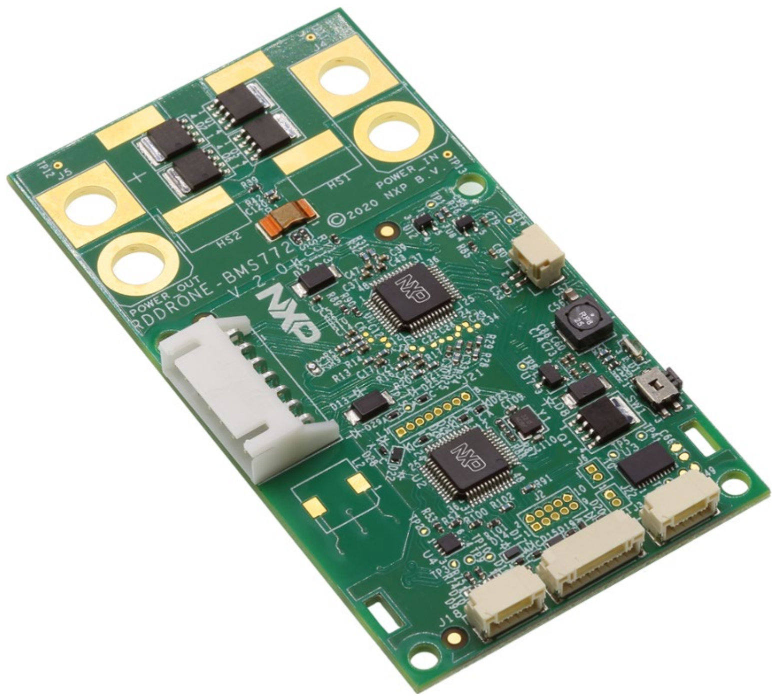
BMS772 software

Release Notes: v3.4 Release

Rev. 3.4 — Sept 11, 2020 **Release Notes**



Document information

Info	Content
Keywords	BMS772, RDDRONE, S32K144, UJA1169, MC33772
Abstract	Release notes describing package contents, instructions, open issues, fixes and limitations.

Revision history

Rev	Date	Description
0.1	20200630	Release notes for RDDRONE-BMS772
0.2	20200702	Completing the document
2.6	20200725	Updated after internal review
3.x	20200805	Updated to version 3.x (new help, features, parameters, etc)
3.4	20200911	Updated to the new release, modified state diagrams, tables and more improvements are implemented. Added self-discharge-enable variable and corrected some variables. Changed the software block diagram and added the SBC part to it.

Contact information

For more information, please visit: http://www.nxp.com

1. Introduction

This document is the release notes for the **RDDRONE-BMS772.** This evaluation kit allows customers to evaluate and perform early (software) development/prototyping on the BMS772 chipset.

The release notes describe the contents of the kit, open issues, changes, fixes and limitations of the released version.

Instructions on how to use the BMS772 evaluation kit can be found on-line:

- Gitbook: https://nxp.gitbook.io/rddrone-bms772/
- NXP webpage: https://www.nxp.com/design/designs/rddrone-bms772-smart-battery-management-for-mobile-robotics:RDDRONE-BMS772

.

2. Release package

The released package consists of:

- Hardware
    - RDDRONE-BMS772 board
    - Battery selection connectors for 3S up to 6S batteries.
- Documentation and software
    - Release notes
    - On-line documentation on gitbook
    - Software updates will be available through gitbook
    - NXP webpages with a link to gitbook

3. Changes

Table 1. Changes

Item	Description
Release package	Basic functionality to support 3S to 6S. For normal use and basic charging.
Documentation	More detailed info can be found in the on-line gitbook.

4. Limitations

Table 2. Limitations

Item	Description
Software stack	**Limitation:** The evaluation kit only contains the basic battery management system code to access and evaluate the NXP technology. **Impact:** A full BMS requires additional software. The code is supplied as is to be used at own risk.
Charging	**Limitation:** The BMS will not limit the incoming current or voltage while charging, it can only disconnect the battery. **Impact:** A current and voltage limiting power supply needs to be attached to the BMS to charge the battery.

5. Known issues

Table 3. Known issues

Item	Description
Sleep	**Applies to:** Release Package. **Issue:** Low power isn't implemented. **Workaround:** Upcoming releases.
UAVCAN battery status	**Applies to:** Release Package. **Issue:** Only the draft of the status message is implemented. **Workaround:** Update the UAVCAN code to implement the standard.
NFC	**Applies to:** Release Package. **Issue:** Isn't implemented. **Workaround:** Upcoming releases
Authentication	**Applies to:** Release Package. **Issue:** Isn't implemented. **Workaround:** Upcoming releases.
State of charge	**Applies to:** Release Package. **Issue:** Initial state of charge isn't calibrated for every battery. **Workaround:** If the initial state of charge is known, configure the remaining capacity.
Diagnostics	**Applies to:** Release Package. **Issue:** Diagnostics will not be part of the basic release **Workaround:** Diagnostics functionality can be added by obtaining additional license

For issues of older releases, please consult the respective release notes.

6. Block diagram

In the following images you can see the hardware block diagram. For more information about the hardware look at the gitbook: https://nxp.gitbook.io/rddrone-bms772/ or see the user manual RDDRONE-BMS772_UG.

6.1 Board organization

The board is organized as shown in the figures below:

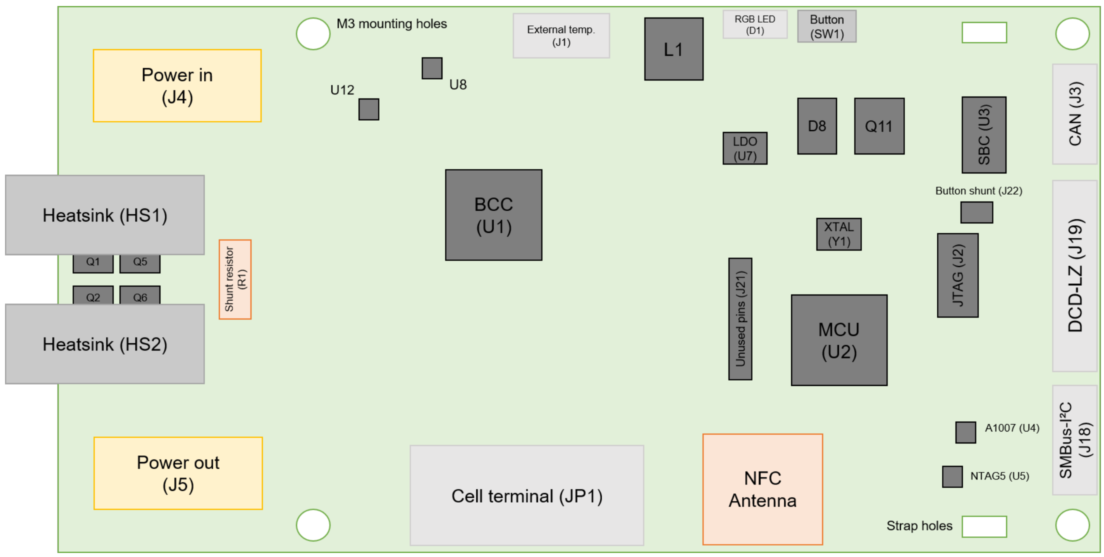


Figure 2 Board block layout -- Top

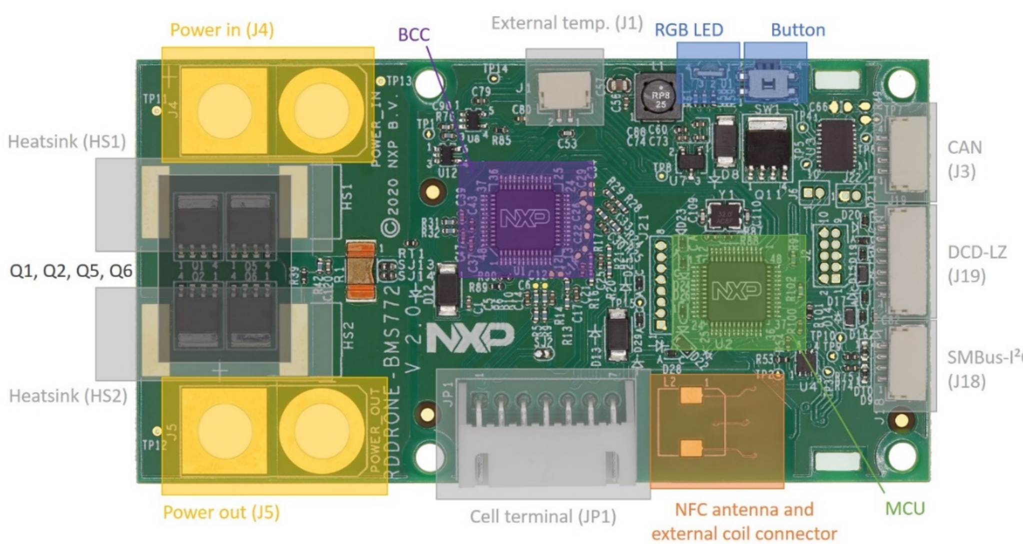


Figure 1 Board map -- Top

On the top level the main connectors are located. The battery should be connected to Power in (J4) and balance input (JP1). A correct cell terminal should be mounted, which fits the type of battery.

