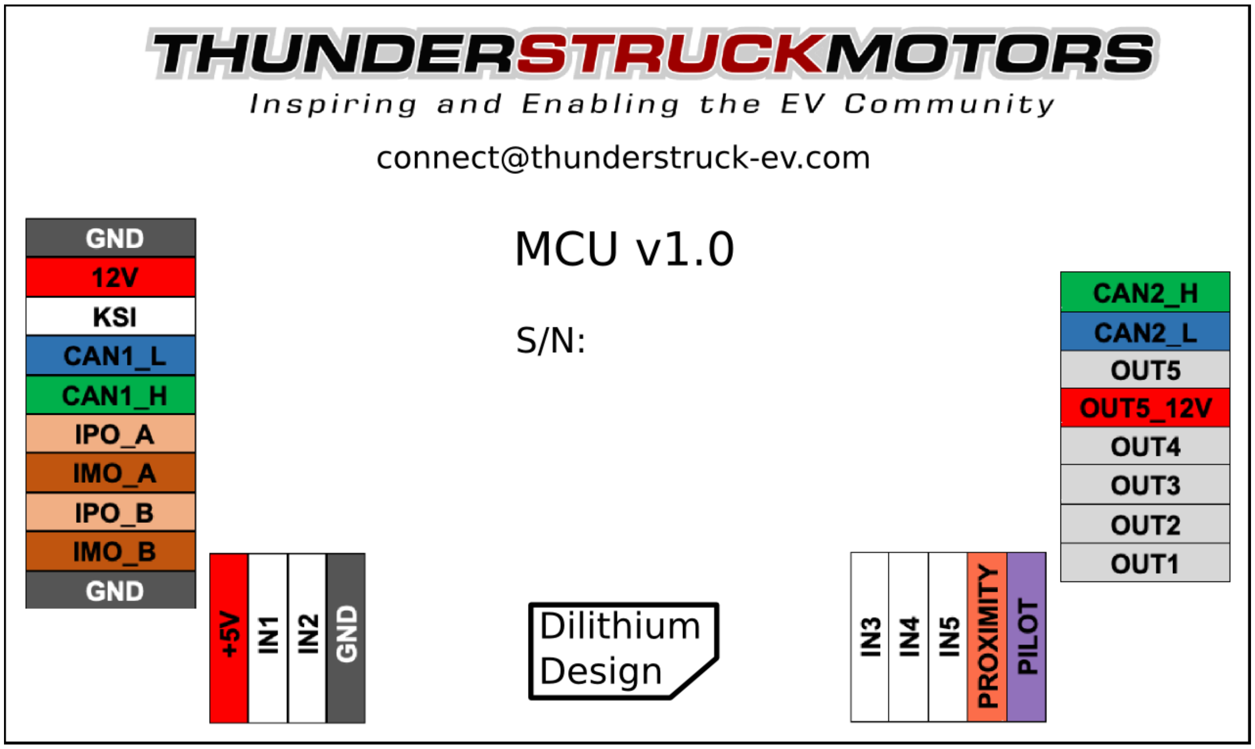
Thunderstruck Motors
EV Master Control Unit v1.0



Doc Rev 1.2, November 2021

Contents

OVERVIEW 5
INSTALLATION 6
Power 7
Inputs 8
Outputs 8
CAN 8
IsoSPI 10
LED 10
Serial Port 11
Firmware Upgrade 11
SYSTEM 13
Inputs 13
Outputs 15
CAN Configuration 17
CAN Services 17
BATTERY MANAGEMENT 18
LTC12 and LTC18 Architecture 18
LTC Cell Group Wiring 21
Thermistors 22
A123 23
Probe and Diagnostics 26
Pack, Group, and Cell Numbering 27
Cell Measurement 31
Cell Balancing 35
ONBOARD CHARGING 38
Charge Plugs and Line Powering 39
Charging 45
Charge Configuration 49
Charger Support 53
Debugging 55
OFFBOARD CHARGING 57
Fast Charging 57
ZEV CCS Charging 57

INSTRUMENTATION 61
State of Charge Measurement 61
Current Sensors 61
MCU EV Display 64
Analog Fuel Gauge 69
Speedhut Gauges 70
Curtis 3100r 71
OBDII Dictionary 72
SERIAL PORT MESSAGES 74
Startup Banner 74
LTC Discovery Messages 74
Cell HVC and LVC Messages 74
Thermistor Temperature Messages 74
IsoSPI Error Messages 75
SERIAL PORT COMMANDS 75
help 76
show 79
set 88
enable | disable 91
edit | delete 92
reset 93
measure 93
trace 94
lock 95
upgrade 96
Warrantee and Support 97
Document Revision 97

Figures

Figure 1 – MCU Enclosure 6
Figure 2 – Push-In Connector 6
Figure 3 – MCU Connectors 7
Figure 4 – CAN Network Diagram 9
Figure 5 – CAN Termination Solder Jumper 10
Figure 6 – MCU Blink Patterns 11
Figure 7 - Firmware Upgrade Utility 12
Figure 8 - Input Functions 13
Figure 9 - Output Functions 15
Figure 10 - CAN Services 17
Figure 11 – BMSS System Connectors 19
Figure 12 – BMSS Cell Harness Connectors 19
Figure 13 – BMSS Thermistor Connectors 20
Figure 14 - BMSS18 System Connector 20
Figure 15 - BMSS18 Harness Connector 20
Figure 16 – Cell Group Wiring 21
Figure 17 – Cell Group Wiring Spanning A Fuse 22
Figure 18 - Thermistor Connections 22
Figure 19 – Using the Cell Harness Tester 23
Figure 20 - MBB Connector Pinout 25
Figure 21 – 48 Cell Pack Wiring 26
Figure 22 - 44 Cell Pack, Cell Groups Connected Where Convenient 28
Figure 23 – Parallel Packs 30
Figure 24 – J1772 Type 1 System Diagram 40
Figure 25 – J1772 Type 1 EV Connector 41
Figure 26 – J1772 Type 2 Connector 42
Figure 27 – J1772 Type 2 - One Charger Per Phase 43
Figure 28 - Onboard Charge Termination Reasons 48
Figure 29 - Current Sensor Support 61
Figure 30 – DHAB V2 Hall Sensor Installation 63
Figure 31 - DHAB Hall Sensor Connections 63
Figure 32 - HTFS Hall Sensor Connections 64
Figure 33 - MCU EV Display Hardware 65
Figure 34 - MCU Display Startup and Status Screens 66
Figure 35 - MCU Display Message Region 67
Figure 36 – MCU Display Cell Map Examples 68
Figure 37 – Analog Fuel Gauge Connections 69
Figure 38 – 3100r Display 71
Figure 39 - OBDII PID Support 73

OVERVIEW

The Thunderstruck EV Master Control Unit (MCU) performs Battery Management, Charge Control, and Instrumentation Control for Electric Vehicles.

The MCU supports three Battery Management architectures: LTC12, LTC18, and A123. The BMS architecture determine the measurement technology and the cell group size.

- LTC12: 12 cells per cell group, uses Dilithium Design BMSS
- LTC18: 18 cells per cell group, uses Dilithium Design BMSS18
- A123: 13 cells per cell group, uses the 13-cell A123 MBB (Measurement and Balance Board)

The MCU supports two Analog Device IsoSPI interfaces to communicate with BMSS and BMSS18 measurement boards. Each IsoSPI interface can support up to 8 daisy-chained measurement devices. An LTC12 architecture can support up to 196 cells and a LTC18 architecture can support up to 288 cells.

The A123 MBBs use CAN to communicate. Up to 16 MBBs are supported, for a total of 208 cells.

The MCU Charge Control functions support the J1772 Type 1 and Type 2 charge protocols and support a variety of CAN enabled chargers, including the Thunderstruck TSM2500 charger. The MCU can automatically start upon charge plug insertion. Multiple chargers and multiple charge profiles are supported. The MCU has hardware support for fast charging: software support for fast charging is in development.

The MCU supports an external Hall current sensor that performs State of Charge calculation. It can drive an analog OEM “fuel gauge”, as well as CAN based gauges such as the Curtis 3100r display, CAN-enabled Speedhut gauges, and the Dilithium Display.

The MCU supports ODBII and can operate with Torque PRO. The MCU supports CANopen allowing CAN based monitoring and configuration.

The MCU has five general purpose inputs and five general purpose outputs that may be flexibly mapped to a variety of functions, depending upon application.

The MCU has a green/red LED to show overall system status and can report system alerts using discrete outputs. MCU configuration is performed using a serial interface and a termination emulation program such as PuTTY. The firmware supports diagnostic commands to verify proper hardware operation and wiring, to trace CAN messages, and to retrieve statistics. MCU firmware can be upgraded using the serial interface should that be necessary.

INSTALLATION

The MCU is enclosed in a Serpac WM021I enclosure, a 4.1 x 2.65 x 1.1 plastic enclosure with mounting flanges. The enclosure and connector are not waterproof and should be appropriately protected from the elements.

The datasheet for the enclosure is at https://www.serpac.com/userprints/wm021i_up.pdf.

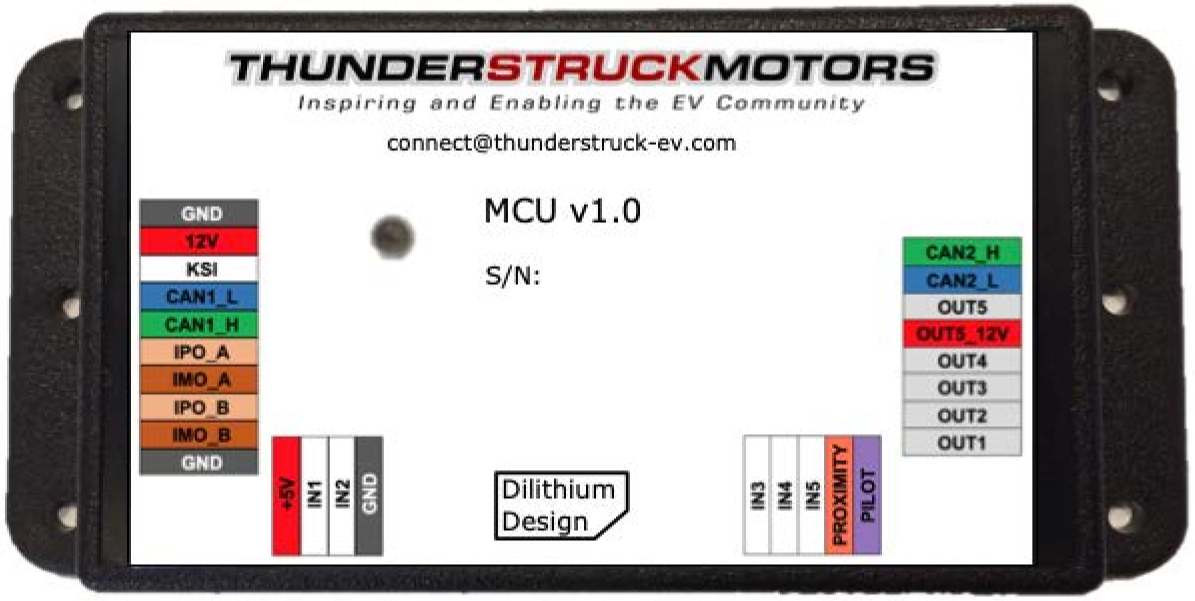


Figure 1 – MCU Enclosure

The four connectors (labelled A, B, C, and D) are "push-in" connectors. These connectors accept 20-24 gauge stranded or solid wire: stranded 20-gauge wire is recommended. To make a connection, strip the wire back 1/4". Twist the wire end and insert. Be sure that all strands of wire get fully inserted to prevent shorting between adjacent wires.

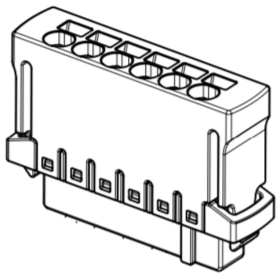


Figure 2 – Push-In Connector

Removing the wire from the connector requires a removal tool, supplied. To remove a wire, insert the tool into the associated slot above the wire and wiggle it in. This will collapse the spring holding the wire and the wire can be removed.

The connector part numbers are:

• 10p Connector A	Harting 14311013101000
• 4p Connector B	Harting 14310413101000
• 5p Connector C	Harting 14310513101000
• 8p Connector D	Harting 14310813101000

The following figure lists the MCU connections:

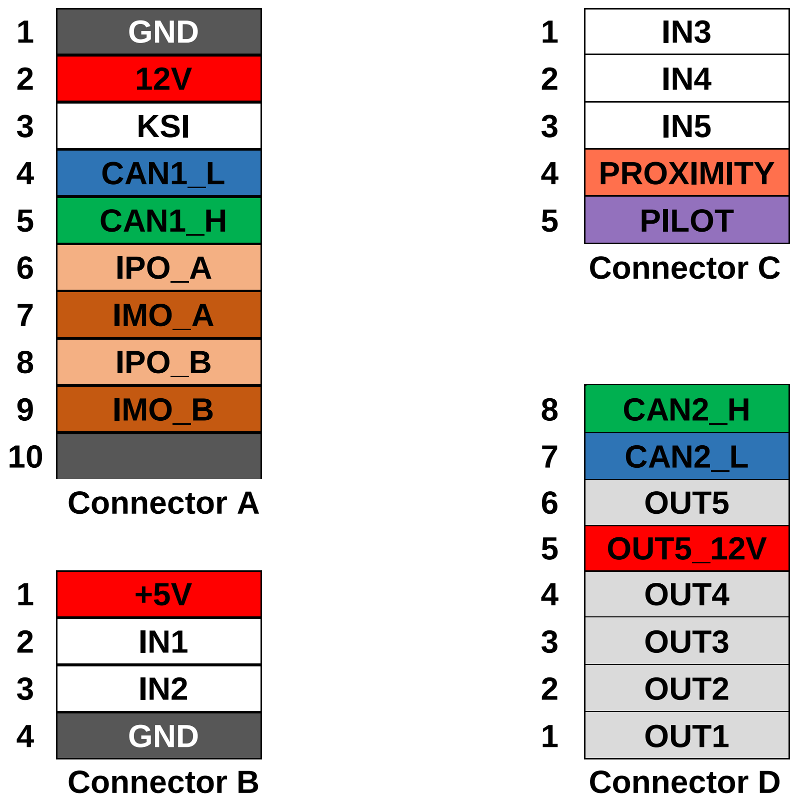


Figure 3 – MCU Connectors

Power

12V (input) and **GND** (Connector A, pins 1 and 2) provide "always on" power.

+5V (output) and **GND** is provided on Connector B, pins 1 and 4. Connector B is intended to connect directly to a dual range Current Sensor such as the LEM DHAB S137.

The pin **OUT5_12V** (Connector D, pin 5) is an input which provides power for the OUT5 output.

Inputs

IN1 to **IN5** are general purpose inputs.

- **IN1**, **IN2**, **IN3**, **IN4** measure voltage from 0 to 5V. These inputs are tolerant up to 14V.
- **IN5** contains an onboard pullup resistor and measures resistance to ground.

General purpose inputs are assigned to an **Input Function** which determines how the input is interpreted by MCU firmware. The full list of Input Functions is described in the SYSTEM section, below.

KSI, **Proximity**, **Pilot**, and are fixed function inputs.

- **KSI** (Connector A, pin 3) is the "Key Switch Input. This input is connected to +12V when the EV KeySwitch is in the "ON" position.
- **Proximity** (Connector C, pin 4) handles the J1772 PROXIMITY signal. It measures the proximity resistor and has autostart support.
- **Pilot** (Connector C, pin 5) handles the J1772 PILOT signal. As an input, it measures the duty cycle of the J1772 PILOT square wave.

Outputs

OUT1 to **OUT5** are general purpose outputs.

- **OUT1**, **OUT1**, **OUT3** and **OUT4** are "open collector" outputs[1] rated to 200ma. OUT1, OUT2, and OUT3 support a PWM output mode.
- **OUT5** is an output that can switch to 1.5A of +12V. When using this output**, OUT5_12V** must be connected to a 12V source which provides the current for this output.

General purpose outputs are assigned to an **Output Function** which determines how the output is used by MCU firmware. The full list of Output Functions is described in the SYSTEM section, below.

Pilot is both an input and output. As an output, it can apply the J1772 PILOT resistor and diode as required by J1772.

CAN

CAN is a communications protocol widely used in automotive applications. The CAN protocol has sophisticated error detection and recovery mechanisms that allow for automatic retry and recovery as well as ways of detecting and isolating misbehaving devices.

The MCU supports two independent, non-isolated CAN interfaces, CAN1 and CAN2.

CAN1_L and **CAN1_H**, on Connector A, are the CANL and CANH signals for CAN1.
CAN2_L and **CAN2_H**, on Connector D, are the CANL and CANH signals for CAN2.

[1] An "open-collector output" is high impedance (or not connected) when not enabled and connected to ground when enabled.

CAN baud rate programming and service configuration is described in the SYSTEM section, below.

A CAN network is a multi-station network that is terminated at both ends of the network by 120 ohm termination resistors. Each device on the CAN bus uses two connections: CANH ("CAN high") and CANL ("CAN low") to communicate on the bus. Although usually not shown, but important, is that each node on the CAN network share a common ground (GND).

CAN wiring should be kept short and the conductors should be twisted. Wiring stubs between the CAN network and a device should be kept as short as possible, ideally less than a few inches. Network wiring should be placed away from EMI (Electro-Magnetic Interference), such as the motor and controller, and parallel runs next to EV traction cabling should be avoided.

A CAN network might only consist of two devices. In this case a short direct connection between the two using hand-twisted wire is sufficient. For longer runs, more devices, or cases where EMI may be an issue, shielded cable may be used. If a shield is used, it should be connected to chassis ground at a single place.

The following generic diagram depicts a typical CAN network:

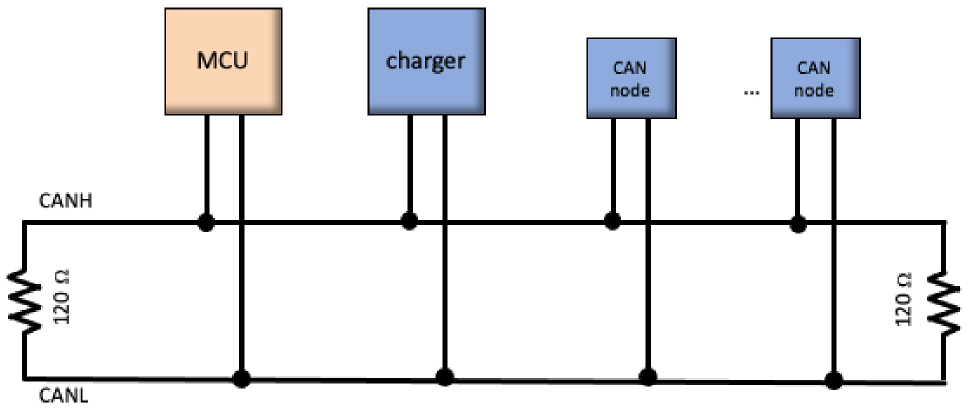


Figure 4 – CAN Network Diagram

CAN termination resistors are installed on both CAN interfaces in the MCU, which can obviate the need for an external termination resistor. The assumption, however, is that the MCU is placed as a terminal node in the CAN network(s). If necessary, the MCU termination resistors may be disabled: open the enclosure, locate JP1 or JP2 and cut the trace connecting the gold pads using an X-acto knife or equivalent. (If, later, it is necessary to reinstate the jumper, a blob of solder may be placed on the jumper to bridge the termination).

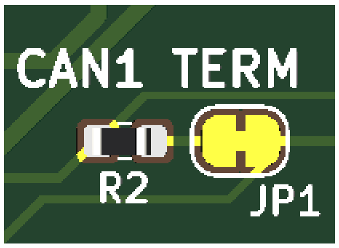


Figure 5 – CAN Termination Solder Jumper

In order to verify that the CAN network is properly terminated, make the connections to all CAN nodes and measure the resistance between CANH and CANL: it should be 60 ohms, which indicates the presence of two 120-ohm resistors in parallel.

IsoSPI

IsoSPI is a protocol defined by Analog Devices used to communicate with its battery cell measurement devices. IsoSPI provides galvanic isolation between nodes and is resistant to EMI. IsoSPI is a "daisy-chain" protocol: each device receives every message and forwards it downstream. Replies are clocked in a serial fashion from all connected devices by the MCU. IsoSPI can operate at 1 Mbps over 100 meters of cable. The protocol supports a Packet Error Code to detect datalink errors.

The MCU supports two IsoSPI interfaces. These interfaces require no configuration and are automatically used by the MCU if the LTC12 or LTC18 BMS architecture is configured.

IPO_A and **IMO_A**, on Connector A, are the IPO and IMO signals for IsoSPI interface A.
IPO_B and **IMO_B**, on Connector A, are the IPO and IMO signals for IsoSPI interface B.

The rules for wiring an IsoSPI network are similar to those for CAN. Wiring should be kept short and the conductors should be twisted. Wiring should be placed away from EMI (Electro-Magnetic Interference) such as the motor and controller, and parallel runs next to EV traction cabling should be avoided. There is, however, no need to add external termination resistors to the IsoSPI network as all Dilithium hardware has the necessary on-board terminations.

LED

The MCU maintains a list of active alerts (or "faults") and conditions. Each alert and condition have an associated priority and blink pattern. Multiple alerts may be present at once, however the highest priority alert is used to determine which blink pattern is displayed.

The following blink patterns are defined. In the diagram, each rectangle represents 125ms.

Alert																	**Reference**
HARDWARE	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	Probe and Diagnostics
CELL_CENSUS	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥			Cell Census
THERM_CENSUS	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥					Thermistors
HVC	🟥	🟥			🟥	🟥			🟥	🟥			🟥	🟥			HVC and LVC Conditions
LVC	🟥	🟥	🟥	🟥					🟥	🟥	🟥	🟥					HVC and LVC Conditions
THERM_OVERTEMP	🟥	🟥	🟨	🟨	🟥	🟥	🟨	🟨	🟥	🟥	🟨	🟨	🟥	🟥	🟨	🟨	Thermistors
THERM_UNDERTEMP	🟥	🟥	🟨	🟨	🟥	🟥	🟨	🟨	🟥	🟥	🟨	🟨	🟥	🟥	🟨	🟨	Thermistors
NOT_LOCKED	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	Cell Census
Condition																	
ONBOARD_CHARGE	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	
FAST_CHARGE	🟩	🟩	🟩	🟩	🟨	🟨	🟨	🟨	🟩	🟩	🟩	🟩	🟨	🟨	🟨	🟨	
CELL DISCHARGE	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟨	🟨	
“ALL OK”	🟩	🟩															

Figure 6 – MCU Blink Patterns

Note: In addition to these blink patterns, the bootloader has a fast green blink when loading code.

Serial Port

The MCU uses a serial port for configuration, diagnostic output, debugging, and for firmware upgrade. A 5V USB to 3.5mm TRS jack serial cable is required. The FTDI TTL-232R-5V-AJ is recommended. To use the cable, the host computer requires serial port drivers (also known as “Virtual Comm Port Drivers”), which may be downloaded from the FTDI website www.ftdichip.com/Drivers/VCP.htm. In addition, a terminal application such as PuTTY is needed on the host computer. This is available for download at http://www.putty.org/. The MCU operates at 115200 baud, 8 data bits, 1 stop bit. There are good tutorials online on how to install and configure the VCP drivers and PuTTY; search for “ftdi installation” or “putty installation”.

The serial interface may be tested with only three connections to the MCU. Connect +12V to the **12V** and **KSI** inputs and connect ground to the **GND** input. At powerup, the MCU should print the “startup banner” at the serial interface.

When the MCU powers up, it prints a startup banner on the Serial Port which includes firmware version information, it may also print several different types of messages, including inventory messages (such as how many BMS measurement devices have been detected) and alert messages. The MCU firmware is functionally organized into four areas: MCU, System, Battery Management, Charge Control, and Instrumentation. These are abbreviated: MCU, SYS, BMS, EVCC, and INST. Navigation between areas is done by typing the name of the subsystem. For example, to navigate to the BMS settings, type “bms” at any prompt.

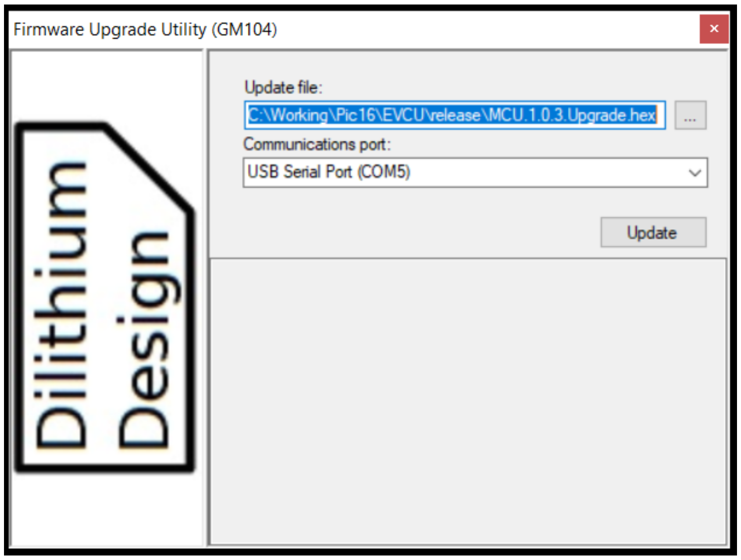
Firmware Upgrade

For upgrade, an upgrader utility is needed which can be downloaded from:

https://drive.google.com/file/d/1jjLtXz-WnXIRrVIQjqFodetpKZLkBSAk/view?usp=sharing. The upgrade utility works on a Windows computer.

