1. CAN 通讯格式说明

说明：电池 BMS 与通讯设备之间的 CAN 通讯协议，实现数据交互功能。

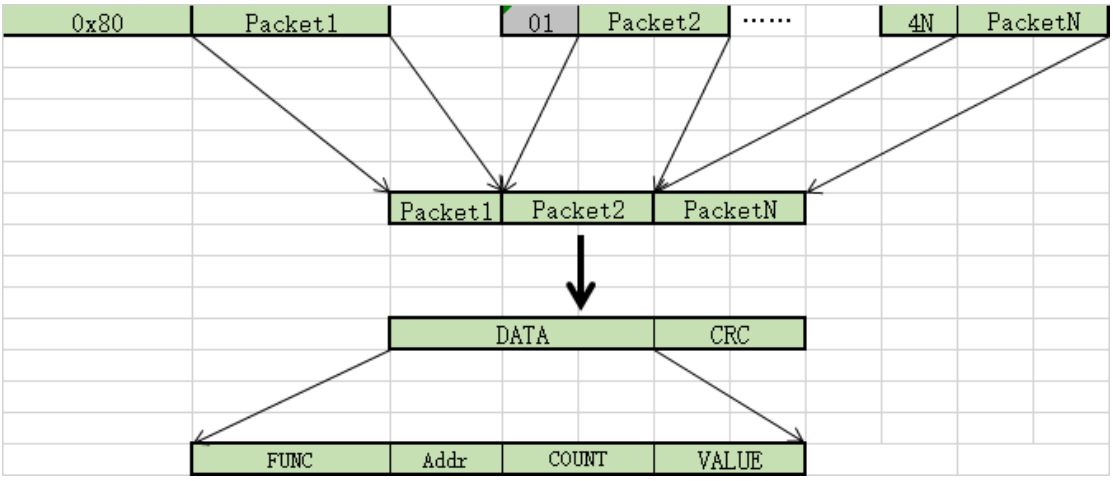
CAN BUS 标准	设备	CAN ID	说明
1、CAN 2.0B 标准； 2、标准帧（11bit）；	电池 BMS	0x00000080	电池 BMS 接收来自通讯设备的 CAN ID 为 0x0000052d 的数据帧
3、波特率固定 500kHz 4、大端模式	通讯设备	0x0000052d	通讯设备接收来自电池 BMS 的 CAN ID 为 0x00000080 的数据帧

发送通讯帧(Request)格式：	
标识符	数据
1 字节	1-7 字节
Bit7: 1->首帧 0->非首帧	待组合的数据包
Bit6: 1->尾帧 0->非尾帧	
Bit0-5：6 个位共同表示帧的索引，范围 0~63，表示第 1 帧到第 64 帧。	
说明：bit7 和 bit6 同时置 1 表示此帧既是首帧又是尾帧	

接收通讯帧(Request)格式：	
标识符	数据
1 字节	1-7 字节
Bit7: 预留 默认 0	待组合的数据包
Bit6: 预留 默认 0	
Bit0-5：6 个位共同表示帧的索引，范围 0~63，表示第 1 帧到第 64 帧。	

1.1 帧解析说明：

如下图所示，每一个帧第 1 字节用于标识当前帧类型以及索引，如 0x80 表示当前帧为第一帧；0x01 表示为处于中间的帧，并且为第二帧；0x4N 表示为尾帧，并且为第 N+1 帧。接收到完整的所有数据后，取出每个帧中的 packet 进行组合，并进行 CRC 校验。验证通过后，则可得到数据包。数据包由功能码 FUNC、数据类型 TYPE、操作数量 COUNT、以及数值 VALUE 组成。数据分解过程和组合过程相反。



1.2 例子/Sample

序号	系统时间	时间标识	传输方向	ID 号	帧类型	帧格式	长度	数据
1759	10:57:36.319	567.1475	接收	0x052D	数据帧	标准帧	0x08	x| 80 06 06 05 00 10 22 05
1760	10:57:36.319	567.1476	接收	0x052D	数据帧	标准帧	0x02	x| 41 68