The important bottom connector is J20, this is normally used to attach the CAN termination.

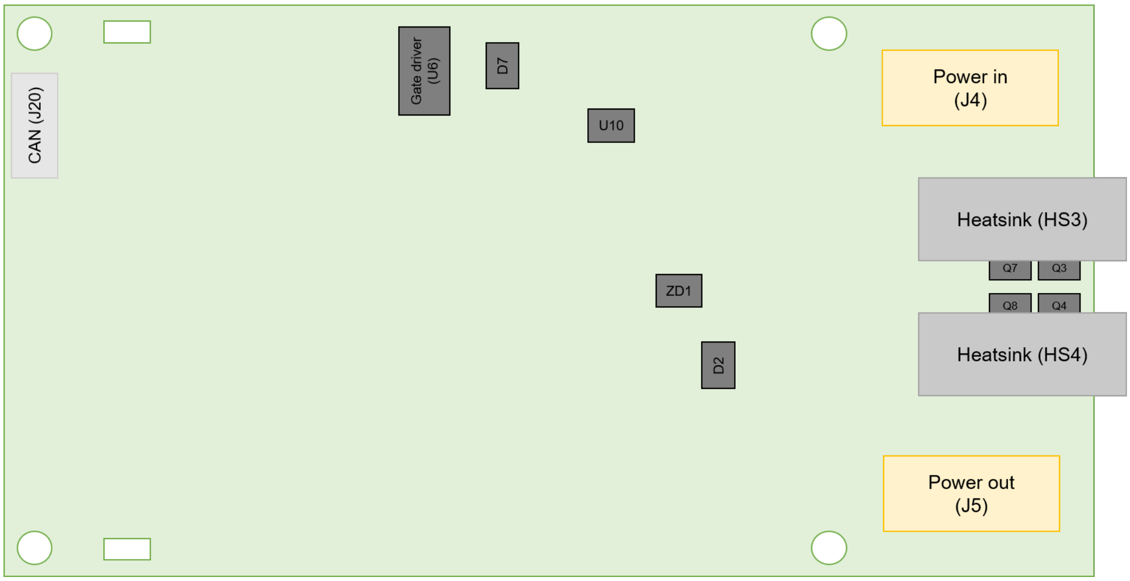


Figure 3 Board block layout -- Bottom

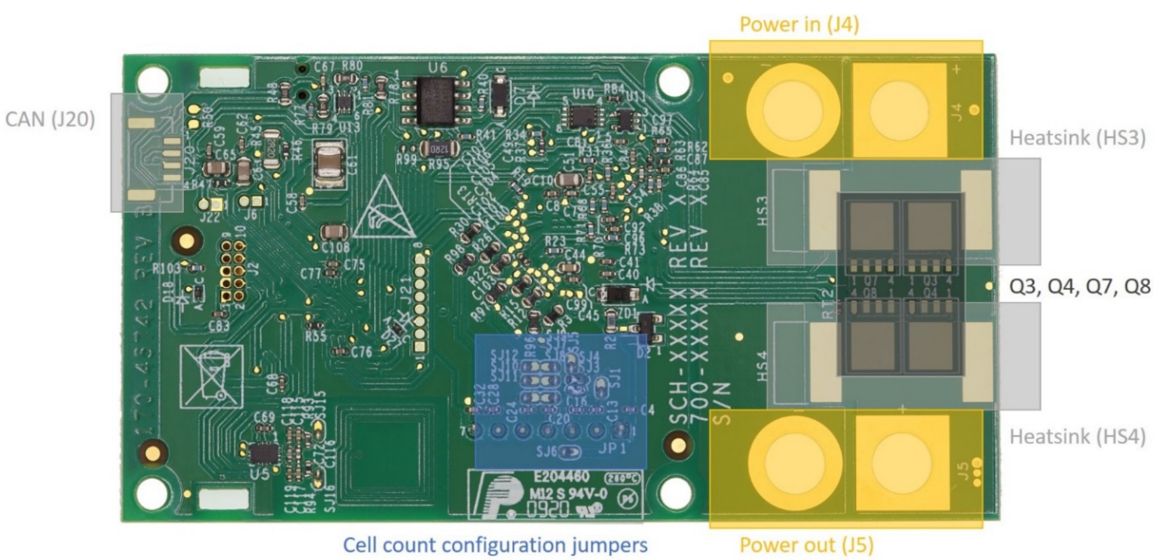


Figure 4 Board map -- Bottom

In order to correctly select the battery, solder jumpers should be set accordingly.

7. How to

7.1 How to program the BMS

7.1.1 Software setup and debugger adapter board

The software can only be written to the board using a debugger. The HoverGames drone kit includes a J-Link EDU Mini debugger. To use it, you need to install the J-Link Software Pack. Links are provided at the Downloads chapter.

The debugger can be plugged into the FMU using a small adapter board. This small PCB comes with a 3D printed case that can easily be put together. The J-Link debugger can be connected using an **SWD cable**. The connectors have to be oriented such that the **wires directly go to the side of the board**, as shown in the picture below.

While you do not need it right now, the adapter board also has a 6-pin connector for a **USB-TTL-3V3 cable**, which you can use to access the system console (CLI) of the BMS. The 3D printed case has a small **notch** on one side of the connector. The USB-TTL-3V3 cable needs to be plugged in such that the **black (ground) wire is on the same side as this notch** in the case. Make sure the cables are plugged in as shown in the picture below.

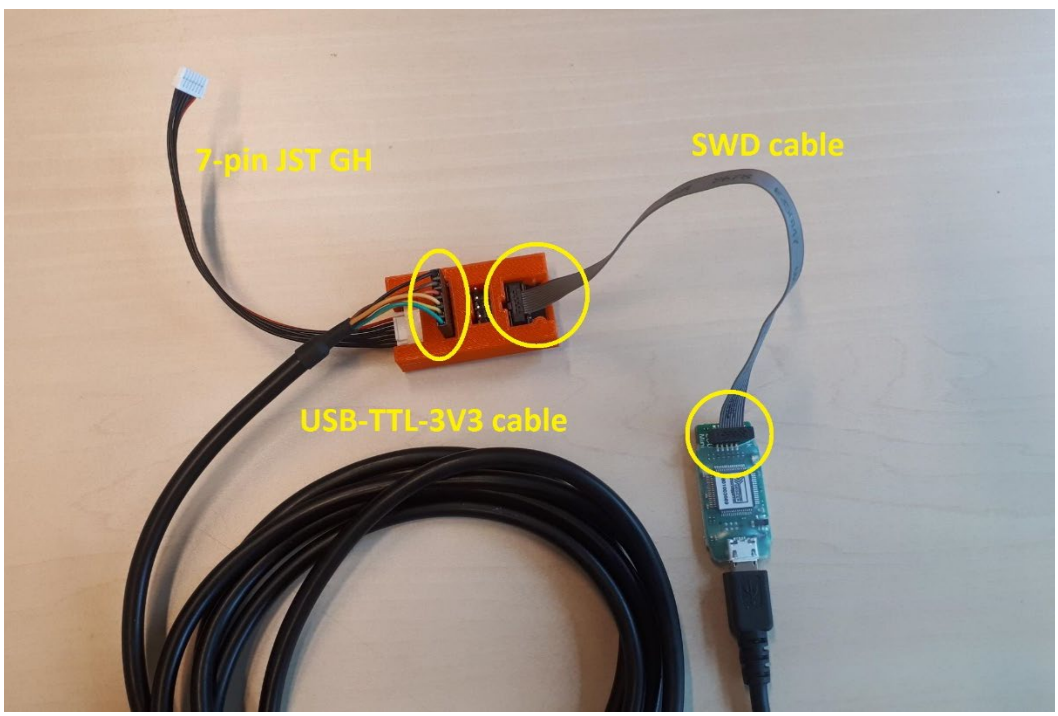


Figure 5: The debug adapter board

7.1.2 Programming the software

The firmware can be programmed to the board with the J-Link debugger. You first need to download a firmware binary (.bin file). We keep up-to-date links to the recommended releases on our downloads page, because it is important that you start with the most stable and up-to-date version of the firmware.

Connect the debugger to the BMS using the 7-pin JST-GH connector from the debug adapter board to J19 and plug the USB coming from the debugger into your computer. You should provide power to the BMS using the battery or a power supply. Then start the J-Link Commander program, and follow these steps:

To program the binary a file could be made, make a new notepad file (to enter some text). This file could be named “flash.jlink” or something else.

Add the following to the file:

```
si 1
speed 1000
device S32K144
connect S32K144
S
r
w1 0x40020007, 0x44
w1 0x40020000, 0x80
sleep 1000
loadbin "<absolute path>nutt.bin" 0
r
g
q
```

note: Change the path to the location of the nuttx binary file.

