bits set to true
    - Count Threshold set to 10
    - Report Options set to "Input State"
- Disable the Reed Switch event-driven transmission, but report the number of times the Reed Switch has been triggered whenever a report is inquired (i.e., in the case of periodic reporting):
  - DL payload: { 0x **AB** 00 00 **AC** 02 }
    - Count Threshold set to 0 (zero)
    - Report Options set to "Counter Value"

3.2.3 Input Configuration

The Armband Sensor has an input connector, which is permanently connected to the skin temperature thermistor internally. In general, this connector can be configured as a digital or analog input. However, for correct operation of the Armband Sensor, the connector has to be configured as an *analog input*.

Table 3-9 shows the Input configuration register. The register has R/W access. In this table, the bit indexing scheme is as shown in Figure 3-1.

Table 3-9: Input Configuration Register

Address	Name	Size	Description	JSON Variable (Type/Unit)
0x2D	Mode	1 B	• Bit 0 = 1 • Bit 1 = 1 • Bits 2-6: Ignored • Bit 7: 0/1 = Digital/Analog Input mode	*input_mode: <value>* *(unsigned/no unit)*

3.2.3.1 Default Configuration

Table 3-10 shows the default value for the Input configuration register.

Table 3-10: Default Values of Input Configuration Register

Mode	Analog Input mode[7]

Examples:

- Set Input Mode to Analog Input:
  - DL payload: { 0x **AD** 83 }
    - Register 0x2D with write bit set to true
    - "Analog Input" mode enabled

3.2.4 Accelerometer Configuration

The accelerometer transducer offers a threshold for an "impact alarm event"[8], and a threshold for an "acceleration event". It can also be polled periodically where the Sensor orientation is of interest. Table 3-11 shows a list of accelerometer configuration registers. All registers have R/W access. In this table, the bit indexing scheme is as shown in Figure 3-1.

Note: Some terminologies with explanations are as follows:

- Accelerometer (transducer) refers to the accelerometer transducer component.
- Impact alarm (event) refers to an accelerometer event based on exceeding an impact alarm event threshold. Impact alarm events are reported with an impact alarm.
- Acceleration (event) refers to an accelerometer event, independent of the impact alarm event, and based on exceeding an acceleration event threshold. Acceleration events are reported with the acceleration magnitude, acceleration vector, or both.

Table 3-11: Accelerometer Configuration Registers

Address	Name	Size	Description	JSON Variable (Type/Unit)
0x30	Impact Alarm Event Threshold	2 B	• Unsigned, 1 milli-*g*/LSB	*impact_event_threshold (unsigned/g)*
0x31	Acceleration Event Threshold	2 B	• Unsigned, 1 milli-*g*/LSB	*acceleration_event_threshold: <value> (unsigned/g)*

[7] SW default value is Digital Input. This default value is restored using register 0x72 (see Section 3.3). **However, for correct operation of the Armband Sensor, the value of this register must be modified to Analog Input.**

[8] Here "impact" generally refers to a Sensor motion event (i.e., not necessarily an *impact* to the Sensor).

0x32	Report Options	1 B	• Bit 0 (applicable to accelerometer periodic reporting only[9]): 0/1 = Impact alarm not reported/reported • Bit 4 (applicable to both accelerometer periodic reporting and acceleration event reporting): 0/1 = Acceleration magnitude not reported/reported • Bit 5 (applicable to both accelerometer periodic reporting and acceleration event reporting): 0/1 = Acceleration vector not reported/reported • Bits 1-3, 6, 7: Ignored	*accelerometer_tx {* *report_alarm_enabled: <value>, (unsigned/no unit)* *report_magnitude_enabled: <value>, (unsigned/no unit)* *report_vector_enabled: <value> (unsigned/no unit)* *}*
0x33	Acceleration Event Debounce Time	2 B	• Seconds to wait before possibly reporting an acceleration event again • Acceptable values: 1, 2, …, 65535 • 0: Invalid and ignored	*acceleration_impact_grace_period: <value>* *(unsigned/seconds)*
0x34	Mode	1 B	• Bit 0: 0/1 = Impact alarm event threshold disabled/enabled • Bit 1: 0/1 = Acceleration event threshold disabled/enabled • Bits 2, 3: Ignored • Bit 4/5/6: 0/1 = X/Y/Z-axis disabled/enabled • Bit 7: 0/1 = Accelerometer power off/on	*accelerometer {* *impact_threshold_enabled: <value>, (unsigned/no unit)* *acceleration_threshold_enabled: <value>, (unsigned/no unit)* *xaxis_enabled: <value>, (unsigned/no unit)* *yaxis_enabled: <value>, (unsigned/no unit)* *zaxis_enabled: <value>,*