解析得到合成的数据包(16 进制)：

06 06 05 00 10 22 05 68

序号	系统时间	时间标识	传输方向	ID 号	帧类型	帧格式	长度	数据
1761	10:57:36.319	567.1485	接收	0x0080	数据帧	标准帧	0x08	x| 00 06 4F 05 00 00 00 00
1762	10:57:36.319	567.1488	接收	0x0080	数据帧	标准帧	0x08	x| 01 00 00 69 78 00 00 69
1763	10:57:36.319	567.1491	接收	0x0080	数据帧	标准帧	0x08	x| 02 78 00 00 44 8D B2 A4
1764	10:57:36.319	567.1494	接收	0x0080	数据帧	标准帧	0x08	x| 03 00 41 00 00 00 00 00
1765	10:57:36.319	567.1497	接收	0x0080	数据帧	标准帧	0x08	x| 04 00 00 03 00 00 00 18
1766	10:57:36.319	567.1499	接收	0x0080	数据帧	标准帧	0x08	x| 05 00 19 00 19 00 19 00
1767	10:57:36.319	567.1502	接收	0x0080	数据帧	标准帧	0x08	x| 06 1A 00 19 00 00 0B EF
1768	10:57:36.319	567.1505	接收	0x0080	数据帧	标准帧	0x08	x| 07 0B DB 0B E3 0B E4 0B
1769	10:57:36.319	567.1507	接收	0x0080	数据帧	标准帧	0x08	x| 08 E6 0B EB 0B ED 0B E1
1770	10:57:36.319	567.151	接收	0x0080	数据帧	标准帧	0x08	x| 09 0B EA 0B E6 0B F0 0B
1771	10:57:36.319	567.1513	接收	0x0080	数据帧	标准帧	0x08	x| 0A F4 0B EB 0B F0 0B E5
1772	10:57:36.319	567.1516	接收	0x0080	数据帧	标准帧	0x08	x| 0B 00 00 51 90 00 00 00

解析得到合成的数据包(16 进制)：

06 4F 05 00 00 00 00 00 00 69 78 00 00 69 78 00 00 44 8D B2 A4 00 41 00 00 00 00 00 00 00 03 00 00 00 18 00 19 00 19 00 19 00 1A 00 19 00 00 0B EF 0B DB 0B E3 0B E4 0B E6 0B EB 0B ED 0B E1 0B EA 0B E6 0B F0 0B F4 0B EB 0B F0 0B E5 00 00 51 90

注：数据组合后按下列协议解析

2. 通讯协议：

2.1、协议概述

地址(Address)	长度(Length)	功能(Function)	数据(Data)	校验码(CRC)

8bits	8bits	8bits	n×8bits	16bits

数据包的发送序列总是相同的地址、长度、功能码、数据以及校验码，长度（Length）=数据包总长度-2，单位为字节，其中每个数据包需作为一个连续的位流传输。当主站数据包到达从站后，与数据包中地址域相匹配的从站将接收数据，从站对数据校验后，如果没有错误，就执行数据包中的请求，并将响应数据组包后发给主站，从站返回的响应数据包中包含有以下内容：从站地址（Address）、长度（Length）、执行的功能（Function）、功能执行生成的请求数据（Data）和校验码（CRC）。

- 地址域（Address）

地址域在数据包的开始部分，由一个八 bits 数据组成，这个数据表示主站指定的从站地址，总线上每个从站地址是唯一的，从站的有效地址范围在 0～247 之内。当主站发送数据包后，只有与主站查询地址相同的从站才会有响应。

- 功能域（Function）

功能域描述了从站所执行的何种功能，下表说明了所有功能码的意义。

代　码	定　　义	具体功能
05H	读数据	读取一个或多个变量的当前二进制值

- 数据域（Data）

数据域包含有从站执行特定功能所需要的数据或从站响应主站查询时采集到的数据。其中这些数据的内容可能是地址代码，或数据。

- 校验码域（CRC）

校验码是主站、从站在 CRC 校验传输数据时形成的 16bits 的校验数据。由于通信中存在各种干扰，因此通信中传输的数据可能会发生改变，CRC 校验能够有效保证主站、从站不会响应传输过程中发生了失真的数据，提高了系统的安全性和效率。校验码的形成规律见附录一中的说明。

2.2、应用层功能详解

(1) 读数据命令(功能码 05H)

- 读数据下行帧格式

设备通信地 06H	接收长度 单字节	功能代码 (05H)	数据域 起始地址 高字节	数据域 起始地址 低字节	数据域 长度字节	CRC 校验 低字节	CRC 校验 高字节