To program the binary to the BMS use the command “JLinkExe flash.jlink” in the terminal where the “flash.jlink” script is located. If you named the “flash.jlink” script differently, this needs to change in the command as well.

7.1.3 Downloads

Download J-Link Software and Documentation Pack

J-Link Commander is used to flash binaries onto the RDDRONE-BMS772 board. The latest (stable) release of the J-Link Software and Documentation Pack is available at the SEGGER website for different operating systems.

7.2 How to use the (UAV)CAN interface

Connect the UAVCAN device (like the RDDRONE-UCANS32K146) to the BMS using 2x JST-GH 4 pin connectors with 1 on 1 wires. End the CAN bus with a 120Ω terminator resistor between CAN high and CAN low.

7.2.1 Get the Battery status draft using the UCAN board in Linux:

Connect the UCAN board with a USB-TTL-3V3 cable to the laptop.
See the Get the UAVCAN messages with the UCAN board page on the gitbook for more information.

7.3 How to use the CLI

To use the command line interface, connect the debugger to the BMS using the 7-pin JST-GH connector from the debug adapter board to J19 and plug the USB coming from the debugger converter board into your computer. Open a UART terminal like minicom on a Linux machine or PuTTY or Tera Term for a windows machine.

Type “bms help” to get the help for the CLI. The CLI works only with lowercase commands. The settings are:

- 115200 Baud
- 8 data bits
- 1 stop bit

7.4 How to configure the temperature sensor

By default the temperature sensor is not enabled. To configure the temperature sensor to be used, this temperature sensor need to be connected to J1 using a 2-pin JST-GH connector. The maximum length of the cable that leads to the temperature sensor needs to be 20cm. This temperature sensor needs to be a 10k NTC, like the NTCLE100E3103JB0 from Vishay. With the CLI type:
“bms set sensor-enable 1”.

8. The parameters

8.1 How to configure the parameters

At startup the saved parameters are read from the flash. If there is nothing saved, the CRC check will fail, and it will set the default parameters.
The parameters can be set with the CLI. use “help parameters” or look at the parameter list to see what parameter you would like to change.

The parameter can be read with a “bms get <parameter>” command in the CLI, where <parameter> is the parameter **in full lower case**.

A parameter can be written with a “bms set <parameter> <x>” command in the CLI, where <parameter> is the to be written parameter and <x> is the new parameter value. Keep in mind that that this new value needs to be in the range of the to be written values. For decimals (float) use a “.” to separate it. Some parameters can’t be saved.

When you want to save a configuration to flash use “bms save” in the CLI, this will be loaded at startup or when the user types “bms load” in the CLI. If you want to restore the default values, type “bms default”. Check 8.2 to see which parameters need to be configured.

8.2 The parameter lists

8.2.1 The BMS variable list

Table 4. BMS variable list

Parameter	Unit	Datatype	Description	Default	RO/ RW	No
c-batt	C	float	The temperature of the battery	0	RO	0
v-out	V	float	The voltage of the BMS output	0	RO	1
v-batt	V	float	The voltage of the battery pack	0	RO	2
i-batt	A	float	The last recorded current of the battery	0	RO	3
i-batt-avg	A	Float	The average current since the last measurement (period T_meas (defaults))	0	RO	4
s-out	-	bool	This is true if the output power is enabled	0	RO	5
p-avg	W	float	Average power consumption over the last 10 seconds	0	RO	6
e-used	Wh	float	Power consumption since device boot	0	RO	7
a-rem	Ah	float	Remaining capacity in the battery	2.6	RW	8
a-full	Ah	float	Predicted battery capacity when it is fully charged. falls with aging	4.6	RW	9
t-full	h	float	Charging is expected to complete in this time; zero if not charging	0	RO	10
s-flags	-	uint8_t	This contains the status flags as described in BMS_status_flags_t	255	RO	11
s-health	%	uint8_t	Health of the battery in percentage, use STATE_OF_HEALTH_UNKNOWN = 127 if cannot be estimated	127	RO	12
s-charge	%	uint8_t	Percentage of the full charge 0, 100. This field is required.	55	RO	13
batt-id	-	uint8_t	Identifies the battery within this vehicle, 0 - primary battery.	0	RW	14
model-id	-	int32_t	Model id, set to 0 if not applicable	0	RW	15
model-name	-	Char[32]	Battery model name, model name is a human-readable string that normally should include the vendor name, model name and chemistry	"BMS test"	RW	16
v-cell1	V	float	The voltage of cell 1	0	RO	17
v-cell2	V	float	The voltage of cell 2	0	RO	18
v-cell3	V	float	The voltage of cell 3	0	RO	19
v-cell4	V	float	The voltage of cell 4	0	RO	20
v-cell5	V	float	The voltage of cell 5	0	RO	21
v-cell6	V	float	The voltage of cell 6	0	RO	22
c-afe	C	float	The temperature of the analog front end	0	RO	23
c-fet	C	float	The temperature of the transistor	0	RO	24
c-r	C	float	The temperature of the sense resistor	0	RO	25
n-charges	-	uint16_t	The number of charges done	0	RW	26
n-charges-full	-	uint16_t	The number of complete charges	0	RW	27

8.2.2 The BMS configuration parameters list

Table 5. BMS configuration parameters list

Parameter	Unit	Datatype	Description	Default	RO/ RW	No
n-cells	-	uint8_t	Number of cells used in the BMS board	3	RW	28
t-meas	ms	uint16_t	Cycle of the battery to perform a complete battery measurement and SOC estimation can only be 10000 or a whole division of 10000 (For example: 5000, 1000, 500).	1000	RW	29
~~**t-ftti**~~	~~ms~~	~~uint16_t~~	~~Cycle of the battery to perform diagnostics (Fault Tolerant Time Interval)~~	~~1000~~	RW	~~30~~
t-cyclic	s	uint8_t	Wake up cyclic timing of the AFE (after front end) during sleep mode	1	RW	31
i-sleep-oc	mA	uint8_t	Overcurrent threshold detection in sleep mode that will wake up the BMS and also the threshold to detect the battery is not in use	30	RW	32
v-cell-ov	V	float	Battery maximum allowed voltage for one cell. Exceeding this voltage, the BMS will go to fault mode.	4.2	RW	33
v-cell-uv	V	float	Battery minimum allowed voltage for one cell. Going below this voltage, the BMS will go to deep_sleep mode.	3	RW	34
c-cell-ot	C	float	Over temperature threshold for the cells. Going over this threshold and the BMS will go to FAULT mode	45	RW	35
c-cell-ot-charge	C	float	Over temperature threshold for the cells during charging. Going over this threshold and the BMS will go to FAULT mode	40	RW	36
c-cell-ut	C	float	Under temperature threshold for the cells. Going under this threshold and the BMS will go to FAULT mode	(-20)	RW	37
c-cell-ut-charge	C	float	Under temperature threshold for the cells during charging. Going under this threshold during charging and the BMS will go to FAULT mode	0	RW	38
a-factory	Ah	float	Battery capacity stated by the factory	4.6	RW	39
t-bms-timeout	s	uint16_t	Timeout for the BMS to go to SLEEP mode when the battery is not used.	600	RW	40
~~**t-fault-timeout**~~	~~s~~	~~uint8_t~~	~~After this timeout, the battery will leave the FAULT mode and go to SLEEP mode. T~~	~~60~~	RW	~~41~~
t-charge-detect	s	uint8_t	During NORMAL mode, if the battery voltage is positive for more than this time, then the BMS will go to CHARGE mode	1	RW	42
t-cb-delay	s	uint8_t	Time for the cell balancing function to start after entering the CHARGE mode	120	RW	43
t-charge-relax	s	uint16_t	Relaxation after the charge is complete before going to another charge round.	300	RW	44
i-charge-full	mA	uint16_t	Current threshold to detect end of charge sequence	50	RW	45
i-charge-max	A	float	Maximum current threshold to open the switch during charging	9.2	RW	46

i-out-max	A	float	Maximum current threshold to open the switch during normal operation, if not overruled	60	RW	47
v-cell-margin	mV	uint8_t	Cell voltage charge margin to decide or not to go through another topping charge cycle	30	RW	48
~~**t-ocv-cyclic0**~~	~~s~~	~~int32_t~~	~~OCV measurement cyclic timer start (timer is increase by 50% at each cycle)~~	~~300~~	RW	~~49~~
~~**t-ocv-cyclic1**~~	~~s~~	~~int32_t~~	~~OCV measurement cyclic timer final (timer is increase by 50% at each cycle)~~	~~86400~~	RW	~~50~~
c-pcb-ut	C	float	Minimal ambient temperature (measured on the PCB)	-20	RW	51
c-pcb-ot	C	float	Maximal ambient temperature (measured on the PCB)	45	RW	52
v-storage	V	float	The voltage what is specified as storage voltage for a cell	3.8	RW	53
~~**ocv-slope**~~	~~V/A.min~~	~~float~~	~~The slope of the OCV curve~~	~~0~~	RW	~~54~~
batt-eol	%	uint8_t	Percentage at which the battery is end-of-life and shouldn't be used anymore Typically between 90%-50%	80	RW	55
sensor-enable	-	bool	This variable is used to enable or disable the battery temperature sensor, 0 is disabled, 1 is enabled	0	RW	56
self-discharge-enable	-	bool	this variable is used to enable or disable the SELF_DISCHARGE state, 0 is disabled, 1 is enabled	1	RW	57
uavcan_node_static_id*	-	uint8_t	This is the node ID of the UAVCAN message	255	RW	58
uavcan-subject-id*	-	uint16_t	This is the subject ID of the UAVCAN message	4096	RW	59
uavcan-fd-mode*	-	uint8_t	If true CANFD is used, otherwise classic CAN is used	0	RW	60
uavcan-bitrate*	bit/s	int32_t	the bitrate of classical can or CAN FD arbitration bitrate	1000000	RW	61
uavcan-fd-bitrate*	bit/s	int32_t	the bitrate of CAN FD data bitrate	4000000	RW	62

