| | | | | |
|---|---|---|---|---|
| 文档标题 | **FLIPSKY FTESC CAN通信协议 V1.4** | 文件版本 | V1.4 | |
| | | **FT**固件版本 | V1.4 | |
| | | 发布日期 | 2024/11/04 | |
| 文件编号 | FT-CM-CAN-20241104-04 | | | |
| 产品型号 | 翻天科技**FT**系列电调 | 共 **20** 页 | 级别 | 公开 |

# FLIPSKY FTESC CAN

# 通信协议 V1.4

广东翻天科技有限公司

修订控制

| 版本 | 修订日期 | 修订人 | 修订提要 |
|---|---|---|---|
| 1.4 | 2024-11-04 | 陆 | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# 目录

# 1 概述

本文描述了电机控制单元（Motor Control Unit，MCU）与电子控制单元（Electronic Control Unit，ECU）之间的CAN通信协议。在此文档中，MCU 特指翻天科技公司研发的FT系列电调，而ECU特指由用户开发的CAN通信设备。本文旨在协助客户如何使用CAN通信端口访问以及控制FT系列电调。

# 2 报文格式说明

FT系列电调均带有CAN端口，故用户在自行开发ECU时可使用CAN端口与MCU进行通信。

## 2.1 通信参数

波特率： 250K、500K、750K、1000K
IDE： 扩展型ID
EID： 29 bit，目前使用bit0~bit7用于CAN ID指令，bit8~bit15用于电调ID
RTR：数据帧
DLC：根据实际数据长度决定
具体帧格式如下：

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | CRC | ACK | EOF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | 8 bytes | 15 bits | 2 bit | 7bits |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CAN COMMAND ID | 0 | 0 | length | data[0-8] | | | |

## 2.2 实现注解

在报文、应答的帧协议中，

(1) 涉及到的 **uint16_t** 型数据、**int16_t** 型数据、**uint32_t** 型数据以及 **int32_t** 型数据均按照大端模式进行传输，即高位字节数据在前（低地址），低位字节数据在后（高地址）。
(2) 无特殊标记的单字节数据均为 **uint8_t** 型数据。

## 2.3 CRC 校验码

```
static const unsigned char aucCRCHi[] = {
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 unsigned char aucCRCLo[] = {
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 crc16(uint8_t *Buffer, uint32_t Length)
{
  uint8_t ucCRCHi = 0xFF;
  uint8_t ucCRCLo = 0xFF;
  int i = 0;
  while( Length-- )
  {
    i = ucCRCLo ^ *( Buffer++ );
    ucCRCLo = ( uint8_t )( ucCRCHi ^ aucCRCHi[i] );
    ucCRCHi = aucCRCLo[i];
  }
  return (uint16_t)((((uint16_t)ucCRCHi) << 8) | ucCRCLo);
}
```

## 2.4 CAN COMMAND 指令码

```
typedef enum {
    CAN_SET_CURRENT_GEAR_CRC16              = 0, // 电流控制指令（挡位、CRC16校验）
    CAN_SET_CURRENT_BRAKE_CRC16             = 1, // 刹车电流控制指令（CRC16校验的）
    CAN_SET_DUTY_GEAR_CRC16                 = 2, // 占空比控制指令（挡位、CRC16校验）
    CAN_SET_CURRENT_PERCENT_CRC16           = 3, // 电流百分比控制指令（CRC16校验）
    CAN_SET_CURRENT_PERCENT_GEAR_CRC16      = 4, // 电流百分比控制指令（挡位、CRC16校验）
    CAN_SET_CURRENT_BRAKE_PERCENT_CRC16     = 5, // 刹车电流百分比控制指令（CRC16校验）
    ...
    CAN_ESC_REALTIME_DATA_0_CRC16           = 11,// 返回电机电流、电池电流（每隔20ms自动回传一次）
    CAN_ESC_REALTIME_DATA_1_CRC16           = 12,// 返回电机电子转速、占空比
    CAN_ESC_REALTIME_DATA_2_CRC16           = 13,// 返回MOS管温度、电机温度、电池电压
    ...
    CAN_SET_ERPM_GEAR_CRC16                 = 18,// 带有挡位、CRC16校验的转速控制指令
    CAN_SET_POSITION_GEAR_CRC16             = 19,// 带有挡位、CRC16校验的位置控制指令
    CAN_SET_ID_CURRENT_CRC16                = 20,// 电机D轴电流控制指令

    // 以下指令不带CRC16校验
    CAN_SET_CURRENT_GEAR                    = 21,// 电流控制指令（挡位）
    CAN_SET_CURRENT_PERCENT_GEAR            = 22,// 电流百分比控制指令（挡位）
    CAN_SET_DUTY_GEAR                       = 23,// 占空比控制指令（挡位）
    CAN_SET_ERPM_GEAR                       = 24,// 转速控制指令（挡位）
    CAN_SET_POSITION_GEAR                   = 25,// 位置控制指令（挡位）
    CAN_SET_CURRENT_BRAKE                   = 26,// 刹车电流控制指令
    CAN_SET_CURRENT_BRAKE_PERCENT           = 27,// 刹车电流百分比控制指令
} CAN_COMMAND;
```