The Upgrade tool and the Serial Port cable share the same connector on the MCU. In order to upgrade, the user must invoke the upgrade application on the MCU (type "**`upgrade`**"). The Terminal Application (PuTTY) must be exited, and then the upgrade utility is launched.

The upgrade tool requires two inputs: the name and location of the upgrade hex file and the serial comm port that is used. Once those are chosen, press the "Update" button to upgrade.


Firmware Upgrade Utility (GM104)

Update file: C:\Working\Pic16\EVCU\release\MCU.1.0.3.Upgrade.hex

Communications port: USB Serial Port (COM5)

Figure 7 - Firmware Upgrade Utility

After upgrade, it is necessary to cycle power to the MCU which can be done by removing and reconnecting "Connector A".

SYSTEM

System settings can be retrieved at the serial port by navigating to the “sys” prompt and typing the command **`show config`**. This will show Inputs, Outputs, CAN configuration, and CAN services.

```
mcu> sys
sys> show config
--inputs---------------------------------
  in1       : hall1
  in2       : hall2
--outputs--------------------------------
  out1      : fuelg
  out2      : hvc
--can------------------------------------
  can1br    : 250 Kbps
  can2br    : 500 Kbps
--services-------------------------------
  canopen   : enabled
  obcharge  : can1
```

Inputs

The following table shows the list of Input Functions:

Input Function	Description	IN1/2/3/4	IN5	Reference
hall1	Hall Sensor Low	X		see *Hall Sensor Support*
hall2	Hall Sensor High	X		see *Hall Sensor Support*
loop	Cell Loop	X	X	see *Charge Configuration*
pselect	Profile Selection	X	X	see *Charge Profiles*
socres	SOC Reset	X	X	see *Current Sensor Support*
cseq1	Charge SEQUENCE 1	X		see *Fast Charging*
cseq2	Charge SEQUENCE 2	X		see *Fast Charging*
plugin	Charge Cable Present	X		see *ZEV CCS Charging*

Figure 8 - Input Functions

An Input may be mapped to at most one Input Function and vice versa. As an example, the **pselect** Input Function is used to select a charge profile. The **pselect** Input Function may be mapped to **IN5** using the following command:

```
sys> set in5 pselect
```

The command **`show input`** will show the mappings from Input to Input Function, and will display both the hardware value (e.g., the voltage or resistance reading at the Input) and the “logical value” (e.g., the interpretation of that reading, depending on the mapped Input Function).

```
sys> show input
  in1   1.21v  hall1   DHABS137 H = -643.0A
  in2   0.66v  hall2   DHABS137 L = -68.1A
  in3   0.92v
  in4   0.57v
  in5    inf   pselect 1, profile=1
  prox   2.7K  J1772 disconnected
  pilot        not detected
  ksi      1   detected
sys>
```

Inputs IN1 through IN4 measure voltage; Input IN5 measures resistance. In this example, IN5 has been mapped to **`pselect`**. The reading at IN5 is “inf”, interpreted as profile 1. The **`show input`** command also shows the values of fixed function inputs (proximity, pilot, and ksi).

A mapping may be deleted by setting an input to “none”.

```
sys> set in3 none
```

The **`measure`** command can be used to report the value of an input and is intended as an aid to hardware troubleshooting. Note that the **`measure`** command takes one argument which can either an Input (such as **IN3**) or an Input Function (such as **pselect**). Once initiated, the output will repeatedly print until any key is pressed.

```
sys> measure in3
sys>   in3   0.90v
  in3   0.89v
  in3   0.89v
  in3   0.90v
  in3   0.90v

sys> measure hall1
sys>   in1   1.21v  hall1   DHABS137 H = -642.5A
  in1   1.21v  hall1   DHABS137 H = -642.5A
  in1   1.21v  hall1   DHABS137 H = -642.5A
  in1   1.21v  hall1   DHABS137 H = -642.5A
sys>
```

Outputs

The following table shows the list of Output Functions:

Output Function	Description	OUT1/2/3	OUT4/5	Reference
power	Power	X	X	see *Charging*
charge	Charging	X	X	see *Charge Configuration*
fuelg	Analog Fuel Gauge	X		see *Analog Fuel Gauge*
hvc	High Voltage Cutoff	X	X	see *HVC and LVC Conditions*
lvc	Low Voltage Cutoff	X	X	see *HVC and LVC Conditions*
driveok	Charge Plug unplugged	X	X	see *Charge Configuration*
cperm	Charge Permission	X	X	see *Fast Charging*
ccont	Contactor Control	X	X	see *Fast Charging*

Figure 9 - Output Functions

OUT1, **OUT2**, **OUT3**, and **OUT4** are open-collector outputs, which are high impedance (e.g., not connected to anything) when disabled, and switched to ground when enabled. The switched output, **OUT5**, is high impedance when disabled, and switched to +12V when enabled.

OUT1, **OUT2**, **OUT3** support a PWM output mode. In this mode, the output will be square wave with a software-controlled duty cycle. Currently, only the **fuelg** Output Function uses PWM output.

An Output may be mapped to at most one Output Function and vice versa. As an example, the **charge** Output Function indicates when the MCU is charging. The **charge** Output Function is mapped to Output OUT5 using the **`set`** command:

```
sys> set out5 charge
```

The command **`show output`** will show the Output Function mappings and display both the hardware output (e.g., high impedance or ground), and the logical value (e.g., the interpretation of that reading, which depends on the mapping).

```
sys> show output
  out1   123   fuelg   soc = 32%
  out2   ---   hvc  disabled
  out3   GND   charge  enabled
  out4   ---
  out5   ---
sys>
```

The second column of the example shows the hardware output. (“---” means “high impedance, “GND” means grounded, “+12V” means connected to +12V, and a number is the PWM setpoint).

The third column of the example is the Logical Output; the fourth column indicates the logical interpretation of the output.

A Logical Output can either be "TRUE" or "FALSE". By default, if the Logical Output is TRUE, then the associated Output will be enabled.

Depending on application, it may be convenient to invert this mapping. Suppose, for example, it is necessary that OUT3 be enabled when charging is NOT occurring. This can be achieved by the command:

```
sys> set out3 -charge
```

The leading "minus sign" before the Logical Output will invert the sense of the output.

```
sys> set out2 -lvc
sys> sh out
  out1   123   fuelg   soc = 32%
  out2   ---   hvc  disabled
  out3   ---  -charge  disabled
  out4   ---
  out5   ---
sys>
```

A mapping may be deleted by using the command:

```
sys> set out3 none
```

Finally, it is possible to **enable** or **disable** outputs. These functions are intended as an aid to hardware troubleshooting. Note that it is best to **enable** or **disable** an output that has not been mapped, otherwise running firmware may overwrite any "enable" or "disable" setting.

```
sys> enable out5
  out5  +12v
sys> sh out
  out1   123   fuelg   soc = 32%
  out2   ---   hvc  disabled
  out3   ---
  out4   ---
  out5  +12v
sys>

sys> disable out5
  out5   ---
sys> sh out
  out1   123   fuelg   soc = 32%
  out2   ---   hvc  disabled
  out3   ---
  out4   ---
  out5   ---
sys>
```

CAN Configuration

The MCU supports two CAN interfaces, CAN1 and CAN2.

The can baud rate for the CAN1 or CAN2 can be changed using the commands:

```
sys> set can1br 500
sys> set can2br 250
```

CAN Services

The MCU supports several CAN Services. By default, the **canopen** and **obcharge** services are enabled on CAN1. The **canopen** service communicates with the Dilithium MCU display. The **obcharge** service communicates with the onboard CAN-enabled chargers.

The following table shows the list of CAN services:

CAN Service	Type	Reference
canopen	Client	CANOPEN
obdii	Client	OBDII
a123	Client/server	A123 MBB
obcharge	Client/server	Onboard Charging
fcharge	Client/server	Fast Charging
zevccs	Client/server	ZEV CCS Charging
3100r	Client/server	Curtis 3100r
sh_fuel	TX Only	Speedhut
sh_amps	TX Only	Speedhut
sh_temp	TX Only	Speedhut

Figure 10 - CAN Services

CAN services are enabled and assigned to a CAN interface using the **enable** command. CAN services can be disabled using the **disable** command.

```
sys> enable obdii
sys> enable 3100r can2
sys> disable canopen
```

CAN Services of type “Client” respond to requests from either CAN1 or CAN2 and transmit responses on the interface that received the request.

CAN Services of type “Client/server” only operate on a single CAN interface. Messages received are only processed if they arrive on the configured interface. Messages transmitted are only sent on the configured interface.

CAN Services of type “TX only” only transmit messages on a single CAN interface; they never receive messages.

BATTERY MANAGEMENT

The MCU supports three Battery Management architectures which determines the measurement technology and cell group modularity. In the BMS a **cell group** is a series connected set of cells with dedicated measurement hardware. Cell groups can be flexibly mapped to "packs" to create different configurations. All architectures isolate pack voltage from the MCU and communicate using an isolated datalink.

The BMS architecture is set using the **`arch`** parameter (to one of **`ltc12`**, **`ltc18`**, **`a123`** or **`none`**, which disables the BMS). The **`arch`** parameter is not affected by the **`reset config`** command and must be edited separately.

The **LTC12** architecture has a cell group size of 12 cells. This architecture is used with Dilithium Design BMSS measurement boards which can monitor two cell groups or 24 cells.

The **LTC18** architecture has a cell group size of 18 cells. This architecture works with the Dilithium Design BMSS18 measurement board which can monitor up to 18 cells. This architecture can also support a mixed configuration of BMSS18 and ltc6811-based BMSS modules.

The **A123** architecture has been optimized for 26 cell A123 modules, where each module contains two groups of 13 cells; each group is connected to an A123 Measurement and Balance Board (MBB).

LTC12 and LTC18 Architecture

The measurement technology used in the BMSS and BMSS18 are the Analog Devices ltc6811, ltc6804, and ltc6813 devices ("LTC"s). These devices share many features and specifications. An LTC device can measure its cells in less than 2.5 ms with a total measurement error of less than 1.2mV. Accuracy is stable over voltage, temperature, and time. LTCs support diagnostics to identify hardware and wiring problems and are tolerant of common wiring errors such as shorted or open connections. The BMSS and BMSS18 support passive cell balancing.

LTCs use the Analog Devices IsoSPI protocol to communicate between devices and with the MCU. IsoSPI is a two-wire protocol which provides galvanic isolation between cell groups.

The LTC devices have been designed to minimize power consumption, especially during long-term storage where battery drain is unacceptable. When the MCU enters "sleep" mode it stops sending IsoSPI messages to the LTCs, and the LTCs automatically enter a low power mode which draws minimal current (~4 µA).

When an IsoSPI message arrives, the LTCs will power up. LTCs are powered from the pack connections: current is used in short bursts to operate the IsoSPI datalink and for cell measurement. The same amount of current is drawn from each group, regardless of the number of cells connected, which minimizes the possibility of unbalancing the pack due to LTC current needs.[2]

BMSS

The Dilithium Design BMS Satellite (BMSS) uses the Analog Devices ltc6811 or ltc6804 battery monitor chips and supports two cell groups of 12 cells each. A cell group can consist of fewer than 12 cells, but cells must be series connected and in total result in greater than 11V. Depending on nominal cell voltage, the minimum cell group size should be at least four (or five) cells.

The BMSS has an upstream IsoSPI interface (IPI and IMI) and a downstream IsoSPI interface (IPO and IMO) which connects to downstream BMSS devices, if any. There is also an internal IsoSPI connection between the two onboard measurement devices.

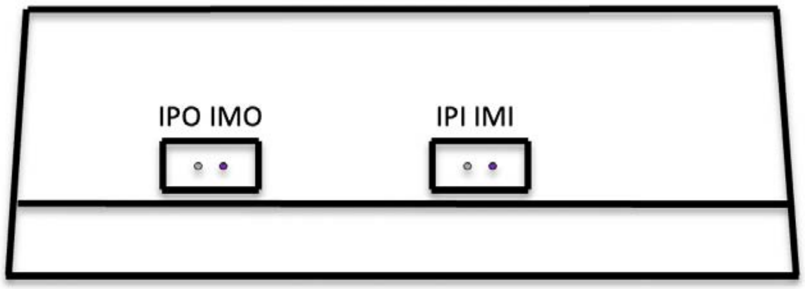


Figure 11 – BMSS System Connectors

A BMSS has two cell harness connectors, A and B. If only one cell group is used in the BMSS, it must be connected to the "A" harness connection. The two LTCs in a BMSS are electrically isolated from each other and as a result, LTCs can be connected to Cell Groups anywhere in the pack and in any order, wherever convenient.

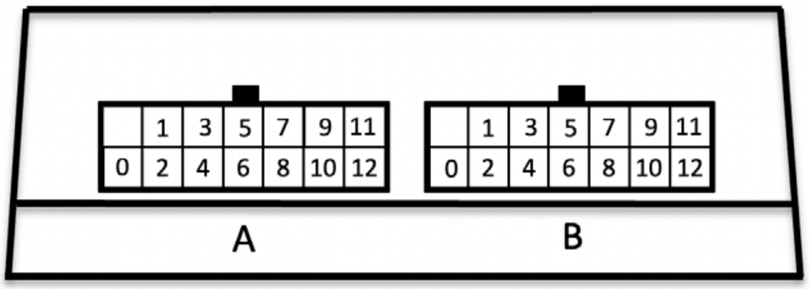


Figure 12 – BMSS Cell Harness Connectors

The BMSS supports two sets of thermistors; connectors are located on the sides of the enclosure.

[2] For datasheets, see https://www.analog.com/en/products/ltc6811-1.html, https://www.analog.com/en/products/ltc6804-1.html, https://www.analog.com/en/products/ltc6813-1.html

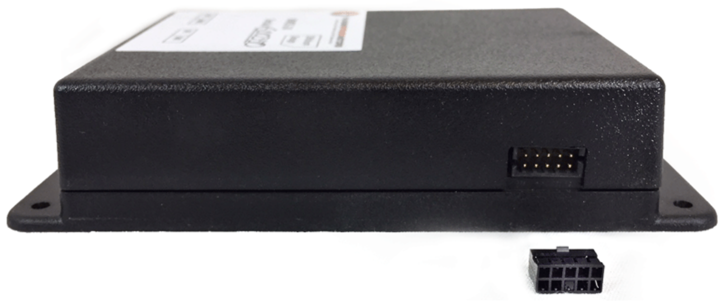


Figure 13 – BMSS Thermistor Connectors

The BMSS and BMSS18 Thermistor Connection pinouts are identical. See below, *Thermistors*.

BMSS18

The 18 cell BMSS18 has one ltc6813 18-cell measurement device and must be used with the LTC18 architecture. These devices support a single cell group of 18 cells. A cell group can consist of fewer than 18 cells, but cells must be series connected and in total result in greater than 16V. Depending on nominal cell voltage, the minimum cell group size should be at least six (or more) cells. When using a BMS architecture of LTC18, it is permissible to mix 12 cell ltc6811 BMSSs with 18 cell BMSS18 devices.

The BMSS18 has one four-pin connector that connects to two IsoSPI interfaces: signals are labeled IPO, IMO, IPI, IMI. A BMSS18 has one cell harness connector.

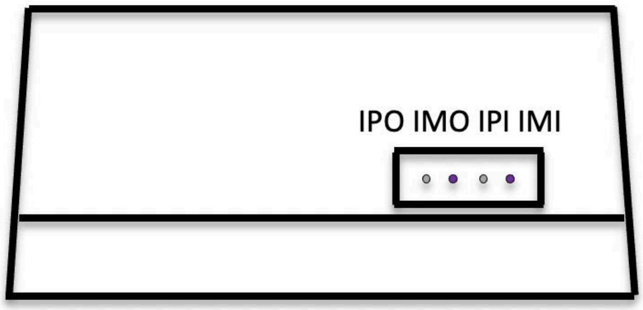


Figure 14 - BMSS18 System Connector

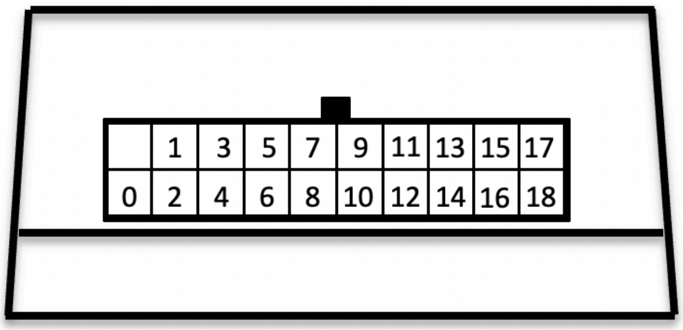


Figure 15 - BMSS18 Harness Connector

The BMSS18 has one thermistor connector and supports five thermistor connections, two pins for each thermistor. The BMSS and BMSS18 Thermistor Connection pinouts are identical. See below, *Thermistors*.

LTC Cell Group Wiring

The figure below, shows two LTCs, one connected to a Cell Group of 12 cells and a second LTC connected to a partially populated Cell Group with only 8 cells. Note that it requires 13 connections (or "wires") to connect to the 12 cells in a Cell Group. The wires can be numbered 0 to 12 and cells numbered 1 to 12. All cells in a Cell Group must be wired consecutively: the positive terminal of one cell must connect to the negative terminal of the next cell. If fewer than 12 cells are to be monitored, the unused cell inputs should be at the top, connected together to cell 12.

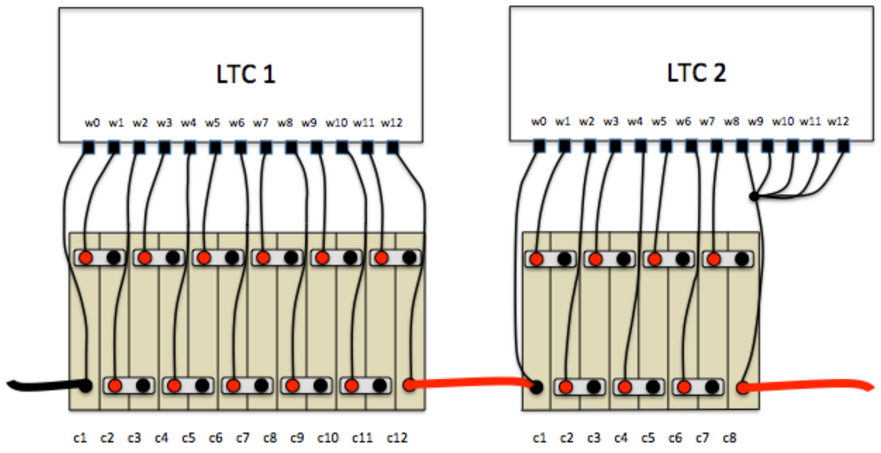

LTC 1
w0 w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12
LTC 2
w0 w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12
c1 c2 c3 c4 c5 c6 c7 c8

Figure 16 – Cell Group Wiring

A Cell Group should not "span" a circuit protection device (such as a contactor, fuse, or a circuit breaker). This guideline ensures that if the fuse blows (or contactor opens, or circuit breaker trips), then the LTC will not be destroyed. Otherwise, the LTC will span the newly open circuit and could be damaged by full pack voltage.

See the figure below for an example.

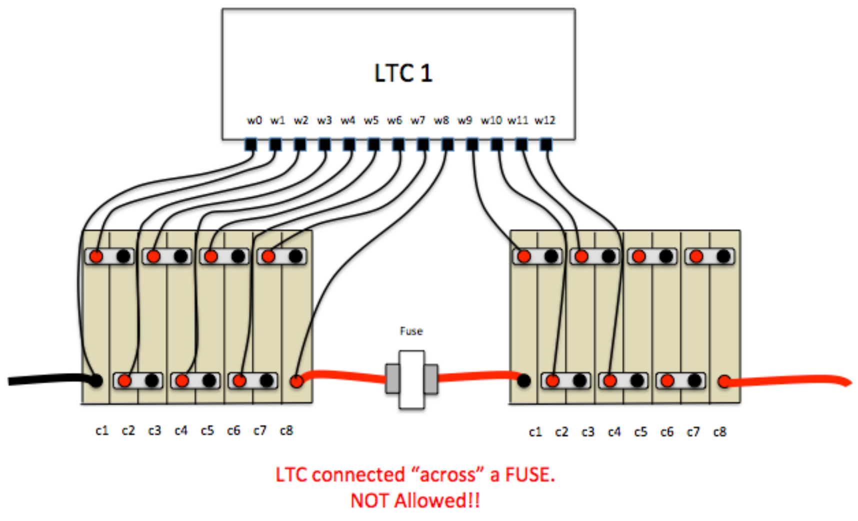

LTC 1
w0 w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11 w12
Fuse
c1 c2 c3 c4 c5 c6 c7 c8
c1 c2 c3 c4 c5 c6 c7 c8
LTC connected "across" a FUSE.
NOT Allowed!!

Figure 17 – Cell Group Wiring Spanning A Fuse

A Cell Group should not span a traction pack jumper. If, despite these guidelines, this cannot be avoided, then the Cell Group must be disconnected at the BMS before the traction jumper is disconnected. A warning label is advised.

Thermistors

Each LTC device on the BMSS or BMSS18 has an on-die thermistor and support for five external thermistors. All external thermistor connectors have the following pinout:

T5	T4	T3	T2	T1
T5	T4	T3	T2	T1

Figure 18 - Thermistor Connections

The thermistor connector is a 10p Molex 87833-1031, with the matching connector Molex 51110-1060 with crimp terminal 50394-8054. A harness with pigtail wire connections is available. This can be soldered to a wire lead thermistor or a connector may be added to allow it to connect directly to the thermistor connection on an OEM battery pack.

Harness Verification

A Cell Harness tester is available for the BMSS and BMSS18 measurement boards. Once the cell group is wired to the cell harness, a voltmeter can be used to verify that cells are connected in the proper order and with the proper polarity. The voltage difference between adjacent cells should be the voltage of a single cell in the group. If fewer than 12 cells are used in the group then the top cells should all have identical voltage readings, indicating that they are correctly bridged together at the highest cell.

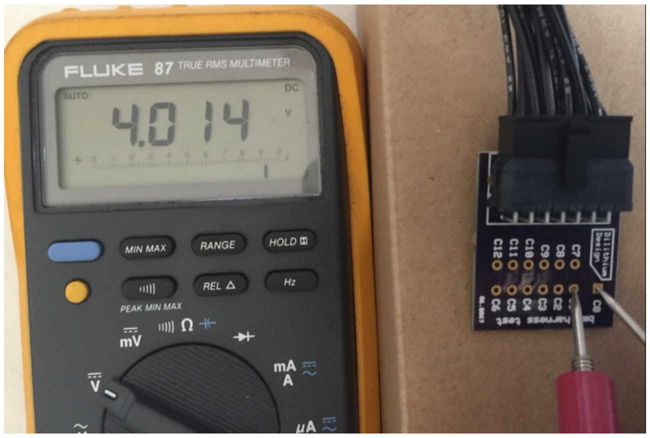


Figure 19 – Using the Cell Harness Tester

NOTE: The Harness Tester is connected directly to pack voltage and so there may be up to 60V present on the harness. When using the Harness Tester, be careful with metal tools and make sure the working surface is nonconductive.

Once the cell harness is wired and the connections verified, it is safe to plug the harness into the BMSS or BMSS18. In general, Cell Harness connectors may be plugged in or unplugged in any order.

A123

The **A123** architecture supports A123 13-cell Measurement and Balance Boards (MBBs). The command to enable this architecture is:

```
mcu> set arch a123
```

The A123 26-cell module integrates 26 cells and two MBBs into one assembly. A harness is used to connect the modules together into a series string. The harness provides 12V power and CAN. Two additional signals, “Fault” and “Id Reset”, are normally left disconnected,

MBB 12V power can be provided by the MCU, using the OUT5 output, mapped to the Output Function **power**. This will take advantage of the autostart function in the MCU and will provide 12V to the MBBs whenever the MCU is powered. When using OUT5, “always on” 12V power must be connected to the OUT5_12V pin.

```
mcu> set out5 power
```

A123 MBBs communicate with the MCU using CAN. It is recommended that the MCU CAN1 interface be used as it has higher performance than CAN2. This can be configured with the command:

```
mcu> enable a123 can1
```

A CAN network must be properly terminated with 120 ohm resistors at both ends. The MCU is provided with a CAN termination resistor installed internally on its PCB. The factory MBB harness *may* have a 120 ohm termination resistor installed at the end of the series string of modules. This should be verified, and if not, one should be added.

Each MBB has a unique "MBB Id" from 1 to 16 which is stored in MBB nonvolatile memory. The MBB Id uniquely identifies the MBB and is used in the CAN messages with the MCU.

Connections with the MBBs and their proper addressing can be verified by the BMS Probe and Diagnostics output, which is described below. Connect a serial cable to the MCU, start a serial session, and reset the MCU. A Probe operation runs automatically and prints any change in configuration. At startup, the following message might appear:

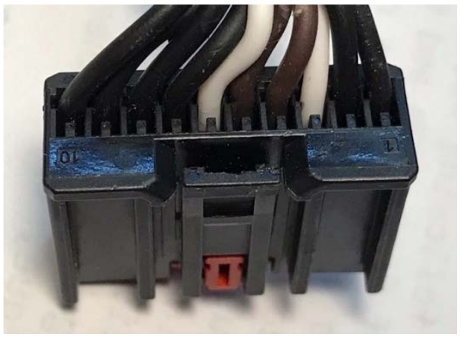
```
mbb1 detected
mbb2 detected
mbb3 detected
mbb5 detected
```

This indicate that there are four MBBs connected, with IDs 1, 2, 3 and 5.