A ~~line~~ means this is not implemented yet.
*these parameters will only be implemented during startup of the BMS

8.2.3 The hardware parameters

Table 6. BMS hardware parameters list

Parameter	Unit	Datatype	Description	Default	RO/ RW	No
v-min	V	uint8_t	Minimum stack voltage for the BMS board to be fully functional	6	RW	63
v-max	V	uint8_t	Maximum stack voltage allowed by the BMS board	26	RW	64
i-peak	A	uint16_t	Maximum peak current that can be measured by the BMS board	200	RW	65
i-max	A	uint8_t	Maximum DC current allowed in the BMS board (limited by power dissipation in the MOSFETs)	60	RW	66
i-short	A	uint16_t	short circuit current threshold (typical: 550A, min: 500A, max: 600A)	500	RW	67
t-short	us	uint8_t	Blanking time for the short circuit detection	20	RW	68
i-bal	mA	uint8_t	Cell balancing current under 4.2V with cell balancing resistors of 82 ohms	50	RW	69

A ~~line~~ means this is not implemented yet.

9. Charging

9.1 How to charge the battery using the BMS

The BMS cannot limit the current. It can only disconnect the battery when a fault occurs. To charge the BMS, connect a power supply that can **limit the current and the voltage**. Set this current limitation to the correct charge current and the voltage to the maximum battery pack voltage. The BMS will monitor the current and voltages. It can balance the cells by discharging cells having a higher voltage. See Main state machine explained for more information.

10. Software guide – NuttX

10.1 Introduction

The NuttX software of the BMS uses an RTOS names NuttX. NuttX is a real-time operating system (RTOS) with an emphasis on standards compliance and small footprint. Scalable from 8-bit to 32-bit microcontroller environments, the primary governing standards in NuttX are POSIX and ANSI standards.

At startup the CLI will print the version number: this explanation is about bms3.4-9.1. the first number before the – is the BMS application revision. The second number after the – is the NuttX version.

10.2 Software block diagram

In Figure 1 the software block diagram can be found. The BMS application consists of several parts. Functions from these parts can be called from the BMS application. These parts can create tasks that will run semi parallel (since it is still a single core processor). NuttX will take care of this. The CLI part is called by calling the BMS application from the nuttshell interface with commands. The explanation of the blocks can be found in below in Module description.

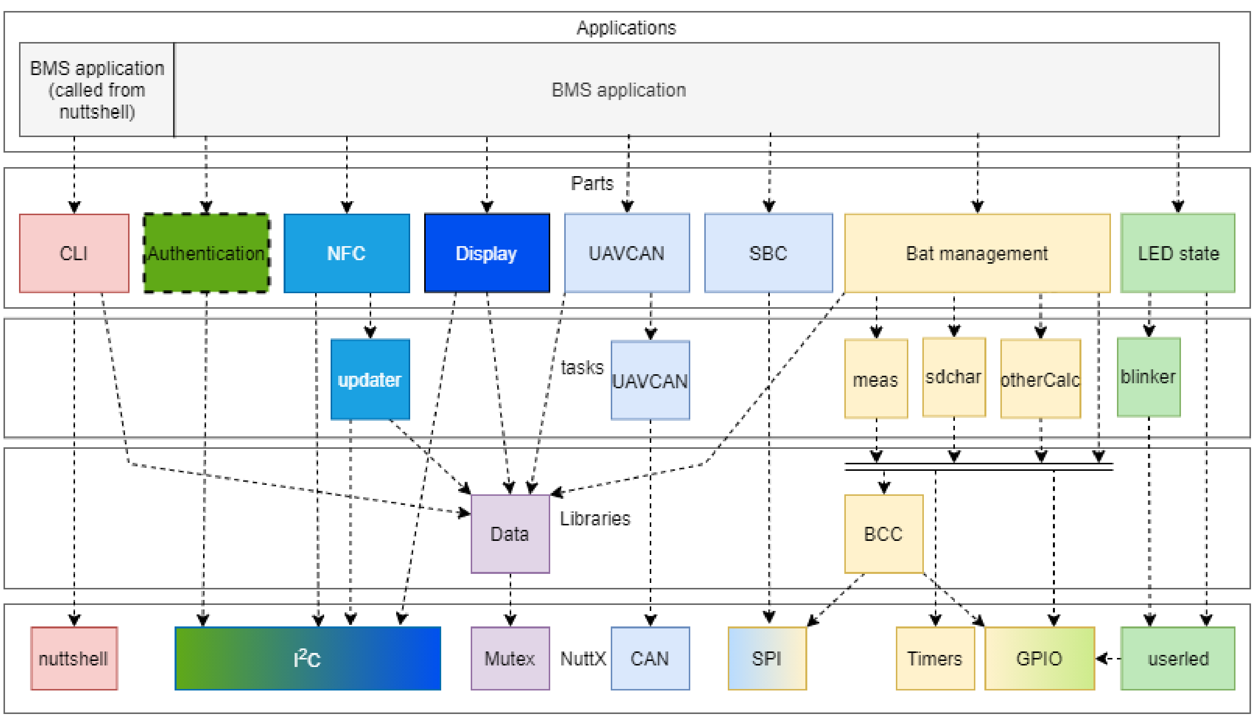


Figure 6: Software block diagram

10.2.1 Module description

CLI

The command line interface (CLI) module takes care of communication with the user through the NuttX nutshell, it can be used during debugging of the smart battery application or a specific battery under test. The communication is mapped to use a universal asynchronous receiver-transmitter (UART) also known as the root console.

The application command may be followed by optional arguments such as sleep, deepsleep, wake, reset, help, show, set or get. With the set or get command the user can read and write every value, including the configuration parameter list. These values can be read/written by calling the BMS application followed by a set or get command followed by the name of the variable. In the case of a set command this would instead be followed by the new value of the variable. Try the command “bms help” to see the help of the CLI.

Authentication - A1007

The authentication module will take care of the authentication using the A1007 chip. The A1007 is capable of secure asymmetric key exchange and storage as well as secure monotonic counters and flags for use in such things as counting charge or discharge cycles or permanently flagging under-voltage or over-temperature conditions.
This module is not implemented yet. Only verification via I2C is implemented.

NFC - NTAG5

The NFC module manages NFC communication. It needs to read all the values and should be able to write the configuration parameter list. It should be able to read the values with a refresh rate of once a second. NFC will allow the user to insert commands like wake, reset, sleep, deepsleep, etc. The updater task will be used to update the data. The NTAG5 chip is capable of operating using energy harvested from the NFC field of a reading device. It can operate in a similar manner to a double ported EEPROM, and NFC records can include standardized messages for HTTP records. In this way the NFC tag could be updated regularly with status information. That information could be added to a URL, and a smartphone would be capable of reading the URL with data attached and rendering a human readable webpage with minimal coding effort. This method removes the need for any custom software on the reading device.
This module is not implemented yet. Only verification via I2C is implemented.

Display

The display module manages information presented on an optional local I2C LCD display.
This module is not implemented yet.

UAVCAN

The UAVCAN module manages UAVCAN communication. UAVCAN V1 protocol is used to relay battery and power usage to the FMU (or host processor). It sends the battery status list on a cyclic time interval. It sends configuration data if requested. It has a task named UAVCAN that will check if data is received and will send the data if needed. The CAN PHY is in the SBC (UJA1169).

SBC - UJA1169

The SBC module manages the power of the voltage regulators in the SBC. With this module the SBC can be set in normal mode, standby mode and sleep mode. In the normal mode both V1 (powers the MCU and more) and V2 (powers internal CAN PHY) are powered. In standby mode, V2 is off and in sleep mode both regulators V1 and V2 are off. The sleep mode is needed for the DEEP SLEEP state.

Bat management

The Bat management (battery management) part is the most important part, it will oversee the whole battery management. It will be used to monitor the battery, the PCB (temperatures) and calculate voltages, temperatures, current, SoC, SoH, average power and more, it will ensure the BCC chip reacts if thresholds are exceeded. Function of this part can be used to drive the gate driver, which allows it to disconnect the battery from the output power connector on the BMS. Because this is such a large part of the system, the Bat management part can create some tasks. These tasks can all access the BCC, the timers and the GPIO. These are the tasks:

- The meas task will oversee the measurements and if triggered do the calculations.
- The otherCalc task will make sure that once every measurement cycle, the meas task will do the calculations.
- The sdchar task will oversee the self-discharging.