[9] This bit only controls whether the impact alarm status (i.e., raised or cleared) will be present in periodic reporting when such accelerometer periodic reporting is enabled (see Section 3.2.1). This bit does not control reporting of the impact alarm status for impact alarm events. If the impact alarm event threshold is enabled (register 0x34, bit 0), an impact alarm is always raised (reported) when the impact alarm event threshold (register 0x30) is exceeded, and is cleared after an impact alarm event grace period (register 0x36) elapses without any impact alarm events (see Section 3.2.4.7).

				(unsigned/no unit) *poweron: <value>* *(unsigned/no unit)* *}*
0x35	Sensitivity	1 B	• Bits 0-2 (Sample Rate): 0: Invalid and ignored 1/2/3/4/5/6/7 = 1/10/25/50/100/200/400 Hz • Bits 4-5 (Measurement Range[10]): 0/1/2/3 = ±2 *g*/±4 *g*/±8 *g*/±16 *g* • Bits 3, 6, 7: Ignored	*sensitivity {* *accelerometer_sample_rate: <value>,* *(unsigned/Hz)* *accelerometer_measurement_range: <value>,* *(unsigned/g)* *}*
0x36	Impact Alarm Event Grace Period	2 B	• Impact alarm grace period in sec (time to pass after the last impact alarm before the alarm can be cleared) • Acceptable values: 15, 16, …, 65535 • Other values: Invalid and ignored	*impact_alarm_grace_period: <value>* *(unsigned/seconds)*
0x37	Impact Alarm Event Threshold Count	2 B	• Number of impact alarm events before an impact alarm is raised • Acceptable values: 1, 2, …, 65535 • 0: Invalid and ignored	*impact_alarm_threshold_count: <value>* *(unsigned/no unit)*
0x38	Impact Alarm Event Threshold Period	2 B	• Period in sec over which impact alarm events are counted for threshold detection • Acceptable values: 5, 6, …, 65535 • Other values: Invalid and ignored	*impact_alarm_threshold_period: <value>* *(unsigned/seconds)*

3.2.4.1 Impact Alarm Event Threshold

This parameter is the *g*-threshold for an impact alarm event. Impact alarm events are reported only if,

- the impact alarm event threshold (bit 0 of register 0x34) is enabled; and

[10] Measurement ranges ±2 *g*, ±4 *g*, ±8 *g*, ±16 *g* correspond to typical transducer output precisions of 16 m*g*, 32 m*g*, 64 m*g*, 192 m*g*, respectively. Note that if a threshold configured in register 0x30 or register 0x31 is equal to or greater than the configured measurement full scale (2 *g*, 4 *g*, 8 *g*, 16 *g*), then the corresponding event (impact alarm or acceleration event) will never be triggered.

- the impact alarm event threshold is exceeded on at least one of the enabled axes (X, Y, Z) within a period (Impact Alarm Event Threshold Period—register 0x38) for more than a number of times (Impact Alarm Event Threshold Count—register 0x37).

3.2.4.2 Acceleration Event Threshold

This parameter is the g-threshold for an acceleration event. Provided that the acceleration threshold is enabled (bit 1 of register 0x34), acceleration events are reported as soon as the Acceleration Event Threshold is exceeded on at least one of the enabled axes (X, Y, Z). However, acceleration event interrupts are totally ignored (not registered) for a time period equal to the Acceleration Event Debounce Time (register 0x33) after a registered (and thus reported) acceleration event.

3.2.4.3 Report Options

Determines what is reported (transmitted) in the case of an acceleration event or accelerometer periodic transmission. The parameters to report include the status of the impact alarm (alarm on/off), the acceleration magnitude $\|\langle x, y, z\rangle\| = \sqrt{x^2 + y^2 + z^2}$, and the acceleration vector $\langle x, y, z\rangle$.