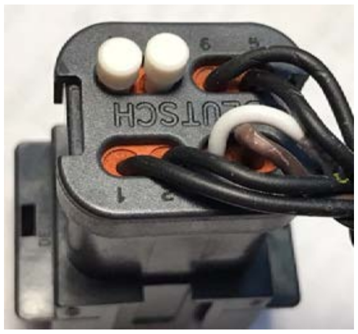
The MBBs can be renumbered using the following procedure:

1. Connect the MBB "ID Reset" line to ground.
2. Enter **`set mbbaddr 1 <n>`** (where n ia a number from 2 to 17; 1 more than the total number of MBBs in the string.
3. Disconnect the MBB "ID Reset" line.
4. Cycle power to the MCU
5. Enter **show mbb** to ensure the expected MBBs are listed.
6. Enter **show cells** to verify cell connections.
7. Enter **lock** to initiate error reporting with the found modules and cells.

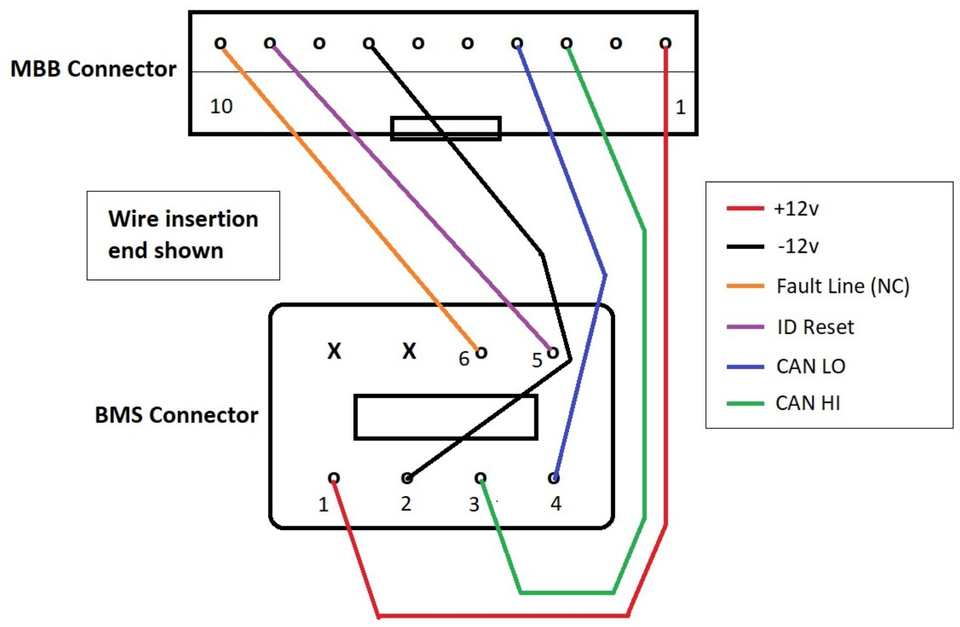
A123 26s3p 60Ah Module Connections



MBB Connector



BMS Connector



Pin #	Pin Name	Description
1	12V_PWR (In)	12V power.
2	12V_PWR (Out)	12V power for downstream MBBs.
3	CAN_H (In)	CANH twisted with CAN_L (In)
4	CAN_L (In)	CANL twisted with CAN_H (In)
5	CAN_L (Out)	CANL for downstream MBBs twisted with CAN_H (Out)
6	CAN_H (Out)	CANH for downstream MBBs twisted with CAN_L (Out)
7	12V_GND (In)	GND
8	12V_GND (Out)	GND for downstream MBBs
9	FLT_IN	Fault / addressing line provided by downstream MBB FLT_OUT or CSM heartbeat output. 5V logic levels.
10	FLT_OUT	Fault / addressing line provided to upstream MBB FLT_IN or BCM fault input. Output requires a 5V pull-up through 10kΩ, which is provided by upstream MBB FLT_IN or BCM FLT_IN.

Figure 20 - MBB Connector Pinout

Probe and Diagnostics

Upon startup and periodically afterwards, the MCU performs a “probe” to determine how many measurement devices are present, and diagnostics are performed on all discovered devices. The results of discovery and diagnostics are printed at the serial port.

For the **LTC12** and **LTC18**, each LTC detected will be assigned an address from A1 to A8 or B1 to B8, depending upon which IsoSPI interface is used. The LTC that is wired directly to the MCU IsoSPI A interface will be numbered A1.

The following example shows a 48-cell pack wired as one series string of cells. The 48 cells are managed by two 24-cell BMSSs; each BMSS manages two cell groups as it has two 12-cell LTCs. MCU IsoSPI interface A is wired to the first BMSS, which is then daisy-chained to the second BMSS.

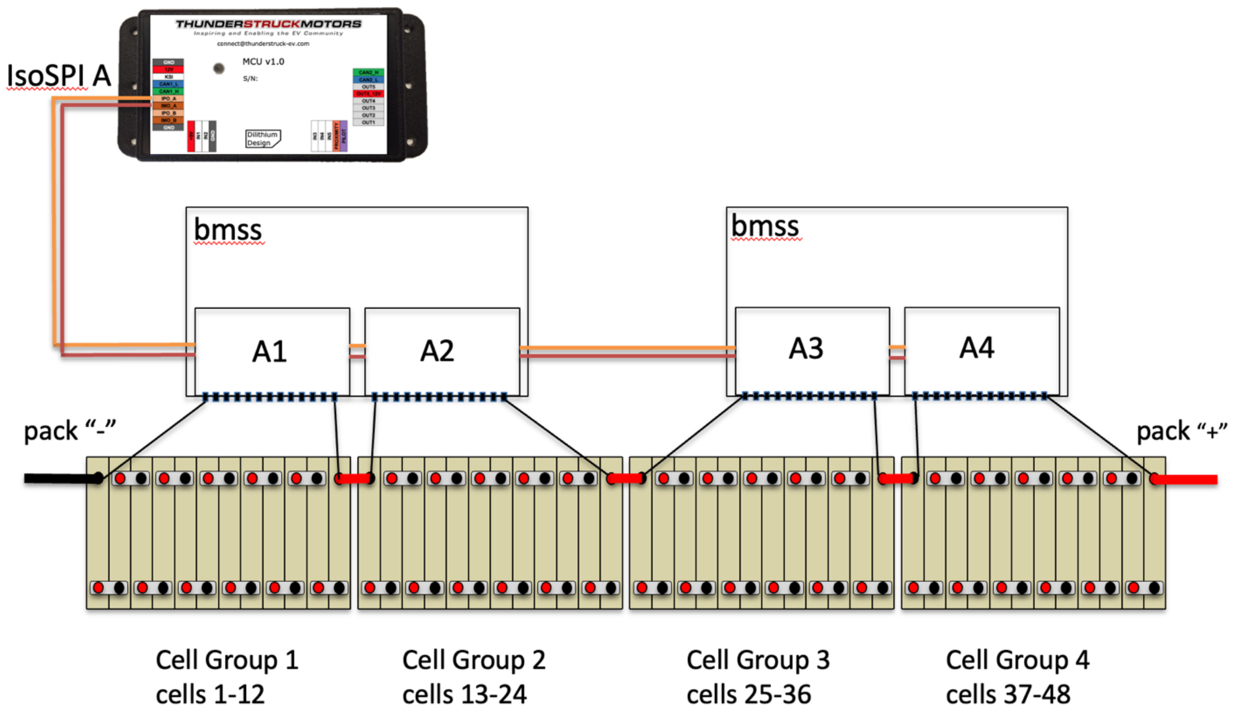


Figure 21 – 48 Cell Pack Wiring

In this case, four LTCs will be discovered by the MCU and be assigned the addresses A1, A2, A3 and A4, in the order that the links are connected. The following output will show at the serial port upon powerup:

```
ltc A1 detected
ltc A2 detected
ltc A3 detected
ltc A4 detected
```

The **show ltc** command can be used to retrieve the results of diagnostics, the chip type and revision, values of different internal voltages, and the total voltage measured by this device.

```
mcu> show ltc
 ltc| status |  rev |  Van  |  Vdig |  Vref |  SOC  |  SUM   |  conf  |  disc
----|--------|------|-----------------------------------------------------------
 A1 |   ok   |6811-6| 5.092v| 3.205v| 3.006v| 38.16v  38.15v | 0x00000  0x00fff
 A2 |   ok   |6811-6| 5.127v| 3.204v| 3.003v| 38.17v  38.16v | 0x00000  0x00fff
 A3 |   ok   |6811-6| 5.043v| 3.214v| 3.004v| 38.21v  38.20v | 0x00000  0x00fff
 A4 |   ok   |6811-6| 5.089v| 3.208v| 3.003v| 38.22v  38.21v | 0x00000  0x00fff
```

If there is a diagnostic failure, it will be reported in this initial discovery message, will be shown in the status field of the **show ltc** command and will generate a HARDWARE alert.

```
ltc A3 detected, diagnostic failure=0x1000
```

The Probe and Diagnostic procedure also occurs for the A123 MBBs, however addressing is different and the diagnostics are not as extensive. Discovered MBBs are printed:

```
mbb1 detected
mbb2 detected
mbb3 detected
mbb5 detected
```

The **show mbb** command can be used to retrieve the MBBs that have been discovered. This listing shows the MBB address, the minimum, maximum, and mean cell voltage in the MBB, as well as additional debugging information.

```
mcu> show mbb
 mbb| status |  min  |  max  |  mean | flt_xx |  conf  |  disc
----|--------|-------|-------|-------|-------------------------
  1 |   OK   | 3.119v| 3.142v| 3.133v| in     |0x00000  0x01fff
  2 |   OK   | 3.110v| 3.144v| 3.132v| in     |0x00000  0x01fff
  3 |   OK   | 3.126v| 3.189v| 3.177v| in     |0x00000  0x01fff
  5 |   OK   | 3.126v| 3.189v| 3.177v| in     |0x00000  0x01fff
```

Pack, Group, and Cell Numbering

Group and Cell Numbering

By default, the MCU assumes that all cell groups are series connected and assigns cell numbers depending upon the LTC or MBB address. A pack with four 12-cell LTCs will number the cells 1 to 48 by default.

The numbering and the cell values can be shown by the commands **show cmap** (e.g., “show cell map”) and **show cells**.

```
bms> show cmap
 ltc|pack|group| cells
----|----|-----|--------------------------------
 A1 |  1 |  1  |(c1 -c12) . . . . . . . . . . . .
 A2 |    |  2  |(c13-c24) . . . . . . . . . . . .
 A3 |    |  3  |(c25-c36) . . . . . . . . . . . .
 A4 |    |  4  |(c37-c48) . . . . . . . . . . . .

mcu> sh cells
 c1  - 3.177v        c13 - 3.177v        c25 - 3.175v        c37 - 3.183v
 c2  - 3.179v        c14 - 3.179v        c26 - 3.180v        c38 - 3.186v
 c3  - 3.181v        c15 - 3.181v        c27 - 3.179v        c39 - 3.187v
 c4  - 3.175v        c16 - 3.179v        c28 - 3.180v        c40 - 3.181v
 c5  - 3.189v        c17 - 3.179v        c29 - 3.180v        c41 - 3.184v
 c6  - 3.175v        c18 - 3.176v        c30 - 3.177v        c42 - 3.185v
 c7  - 3.178v        c19 - 3.181v        c31 - 3.179v        c43 - 3.183v
 c8  - 3.175v        c20 - 3.176v        c32 - 3.212v        c44 - 3.184v
 c9  - 3.176v        c21 - 3.178v        c33 - 3.179v        c45 - 3.184v
 c10 - 3.179v        c22 - 3.179v        c34 - 3.182v        c46 - 3.181v
 c11 - 3.178v        c23 - 3.180v        c35 - 3.183v        c47 - 3.184v
 c12 - 3.175v        c24 - 3.178v        c36 - 3.182v        c48 - 3.176v
```

It is possible to modify the default numbering. It might be more convenient, for example, to connect the LTC IsoSPI links in a different order than the connection order of the cell group connections. The figure below shows an example. This is a 44-cell pack where the LTC IsoSPI links are connected to Cell Groups without regard to cell order. LTC A4, for example, monitors cells 1 to 12.

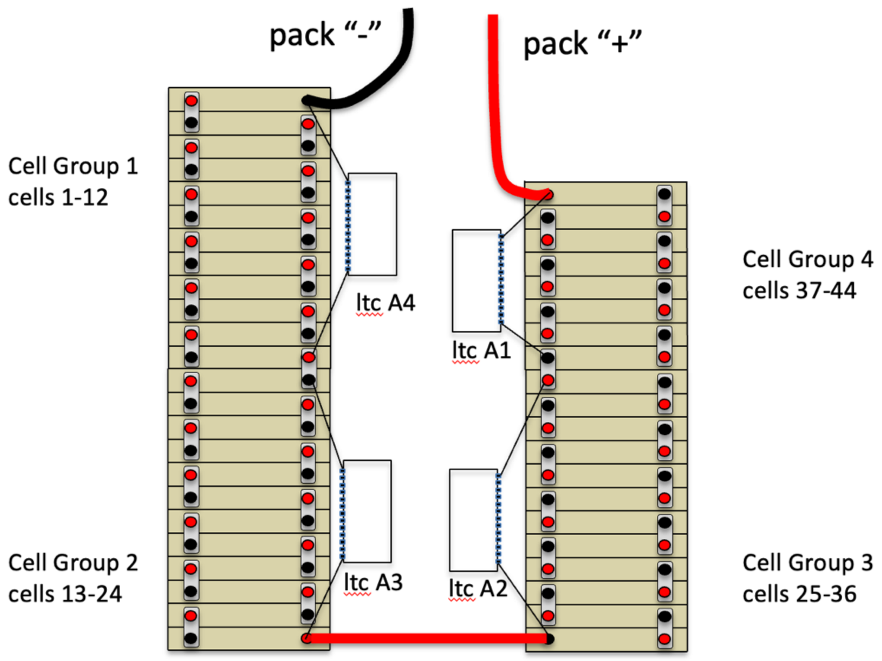


Figure 22 - 44 Cell Pack, Cell Groups Connected Where Convenient

The default numbering for an LTC can be changed using the **`set cmap`** command. This command takes three arguments: the LTC address, the pack, and the group. The mapping can be changed to renumber the cells with the following commands:

```
bms> set cmap A1 1 4
bms> set cmap A2 1 3
bms> set cmap A3 1 2
bms> set cmap A4 1 1
```

In this example, there is only one pack (pack 1). The results of this mapping can be verified by using the **`show cmap`** command.

```
bms> show cmap
 ltc|pack|group| cells
----|----|-----|---------------------------------
 A4 |  1 |  1  |(c1 -c12) . . . . . . . . . . . .
 A3 |    |  2  |(c13-c24) . . . . . . . . . . . .
 A2 |    |  3  |(c25-c36) . . . . . . . . . . . .
 A1 |    |  4  |(c37-c48) . . . . . . . .
```

Note that the cell group monitored by ltc A1 now consists of cells c37 to c48.

If the **A123** architecture is used, the default numbering for an MBB can also be changed using the **`set cmap`** command. By default, the cells are numbered 1 to 13 for mbb1, 14 to 26 in mbb2, and so on. In order to swap these two groups, the following commands can be used:

```
bms> set cmap 1 1 2
bms> set cmap 2 1 1
```

This maps mbb1 to pack 1, group 2 (cells 14 to 26), and mbb2 to pack 1 group 1 (cells 1 to 13).

Pack Numbering

The MCU also supports configurations of multiple parallel strings of cells. In the MCU, each parallel string of cells is called a **pack**. This is shown in the figure below.

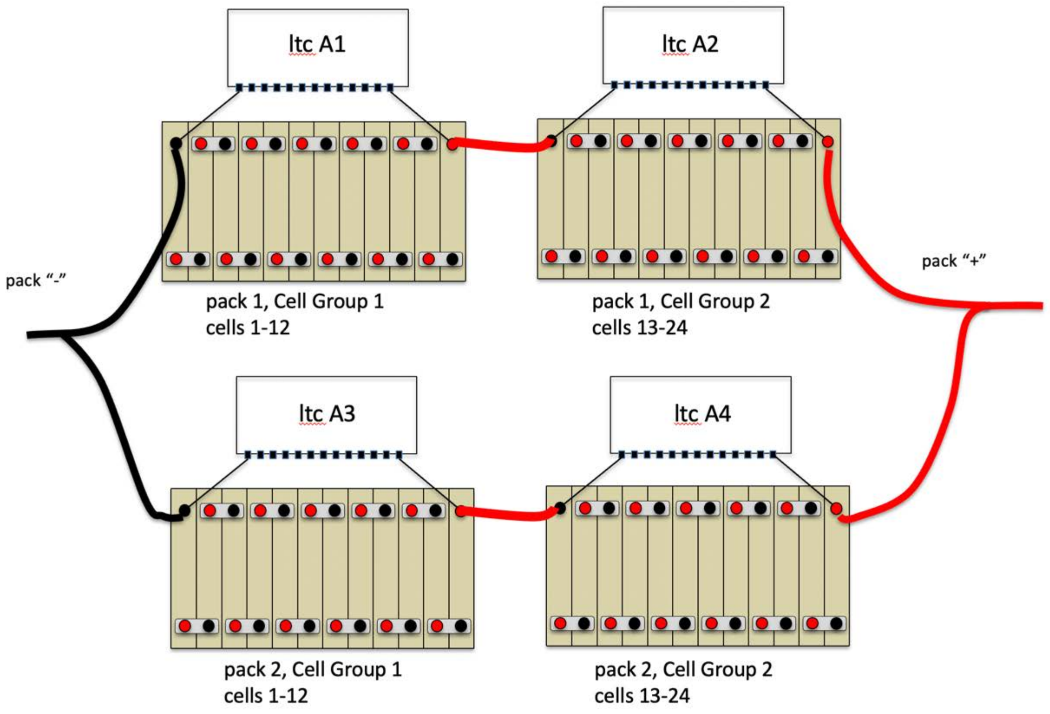


Figure 23 – Parallel Packs

Use the following commands to map LTC A3 to Pack 2, Cell Group 1 and LTC A4 to Pack 2, Cell Group 2:

```
bms> set cmap a3 2 1
bms> set cmap a4 2 2
```

Check the configuration with **`show cmap`**. Now there are now two packs.

```
bms> show cmap
 ltc|pack|group| cells
----|----|-----|----------------------------------
  1 |  1 |  1  |(c1 -c12) . . . . . . . . . . . .
  2 |    |  2  |(c13-c24) . . . . . . . . . . . .
----|----|-----|----------------------------------
  3 |  2 |  1  |(c1 -c12) . . . . . . . . . . . .
  4 |    |  2  |(c13-c24) . . . . . . . . . . . .
```

Here is example output of **`show cells`** with two packs.

```
bms> show cells
pack 1 ----------
 c1 - 3.646v ++      c13- 3.630v ++
 c2 - 3.477v +       c14- 3.476v +
 c3 - 3.547v +       c15- 3.553v +
 c4 - 3.478v +       c16- 3.468v
 c5 - 3.534v +       c17- 3.540v +
 c6 - 3.469v         c18- 3.467v
 c7 - 3.352v         c19- 3.311v
 c8 - 3.234v         c20- 3.223v
 c9 - 3.209v        c21- 3.222v
 c10- 3.206v        c22- 3.230v
 c11- 3.216v        c23- 3.229v
 c12- 3.222v        c24- 3.212v
pack 2 ----------
 c1 - 3.249v        c13- 3.143v -
 c2 - 3.316v        c14- 3.082v -
 c3 - 3.352v        c15- 3.311v
 c4 - 3.480v +      c16- 3.321v
 c5 - 3.052v --     c17- 3.366v
 c6 - 3.278v        c18- 3.394v
 c7 - 3.199v        c19- 3.424v
 c8 - 3.325v        c20- 3.308v
 c9 - 3.068v -      c21- 3.315v
 c10- 3.370v        c22- 3.082v -
 c11- 3.067v -      c23- 3.090v -
 c12- 3.282v        c24- 3.075v -
```

Cell Measurement

In operation, all cells in the system are measured approximately 8 times a second. Cell High Voltage Cutoff (HVC) and Low Voltage Cutoff (LVC) condition states are reevaluated every cell scan and pack-based and cell-based statistics are collected.

A cell census is performed to ensure all cells that are supposed to be there ("configured"), really are there ("discovered"). Cells are discovered automatically; cells are configured using the **`lock`** command.

Cell Census

There are two alerts associated with the cell census: the `NOT_LOCKED` alert and the `CELL_CENSUS` alert.

The **`show cmap`** command gives a list of all cells. Each cell has four possible entries. The following is an example of **`show cmap`** before the **`lock`** command has been executed. The list shows 44 cells are discovered (but not configured):

```
bms> sh cmap
 ltc|pack|group| cells
----|----|-----|-----------------------------------
 A1 |  1 |  1  |(c1  -c12 ) . . . . . . . . . . . .
 A2 |    |  2  |(c13 -c24 ) . . . . . . . . . . . .
 A3 |    |  3  |(c25 -c36 ) . . . . . . . . . . . .
 A4 |    |  4  |(c37 -c48 ) . . . . . . . .
```

The "dots" indicate that the cells are detected but not configured. The **`lock`** command marks all currently discovered cells as "configured" and clears the `NOT_LOCKED` alert.

```
bms> lock
bms> sh cmap
 ltc|pack|group| cells
----|----|-----|------------------------------------
 A1 |  1 |  1  |(c1  -c12 ) X X X X X X X X X X X X
 A2 |    |  2  |(c13 -c24 ) X X X X X X X X X X X X
 A3 |    |  3  |(c25 -c36 ) X X X X X X X X X X X X
 A4 |    |  4  |(c37 -c48 ) X X X X X X X X
```

When the configuration is locked, cells will display with an "X".

The MCU will periodically probe all LTCs and monitor if LTCs or cells arrive or disappear. Suppose, for example, that the harness for LTC A4 is disconnected. The MCU will detect this condition and report the following:

```
ltc A1 detected
ltc A2 detected
ltc A3 detected
ltc A4 NOT DETECTED!
ltc A4 CELL CENSUS ALERT!
```

The MCU will assert the `CELL_CENSUS` alert and the results of **`show cmap`** will indicate the missing cells.

```
bms> show cmap
 ltc|pack|group| cells
----|----|-----|----------------------------------
 A1 |  1 |  1  |(c1 -c12) X X X X X X X X X X X X
 A2 |    |  2  |(c13-c24) X X X X X X X X X X X X
 A3 |    |  3  |(c25-c36) X X X X X X X X X X X X
 A4 |    |  4  |(c37-c48) ? ? ? ? ? ? ? ?
```

If the harness is then modified and two additional cells are connected, when the harness gets reconnected, the output would look like:

```
bms> show cmap
 ltc|pack|group| cells
----|----|-----|----------------------------------
 A1 |  1 |  1  |(c1 -c12) X X X X X X X X X X X X
 A2 |    |  2  |(c13-c24) X X X X X X X X X X X X
 A3 |    |  3  |(c25-c36) X X X X X X X X X X X X
 A4 |    |  4  |(c37-c48) X X X X X X X X . .
```

Although the configuration has been locked already, it can be locked again to record this new configuration.

```
bms> lock
bms> show cmap
 ltc|pack|group| cells
----|----|-----|--------------------------------
 A1 |  1 |  1  |(c1 -c12) X X X X X X X X X X X X
 A2 |    |  2  |(c13-c24) X X X X X X X X X X X X
 A3 |    |  3  |(c25-c36) X X X X X X X X X X X X
 A4 |    |  4  |(c37-c48) X X X X X X X X X X
```

HVC and LVC Conditions

The High Voltage Cutoff (HVC) and Low Voltage Cutoff (LVC) thresholds are configurable, using the parameters **hvc** and **lvc**. In general, when any cell voltage exceeds **hvc** or falls lower than **lvc**, an HVC or LVC alert is generated, and the associated Output Functions **hvc** and **lvc** are set TRUE. Alerts can be configured to operate an external relay or indicator to notify the driver or disable the EV by mapping the appropriate Output Function to an Output.

Reporting these alerts can be modified by the optional parameters **hvcdelay** and **lvcdelay**. These parameters, given in seconds, allow for a delay in reporting the HVC or LVC alert. The **lvcdelay** parameter can be used to minimize LVC alarms under a short period of high current draw.

The logic necessary to clear these conditions can be modified by the optional parameters **hvcc** and **lvcc**. Using these parameters, the HVC or LVC alerts are reported normally, but the alert is not cleared until the cell voltage drops below **hvcc** or rises above **lvcc**. The **hvcc** parameter would typically be a bit less than the **hvc** setting; the parameter allows hysteresis in reporting. (Example usage: In a solar application, the charger may always be connected to the pack and the HVC alert is used to stop the charger. Without hysteresis, the charger will cycle ON and OFF as a cell voltage exceeds **hvc** only to drop back quickly once charging current stops).