- 读数据应答帧格式

设备通信地址 06H	接收长度 单字节	功能代码 (05H)	数据内容	CRC 校验 低字节	CRC 校验 高字节

本设备地址为 06H

- 示例

读取总电压数据下行帧为：

设备地址 06H	数据包 长度-2	05H	00H	14H	01H	校验低	校验高

应答帧为：

设备地址 06H	数据包 长度-2	05H	总电压数据 高字节	总电压数据 低字节	校验低	校验高

发送：06 06 05 00 14 01 46 71

接收：06 05 05 5E D4 84 F2

总电压：5E D4 换成十进制 24276 单位：mV

读取 8 串电芯电压数据下行帧为：

设备地址 06H	数据包 长度-2	05H	22H	00H	08H	校验低	校验高

应答帧为：

设备地址 06H	数据包 长度-2	05H	电压 1 数据 高字节	电压 1 数据 低字节	电压 2 数据 高字节	电压 2 数据 低字节

…	…	…	电压 8 数据 高字节	电压 8 数据 低字节	校验低	校验高

发送：06 06 05 00 22 08 91 D7
接收：06 13 05 0B D5 0B DE 0B DB 0B D1 0B F0 0B E2 0B DB 0B DD 4E EB
电芯 1 电压：0B D5 转换成十进制 3029 单位：mV

以此类推

2.3、CRC 校验方法

冗余循环码（CRC）包含2个字节，即16位二进制。CRC码由发送设备计算，放置于发送信息的尾部。接收信息的设备再重新计算接收到信息的 CRC码，比较计算得到的CRC码是否与接收到的相符，如果两者不相符，则表明出错。

CRC码的计算方法是，先预置16位寄存器全为1。再逐步把每8位数据信息进行处理。在进行CRC码计算时只用8位数据位，起始位及停止位，如有奇偶校验位的话也包括奇偶校验位，都不参与CRC码计算。

在计算CRC码时，8位数据与寄存器的数据相异或，得到的结果向低位移一字节，用0填补最高位。再检查最低位，如果最低位为1，把寄存器的内容与预置数相异或，如果最低位为0，不进行异或运算。

这个过程一直重复8次。第8次移位后，下一个8位再与现在寄存器的内容相相异或，这个过程与以上一样重复8次。当所有的数据信息处理完后，最后寄存器的内容即为CRC码值。CRC码中的数据发送、接收时低字节在前。

计算CRC码的步骤为：

- 预置16位寄存器为十六进制FFFF（即全为1），称此寄存器为CRC寄存器。
- 把第一个8位数据与16位CRC寄存器的低位相异或，把结果放于CRC寄存器。
- 把寄存器的内容右移一位(朝低位)，用0填补最高位，检查最低位。
- 如果最低位为0：重复第3步(再次移位)； 如果最低位为1：CRC寄存器与多项式A001（1010 0000 0000 0001）进行异或。
- 重复步骤3和4，直到右移8次，这样整个8位数据全部进行了处理。
- 重复步骤2到步骤5，进行下一个8位数据的处理。
- 最后得到的CRC寄存器即为CRC码。

采用的 CRC 参数模型	CRC-16/MODBUS
多项式公式	x16+x15+x2+1
宽度	16
多项式	0x8005
初始值	0xFFFF
结果异或值	0x0000
输入值反转	true
输出值反转	true

CRC 算法参考：

```
static const uint8_t chCRCHTalbe[] =
{
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
```

```
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40
};
static const uint8_t chCRCLTalbe[] =
{
    0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7,
    0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E,
    0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09, 0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9,
    0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC,
    0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
    0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32,
    0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D,
    0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38,
    0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF,
    0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
    0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1,
    0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4,
    0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB,
    0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA,
    0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
    0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0,
    0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97,
    0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C, 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E,
    0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89,
    0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
    0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83,
    0x41, 0x81, 0x80, 0x40
};
uint16_t calculate_crc16(uint8_t *crc_buf, const uint8_t crc_count)
{
    uint16_t wIndex;
    uint16_t chCRCHi = 0xFF;
```

```
    uint16_t chCRCLo = 0xFF;
    uint16_t i;
    for(i=0;i<crc_count;i++)
    {
        wIndex = chCRCLo ^  crc_buf[i];
        chCRCLo = chCRCHi ^ chCRCHTalbe[wIndex];
        chCRCHi = chCRCLTalbe[wIndex] ;
    }
    return ((chCRCHi << 8) | chCRCLo) ;
}
```