# 3 指令/应答 消息详解

## 3.1 指令解析

**注意：当使用CAN通信控制FT电调时，请先通过上位机将Input Signal Type参数设置为Off再进行通信控制，并确保ECU与MCU的CAN波特率参数设置一致。**

### 3.1.1 CAN_SET_CURRENT_GEAR_CRC16 (00H)

1）帧格式以及功能描述

发送一次电机电流控制指令，并带有挡位设置以及CRC16校验。

注意：请至少保持每隔450ms发送一次，否则超时后指令失效。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 00H | 0 | 0 | 7 | 0 | crc16_high | crc16_low | gear | motor current*100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Motor current*100000 | 实际需要驱动的电机电流值(单位:A)放大100000倍。如：需要控制的电流值为50A，则实际发送时需要放大100000倍，即5000000。 |
| D4 | Gear | 00H: 无挡位<br>01H: 低速挡<br>02H: 中速挡<br>03H: 高速挡<br>04H: 倒车挡<br>注意：如不需要挡位限速控制，请将D4置00H； |
| D5 D6 | CRC16校验码 | 具体计算方法参考3.2.1节 |

### 3.1.2 CAN_SET_CURRENT_BRAKE_CRC16 (01H)

1）帧格式以及功能描述

发送一次刹车电流控制指令，并带有CRC16校验。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 01H | 0 | 0 | 6 | 0 | 0 | crc16_high | crc16_low | Brake*100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Brake*100000 | 实际需要驱动的刹车电流值(单位:A)放大100000倍。 |
| D4 D5 | CRC16校验码 | |

### 3.1.3 CAN_SET_DUTY_GEAR_CRC16 (02H)

1）帧格式以及功能描述

发送一次占空比控制指令，并带有挡位设置以及CRC16校验。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 02H | 0 | 0 | 7 | 0 | crc16_high | crc16_low | gear | Duty*1000000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Duty*1000000 | 实际需要驱动的占空比值。如：占空比为0.2，则实际发送的值为0.2*1000000=200000 |
| D4 | Gear | 00H: 无挡位<br>01H: 低速挡<br>02H: 中速挡<br>03H: 高速挡<br>04H: 倒车挡<br>注意：如不需要挡位限速控制，请将D4置00H； |
| D5 D6 | CRC16校验码 | |

### 3.1.4 CAN_SET_CURRENT_PERCENT_CRC16 (03H)

1）帧格式以及功能描述

发送一次电流百分比控制指令，并带有CRC16校验。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 03H | 0 | 0 | 6 | 0 | 0 | crc16_high | crc16_low | Current Percent *100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Current Percent *100000 | 实际需要驱动电流百分比值。如：该值为0.2，则实际发送的值为0.2*100000=20000，此时电机实际运行电流为0.2*Motor Current。<br>注意：Motor Current参数值大小可通过上位机进行 |

| | | |
|---|---|---|
| | | 设置。 |
| D4 D5 | CRC16校验码 | |

### 3.1.5 CAN_SET_CURRENT_PERCENT_GEAR_CRC16 (04H)

1）帧格式以及功能描述

发送一次电流百分比控制指令，并带有挡位设置以及CRC16校验。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 04H | 0 | 0 | 7 | 0 | crc16_high | crc16_low | gear | Current Percent *100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Current Percent*100000 | 实际需要驱动电流百分比值。如：该值为0.2，则实际发送的值为0.2*100000=20000，此时电机实际运行电流为0.2*Motor Current。<br>注意：Motor Current参数值大小可通过上位机进行设置。 |
| D4 | Gear | 00H: 无挡位<br>01H: 低速挡<br>02H: 中速挡<br>03H: 高速挡<br>04H: 倒车挡<br>注意：如不需要挡位限速控制，请将D4置00H； |
| D5 D6 | CRC16校验码 | |