When an HVC condition is reported or cleared, a message will be sent to the serial port. This information includes the pack and cell number:

```
1/c25: in HVC !
1/c26: in HVC !
1/c25: OK
1/c26: OK
1/c12: in LVC !
1/c11: in LVC !
```

HVC and LVC conditions are available in the output to **show cells**. In this example, **hvc** is set to 4.1 and **lvc** is set to 3.0.

```
bms> show cells
 c1  - 4.059v        c13 - 4.060v        c25 - 4.066v        c37 - 4.074v
 c2  - 4.062v        c14 - 4.062v        c26 - 4.071v        c38 - 4.077v
 c3  - 4.063v        c15 - 4.064v        c27 - 4.070v        c39 - 4.079v
 c4  - 4.058v        c16 - 4.062v        c28 - 4.071v        c40 - 4.072v
 c5  - 4.169v >HVC   c17 - 4.060v        c29 - 2.970v <LVC   c41 - 4.074v
 c6  - 4.055v        c18 - 4.056v        c30 - 4.067v        c42 - 4.075v
 c7  - 4.061v        c19 - 4.064v        c31 - 4.070v        c43 - 4.075v
 c8  - 4.058v        c20 - 4.059v        c32 - 4.203v >HVC   c44 - 4.076v
 c9  - 4.059v        c21 - 4.061v        c33 - 4.070v        c45 - 4.076v
 c10 - 4.062v        c22 - 4.063v        c34 - 4.073v        c46 - 4.073v
 c11 - 4.058v        c23 - 4.061v        c35 - 4.072v        c47 - 4.075v
 c12 - 4.055v        c24 - 4.058v        c36 - 4.072v        c48 - 4.066v
```

The `HVC` and `LVC` alerts are available at the mcu **`show`** command, are visible as LED blinking, and they are available as Output Functions.

Pack Statistics

Mean and Standard Deviation are recalculated after every cell scan and are available at the bms **`show`** command.

```
bms> show
  pack
    voltage : 228.72v
    cells   : 72
    mean    : 3.035v
    std dev : 0.364v
  alerts    : none
bms>
```

Cell Statistics

Standard Deviation is a measure of how well balanced the cells are … the lower the better.

For each cell, the Standard Deviation Multiple (SDM) is calculated. If the SDM is 1, then the cell voltage is "one standard deviation from the mean". SDM is stored as a number from -9.9 to +9.9.

Example:

```
      if
             SD           = 0.020v    // value of standard deviation
             ave          = 3.800v    // the average cell voltage
             c8           = 3.820v    // current voltage of cell c8
             c9           = 3.790v    // current voltage of cell c9
      then
             SDM(c8)      =  1.0      // cell c8 is ave + 1.0*SD
             SDM(c9)      = -0.5      // cell c9 is ave – 0.5*SD
```

Positive values of SDM mean the cell voltage is higher than average, and negative values of SDM means the cell voltage is lower than average. The SDM is used to flag especially high (+), especially high (++), low (-), or especially low (--) cells in the **`show cells`** command.

```
bms> show cells
 c1 - 3.646v ++     c13- 3.630v ++
 c2 - 3.477v +      c14- 3.476v +
 c3 - 3.547v +      c15- 3.553v +
 c4 - 3.478v +      c16- 3.468v
 c5 - 3.534v +      c17- 3.540v +
 c6 - 3.469v        c18- 3.467v
 c7 - 3.352v        c19- 3.311v
 c8 - 3.234v        c20- 3.223v
 c9 - 3.209v        c21- 3.222v
 c10- 3.206v        c22- 3.230v
 c11- 3.216v        c23- 3.229v
 c12- 3.222v        c24- 3.212v
```

The BMS records "low watermark" and "high watermark" cell values (e.g., the lowest and highest values for the cell), as well as the minimum and maximum values of SDM. The difference between the maximum and minimum SDM (the "delta") can give an idea of cell performance under load: the larger the "delta" the more nonlinear the cell.

Cell statistics are displayed with the command **`show stats`**. The statistics are cleared on powerup, however they are also cleared if the Probe discovers a change in how many cell groups are present, and can be reset using the command **`reset stats`**.

```
bmsc> show stats
total|-mean cell voltage------|-standard deviation----------------
     | 3.414v                 |  0.261v
pack1|-voltage----min----max--|----deviation-----min---max--delta-
 c1  | 3.447v   3.392v 3.520v |  0.033v  +0.1s  +0.1s +0.1s  0.0s
 c2  | 3.400v   3.345v 3.471v | -0.014v  +0.0s  +0.0s +0.0s  0.0s
 c3  | 3.473v   3.417v 3.546v |  0.059v  +0.2s  +0.2s +0.2s  0.0s
 c4  | 3.414v   3.359v 3.486v |  0.000v  +0.0s  +0.0s +0.0s  0.0s
 c5  | 3.463v   3.407v 3.536v |  0.049v  +0.1s  +0.1s +0.1s  0.0s

<etc>
```

Cell Balancing

Automatic cell balancing by discharging is enabled using the command **`enable balance`**. If enabled, cell discharging runs whenever the MCU is powered up, whether the car is being driven or being charged. A minimum cell balancing voltage **`bvmin`** can be configured: a cell with a voltage lower than this value will not be balanced.

Cell Balancing – LTC12 and LTC18

For LTC devices, the MCU determines one cell from each group of six cells to balance. Once a cell has been chosen for discharge, its associated shunting resistor is enabled. BMSS and BMSS18 hardware supports 24 ohm, 1W balancing resistors, which will shunt about 170ma at 4V. The best candidate for balancing will have a cell voltage that is:

- the highest voltage of any other cell in its six cell group,
- higher than the configured **`bvmin`** value, and
- higher than the pack mean voltage + 2.5mv.

Approximately once a minute, discharge candidates are reevaluated, new discharge candidates are determined, and then charge shunting is resumed.

Cell Balancing – A123

A123 MBBs contain integrated balancing hardware. A123 MBBs will balance any cell with a voltage higher than a specified balance target. A balance target is calculated by the MCU that is:

- 3.0mv below the voltage of the highest cell in the pack,
- 3.0mv above the average voltage of the pack, and
- higher than the configured **`bvmin`** value

The MCU has been designed to work with a 1% 10K NTC thermistor, with a "B value" of 3435K. One possible choice is the Vishay NTCLE413E2103F520.

By default, thermistor operation is disabled and no thermistor alerts will be reported.
The **`show thermistor`** command shows thermistor status. The following example shows four LTC devices and their thermistor readings. In this example, die thermistor readings are shown for all LTCs, but only LTC A2 has external thermistors connected. The readings all show in parenthesis because the thermistors have been detected but not enabled.

```
sys> sh th
 ltc|  die  |   T1  |   T2  |   T3  |   T4  |   T5
----|-----------------------------------------------
 A1 |  (30C)
 A2 |  (30C)   (24C)   (24C)   (23C)   (25C)
 A3 |  (39C)
 A4 |  (30C)
```

The **`enable thermistor all`** command will enable all discovered thermistors. There are also commands to enable all die thermistors "**`en th die`**", all external thermistors "**`en th ext`**", and all thermistors on a particular device "**`en th A1`**", as well as commands to enable and disable individual thermistors.

```
bms> enable thermistor all
bms> sh th
ltc|   T0  |   T1  |   T2  |   T3  |   T4   |   T5
----|------------------------------------------------
 A1 |   25C
 A2 |   22C            16C     16C     15C     16C
 A3 |   22C
 A4 |   22C
```

A thermistor that is "enabled" will report alerts. There are three alerts associated with thermistors: `THERM_CENSUS`, `THERM_OVERTEMP`, and `THERM_UNDERTEMP`.

If an enabled thermistor is disconnected and not detected, this will result in serial output and in the `THERM_CENSUS` alert.

```
mcu> ltc A2:T2 NOT DETECTED!
ltc A2:T3 NOT DETECTED!
ltc A2:T4 NOT DETECTED!
ltc A2:T5 NOT DETECTED!
```

The thermistor maximum and minimum threshold temperatures are set by the configuration parameters **`thmax`** and **`thmin`**.

If a thermistor is enabled and if the measured thermistor value is greater than **`thmax`** (or less than **`thmin`**) an alert is raised and a message is printed. If, for example, **`thmax`** is set to 40, and the thermistor temperature reaches 41C, the following message will be printed:

```
ltc A1:T2 41C, OVERTEMP!
```

ONBOARD CHARGING

With onboard charging, one or more chargers are typically installed in the EV and AC Line Power is connected to the chargers using an EV charge plug. The MCU monitors for charge plug insertion to start a charge, and once charging begins, the MCU directly controls the chargers, typically by using CAN messages.

The MCU supports the J1772 Type 1 and J1772 Type 2 charge protocols. These protocols define the function of the Electric Vehicle Service Equipment (EVSE) which provides the charge plug. J1772 Type 2 optionally supports three-phase line power. In this case, chargers can be individually wired to different phases.

The MCU also supports a “direct plug-in” option, which allows the user to directly connect line power to the charger without using a J1772 (or J1772T2) EVSE. The three options (**`J1772`**, **`J1772T2`**, and **`direct`**) are mutually exclusive and are set by using the **`set plug`** command. The MCU also supports onboard charging using the Zero EV CCS; this case is handled separately: see *ZEV CCS Charging*, below.

The MCU has a sleep and autostart feature that uses the **KSI** and **Proximity** inputs. If the MCU is not charging, the MCU will enter a “sleep” mode if **KSI** is not present. The **Proximity** input has a threshold detector that can wake the MCU. In the case of J1772 and J1772T2 charging, plugging in the charge cable will trigger the proximity detector and wake up the MCU.

The sleep and autostart functions can be used for non J1772/J1772T2 charging applications. In this case, **KSI** is wired to the Key Switch as before. The autostart function can be initiated by applying a ground the **Proximity** input in order to wake the MCU and start the charge. Similarly, releasing the ground on **Proximity** will stop the charge.

The sleep and autostart functions can be disabled by connecting the **KSI** input tied to **12V**; the MCU can be powered down by disabling power to both the 12V and KSI inputs.

“Time of use” charging is supported, in which case, **Proximity** charge plug detection will wake the MCU, but the MCU can be configured to wait several hours for the EVSE to become active.

Charge voltage, charge current, and maximum charge time are configured in the MCU and used during charging to control the chargers.

The MCU supports several CAN-enabled chargers, including the Thunderstruck TSM2500 and Elcon chargers. The MCU supports up to four parallel chargers for faster charging. When multiple chargers are configured, each charger is individually CAN addressed. Work is divided evenly between the chargers and statistics are gathered and recorded on each charger separately.

The MCU supports three step charging: a Bulk charge, a Finishing charge, and a Float charge. The Bulk charge phase is used by all battery cell chemistries. The Finishing charge is optional and typically used by Lead Acid batteries. The Float (or "trickle") charge, also optional, is used to maintain pack charge for an extended period of storage.

It is the responsibility of the user to configure charge parameter values appropriate for the pack.

> WARNING: Batteries can be dangerous if overcharged and it is strongly recommended that the user check with their battery supplier to determine appropriate charging parameters.

Up to four "charge profiles" can be created for onboard charging: each profile contains a separate copy of all necessary parameters for charging. In operation, a "profile selection" switch is defined and connected to an MCU input; the switch setting determines which profile is used.

Charging normally completes when the charging current drops below a minimum configured value. The MCU will also stop charging if an error condition occurs, such as: the charge plug becomes unplugged, a cell High Voltage Cutoff error occurs, loss of communication between the MCU and Charger, or maximum configured charge time is reached, among other conditions.

There are two Output Functions relevant to charging: the **charge** output is TRUE when charging, and the **driveok** output is TRUE when the charge plug is unplugged.

The serial interface has commands to monitor charging progress and retrieve charge history.

Charge Plugs and Line Powering

J1772 Type 1

SAE J1772 defines the physical connector and protocols used between the charging station or "Electric Vehicle Service Equipment" (EVSE), and the Electric Vehicle (EV). This standard is widely used and provides single phase 220V 30-50A or 110V at 15A to an EV onboard charger.

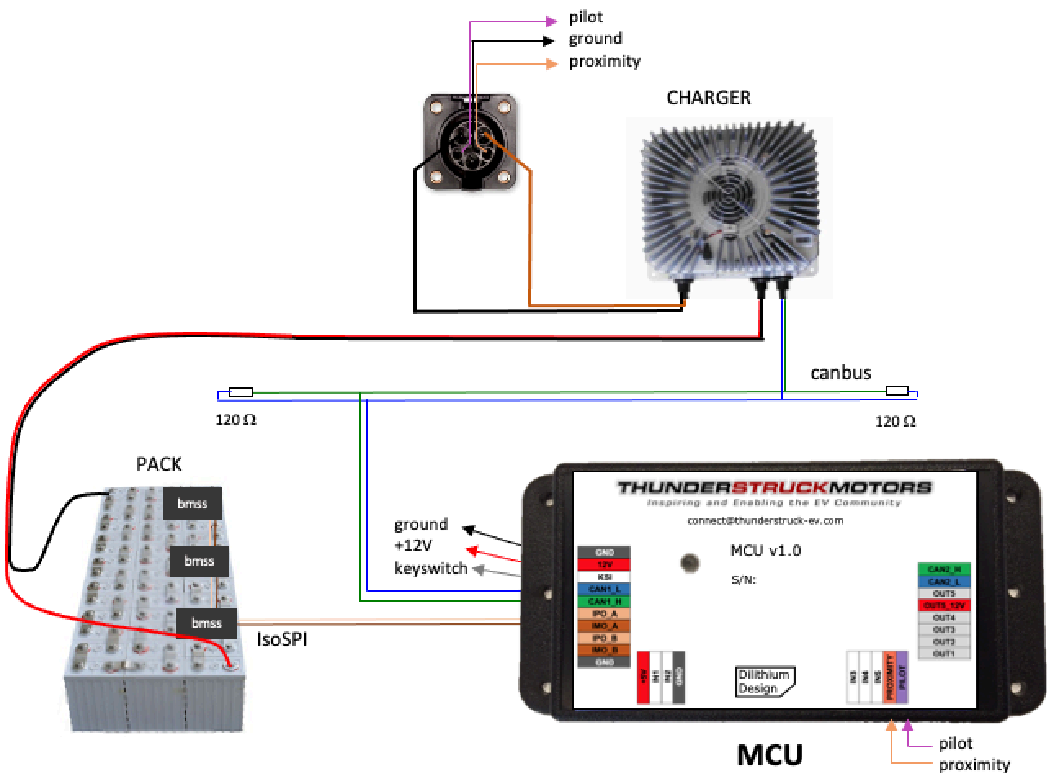


Figure 24 – J1772 Type 1 System Diagram

The system diagram, above, shows the main components of a J1772 Type 1 charging system, including the charge plug, the charger, the EV pack, and the MCU. Also shown are the CAN bus used to communicate with the charger and connections to BMS measurement modules.

The J1772 Type 1 connector has five conductors: three conductors for line power, a **Proximity** signal, and a **Pilot** signal. **Proximity** is used to determine if the charger plug is connected and is wired directly to the proximity input on the MCU. **Pilot** has a dual function: it is used by the EVSE to advertise how much charge current is available, and it is used by the EV to enable or disable charging. It is wired directly to the Pilot input on the MCU.

The following diagram shows the connections to the J1772 Type 1 EV connector:

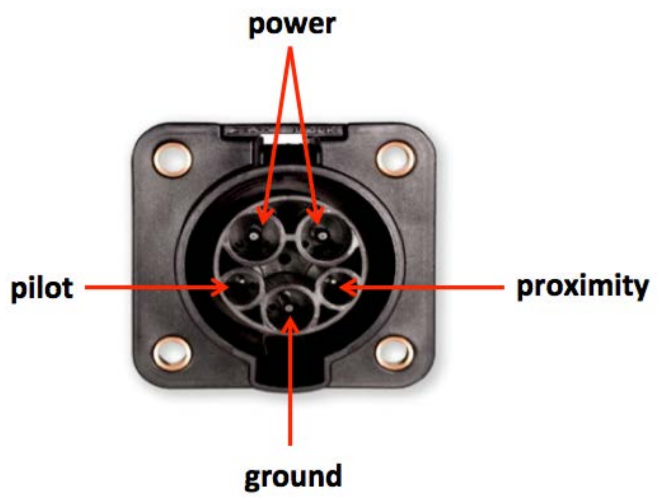


Figure 25 – J1772 Type 1 EV Connector

The J1772 Type 1 charge plug is a latching plug with a charger release button. Resistors in the charge plug are connected to the charger release button, to **Proximity**, and to **Ground**. The resistance between **Proximity** and **Ground** allows the EV to determine if the plug is "disconnected", "connected", or "locked". Expected readings are:

Proximity	Plug State
Inf	disconnected
480 Ω	connected
150 Ω	locked

The **Proximity** signal provides important safety features. Should the plug become unlocked while charging, the MCU can detect this and will immediately stop charging. This helps prevent arcing at the contacts if the plug is suddenly removed. The MCU supports the Output Function (**driveok**) that indicates when the plug is connected; this signal can be used to prevent the EV from being driven if the charge plug is connected.

The **Pilot** signal allows the EVSE to advertise how much line current is available for charging. The EVSE indicates this by placing a 1KHz square wave on the Pilot signal; the duty cycle of the square wave indicates the available charging current. If the duty cycle is <=50%, the line current in amps is the duty cycle times 6/10. As an example: a duty cycle of 50% advertises 30A of line current.

The **Pilot** signal also allows the EVSE to detect the presence of the EV and allows the EV to enable and disable charging. The EV provides a series diode and 2.7K resistor to ground and bridges an additional 1.3K resistor to ground to enable charging.

IMPORTANT: When wiring the J1772 signals, ensure that there is a good ground connection between the EV Chassis, J1772 Ground, and MCU GND. A poor ground connection can result in erratic operation of the J1772 circuitry.

For more information on J1772 see http://en.wikipedia.org/wiki/SAE_J1772

J1772 Type 2

The International Electrotechnical Commission (IEC) has adopted and extended the SAE J1772 standard for use internationally. In its IEC 62196 standard, "J1772 Type 1" and J1772 Type 2" protocols are defined. J1772 Type 1 uses the same plug and signaling protocols as SAE J1772.

J1772 Type 2 uses a different plug which allows the use of either one- or three-phase power, common in Europe. If only one-phase power is available, a simple adapter cable can be used that mechanically converts from one connector to the other. The J1772 Type 2 connector is commonly known as the "Mennekes" connector. The figure below shows the J1772 Type 2 connector.

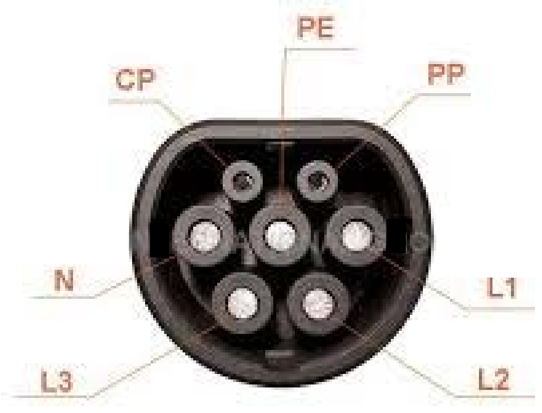


Figure 26 – J1772 Type 2 Connector

The following table lists the J1772 Type 2 signals and the correspondence to J1772 Type 1 signals.

J1772 T2 Abbreviation	J1772 T2 Signal	J1772 Signal
PP	Plug Present	Proximity
CP	Control Pilot	Pilot
PE	Protective Earth	Ground
N	Neutral	Power
L1	Phase 1	Power
L2	Phase 2	n/a
L3	Phase 3	n/a

The following diagram shows a J1772 Type 2 connector wired with a charger on each phase.

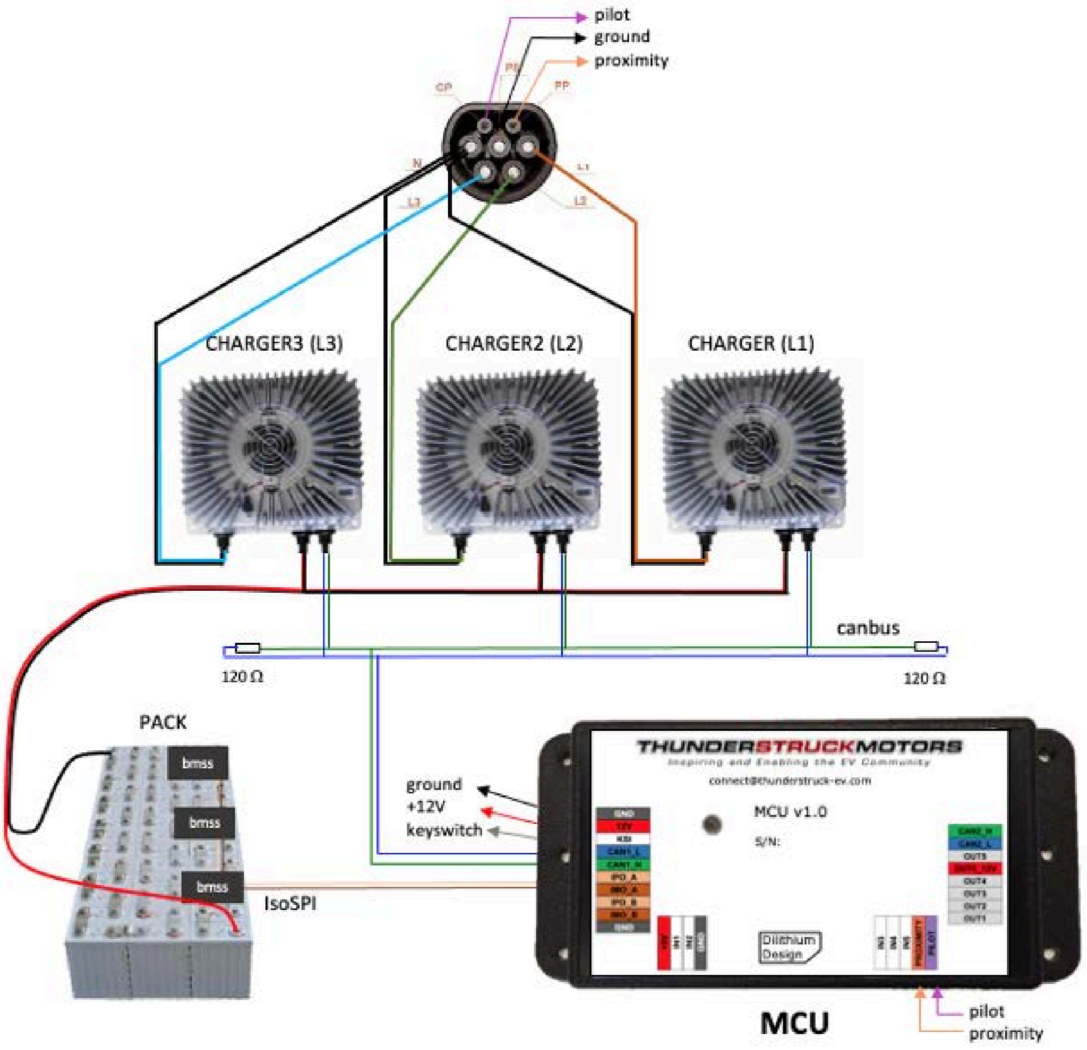


Figure 27 – J1772 Type 2 - One Charger Per Phase

J1772 Type 2 has some minor differences to the J1772 Type 1 protocol. J1772 Type 2 supports a removable charge cable without a pushbutton lock: the **proximity** signal only supports “connected” and “disconnected” (no “locked”). And, since charge cables can be rated at different line currents; the allowable values of the proximity resistance have been extended in order to communicate the charge cable ampacity to the EV.

The J1772 Type 2 **Proximity** resistor values are:

Proximity	Cable Rating
1.5 K	13A
680 Ω	20A
220 Ω	32A
100 Ω	63A

The J1772 Type 2 **Pilot** protocol works similarly to J1772 Type 1. As with Type 1, if the duty cycle is <=50%, the line current in amps (per phase) is calculated by the same formula (amps = duty cycle times 6/10). For duty cycles larger than 50%, the J1772 Type 2 protocol has extensions in in order to handle higher currents.

Direct Plug

In a solar power or marine application, J1772 charging may not be available or appropriate. The charge plug parameter may be set to "direct" (**`set plug direct`**), to disable the line cable ampacity calculation and as the J1772 pilot protocol.

In the most barebones implementation, the **Proximity** input is left disconnected. In this case, autostart is not available, and the wiring must connect the KSI input to the 12V input power. Charging current must be manually entered (**`maxc`**). Since there is no notion of charge plug presence, a charge cycle begins when CAN messages start arriving from the onboard charger. This would be the case, for example, if the onboard charger were plugged into line power. When a charge cycle completes, the MCU will stop sending CAN messages to the charger and will require that the Charger stop sending CAN messages for a "warmdown period" before it considers the charge cycle complete and before it will allow a new charge cycle to begin. This would occur, for example, if the onboard charger were unplugged from line power.

Alternately, the sleep/autostart and charge plug presence functions can still be used with the **`direct`** option. In this case, **KSI** is wired to the Key Switch as before. The autostart function can be initiated by applying a ground the **Proximity** input which will wake the MCU and start the charge. Similarly, releasing the ground on **Proximity** will stop the charge.

ZEVCCS Onboard Charging

When the ZEVCCS option is enabled, different mechanisms are used to implement autostart, determine line ampacity, and determine charge plug presence. See the Section *ZEV CCS Charging*.