3.2.4.4 Acceleration Event Debounce Time

Interrupts due to acceleration events are disabled for a configurable time frame, called the Acceleration Event Debounce Time, after an acceleration event is registered. This is done to prevent a single acceleration event from being transmitted as multiple events. The minimum debounce time is 1 (one) sec. Value 0 is invalid and ignored.

3.2.4.5 Mode

When not being used in an end-user application, the accelerometer transducer can be put in the power-down mode to save battery life. Otherwise, the accelerometer is put in the low-power mode, which is an active and operational, but a low consumption, mode for the accelerometer.

Additionally, impact alarm and acceleration event thresholds can be enabled/disabled. Disabling a threshold prevents the Sensor from generating the corresponding event. It is also possible to enable/disable X, Y, Z axes independently. When an axis is disabled, it is not considered in monitoring impact alarm or acceleration events.

3.2.4.6 Sensitivity

When powered on, the accelerometer always samples the transducer element at a fixed rate, called the Sample Rate. To capture an impact alarm or acceleration event, the physical event needs to last longer than the sample period. Larger sample rates have a shorter period and can therefore resolve shorter impacts. However, sampling the transducer at a larger rate increases the power usage, impacting the battery life. Table 3-12 shows how much continuous current draw is expectable to be drawn from a 3.2-V battery for the

different sample rates when the accelerometer is powered on. For example, the sample rate of 1 Hz would translate to about 15 mAh/year battery usage, while a sample rate of 50 Hz would triple that usage.

Table 3-12: Typical Current Draws at 3.2 V for Different Accelerometer Sample Rates

Sample Rate [Hz]	1	10	25	50	100	200	400
Current Draw [μA]	1.6	2.3	3.1	4.7	7.8	14.1	28.1

Furthermore, the Sensitivity register sets the measurement range or full scale, which shows the dynamic range of accelerations that can be monitored on any enabled axis. Note that when active, the accelerometer is always put in its low power mode, which means the output acceleration values on any given axis (X, Y, or Z), is an 8-bit signed number. Therefore, a measurement range of ±2 *g* implies a precision of 4/256 *g*/LSB.

3.2.4.7 Impact Alarm Event Grace Period

The Grace Period determines how long the Sensor waits before the previously reported impact alarm event is considered clear. For example, a Grace Period of 5 (five) min results in the sensor transmitting “Impact Detected” when there is movement, and “Impact Alarm Cleared” 5 (five) min after the Sensor has been still.

The minimum acceptable value for this register is 15. Values smaller than 15 are invalid and ignored.

3.2.4.8 Impact Alarm Event Threshold Count

The accelerometer generates an impact alarm event each time it detects movement. Depending on the customer use case, it may be desirable to increase the threshold count to reduce sensitivity. This feature is to allow customers to filter out short impact events, while still allowing longer impact events to be reported.

The minimum acceptable value for this register is 1. Value 0 is invalid and ignored.

3.2.4.9 Impact Alarm Event Threshold Period

The Impact Alarm Event Threshold Period is the amount of time that impact alarm events are accumulated for threshold detection. For example, an Impact Alarm Event Threshold Period of 10 (ten) sec accumulates impact alarm events over a 10 (ten)-sec period from the time of first detection. If the Impact Alarm Event Threshold Count is reached before the time expires, the sensor reports “Impact Detected”, otherwise it does not report.

The minimum acceptable value for this register is 5. Values smaller than 5 are invalid and ignored.

3.2.4.10 Default Configuration

Table 3-13 shows the default values for the accelerometer configuration registers.