LED state

The LED state module can be used to set the RGB LED. It can set a RGB color on or off and blink the LEDs at given intervals. If a LED needs to blink a blinker task will be created to ensure it blinks. This module is used to inform the user visually of various states and status.

This part is used implement the LED states of Table 7.

Table 7. LED states

State	LED state
Deep sleep	Off (after 1 second white LED on)
Sleep	Off
Wake-up	Green
Normal	Green blinking (with state indication) 1 blink 0-40% 2 blinks 40-60% 3 blinks 60-80% 4 blinks 80-100%
Fault	Red blinking
Charging	Blue
Charging done	Green
Balancing/self-discharge	Blue blinking
NFC communication	Yellow blinking (not implemented yet)
Charger connected at startup	Red-blue blinking

Data

Since different parts need to use the same data, a data library will be made to take care of this. This library will make sure it is protected against usage at the “same” time by multiple tasks.

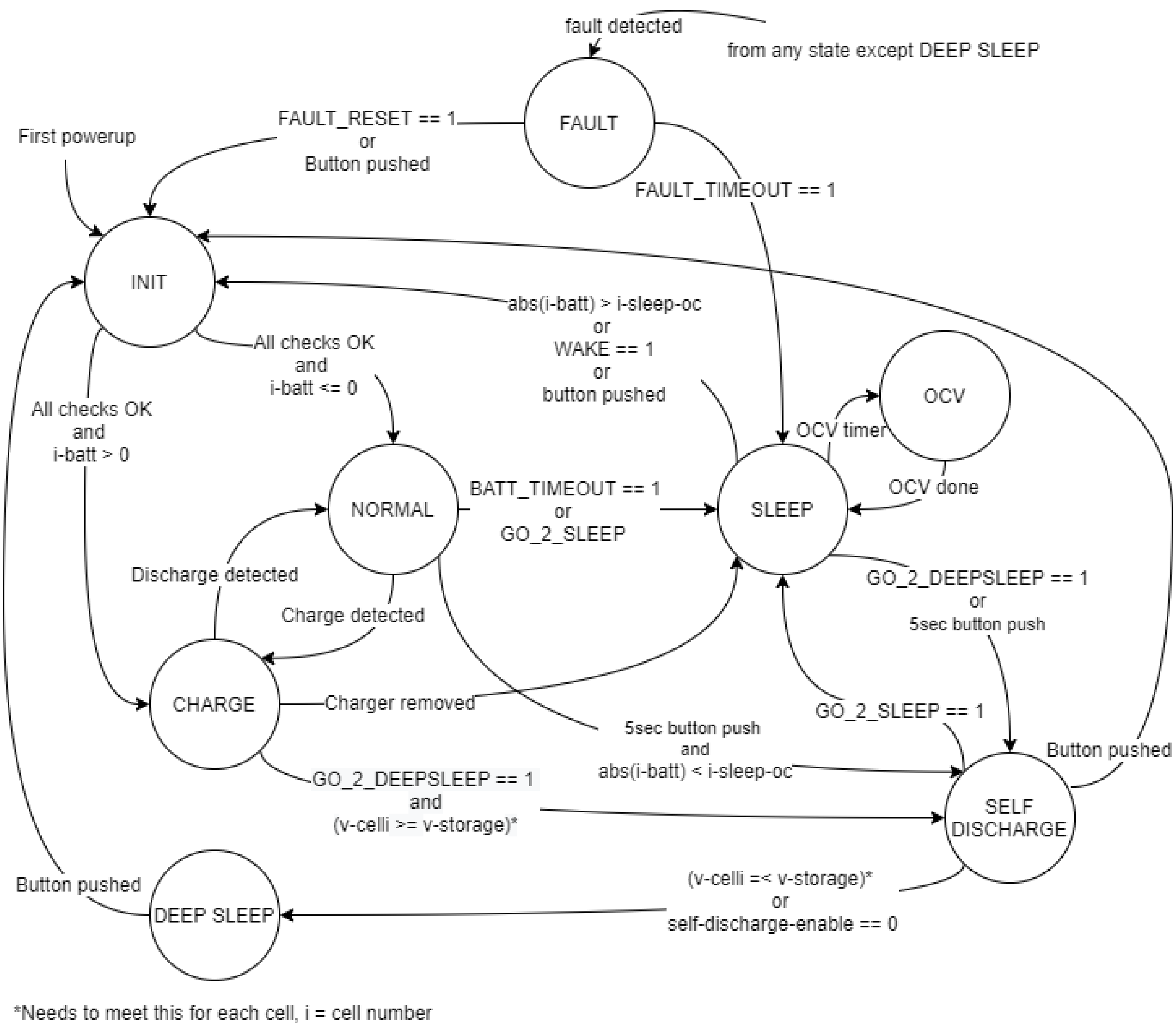
10.3 BMS application state machine

This chapter will show the designed state machine and the description of its different states

10.3.1 The main state machine

In Figure 7 the main state machine that will be implemented in the BMS can be found. This state diagram will be implemented in the BMS application.

Figure 7: Battery main state machine



10.3.2 Main state machine explained

INIT state

The INIT state is typically entered from the SLEEP state. In this state the microcontroller unit (MCU) will wake up and it will verify configurations, fault registers and functions. This is needed because it can enter the INIT state when the user resets from a fault in the FAULT state as well. When everything is OK, it will close the switches if not already closed and proceed to the next state depending on the current direction. The LED will be steady green in this state.

NORMAL state

This is the state were the battery operates how it should be, it is being discharged by the drone. Meaning that the power switches are closed. The LED will be blinking green to indicate the state of charge. In this state the BMS performs the following tasks:

- Battery voltage, cell voltage and current is measured and calculated every measurement cycle.
- SoC and SoH are estimated every measurement cycle.
- The UAVCAN BMS battery status will be send over the UAVCAN bus every measurement cycle.
- The user can read the BMS status and parameters with NFC and the CLI. The user may change the state to SLEEP.
- A timer will monitor if the current is below the sleep current for more than the timeout period. If this happens, it will go to the SLEEP state.
- It will monitor if the current flows into the battery and if the current is more than the sleep current for more than the charge detect time, the state will change to the CHARGE state.
- If the current is less than the sleep current while the button is pressed for 5 seconds, it will transition to the SELF DISCHARGE state.

CHARGE state

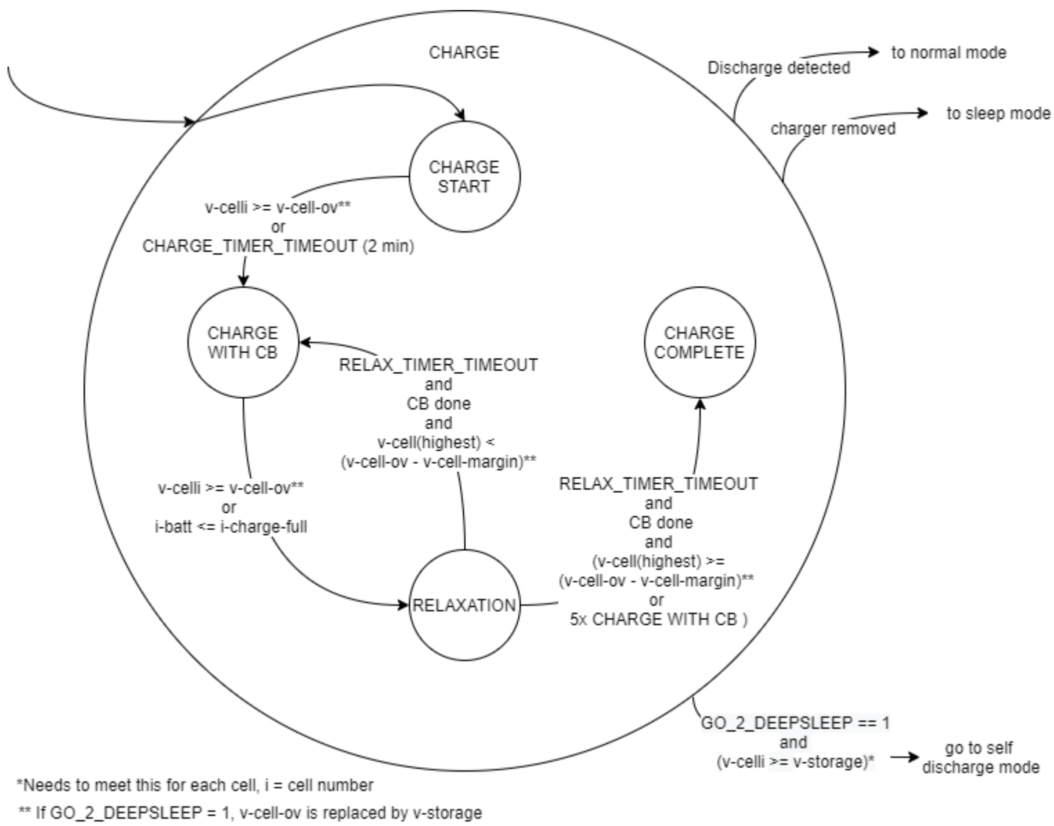
During this state the same functions as in the NORMAL state are implemented as well. The charging of the battery is done in different stages and is reflected in the charging state diagram in Figure 18. These are the states and their description:

