Jikong BMS CAN Communication Protocol

Clean reference version - easier to read, decode, and implement

Important integration warning

This document describes a simple one-way CAN telemetry protocol from the BMS to a display/instrument. It does not define inverter handshakes, requested charge/discharge limits, inverter commands, contactor control, or any BMS receive frames. Treat it as a display/telemetry protocol unless your inverter documentation explicitly supports these exact frames.

1. One-page quick reference

Item	Value / rule
CAN standard	CAN 2.0A, standard 11-bit identifier
Bit rate	250 kbit/s
Data length	8 data bytes per frame
Direction	BMS transmits; display/instrument/controller receives
Byte order	Little-endian for multi-byte values. Low byte is sent first.
Source address	0xF4
Current sign	Positive = discharge. Negative = charge.
Reserved bytes	Shown as XX in the source. Ignore them when decoding.

CAN ID	Name	Purpose	Period / trigger
0x02F4	BATT_ST	Pack voltage, pack current, SOC, discharge time	20 ms
0x04F4	CELL_VOLT	Highest and lowest cell voltage and cell numbers	100 ms
0x05F4	CELL_TEMP	Highest, lowest, and average cell temperature	100 ms
0x07F4	ALM_INFO	Alarm status and alarm level bitfield	Event triggered; repeats while alarm is active

Core decoder formulas

```
u16_le(byte_n, byte_n1) = byte_n + (byte_n1 << 8)
pack_voltage_V = u16_le(data[0], data[1]) * 0.1
pack_current_A = u16_le(data[2], data[3]) * 0.1 - 400
soc_percent = data[4]
discharge_time_h = u16_le(data[6], data[7])
temperature_degC = raw_byte - 50
alarm_level = (data[bit_start // 8] >> (bit_start % 8)) & 0x03
```

2. Data conventions

- Frame identifiers are 11-bit standard CAN identifiers. The identifier combines the function code and BMS source address. For example, function 0x02 from source address 0xF4 gives CAN ID 0x02F4.

- Multi-byte numeric fields are little-endian: byte 0 is the low byte, byte 1 is the high byte.

- Bit positions start from bit 0 at the least significant bit of data byte 0. This is especially important for the alarm message.

- The source uses XX for unused/reserved bytes. Do not use XX bytes as decoded data. In examples in this cleaned document they are shown as 00 unless the value matters.

- For pack current, the raw 16-bit value is offset by -400 A after scaling. Formula: current_A = raw * 0.1 - 400.

Use direct cell numbers unless proven otherwise

The original field tables list an offset of 1 for cell-number fields, but the examples decode the raw byte directly as the cell number. Example: byte 0x05 is described as cell 5, not cell 6. This cleaned version follows the examples and uses raw cell number = byte value. Confirm this against your specific hardware if cell indexing matters.

3. Message 0x02F4 - BATT_ST - battery status

This is the main pack status frame. It is sent every 20 ms.

Bytes	Bits	Field	Raw type	Decode formula	Unit / notes
0-1	0-15	Battery voltage	U16 little-endian	raw * 0.1	V
2-3	16-31	Battery current	U16 little-endian	raw * 0.1 - 400	A; positive = discharge, negative = charge
4	32-39	SOC	U8	raw	%
5	40-47	Reserved	-	ignore	Source shows XX
6-7	48-63	Discharge time	U16 little-endian	raw	hours

Worked example

CAN ID	Data bytes	Decoded result
0x02F4	13 01 D7 11 33 00 64 00	Voltage = 0x0113 * 0.1 = 27.5 V Current = 0x11D7 * 0.1 - 400 = 56.7 A SOC = 0x33 = 51% Discharge time = 0x0064 = 100 h

4. Message 0x04F4 - CELL_VOLT - cell voltage

This frame reports the maximum and minimum cell voltages and the cell numbers where they occur. It is sent every 100 ms.

Bytes	Bits	Field	Raw type	Decode formula	Unit / notes
0-1	0-15	MaxCellVolt	U16 little-endian	raw	mV
2	16-23	MaxCvNO	U8	raw	Cell number of highest cell
3	24-31	Reserved	-	ignore	Source shows gap/XX
4-5	32-47	MinCellVolt	U16 little-endian	raw	mV
6	48-55	MinCvNO	U8	raw	Cell number of lowest cell
7	56-63	Reserved	-	ignore	Source shows XX

Worked example

CAN ID	Data bytes	Decoded result
0x04F4	8C 0A 05 00 92 09 08 00	Max cell voltage = 0x0A8C = 2700 mV; max cell number = 5 Min cell voltage = 0x0992 = 2450 mV; min cell number = 8

5. Message 0x05F4 - CELL_TEMP - cell temperature

This frame reports maximum, minimum, and average cell temperature. It is sent every 100 ms.

