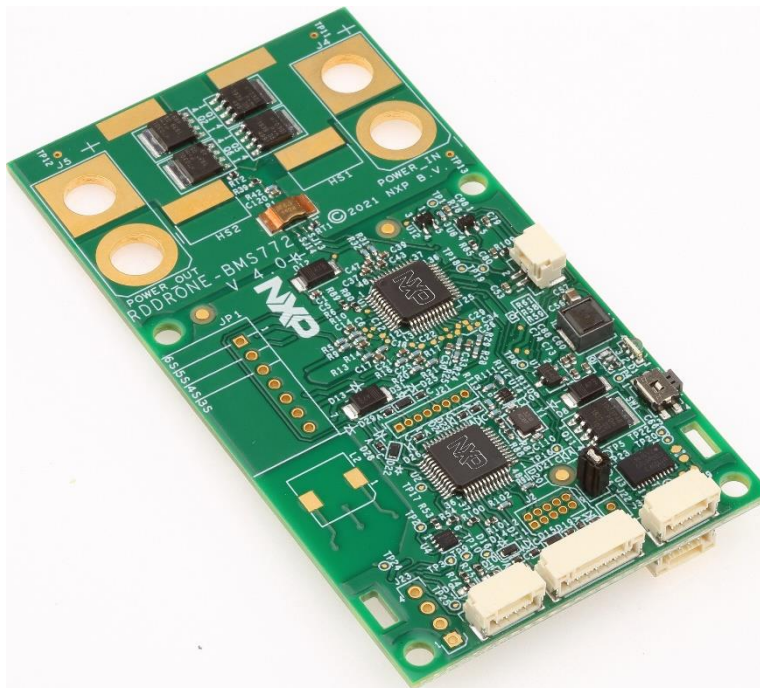


BMS772 software

Release Notes: v6.0 Release

Rev. 6.0 — Feb 24, 2023

Release Notes



Document information

Info	Content
Keywords	BMS772, RDDRONE, S32K144, UJA1169, MC33772, Mobile Robotics
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.
3.6	20210125	Updated to the new release, added Changes table, added some variables, modified state diagrams, adjusted the description of the states, modified the UAVCAN table and added the self-test and timed based and voltage based balancing.
3.7	20210712	Updated to the new release, changed state diagrams, adjusted state description. Added SMBus, added NFC, added watchdog, added task priorities, changed the software block diagram.
4.0	20210830	Processed comments from GK, added correct UAVCAN information, changed pictures and diagrams.
5.0	20211209	Added display and the NFC fix.
6.0	20230224	Combined tasks, code cleanup, upgraded to new NuttX RTOS version: 11.0+. Added display 5V support, added DroneCAN support for communicating information and controlling the BMS parameters.

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

3.1 Changes relative to the last 5.0 release

Table 1. Changes 6.0

Item	Description
tasks	Combined the meas, otherCalc, sdChar changedDataHandler tasks in a new batManag task, changed the task priorities picture.
OS	Updated to new NuttX 11.0 version.
Updater parameters	The updater task will now first copy the 2 structs and use that data to update over all the communication ways.
Bms default	This will now wake up the BMS in sleep state before writing the default values.
Fault state	Split the FAULT state in FAULT_ON and FAULT_OFF (as it might be that the output is kept on).
Code style	Used a .clang-format file to format the adjusted code.
Display	Added 5V support for the display.
DroneCAN	Added DroneCAN support, both the BMS messages and a way to control the BMS, with setting parameters, saving, rebooting, etc.
parameters	<p>Added: gate-chaeck-enable, dronecan-node-static-id, dronecan-bat-continueus, dronecan-bat-periodic, dronecan-bat-cells, dronecan-bat-info, dronecan-bat-info-aux, can-mode, f-v-out-divider-factor.</p> <p>Changed: Changed the order of all parameters. Changed parameter name t-cyclic to t-sleep-bcc-cyclic. t-full is 0 when not charging. Changed UAVCAN to Cyphal: uavcan-node-static-id to cyphal-node-static-id, uavcan-es-sub-id to cyphal-es-sub-id, uavcan-bs-sub-id to cyphal-bs-sub-id, uavcan-bp-sub-id to cyphal-bp-sub-id, uavcan-legacy-bi-sub-id to cyphal-legacy-bi-sub-id. Changed uavcan-fd-mode to can-fd-mode, uavcan-bitrate to can-bitrate and uavcan-fd-bitrate to can-fd-bitrate.</p> <p>Removed: s-charge-stdev.</p>

3.2 Changes of 5.0 relative to 4.0

Table 2. Changes 5.0

Item	Description
NFC	Added the NFC bug fix that works on all batches.
Display	Added LCD/OLED to display BMS information.

3.3 Changes of 4.0 relative to 3.6

Table 3. Changes 4.0

Item	Description
Power	<p>Implemented the sleep state to be low power (VLPR mode).</p> <p>Added a lower power state during the charge-relaxation mode to not discharge the cells too much.</p>
NFC	Enabled NFC to read out BMS information using the NTAG5.
SMBus	SMBus can be enabled to retrieve BMS information using the I ² C slave interface.
Watchdog	Added the watchdog on the SBC (UJA1169TK/F/3).
Task priority	Added task priorities.

Item	Description
Time	Added the "bms time" command.
Charge end current	Added a variable to enter the system current to subtract this from the measured current during charging.
Measurements in fault	Enabled the measurements in the fault state.
Emergency button	Enabled the option to have an emergency button to disconnect the power.
Self-test watchdog	Added a self-test to check if the watchdog jumper is mounted.
CLI	Added the BMS state, added that the top command enables/disables the measurements on the CLI, re-arranged the output.
Cell OV TH	Made sure that the BCC OV TH is at least the set OV (8 bit register). Added a software cell OV TH to check for the actual threshold using the cell voltage measurements.
Updater task	Added an updater task to take care of updating the UAVCAN, SMBus, CLI and the NFC.
Charge re-charge	Added that if the voltage drop is too much again after charge-complete, it will initiate a re-charge.
Flight mode	The s-in-flight parameter will be used to determine if the power should be cut off or not during a fault. A peak overcurrent will always cut off the power of the battery. the flight mode parameter now is saved, and read during startup. The way the s-in-flight is determined is different than the previous flight mode.
Current check	i-batt is now used for i-peak-max and i-batt-avg is used for the i-out-max and the i-charge-max.
UAVCAN	Implemented the UAVCAN V1 0.2 battery service message (source state, battery status and battery parameters). Added a way to get the UAVCAN messages on a terminal. Added the UAVCAN V0 legacy message.
Parameters	<p>Changed uavcan-ess-sub-id to uavcan-es-sub-id . Changed the default value for the subject ids (uavcan-es-sub-id, uavcan-bs-sub-id and uavcan-bp-sub-id) from 65535 to 4096, 4097 and 4098, i-peak to i-range-max.</p> <p>Added: i-batt-10s-avg, s-in-flight, v-cell-nominal, i-system, i-peak-max, v-recharge-margin, emergency-button-enable, smbus-enable, uavcan-legacy-bi-sub-id</p>

3.4 Changes of 3.6 relative to 3.4

Table 4. Changes 3.6

Item	Description
Power on self-test	More components are tested at startup (NTAG5, A1007), with better output on terminal.
NuttX version	Upgraded to NuttX version 10.0 (stable release).
Sense resistor test	Added a test that will detect if the sense resistor is connected to the measurement input.
Negative input for unsigned variables	The CLI will notify you and will not set the variable when a negative number is entered for an unsigned variable.
Self-test state	The self-test is only done at startup, made it a separate state (not only init state anymore).
LED color	LED is now solid red in the self-test state.
SoC calibration	<p>Added the OCV state to calibrate the state of charge when in sleep mode.</p> <p>Added the SoC calibration to the self-discharge state.</p>
CLI set parameter	Fixed that using the CLI, variables can't be set with more than its variable type can hold.
UV fault to deepsleep	Added that when an undervoltage occurs, it will go to the deepsleep mode after some time to protect the battery.
Reboot	Added the reboot command to the CLI to reboot the microcontroller.

Item	Description
Flight-mode	Added the flight-mode-enable and i-flight-mode parameters. This can be used to not cut the power to the FMU and motors in flight.
Modified a-factory parameter	After setting the factory capacity, the BMS will recalculate and set the full charge capacity and end of charge current as well.
Floating point variables	The floating point variables are better limited now.
Charge to deepsleep	Added that if the button is hold for five seconds in the charge state, it will go to deepsleep.
NFC	Is hard-powered down with GPIO.
Discharge to storage	After a long time-out, the BMS will discharge to storage level and go to the deepsleep state.
CLI syntax	Some wrong syntaxes are now fixed in the CLI.
CLI help messages	Improved the help message if the wrong number of cells are attached/inserted.
Wrong messages	The wrong BMS under-temperature detected message if the sensor is disabled has been fixed. The first "pin rising edge BCC_FAULT" message has been deleted.
Message limit	Limited the number of "Rising edge BCC_FAULT" and "clearing CC overflow" messages.
CLI color messages	Added functions for the green (good), yellow (warning) and red (error) messages.
BCC com errors	This will not be reset (wrongly), but it will be counted and with 255 errors it will reset it correctly.
UAVCAN	New draft of the UAVCAN V1 standard message is implemented.
Data struct	Added the unit and type string to the data struct. Added floating point accuracy to the floating point limitations.
Parameters	Added the t-sleep-timeout, i-charge-nominal, i-out-nominal, i-flight-mode, battery-type, flight-mode-enable and m-mass. Changed the model-id to uint64_t, t-fault-timeout to use with uv to go to the deepsleep state, t-ocv-cyclic0 and t-ocv-cyclic1 are implemented, changed the uavcan-subject-id to 3 different parameters for each message: uavcan-ess-sub-id, uavcan-bs-sub-id and uavcan-bp-sub-id. v-cell-margin is default 50mV instead of 30mV. ocv-slope is in mV/Amin instead of V/Amin.
State diagram	Added LED indication, Added SELF TEST state, added button hold to the charge to self-discharge state transition, Added SLEEP_TIMEOUT to the sleep to self-discharge state transition, FAULT_TIMEOUT used for transition to go to deepsleep with an undervoltage.
Charge state diagram	Added LED indication and added button hold to the charge to self-discharge state transition.
Cell balancing	The cell balancing is not only based on the cell voltage compared to the desired voltage, but it is based on the calculated estimated cell balance minutes as well.