- CHARGE START: in this state the charging will begin, and a timer will start. The LED will be blue to indicate charging. After a set time (default 120sec) or if the voltage of one of the cells reaches the cell overvoltage level (to make sure there is no cell overvoltage error) the state will change to CHARGE WITH CB.
- CHARGE WITH CB: in this state the cell balancing (CB) function will be activated. This function does cell balancing based on the cell voltage and the difference of each cell voltage compared to the lowest cell voltage. When the voltage of one of the cells reaches the cell overvoltage level or the charging current is less than the charge complete current, it will go to the RELAXATION state. The LED will stay blue and will blink if cell balancing is active. Balancing is finished if the cells that are balanced, reach the same voltage as the lowest cell voltage.
- RELAXATION: in this state the power switches are set open, disconnecting the battery from the charger. The battery will relax for the specified relax time (default 300 sec). During this relaxing, the cells can still be balanced since this happens with a low balancing current. At the end of the relaxation period, the system will check whether the balancing is done. If balancing is finished and the highest cell voltage is lower than the cell overvoltage minus the voltage margin, it will return to the CHARGE WITH CB state to continue the charge process. If the highest cell is within this margin, the charging is complete, and it will go to the CHARGE COMPLETE state. To make sure it won't endlessly go through this cycle with the CHARGE WITH CB state (this can happen if the end of charge current is met but

the voltage requirement is not met), after 5 times it will not check if the highest cell voltage is within this margin and will go to the CHARGE COMPLETE state as well.
- CHARGE COMPLETE: in this state, the charging is done and te LED will be steady green. The power switches will remain open and if the charger is disconnected it will go to the SLEEP state after the defined period of time.

If at any time the current flows from the battery to the output and this current is higher than the sleep current, the BMS transitions to the NORMAL mode. If a charger is disconnected, the state will transition to the SLEEP state. If the go to deep sleep command has been given there are two options: If one cell voltage is less than the storage voltage it will complete charging until each cell has reached the storage voltage, after this is done the BMS will transition to the SELF DISCHARGE state and this will transition to the DEEP SLEEP state. The other option is that no cell voltage is less than the storage voltage, than the BMS will transition to the SELF DISCHARGE state.

Figure 8: Charging state diagram



SLEEP state

The sleep state is typically entered when the current is very low for an amount of time. The power switches will be closed to make sure the battery could be used. If any threshold is met during a cyclic measurement or the button is pressed, it will wake the MCU and the BMS will transition to the INIT state to check status. If the button is pressed for five seconds, the state will change to the SELF DISCHARGE state, in order to go to the DEEP SLEEP state. In this state the LED will be off. In a later release, this SLEEP state is used to preserve power.

OCV state

This state is not implemented yet.

FAULT state

The FAULT state is entered when a critical fault requires the battery switches to be opened has been detected (over-current, over-voltage, cell over-temperature). But only in extreme cases, to prevent sudden power outage on devices like flying drones. To exit this FAULT state the user can manually force the BMS to go to the INIT state via the reset fault command with the CLI or by activating the push button. The LED will blink red in this state.

SELF DISCHARGE state

This state is used to discharge the cells to the cell storage voltage in order to improve its life duration, when storing the battery for long time. In this mode, the power switches are open, the MCU is on and the CB function is activated. When the storage voltage is reached for each cell or if cells have a lower voltage, it will transition to the DEEP SLEEP state. CAN communication is disabled. The LED will blink blue in this state. To exit this state and to go back to the INIT state, the button needs to be pressed.

DEEP SLEEP state

This state is used for transportation and storage. In this state; the power switches are open, disconnecting the battery, all protections are turned off, there are no cyclic measurements done, the LED is off, and it will set everything to sleep or off to ensure the lowest power usage (<100uA). Only the button can wake everything in this state. When the button is pressed it will transition to the INIT state. If configuration parameters have changed, it will save the parameters to flash to make sure they are loaded at startup.

10.4 Realization

This chapter will show how the realization has been done. It will use diagrams to show how some of the parts were designed and it will describe each part in more detail.

10.4.1 Main

In the main source file, the BMS main can be found, this is the BMS application. This function will initialize each part and start the main loop task. This task will implement the battery main state machine. In the main source code, the state is changed. If for example flight mode is enabled, when the drone is in the air, and there is an undervoltage, the state should not cut the power. Meaning that the state will not transition to the fault mode, but only report the fault in the status flags. In this source file there is a function to handle a changed parameter as well. This function will call functions from the needed parts to do something with the parameter that is changed. If for example a configuration changed, such that a configuration of the BCC needs to change, it will call the right function from the Bat management part to change the configuration of the BCC as well.

10.4.2 Data

There is a lot of data that is needed or set by different tasks. Because it is not wise to move this big chunk of data through all the tasks there needs to be some sort of shared memory. Because NuttX is POSIX compliance there are shared memory functions that could be used. But for these shared memory functions a memory management unit (MMU) is needed and this microcontroller does not have an MMU. That is why the whole data management will be made in a data source file. This makes sure the data is only made once but is not global. With functions the data can be read or written, and these functions ensures protection against multiple threads accessing the data at the same time. These functions can be seen in Figure 10 and Figure 9.

To protect the data from multiple threads trying to access it at the same time, a mutex is used. A mutex is an object that can be locked and unlocked in an atomic operation. Meaning that if both threads want to lock the mutex, the threads cannot lock the same mutex at the same time. A mutex is needed to prevent data race. The other thread needs to wait until it is available.

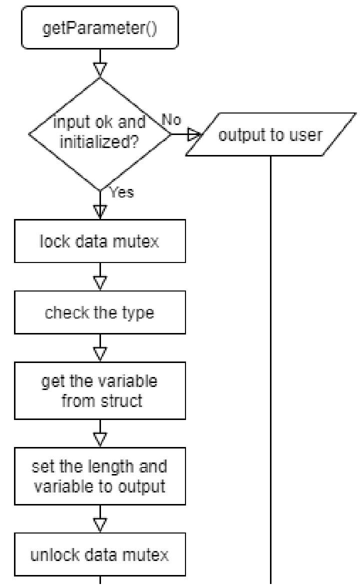


Figure 9: Get parameter flowchart

The big data chuck is in a struct, together with a parameter info array. This array supports a fast access of the data type, the minimum, the maximum and the address of the data. This ensures it is faster to get and set data than with a switch.

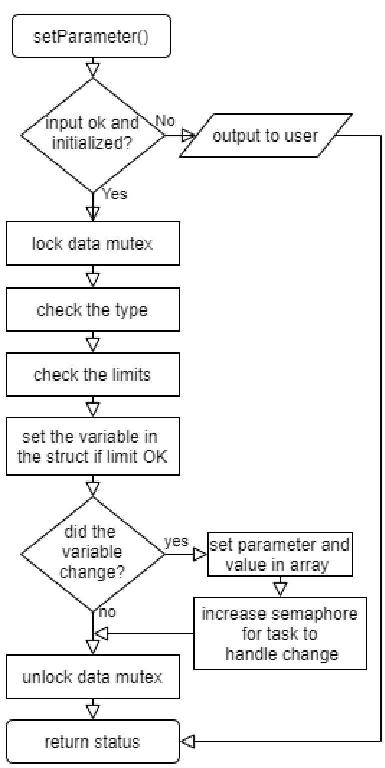


Figure 10: Set parameter flowchart

If a variable is changed with the set parameter function, that function will set that variable and value in a global array. So, the task can access it. And it will increase a semaphore for the task to handle the change. This is done in a callback function to the main. This task is waiting for an available semaphore. A semaphore is an integer variable that can be increased and decreased in an atomic operation. It looks like a mutex because the thread will wait if the semaphore is not positive and tries to decrease it if positive.

10.4.3 CLI

In NuttX there is a nuttshell, this is the UART communication with the MCU. In this nuttshell, applications can be called with and without arguments. There arguments will be given to the function it calls, in this case the BMS main. This means that a CLI can be created with calling the application with some arguments.

This CLI that is made, can be used by calling the BMS application in the nuttshell with a command and optionally 2 arguments for that command. When this happens the BMS main is called. Meaning that this main needs to be resistant against multiple calls, this should not restart the BMS application because than the battery power will be cut.

If there are commands given when calling the BMS application, the CLI process commands function will be called to handle it. It will parse the command and optionally the arguments and check if the inputs are valid. If it is valid, it will act based on which command has been given. The flowchart can be seen in Figure 11.