Table 3-13: Default Values of Accelerometer Configuration Registers

Impact Alarm Event Threshold	1500 milli-*g*
Acceleration Event Threshold	3000 milli-*g*
Report Options	Acceleration vector

Acceleration Event Debounce Time	2 sec
Mode	• Impact alarm threshold disabled • Acceleration threshold disabled • X-axis, Y-axis, and Z-axis enabled • Accelerometer powered off
Sensitivity	• Sample rate 1 Hz • Measurement range ±8 *g*
Impact Alarm Event Grace Period	300 sec (5 min)
Impact Alarm Event Threshold Count	1
Impact Alarm Event Threshold Period	15 sec

3.2.5 Temperature/RH/Analog Input Threshold Configuration

The Armband Sensor supports threshold transmission on three different transducer values:

- (On-board) Temperature: Measured by the on-board temperature/RH transducer
- Ambient RH: Measured by the on-board Temperature/RH transducer
- MCU Temperature: Measured by the MCU
- Analog Input Voltage: When the Input is in the Analog Input mode. The Input of the Sensor is always configured as Analog for the Armband Sensor. In the Analog Input mode, threshold transmission can be configured on the thermistor voltage reading (the voltage reading has 1-1 correspondence to a °C reading).

When a threshold on a transducer is enabled, the Sensor reports the transducer value when it leaves the configured threshold window, and once again when the transducer value re-enters the threshold window.[11] The Threshold mode is compatible with periodic reporting. Table 3-14 shows a list of configuration registers for the temperature/RH/Analog Input threshold setting. All the registers have R/W access. In this table, the bit indexing scheme is as shown in Figure 3-1.

Table 3-14: Temperature/RH/Analog Input Threshold Configuration Registers

Address	Name	Size	Description	JSON Variable (Type/Unit)
0x39	On-board Temperature/RH Sample Period: Idle	4 B	• Sample period of On-board Temperature/RH transducer: Idle state (sec) • Acceptable values: 30, 31, …, 86400 • Other values: Invalid and ignored	*temperature_relative_humidity_sample_period_idle: <value> (unsigned/sec)*

[11] Note that the threshold window here is defined as the open interval "(Low Threshold, High Threshold)", not e.g., the closed interval "[Low Threshold, High Threshold]"; i.e. even if the transducer value is equal to Low Threshold or High Threshold, the Sensor is considered to have left the threshold window.

0x3A	On-board Temperature/RH Sample Period: Active	4 B	• Sample period of On-board Temperature/RH transducer: Active state (sec) • Acceptable values: 30, 31, …, 86400 • Other values: Invalid and ignored	*temperature_relative_humidity_sample_period_active: <value> (unsigned/sec)*
0x3B	Low/High On-board Temperature Thresholds	2 B	• Bits 8-15: High temperature threshold (signed, 1°C / LSB) • Bits 0-7: Low temperature threshold (signed, 1°C / LSB) • High threshold ≤ Low threshold: Invalid and ignored	*onboard_temperature_threshold {* *high: <value>* *(signed/°C)* *low: <value>* *(signed/°C)* *}*
0x3C	On-board Temperature Thresholds Enabled	1 B	• Bit 0: 0/1 = Thresholds disabled/enabled • Bits 1-7: Ignored	*onboard_temperature_threshold_enabled: <value> (unsigned/no unit)*
0x3D	Low/High Ambient RH Thresholds	2 B	• Bits 8-15: High RH threshold (unsigned, 1% RH / LSB) • Bits 0-7: Low RH threshold (unsigned, 1% RH / LSB) • High threshold ≤ Low threshold: Invalid and ignored	*relative_humidity_threshold {* *high: <value>,* *(unsigned/%)* *low: <value>* *(unsigned/%)* *}*
0x3E	Ambient RH Thresholds Enabled	1 B	• Bit 0: 0/1 = Thresholds disabled/enabled • Bits 1-7: Ignored	*relative_humidity_threshold_enabled: <value> (unsigned/no unit)*
0x40	MCU Temperature Sample Period: Idle	4 B	• Sample period of MCU temperature transducer: Idle state (sec) • Acceptable values: 30, 31, …, 86400 • Other values: Invalid and ignored	*mcu_temperature_sample_period_idle: <value> (unsigned/sec)*
0x41	MCU Temperature Sample Period: Active	4 B	• Sample period of MCU temperature transducer: Active state (sec) • Acceptable values: 30, 31, …, 86400 • Other values: Invalid and ignored	*mcu_temperature_sample_period_active: <value> (unsigned/sec)*
0x42	Low/High MCU Temperature Thresholds	2 B	• Bits 8-15: High MCU temperature threshold (signed, 1°C / LSB) • Bits 0-7: Low MCU temperature threshold (signed, 1°C / LSB) • High threshold ≤ Low threshold: Invalid and ignored	*mcu_temperature_threshold {* *high: <value>,* *(signed/°C)* *low: <value>* *(signed/°C)*