### 3.1.6 CAN_SET_CURRENT_BRAKE_PERCENT_CRC16 (05H)

1）帧格式以及功能描述

发送一次刹车电流百分比控制指令，并带有CRC16校验。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 05H | 0 | 0 | 6 | 0 | 0 | crc16_high | crc16_low | Brake Current Percent *100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Brake Current Percent*100000 | 实际需要驱动刹车电流百分比值。如：该值为0.2，则实际发送的值为0.2*100000=20000，此时电机实际运行刹车电流为0.2*Motor Current Brake。<br>注意：Motor Current Brake参数值大小可通过上位 |

| | | |
|---|---|---|
| | | 机进行设置。 |
| D4 D5 | CRC16校验码 | |

### 3.1.7 CAN_ESC_REALTIME_DATA_0_CRC16 (0BH)

1）帧格式以及功能描述

MCU主动发出一次实时数据0信息帧，返回的数据包含有电机电流、电池电流，并带有CRC16校验码。

注意：

① 该指令由MCU每隔10ms自动发送到CAN总线上，用户可以通过解析该指令实时获取数据0信息；

② 用户可通过MCU ID参数识别出具体是哪一个电调的实时数据；

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 0BH | 0 | 0 | 8 | crc16_high | crc16_low | battery current*1000 | | | motor current*1000 | | |
| | | | | | | | | | | | | | Int32_t | | | Int32_t | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 | Motor Current Percent*1000 | 读取到的为电机电流放大1000倍值，除以1000后才是实际电机电流值。 |
| D3 D4 D5 | Battery Current Percent*1000 | 读取到的为电池电流放大1000倍值，除以1000后才是实际电池电流值。 |
| D6 D7 | CRC16校验码 | |

### 3.1.8 CAN_ESC_REALTIME_DATA_1_CRC16 (0CH)

1）帧格式以及功能描述

MCU主动发出一次实时数据1信息帧，返回数据包含电机电子转速、占空比，并带有CRC16校验。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 0CH | 0 | 0 | 8 | crc16_high | crc16_low | duty*1000 | | | motor erpm | | |
| | | | | | | | | | | | | | Int32_t | | | Int32_t | | |

注意：

① 该指令由MCU每隔10ms自动发送到CAN总线上，用户可以通过解析该指令实时获取数据1信息；

② 用户可通过MCU ID参数识别出具体是哪一个电调的实时数据；

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 | Motor erpm | 电机电子转速值，即电机机械转速*电机磁极对数。 |

| | | |
|---|---|---|
| D3 D4 D5 | duty*1000 | 占空比放大1000倍后的数值，该值除以1000后即为实际的占空比值。 |
| D6 D7 | CRC16校验码 | |

### 3.1.9 CAN_ESC_REALTIME_DATA_2_CRC16 (0DH)

1）帧格式以及功能描述

MCU主动发出一次实时数据2信息帧，返回数据包含MOS管温度、电机温度以及电池电压，并带有CRC16校验。

注意：

① 该指令由MCU每隔10ms自动发送到CAN总线上，用户可以通过解析该指令实时获取数据2信息；

② 用户可通过MCU ID参数识别出具体是哪一个电调的实时数据；

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 0DH | 0 | 0 | 8 | crc16_high | crc16_low | Battery voltage*100 | | motor temp*100 | | mosfet temp*100 | |
| | | | | | | | | | | | | | Int16_t | | Int16_t | | Int16_t | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 | MOSFET temp*100 | MOSFET温度，放大100倍。 |
| D2 D3 | Motor temp*100 | 电机温度，放大100倍。 |
| D4 D5 | Battery voltage*100 | 电池电压，放大100倍。 |
| D6 D7 | CRC16校验码 | |