4. Limitations

Table 5. Limitations

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

Item	Description
	Impact: A current and voltage limiting power supply needs to be attached to the BMS to charge the battery.

DRAFT / CONFIDENTIAL

5. Known issues

Table 6. Known issues

Item	Description
Cyphal battery status	Applies to: Release Package. Issue: Only the draft of the status message is implemented. Workaround: Update the Cyphal code to implement the standard.
Authentication	Applies to: Release Package. Issue: Isn't implemented. Workaround: Upcoming releases.
State of charge calibration	Applies to: Release Package. Issue: The state of charge calibration table can be different for each battery. Workaround: Add more precise dedicated calibration table of your own used battery.
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:

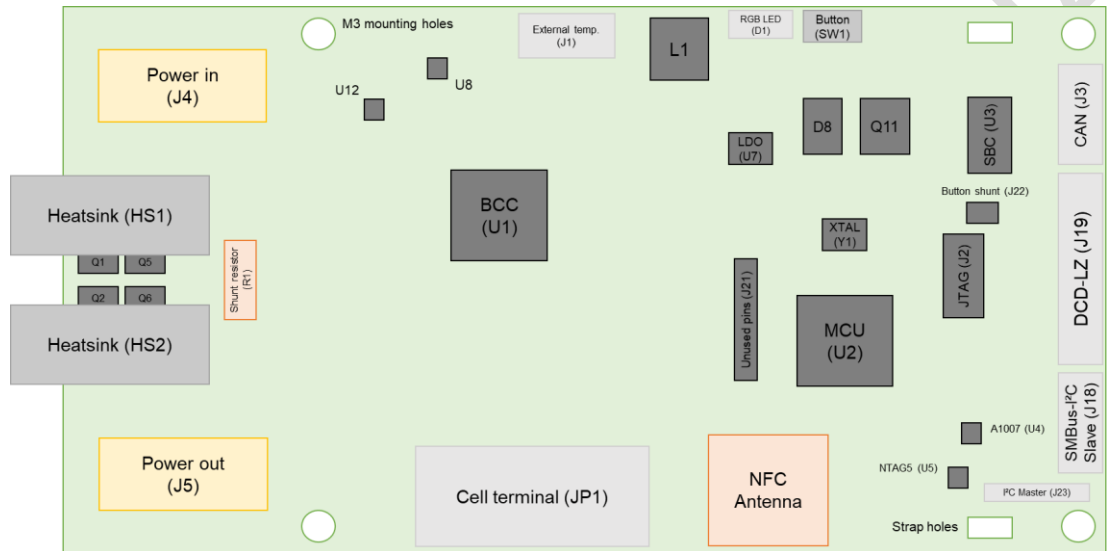


Figure 2 Board block layout -- Top

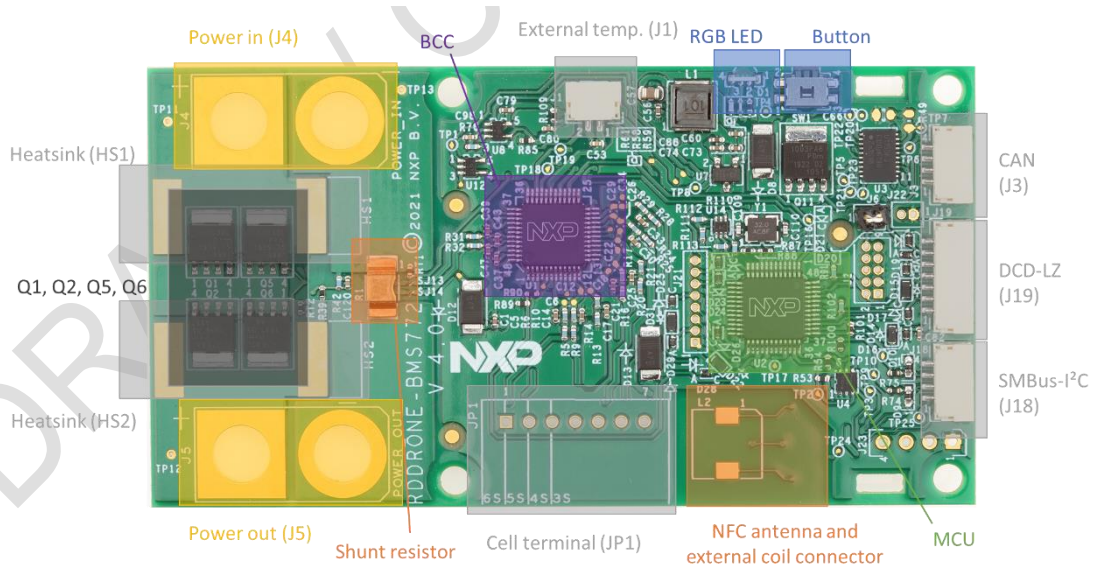


Figure 1 Board map -- Top

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.

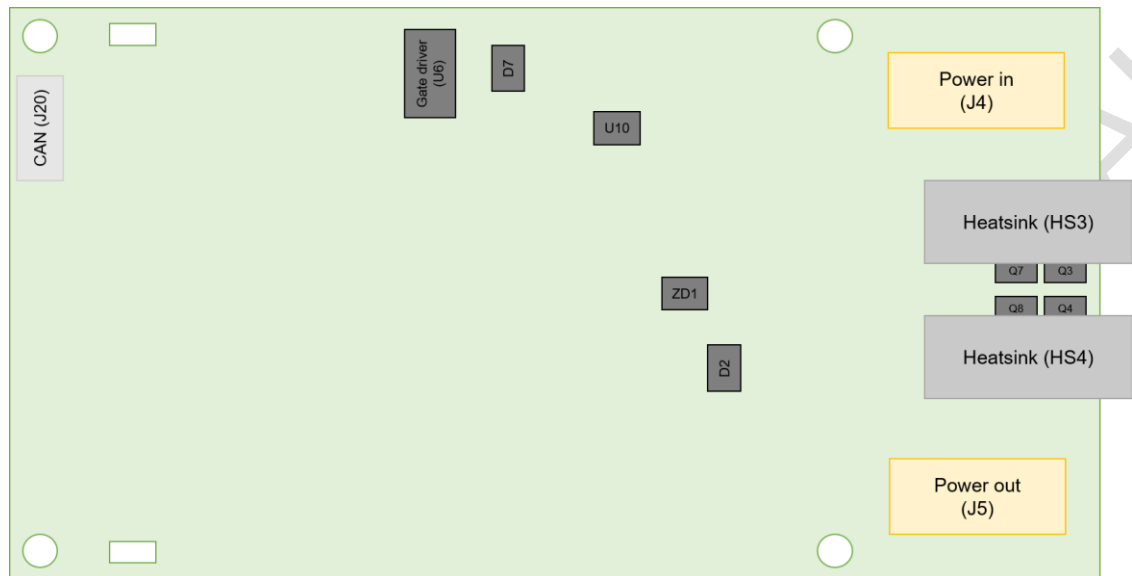


Figure 3 Board block layout -- Bottom

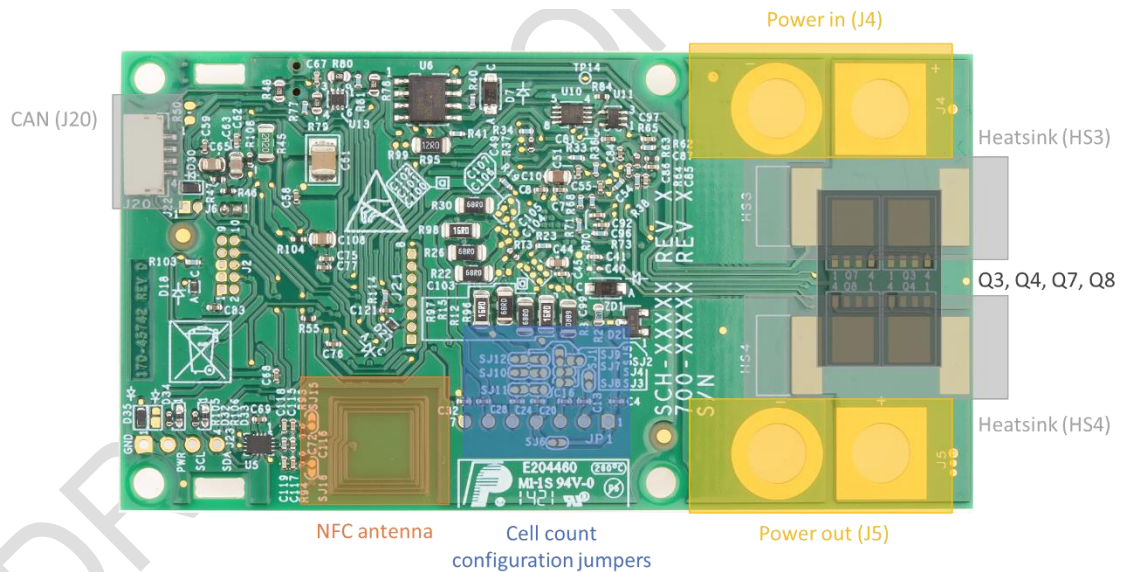


Figure 4 Board map -- Bottom

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 [Figure 5](#) 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.

note: If you are using the JLink commander IDE, don't forget to stop the core and erase the chip.

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 DroneCAN interface

Connect the DroneCAN device (like the RDDRONE-UCANS32K146, an FMU or a PCAN-USB) 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.

More information on DroneCAN can be found at: <https://dronecan.github.io/>

7.2.1 How to set up the DroneCAN GUI tool

The DroneCAN GUI tool information and download can be found here:

https://github.com/dronecan/gui_tool

Download and install the DroneCAN GUI tool on a native Linux device. Attach this native Linux device to the BMS via a USB to CAN converter, like PCAN-USB. Check if can0 is present with the Linux command

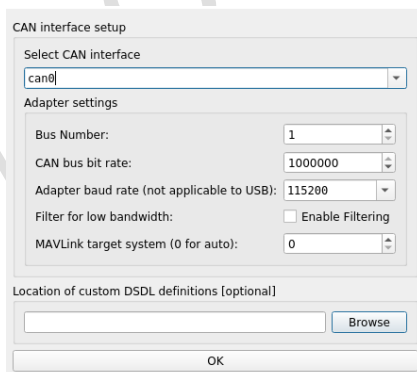
```
ifconfig -a
```

To enable this can device enter the Linux command:

```
sudo ip link set can0 type can bitrate 1000000 dbitrates 4000000 fd on  
sudo ip link set up can0
```

Then start the DroneCAN GUI tool with the following command:

`dronecan_gui_tool` You should be able to see can0 as the CAN interface, like the CAN interface setup screenshot:



Click OK to proceed.