				}
0x43	MCU Temperature Thresholds Enabled	1 B	• Bit 0: 0/1 = Thresholds disabled/enabled • Bits 1-7: Ignored	*mcu_temperature_threshold_enabled: <value>* *(unsigned/no unit)*
0x44	Analog Input Sample Period: Idle	4 B	• Sample period of analog input: Idle state (sec) • Acceptable values: 30, 31, …, 86400 • Other values: Invalid and ignored	*analog_sample_period_idle: <value>* *(unsigned/sec)*
0x45	Analog Input Sample Period: Active	4 B	• Sample period of analog input: Active state (sec) • Acceptable values: 30, 31, …, 86400 • Other values: Invalid and ignored	*analog_sample_period_active: <value>* *(unsigned/sec)*
0x46	Low/High Analog Input Thresholds	4 B	• Bits 16-31: High analog input threshold (unsigned, 1 mV/LSB) • Bits 0-15: Low analog input threshold (unsigned, 1 mV/LSB) • High threshold ≤ Low threshold: Invalid and ignored	*analog_input_threshold {* *high: <value>,* *(unsigned/V)* *low: <value>* *(unsigned/V)* *}*
0x4A	Analog Input Thresholds Enabled	1 B	• Bit 0: 0/1 = Thresholds disabled/enabled • Bits 1-7: Ignored	*analog_input_threshold_enabled: <value>* *(unsigned/no unit)*

3.2.5.1 Temperature/RH/Analog Input Sample Period: Idle

The idle sample period determines how often the transducer is checked when the reported value is within the threshold window. When first enabled, the transducer starts in the Idle state.

The minimum Sample Period in the Idle state is 30 sec, and the maximum is 86,400 sec (one day). Values smaller than 30 for this register are invalid and ignored.

3.2.5.2 Temperature/RH/Analog Input Sample Period: Active

The active sample period determines how often the transducer is checked when the reported value is outside the threshold window.

The minimum Sample Period in the Active state is 30 sec, and the maximum is 86,400 sec (one day). Values smaller than 30 for this register are invalid and ignored.

3.2.5.3 Temperature/RH/Analog Input Thresholds

The thresholds are stored in a single 2-byte register, with the MSB storing the upper threshold, and the LSB storing the lower threshold. On-board or MCU Temperature thresholds have a precision of 1°C per bit, and are

stored/transmitted as 2-s complement numbers. The RH thresholds have a precision of 1% per bit, and are stored/transmitted as unsigned numbers. The Analog Input thresholds are also unsigned numbers, and have a precision of 1 mV per bit.

In all cases, the upper threshold must be greater than the lower threshold. Otherwise, the configuration is considered invalid and ignored.

3.2.5.4 Temperature/RH/Analog Input Thresholds Enabled

The Thresholds Enabled registers enable and disable the threshold reporting on the specified transducer. Thresholds and Sample Periods can be configured but are not activated unless the Thresholds Enabled bit is set.

3.2.5.5 Default Configuration

Table 3-15 shows the default values for the threshold configuration registers.

Table 3-15: Default Values of Threshold Configuration Registers

On-board Temperature/RH Sample Period: Idle	60 sec
On-board Temperature/RH Sample Period: Active	30 sec
On-board Temperature Threshold: High	30°C
On-board Temperature Threshold: Low	15°C
On-board Temperature Thresholds Enabled	Disabled
Ambient RH Threshold: High	80%
Ambient RH Threshold: Low	15%
Ambient RH Thresholds Enabled	Disabled
MCU Temperature Sample Period: Idle	300 sec
MCU Temperature Sample Period: Active	60 sec
MCU Temperature Threshold: High	30°C
MCU Temperature Threshold: Low	15°C
MCU Temperature Thresholds Enabled	Disabled
Analog Input Sample Period: Idle	60 sec
Analog Input Sample Period: Active	30 sec
Analog Input Threshold: High	1200 mV
Analog Input Threshold: Low	600 mV
Analog Input Thresholds Enabled	Disabled