### 3.1.10 CAN_SET_ERPM_GEAR_CRC16 (12H)

1）帧格式以及功能描述

发送一次转速控制指令，并带有挡位设置以及CRC16校验。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 12H | 0 | 0 | 7 | 0 | crc16_high | crc16_low | gear | motor erpm | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Motor erpm | 电机电子转速值，即电机转速*电机磁极对数。 |
| D4 | Gear | 00H: 无挡位<br>01H: 低速挡<br>02H: 中速挡<br>03H: 高速挡<br>04H: 倒车挡<br>注意：如不需要挡位限速控制，请将D4置00H； |

| | | |
|---|---|---|
| D5 D6 | CRC16校验码 | |

### 3.1.11 CAN_SET_POSITION_GEAR_CRC16 (13H)

1）帧格式以及功能描述

发送一次位置控制指令，并带有挡位设置以及CRC16校验。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 13H | 0 | 0 | 7 | 0 | crc16_high | crc16_low | gear | Position*100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Position*100000 | 角度位置控制，-180°~180°，放大100000倍。比如，需要控制角度为45.6°，则实际发送指令的值为45.6*100000=4560000。 |
| D4 | Gear | 00H: 无挡位<br>01H: 低速挡<br>02H: 中速挡<br>03H: 高速挡<br>04H: 倒车挡<br>注意：如不需要挡位限速控制，请将D4置00H； |
| D5 D6 | CRC16校验码 | |

### 3.1.12 CAN_SET_ID_CURRENT_CRC16 (14H)

1）帧格式以及功能描述

向电机D轴发送一次注入电流控制指令，并带有挡位设置以及CRC16校验。该指令可用于在无感模式或者霍尔模式下，向电机D轴注入电流，可让电机实现驻车功能，但是会消耗电流，电机会发烫，不建议长时间使用。

注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26 -28 | bits 18 -25 | 1 bit | 1 bit | bits 16 -17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 14H | 0 | 0 | 6 | 0 | 0 | crc16_high | crc16_low | current*100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Current*100000 | 需要注入的电机D轴的电流值，放大100000倍。 |
| D4 D5 | CRC16校验码 | |

### 3.1.13 CAN_SET_CURRENT_GEAR (15H)

1）帧格式以及功能描述

发送一次电机电流控制指令，并带有挡位设置。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 15H | 0 | 0 | 5 | 0 | 0 | 0 | gear | Current*100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Current*100000 | 实际需要驱动的电机电流值(单位:A)放大100000倍。如：需要控制的电流值为50A，则实际发送时需要放大100000倍，即5000000。 |
| D4 | Gear | 00H: 无挡位<br>01H: 低速挡<br>02H: 中速挡<br>03H: 高速挡<br>04H: 倒车挡<br>注意：如不需要挡位限速控制，请将D4置00H； |

### 3.1.14 CAN_SET_CURRENT_PERCENT_GEAR (16H)

1）帧格式以及功能描述

发送一次电流百分比控制指令，并带有挡位设置。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 16H | 0 | 0 | 7 | 0 | crc16_high | crc16_low | gear | Current Percent *100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Current Percent*100000 | 实际需要驱动电机电流百分比值。如：该值为0.2，则实际发送的值为0.2*100000=20000，此时电机实际运行电流为0.2*Motor Current。<br>注意：Motor Current参数值大小可通过上位机进行设置。 |
| D4 | Gear | 00H: 无挡位<br>01H: 低速挡<br>02H: 中速挡<br>03H: 高速挡<br>04H: 倒车挡<br>注意：如不需要挡位限速控制，请将D4置00H； |

### 3.1.15 CAN_SET_DUTY_GEAR (17H)

1）帧格式以及功能描述

发送一次占空比控制指令，并带有挡位设置。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 17H | 0 | 0 | 5 | 0 | 0 | 0 | gear | Duty*1000000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Duty*1000000 | 实际需要驱动的占空比值。如：占空比为0.2，则实际发送的值为0.2*1000000=200000 |
| D4 D5 | CRC16校验码 | |

### 3.1.16 CAN_SET_ERPM_GEAR (18H)

1）帧格式以及功能描述

发送一次转速控制指令，并带有挡位设置。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 18H | 0 | 0 | 5 | 0 | 0 | 0 | gear | motor erpm | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Motor erpm | 电机电子转速值，即电机转速*电机磁极对数。 |
| D4 | Gear | 00H: 无挡位<br>01H: 低速挡<br>02H: 中速挡<br>03H: 高速挡<br>04H: 倒车挡<br>注意：如不需要挡位限速控制，请将D4置00H； |

### 3.1.17 CAN_SET_POSITION_GEAR (19H)

1）帧格式以及功能描述

发送一次位置控制指令，并带有挡位设置。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 19H | 0 | 0 | 5 | 0 | 0 | 0 | gear | Position*100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Position*100000 | 角度位置控制，-180°~180°，放大100000倍。 |

| | | |
|---|---|---|
| | | 比如，需要控制角度为45.6°，则实际发送指令的值为45.6*100000=4560000。 |
| D4 | Gear | 00H: 无挡位<br>01H: 低速挡<br>02H: 中速挡<br>03H: 高速挡<br>04H: 倒车挡<br>注意：如不需要挡位限速控制，请将D4置00H； |