Charge Plug State

In all, the charge plug is in one of five "states": **disconnected**, **connected**, **locked**, **active**, and **waiting for disconnect.**

In the **disconnected** state, the MCU is prepared to start a charge. The **connected** state means that the charge plug has been inserted. In the case of J1772, there is an additional **locked** state.

The **active** state is used in the direct plug option and means that CAN messages are being received from the charger.

Once a charge has completed, the plug enters the **waiting for disconnect** state. This state is used to insure that the plug be completely disconnected (or that charger CAN messages stop) before the MCU will start a new charge session.

The valid states and state transitions depend on the configuration of the charge plug.

- J1772 uses: **disconnected**, **connected**, **locked**, and **waiting for disconnect**.
- J1772T2 uses: **disconnected**, **connected**, and **waiting for disconnect**.
- Direct Plug with proximity uses: **disconnected**, **connected**, and **waiting for disconnect**.
- Direct Plug without proximity uses: **disconnected**, **active**, and **waiting for disconnect**.

Charging

The MCU has a supervisory circuit which monitors for a change in the **Proximity** resistor value. It will wake the MCU if it detects a high to low transition of proximity resistance to 680 Ω or less.

> NOTE: Autostart will not work for a J1772 Type 2 13A cable. To charge with a 13A cable the user must install the cable and then apply power to the KeySwitch input. This will power up the MCU, charging will start, and the MCU will be able to determine the ampacity of the charge cable.

When the MCU powers on or wakes up, it enables the **power** Logical Output. The MCU checks if the charge plug is connected, and if so, the MCU will attempt a new charge cycle. When a charge completes, the MCU will remain powered if the **KSI** input is on, otherwise it will disable the **power** Logical Output and go to sleep.

Target Current

Before CAN messages can be sent to the chargers, the MCU must determine Target Charging Voltage and Target Charging Current. For the bulk charge phase, Target Charging Voltage is determined by the configuration parameter **`maxv`**.

Target Charging Current might be directly set by using the parameter **`maxc`**. Alternately, Target Charging Current might be calculated from line current. In turn, Line current may be directly configured (by setting the parameter **`linec`** to a value specified in amps) or if **`linec`** is configured to the special value **`J1772`**, the J1772 duty cycle measurement determines line current. (If the **`J1772T2`** option is set, a modified calculation is used and calculated line current may be reduced based on the ampacity of the charge cable, as determined by the **Proximity** reading). Once available line current is determined, Target Charging Current is determined based on a power calculation which converts Line Watts to Charge Watts. To do this calculation, it is necessary to know line voltage, line current, and target voltage. If **linev** is configured, then it is used for Line Voltage. If **linev** NOT provided, a heuristic is used[3].

If both **linec** and **maxc** are provided, and there will be two ways to determine Target Current. In this case, the MCU will choose the minimum of the two.

The Pseudo code, below, captures the algorithm used

```
// best guess for line current
if linec is defined
    if linec is J1772
        LineCur = DutyCycle * (6/10)
    else if linec is specified in amps
        LineCur = linec

    // determine line voltage
    if linev is defined
        LineVol = linev
    else if (DutyCycle > 25)
        LineVol = 220
    else
        LineVol = 110

    // calculate target current using a power calculation
    TC = (LineVol * LineCur * .9)/ maxv

    if maxc is defined
        TargetCurrent = min(maxc, TC)
    else
        TargetCurrent = TC
else
    TargetCurrent = maxc
```

If multiple chargers are defined, then available Target Current is divided among the chargers on each phase to insure that the calculated line current limit is not exceeded.

J1772 Startup

When the charge plug is first inserted to start a J1772 Charge, several events occur:

The MCU:

1. If the MCU is in sleep mode, the the plug in of the charge cable will cause the MCU to wake.
2. The MCU determines the presence of the EVSE by measuring the the **Proximity** input. It then connects the **Pilot** "charge enable" resistor and starts a **Pilot** duty cycle measurement.
3. Once the MCU completes the duty cycle measurement, it then determines Charge Voltage and Charge Current and begins sending CAN messages to the chargers to begin charging.

[3] If the J1772 duty cycle is less than 20% then the MCU assumes the line voltage is 110V, otherwise the MCU assumes a line voltage of 220. This is driven by commonly available EVSE equipment used in the North American market.

The EVSE

1. The EVSE detects MCU presence using the **Proximity** signal. The EVSE then starts sending a square wave on the **Pilot** signal to advertise its ampacity.
2. When the EVSE detects the **Pilot** "charge enable" resistor, it will close its contactors allowing line current to flow to the EV Chargers.
3. When the EV Chargers receive line power, they will power up and begin sending CAN messages.

The EVSE has a delay before the **Pilot** square wave starts and the chargers have a delay before powering up. Still, this entire sequence of events typically happens in 3 to 5 seconds. If the startup sequence does not complete quickly enough, the MCU will terminate the charge session.

Slowstart Option

Some public EVSEs have been found to have several tens of seconds of startup delay and so the default startup time can be extended (to up to 45 seconds) with the **`SLOWSTART`** option.

J1772 Time Of Use Charging

Time of Use charging is supported. In this case, the charge plug is inserted into the EV several hours before charging is to begin. Upon charge plug insertion, the MCU will start as above. The EVSE then may perform the initial startup sequence but "decide" to delay charging until some later time. The EVSE will suspend the charge and then wait for the low-rate period to begin. When the time comes, the EVSE attempts to restart the charging.

The MCU supports this operation with the configuration parameter **`evsewait`**, specified in hours. If this parameter is set, the MCU will autostart, will "forgive" a short, suspended, charge such as described above, but will remain powered up and will wait for the charge to start later.

Direct Plug Startup

With this option, grounding **Proximity** starts the MCU. Target Current is specified by **`maxc`**. Releasing the ground on **Proximity** is the equivalent to charge plug removal.

Bulk Charge

Onboard charging begins in the Bulk Charge phase. Typically, two types of CAN messages are used for charging. The first, sent the charger, provides the charger with Target Voltage and Target Current, and the second message, sent from the charger, reports actual Pack Voltage and Charge Current. These messages are usually sent twice a second, both from MCU to Charger and from Charger to MCU. If there are multiple chargers, wach charger must have a unique CAN address and charge statistics are collected on each charger. If the MCU stops receiving periodic messages from the Chargers, charging will terminate with a termination reason of `CHARGER RX TIMEOUT.`[4]

Onboard charging can terminate for several other reasons, given in the following table:

[4] The timeout is only enforced for the first charger. If multiple chargers (e.g., **`charger2`**, **`charger3`** or **`charger4`**) are defined, their participation is optional and a CAN message timeout does not stop the charge.

Termination Reason	**Description**
Conditions	
PLUG DISCONNECTED	Charge Plug became disconnected
CELL LOOP	Cell Loop not OK
EVSE TIMEOUT	Timeout waiting for Pilot
CHARGER RX TIMEOUT	Charger CAN message timeout
NO PACK	No pack connected
CHARGER STOP	Charger unexpectedly stops delivering current
CHARGE TIMEOUT	Charge timeout
Alerts	
HARDWARE	Hardware Fault Alert
CELL_CENSUS	Cell Census Alert
THERM_CENSUS	Thermistor Census Alert
HVC	HVC Alert
THERM_OVERTEMP	Thermistor Overtemperature Alert
THERM_UNDERTEMP	Thermistor Undertemperature Alert
NOT_LOCKED	Configuration Not Locked Alert

Figure 28 - Onboard Charge Termination Reasons

In all, the bulk charge phase is controlled by the configuration parameters: **`linev`**, **`linec`**, **`maxv`**, **`maxc`**, **`termc`** and **`termt`**. The charge cycle normally completes when the charge current drops below the configured minimum charge current, set by the configuration parameter **`termc`**.

Finishing Charge

Lead Acid Batteries require a multi-stage charging algorithm. After the bulk charge phase a "finishing charge" phase may be defined. The Finishing Charge phase is controlled by the configuration parameters: **`fin_maxv`**, **`fin_maxc`** and **`fin_termt`**. This phase is enabled if **`fin_maxv`** is set to something other than 0.

The Finishing Charge phase will normally complete when the charge voltage rises above **`fin_maxv`**. This is in contrast to the Bulk Charge phase, which terminates when the charging current drops below **`termc`**.

For additional information about lead acid battery charging, see https://www.trojanbattery.com/pdf/TrojanBattery_UsersGuide.pdf.

Float Charge

Once batteries are charged, they may be kept on a "float charge" or "trickle charge". Lead Acid batteries, in particular, have a relatively high self-discharge rate and this phase can keep them topped up if the EV sits for an extended period of nonuse. However, this phase may apply to other battery chemistries as well.

A Float Charge phase is controlled by the configuration parameters: **`flt_maxv`**, **`flt_maxc`**, and **`flt_termt`**. This phase is enabled **`flt_maxv`** is set to something other than 0. Note that the value of **`flt_termt`** may be set to 0, which in this case means "forever".

Top Balancing

The “topbalancing” feature can be used to balance the pack. If this option is enabled, charging proceeds and terminates normally, with one exception: the charge does not stop with a cell `HVC` alert. When that occurs, charging is paused. When the `HVC` alert clears, charging resumes and the procedure repeats. The idea is that each such cycle will add more charge to lower cells and the pack gradually comes into balance. Note that if this technique is used, it is recommended to also use the parameter **`hvcc`** to prevent the charger from cycling on and off rapidly.

To enable this feature, the **`TOPBALANCE`** option needs to be enabled.

Charge History

Once charging starts, the MCU begins collecting per-charger statistics for the charge session, including Watt Hours delivered.

When the charge stops, a charge history record is written which records charge time, reason for completion, and charge session statistics such as highest voltage delivered and watt hours.

Charge Configuration

This section summarizes onboard charge configuration.

Plug Type

The plug type must be configured:

- **J1772** for J1772 Type 1 charging.
- **J1772T2** for J1772 Type 2 charging.
- **direct** for “direct plug” charging.

Input Functions

The following Input Functions are relevant to onboard charging:

- **pselect** is used for profile selection. If pselect is not mapped, then charge profile 1 is always used for onboard charging
- **plugin** is used to indicate that a charge plug is inserted. This input is used for ZEVCCS onboard charging**.**
- **loop** is used to as a “go/no-go”to indicate that charging is allowed.

The **loop** Input Function allows external circuitry to control the charge process. This input function may be mapped to any input. If **loop** is mapped to an input that measures voltage (IN1 to IN4), then the input must be +5V to be considered “TRUE”. If the **loop** is mapped to IN5 (which measures resistance, then it must be 0R to be considered. “TRUE”.

Output Functions

The following Logical Outputs are relevant to onboard charging:

- **power** on when power is on.
- **charge** on when charging
- **driveok** on when the charge plug is not connected.

Options

The following options are relevant to onboard charging:

- **`SLOWSTART`** – This option allows additional time for the EVSE to begin charging
- **`TOPBALANCE`** – This option allows the "topbalance" feature. See above.

Chargers

The MCU supports up to four chargers (named: **`charger`**, **`charger2`**, **`charger3`**, and **`charger4`**). Chargers are defined using the **`set charger`** command. The charger type indicates both the charger type and its CAN address. It is possible to have chargers from multiple manufacturers (e.g., one ELCON and one TSM2500) at the same time.

The following example defines a single charger and sets its type to tsm2500:

```
evcc> set charger tsm2500
evcc> show config
  charger  : tsm2500
  maxv     : 158.0V
  maxc     :  12.0A
  termc    :   0.5A
  termt    : 720.0hr
evcc>
```

This example defines a second charger and sets its type to tsm2500_42.

```
evcc> set charger2 tsm2500_42
evcc> show config
  charger  : tsm2500
  charger2 : tsm2500_42
  maxv     : 158.0V
  maxc     :  12.0A
  termc    :   0.5A
  termt    : 720.0hr
evcc>
```

A charger can be deleted by setting the charger to "none".

```
evcc> set charger2 none
```

With **J1772 Type 2**, three-phase power can be used and chargers may be associated with a phase: one of L1, L2 or L3. Here is an example:

```
evcc> set charger tsm2500 L1
evcc> show config
  charger  : tsm2500    L1
  maxv     : 158.0V
  maxc     :  12.0A
  termc    :   0.5A
  termt    : 720.0hr
evcc>
```

Charge Profiles

Up to four charging profiles may be defined, numbered from 1 to 4. Each profile contains a copy of all charging parameters. By default, Profile 1 is created and cannot be deleted. Only one profile may be edited at a time: a profile is chosen for editing by using the command **`edit profile <n>`**, where n is from 1 to 4. This command will create the profile if it does not exist; and subsequent edits to profile parameters will affect that profile. A profile may be deleted by using the command **`delete profile <n>`**. Profile 1 may not be deleted.

In the following example, there is one profile to start. Profile 2 is created using the command **`edit profile 2`**. Note that the **`show config`** output changes if there is more than one profile, and that the profile that is currently being edited is marked with a (*). Also, note that **`linev`** and **`linec`**, normally optional parameters, are displayed once they are defined.

```
evcc> sh config
--charging--------------------------------
  charger   : tsm2500
  maxv      :    20.0V
  maxc      :     0.0A
  termc     :     2.0A
  termt     :    72.0hr
evcc>
evcc> edit profile 2
evcc> sh config
--charging--------------------------------
  profile   :      1               2 (*)           3               4
  charger   : tsm2500         tsm2500
  maxv      :    20.0V           20.0V
  maxc      :     0.0A            2.0A
  termc     :     2.0A            2.0A
  termt     :    72.0hr          72.0hr
evcc>
evcc> set linev 220
evcc> set linec 25.5
evcc> sh config
--charging--------------------------------
  profile   :      1               2 (*)           3               4
  charger   : tsm2500         tsm2500
  linev     :                   220.0V
  linec     :                    25.5A
  maxv      :    20.0V           20.0V
  maxc      :     0.0A            2.0A
  termc     :     2.0A            2.0A
  termt     :    72.0hr          72.0hr
evcc>
```

In order to use a charge profile when charging, the **pselect** Input Functions must be mapped to an Input. When charging, **pselect** is used to select the profile.

The MCU measures the resistance to GND at this input and determines four possible selections: “inf”, 20K, 5K, and “0”, using the following formula:

R >= 30K the result is “inf”

30K > R >= 10K	the result is “20K”
10K > R >= 2K	the result is “5K”
20K > R	the result is “0”

If **pselect** input is left unconnected, it will read “open” (or “infinite” resistance), and maps to “inf”. If **pselect** is shorted to ground it will measure “0”. If a resistor is connected between the **pselect** input and ground, the remaining two choices “20K” and “5K” can be selected. The user might connect the input to a switch to GND to enable two profiles, or connect it to a multi-position switch and a resistor network and enable up to four profiles.

Note that the four input values represent a “switch setting” and not a “profile number”. The mapping from input value to profile number is done using the command **`set pmap`**.

In the following example, **IN5** is mapped to **pselect**. Note that the output of **`show config`** changes to show the **pselect** mappings; all **pselect** mappings are set to choose profile 1.

```
evcc> set in5 pselect
  in5       : pselect
evcc> sh config
--charging--------------------------------
  profile   :      1               2 (*)           3               4
  charger   : tsm2500         tsm2500
  linev     :                   220.0V
  linec     :                    25.5A
  maxv      :    20.0V            20.0V
  maxc      :     0.0A             2.0A
  termc     :     2.0A             2.0A
  termt     :    72.0hr           72.0hr
  pmap      :
    inf     :      x
    20K     :      x
    5K      :      x
    0       :      x
evcc>
```

Then the profile map is edited to make “inf” at **pselect** choose profile 2.

```
evcc> set pmap inf 2
evcc> sh config
--charging--------------------------------
  profile   :      1               2 (*)           3               4
  charger   : tsm2500         tsm2500
  linev     :                   220.0V
  linec     :                    25.5A
  maxv      :    20.0V            20.0V
  maxc      :     0.0A             2.0A
  termc     :     2.0A             2.0A
  termt     :    72.0hr           72.0hr
  pmap      :
    inf     :                      x
    20K     :      x
    5K      :      x
    0       :      x
evcc>
```

Charger Support

TSM2500

See *TSM2500 Series High Efficiency Intelligent Charger, ThunderStruck User Manual Ver 1.0.8*.
http://www.thunderstruck-ev.com/images/companies/1/ThunderStruck-TSM2500-ManualV1.08.pdf

The CAN connections are found on the four-pin connector J3. CANL is pin #8 (wired with a blue wire) and CANH is pin #9 (wired with a green wire). No other connections are required on J3.

The TSM2500 charger does not have an integrated termination resistor (however a termination resistor may have been provided by Thunderstruck in the supplied harness). The charger is configured with a default CAN address. However, when using multiple chargers in an installation, the CAN address can be reprogrammed. Note that address programming for multiple chargers may have been done at Thunderstruck as part of the order.

The MCU defines the following TSM2500 charger types:

- `tsm2500` - default
- `tsm2500_41`
- `tsm2500_42`
- `tsm2500_43`

The default value for tsm2500 chargers is “40”. (Which is to say, the MCU uses the CAN address 0x18e5**40**24 for messages TO the charger and 0x18eb24**40** FROM the charger to the MCU).

The TSM2500 can report the following errors reported in the “trace charger” output:

- `rxerr`
- `hwfail`
- `overtemp`
- `not charging`
- `input voltage err`
- `pack voltage err`

Setting the CAN address of a TSM2500 Charger

This section describes how to set the CAN addresses of a tsm2500 charger.

For this procedure, the charger can either be directly connected to mains power or can be installed in the vehicle and the J1772 charge plug can be used to supply line power. When doing this procedure, ensure that only one charger is powered.

In this example, we want to define a second charger as type tsm2500_42. In order to program the charger, it is necessary to use the **program** keyword.

To do this, power up the MCU and provide line power to the charger. Then type the following command:

```
mcu> set charger2 tsm2500_42 program
```

The MCU will then print

```
 ***
 ***                      tsm2500 PROGRAMMING                       ***
 *** WARNING: This command changes the CAN ID of a tsm2500 charger  ***
 *** ONLY ONE tsm2500 charger should be powered up at this time     ***
 ***

Proceed [Y/N] ?
```

If you type "y", the MCU then prints

```
Programming the charger ...
```

and then 5-10 seconds later it prints

```
Programming the charger ... done.
The charger must now be power cycled.
mcu>
```

At that point the new charger will be programmed to tsm2500_42 and it will be configured in the MCU as "charger2".

ELCON PFC

In order to support a CAN interface, ELCON PFC chargers must programmed with the CAN option and a external ELCON-provided CAN module is needed. The CAN module has two pins are provided for the CAN connection: CANH and CANL and does not contain an integrated termination resistor.

The CAN addresses of the ELCON chargers are determined by the outboard serial to CAN converter: in order to change the CAN address, a different serial to CAN module is needed.

The MCU supports the following ELCON charger types:

- `elcon` - default
- `elcon_e7`
- `elcon_e8`
- `elcon_e9`

The default value for ELCON chargers is “E5”. (Which is to say, the MCU uses the CAN address 1806**e5**f4 for messages TO the charger and 18ff50**e5** FROM the charger to the MCU).

The ELCON charger can report the following errors reported in the “trace charger” output:

- `rxerr`
- `hwfail`
- `overtemp`
- `input voltage err`
- `pack voltage err`

LEAR

The MCU supports some Lear chargers. This support is limited to the “control message” to the charger with CAN ID “0x00000050” and the status message from the charger with CAN ID “0x00000617”. It has been found that not all Lear chargers use these messages due to different firmware and may not support this message set.

The MCU only defines a single lear charger type, named “`lear`”.

Having more than one Lear charger is not supported. Having one Lear charger and another non-Lear charger is not supported. Since the Lear charger runs at 500kbps, all CAN devices on that network must be set to the 500kbps datarate.

Debugging

Show

The **`show`** command gives high level charging status, including charge time, target voltage and target current, and the status of each charger.

```
evcc> sh
  state     : CHARGE, CC, CV
  plug state: CONNECTED
  chg time  : 9 mins
  voltage(t):  120.0V
  current(t):   14.0A
  charger   : tsm2500
    status  : 1135 msgs sent; 820 msgs received
    voltage :  118.7V
    current :   12.8A
    charge  : 244Wh
evcc>
```

Tracing

The command **`trace state`** shows real-time information about charge progress, including information about plug state as well as charge state.

```
evcc> trace state
state tracing is now ON
```

```
evcc>  00:06:41.5 Plug State: DISCONNECT_WAIT => DISCONNECTED
00:06:46.0 Plug State: DISCONNECTED => CONNECTED
00:06:46.0 Charge State: STANDBY => STARTUP
00:06:46.3 Charge State: STARTUP => CHARGE, Charge Type: J1772, CurCalc: MAXC
00:07:21.1 Plug State: CONNECTED => DISCONNECT_WAIT
00:07:21.2 Charge State: CHARGE => WARMDOWN, term rsn=PLUG DISCONNECTED
00:07:26.2 Charge State: WARMDOWN => STANDBY
00:07:26.2 Plug State: DISCONNECT_WAIT => DISCONNECTED
```

The command **`trace charger`** decodes charger CAN messages.

```
evcc> trace charger
charger tracing is now ON
00:10:28.8  tsm2500_42: V=126.0, A= 5.8, W=730, Wh= 0.10, TMP = 26C
00:10:28.9  tsm2500   : V=126.3, A= 5.9, W=745, Wh= 0.09, TMP = 26C
00:10:29.3  tms2500_42: V=126.6, A= 5.7, W=721, Wh= 0.19, TMP = 26C
00:10:29.3  tsm2500   : V=126.6, A= 5.8, W=734, Wh= 0.19, TMP = 26C
00:10:29.8  tsm2500_42: V=127.2, A= 5.9, W=750, Wh= 0.30, TMP = 26C
00:10:30.1  tsm2500_42: V=127.2, A= 5.9, W=750, Wh= 0.37, TMP = 26C
```

Charge History

Charging history is provided which records the reason that charging stopped, total charge time, maximum voltage, maximum current, final current, and watt hours delivered.

```
evcc> show history
      |   term    |  charge |           |   watt | maximum| maximum|  ending|
  num |  reason   |   time  |  charger  |  hours | voltage| current| current|
--------------------------------------------------------------------------
 last |  normal   | 217 mins|tsm2500    |  2431Wh| 120.0V |  12.8A |   2.0A |
  - 1 | plug disc|   1 mins|tsm2500    |    14Wh| 115.0V |  12.8A |  12.8A |
```

OFFBOARD CHARGING

The MCU supports two types of offboard charging: fast charging and ZEVCCS charging. Both types of offboard charging are currently in development, and the following information is preliminary. Only one type of offboard charging may be enabled.

Fast Charging

Configuring Fast Charging is performed by enabling the feature (with the **`enable fcharge`** command) and by configuring the offboard charge profile.

```
evcc> enable fcharge CAN2
mapping IN3 to CSEQ1
mapping IN4 to CSEQ2
mapping OUT2 to CPERM
mapping OUT3 to CCONT
evcc>
```

This command will also attempt to assign the Input Function **CSEQ1** and **CSEQ2** to default inputs, and **CPERM** and **CCONT** to default outputs.

An offboard charge profile needs to be created:

```
evcc> edit profile offboard
evcc> set maxv 280
evcc> set maxc 35
evcc> set termt 1
evcc> sh config
--offboard charging (edit)----------------
  protocol  : fcharge
    cseq1   : in3
    cseq2   : in4
    cperm   : out2
    ccont   : out3
  maxv      :   280.0V
  maxc      :    35.0A
  termt     :     1.0hr
--onboard charging------------------------
  charger   : tsm2500
  maxv      :    20.0V
  maxc      :     2.0A
  termc     :     2.0A
  termt     :    72.0hr
evcc>
```

The commands **`trace state`**, **`trace charger`**, and **`show history`** can be used to verify proper operation of offboard charging.

ZEV CCS Charging

The Zero EV CCS charging system (ZEVCCS) manages all aspects of the CCS fast charging protocol. It has all connections to the CCS charge plug, handles EV power up when a charge is started, and performs all messaging with the CCS charger. When fast charging, the MCU performs BMS functions: it provides pack target charge voltage and current as well as BMS status such as `HVC` or `THERM_OVERTEMP` alerts.

Although ZEVCCS does not perform onboard charging, it manages the charge plug, and a special interface is used between the ZEVCCS system and the MCU so the MCU can be notified of charge plug presence and its ampacity. Charge plug presence, normally provided by the **Proximity** input, is instead provided by a new input called **plugin**. Charge plug ampacity, normally provided by the pilot signal, is determined by the ZEVCCs system and communicated to the MCU by a CAN message.

Enabling ZEVCCS charging is performed by the **`enable zevccs`** command.

```
sys> enable zevccs can1
mapping IN3 to PLUGIN
sys>
```

This command will also attempt to assign the Input Function **plugin** to a default input. If **IN3** is used for other purposes, plugin can be assigned to any of **IN1** to **IN4**.

Now create and edit the offboard charge profile:

```
evcc> edit profile offboard
evcc> sh config
--offboard charge profile (edit)----------
  protocol  : zevccs
    plugin  : in3
  maxv      :    20.0V
  maxc      :     2.0A
--onboard charge profile(s)---------------
  charger   : tsm2500
  maxv      :    20.0V
  maxc      :     2.0A
  termc     :     2.0A
  termt     :    72.0hr
evcc>
```

Now set the Target Voltage and Target Current for offboard charging.

```
evcc> set maxv 320
evcc> set maxc 25
evcc> sh config
--offboard charge profile (edit)----------
  protocol  : zevccs
    plugin  : in3
  maxv      :   320.0V
  maxc      :    25.0A
--onboard charge profile(s)---------------
  charger   : tsm2500
  maxv      :    20.0V
  maxc      :     2.0A
  termc     :     2.0A
  termt     :    72.0hr
evcc>
```

Note that when using the ZEVCCS system for both offboard and onboard charging, it is necessary to define a profile for onboard charging. This is done in the normal way, see *Onboard Charging, above*.