Examples:

- Set On-board Temperature Thresholds:
  - DL payload: { 0x **BB** 19 0A }
    - Register 0x3B with write bit set to true
    - High threshold set to 25°C

        - Low threshold set to 10°C
- Read On-board Temperature/RH Sample Periods:
    - DL payload: { 0x **39 3A** }
        - Registers 0x39 and 0x3A with their write bits set to false
- Set and enable Ambient RH thresholds:
    - DL payload: { 0x **BD** 3C 14 **BE** 01 }
        - Registers 0x3D and 0x3E with their write bits set to true
        - High RH thresholds set to 60% RH
        - Low RH threshold set to 20% RH
        - RH thresholds enabled

3.3 Command and Control

Configuration changes are not retained after a power cycle unless they are saved in the Flash memory. Table 3-16 shows the structure of the Command & Control Register. In this table, the bit indexing scheme is as shown in Figure 3-1.

Table 3-16: Sensor Command & Control Register

Address	Access	Name	Size	Description	JSON Variable (Type/Unit)
0x70	W	Flash Write Command	2 B	• Bit 14: • 0/1 = Do not write/Write LoRaMAC Config • Bit 13: • 0/1 = Do not write/Write App Config • Bit 0: • 0/1 = Do not restart/Restart Tracker • Bits 1-12, 15: Ignored	*write_to_flash {* *app_configuration: <value>, (unsigned/no unit)* *lora_configuration: <value>, (unsigned/no unit)* *restart_sensor: <value> (unsigned/no unit)* *}*
0x71	R	FW Version	7 B	• Bits 48-55: App version major • Bits 40-47: App version minor • Bits 32-39: App version revision • Bits 24-31: LoRaMAC version major • Bits 16-23: LoRaMAC version minor • Bits 8-15: LoRaMAC version revision • Bits 0-7: LoRaMAC region number (see Section 3.3.1)	*firmware_version {* *app_major_version: <value>, (unsigned/no unit)* *app_minor_version: <value>, (unsigned/no unit)* *app_revision: <value>, (unsigned/no unit)* *loramac_major_version: <value>, (unsigned/no unit)*

					loramac_minor_version: <value>, (unsigned/no unit) *loramac_revision: <value>, (unsigned/no unit)* *region: <value> (unsigned/no unit)* *}*
0x72	W	Reset Config Registers to Factory Defaults[12]	1 B	• 0x0A = Reset App Config • 0xB0 = Reset LoRa Config • 0xBA = Reset both App and LoRa Configs • Any other value: Invalid and ignored	*configuration_factory_reset: <value> (unsigned/no unit)*

Note: The Command & Control Register is always executed after the full DL configuration message has been decoded. The reset command should always be sent as an "unconfirmed" DL message. Failure to do so may cause a poorly designed NS to continually reboot the Sensor.

3.3.1 LoRaMAC Region

The LoRaMAC region is indicated by B_6 in the FW Version register (register 0x71). Current LoRaMAC regions and corresponding region numbers are listed in Table 3-17.

Table 3-17: LoRaMAC Regions and Region Numbers

LoRaMAC Region	Region Number
EU868	0
US915	1
AS923	2
AU915	3
IN865	4
KR920	6
RU864	7

Examples:

[12] After sending the reset-to-factory-defaults command, the Sensor is automatically reset with corresponding default configuration values.

- Write Application Configuration to Flash memory
    - DL payload: { 0x **F0** 20 00 }
- Write Application and LoRa Configurations to Flash memory
    - DL payload: { 0x **F0** 60 00 }
- Reboot Device
    - DL payload: { 0x **F0** 00 01 }
- Get FW version, and reset App Config to factory defaults
    - DL payload: { 0x **71 F2** 0A}