7.2.2 How to use the DroneCAN GUI tool to monitor the DroneCAN messages

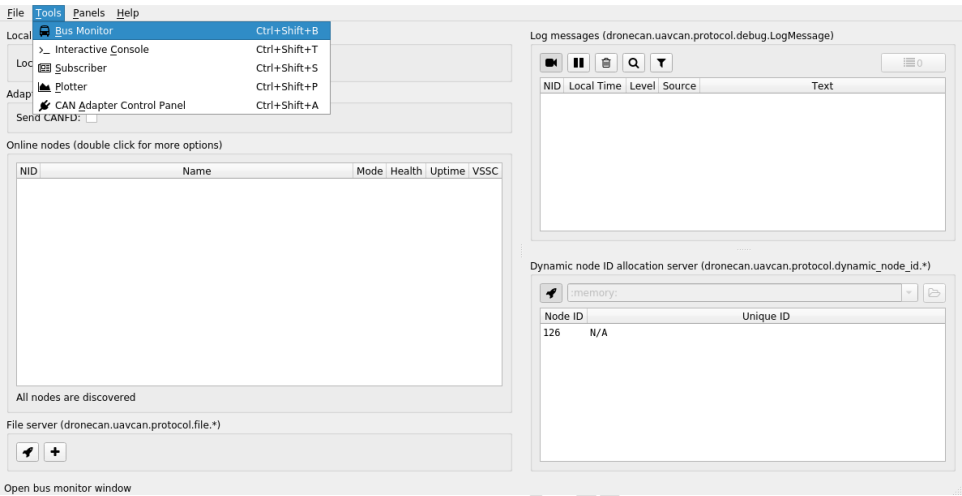
In order to monitor the DroneCAN messages with the DroneCAN GUI tool, make sure you have the DroneCAN DSDL messages.

Clone the following git repository to your device where the DroneCAN GUI tool is running: <https://github.com/dronecan/DSDL>

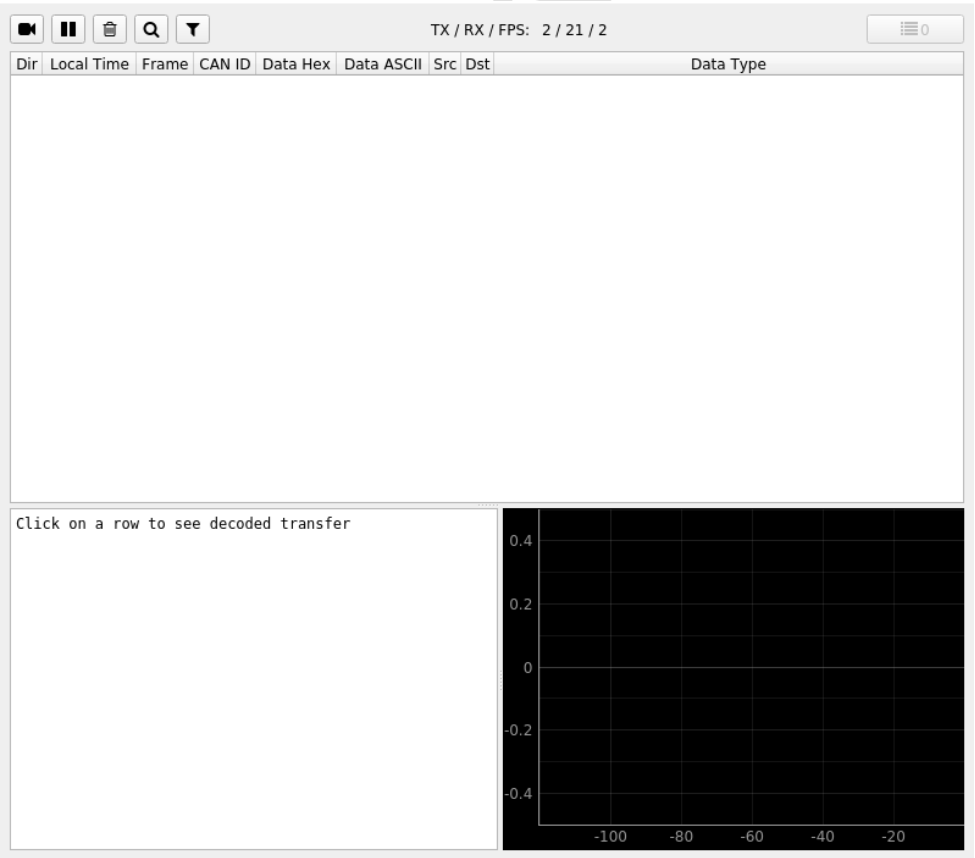
In the CAN interface setup above, click Browse and navigate to DSDL/ardupilot. Set your local node ID (or keep the default), and press right next to it to enable the local node ID.

Keep in mind that the BMS needs to have a node ID allocated via a different instance, see chapter 8.2.

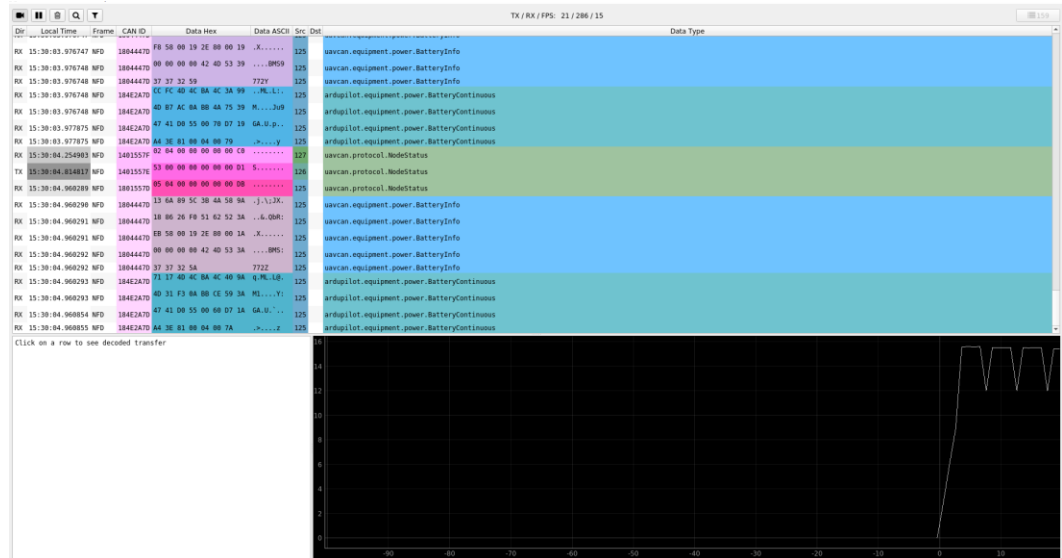
Then go to Tools -> Bus Monitor.
In the Bus Monitor window, click on the recording symbol to start seeing the messages.
Keep in mind that you can filter the messages.



Here the Bus Monitor tool can be seen before the recording is enabled:



Here the Bus Monitor tool can be seen with the recording enabled and multiple messages are shown to be received.



7.3 How to use the CyphalCAN interface

For Cyphal these messages are implemented:

https://github.com/OpenCyphal/public_regulated_data_types/tree/master/reg/udral/service/battery

The recommendation is to use DroneCAN.

7.4 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.

For minicom, start minicom with: "minicom -c on" to enable the colors on the CLI.

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

7.5 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".

7.6 How to use SMBus (I²C slave)

To enable the update, be sure to set `smbus-enable` to 1 with “bms set smbus-enable 1”. If this is enabled, one can use the BMS as an I²C slave device to get the information. Use the J18 connector (I2C/SMBUS) on the BMS, hook up your I²C master device with pull-ups to this connector. The SMBus information is based on the [SBS1.1 specification](#). These are the supported messages:

Table 7. SMBus variable list

Parameter	Unit	Datatype	Description	I ² C Address
temperature	K	uint16_t	The temperature of the external battery temperature sensor, 0 otherwise.	0x08
voltage	mV	uint16_t	The voltage of the battery.	0x09
current	mA	uint16_t	The last recorded current of the battery.	0x0A
average_current	mA	uint16_t	The average current since the last measurement (period t-meas (default 1s)).	0x0B
max_error	%	uint16_t	Just set to 5%. Not tracked.	0x0C
relative_state_of_charge	%	uint16_t	Set to the state of charge value.	0x0D
absolute_state_of_charge	%	uint16_t	Set to the state of charge value.	0x0E
remaining_capacity	mAh	uint16_t	The remaining capacity of the battery.	0x0F
full_charge_capacity	mAh	uint16_t	The full charge capacity of the battery.	0x10
run_time_to_empty	min	uint16_t	Calculated time to empty based on current and remaining_capacity.	0x11
average_time_to_empty	min	uint16_t	Calculated the time to empty based on average_current and remaining_capacity.	0x12
cycle_count	cycle	uint16_t	Set to the n-charges value.	0x17
design_capacity	mAh	uint16_t	Set to the factory capacity.	0x18
design_voltage	mV	uint16_t	Set to the cell overvoltage value.	0x19
manufacture_date	-	uint16_t	Set to the defines in the code. Not actual manufacturer dates. (year-1980)*512 + month*32 + day	0x1B
serial_number	-	uint16_t	Set to the battery id (batt-id).	0x1C
manufacturer_name	-	char *	Set to "NXP".	0x20
device_name	-	char *	Set to "RDDRONE-BMS772"	0x21
device_chemistry	-	char *	This is a 3 letter battery device chemistry "LiP", "LFP" or "LFY" (LiPo, LiFePo4, LiFeYPo4).	0x22
manufacturer_data	-	uint8_t *	Set to 0x0. (length 1).	0x23
cell1_voltage	mV	uint16_t	Cell voltage of cell1.	0x3A
cell2_voltage	mV	uint16_t	Cell voltage of cell2.	0x3B
cell3_voltage	mV	uint16_t	Cell voltage of cell3.	0x3C
cell4_voltage	mV	uint16_t	Cell voltage of cell4.	0x3D
cell5_voltage	mV	uint16_t	Cell voltage of cell5.	0x3E
cell6_voltage	mV	uint16_t	Cell voltage of cell6.	0x3F

7.7 How to use NFC

The BMS has an NTAG5 on board to have NFC communication with an NFC device. In the current example an NDEF text record is implemented. This text record has the actual battery information and is updated each measurement time. If the data is read out via NFC, the new updated data cannot be written to the NTAG at the same time. To read the data with NFC, approach the BMS with a NFC enabled mobile phone. It should automatically pop up with the text message after a read, as can be seen in [Figure 6: NFC read screenshot](#). A NFC read application could be used as well. If the BMS is in a low power state, the NFC is disabled and it will show some information on the state. If the BMS is in the sleep state, an NFC interaction can be used to wake up the BMS.