3. BMS 数据列表

参数名称	地址	数据类型	长度	读写	数据范围	单位
电流	0010H	signed long int	4 字节	只读	-2147483647~2147483647	mA
满充容量	0011H	unsigned long int	4 字节	只读	0~4294967295	mAH
满放容量	0012H	unsigned long int	4 字节	只读	0~4294967295	mAH
剩余容量	0013H	unsigned long int	4 字节	只读	0~4294967295	mAH
总电压	0014H	unsigned int	2 字节	只读	0~65535	mV
RSOC	0015H	unsigned int	2 字节	只读	0~65535	1%
循环次数	0016H	unsigned int	2 字节	只读	0~65535	次
保护状态字节	0017H	unsigned int	2 字节	只读	0~65535	BIT
告警状态字节	0018H	unsigned int	2 字节	只读	0~65535	BIT
BMS 状态字节	0019H	unsigned int	2 字节	只读	0~65535	BIT
均衡状态字节	001AH	unsigned int	2 字节	只读	0~65535	BIT
电芯温度 1	001BH	signed int	2 字节	只读	-32767~32767	℃
电芯温度 2	001CH	signed int	2 字节	只读	-32767~32767	℃
电芯温度 3	001DH	signed int	2 字节	只读	-32767~32767	℃
电芯温度 4	001EH	signed int	2 字节	只读	-32767~32767	℃
环境温度	001FH	signed int	2 字节	只读	-32767~32767	℃
功率板 1 温度	0020H	signed int	2 字节	只读	-32767~32767	℃
功率板 2 温度	0021H	signed int	2 字节	只读	-32767~32767	℃
Cell1 电压	0022H	unsigned int	2 字节	只读	0~65535	mV

Cell2 电压	0023H	unsigned int	2 字节	只读	0~65535	mV
Cell3 电压	0024H	unsigned int	2 字节	只读	0~65535	mV
Cell4 电压	0025H	unsigned int	2 字节	只读	0~65535	mV
Cell5 电压	0026H	unsigned int	2 字节	只读	0~65535	mV
Cell6 电压	0027H	unsigned int	2 字节	只读	0~65535	mV
Cell7 电压	0028H	unsigned int	2 字节	只读	0~65535	mV
Cell8 电压	0029H	unsigned int	2 字节	只读	0~65535	mV
Cell9 电压	002AH	unsigned int	2 字节	只读	0~65535	mV
Cell10 电压	002BH	unsigned int	2 字节	只读	0~65535	mV
Cell11 电压	002CH	unsigned int	2 字节	只读	0~65535	mV
Cell12 电压	002DH	unsigned int	2 字节	只读	0~65535	mV
Cell13 电压	002EH	unsigned int	2 字节	只读	0~65535	mV
Cell14 电压	002FH	unsigned int	2 字节	只读	0~65535	mV
Cell15 电压	0030H	unsigned int	2 字节	只读	0~65535	mV
Cell16 电压	0031H	unsigned int	2 字节	只读	0~65535	mV

4. 保护状态字节

数据定义：0 -> 正常 1 -> 保护

位地址	定义
Bit0	电芯低压
Bit1	电芯高压
Bit2	总压低压
Bit3	总压高压
Bit4	一级放电过流
Bit5	二级放电过流
Bit6	短路
Bit7	一级充电过流
Bit8	二级充电过流
Bit9	放电高温
Bit10	放电低温
Bit11	充电高温
Bit12	充电低温
Bit13	MOS 高温
Bit14	无
Bit15	无

5. 告警状态字节

数据定义：0 -> 正常 1 -> 告警

位地址	定义

Bit0	电芯低压
Bit1	电压高压
Bit2	总压低压
Bit3	总压高压
Bit4	放电过流
Bit5	充电过流
Bit6	电流采集
Bit7	放电高温
Bit8	放电低温
Bit9	充电高温
Bit10	充电低温
Bit11	MOS 高温
Bit12	低容量
Bit13	AFE 采集
Bit14	无
Bit15	

6. BMS 状态字节

位地址	定义	数据解析
Bit0	放电管	0：关闭 1：开启
Bit1	充电管	0：关闭 1：开启
Bit2	无	
Bit3	无	
Bit4	放电状态	0：无 1：放电
Bit5	充电状态	0：无 1：充电
Bit6	无	
Bit7	无	
Bit8	无	
Bit9	无	
Bit10	无	
Bit11	无	
Bit12	无	
Bit13	无	
Bit14	无	
Bit15	无	

7. 均衡状态字节

数据定义：0 -> 关闭 1 -> 开启

位地址	定义
Bit0	第 1 串均衡

Bit1	第 2 串均衡
Bit2	第 3 串均衡
Bit3	第 4 串均衡
Bit4	第 5 串均衡
Bit5	第 6 串均衡
Bit6	第 7 串均衡
Bit7	第 8 串均衡
Bit8	第 9 串均衡
Bit9	第 10 串均衡
Bit10	第 11 串均衡
Bit11	第 12 串均衡
Bit12	第 13 串均衡
Bit13	第 14 串均衡
Bit14	第 15 串均衡
Bit15	第 16 串均衡