Byte	Bits	Field	Raw type	Decode formula	Unit / notes
0	0-7	MaxCellTemp	U8	raw - 50	deg C
1	8-15	MaxCtNO	U8	raw	Cell number of highest temperature
2	16-23	MinCellTemp	U8	raw - 50	deg C
3	24-31	MinCtNO	U8	raw	Cell number of lowest temperature
4	32-39	AvrgCellTemp	U8	raw - 50	deg C
5-7	40-63	Reserved	-	ignore	Source shows XX

Worked example

CAN ID	Data bytes	Decoded result
0x05F4	48 06 2F 01 3F 00 00 00	Max temperature = 0x48 - 50 = 22 deg C; max cell number = 6 Min temperature = 0x2F - 50 = -3 deg C; min cell number = 1 Average temperature = 0x3F - 50 = 13 deg C

Source inconsistency - average temperature example

The original text says data byte 0x3F means average temperature 33 deg C. Using the stated formula raw - 50, 0x3F decodes to 13 deg C. To represent 33 deg C, the byte would need to be 0x53. Treat the original 33 deg C value as a likely typo unless your device proves otherwise.

6. Message 0x07F4 - ALM_INFO - alarm information

The alarm frame is event triggered. When alarms are active, the BMS sends the alarm message periodically. If there are no alarms, this frame may not be sent.

Level	Meaning	Action priority
0	No alarm	Normal
1	Serious alarm	Highest priority
2	Important alarm	Medium priority
3	General alarm	Lowest non-zero priority

Alarm bit packing

Each alarm uses 2 bits. Decode with: level = (data[bit_start // 8] >> (bit_start % 8)) & 0x03.

Alarm #	Bit start	Byte / bits	Alarm name
1	0	data[0] bits 0-1	Unit overvoltage
2	2	data[0] bits 2-3	Unit undervoltage
3	4	data[0] bits 4-5	Total voltage overvoltage
4	6	data[0] bits 6-7	Total voltage undervoltage
5	8	data[1] bits 0-1	Large cell voltage difference
6	10	data[1] bits 2-3	Discharge overcurrent
7	12	data[1] bits 4-5	Charge overcurrent
8	14	data[1] bits 6-7	Temperature too high
9	16	data[2] bits 0-1	Temperature too low
10	18	data[2] bits 2-3	Excessive temperature difference
11	20	data[2] bits 4-5	SOC too low
12	22	data[2] bits 6-7	Insulation too low
13	24	data[3] bits 0-1	High voltage interlock fault
14	26	data[3] bits 2-3	External communication failure
15	28	data[3] bits 4-5	Internal communication failure

6.1 Alarm examples

Example A - mixed alarms

CAN ID	Data bytes	Decoded result
0x07F4	43 00 20 00 00 00 00 00	Alarm 1 Unit overvoltage = level 3 Alarm 4 Total voltage undervoltage = level 1 Alarm 11 SOC too low = level 2

Example B - low SOC alarm

CAN ID	Data bytes	Decoded result
0x07F4	00 00 30 00 00 00 00 00	Alarm 11 SOC too low = level 3

Example C - cell overvoltage and undervoltage

CAN ID	Data bytes	Decoded result
0x07F4	0F 00 00 00 00 00 00 00	Alarm 1 Unit overvoltage = level 3 Alarm 2 Unit undervoltage = level 3

Example D - temperature high and low

CAN ID	Data bytes	Decoded result
0x07F4	00 C0 03 00 00 00 00 00	Alarm 8 Temperature too high = level 3 Alarm 9 Temperature too low = level 3

7. Test cases and implementation checks

Use these cases to confirm your decoder before connecting to real hardware.