The following information can be found using an NFC read:

- Output voltage
- Battery current
- State of charge
- State of health
- Output current
- Number of charges
- Battery id
- Model id
- Current BMS state.

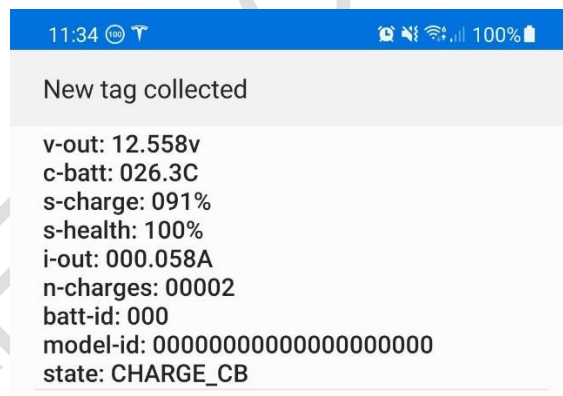


Figure 6: NFC read screenshot

8. The parameters

8.1 How to configure the parameters via the CLI

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**.

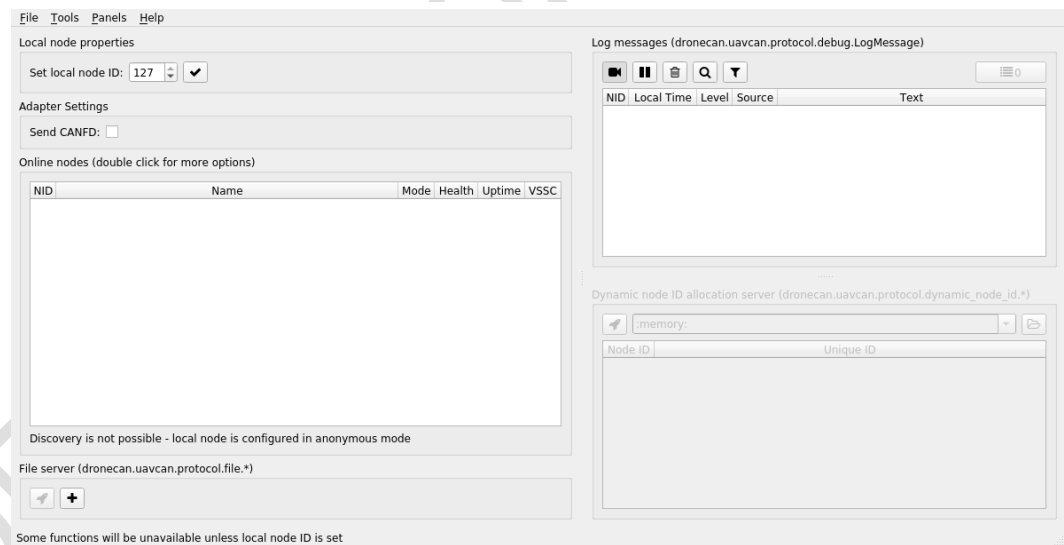
A parameter can be written with a “bms set <parameter> <x>” command in the CLI, where <parameter> is the to be written parameter and <x> is the new parameter value. Keep in mind that that this new value needs to be in the range of the to be written values. For decimals (float) use a “.” to separate it. 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 8.3 to see which parameters need to be configured.

8.2 How to configure the parameters via the DroneCAN GUI tool

After starting the DroneCAN GUI tool and using the correct CAN device.

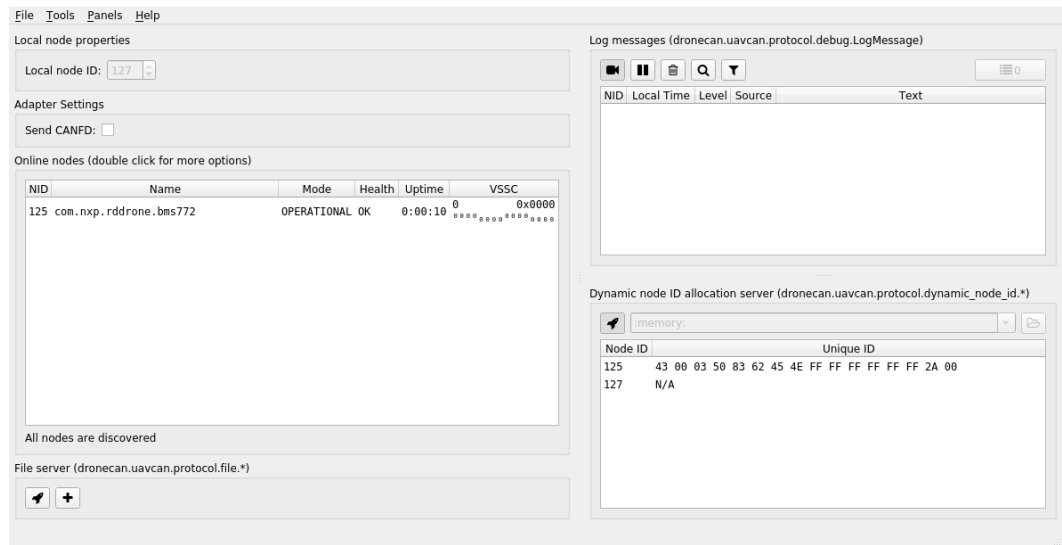
You should see the following screen:



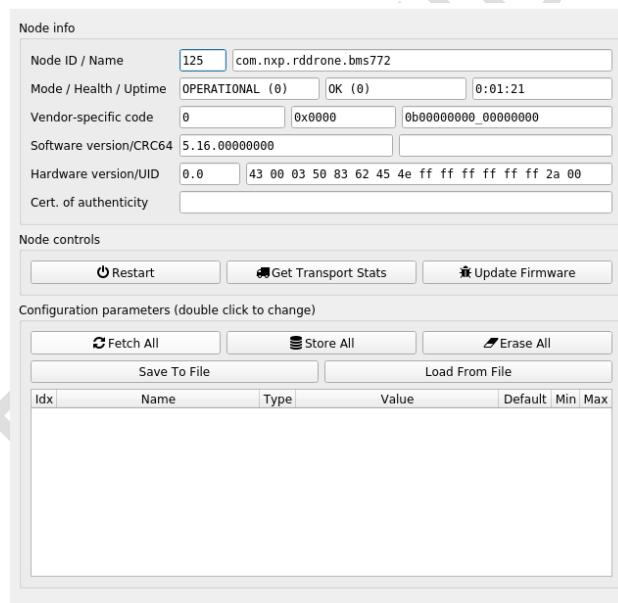
Set your local node ID (or keep the default), and press right next to it to enable the local node ID.

After setting your local node ID, you should be able to click on the rocket in the right bottom of the screen to start the dynamic node ID allocation server.

You should see the following screen:



Double click on the online node, com.nxp.rddrone.bms772 to get the node info. After double clicking the node name, you should get the following view:



Click on "Fetch All" to get all the BMS parameters, plus an extra parameter to reset the BMS fault, called "BMS_RESET_FAULT".

You can now click on the parameters and change them.

The "BMS_RESET_FAULT" parameter is used to reset the BMS when it is in a FAULT state.

"Store All" will save the parameters to flash.

"Erase All" will set all parameters to its default value.

"Restart" will issue the NuttX reboot command and reboot the BMS application.

8.3 The parameter lists

8.3.1 The common battery variables list

Table 8. Common battery variables list

Parameter	Unit	Datatype	Description	Default	RO/ RW	No
v-out	V	float	The voltage of the BMS output	0	RO	0
v-batt	V	float	The voltage of the battery pack	0	RO	1
n-cells	-	uint8_t	Number of cells used in the BMS board	3	RW	2
v-cell1	V	float	The voltage of cell 1	0	RO	3
v-cell2	V	float	The voltage of cell 2	0	RO	4
v-cell3	V	float	The voltage of cell 3	0	RO	5
v-cell4	V	float	The voltage of cell 4	0	RO	6
v-cell5	V	float	The voltage of cell 5	0	RO	7
v-cell6	V	float	The voltage of cell 6	0	RO	8
i-batt	A	float	The last recorded current of the battery	0	RO	9
i-batt-avg	A	float	The average current since the last measurement (period t-meas (default 1s))	0	RO	10
i-batt-10s-avg	A	float	The 10s rolling average current, updated each 1s with t-meas 1000 (ms)	0	RO	11
sensor-enable	-	bool	This variable is used to enable or disable the battery temperature sensor, 0 is disabled, 1 is enabled	0	RW	12
c-batt	C	float	The temperature of the external battery temperature sensor	0	RO	13
c-afe	C	float	The temperature of the analog front end	0	RO	14
c-t	C	float	The temperature of the transistor	0	RO	15
c-r	C	float	The temperature of the sense resistor	0	RO	16