### 3.1.18 CAN_SET_CURRENT_BRAKE (20H)

1）帧格式以及功能描述

发送一次刹车电流控制指令。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 20H | 0 | 0 | 4 | 0 | 0 | 0 | 0 | Brake*100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Brake*100000 | 实际需要驱动的刹车电流值(单位:A)放大100000倍。 |

### 3.1.19 CAN_SET_CURRENT_BRAKE_PERCENT (21H)

1）帧格式以及功能描述

发送一次刹车电流百分比控制指令。注意：同3.1.1。

| SOF | Top 11 Bits of ID | | SRR | IDE | Bottom 18 Bits of ID | | | RTR | Reserved | DLC | Data Field | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 bit | bits 26-28 | bits 18-25 | 1 bit | 1 bit | bits 16-17 | bits 8-15 | bits 0-7 | 1 bit | 2 bits | 4bits | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| 1 | 0 | 0 | 0 | 1 | 0 | MCU ID | CMD: 21H | 0 | 0 | 4 | 0 | 0 | 0 | 0 | Brake Current Percent*100000 | | | |
| | | | | | | | | | | | | | | | Int32_t | | | |

2）Data Field 参数解释

| 项目 | 值 | 说明 |
|---|---|---|
| D0 D1 D2 D3 | Brake Current Percent*100000 | 实际需要驱动刹车电流百分比值。如：该值为0.2，则实际发送的值为0.2*100000=20000，此时电机实际运行刹车电流为0.2*Motor Current Brake。<br>注意：Motor Current Brake参数值大小可通过上位机进行设置。 |

