CAN bus protocol for BMS12 V3 modules

Ian Hooper, December 2017

Introduction

This document is intended to assist people integrating the ZEVA BMS12 V3 modules with their own BMS master controller over CAN bus. It describes the required specifications for the CAN bus, packet format and communications protocols.

Standard bus settings

The standard bus speed is 250kbps. Packet format is CAN 2.0B, which uses 29-bit IDs. Please contact us if you require modules programmed to suit a different bus speed or packet format. All packets are the data frame type (no remote frames).

Physical Layer

Most ZEVA devices use 5-pin Molex Eurostyle connectors for the CAN bus. Two pins are for the CAN H and CAN L signals, two pins for the 12V supply and Ground, and the fifth pin for transferring shield grounding along the chain of devices (and should be attached to the vehicle chassis at one end of the CAN bus chain). The BMS12 modules power themselves off the CAN bus, so it is recommended that the 12V supply should be capable of supplying up to 20mA per BMS12 module.

CAN bus transceivers often draw around 20mA per node, which can accumulate to a significant quiescent load on the vehicle's 12V auxiliary battery, causing it to go flat if the CAN bus is left powered up while the vehicle is idle for extended periods of time. As such it is recommended that the CAN bus is only enabled when the vehicle is either driving or being charged. (This is handled automatically by all CAN-enabled ZEVA BMS master controllers.)

We recommend using Shielded Twisted Pair (STP) cable, with two pairs of conductor – one pair for CAN H and CAN L, the other for Ground and +12V supply. Conductors should be around AWG20-24 for sufficient mechanical strength and current rating. A good option is Belden 8723, or equivalent. CAN buses work best as a single chain of devices, without any branching, and with 120Ω termination resistors at both ends to prevent signal reflection.

Packet Summary

There are five different packet types used by the BMS12 V3 modules, one being a data request sent by the master controller and four for data replies sent by the BMS module. Packet IDs are provided in **decimal** format (*not hexadecimal*).

The module ID selector on the board determines the **Base Packet ID** (BPID), in increments of 10 and starting at 300. For example, module ID 0 will use packet IDs 300 to 304, module ID 1 will use packet IDs 310 to 314, and so on.

CAN bus was invented as a realtime, low latency bus so the standard only supports small data packets up to 8 bytes in length, which can result in the need to split data up across multiple packets.

Packet IDs and Structure

BPID + 0: Request Data *(Master to BMS12, 2 bytes)*

The master controller should send the BMS module a Request Data packet with 2 bytes containing the desired shunt voltage, in millivolts ("big endian" format). If no request packet is received every 1 second or less, shunt balancers will be disabled. Sending a shunt voltage of 0 will disable shunts.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	16-bit shunt voltage, high byte							
Byte 2	16-bit shunt voltage, low byte							

BPID + 1: Reply Data 1 *(BMS12 to Master, 8 bytes)*

After receiving a Request Data packet at the appropriate ID, the BMS module will send four Reply Data packets in quick succession. The first three contain four 16-bit cell voltages, in millivolts (biig endian format), and the fourth contains two 8-bit temperatures in degrees celcius with a +40 offset (e.g a value of 65 means 25˚C). Values of zero for cells or temperatures indicate no cell or temp sensor present respectively.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Voltage 1, high byte							
Byte 2	Voltage 1, low byte							
Byte 3	Voltage 2, high byte							
Byte 4	Voltage 2, low byte							
Byte 5	Voltage 3, high byte							
Byte 6	Voltage 3, low byte							
Byte 7	Voltage 4, high byte							
Byte 8	Voltage 4, low byte							

BPID + 2: Reply Data 2 *(BMS12 to Master, 8 bytes)*

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Voltage 5, high byte							
Byte 2	Voltage 5, low byte							
Byte 3	Voltage 6, high byte							
Byte 4	Voltage 6, low byte							
Byte 5	Voltage 7, high byte							
Byte 6	Voltage 7, low byte							
Byte 7	Voltage 8, high byte							
Byte 8	Voltage 8, low byte							

BPID + 2: Reply Data 3 *(BMS12 to Master, 8 bytes)*

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Voltage 9, high byte							
Byte 2	Voltage 9, low byte							

Byte 3	Voltage 10, high byte
Byte 4	Voltage 10, low byte
Byte 5	Voltage 11, high byte
Byte 6	Voltage 11, low byte
Byte 7	Voltage 12, high byte
Byte 8	Voltage 12, low byte

BPID + 2: Reply Data 4 *(BMS12 to Master, 2 bytes)*

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Temperature 1, in ˚C +40							
Byte 2	Temperature 2, in ˚C +40							

Code examples

The protocol should be fairly simple to implement, only needing to combine some pairs of 8-bit numbers into 16-bit, and vice versa.

Sending request with shunt voltage

```
unsigned int exampleVoltage = 3600; // millivolts
data[0] = exampleVoltage >> 8;
data[1] = exampleVoltage & 0xFF;
CanTX(300 + 10*moduleID, data);
```

Unpacking cell voltages

```
unsigned int voltage[4];
for (int n=0; n<4; n++)
     voltage[n] = (receivedData[n*2]<<8) + receivedData[n*2+1];
```

Unpacking temperatures

```
int temperature1 = receivedData[0]-40;
int temperature2 = receivedData[1]=40;
```