8.3.2 The calculated battery variables list

Table 9. Calculated battery variables list

Parameter	Unit	Datatype	Description	Default	RO/ RW	No
p-avg	W	float	Average power consumption over the last 10 seconds.	0	RO	17
e-used	Wh	float	Power consumption since device boot	0	RO	18
t-full	h	float	Charging is expected to complete in this time; zero if not charging CHECK THIS	0	RO	19
a-rem	Ah	float	Remaining capacity in the battery	0	RW	20
a-full	Ah	float	Full charge capacity, predicted battery capacity when it is fully charged. Falls with aging	4.6	RW	21
a-factory	Ah	float	Battery capacity stated by the factory	4.6	RW	22
s-charge	%	uint8_t	Percentage of the full charge 0, 100.	0	RO	23
s-health	%	uint8_t	Health of the battery in percentage, use STATE_OF_HEALTH_UNKNOWN = 127 if cannot be estimated	127	RO	24
s-out	-	bool	This is true if the output power is enabled	0	RO	25
s-in-flight	-	bool	This is true if the system is in flight (with flight-mode-enable and i-flight-mode)	0	RO	26
batt-id	-	uint8_t	Identifies the battery within this vehicle, 0 - primary battery.	0	RW	27

8.3.3 The additional battery variables list

Table 10. Additional battery variables list

Parameter	Unit	Datatype	Description	Default	RO/ RW	No
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	28
v-cell-uv	V	float	Battery minimum allowed voltage for one cell. Going below this voltage, the BMS will go to fault mode (followed by deepsleep after t-fault-timeout)	3	RW	29
v-cell-nominal	V	float	Battery nominal voltage for one cell. will be used for energy calculation.	3.7	RW	30
v-storage	V	float	The voltage what is specified as storage voltage for a cell	3.8	RW	31
v-cell-margin	mV	uint8_t	Cell voltage charge margin to decide or not to go through another topping charge cycle	50	RW	32
v-recharge-margin	mV	Uint16_t	Cell voltage charge complete margin to decide or not to do a battery re-charge, to keep the cell voltages at max this much difference with the cell-ov	200	RW	33
i-peak-max	A	float	Maximum peak current threshold to open the switch during normal operation, can't be overruled	200	RW	34
i-out-max	A	float	Maximum current threshold to open the switch during normal operation, if not overruled	60	RW	35
i-out-nominal	A	float	Nominal discharge current (informative only)	60	RW	36
i-flight-mode	A	uint8_t	Current threshold to not disable the power in flight mode	5	RW	37
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	38
i-system	mA	uint8_t	Current of the BMS board itself, this is measured (as well) during charging, so this needs to be subtracted	40	RW	39
i-charge-max	A	float	Maximum current threshold to open the switch during charging	4.6	RW	40
i-charge-nominal	A	float	Nominal charge current (informative only)	4.6	RW	41
i-charge-full	mA	uint16_t	Current threshold to detect end of charge sequence	50	RW	42
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	43
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	44
c-pcb-ot	C	float	PCB Ambient temperature over temperature threshold	45	RW	45
c-pcb-ut	C	float	PCB Ambient temperature under temperature threshold	-20	RW	46
c-cell-ot-charge	C	float	Over temperature threshold for the cells during charging. Going over this	40	RW	47

			threshold and the BMS will go to FAULT mode			
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	48
n-charges	-	uint16_t	The number of charges done	0	RW	49
n-charges-full	-	uint16_t	The number of complete charges	0	RW	50
ocv-slope	mV/A. min	float	The slope of the OCV curve. This will be used to calculate the balance time.	5.3	RW	51
battery-type	-	uint8_t	The type of battery attached to it. 0 = LiPo, 1 = LiFePo4, 2 = LiFeYPo4. Could be extended. Will change OV, UV, v-storage, OCV/SoC table if changed runtime.	0	RW	52

8.3.4 The configuration variables list

Table 11. Configuration variables list

Parameter	Unit	Datatype	Description	Default	RO/ RW	No
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	53
t-ftti	ms	uint16_t	Cycle of the battery to perform diagnostics (Fault Tolerant Time Interval)	1000	RW	54
t-bms-timeout	s	uint16_t	Timeout for the BMS to go to SLEEP mode when the battery is not used.	600	RW	55
t-fault-timeout	s	uint16_t	After this timeout, with an undervoltage fault the battery will go to DEEPSLEEP mode to preserve power. 0 sec is disabled.	60	RW	56
t-bcc-sleep-cyclic	s	uint8_t	Wake up cyclic timing of the AFE (after front end) during sleep mode	1	RW	57
t-sleep-timeout	h	uint8_t	When the BMS is in sleep mode for this period it will go to the self-discharge mode, 0 if disabled.	24	RW	58
t-ocv-cyclic0	s	int32_t	OCV measurement cyclic timer start (timer is increase by 50% at each cycle)	300	RW	59
t-ocv-cyclic1	s	int32_t	OCV measurement cyclic timer final value (limit)	86400	RW	60
t-charge-detect	s	uint8_t	During NORMAL mode, if the battery current is positive for more than this time, then the BMS will go to CHARGE mode	1	RW	61
t-cb-delay	s	uint8_t	Time for the cell balancing function to start after entering the CHARGE mode	120	RW	62
t-charge-relax	s	uint16_t	Relaxation time after the charge is complete before going to another charge round.	300	RW	63
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	64

s-flags	-	uint8_t	This contains the status flags as described in BMS_status_flags_t	255	RO	65
self-discharge-enable	-	bool	This variable is used to enable or disable the SELF_DISCHARGE state, 0 is disabled, 1 is enabled	1	RW	66
flight-mode-enable	-	bool	This variable is used to enable or disable flight mode, is used together with i-flight-mode.	0	RW	67
emergency-button-enable	-	bool	This variable is used to enable or disable the emergency button on PTE8.	0	RW	68
smbus-enable	-	bool	This variable is used to enable or disable the SMBus update.	0	RW	69
gate-check-enable	-	bool	This variable is used to enable or disable the gate safety check. If true, it will check if it can be turned off, based on output voltage.	0	RW	70
model-id	-	uint64_t	Model id, set to 0 if not applicable	0	RW	71
model-name	-	char[32]	Battery model name, model name is a human-readable string that could include the vendor, model, chemistry.	"BMS772"	RW	72

8.3.5 The can variables list

Table 12. Can variables list

Parameter	Unit	Datatype	Description	Default	RO/ RW	No
cyphal-node-static-id*	-	uint8_t	This is the node ID of the CYPHAL message	255	RW	73
cyphal-es-sub-id*	-	uint16_t	This is the subject ID of the energy source CYPHAL message (1...100Hz)	4096	RW	74
cyphal-bs-sub-id*	-	uint16_t	This is the subject ID of the battery status CYPHAL message (1Hz)	4097	RW	75
cyphal-bp-sub-id*	-	uint16_t	This is the subject ID of the battery parameters CYPHAL message (0.2Hz)	4098	RW	76
cyphal-legacy-bi-sub-id*	-	uint16_t	This is the subject ID of the battery info legacy CYPHAL message (0.2 ~ 1Hz)	65535	RW	77
dronecan-node-static-id	-	uint8_t	This is the node ID of the DRONECAN node, 255 means dynamic node id.	0	RW	78
dronecan-bat-continuous	-	uint8_t	This is indicate if the particular DroneCAN topic has to be published	0	RW	79
dronecan-bat-periodic	-	uint8_t	This is indicate if the particular DroneCAN topic has to be published	0	RW	80
dronecan-bat-cells	-	uint8_t	This is indicate if the particular DroneCAN topic has to be published	0	RW	81
dronecan-bat-info	-	uint8_t	This is indicate if the particular DroneCAN topic has to be published	1	RW	82
dronecan-bat-info-aux	-	uint8_t	This is indicate if the particular DroneCAN topic has to be published	1	RW	83
can-mode	-	char[32]	Options "DRONECAN" and "CYPHAL". To indicate which is used.	"DRONECAN"	RW	84
can-fd-mode*	-	uint8_t	If true CANFD is used, otherwise classic CAN is used	0	RW	85
can-bitrate*	bit/s	int32_t	The bitrate of classical can or CAN FD arbitration bitrate	1000000	RW	86
can-fd-bitrate*	bit/s	int32_t	The bitrate of CAN FD data bitrate	4000000	RW	87

A line means this is not implemented yet.

* these parameters will only be implemented during startup of the BMS

8.3.6 The hardware parameters

Table 13. 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	88
v-max	V	uint8_t	Maximum stack voltage allowed by the BMS board	26	RW	89
i-range-max	A	uint16_t	Maximum current that can be measured by the BMS board	300	RW	90
i-max	A	uint8_t	Maximum DC current allowed in the BMS board (limited by power dissipation in the MOSFETs)	60	RW	91
i-short	A	uint16_t	Short circuit current threshold (typical: 550A, min: 500A, max: 600A)	500	RW	92
t-short	us	uint8_t	Blanking time for the short circuit detection	20	RW	93
i-bal	mA	uint8_t	Cell balancing current under 4.2V with cell balancing resistors of 82 ohms	50	RW	94
m-mass	kg	float	The total mass of the (smart) battery	0	RW	95
f-v-out-divider-factor	-	float	The factor of the output voltage divider as component tolerances could be different to not result in 11.0.	11.0	RW	96

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.

DRAFT / CONFIDENTIAL

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 bms4.0-10.1. 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 7](#) 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.