Case	Frame(s)	Expected result
Normal battery status	0x02F4: 13 01 D7 11 33 00 64 00	27.5 V; 56.7 A discharge; SOC 51%; discharge time 100 h. No alarm frame expected in normal state.
Low SOC	0x02F4: E1 00 8A 10 10 00 00 00 0x07F4: 00 00 30 00 00 00 00 00	22.5 V; 23.4 A discharge; SOC 16%; alarm 11 SOC too low level 3.
Cell voltage high/low	0x04F4: 8C 0A 05 00 92 09 08 00 0x07F4: 0F 00 00 00 00 00 00 00	Max cell 2700 mV at cell 5; min cell 2450 mV at cell 8; alarms 1 and 2 level 3.
Cell temperature high/low	0x05F4: 48 06 2F 01 3F 00 00 00 0x07F4: 00 C0 03 00 00 00 00 00	Max temp 22 deg C at cell 6; min temp -3 deg C at cell 1; average temp 13 deg C; alarms 8 and 9 level 3.

8. Developer checklist

- Configure the CAN interface as 250 kbit/s, CAN 2.0A, standard 11-bit identifiers.

- Listen for IDs 0x02F4, 0x04F4, 0x05F4, and 0x07F4.

- Always require DLC = 8 before decoding a frame.

- Decode U16 fields as little-endian.

- Apply the current offset: current_A = raw * 0.1 - 400.

- Ignore reserved bytes. Do not reject a valid frame because reserved bytes are non-zero unless your own test hardware requires that behavior.

- Do not assume 0x07F4 will always be present. It may only appear when an alarm exists.

- Confirm cell numbering and temperature examples against your specific BMS firmware; the original document contains inconsistencies in those areas.

Python-style reference decoder

```
        def u16le(d, i):
            return d[i] | (d[i + 1] << 8)

        def decode_batt_st(d):
            return {
                "voltage_V": u16le(d, 0) * 0.1,
                "current_A": u16le(d, 2) * 0.1 - 400,
                "soc_percent": d[4],
                "discharge_time_h": u16le(d, 6),
            }

        def decode_cell_volt(d):
            return {
                "max_cell_mV": u16le(d, 0),
                "max_cell_no": d[2],
                "min_cell_mV": u16le(d, 4),
                "min_cell_no": d[6],
            }

        def decode_cell_temp(d):
            return {
                "max_temp_degC": d[0] - 50,
                "max_temp_cell_no": d[1],
                "min_temp_degC": d[2] - 50,
                "min_temp_cell_no": d[3],
                "avg_temp_degC": d[4] - 50,
            }

        def decode_alarm_levels(d):
            names = [
                "Unit overvoltage", "Unit undervoltage",
                "Total voltage overvoltage", "Total voltage undervoltage",
                "Large cell voltage difference", "Discharge overcurrent",
                "Charge overcurrent", "Temperature too high",
                "Temperature too low", "Excessive temperature difference",
                "SOC too low", "Insulation too low",
                "High voltage interlock fault", "External communication failure",
                "Internal communication failure",
            ]
            out = []
            for n, name in enumerate(names, start=1):
                bit = (n - 1) * 2
                level = (d[bit // 8] >> (bit % 8)) & 0x03
                if level:
                    out.append({"alarm_number": n, "name": name, "level": level})
            return out
```

9. Known source issues corrected or flagged

The original PDF is useful, but several parts can confuse implementation. This cleaned version keeps the stated protocol while flagging the parts that need caution.

Area	Original issue	How this cleaned version handles it
One-way protocol	The source states communication is point-to-point and unidirectional from BMS to instrument, but the original title can make it sound like a general inverter protocol.	This document labels it as telemetry/display protocol and warns that it is not a full inverter control protocol.
Temperature example	The source example says byte 0x3F decodes to average temperature 33 deg C, but raw - 50 gives 13 deg C.	The worked example shows 13 deg C and notes that 33 deg C would require byte 0x53.
Cell-number offset	Field tables list offset 1 for cell-number fields, but examples use raw cell numbers directly.	The decoder uses raw cell number directly and adds a caution to verify with hardware.
CELL_VOLT test data	The later test-case table appears to reuse unrelated data bytes that do not match the stated max/min cell voltages.	The test case uses the corrected CELL_VOLT example bytes from the message definition.
Low-SOC wording	The source wording appears to say the low-power alarm is given at SOC >= 20%, while the example shows SOC 16%. The wording is likely incorrect or poorly translated.	This document describes the example as a low-SOC condition and avoids relying on the unclear threshold wording.

Final practical note

For real integration, verify these decoded values with live CAN captures from the exact Jikong/JK BMS model and firmware you are using. Treat this cleaned document as a clearer implementation reference, not a substitute for hardware validation.