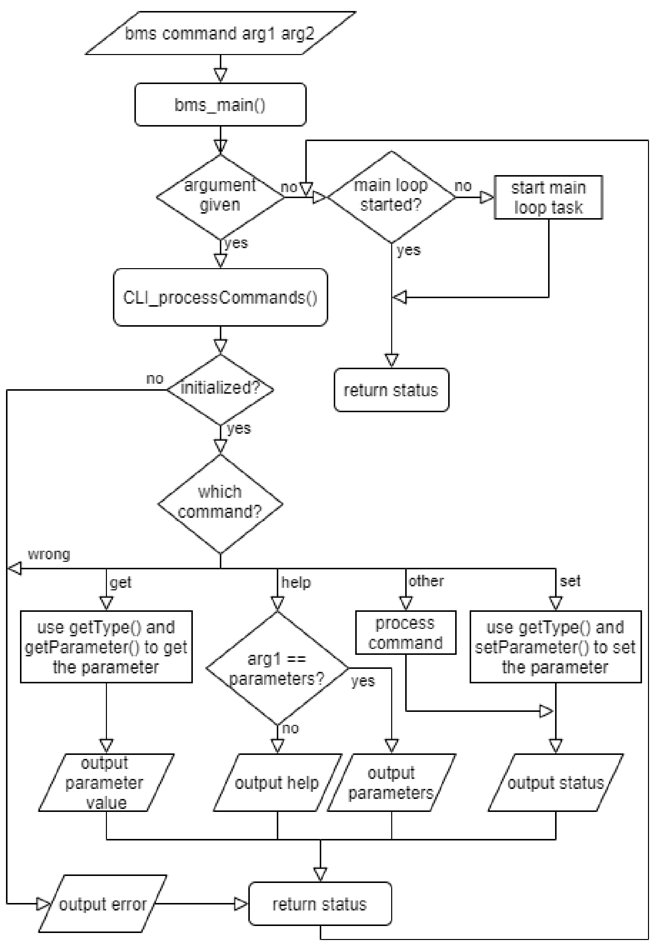


Figure 11: CLI flowchart

10.4.4 LED state

In order to set a color to the RGB LED, the set led color function should be used. The flowchart of this function can be found in Figure 12. Because this function can be used from different tasks, a mutex will be locked before it checks if the color and if blink is already set. This function will set the semaphore to start or stop the blink sequence. It will skip the semaphore timed wait function to ensure the blink sequence restarts if needed. It will begin with the new color. This function will use the NuttX userled functions.

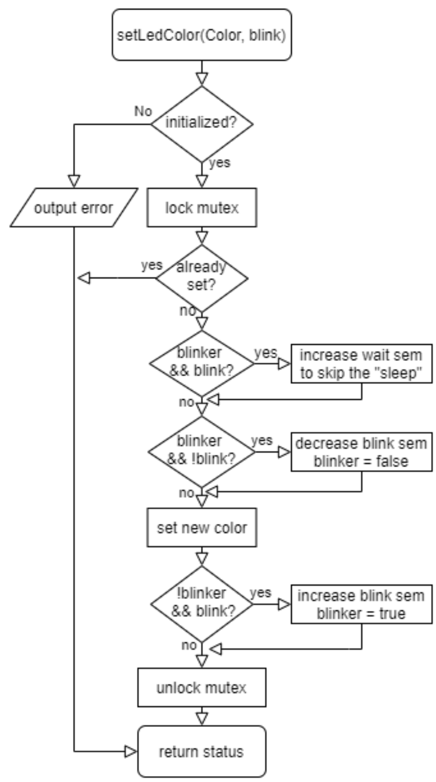


Figure 12: Set LED color flowchart

10.4.5 GPIO

In order to use a GPIO in NuttX, these GPIO's need to be defined in the board file and the board specific GPIO file. This will create devices for each GPIO pin. To use the GPIO in the application an IOCTL call needs to be used. IOCTL means input-output control and it is a device specific system call (wallyk & inductiveload, n.d.).

The IOCTL is used to give commands to a driver to control a device, in this case the GPIO pins. But for an IOCTL to work an open file descriptor needs to be given (Linux man-pages, n.d.). This is obtained by giving the path to the device as a string. This is too much work to do in the application for setting or reading a GPIO, that is why a GPIO BMS application driver is made. This will make sure that a GPIO can be read or written with simple write/read pin functions and a define to indicate the pin.

10.4.6 Bat management

The Bat management part can be used to monitor the battery and control the gate. Because the Bat management part is quite large there are other source files made to help with the BCC, to keep it organized. Like monitoring, to take care of measurements and configuration, to take care of the whole configuration for the BCC. For the main to implement the state machine, functions are made to let the main implement the functionality. Some functions are made to enable the measurements, to check for faults, to self-discharge etc.

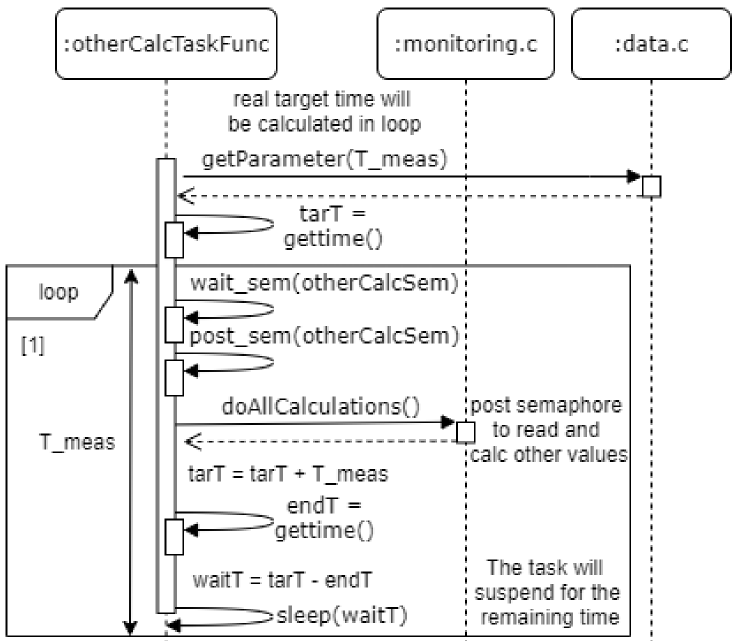


Figure 13: Calculate other values sequence diagram

The meas task is created to do the manual measurements with the BCC and calculate the variables. Because the BCC will not check for an overcurrent, the current needs to be read and calculated every time continuously to be compared with the current threshold. For short circuit protection there is a hardware circuit. The rest of the measurements will only be read and calculated if a semaphore is available. An extra task called otherCalc task is created to increase this semaphore each time the other measurement data is required to be known. This is needed at period T_meas. If the semaphore is increased, the meas task will decrease the semaphore, read the other registers and calculate battery voltage, battery current, cell voltages, the temperatures, remaining charge, the average power and set them. Then it will signal back to the main that the measurements are done. The sequence of the meas task can be seen in Figure 14. The sem_wait and sem_post functions are called consecutively in the endless loop, this is used to start and stop the task with a function from the main.

After the semaphore is increased, the otherCalc task will suspend for the remaining time. This can be seen in Figure 13. Gettime() returns the time from the start of the whole application. To make sure the cyclic measurements does not drift, the time before the loop starts and the period T_meas are gained. The target time is calculated by adding the period with the previous target time. If T_meas should change, this is updated in the sequence using a global variable. This is left out of the sequence diagram because it is too detailed. If the semaphore is increased, the end time is gained. The difference of the target time and the end time give the time that the task needs to sleep.

To calculate the state of charge, the coulomb counter is used. The coulomb counter register holds the sum of the measured currents (until read). There is another register that