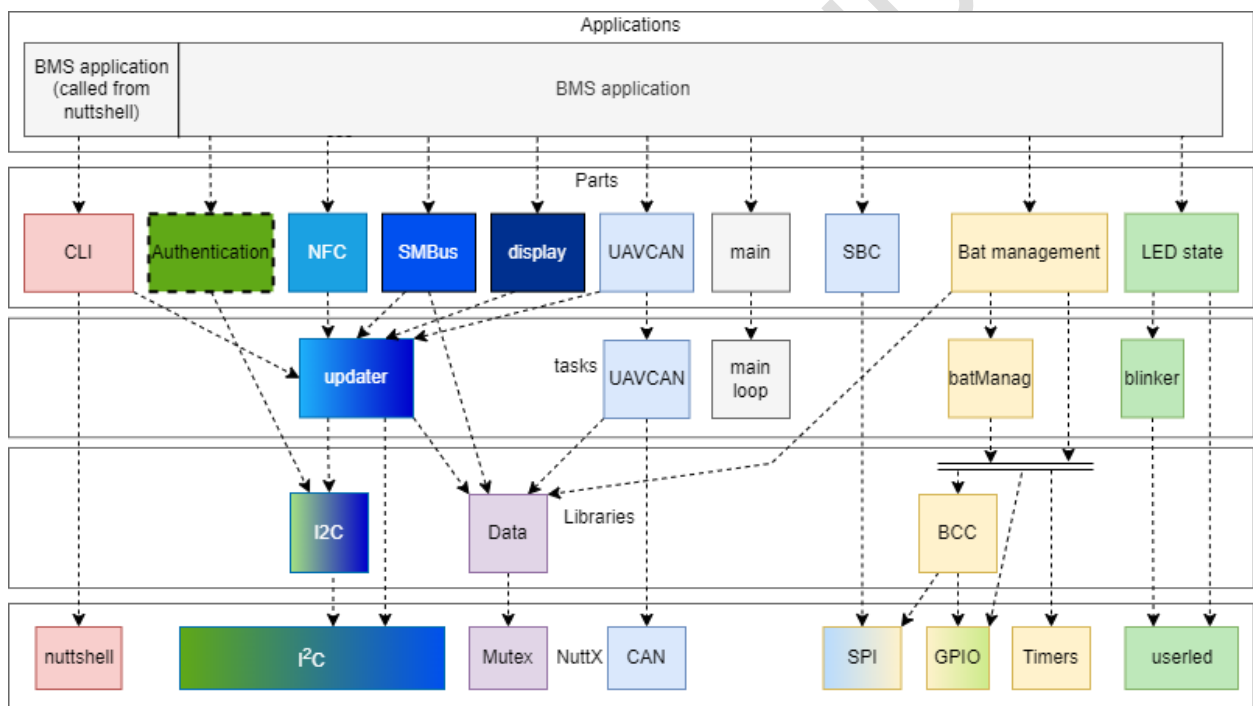


Figure 7: Software block diagram

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 CLI can output messages in colors if the ANSI escape sequences are enabled in the terminal.

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 can be used to read all kind of parameters. It should be able to read the values with a refresh rate of once a second. The updater task will be used to update the data. It can operate in a similar manner to a double ported EEPROM, and NFC records can include standardized messages for HTTP or text records. In this way the NFC tag could be updated regularly with status information. That information could be added to text message, and a smartphone would be capable of reading the message with data attached with minimal coding effort. This method removes the need for any custom software on the reading device.

SMBus

SMBus is an alternative way to communicate information to the FMU. The BMS could be seen as an I²C slave device. Reading from specified registers returns the BMS data. Things like voltages, temperatures, state of charge, average current could all be read from these registers. The SMBus needs to be enabled with the variable in order to work.

Display

The display module manages information presented on an optional local I2C LCD display (e.g. SSD1306 type). The display should be connected to J23. Both 3.3V and 5V are supported in the software. There is a framebuffer which makes sure the data is transferred to the display. Information like SoC, SoH, output status, BMS mode, battery voltage, average current, temperature and ID can be found on the display.

DroneCAN

The DroneCAN module manages DroneCAN communication. The DroneCAN protocol is used to relay battery and power usage to the FMU (or host processor). It sends the battery information on a cyclic time interval. It has a task named DRONECAN 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 and 5V) 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. The SBC provides a watchdog.

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 has its own tasks. This task can access the BCC, the timers and the GPIO. This is the task:

The batManag task will oversee the measurements and if triggered do the calculations, check for errors, check for transition currents and handle the cell balancing / self-discharge.

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 14.

Table 14. LED states

State	LED state
Self-test	Red
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_on (output power on)	Red
Fault_off (output power off)	Red blinking
Charging	Blue
Charging done	Green
Balancing/self-discharge	Blue blinking
Charger connected at startup	Red-blue blinking

Data

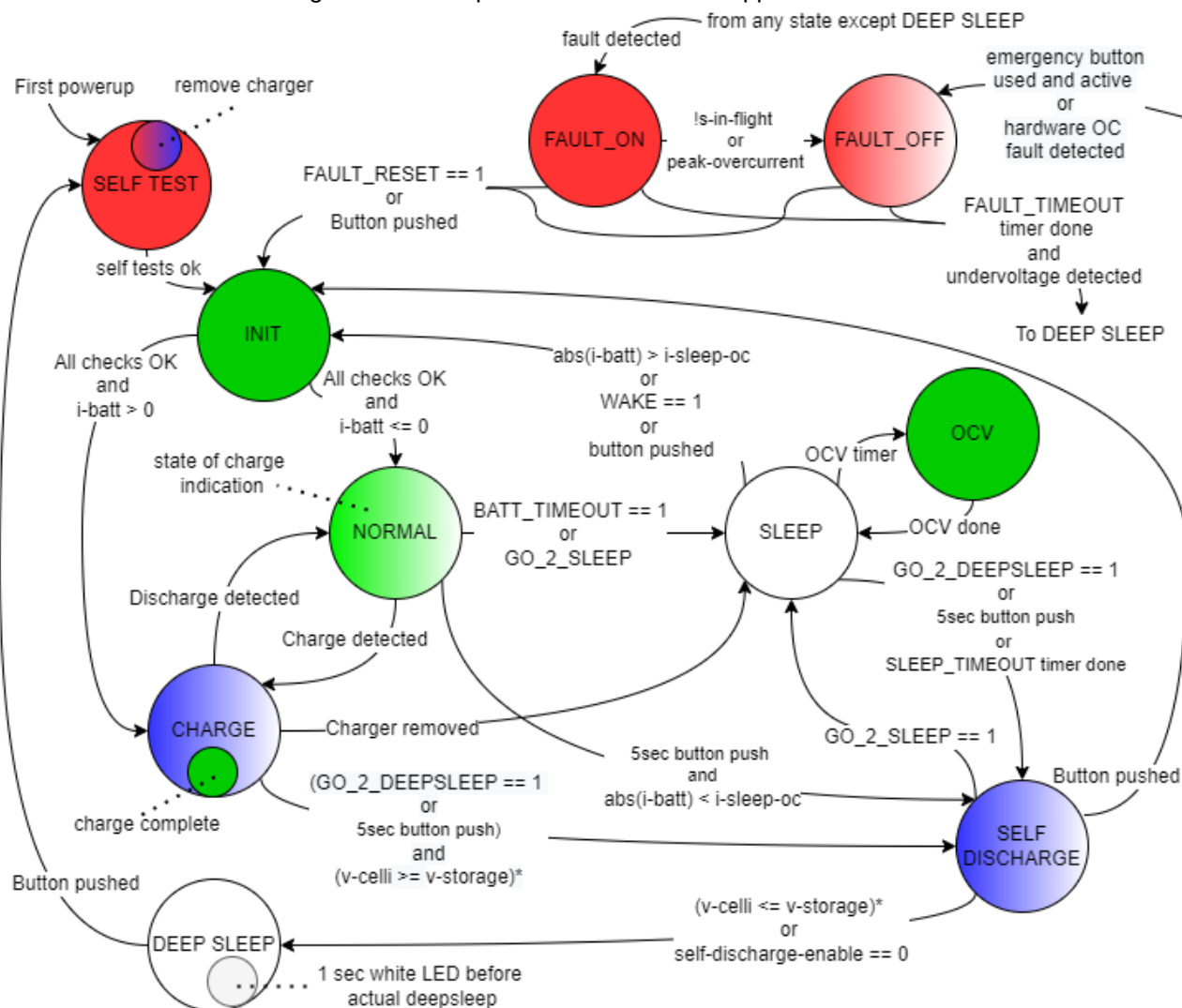
Since different parts need to use the same data, a data library will be made to take care of this. This library will make sure it is protected against usage at the “same” time by multiple tasks.

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 8](#) the main state machine that will be implemented in the BMS can be found. This state diagram will be implemented in the BMS application.



*Needs to meet this for each cell, i = cell number

Figure 8: Battery main state machine

10.3.2 Main state machine explained

SELF TEST state

The SELF TEST state is entered at power-up of the microcontroller. In this state the microcontroller initializes everything and performs the self-test, for example check if communication with a component is possible or check if the set output can be read. If everything is OK, it will go to the INIT state. If a watchdog reset has occurred not every self-test will be done to make sure the power is not turned off. The LED will be solid red in this state.

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 one of the FAULT states 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 where the battery operates as intended, 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 when the load is below the set threshold.
- 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 automatically.
- 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 9](#). 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 will calculate the estimated cell balance minutes per cell, which is based on the cell voltages, the difference compared to the lowest cell voltage, the balance resistor and the ocv-slope. The formula to calculate this estimated balance time is:

$$\text{Estimated balance minutes} = (V_{\text{cell}} - V_{\text{cell}_{\min}}) * \frac{R_{\text{bal}}}{V_{\text{cell}} * \text{ocvslope}}$$

Other than this calculated time, the BMS will check if the voltage of a cell that is being balanced, has reached the desired voltage as well. 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 all the calculated cell balance minutes are expired or it has reached the lowest cell voltage.

- RELAXATION: in this state the power switches are set open, disconnecting the battery from the charger. The MCU will be put in a very low power run (VLPR) mode, SBC in standby mode and the BCC to measure only at 10Hz. This will reduce the power in this state. 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 not finished, the BMS will re-estimate the balance minutes. 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 the LED will be steady green. If the lowest cell voltage decreases again below the cell overvoltage minus the recharge voltage margin, it will go to the CHARGE WITH CB state again. The power switches will remain open and if the charger is disconnected it will go to the SLEEP state after the defined period of 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 or the button is pressed for five seconds 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, then the BMS will transition to the SELF DISCHARGE state.