When ZEVCCS is enabled, the MCU will broadcast BMS status by sending the following five messages:

```
ZEV_BMS_LIMITS          0x0351
ZEV_BMS_SOC             0x0355
ZEV_BMS_STATUS          0x0356
ZEV_BMS_ERRORS          0x035a
ZEV_BMS_STATUS2         0x035b
```

Two messages are expected from the ZEVCCS Fast Charge Controller:

```
ZEV_CCS_COMMAND         0x0357
ZEV_CCS_DIAGNOSTIC      0x036a
```

The ZEV_CCS_COMMAND message is used to disable any Ground Fault monitoring circuitry, if present, and it provides the ampacity of the line connection, needed for Onboard Charging.

The ZEV_CCS_DIAGNOSTIC command contains debug information.

It can be verified that these messages are received in the MCU by typing **`show`**:

```
evcc> show
  state     : STANDBY
  plug state: DISCONNECTED
  ZEVCMD  rx: 56
     curlim :  22.4A
  ZEVDIAG rx: 7
       diag : 10 03
         D0 : BMS_FLT
         D1 : INC_CHARGER CHARGE_MODE
evcc
```

The commands **`trace state`**, **`trace charger`**, and **`show history`**, can be used to verify proper operation.

Tracing ZEV CCS messages:

```
evcc> tr charge
charge tracing is now ON
evcc> 01:30:15.6 zevccs    : CCS_CMD; curlim= 22.4A
01:30:16.2 zevccs    : CCS_CMD; curlim= 22.4A
01:30:16.6 zevccs    : CCS_CMD; curlim= 22.4A
01:30:17.1 zevccs    : CCS_CMD; curlim= 22.4A
01:30:17.6 zevccs    : CCS_CMD; curlim= 22.4A
```

```
01:30:17.6 zevccs    : CCS_DIAG; bvs=105.7V, cvs=115.2V, d= 10 03
01:30:18.1 zevccs    : CCS_CMD; curlim= 22.4A
01:30:18.6 zevccs    : CCS_CMD; curlim= 22.4A
```

Here is an example of **`trace state`** output while onboard charging (simulating charge plug removal):

```
mcu> trace state
state tracing is now ON
mcu>
mcu> 00:17:23.3 Plug State: DISCONNECTED => CONNECTED
00:17:23.3 Charge State: STANDBY => STARTUP
00:17:23.9 Charge State: STARTUP => CHARGE, Charge Type: ZEV_OB, CurCalc: ZEVCCS
00:17:46.0 Plug State: CONNECTED => DISCONNECTED
00:17:46.1 Charge State: CHARGE => WARMDOWN, term rsn=PLUG DISCONNECTED
00:17:51.1 Charge State: WARMDOWN => STANDBY
```

An example of **`show`** while using ZEVCCS onboard charging:

```
evcc> sh
  state     : CHARGE
  plug state: CONNECTED
  ZEVCMD  rx: 2114
     curlim :  22.4A
  ZEVDIAG rx: 262
       diag : 10 03
         D0 : BMS_FLT
         D1 : INC_CHARGER CHARGE_MODE
  chg time  : 0 mins
  voltage(m):  310.0V
  current(m):  10.0A
  charger   : tsm2500
    status  : 26 msgs sent; 19 msgs received
    voltage :  120.0V
    current :   12.8A
    charge  : 5.06Wh
evcc>
```

INSTRUMENTATION

State of Charge Measurement

State of Charge (SOC) is a measurement of EV pack charge as a percentage (0 represents “empty”, and 100 represents “full”), (SOC) the EV equivalent of a Fuel Gauge. The MCU computes SOC in two ways: by Charge (or Coulomb) Counting and by Voltage Estimation.

Voltage Estimation uses BMS measured cell voltages and configured HVC and LVC limits to estimate SOC. Voltage Estimation is not very accurate but may be used when Charge Counting is not available. It can also be used in addition to Charge Counting as a consistency check.

Charge Counting requires a current sensor to measure charge put into or taken from the pack. Ongoing charge counting adjusts the SOC up or down and has ways to synchronize the SOC value periodically. At the end of a complete, and successful, charge cycle, for example, the SOC value is typically set to 100. The SOC can also be manually reset by using the **socreset** Input Function.

In order to configure SOC by Charge Counting, a Current Sensor needs to installed and the pack capacity in kilowatt hours must be configured.

Current Sensors

The MCU supports the following Current sensors:

Sensor	Model	Low Range	High Range
dhabs44	DHAB S/44	+/-20A	-420A to 230A
dhabs125	DHAB S/125	+/-25A	+/-200A
dhabs133	DHAB S/133	+/-75A	+/-750A
dhabs137	DHAB S/137	+/-75A	+/-1000A

Sensor	Model	with Offset	Range
htfs200	HTFS-200	-	+/- 300A
		yes	-420A to 180A
htfs400	HTFS-400	-	+/- 600A
		yes	-820A to 380A
htfs600	HTFS-600	-	+/- 900A
htfs800	HTFS-700	-	+/- 1200A

Figure 29 - Current Sensor Support

All of these sensors are manufactured by LEM (see https://www.lem.com/en). These sensors require +5V and GND to operate and provide two outputs; they may be connected directly to Connector B of the MCU.

The current sensor Input Functions **hall1** and **hall2** are mapped by default to IN1 and IN2 on Connector B.

```
sys> show inputs
  in1   2.54v  hall1   DHABS137 H = 23.5A
  in2   3.23v  hall2   DHABS137 L = 27.0A
  in3   1.62v
  in4   4.77v
  in5    inf
  prox   inf   J1772 disconnected
  pilot        not detected
  ksi      1   detected
```

(If, for some reason, **hall1** and **hall2** are not mapped, the commands **`set in1 hall1`** and **`set in2 hall2`** can be used).

In addition, the sensor type and pack capacity must be configured:

```
inst> set isensor dhabs137
inst> set packkwh 24

inst> sh config
--soc------------------------------------
  isensor   : DHABS137
    hall1   : in1
    hall2   : in2
  packkwh   : 24.0

…
inst>
```

Once **`isensor`** is configured and the inputs connected, SOC by Charge Counting is automatically enabled.

SOC synchronization will automatically be reset to 100 at the conclusion of a successful charge. Optionally, the **socreset** input function may be defined, which, when enabled, can be used to manually reset the SOC, by applying a ground to the input.

LEM DHAB Current Sensors

The DHAB sensors are dual range, bidirectional sensors. The high range determines the working limts of the sensor and the low range is used to achieve high accuracy at low current. The MCU firmware automatically selects which range to use when measurements are taken.

See the diagram below for correct orientation at installation.

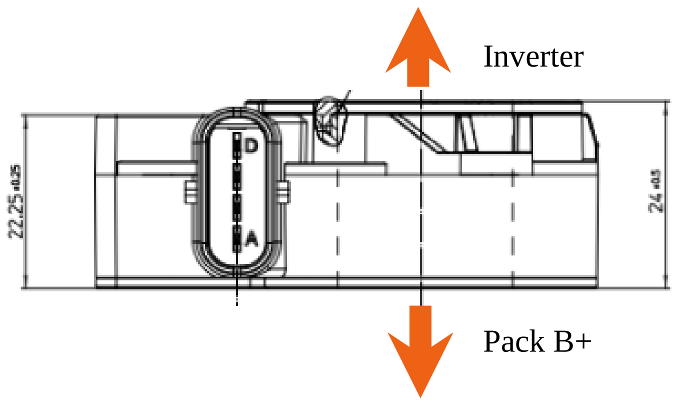


Figure 30 – DHAB V2 Hall Sensor Installation

The sensors have an integrated waterproof connector. The mating connector is a Tyco 1-1456426-5. Connector B on the MCU can connect directly to a DHAB sensor as it has all necessary connections. Connections to the Hall Sensor are made using the following table. Note that the connections to the DHAB sensors vary by model.

MCU Connector	MCU Pin Name	DHAB Signal	DHAB S/44	DHAB S/125 DHAB S/133	DHAB S/137
B1	+5V	Vcc	B	A	B
B2	hall1	High Range	A	B	D
B3	hall2	Low Range	D	D	A
B4	GND	Gnd	C	C	C

Figure 31 - DHAB Hall Sensor Connections

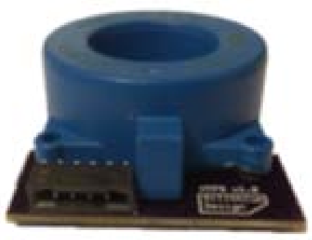
LEM HTFS Current Sensors

The LEM HTFS Hall sensors are single range, bidirectional sensors. There are several models: HTFS-200, HTFS-400, HTFS-600, and HTFS-800. These sensors have a measurement output (Vo) and a reference input (Vr). The reference input sets the "0" value of the sensor and can be overdriven to shift the working range of the sensor. However, if this feature is not used, it is recommended that this input be connected as it serves to calibrate the sensor.

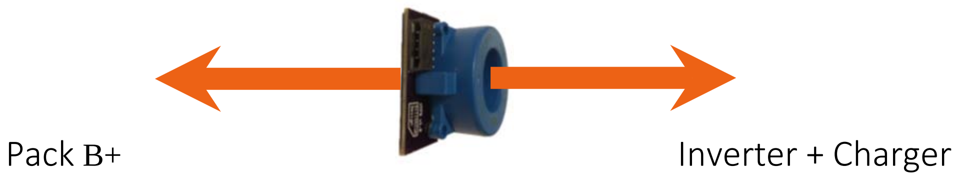
The LEM HTFS datasheet can be found at: https://www.lem.com/sites/default/files/products_datasheets/htfs_200_800-p.pdf.

The HTFS sensors are PCB mount. In the past, Dilithium Design has provided HTFS-200 and HTF-400 sensors mounted on a PCB board. In addition, the PCB has an onboard resistor which sets the Vo Offset voltage input in order to shift the working range.

See diagrams below for pinouts and orientation for the HTFS sensors



Vref	Vout	0V	+5



The following table indicates the MCU connections:

MCU Connector	MCU Pin Name	HTFS Signal	HTFS PCB Connector
B1	+5V	+5V	4
B2	hall1	Vout	2
B3	hall2	Vref	1
B4	GND	Gnd	3

Figure 32 - HTFS Hall Sensor Connections

MCU EV Display

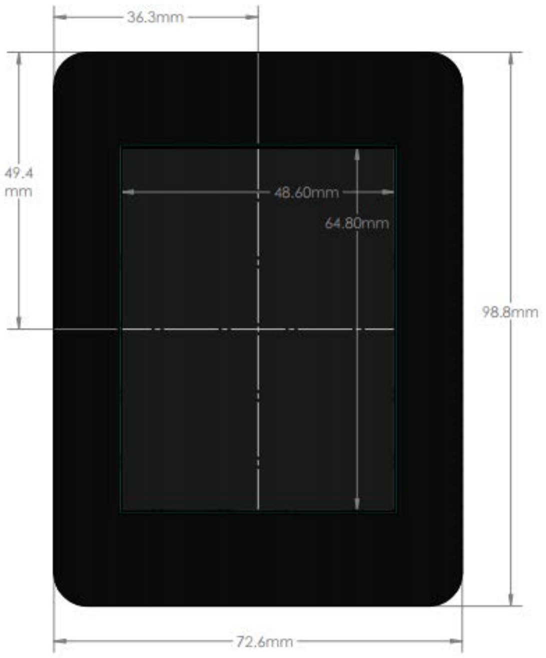
The MCU Electric Vehicle Display displays pack State of Charge, Pack Current, Voltage, and Watts, Thermistors, as well as BMS alarms. The MCU Display uses the same hardware and as the existing Dilithium Display used with the existing BMSC product.

The Display hardware uses the 4D systems gen4 32-DCT-CLB. This display is a 3.2" 240x320 pixel LCD capacitive touch display with an attached bezel. The datasheet can be found at http://www.4dsystems.com.au/product/gen4_uLCD_32D/.

With bezel, the display measures 98.8mm x 72.6mm. The display is mounted on a flat surface with a rectangular opening (approximately 80mm x 56mm) and affixed to the surface using double-sided tape.

Mechanical drawings can be found at http://www.4dsystems.com.au/productpages/gen4-uLCD-32D-CLB/downloads/gen4-uLCD-32D-CLB_drawing_R_1_0.PDF

A PCB is mounted directly on the back of the display and connects to the display using a 30pin flat cable. The PCB has two connectors: a 5-pin Systems connector and a 4-pin connector, but only the 5-pin system connector is used. When mounting the display, the 5-pin Systems connector is towards the bottom.



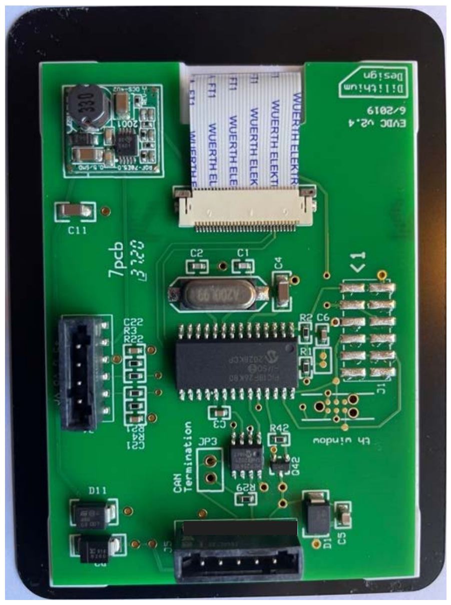


Figure 33 - MCU EV Display Hardware

The System Connector provides 12V power, and CAN; one pin is not used.

GND	12V	CANL	CANH	

The **GND** and **12V** inputs on the System Connector provide Display power. Power should be enabled when the EV is being driven or being charged. The maximum 12V power consumption is approximately 100ma. Display power may be provided directly from the MCU OUT5 output.

The **CANL** and **CANH** inputs connect to the EV CAN network. The Display will automatically sense a CAN datarate of 250Kb or 500Kb.

A CAN termination resistor in the Display may be enabled by installing the bridge between the right two pins of jumper JP3. This jumper is adjacent the 5p Systems Connector.

Startup Screen

When the Display is first powered up, it shows a Startup screen for a few seconds. During this time, the Display attempts to communicate with the MCU. The Display will then show both its build date as well as the firmware version number of the MCU, assuming the MCU is connected. In the example below, the display build date is “Aug 2021” and the MCU firmware version is “1.0.3”.

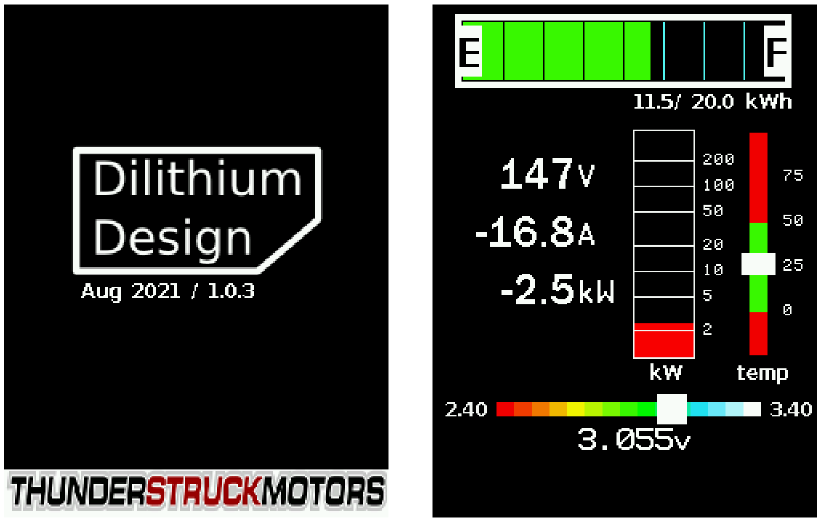


Figure 34 - MCU Display Startup and Status Screens

While the Startup screen is being displayed, it is possible to “freeze” it by touching the screen before it transitions to the next screen. This gives time to make note of the firmware version numbers if needed. Touching the screen again will unfreeze the display.

Status Screen

The **Status** Screen is the main screen. The Status screen is divided into several regions: State of Charge, Numeric Results, Wattmeter, Thermistors, Cell Voltage Summary, and Message.

The **State of Charge** (top) is a digital fuel gauge. This gauge shows the current SOC as calculated by the MCU. It also shows the estimated Kilowatt Hours of pack capacity remaining.

The **Numeric Results** (middle left) shows the Pack Voltage, Pack Amps, and Pack Watts.

The **KiloWatt** meter (middle center) shows Pack Watts. This scale is logarithmic. The bar will be red for power taken from the pack (as when driving) and is green for power being added to the pack (as during regeneration or charging).

The **Thermistor** (middle right) is an optional meter that is shown if any thermistors are enabled. The markings are in C. The green range are determined by the configured parameters **`thmax`** and **`thmin`**. The white bar indicates the range of thermistor temperatures from low to high; in the example, all thermistor temperatures are within 25C by a few degrees.

The **Cell Voltage Summary** lies just below the middle of the screen. It shows the range of cell voltages present within the pack as well as the average cell voltage. The scale is calculated from the configured **`hvc`** and **`lvc`** values in the BMS. In the example above:

- 2.40 is the BMS programmed Low Voltage Cutoff Voltage
- 3.40 is the BMS programmed High Voltage Cutoff Voltage
- 3.055v is the average cell voltage of all cells in all packs

The “white” indicator bar of the Cell Voltage Summary indicates the range of cell voltages present in the pack. The left border of the white indicator is the lowest cell voltage in the pack, the right border is the highest cell voltage in the pack and the width of the indicator gives an overall indication of pack balance.

In the example above, the pack is somewhat unbalanced which is indicated by the width of the white bar.

The **Message** region is at the very bottom of the. In the preceding example, there are no alerts present and the Message region is blank. If a thermistor reading exceeds the **`thmax`** setting an `THERM_OVERTEMP` alert will be generated and a blinking message will appear in the **Message** area. See example:

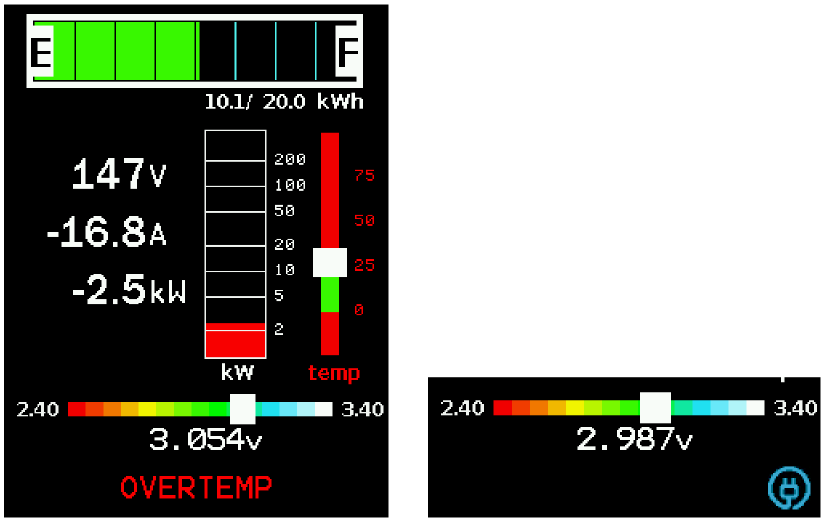


Figure 35 - MCU Display Message Region

The **Message** area also has a charge plug indicator at the right.

In all, the following messages are supported:

- HARDWARE – indicates a diagnostic failure by one of the measurement devices
- HVC – high voltage cutoff alert
- LVC – low voltage cutoff alert
- CELL CENSUS – cell census alert
- OVERTEMP – thermistor overtemperature alert
- UNDERTEMP – thermistor undertemperature alert
- THERM CENSUS – thermistor census alert
- NOT LOCKED – configuration not locked alert.

In the example on the right, the charge plug is shown in blue, which indicates that the charge plug is connected but the MCU is not charging. When charging, the plug is shown in red.

From the **Status** screen, the user can navigate to the **Cell Map** by touching the display anywhere above the **Cell Voltage Summary**.

Cell Map

The **Cell Map** shows a graphical representation of all cells. Cells are arranged in groups and each row of cells represents a cell group. The BMS architecture determines the nominal cell group size: cell groups may contain 12, 13 or 18 cells. Cell groups are connected in series to create a “pack”. Multiple packs may then be paralleled together. The pack is shown by the colored bar to the left of each group of cells: each pack is represented by a different color. Pack 1 is turquoise, Pack 2 is brown, and Pack 3 is cornflower blue, pack 4 is gold, up to 8 packs can be shown.

In the cell map, a color is assigned to every cell based on its voltage. The order is roughly: red, yellow, green, blue, white; where a “red” cell is low and a “white” cell is high.

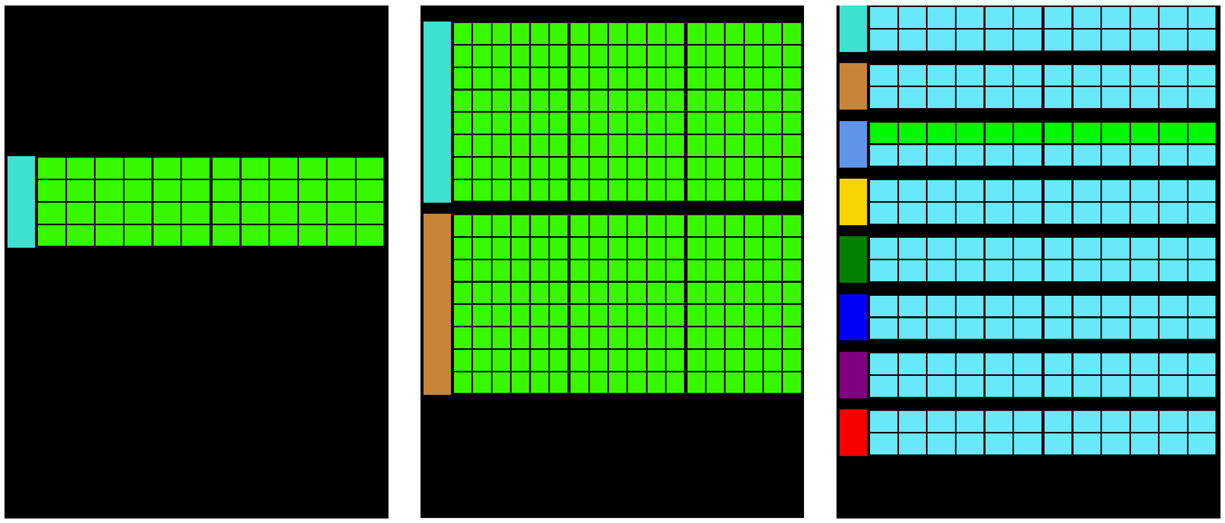


Figure 36 – MCU Display Cell Map Examples

- The example on the left shows a single pack, with four 12-cell groups.
- The example in the middle shows two packs, each with eight 18-cell groups.
- The example to the right shows eight packs, each with two 12-cell groups.

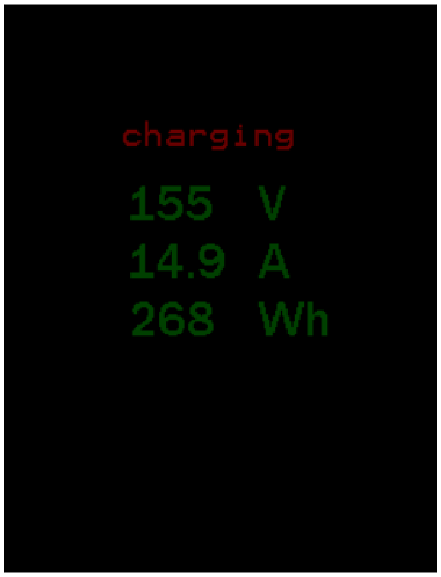
Note that in the first two examples the cells are in good balance with each other and show the same color. In the last example, the first group of the third pack (with cells in “green”) is lower than the other cells.

This screen updates in real time and can be used to track the performance of the cells under load.

To Navigate from this screen, return to **Status** by touching the Cell Map display

Charge Progress

When charging, the Display reverts to a charge progress screen, that shows voltage, amps, and Watt Hours. This screen is purposely dim as it may be on for several hours while charging. Touching the charge progress screen will return to the Status Screen where all operations are available.



```
charging
155  V
14.9 A
268  Wh
```

Analog Fuel Gauge

Analog Fuel gauges typically have three connections: +12V, Ground, and “sender”.