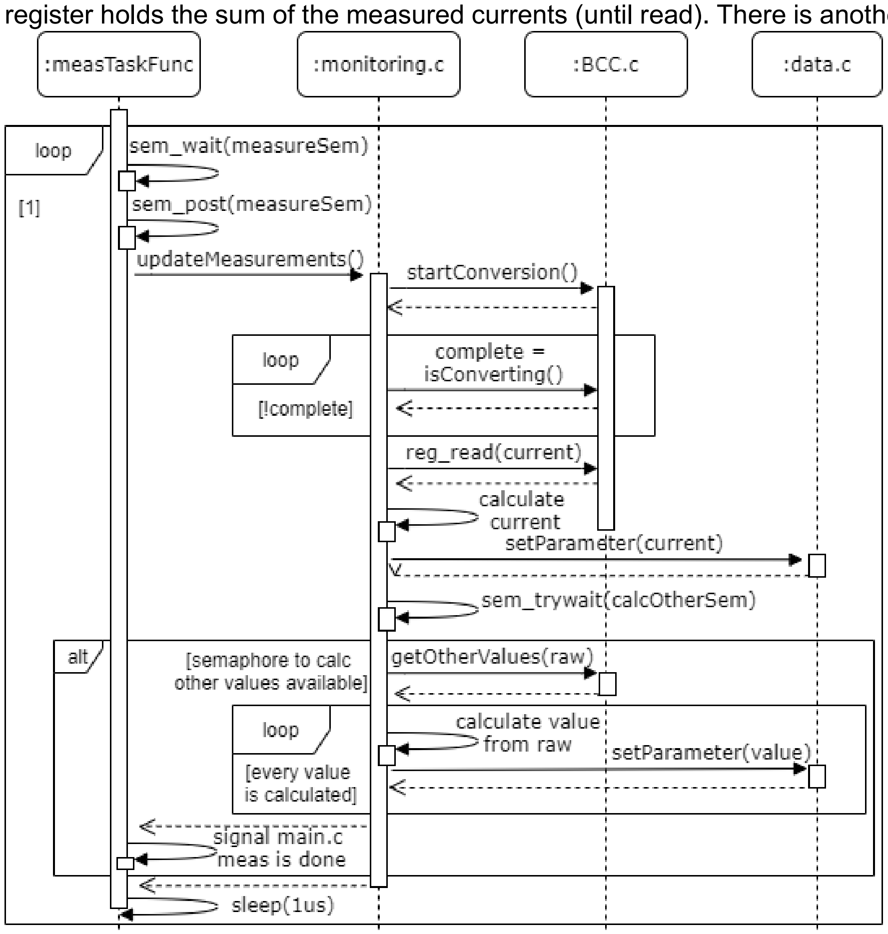


Figure 14: Update measurement sequence diagram

holds the number of samples in the coulomb counter register. The average current is calculated by dividing the sum of the currents by the number of samples. When the time is known for which the average current is calculated, the difference in charge can be calculated with the following formula: $\Delta Q = I_{avg} * \Delta t$. The new remaining charge is calculated by adding the difference in charge with the old remaining charge. The state of charge can then be calculated by dividing the FCC by the remaining charge.

In order to provide the average power consumption over a time period of ten seconds, a constant moving average is taken. This moving average is constructed by removing the oldest measurement and adding the new measurement, which is than divided by the amount of measurements. This way the average will only be of the last ten seconds. In order to be memory efficient, the measurements used in the moving average will be sub-sampled if the measurement period is configured as less than one second. This way maximum ten old measurements need to be known. Measurements are not lost when sub-sampling, because the BCC chip will remember an average of it.

The BCC chip will take care of fault monitoring for the overvoltage, undervoltage, over temperature and under temperature. It will set the fault pin high when there is an error. If this happens it will trigger an interrupt in the main and it will check what fault happened. The main can than act on the fault. This ensures that the main is in control of what happens.

Since the user can change configurations in run time, sometimes a configuration needs to be changed in the BCC as well. When there is a change in the configuration, this is set with the setParameter function and a task in the main source file will handle the change. This function will call a function to handle the change in the bat management part. In this part it will call the right function from the configuration source file to change the configuration of the BCC.

Since the charging state machine and the main state machine is implemented in the main, but it needs information that is from the bat management part, a callback function will be used to give this information to the main if needed. This way the task to implement the state machine is not constant polling for information but will react if the information changes. This will ensure that this task is not always active, and the resources are used for other tasks.

10.4.7 SBC

The SBC part is used to control the power of voltage regulators V1 (The most used 3.3V) and V2 (CAN PHY). With the setSbcMode() function the mode of the SBC can be set. In the normal mode both V1 and V2 are active, in the standby mode V2 is off, turning off the CAN transceiver and in the sleep mode both V1 and V2 are off, turning off almost the whole BMS board. In Figure 15 the simplified flowchart of this function can be seen.

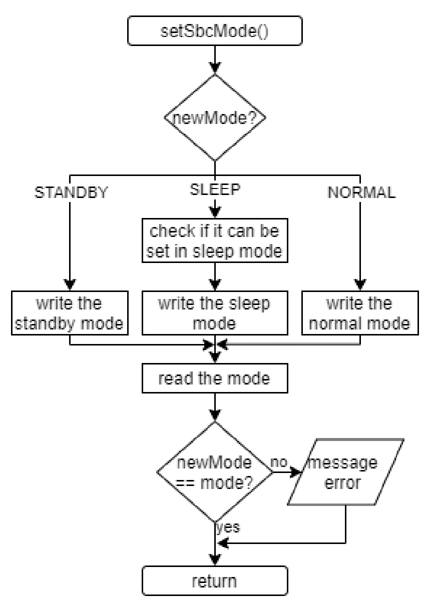


Figure 15: Set SBC mode flowchart

10.4.8 UAVCAN

In the beginning of the project everything was designed towards UAVCAN V0 (UAVCAN, n.d.). Later in the project it was clear that UAVCAN V0 will not work in NuttX. But there was a new version of the UAVCAN protocol, version one (V1). Within NXP, a solution has been made to make the UAVCAN V1 protocol in NuttX. In this new version, the battery info standard as stated in V0 was not specified (UAVCAN, 2020). This is a problem since the BMS should eventually communicate over UAVCAN. And since more companies were interested in a battery info standard. A draft standard has been made. This standard has been proposed to a company that is working together with NXP and other companies to make UAVCAN V1 standards for drones. This standard is still being developed. Because the company would like to see an example working with UAVCAN, a snapshot of the draft protocol was taken, and this has been implemented with the BMSStatus message. Unlike the V0 variant, this message doesn't include SoH, FCC and hours to full charge. This part works with a UAVCAN task that waits (it sleeps until a CAN transceiver signal comes in) for an incoming UAVCAN transmission or a signal from the main that new data needs to be send. When new data needs to be sent, it will put the data that needs to be sent in the transmit buffer. It will check if the transmit buffer if it is filled and transmit the data if it is. Then it will wait for an incoming transmission again. To see this message or the flowchart see Figure 15: UAVCAN flowchart BMSStatus UAVCAN message
and Table 1: BMSStatus UAVCAN message status values.

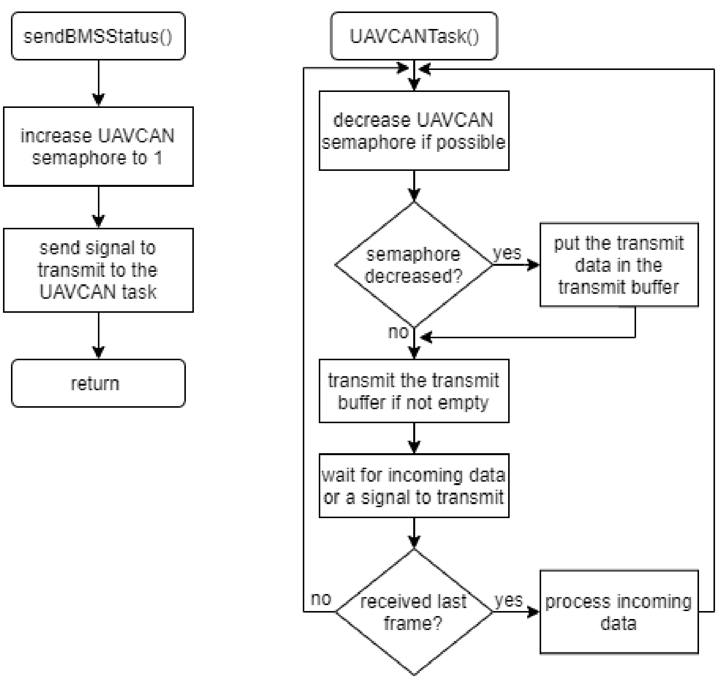


Figure 16: UAVCAN flowchart

Table 8. BMSStatus UAVCAN message

Type	Bits	Name	Def.	Unit	Description
uint7	7	state_of_charge	0	%	Percentage of the full charge [0..100]
bool	1	output_status	0	-	0 = battery output disabled 1 = battery output enabled
float16	16	temperature	NaN	C	Battery temperature
float16	16	voltage	NaN	V	Battery pack voltage
float16	16	current	NaN	A	Last measured current
float16	16	energy_consumed	NaN	Wh	Consumed power since boot
uint8	8	battery_id	0	-	Battery ID
regulated.drone.sensor. BMSStatusValue.1.0	8	status	255	-	Status bitmask. STATUS_UKNOWN = 255

Table 9. BMSStatus UAVCAN message status value

Bitmask Type	Name	Value	Description
uint8	STATUS_ASK_PARS	1	There is a change in the extra parameters so these should be asked
	STATUS_TEMP_ERROR	2	Battery temperature limit failure, the temperature is either too high or too low
	STATUS_OVERLOAD	4	Safe operating area violation, the controller should look at drawing less current
	STATUS_BAD_BATTERY	8	This battery should not be used anymore (e.g. low SoH)
	STATUS_NEED_SERVICE	16	This battery requires maintenance (e.g. balancing, full recharge)
	STATUS_BMS_ERROR	32	Battery management system/controller error, smart battery interface error
	STATUS_OPTIONAL1	64	To be applied to another status
	STATUS_OPTIONAL2	128	To be applied to another status
	STATUS_UKNOWN	255	When the status is unknown

11. Legal information

11.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

11.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

11.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

11.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

11.5 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

<Name> — is a trademark of NXP Semiconductors N.V.

12. Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.

For more information, please visit: http://www.nxp.com

Date of release: Sept 11, 2020
Document identifier: Release Notes