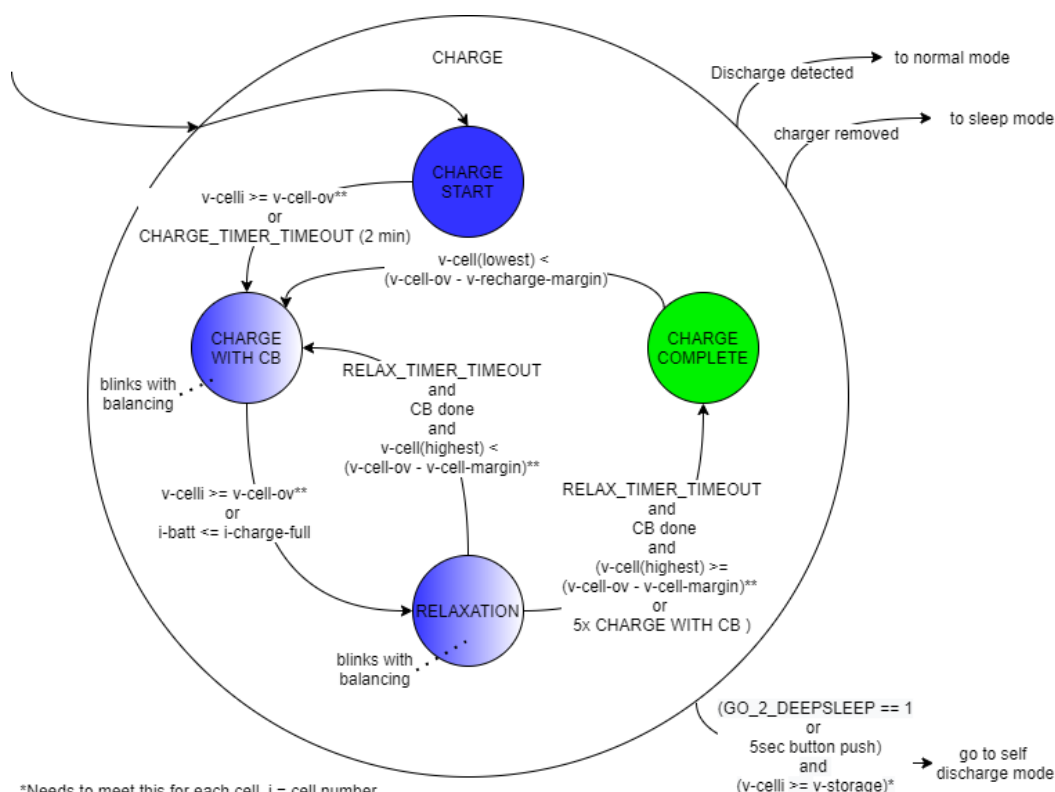


Figure 9: Charging state diagram

SLEEP state

The sleep state is typically entered when the current is very low for an amount of time. This SLEEP state is used to preserve power. The MCU will be in VLPR mode, the SBC in standby mode, the BCC in sleep mode with measurement on to check for a sleep overcurrent and other faults. 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. If the t-sleep-timeout happens the BMS will go to the SELF DISCHARGE state and will discharge the battery to storage level. After this it will go to the DEEP SLEEP state. In this state the LED will be off.

OCV state

The OCV state will allow to record the latest open cell voltage (OCV) of the battery which is used in the state of charge (SoC) computation. Cyclically the Battery will enter this mode when the Battery stays in the SLEEP state. The period the system will go from the SLEEP state to the OCV state will depend on the time since the battery has entered the SLEEP state in the first place, without going to another state except the OCV state. The time to enter the OCV state will gradually increase each time with 50% from the set begin time until the set end time is reached. If for example the set begin time is five minutes and the set end time is twenty-four hours, it will take fifteen times to have a period of is twenty-four hours. When entering this mode, the MCU will wake up the AFE and measure. The LED will blink green. After it has calibrated the SoC, it will go to the SLEEP state again.

FAULT_ON state

The FAULT_ON state is entered when a critical fault has been detected (over-current, over-voltage, cell over-temperature) and requires to evaluate if the battery needs to disable the output power. With the flight-mode-enable and the i-flight-mode parameter, the user can make sure that the power will not be disconnected and the BMS will stay in this state. If the s-in-flight parameter is high, it will stay in this state and not disconnect the power. Except, when the i-peak-max threshold is reached. The s-in-flight parameter is high if the i-batt-avg (1s) is higher than i-flight-mode and the i-batt-avg (1s) is less than the i-out-max AND the flight-mode-enable is high. The s-in-flight will be low again when: the i-batt-10s-avg is lower than the i-flight-mode and the i-batt-avg (1s) is less than the i-flight-mode. OR s-in-flight will be low when the flight-mode-enable is set to 0 OR when there is a i-batt-avg (1s) charge current higher than the i-sleep-oc ($(1s_current_avg - i-sleep-oc) > 0$). If the BMS is in the FAULT_ON state and s-in-flight will become false, it will go to the FAULT_OFF state.

If it needs to turn off the power, it will go to the FAULT_OFF mode. Other ways to exit this FAULT_ON state is that 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. If there is an undervoltage fault, it will transition to the DEEP SLEEP state after the t-fault-timeout time. In this state the LED will be solid RED to indicate the power is still on.

FAULT_OFF state

The FAULT_OFF state is entered when there is a fault and it needs to turn off the output power. Usually this state is entered because in the FAULT_ON state it noticed that it needs to turn off the power. It will go directly in the FAULT_OFF state if the emergency button is used and this button is active or if a hardware overcurrent is detected. To exit this FAULT_OFF state is that 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. If there is an undervoltage fault, it will transition to the DEEP SLEEP state after the t-fault-timeout time. In this state the LED will be blinking RED to indicate the power is off.

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. To get a better SoC estimation, the OCV is measured and this will update the SoC measurement of the battery. 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 SELF_TEST state. If configuration parameters have changed, it will save the parameters to flash to make sure they are loaded at startup.

10.4 Tasks priorities

The tasks of the BMS have different priorities, this is needed because some tasks/activities are more important than others. For example, one needs to react fast on a fault, so the system needs to prioritize a fault above blinking the LED. The BMS uses the NuttX preemption, which means that lower priority tasks get interrupted by a higher priority task. The task priorities can be seen in [Figure 10](#).

When one of the BCC functions is called (via the spiwrapper), the task is locked so it can't switch tasks. This way it makes sure it will execute the function OK, otherwise there could be CRC errors or NULL responses.

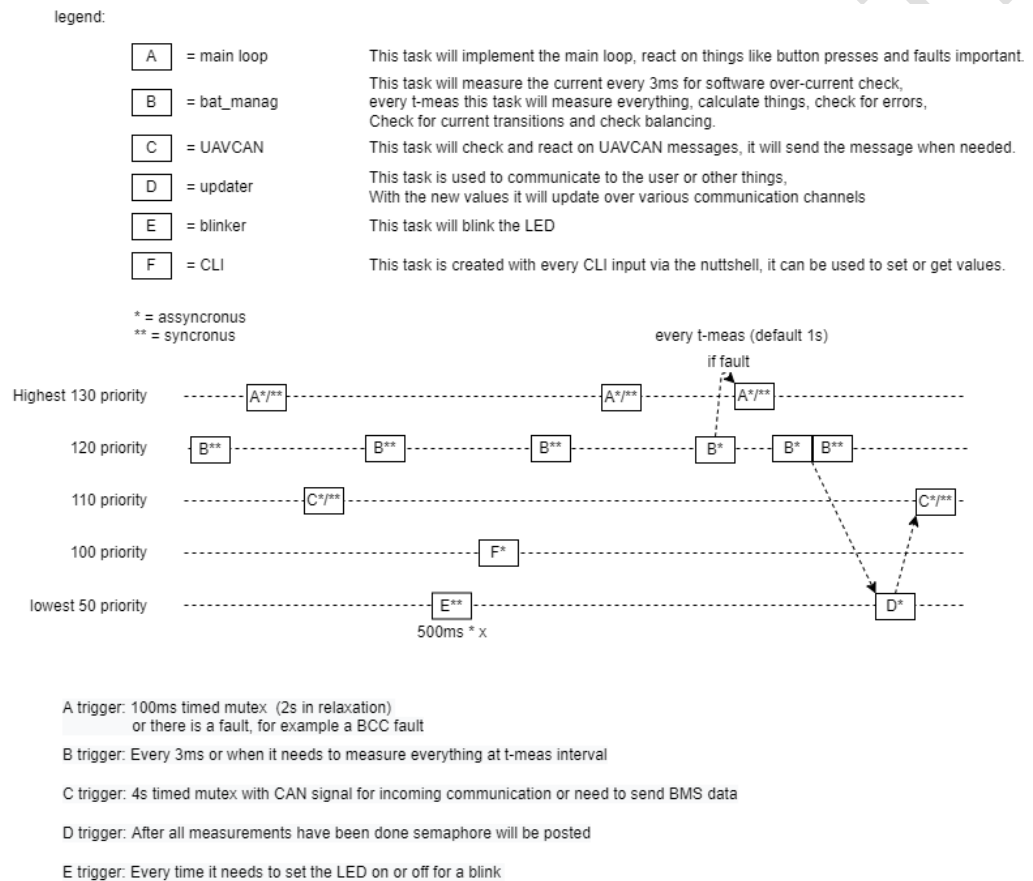


Figure 10: BMS task priorities

10.5 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.

10.5.1 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 as seen in [Figure 8](#). In the main source code, the state is changed. In this source file there is a function to handle a changed parameter as well. This function will call other 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. In the main loop, the watchdog is kicked as well. So if this is not done in time, it will reset the MCU.

DRAFT / CONFIDENTIAL

10.5.2 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](#).

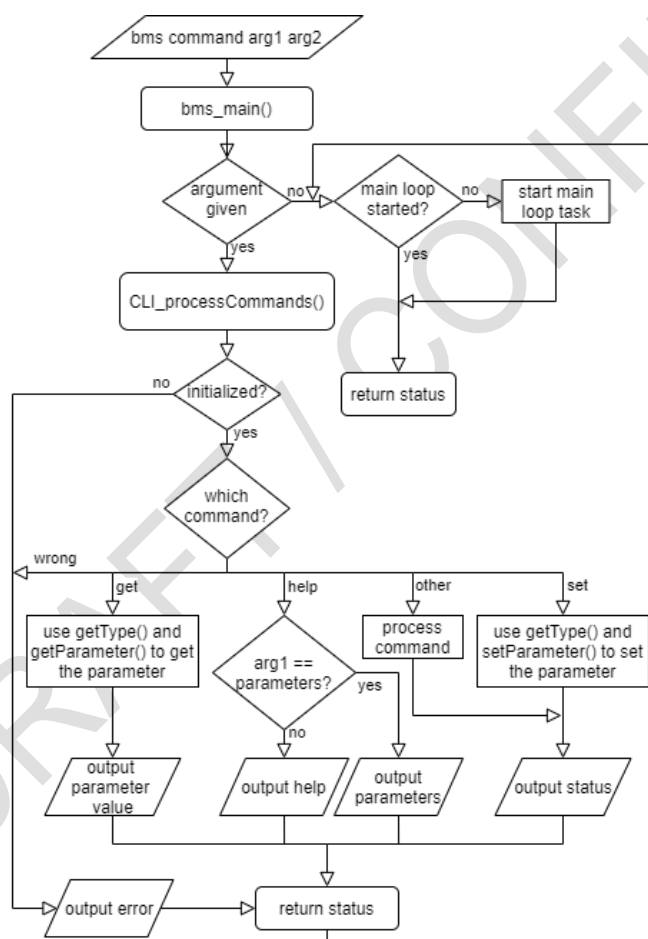


Figure 11: CLI flowchart

10.5.3 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.