## 3.2 指令/应答 操作举例（C语言）

### 3.2.1 ECU发送控制指令

```
typedef enum {
     REMOTE_SPEED_NON = 0,
     REMOTE_SPEED_LOW,
     REMOTE_SPEED_MIDDLE,
     REMOTE_SPEED_HIGH,
     SMART_REVERSE
} Motor_Gear;

// 占空比控制指令
// ftescId: 被控制的电调ID
// duty: 占空比 [-1.0, 1.0]
// gear: 速度挡位，若不需要，设置为REMOTE_SPEED_NON
void canControlDutyGear(uint8_t ftescId, float duty, Motor_Gear gear) {
     int32_t buff_index_t = 0;
     uint8_t can_signalframe_buff[8];
     valueDecompose_I32(can_signalframe_buff, (int32_t)(duty * 1000000.0f), &buff_index_t);
     can_signalframe_buff[buff_index_t++] = gear;
     unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
     canPackTransmitLowHardware(ftescId, CAN_SET_DUTY_GEAR_CRC16, can_signalframe_buff,
                                buff_index_t);
}

// 转速控制指令
// ftescId: 被控制的电调ID
// erpm: 电子转速
// gear: 速度挡位，若不需要，设置为REMOTE_SPEED_NON
void canControlErpmGear(uint8_t ftescId, float erpm, Motor_Gear gear) {
     int32_t buff_index_t = 0;
     uint8_t can_signalframe_buff[8];
     valueDecompose_I32(can_signalframe_buff, (int32_t)(erpm), &buff_index_t);
     can_signalframe_buff[buff_index_t++] = gear;
     unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
     can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
     canPackTransmitLowHardware(ftescId, CAN_SET_ERPM_GEAR_CRC16, can_signalframe_buff,
                                buff_index_t);
}

// 位置控制指令
// ftescId: 被控制的电调ID
// pos: 位置, [-180°, 180°]
// gear: 速度挡位，若不需要，设置为REMOTE_SPEED_NON
void canControlPositionGear(uint8_t ftescId, float pos, Motor_Gear gear) {
```

```
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_I32(can_signalframe_buff, (int32_t)(pos * 100000.0f), &buff_index_t);
    can_signalframe_buff[buff_index_t++] = gear;
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_POSITION_GEAR_CRC16, can_signalframe_buff,
                               buff_index_t);
}

// 电流控制指令
// ftescId: 被控制的电调ID
// current: 电机电流，单位：A
// gear: 速度挡位，若不需要，设置为REMOTE_SPEED_NON
void canControlCurrentGear(uint8_t ftescId, float current, Motor_Gear gear) {
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_I32(can_signalframe_buff, (int32_t)(current * 100000.0f), &buff_index_t);
    can_signalframe_buff[buff_index_t++] = gear;
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_GEAR_CRC16, can_signalframe_buff,
                               buff_index_t);
}

// 电机D轴电流控制指令
// ftescId: 被控制的电调ID
// current: 电机D轴电流，单位：A
void canControlIdCurrent(uint8_t ftescId, float current) {
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_I32(can_signalframe_buff, (int32_t)(current * 100000.0f), &buff_index_t);
    can_signalframe_buff[buff_index_t++] = 0;
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId,     CAN_SET_ID_CURRENT_CRC16,     can_signalframe_buff,
buff_index_t);
}

// 电机刹车电流控制指令
// ftescId: 被控制的电调ID
// current: 电机刹车电流，单位：A
void canControlCurrentBrake(uint8_t ftescId, float current) {
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_I32(can_signalframe_buff, (int32_t)(current * 100000.0f), &buff_index_t);
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
```

```
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_BRAKE_CRC16, can_signalframe_buff,
                               buff_index_t);
}

// 电机电流百分比控制指令
// ftescId: 被控制的电调ID
// current_percent: 电机电流百分比，[-1.0, 1.0]
void comm_can_set_current_rel(uint8_t ftescId, float current_percent) {
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_F32(can_signalframe_buff, current_percent, 1e5, &buff_index_t);
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_PERCENT_CRC16, can_signalframe_buff,
                               buff_index_t);
}

// 电机电流百分比控制指令，带挡位
// ftescId: 被控制的电调ID
// current_percent: 电机电流百分比，[-1.0, 1.0]
// gear: 速度挡位，若不需要，设置为REMOTE_SPEED_NON
void canControlCurrentPercentGear(uint8_t ftescId, float current_percent, Motor_Gear gear) {
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_F32(can_signalframe_buff, current_percent, 1e5, &buff_index_t);
    can_signalframe_buff[buff_index_t++] = gear;
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_PERCENT_GEAR_CRC16, can_signalframe_buff,
                               buff_index_t);
}

// 电机刹车电流百分比控制指令
// ftescId: 被控制的电调ID
// current_percent: 电机刹车电流百分比，[-1.0, 1.0]
void canControlCurrentBrakePercent(uint8_t ftescId, float current_percent) {
    int32_t buff_index_t = 0;
    uint8_t can_signalframe_buff[8];
    valueDecompose_F32(can_signalframe_buff, current_percent, 1e5, &buff_index_t);
    unsigned short crc = crc16(can_signalframe_buff, buff_index_t);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc >> 8);
    can_signalframe_buff[buff_index_t++] = (uint8_t)(crc & 0xFF);
    canPackTransmitLowHardware(ftescId, CAN_SET_CURRENT_BRAKE_PERCENT_CRC16, can_signalframe_buff,
                               buff_index_t);
}
```