4 UL Data Converter

The UL data converter is implemented in the (network-side) application as a function that receives the decrypted FRMPayload bytes and the LoRaWAN UL port, and returns a structure with different fields that represent decoded transducers data. The following shows an example of such a data converter in JavaScript:

```
function Decoder(bytes, port) {

copy_bytes = [];
for (var i = 0; i < bytes.length; i++) {
      copy_bytes.push(bytes[i] >= 0 ? bytes[i] : bytes[i]+256);
}

return decode_data(copy_bytes, port);

function decode_data(data, port) {

      decoded_data = {};
      decoder = [];

      if (port === 10) {
            decoder = [
            {
                  key: [0x00, 0xFF],
                  fn: function(arg) {
                        // Battery voltage
                        decoded_data.battery_voltage = Math.round( decode_field(arg, 0,
15, "unsigned") ) / 100;
                        return 2;
                  }
            },
            {
                  key: [0x01, 0x00],
                  fn: function(arg) {
                        // Reed switch state
                        decoded_data.reed_switch_state = decode_field(arg, 0, 0,
"unsigned");
                        return 1;
                  }
            },
            {
                  key: [0x03, 0x67],
                  fn: function(arg) {
                        // On-board temperature
                        decoded_data.temp_shtc3 = Math.round( decode_field(arg, 0, 15,
"signed") ) / 10;
                        return 2;
                  }
            },
            {
                  key: [0x0B, 0x67],
                  fn: function(arg) {
                        // MCU temperature
                        decoded_data.temp_mcu = Math.round( decode_field(arg, 0, 15,
"signed") ) / 10;
```

```
                        return 2;
                  }
            },
            {
                  key: [0x04, 0x68],
                  fn: function(arg) {
                        // Relative humidity
                        decoded_data.relative_humidity = Math.round( decode_field(arg, 0,
7, "unsigned") * 5 ) / 10;
                        return 1;
                  }
            },
            {
            key: [0x05, 0x02],
            fn: function(arg) {
                // Acceleration magnitude
                decoded_data.acceleration_magnitude = decode_field(arg, 0, 15,
"signed")*0.001;
                return 2;
            }
        },
        {
            key: [0x07, 0x71],
            fn: function(arg) {
                // Acceleration vector
                decoded_data.acceleration_x = decode_field(arg, 0, 15, "signed")*0.001;
                decoded_data.acceleration_y = decode_field(arg, 16, 31, "signed")*0.001;
                decoded_data.acceleration_z = decode_field(arg, 32, 47, "signed")*0.001;
                return 6;
            }
        },
            {
            key: [0x0C, 0x00],
            fn: function(arg) {
                // Impact alarm
                decoded_data.impact_alarm = decode_field(arg, 0, 0, "unsigned");
                return 1;
            }
        },
        {
                  key: [0x08, 0x04],
                  fn: function(arg) {
                        // Reed switch count
                        decoded_data.reed_switch_count = decode_field(arg, 0, 15,
"unsigned");
                        return 2;
                  }
            },
            {
                  key: [0x11, 0x02],
                  fn: function(arg) {
                        // Thermistor temperature
                        raw_voltage = decode_field(arg, 0, 15, "unsigned")*1;
                        decoded_data.analog_input = raw_voltage;
                        decoded_data.temp_therm = Math.round(10 * (1 / (Math.log(68.1 *
raw_voltage / (1800 - raw_voltage) / 100) / 4250 + 1 / (273.15 + 25)) - 273.15)) / 10;
                        return 2;
```

```
            }
      }
      ];
}

decoded_data.port = port;
decoded_data.payload = Byte2Hex(data);

var first_round_passed = true;
var data_processed = data;
for ( var bytes_left = data.length; bytes_left > 0; ) {
      var found = false;
      for ( var i = 0; i < decoder.length; i++ ) {
            var item = decoder[i];
            var key = new Uint8Array(item.key);
            var keylen = key.length;

            header = data_processed.slice(0, keylen);

            // Header in the data matches to what we expect