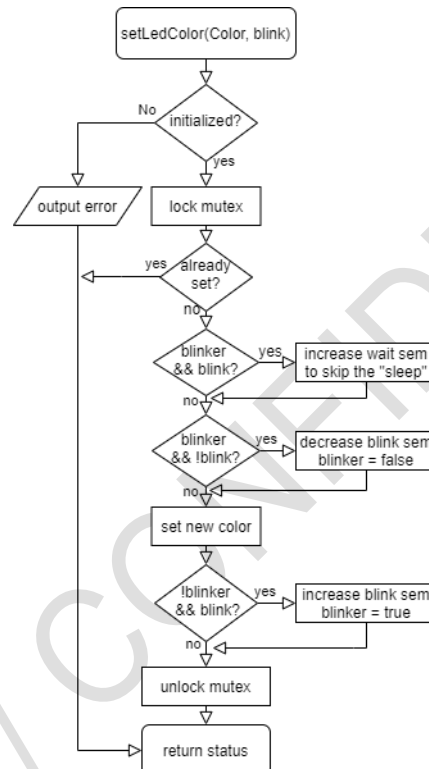


Figure 12: Set LED color flowchart

10.5.4 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.

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. An input pin can be an interrupt pin, on an interrupt it will generate a signal that will queue the action for the handler. Keep in mind that these signals can be very intrusive.

10.5.5 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.

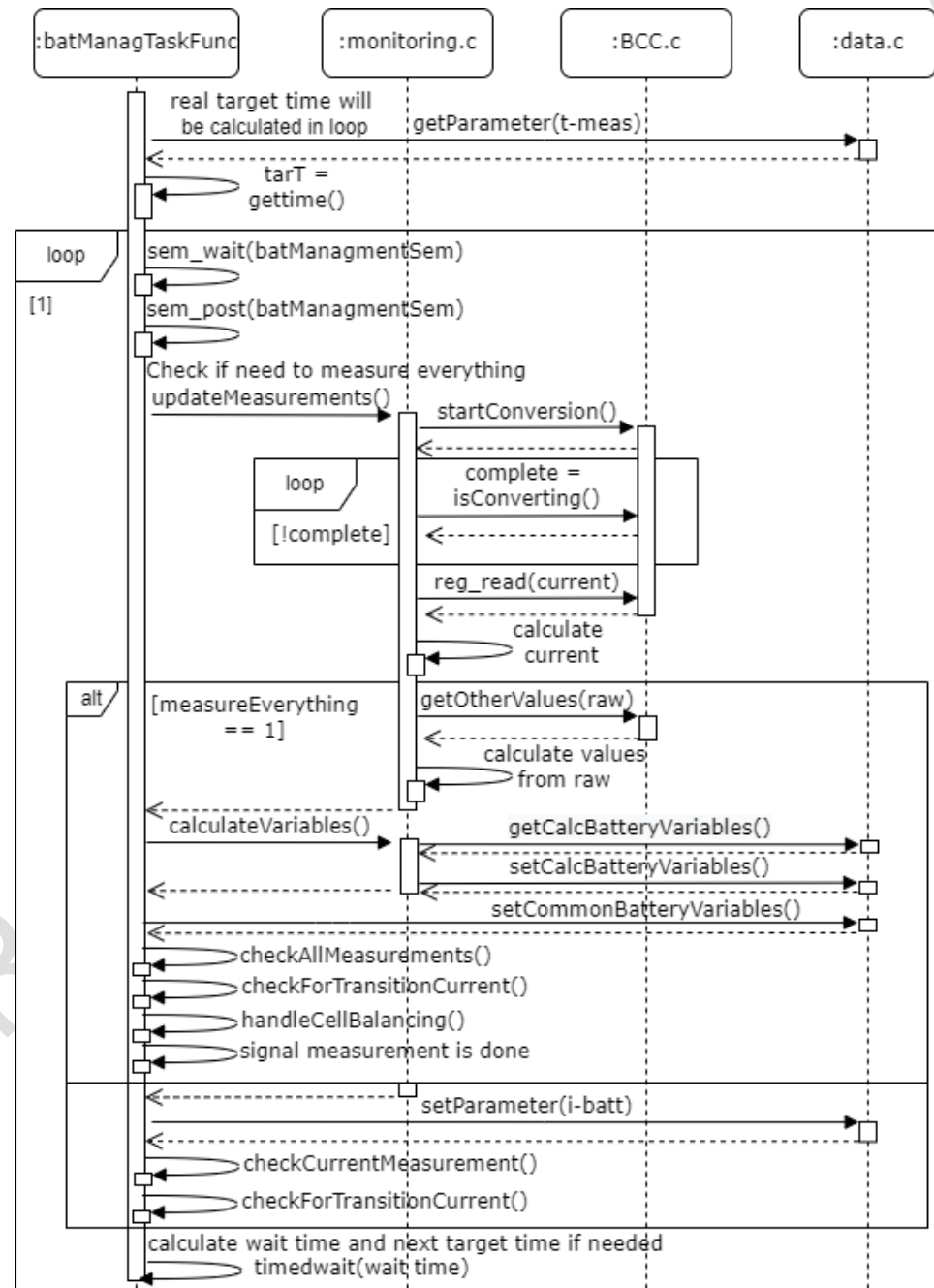


Figure 13: Calculate other values sequence diagram

The batManag task is created to handle the battery management. It will do the manual measurements with the BCC. Mostly it will calculate and check the current or it will measure and calculate every variable at t-meas interval. It will then perform a measurement, calculate the variables, save the values, it will check for software faults, it will calculate the new transition variables, handle the cell balancing if needed and it will trigger the callback that new measurements have been done. When saving the data, it will save the new measured and calculated variables in one go using the structs. Because the BCC will not check for an overcurrent, the current needs to be read and calculated every time to be compared with the current threshold. For short circuit protection there is a hardware circuit. If it measures a fault it will trigger the main to act on it.

The sequence of the batManag task can be seen in [Figure 13](#). 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. The meas task will check if the next measurement needs to be the measure everything or just the current and calculate the wait time to make sure the t-meas interval is met or with the remaining wait time. The task will wait for the next time or when the measurement is enabled, it will measure right away. 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.

To calculate the state of charge (s-charge), the coulomb counter is used. The coulomb counter register holds the sum of the measured currents (until read). There is another register that 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 full charge capacity 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 then 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 batManag task will check for software measured faults. The BCC chip will take care of hardware 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 then 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.

10.5.6 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 14](#) the simplified flowchart of this function can be seen. Besides power regulators, the SBC has a watchdog, which is used to reset the MCU if it doesn't kick the watchdog within the set time, this is done via the NRST pin.

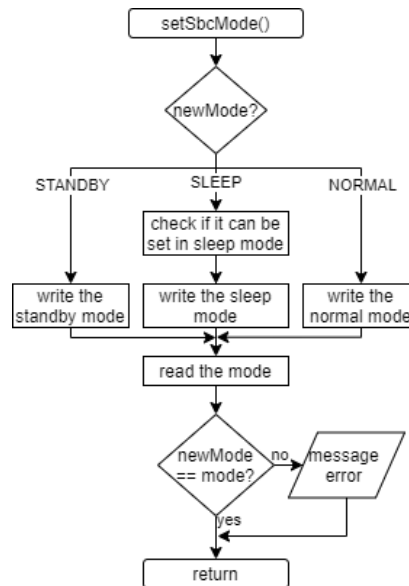


Figure 14: Set SBC mode flowchart

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 of 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

1.	Introduction	3	8.3.6	The hardware parameters	26
2.	Release package	4	9.	Charging	27
3.	Changes	5	9.1	How to charge the battery using the BMS	27
3.1	Changes relative to the last 5.0 release	5	10.	Software guide – NuttX	28
3.2	Changes of 5.0 relative to 4.0	5	10.1	Introduction	28
3.3	Changes of 4.0 relative to 3.6	5	10.2	Software block diagram	28
3.4	Changes of 3.6 relative to 3.4	6	10.2.1	Module description	29
4.	Limitations	7	10.3	BMS application state machine	31
5.	Known issues	9	10.3.1	The main state machine	31
6.	Block diagram	10	10.3.2	Main state machine explained	32
6.1	Board organization	10	10.4	Tasks priorities	36
7.	How to	12	10.5	Realization	37
7.1	How to program the BMS	12	10.5.1	Main	37
7.1.1	Software setup and debugger adapter board ...	12	10.5.2	CLI	38
7.1.2	Programming the software	13	10.5.3	LED state	39
7.1.3	Downloads	13	10.5.4	GPIO	39
7.2	How to use the DroneCAN interface	14	10.5.5	Bat management	40
7.2.1	How to set up the DroneCAN GUI tool	14	10.5.6	SBC	42
7.2.2	How to use the DroneCAN GUI tool to monitor the DroneCAN messages	14	11.	Legal information	43
7.3	How to use the CyphalCAN interface	16	11.1	Definitions	43
7.4	How to use the CLI	17	11.2	Disclaimers	43
7.5	How to configure the temperature sensor	17	11.3	Licenses	43
7.6	How to use SMBus (I ² C slave)	18	11.4	Patents	43
7.7	How to use NFC	19	11.5	Trademarks	43
8.	The parameters	20	12.	Contents	44
8.1	How to configure the parameters via the CLI ...	20			
8.2	How to configure the parameters via the DroneCAN GUI tool	20			
8.3	The parameter lists	22			
8.3.1	The common battery variables list	22			
8.3.2	The calculated battery variables list	22			
8.3.3	The additional battery variables list	23			
8.3.4	The configuration variables list	24			
8.3.5	The can variables list	25			

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