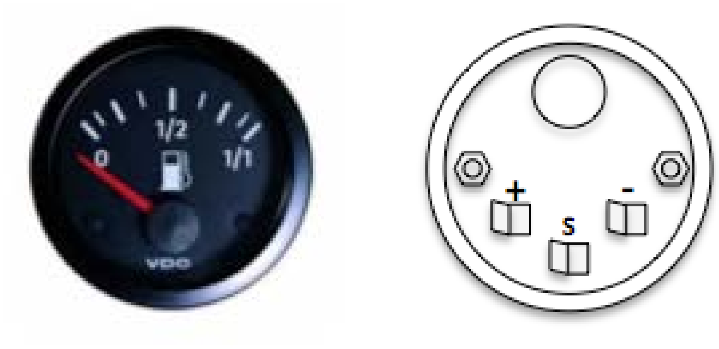
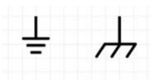


Figure 37 – Analog Fuel Gauge Connections

Note that the “Ground” connection on the meter might be indicated by “-“, or by a symbol:



Conventionally, the Fuel Gauge is connected to a gas tank float “sender” which is a variable resistor depending on the level of the float. Usually, this connection only uses one wire as the return connection is made directly to chassis ground. The fuel gauge and the sender are a matched set, and, although there are common values, there are no standard ones.

The MCU uses Pulse Width Modulation (PWM) to drive the meter. In order to accommodate the different fuel gauges, the **fuelg** PWM output must be mapped to a PWM-capable physical outputs (one of **OUT1**, **OUT2**, or **OUT3**).

```
inst> set out1 fuelg
```

The PWM settings **`fuel_0`**, **`fuel_25`**, **`fuel_50`**, **`fuel_75`** and **`fuel_100`** must be configured in order to calibrate the gauge. Each PWM setting is a number from 0 to 255. When the MCU updates the gauge it interpolates between the calibration values to determine an appropriate PWM values from 0 to 255.

An example:

```
inst> set fue_l0 30
```

When a **`set fuel_xx yy`** command is entered, the gauge will operate with the newly entered duty cycle. (And will time out after ten or twenty seconds). Repeat the process for each setting so that the gauge reads correctly for all five settings: 0%, 25%, 50%, 75%, and 100%.

The Display has been tested with a variety of fuel gauges, including VDO and Bosch.

Speedhut Gauges

The MCU supports CAN capable Speedhut gauges for fuel State of Charge, pack current, pack voltage, and temperature. The MCU refers to these gauges as **`sh_fuel`**, **`sh_amps`**, and **`sh_temp`**.

The gauges are enabled with the **`enable`** command. This command has two parameters: the gauge type and the CAN interface.

To enable all three gauges on CAN2, the following commands are used:

```
inst> enable sh_fuel can2
inst> enable sh_amps can2
inst> enable sh_temp can2
```

When enabled, the MCU will transmit the following messages with the given period. Note that these gauges are “receive only” and do not transmit any CAN messages.

gauge	**period**	**SID**	**B0**	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**
`fuel`	1s	0x355	SOC							
`amps`	125ms	0x356			Alo	AHi				
`temp`	250ms	0x357	VLo	Vhi	Thi					

- SOC is the State of charge, a number from 0 to 100.
- Alo and Ahi are the low and high bytes of the ammeter reading in AMPS, a signed sixteen bit number.
- Vlo and Vhi are the low and high bytes of the pack voltage in Volts, a sixteen bit number.
- Thi is the temperature of the highest enabled thermistor, a number from 0 to 255C.

Curtis 3100r

The Curtis 3100r display is a CAN controlled 52mm round display. This might be a good choice for simpler conversions as the support for this gauge is limited to a single pack with 96 cells or fewer.

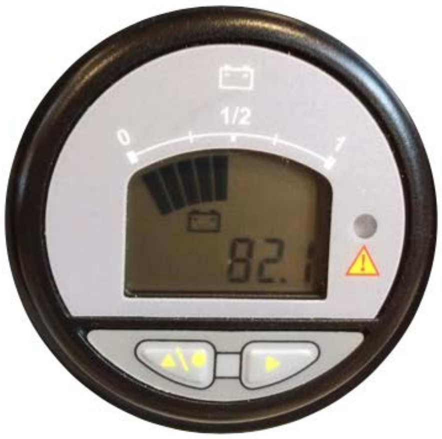


Figure 38 – 3100r Display

This display can display Pack State of Charge, has a 7-segment alphanumeric LCD display, graphical icons, an LED, backlight and two buttons. The 3100r can be programmed to operate at a 250Kb or a 500Kb CAN datarate. Once the 3100r is connected to power and the CAN network, the display may be enabled by the command:

```
inst> enable 3100r
```

Pack **SOC** is shown in the top 10 segment SOC arc. (see *State of Charge Measurement*, above)

The 3100r **LED** blinks when the pack is in HVC or LVC. The **LED** is on for Thermistor over-temperature.

The **Wrench** Icon blinks if there is a configuration error: configuration not locked, cell census error oe thermistor census error.

The **Battery** Icon is on when the pack is balancing.

The left button sequences the alphanumeric display between **pack voltage**, **minimum cell voltage**, **maximum cell voltage**, **average cell voltage**, and **standard deviation**.

- The pack voltage is shown in tenths of a volt.
- The minimum cell voltage shows the cell number, an "underbar" and then three digits which is the cell voltage in hundredths, but with no period. (So "354" is 3.54v).
- The maximum cell voltage shows the cell number, an "upperbar" and the value.
- The average cell voltage shows three "middle bars" and a value.
- The standard deviation shows "Sd", "space" and then three digits of standard deviation. (So “014” is 0.014v).

The right button toggles the **backlight**.

OBDII Dictionary

OBDII is a CAN based management protocol supported by most modern automobiles. In addition to CAN message format and encoding, OBDII defines the notion of “Parameter ID” or “PID”. The PID value identifies the type of data to be retrieved or set. In an implementation of OBDII, both standard PIDs and custom, manufacturer specific, PIDs may be used.

For a reference on ODBII, the CAN structure, and the definition of standard PIDs, see the following:
https://en.wikipedia.org/wiki/On-board_diagnostics
https://en.wikipedia.org/wiki/OBD-II_PIDs

Torque is an application that runs on a smartphone or tablet. It implements the ODBII protocol, supports many standard PIDs and allows the user to customize the display to define meters and to define what data is to be displayed. For a reference on Torque, see:
https://torque-bhp.com/wiki/Main_Page

In use, the smartphone typically uses Bluetooth to communicate with a Bluetooth to OBDII adapter, which acts as a gateway to the CAN network in the vehicle. ODBII Bluetooth adapters are readily available.

The MCU supports ODBII. See the following table for PID support. This table also has information useful to define custom meters within Torque.

Name	ModeAndPID	Equation	Min	Max	Units	Scale
RunTime	011f	int16(B:A)	0	65535	seconds	
Version	22dd80	A				1
Revision	22dd80	B				1
ALERTS	22dd81					
HARDWARE		BIT(A:6)	0	1		
CELL_CENSUS		BIT(A:5)	0	1		
THERM_CENSUS		BIT(A:4)	0	1		
HVC		BIT(A:3)	0	1		
LVC		BIT(A:2)	0	1		
THERM_OVERTEMP		BIT(A:1)	0	1		
THERM_UNDERTEMP		BIT(A:0)	0	1		
Pack Voltage	22dd83	int16(B:A)/10			volts	0.1
Pack Current	22dd84	((signed(B)*256)+A)/10)			amps	0.1
Pack SOC	22dd85	A	0	100	percent	1
Cell Min	22dd86	int16(B:A)/10000			volts	0.0001
Cell Max	22dd87	int16(B:A)/10000			volts	0.0001
Cell Mean	22dd88	int16(B:A)/10000			volts	0.0001
Cell STD	22dd89	int16(B:A)/10000			volts	0.0001

Figure 39 - OBDII PID Support

SERIAL PORT MESSAGES

There are several types of messages that are autonomously sent to the serial port output:

Startup Banner

When the MCU is powered up, it prints a banner message with the version number.

```
*******************************************************
*               MCU - Dilithium Design                *
*           v1.0.91; Mar 20 2021 11:10:29             *
*******************************************************
```

LTC Discovery Messages

On startup, the MCU automatically discovers how many measurement devices are connected and prints a summary:

```
ltc A1 detected
ltc A2 detected
ltc A3 detected
ltc A4 detected
ltc A5 detected
ltc A6 detected
```

Changes in topology and diagnostic failures of the LTCs (or MBBs) are printed:

```
ltc A1 detected
ltc A2 detected
ltc A3 detected, diagnostic failure=0x1000
ltc A4 detected
ltc A5 NOT DETECTED!
ltc A6 NOT DETECTED!
```

Cell HVC and LVC Messages

When a cell reports an `HVC` or `LVC` alert, it is printed:

```
1/c25: in HVC !
1/c26: in HVC !
1/c25: OK
1/c26: OK
1/c12: in LVC !
1/c11: in LVC !
```

Thermistor Temperature Messages

When an enabled thermistor temperature exceeds the **`thmax`** threshold (or drops below the **`thmin`** threshold) a message is printed. The message includes the LTC number and thermistor number.

```
ltc2: T3 OVERTEMP!
```

And when it recovers a message is printed.

```
ltc2: T3 temperature OK
```

```
ltc A1:T0 25C
ltc A2:T0 22C
ltc A2:T2 18C
ltc A2:T3 18C
ltc A2:T4 17C
ltc A2:T5 17C
ltc A3:T0 22C
ltc A4:T0 23C
ltc A5:T0 25C
ltc A6:T0 24C
```

IsoSPI Error Messages

If an IsoSPI error occurs, a debug dump is noted:

```
bms>  0103 000f ffff ffff ffff ffff 4c66
 2703 000f ffff ffff ffff ffff 4c66
 0003 000f ffff ffff ffff ffff 4c66
 0203 000f ffff ffff ffff ffff 4c66
 2703 000f ffff ffff ffff ffff 4c66
 0203 000f ffff ffff ffff ffff 4c66
 0303 000f ffff ffff ffff ffff 4c66
 2703 000f ffff ffff ffff ffff 4c66
 0203 000f ffff ffff ffff ffff 4c66
 0903 000f ffff ffff ffff ffff 4c66
 2703 000f ffff ffff ffff ffff 4c66
 0703 000f ffff ffff ffff ffff 4c66
```

SERIAL PORT COMMANDS

Serial port commands are organized into five contexts: “SYStem”, “BMS”, “EVCC”, “INSTrumentation”, and “MCU”, with MCU is at the “top”.

The prompt shows the current context. To start, the user is in MCU context and the prompt is “**`mcu>`**”.

- The **`system`** command navigates to the SYSTEM context.
- The **`bms`** command navigates to the BMS context.
- The **`evcc`** command navigates to the EVCC context.
- The **`inst`** command navigates to the INSTrumentation context.
- The **`mcu`** or **`x`** command (x is a shortcut for eXit) navigate to the MCU context.

The contexts follow the functional partitioning of the MCU. The commands **`help`**, **`show`**, **`show config`**, and **`reset config`**, depend on context. The command **`help`** may be typed in any context and will give a list of commands and “command objects” available at that context. Likewise, the command **`show config`** in the EVCC context will print out charging related configuration and the command **`reset config`** in the EVCC context will only reset EVCC related configuration.

In many cases, command shortcuts are available. This is shown with the use of uppercase and lowercase letters. For example, the help lists the command **`SHow CONfig`**. The case of the letters indicate that this command may be entered as: **`show config`**, **`sh config`**, or simply **`sh con`**.

Althought certain commands like **`help`** or **`show config`** are context dependent, most commands can be typed in any context. It is possible, for example, to configure EVCC parameters while in the SYSTEM context.

help

The **`help`** command prints context dependent help. This command does not take any options.

Generally, **`help`** prints out the "command verbs" (such as **`set`** and **`show`**) and "command objects" (such as **`config`** and **`version`**) that are available but generally does not define all parameters or the precise syntax of the commands.

The help system is hierarchical. More information by typing a partial command. For example, by typing the command **`set`** (in the SYSTEM context) will give a list of commands available:

```
sys> set
  SEt INn <ifunc>       - map IN1 .. IN5 to input function
  SEt OUTn <ofunc>      - map OUT1 .. OUT5 to output function
  SEt CAN1BR <br>       - set CAN baud rate for CAN1
  SEt CAN2BR <br>       - set CAN baud rate for CAN2
```

In this example, the list of available of "input functions" to set in1 can be printed by typing:

```
sys> set in1
syntax: 'set in1 [HALL1|HALL2|LOOP|PSELECT|PLUGIN|SOCRES|none]'
```

At the MCU context, the **`help`** command shows hints about navigation and prints a few globally applicable commands. Below, **`help`** output for each context (SYS, BMS, EVCC and INST) is given.

mcu> help

```
mcu> help
--navigation------------------------------
  SYS                   - System
  BMS                   - Battery Management
  EVCC                  - Charging
  INST                  - Instrumentation
  eXit                  - exit to MCU
--status and configuration----------------
  SHow                  - MCU status
  SHow CONfig           - MCU configuration
  REset CONfig          - reset configuration to defaults
--firmware and database-------------------
  SHow VERsion          - firmware version
  UPGRADE               - upgrade firmware
```

sys> help

```
sys> help
--system---------------------------------
  SHow                  - system status
  SHow CONFig           - system configuration
  REset CONFig          - reset system configuration to defaults
  SHow INputs           - MCU input status
  SHow OUTputs          - MCU output status
  SHow CAN              - CAN interface status
--hardware configuration-----------------
  SEt INn <ifunc>       - map IN1 .. IN5 to input function
  SEt OUTn <ofunc>      - map OUT1 .. OUT5 to output function
  SEt CAN1BR <br>       - set CAN baud rate for CAN1
  SEt CAN2BR <br>       - set CAN baud rate for CAN2
--services-------------------------------
  ENable OBDII <cif>    - OBDII client
  ENable CANOPEN <cif> - CANOPEN client
  ENable CAN1DISFIL     - enable SID filter on CAN1
  DISable <service>     - disable a service
--diagnostics----------------------------
  MEasure <in>          - measure an input, <in> := [IN1 .. IN5, prox, pilot, ksi]
  ENable <out>          - enable an output, <out> := [OUT1 .. OUT5]
  DISable <out>         - disable an output
  TRace  <cif>          - enable can message trace on CAN1 or CAN2
  TRace- <cif>          - disable can message trace
  TRace OFF             - turn off tracing
```

bms> help

Note that **help** at the bms prompt is architecture dependent; the first line of the help indicates the architecture.

```
bms> help
--bms (LTC12)-----------------------------
  SHow                  - BMS status
  SHow CONFig           - BMS configuration
  REset CONFig          - reset BMS configuration to defaults
  SHow LTC              - Linear Technologies measurement chips
  SHow CMAP             - cell group map
  SHow CELLS            - cell data
  SHow STATS            - cell statistics
  SHow THermistor       - thermistors
--configuration--------------------------
  SEt ARCH <arch>       - set BMS architecture
  SEt CMAP <l> <p> <g> - map ltc to cell group
  LOCK                  - lock cell group map and cells
  -- per-cell --
  SEt HVC <v>           - high voltage cutoff threshold
  SEt LVC <v>           - low voltage cutoff threshold
  SEt HVCDELAY <s>      - delay asserting HVC after detection
  SEt LVCDELAY <s>      - delay asserting LVC after detection
  SEt HVCC <v>          - HVC clear threshold
  SEt LVCC <v>          - LVC clear threshold
  SEt BVMIN <v>         - discharge balancing voltage minimum
  -- thermistor --
  SEt THMAX <t>         - thermistor max temperature
  SEt THMIN <t>         - thermistor max temperature
  ENable THermistor     - enable thermistors
  DISable THermistor    - disable thermistors
```

```
--diagnostics----------------------------
  REset STATS          - reset cell statistics
```

evcc> help

```
evcc> h
--evcc-----------------------------------
  SHow                 - charge status
  SHow CONfig          - charge configuration
  REset CONfig         - reset EVCC configuration to defaults
--configuration--------------------------
  SEt PLUG <p>         - charge plug type
  SEt CHARGER  ...     - defines charger 1
  SEt CHARGER2 ...     - defines charger 2
  SEt CHARGER3 ...     - defines charger 3
  SEt CHARGER4 ...     - defines charger 4
  -- bulk charge parameters--
  SEt LINEV <v>        - line voltage
  SEt LINEC <c>        - line current
  SEt MAXV <v>         - maximum charging voltage
  SEt MAXC <a>         - maximum charging current
  SEt TERMC <a>        - termination charging current
  SEt TERMT <s>        - maximum charge time
  --lead acid charge parameters--
  SEt FIN_MAXV <v>     - finishing charge max voltage
  SEt FIN_MAXC <a>     - finishing charge max current
  SEt FIN_TERMT <s>    - finishing charge max charge time
  SEt FLT_MAXV <v>     - float charge max voltage
  SEt FLT_MAXC <a>     - float charge max current
  SEt FLT_TERMT <s>    - float charge max charge time
  --profiles--
  EDIT PROFILE <n>     - choose profile to edit; <id> := [1|2|3|4|OFFBOARD]
  DELETE PROFILE <n>   - choose profile to delete; <id> := [2|3|4]
  SEt PMAP <ps> <id>   - map pselect to profile
  --options--
  ENable ZEVCCS <cif>  - ZEVCCS charging
  ENable FCHARGE <cif> - FCHARGE charging
  ENable OBCHARGE <cif>- onboard charging
  ENable TOPBALANCE    - topbalancing
  ENable SLOWSTART     - slowstart
  DISable <option>     - disable an option
--diagnostics----------------------------
  SHow HIStory         - charge history
  REset HIStory        - reset charge history
  TRace CHarger        - enable onboard charger tracing
  TRace STATE          - enable onboard charger state tracing
  TRace- <t>           - disables trace
  TRace OFF            - disables all tracing
```

inst> help

```
inst> help
--instrumentation------------------------
  SHow                 - instrumentation status
  SHow CONfig          - instrumentation configuration
  REset CONfig         - reset INST configuration to defaults
--options--------------------------------
  ENable 3100R <cif>   - Curtis 3100R Gauge
  ENable SH_FUEL <cif> - Speedhut Fuel Gauge
  ENable SH_AMPS <cif> - Speedhut Ammeter
  ENable SH_TEMP <cif> - Speedhut Temperature Gauge
  DISable <option>     - disable an option
--configuration---------------------------
  SEt ISENSOR          - current sensor model
  SEt PACKKWH          - pack capacity
  SEt FUEL_0 <p>       - PWM setpoint for 0%
  SEt FUEL_25 <p>      - PWM setpoint for 25%
  SEt FUEL_50 <p>      - PWM setpoint for 50%
  SEt FUEL_75 <p>      - PWM setpoint for 75%
  SEt FUEL_100 <p>     - PWM setpoint for 100%
```

show

The **`show`** command is used to print out configuration and status. The **`show`** and **`show config`** commands, without additional parameters, are context dependent. The **`show`** command can also take a parameter. The following table lists the possible options.

show / MCU	
`show`	MCU Status
`show config`	MCU Configuration
show / SYSTEM	
`show`	System status
`show config`	System configuration
`show version`	Firmware version number
`show can`	CAN configuration and status
`show input`	Input configuration and status
`show output`	Output configuration and status
show / BMS	
`show`	BMS status
`show config`	BMS configuration
`show cmap`	Cell Map configuration
`show ltc`	LTC configuration (LTC12 and LTC18 architectures only)
`show mbb`	MBB configuration (A123 architecture only)
`show cells`	Cell status
`show stats`	Cell statistics
`show thermistor`	Thermistor status
show / EVCC	
`show`	Charging status
`show config`	Charging configuration
show / INST	
`show`	Instrumentation status
`show config`	Instrumentation configuration

mcu> show

```
mcu> show
  pack
    voltage : 229.50v
    cells   : 72
    mean    : 3.186v
    std dev : 0.008v
  alerts    : configuration not locked
  current   : 27.2A
  SOC       : 100%
  uptime: 7 hour(s), 36 minute(s), 57 second(s)
```

sys> show

```
sys> show
--inputs----------------------------------
  in1   2.54v  hall1   DHABS137 H = 23.5A
  in2   3.23v  hall2   DHABS137 L = 27.3A
  in3   1.63v
  in4   4.77v
  in5    inf
  prox   inf   J1772 connected
  pilot        not detected
  ksi      1   detected
--outputs---------------------------------
  out1   240   fuelg   soc = 100%
  out2   ---
  out3   ---
  out4   ---
  out5   ---   charge  disabled
--can1------------------------------------
  can1br    : 250 Kbps
  services  : obcharge, canopen
  tx        : 16223 [16219, 1, 1, 2], overflow= 5
  rx        :  3443 [3443, 0] overflow= 5, RX Passive
--can2------------------------------------
  can2br    : 250 Kbps
  services  :
  tx        : 64500 [64500, 0, 0], TX Passive, abt= 241, err1= 64499
  rx        :     0 [0, 0], RX Early Warn
```

bms> show

```
bms> show
  pack
    voltage : 229.50v
    cells   : 72
    mean    : 3.186v
    std dev : 0.008v
  alerts    : configuration not locked
```

evcc> show

When not charging, **show** would print:

```
evcc> show
  state     : STANDBY
  plug state: DISCONNETED
```

When charging, **show** prints additional information:

```
evcc> show
  state     : CHARGE
  plug state: CONNECTED
  chg time  : 1 mins
  duty cycle = 49%, 29.4A available
  voltage(t):  120.0V
  current(t):  12.0A
  charger   : tsm2500
    status  : 19 msgs sent; 20 msgs received
    voltage :  117.0V
    current :   12.8A
    charge  : 3.60Wh
```

inst> show

```
inst> show
  soc       : 100%
```

mcu> show config

At the MCU context, **`show config`** concatenates all configuration from all contexts.

sys> show config

```
sys> show config
--inputs---------------------------------
  in1       : hall1
  in2       : hall2
--outputs--------------------------------
  out1      : fuelg
  out5      : charge
--can------------------------------------
  can1br    : 250 Kbps
  can2br    : 250 Kbps
--services-------------------------------
  canopen   : enabled
  obcharge  : can1
```

bms> show config

```
bms> show config
  arch      : LTC12
  hvc       : 3.400v
  lvc       : 2.400v
  bvmin     : 3.000v
  thmax     : 50C
```

evcc> show config

In the EVCC context, **`show config`** command shows charging configuration parameters. The output of **`show config`** becomes progressively more complex as more features are enabled.

A simple example with only onboard charging defined:

```
evcc> show config
--charging-------------------------------
  plug      : J1772
  charger   : tsm2500
  maxv      :    20.0V
  maxc      :     2.0A
  termc     :     2.0A
  termt     :    72.0hr
```

An example with additional parameters defined:

```
evcc> show config
--charging--------------------------------
  plug       : J1772
  charger    : tsm2500
  charger2   : tsm2500_42
  charger3   : tsm2500_43
  charger4   : elcon
  linev      : 220.0V
  linec      :  30.0A
  evsewait   :   4.0hr
  maxv       : 155.0V
  maxc       :  15.0A
  termc      :   0.2A
  termt      :   6.0hr
  fin_maxv   : 160.0V
  fin_maxc   :   2.0A
  fin_termt  :   4.0hr
  flt_maxv   : 152.0V
  flt_maxc   :   0.5A
  flt_termt  :   0.0hr
  options    : j1772t2
             : topbalance
             : slowstart
```

If more than one charge profile is defined, **`show config`** will display the charge profiles in "tabular form". The charge profile selected for editing is indicated with a "*". Also, the profile map is shown.

Example output with multiple charge profiles and a profile switch enabled is shown below:

```
evcc> sh config
--charging--------------------------------
  plug       : J1772
  profile    :     1               2 (*)           3               4
  charger    : tsm2500         tsm2500
  linev      :                   220.0V
  linec      :                    25.5A
  maxv       :    20.0V           20.0V
  maxc       :     0.0A            2.0A
  termc      :     2.0A            2.0A
  termt      :    72.0hr          72.0hr
  pmap       :
    inf      :      x
    20K      :      x
    5K       :      x
    0        :      x
```

inst> show config

```
inst> show config
```

```
--soc------------------------------------
  isensor    : DHABS137
    hall1    : in1
    hall2    : in2
  packkwh    : 20.0
--instrumentation------------------------
  fuelg      : out1
        pwm : fuel_0 fuel_25 fue_l50 fuel_75 fuel_100
            :    95     125    150      175     240
```

show version

```
mcu> show version
v1.0.4; Sep  2 2021 11:10:29
```

show can

```
sys> show can
--can1-----------------------------------
  can1br     : 250 Kbps
  services   : obcharge, canopen
  tx         : 16219 [16219, 0, 0, 0]
  rx         :  3443 [3443, 0]
--can2-----------------------------------
  can2br     : 250 Kbps
  services   :
  tx         :     0 [0, 0, 0]
  rx         :     0 [0, 0]
```

show inputs

```
sys> show inputs
  in1   2.54v  hall1   DHABS137 H = 23.5A
  in2   3.23v  hall2   DHABS137 L = 27.0A
  in3   1.62v
  in4   4.77v
  in5    inf
  prox   inf   J1772 disconnected
  pilot        not detected
  ksi      1   detected
```

show outputs

```
sys> show outputs
  out1   240   fuelg   soc = 100%
  out2   ---
  out3   ---
  out4   ---
  out5   ---   charge  disabled
  pilot        EVSE disabled
```

show cmap

```
bms> show cmap
 ltc|pack|group| cells
----|----|-----|------------------------------------
 A1 |  1 |  1* |(c1  -c12 ) . . . . . . . . . . . .
 A2 |    |  2* |(c13 -c24 ) . . . . . . . . . . . .
 A3 |    |  3* |(c25 -c36 ) . . . . . . . . . . . .
 A4 |    |  4* |(c37 -c48 ) . . . . . . . . . . . .
 A5 |    |  5* |(c49 -c60 ) . . . . . . . . . . . .
 A6 |    |  6* |(c61 -c72 ) . . . . . . . . . . . .
```

The LTC number, pack, and group are shown. If the group has an asterisk (*) following, then the assignment was automatic.