### 3.2.2 CAN实时数据帧处理(0BH、0CH、0DH)

```
#define ESC_NUM_MAX 4

typedef struct {
     uint8_t esc_id;
     float motorCurrent;
     float BatteryCurrent;
} can_esc_data_0;

typedef struct {
     uint8_t esc_id;
     float erpm;
     float duty;
} can_esc_data_1;

typedef struct {
     uint8_t esc_id;
     float mosfetTemp;
     float motorTemp;
     float battVoltage;
} can_esc_data_2;

can_esc_data_0 esc_data_0[ESC_NUM_MAX];
can_esc_data_1 esc_data_1[ESC_NUM_MAX];
can_esc_data_2 esc_data_2[ESC_NUM_MAX];


// extern_id:扩展帧ID
// buff:CAN 数据包
// length:数据包长度
static void canCommIDHandle(uint32_t extern_id, uint8_t *buff, int length) {
     int32_t buff_index_t = 0;
     uint8_t crc16_low = 0;
     uint8_t crc16_high = 0;

     uint8_t esc_id = (extern_id >> 8);//获取电调ID
     CAN_COMMAND can_comm_id = (CAN_COMMAND)(extern_id & 0xFF);//获取电调指令

     switch (can_comm_id) {
     case CAN_ESC_REALTIME_DATA_0_CRC16:
     {
          crc16_high = buff[6];
          crc16_low = buff[7];
          if (crc16(buff, 6) == ((unsigned short) crc16_high << 8 | (unsigned short) crc16_low))
          {
               for (int i = 0; i < ESC_NUMBER_MAX; i++) {
                    can_esc_data_0 *esc_data_status_0 = &esc_data_0[i];
                    //“-2”说明有新的电调发送数据过来，第一次更新后，便会保存该ID值。
                    if (esc_data_status_0->esc_id == esc_id ||
```

```
                    esc_data_status_0->esc_id == -2)
                {                       buff_index_t = 0;
                                        esc_data_status_0->esc_id = esc_id;
                    esc_data_status_0->motorCurrent =
                                    (float)valueCompose_I24(buff, &buff_index_t)*0.001f;
                    esc_data_status_0->BatteryCurrent =
                                   (float)valueCompose_I24(buff, &buff_index_t)*0.001f;
                    break;
                }
            }
        }
    }
    break;

    case CAN_ESC_REALTIME_DATA_1_CRC16:
    {
        crc16_high = buff[6];
        crc16_low = buff[7];
        if (crc16(buff, 6) == ((unsigned short) crc16_high << 8 | (unsigned short) crc16_low))
        {
            for (int i = 0; i < ESC_NUMBER_MAX; i++) {
                can_esc_data_1 *esc_data_status_1 = &esc_data_1[i];
                if (esc_data_status_1->esc_id == esc_id ||
                    esc_data_status_1->esc_id == -2) {
                    buff_index_t = 0;
                    esc_data_status_1->esc_id = esc_id;
                    esc_data_status_1->erpm =
                      (float)valueCompose_I24(buff, &buff_index_t);
                    esc_data_status_1->duty =
                      (float)valueCompose_I24(buff, &buff_index_t)*0.001f;
                    break;
                }
            }
        }
    }
    break;

    case CAN_ESC_REALTIME_DATA_2_CRC16:
    {
        crc16_high = buff[6];
        crc16_low = buff[7];
        if (crc16(buff, 6) == ((unsigned short) crc16_high << 8 | (unsigned short) crc16_low))
        {
            for (int i = 0; i < ESC_NUMBER_MAX; i++) {
                can_esc_data_2 *esc_data_status_2 = &esc_data_2[i];
                if (esc_data_status_2->esc_id == esc_id ||
                   esc_data_status_2->esc_id == -2) {
                    buff_index_t = 0;
                    esc_data_status_2->esc_id = esc_id;
                    esc_data_status_2->mosfetTemp =
                        (float)valueCompose_I24(buff, &buff_index_t)*0.01f;
```

```
                        esc_data_status_2->motorTemp =
                            (float)valueCompose_I24(buff, &buff_index_t)*0.01f;
                        esc_data_status_2->battVoltage =
                            (float)valueCompose_I24(buff, &buff_index_t)*0.01f;
                        break;
                    }
                }
            }
        }
        break;

        default:
            break;
        }
}

// 将有符号短整型数据(int16_t)拆分成两个字节传输，高位数据在前，低位数据在后
void valueDecompose_I16(uint8_t* dataNum, int16_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// 将无符号短整型数据(uint16_t)拆分成两个字节传输，高位数据在前，低位数据在后