            if ( is_equal(header, key) ) {
                  var f = item.fn;
                  consumed = f(data_processed.slice(keylen)) + keylen;
                  bytes_left -= consumed;
                  // Advance to the next chunk
                  data_processed = data_processed.slice(consumed);
                  found = true;
                  break;
            }
      }
      if ( !found ) {
            first_round_passed = false;
            break;
      }
}

if ( !first_round_passed ) {
      var data_processed2 = data.slice(4);
      for ( var bytes_left2 = data.length - 4; bytes_left2 > 0; ) {
            var found2 = false;
            for ( var i2 = 0; i2 < decoder.length; i2++) {
                  var item2 = decoder[i2];
                  var key2 = new Uint8Array(item2.key);
                  var keylen2 = key2.length;

                  header2 = data_processed2.slice(0, keylen2);

                  // Header in the data matches to what we expect
                  if ( is_equal(header2, key2) ) {
                        var f2 = item2.fn;
                        consumed2 = f2(data_processed2.slice(keylen2)) + keylen2;
                        bytes_left2 -= consumed2;
                        // Advance to the next chunk
                        data_processed2 = data_processed2.slice(consumed2);
                        found2 = true;
                        break;
                  }
```

```
                }
                if ( !found2 ) {
                      break;
                }
          }
      }

      if ( decoded_data.hasOwnProperty('temp_therm') &&
decoded_data.hasOwnProperty('temp_shtc3') ) {
            // Body temperature
            decoded_data.temp_body = 0.22 * decoded_data.temp_therm + 0.20 *
decoded_data.temp_shtc3 + 23.2;
      }

      return decoded_data;
}

function decode_field( chunk, start_bit, end_bit, data_type ) {
      chunk_size = chunk.length;
      if ( end_bit >= chunk_size * 8 ) {
          return null; // Error: exceeding boundaries of the chunk
      }

      if ( end_bit < start_bit ) {
          return null; // Error: invalid input
      }

      arr = extract_bytes(chunk, start_bit, end_bit);
      return apply_data_type(arr, data_type);
}

function extract_bytes( chunk, start_bit, end_bit ) {
      var total_bits = end_bit - start_bit + 1;
      var total_bytes = total_bits % 8 === 0 ? to_uint(total_bits / 8) :
to_uint(total_bits / 8) + 1;
      var offset_in_byte = start_bit % 8;
      var end_bit_chunk = total_bits % 8;
      var arr = new Uint8Array(total_bytes);

      for ( byte = 0; byte < total_bytes; ++byte ) {
            var chunk_idx = to_uint(start_bit / 8) + byte;
            var lo = chunk[chunk_idx] >>> offset_in_byte;
            var hi = 0;

            if ( byte < total_bytes - 1 ) {
                  hi = (chunk[chunk_idx + 1] & ((1 << offset_in_byte) - 1)) << (8 -
offset_in_byte);
            } else if ( end_bit_chunk !== 0 ) {
                  // Truncate last bits
                  lo = lo & ((1 << end_bit_chunk) - 1);
            }

            arr[byte] = hi | lo;
      }

      return arr;
}
```

```
function apply_data_type( bytes, data_type ) {
	output = 0;
	if ( data_type === 'unsigned' ) {
		for ( var i = 0; i < bytes.length; ++i ) {
			output = (to_uint(output << 8)) | bytes[i];
		}

		return output;
	}

	if ( data_type === 'signed' ) {
		for ( var i = 0; i < bytes.length; ++i ) {
			output = (output << 8) | bytes[i];
		}

		// Convert to signed, based on value size
		if ( output > Math.pow(2, 8*bytes.length-1) )
			output -= Math.pow(2, 8*bytes.length);

		return output;
	}

	if ( data_type === 'bool' ) {
		return !(bytes[0] === 0);
	}

	// Incorrect data type
	return null;
}

// Converts value to unsigned
function to_uint( x ) {
	return x >>> 0;
}

// Checks if two arrays are equal
function is_equal( arr1, arr2 ) {
	if (arr1.length != arr2.length) return false;
	for (var i = 0 ; i != arr1.length; i++) {
		if (arr1[i] != arr2[i]) return false;
	}
	return true;
}

function Byte2Hex( Data ) {
	result = '';
	arr = [];
	for(var i = 0; i < Data.length; i++) {
		arr.push(Data[i]);
		hex = arr[i].toString(16);
		if (i > 0) result += ' ';
		result += (hex.length === 2 ? hex : '0' + hex);
	}
	return result;
}
```

```
}
```

References

[1] LoRa Alliance, "LoRaWAN Specification," ver. 1.0.2, Jul 2016.

[2] LoRa Alliance, "LoRaWAN Regional Parameters," ver 1.0.2, Rev B, Feb 2017.