Each cell entry is represented as a character, according to the following rule:

<space>	the cell was neither detected nor configured.
.	the cell was detected but not configured
?	the cell was configured but not detected
X	the cell was configured and detected

show ltc

The **show ltc** command is used for the LTC12 and LTC18 architectures.

```
bms> sh ltc
 ltc| status |  rev |  Van  |  Vdig |  Vref |  SOC  |  SUM   |  conf  |  disc
----|--------|------|-------------------------------------------------------------
 A1 |   ok   |6811-6| 5.094v| 3.206v| 3.006v| 38.19v  38.18v | 0x00000  0x00fff
 A2 |   ok   |6811-6| 5.130v| 3.209v| 3.003v| 38.19v  38.18v | 0x00000  0x00fff
 A3 |   ok   |6811-6| 5.042v| 3.214v| 3.004v| 38.25v  38.24v | 0x00000  0x00fff
 A4 |   ok   |6811-6| 5.087v| 3.210v| 3.003v| 38.24v  38.23v | 0x00000  0x00fff
 A5 |   ok   |6811-6| 5.070v| 3.219v| 3.004v| 38.32v  38.32v | 0x00000  0x00fff
 A6 |   ok   |6811-6| 5.098v| 3.191v| 3.003v| 38.36v  38.36v | 0x00000  0x00fff
```

The **status** value should be **ok**. If not, then this field will show a hex failure value.

The **rev** shows the LTC chip hardware revision. Expect to see either ltc6804, ltc6811, and ltc6813

The values **Van**, **Vdig**, **Vref** are internal voltage measurements on the chip. The nominal values are be 5v, 3v, and 3v respectively.

The values of **SOC** and **SUM** is the total voltage managed by the chip. These numbers are calculated in two different ways. The LTC can measure the cell group voltage (SOC) and the MCU adds up individual cell measurements (SUM). These numbers should be close to each other.

The **conf** is a hexadecimal representation of which cells have been configured for this LTC using the **lock** command. The **disc** field is a hexadecimal representation of which cells have been discovered. Note that an equivalent, more user friendly, representation of this data is given in **show cmap**.

show mbb

The **show mbb** command is used for the A123 architecture.

```
mcu> show mbb
 mbb| status |  min  |  max  |  mean | flt_xx |  conf  |  disc
----|--------|-------|-------|-------|-------------------------
  1 |   OK   | 3.119v| 3.142v| 3.133v| in     |0x01fff  0x01fff
  2 |   OK   | 3.110v| 3.144v| 3.132v| in     |0x01fff  0x01fff
  3 |   OK   | 3.126v| 3.189v| 3.177v| in     |0x01fff  0x01fff
  5 |   OK   | 3.126v| 3.189v| 3.177v| in     |0x01fff  0x01fff
```

show cells

The **`show cells`** command lists current cell voltage measurements and status.

```
bmsc> show cells
 c1 - 3.042v +      c13- 3.027v +      c25- 2.718v -      c37- 2.713v -
 c2 - 3.472v >HVC   c14- 2.995v        c26- 2.918v        c38- 2.706v -
 c3 - 3.421v >HVCC  c15- 3.054v +      c27- 2.959v        c39- 2.908v
 c4 - 3.012v +      c16- 2.993v        c28- 3.080v ++     c40- 2.913v
 c5 - 3.052v +      c17- 3.045v +      c29- 2.706v -      c41- ??????
 c6 - 3.001v +      c18- 2.991v        c30- 2.910v        c42- ??????
 c7 - 2.799v        c19- 2.766v        c31- 2.708v -      c43- ??????
 c8 - 2.788v        c20- 2.773v        c32- 2.920v        c44- ??????
 c9 - 2.774v        c21- 2.777v        c33- 2.714v -      c45- ------
 c10- 2.771v        c22- 2.783v        c34- 2.980v        c46- ------
 c11- 2.783v        c23- 2.783v        c35- 2.722v -      c47- ------
 c12- 2.788v        c24- 2.770v        c36- 2.917v        c48- ------
```

If the cell value shown is “`------`" then the cell voltage is less than 20mv.
If the cell value shown is “`??????`” then the is configured, but not present.

An individual cell voltage may be an Indicator Flag. There is room for only one Indicator Flag,

Indicator Flag	Description
>HVC	Cell in HVC
>HVCC	Cell > HVCC
<LVCC	Cell < LVCC
<LVC	Cell in LVC
++	SDM is > 1.5
+	SDM is > 1.0
-	SDM is < 1.0
--	SDM is < 1.5

show stats

The **`show stats`** command lists current cell statistics.

Mean cell voltage and **std deviation** (standard deviation) are calculated over all cells. The **`min`** and **`max`** values are the lowest and highest value of the cell since power on or since last time reset stats was entered.

The **`variance`** is the difference, in volts, of the current cell voltage with the average cell voltage. For an explanation of **`sdm`**, **`min`**, **`max`**, and **`delta`**, see the text.

```
bms> sh stats
total|-mean cell voltage------|-standard deviation----------------
     | 3.187v                 |  0.006v
pack1|-voltage----min----max--|----deviation-----min---max--delta-
 c1  | 3.183v   3.182v 3.183v | -0.004v  -0.7s  -0.7s -0.6s  0.1s
 c2  | 3.186v   3.185v 3.186v | -0.002v  -0.3s  -0.3s -0.2s  0.1s
 c3  | 3.187v   3.186v 3.187v | -0.000v  +0.0s  +0.0s +0.0s  0.0s
 c4  | 3.182v   3.181v 3.182v | -0.005v  -0.8s  -0.8s -0.7s  0.1s
 c5  | 3.196v   3.195v 3.196v |  0.008v  +1.3s  +1.2s +1.3s  0.1s
 c6  | 3.181v   3.180v 3.181v | -0.006v  -0.9s  -1.0s -0.9s  0.1s
 c7  | 3.184v   3.183v 3.184v | -0.003v  -0.5s  -0.5s -0.4s  0.1s
 c8  | 3.182v   3.181v 3.182v | -0.006v  -0.9s  -0.9s -0.8s  0.1s
 c9  | 3.183v   3.182v 3.183v | -0.005v  -0.7s  -0.7s -0.6s  0.1s
 c10 | 3.186v   3.185v 3.186v | -0.002v  -0.3s  -0.3s -0.2s  0.1s
 c11 | 3.185v   3.184v 3.185v | -0.003v  -0.4s  -0.5s -0.3s  0.2s
 c12 | 3.182v   3.181v 3.182v | -0.005v  -0.8s  -0.9s -0.7s  0.2s
 c13 | 3.183v   3.182v 3.183v | -0.004v  -0.7s  -0.7s -0.6s  0.1s
 c14 | 3.186v   3.185v 3.186v | -0.002v  -0.2s  -0.3s -0.2s  0.1s
 c15 | 3.188v   3.187v 3.188v |  0.001v  +0.0s  +0.0s +0.0s  0.0s
 c16 | 3.186v   3.185v 3.186v | -0.001v  -0.2s  -0.2s -0.1s  0.1s
 c17 | 3.186v   3.185v 3.186v | -0.001v  -0.2s  -0.3s -0.2s  0.1s
 c18 | 3.183v   3.181v 3.183v | -0.005v  -0.7s  -0.8s -0.7s  0.1s
 c19 | 3.187v   3.187v 3.188v | -0.000v  +0.0s  +0.0s +0.0s  0.0s
 c20 | 3.182v   3.182v 3.183v | -0.005v  -0.8s  -0.8s -0.7s  0.1s
 c21 | 3.185v   3.184v 3.185v | -0.003v  -0.4s  -0.4s -0.3s  0.1s
 c22 | 3.186v   3.185v 3.186v | -0.001v  -0.2s  -0.2s -0.1s  0.1s
 c23 | 3.187v   3.186v 3.187v | -0.000v  +0.0s  +0.0s +0.0s  0.0s
 c24 | 3.185v   3.184v 3.185v | -0.003v  -0.4s  -0.4s -0.3s  0.1s
```

show thermistor

```
bms>sh th
 ltc|   T0   |   T1   |   T2   |   T3   |   T4   |   T5
----|----------------------------------------------------
 A1 |   26C+            NC      NC
 A2 |   23C             18C     18C     18C     18C
 A3 |   23C
 A4 |   23C
```

Value	Discovered	Configured	TH_CENSUS_ERROR	TH_OVERTEMP
<blank>				
NC	no	Yes	Yes	
SHORT	yes	Yes	Yes	
(xx)	yes	No		
xx+	yes	Yes		yes
xx	yes	Yes		

show history

The **show history** command displays data about the last sixteen charge cycles.

In the first example, the system has no charge history.

```
evcc> show history
no charge history
```

The following example shows charge history, with different “termination reasons”. The termination reason contains the reason that the charge cycle stopped. In this example, in the most recent charge attempt, the user disconnected the J1772 plug one minute after charging started. (EVSE disc, 1 mins). The previous attempt (“-1”) shows a normal charge completion with a charge time of 214 minutes and includes the number of watt hours delivered.

```
evcc> show history
      |   term   |  charge |           |   watt | maximum| maximum|  ending|
  num |  reason  |   time  |  charger  |  hours | voltage| current| current|
--------------------------------------------------------------------------
 last | overtemp |  15 mins|tsm2500    |   383Wh| 120.0V |  12.8A |  12.8A |
  - 1 | plug disc|   1 mins|tsm2500    |    19Wh| 120.0V |  12.8A |  12.8A |
  - 2 |   normal | 214 mins|tsm2500    |  3249Wh| 152.9V |   7.9A |   1.9A |
```

The full set of “term reason” codes is:

Term reason	Description
`normal`	Normal completion
`charge t/o`	Charge timeout
`plug disc`	Charge plug disconnected
`hardware`	MCU Hardware alert
`cellcensus`	Cell census alert
`th census`	Thermistor census alert
`HVC`	High Voltage Cutoff alert
`LVC`	Low Voltage Cutoff alert
`overtemp`	Thermistor overtemperature alert
`undertemp`	Thermistor undertemperature alert
`not locked`	BMS configuration not locked
`cell loop`	Cell loop
`EVSE t/o`	EVSE timeout
`chgr t/o`	Charger CAN RX timeout
`no pack`	No pack detected
`chgr stop`	Charger unexpectedly stopped delivering current
`no SEQ1`	FCHARGE: no SEQ1 signal
`no SEQ2`	FCHARGE: no SEQ2 signal
`chgr t/o`	FCHARGE: charger timeout
`sys error`	FCHARGE system error
`chgr flt`	FCHARGE: charger fault
`out cur lo`	FCHARGE: output current low
`evincompat`	FCHARGE: EV incompatible
`ch not on`	FCHARGE: Charger not on
`stop`	FCHARGE: Stopped by user

When multiple chargers are configured, the format of the charge history is modified to show the contribution of each charger.

```
evcc> show history

      |   term    |  charge |            |   watt | maximum| maximum|  ending|
  num |  reason   |   time  | charger    |  hours | voltage| current| current|
------------------------------------------------------------------------------
 last | EVSE disc|   2 mins|tsm2500    |     6Wh| 127.8V |   2.2A |   0.0A |
                           |tsm2500_42|     6Wh| 127.5V |   2.0A |   0.0A |
                           |TOTAL     |    12Wh| 127.8V |   4.2A |   0.0A |
```

set

The **`set`** command assigns a value to a parameter. In most cases, this command takes two arguments: a parameter name and a parameter value.

```
set <param> <value>
```

The following table lists the possible options.

set / SYSTEM	**Value**	**Description**

`set in1`
`set in2`
`set in3`
`set in4`
`set in5`
	Input Function, See Figure 8	Input Function mapping

`set out1`
`set out2`
`set out3`
`set out4`
`set out5`
	Output Function, see. Figure 9	Output Function mapping

`set can1br`	**`125`**, **`250`**, **`500`**	CAN1 Baud Rate
`set can2br`	**`125`**, **`250`**, **`500`**	CAN2 Baud Rate
set / BMS		
`set arch`	**`ltc12`**, **`ltc18`**, **`a123`**	BMS Architecture
`set hvc`	3.0 to 4.6 volts	High Voltage Cutoff Threshold
`set lvc`	1.8 to 4.0 volts	Low Voltage Cutoff Threshold
`set hvcdelay`	0 to 10 seconds	Delay before declaring HVC Alert
`set lvcdelay`	0 to 10 seconds	Delay before declaring LVC Alert
`set hvcc`	0 to 4.6 volts (0=disable)	High Voltage Cutoff Clear Threshold
`set lvcc`	0 to 4.0 volts (0=disable)	High Voltage Cutoff Clear Threshold
`set bvmin`	0 to 5.0 volts (0=disable)	Balance Voltage Minimum Voltage
`set thmax`	0 to 100 degrees C	Thermistor Overtemperature Threshold
`set thmin`	-20 to 30 degrees C	Thermistor Undertemperature Threshold
`set cmap`	See example below	Cell Map
set / EVCC		

`set charger`
`set charger2`
`set charger3`
`set charger4`
	See text below	Set charger type (and optional phase)

`set plug`	**`J1772`**, **`J1772T2`**, **`direct`**	Plug protocol
`set linev`	100 to 300 Volts	Line Voltage
`set linec`	0 to 100 Amps (or **`J1772`**)	Line Current
`set maxv`	0 to 800 Volts	Maximum Charge Voltage

`set maxc`	0 to 140 Amps	Maximum Charge Current
`set termc`	1 to 20 Amps	Termination Charge Current
`set termt`	0 to 672 hours	Termination Charging Time
`set fin_maxv`	0 to 800 Volts (0=disable)	Finishing Charge Voltage
`set fin_maxc`	0 to 30 Amps	Finishing Charge Current
`set fin_termt`	0 to 672 Hours	Finishing Charger Termination Time
`set flt_maxv`	0 to 800 Volts (0=disable)	Float Charge Voltage
`set flt_maxc`	0 to 10 Amps	Float Charge Current
`set flt_termt`	0 to 672 Hours (0=”forever”)	Float Charger Termination Time
`set evsewait`	0 to 672 Hours	Time of use “wait time”
`set pmap`	See text below	Profile Selection Map
set / INST		
`set packkwh`	0 to 120 KWHr	Pack Capacity
`set isensor`	Current Sensor Support See Figure 28	Current Sensor Type
`set soc`	0 to 100 percent	Sets the SOC value manually

**`set fuel_0`
`set fuel_25`
`set fuel_50`
`set fuel_75`
`set fuel_100`**
	0 to 255	Fuel gauge duty cycle setting for 0, 25, 50, 75 and 100%

set charger<n>

This sets the charger type. The first charger is named “charger”. Chargers 2 through 4 are named “charger2”, “charger3”, “charger4”.

The following chargers are supported:

- **`tsm2500`**
- **`tsm2500_41`**
- **`tsm2500_42`**
- **`tsm2500_43`**
- **`elcon`**
- **`elcon_e7`**
- **`elcon_e8`**
- **`elcon_e9`**
- **`lear`**

The following command sets a single charger

```
evcc> set charger tsm2500
```

The following command sets a second charger

```
evcc> set charger2 tsm2500_42
```

A charger may be deleted using the command

```
evcc> set charger2 none
```

This command is also used to associate a phase with a charger, which is used with J1772 Type 2 charging. The following example defines three chargers and associates each of them with a different phase.

```
evcc> set charger tsm2500 L1
evcc> set charger2 tsm2500_41 L2
evcc> set charger3 tsm2500_42 L3
```

The phase association may be reset to “none” using the command:

```
evcc> set charger tsm2500 none
```

set pmap

This command sets the profile map. The syntax is

```
set pmap <ps> <id>;    <ps> := [inf|20K|5K|0]
                       <id> := [1|2|3|4]
```

So, the following command will map “inf” at pselect to choose profile 2:

```
evcc> set pmap inf 2
```

set cmap (LTC12 or LTC18)

If the BMS architecture is set to LTC12 or LT18, the **`set cmap`** is used to set the ltc to cell group mapping. The syntax of this command is

```
set cmap [A|B]<ltc> <pack> <group>
```

where `A` and `B` choose the IsoSPI interface, `<ltc>` is from 1 to 8, `<pack>` is from 1 to 8 and `<group>` is from 1 to 8.

set cmap (A123)

If the BMS architecture is set to A123, the **`set cmap`** is used to set the mbb to cell group mapping. The syntax of this command is

```
set cmap <mbbId> <pack> <group>
```

where `<mbbId>` is from 1 to 16, `<pack>` is from 1 to 8 and `<group>` is from 1 to 8.

enable | disable

The **`enable`** commands can be used to enable an option (which requires a single parameter, the option name). Or the commands can also be used to enable or disable a CAN service (which requires two parameters: the service name, and the CAN interface). And, finally, these commands can be used to enable or disable thermistors, which supports a variety of possible keywords.

enable (option)	
`enable balance`	BMS cell discharge balancing
`enable j1772t2`	EVCC J1772 Type 2 operation
`enable slowstart`	EVCC slowstart option
`enable topbalance`	EVCC topbalancing option
enable (can service)	
`enable obdii   <can1|can2>`	OBDII service
`enable canopen <can1|can2>`	CANOPEN service
`enable a123    <can1|can2>`	A123 service
`enable zevccs  <can1|can2>`	ZEVCCS service
`enable obcharge <can1|can2>`	Onboard charger
`enable fcharge  <can1|can2>`	Fastcharge service
`enable 3100r    <can1|can2>`	31004 display service
`enable sh_fuel  <can1|can2>`	Speedhut fuel gauge
`enable sh_amps  <can1|can2>`	Speedhut amps gauge
`enable sh_temp  <can1|can2>`	Speedhut temp gauge
enable (thermistor)	
enable thermistor	see text

Example of enabling and disabling options:

```
evcc> enable slowstart
evcc> disable topbalance
```

Example of enabling a service:

```
sys> enable obdii can1
```

enable | disable thermistor

The **`enable thermistor`** is used to enable thermistor operation. The command **`disable thermistor`** disables thermistor operation. The list of which thermistors are enabled can be determined by the command **`show thermistor`**.

If a thermistor is enabled, then if the measured value of the thermistor is greater than **`thmax`** then the BMS will generate a `THERM_OVERTEMP` alert.

To enable all thermistors that have been detected, use the command:

```
bms> enable thermistor all
```

To enable or disable individual thermistors, use the commands:

```
      enable thermistor <ltc> <therm_number>
      disable thermistor <ltc> <therm_number>
```

where `<ltc>` is from 1 to 8, and `<therm_number>` is from 1 to 5.

All die thermistors may be enabled using the command:

```
      enable thermistor <ltc> die
```

All external thermistors may be enabled using the command:

```
      enable thermistor <ltc> ext
```

The thermistor threshold temperature is determined by the configuration parameter **`thmax`**. When a thermistor temperature exceeds **`thmax`**, a `THERM_OVERTEMP` alert is present in the **`show`** command. The measured value can be determined by the **`show thermistor`** command.

Examples:

```
bmsc> en th 1 1
bmsc> en th 1 2
bmsc> en th 2 1
```

edit | delete

The “edit/delete” command verb pair is only used to edit charge profiles.

edit profile <n>

The **`edit profile`** command can be used to create, if necessary, and set the edit profile. This command takes one parameter, a number from 1 to 4 for the four onboard profiles, or the keyword **`offboard`** to edit the offboard profile.

```
evcc> edit profile 3
```

Each profile contains the following information:

`charger`	Charger Type and Phase
`charger2`	
`charger3`	
`charger4`	
`evsewait`	Time of use charging
`linev`	Line Parameters

linec	
maxv	Bulk Charge Parameters
maxc	
termc	
termt	
fin_maxv	Finishing Charge Parameters
fin_maxc	
fin_termt	
flt_maxv	Float Charge Parameters
flt_maxc	
flt_termt	

delete profile <n>

The **delete profile** command can be used to delete Profiles 2-4. This parameter takes one parameter, a number from 2 to 4. It is not possible to delete Profile 1.

```
evcc> delete profile 3
```

reset

reset config

The **reset config** command resets configuration to defaults. This command is context dependant. If at the MCU context, all configuration will be reset. Otherwise, only the configuration for the active context is reset.

```
mcu> reset config
SYS config..reset
BMS config..reset
EVCC config..reset
INST config..reset
mcu>
```

reset stats

The **reset stats** command clears out the cell statistics.
Example:

```
bms> reset stats
```

reset history

The **reset history** command resets the charge history.

```
evcc> reset history
charge history has been reset
```

measure

The **measure** command is used to verify the A/D inputs. When this command is issued, the MCU will repeatedly measure and print the value of an analog input. The command will run until the user stops the command by typing any character.

The measure command takes one argument which can either a Physical Input (IN1, IN2, IN3, IN4, IN5), a fixed function input IN6=Proximity, IN7=KSI, or can be an Input Function.

The **measure** command with no parameters will display the expected values of the A/D inputs.

Examples:

```
sys> measure prox
sys>   in6    395R  prox    J1772 connected
  in6    397R  prox    J1772 connected
  in6    406R  prox    J1772 connected
  in6    410R  prox    J1772 connected
  in6    409R  prox    J1772 connected
  in6    395R  prox    J1772 connected

sys> measure in4
sys>   in4   1.01v
  in4   1.01v
  in4   1.01v
  in4   1.01v
  in4   1.01v

sys>

evcc > measure loop
evcc> Loop A/D= 4.97V
Loop A/D= 4.97V
Loop A/D= 4.97V
Loop A/D= 4.97V
Loop A/D= 4.97V
```

trace

The **trace** command enables various forms of message or state tracing. These commands show a timestamp (uptime) and can be useful for logging or debugging. CHARGER, STATE, and CAN tracing may be independently enabled.

Trace configuration is stored in EEPROM and is present after reboot.

trace charger

The **trace charger** command displays messages from the charger. This trace also shows the current number of charging watts and the accumulated watt-hours of charge.

```
evcc> trace charger
charger tracing is now ON
evcc> 00:08:22.7  V=148.6, A= 7.9, W=1173, Wh= 0.96, TMP = 26C
00:08:23.1  V=148.6, A= 7.9, W=1173, Wh= 1.12, TMP = 26C
00:08:23.6  V=148.6, A= 7.9, W=1173, Wh= 1.28, TMP = 26C
```

trace can1 | can2

The **trace can1** and **trace can2** command displays canbus messages to and from the charger. Each line gives a timestamp, the originator of the message (if known), the CAN ID and CAN message contents, in hexadecimal.

```
evcc> trace can
canbus tracing is now ON
evcc> 00:02:20.9        evcc: 18e54024 fc c8 00 6c 0c 01 ff ff
00:02:21.4        evcc: 18e54024 fc c8 00 6c 0c 01 ff ff
00:02:21.9        evcc: 18e54024 fc c8 00 6c 0c 01 ff ff
00:02:22.4        evcc: 18e54024 fc c8 00 6c 0c 01 ff ff
00:02:22.5  tsm2500   : 18eb2440 42 f7 41 fd 00 fe 12 dd
00:02:22.9        evcc: 18e54024 fc c8 00 6c 0c 01 ff ff
00:02:22.9  tsm2500   : 18eb2440 04 fd 13 02 80 0c 3f ff
00:02:23.4        evcc: 18e54024 fc c8 00 6c 0c 01 ff ff
```

trace state

The **trace state** command displays internal EVCC state transitions. For the standard EVCC, there is also J1772 state.

Here is an example of state trace output that shows the charger plug being plugged in and unplugged, also showing the attempt to charge.

```
evcc> trace state
state tracing is now ON
evcc>
evcc> 00:00:31.7 j1772=LOCKED
00:00:31.7 old state=DRIVE, new state=WARMUP, term rsn=0
00:00:31.8 old state=WARMUP, new state=CHARGE, term rsn=0
00:00:52.2 j1772=CONNECTED
00:00:52.3 j1772=WAITING FOR DISC
00:00:52.4 old state=CHARGE, new state=STANDBY, term rsn=EVSE UNLOCKED
00:00:56.1 j1772=DISCONNECTED
00:00:57.4 old state=STANDBY, new state=DRIVE, term rsn=0
```

trace off

The **trace off** command turns off all tracing.

```
evcc> tr off
all tracing is now OFF
```

lock

The **`lock`** command is used to configure the discovered configuration. See text.

```
bmsc> lock
configuration locked
```

upgrade

Firmware upgrade is performed by using the serial port and uses a special bootloader application program. To begin the upgrade process, use the upgrade command. For more details, see *Firmware Upgrade*, above.

```
mcu> upgrade

 ***
 ***                   Starting MCU Upgrade                       ***
 *** 1) Exit from the terminal application                        ***
 *** 2) Start the bootloader and download a new .hex file         ***
 *** 3) Restart                                                   ***
```

Warrantee and Support

The MCU is warranted to be free from defects in components and workmanship under normal use and service for a period of 1 year.

The product is intended for non-commercial use by hobbyists. The warranty does not cover defects arising from miswiring, abuse, negligence, accidents, or reverse engineering. Dilithium Design shall not be responsible for any incidental or consequential damages.

Dilithium Design reserves the right to make changes or improvements in design or manufacturing without assuming any obligation to change or improve products previously manufactured and / or sold.

For errors in this document, or comments about the product, contact djmdilithium@gmail.com

Document Revision

Rev 1.0	Mar 2021	Initial Version
Rev 1.1	Sep 2021	Production Release
Rev 1.2	Nov 2021	Minor Edits