void valueDecompose_U16(uint8_t* dataNum, uint16_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// 将有符号长整型数据(int32_t)拆分成四个字节，最高8bit截取去掉，
// 保留低24bit，高位数据在前，低位数据在后
// 范围值为:-8388608~8388607
void valueDecompose_I24(uint8_t* dataNum, int32_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 16;
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// 将无符号长整型数据(uint32_t)拆分成四个字节，最高8bit截取去掉，
// 保留低24bit，高位数据在前，低位数据在后
// 范围值为:0~16777215
void valueDecompose_U24(uint8_t* dataNum, uint32_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 16;
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// 将有符号长整型数据(int32_t)拆分成四个字节传输，高位数据在前，低位数据在后
void valueDecompose_I32(uint8_t* dataNum, int32_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 24;
  dataNum[(*numInd)++] = value >> 16;
  dataNum[(*numInd)++] = value >> 8;
```

```
  dataNum[(*numInd)++] = value;
}

// 将无符号长整型数据(uint32_t)拆分成四个字节传输，高位数据在前，低位数据在后
void valueDecompose_U32(uint8_t* dataNum, uint32_t value, int16_t *numInd) {
  dataNum[(*numInd)++] = value >> 24;
  dataNum[(*numInd)++] = value >> 16;
  dataNum[(*numInd)++] = value >> 8;
  dataNum[(*numInd)++] = value;
}

// 将两个字节组合成有符号短整型数据(int16_t)，高位数据在前，低位数据在后
int16_t valueCompose_I16(const uint8_t *dataNum, int16_t *numInd) {
  int16_t retval =  ((uint16_t) dataNum[*numInd]) << 8 |
          ((uint16_t) dataNum[*numInd + 1]);
  *numInd += 2;
  return retval;
}

// 将两个字节组合成无符号短整型数据(uint16_t)，高位数据在前，低位数据在后
uint16_t valueCompose_U16(const uint8_t *dataNum, int16_t *numInd) {
  uint16_t retval =   ((uint16_t) dataNum[*numInd]) << 8 |
          ((uint16_t) dataNum[*numInd + 1]);
  *numInd += 2;
  return retval;
}

// 将三个字节组合成有符号长整型数据(int32_t)，
// 高位数据在前，低位数据在后
int32_t valueCompose_I32(const uint8_t *dataNum, int16_t *numInd) {
  int32_t retval = ((uint32_t) dataNum[*numInd]) << 16  |
          ((uint32_t) dataNum[*numInd + 1]) << 8 |
          ((uint32_t) dataNum[*numInd + 2]);
  *numInd += 3;
  return retval;
}

// 将三个字节组合成无符号长整型数据(uint32_t)，高位数据在前，低位数据在后
uint32_t valueCompose_U32(const uint8_t *dataNum, int16_t *numInd) {
  uint32_t retval = ((uint32_t) dataNum[*numInd]) << 16 |
          ((uint32_t) dataNum[*numInd + 1]) << 8 |
          ((uint32_t) dataNum[*numInd + 2]);
  *numInd += 3;
  return retval;
}

// 将四个字节组合成无符号长整型数据(uint32_t)，高位数据在前，低位数据在后
uint32_t valueCompose_U32(const uint8_t *dataNum, int16_t *numInd) {
  uint32_t retval = ((uint32_t) dataNum[*numInd]) << 24 |
          ((uint32_t) dataNum[*numInd + 1]) << 16 |
          ((uint32_t) dataNum[*numInd + 2]) << 8 |
```

```
          ((uint32_t) dataNum[*numInd + 3]);
  *numInd += 4;
  return retval;
}

// 将四个字节组合成有符号长整型数据(int32_t)，高位数据在前，低位数据在后
int32_t valueCompose_I32(const uint8_t *dataNum, int16_t *numInd) {
  int32_t retval =  ((uint32_t) dataNum[*numInd]) << 24 |
          ((uint32_t) dataNum[*numInd + 1]) << 16  |
          ((uint32_t) dataNum[*numInd + 2]) << 8 |
          ((uint32_t) dataNum[*numInd + 3]);
  *numInd += 4;
  return retval;
}

// 将四个字节组合成无符号长整型数据(uint32_t)，高位数据在前，低位数据在后
uint32_t valueCompose_U32(const uint8_t *dataNum, int16_t *numInd) {
  uint32_t retval = ((uint32_t) dataNum[*numInd]) << 24 |
          ((uint32_t) dataNum[*numInd + 1]) << 16 |
          ((uint32_t) dataNum[*numInd + 2]) << 8 |
          ((uint32_t) dataNum[*numInd + 3]);
  *numInd += 4;
  return retval;
}

// 将float型数据根据放大倍数转换成int16_t型数据并拆分成两个字节进行传输，高位数据在前，低位数据在后
void valueDecompose_F16(uint8_t* dataNum, float value, float multiple, int32_t *numInd) {
    valueDecompose_I16(dataNum, (int16_t)(value * multiple), numInd);
}

// 将float型数据根据放大倍数转换成int32_t型数据并拆分成四个字节进行传输，高位数据在前，低位数据在后
void valueDecompose_F32(uint8_t* dataNum, float value, float multiple, int32_t *numInd) {
    valueDecompose_I32(dataNum, (int32_t)(value * multiple), numInd);
}

// 将两个字节组合成有符号短整型数据(int16_t)，高位数据在前，低位数据在后，并根据放大倍数转换成float型数据
float valueCompose_F16(const uint8_t *dataNum, float multiple, int16_t *numInd) {
    return (float)valueCompose_I16(dataNum, numInd) / multiple;
}

// 将四个字节组合成有符号短整型数据(int32_t)，高位数据在前，低位数据在后，并根据放大倍数转换成float型数据
float valueCompose_F32(const uint8_t *dataNum, float multiple, int16_t *numInd) {
    return (float)valueCompose_I32(dataNum, numInd) / multiple;
}
```