文档标题	**FLIPSKY FTESC UART通信协议 V1.4**	文件版本	V1.4	
		FT固件版本	V1.4	
		发布日期	2024/07/12	
文件编号	FT-CM-UART-20240712-01			
产品型号	翻天科技**FT**系列电调	共 **21** 页	级别	公开

FLIPSKY FTESC UART

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	2024-07-05	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Flipsky ESC Tool、VX4接收机、VX3Pro接收机或其他用户开发的控制单元被定义为ECU。本文旨在协助客户如何使用UART端口访问以及控制FT系列电调。

2 报文格式说明

MCU均带有UART端口，故用户在自行开发ECU时可使用UART端口与MCU进行通信，如使用RS232，RS485等电气接口。

2.1 通信参数

波特率： 115200 BPS
起始位： 1 bit
数据位： 8 bit
停止位： 1 bit
校验位： 无
模式： 半/全双工
流控： 无

2.2 报文格式

STX	DLEN	CMD	DATA	CRC16	ETX
报头	数据长度	指令	数据	16位CRC校验（CMD+DATA）	报尾
1 字节	1~2 字节	1 字节	N 字节	2 字节	1 字节

通信数据帧格式：

名称	长度 (bytes)	说明
帧起始符（STX）	1 byte	0xAA / 0xBB： (1) 0xAA：数据长度 DLEN 小于等于255 字节； (2) 0xBB：数据长度 DLEN 大于 255 字节并且小于等于65535字节。

数据长度 (DLEN)		1~2 bytes	链路层信息（INFO）字节总长度（CMD+DATA）： (1) 当链路层信息（INFO）长度小于等于255 字节，则DLEN 为1byte； (2) 当链路层信息（INFO）长度大于255 字节并且小于等于65535字节，则DLEN 为2byte；
链路层信息（INFO）	指令/应答（CMD）	1 byte	(1) 对于ECU发送方，在发送相关控制指令时，该字节用于指示哪种指令，如电流控制指令、速度控制指令； (2) 对于MCU接收方，接收到来自ECU的指令并对该指令进行解析，同时返回应答信号到ECU。 具体指令见2.5。
	数据（DATA）	N	指令附带的数据。
CRC16		2 bytes	16位的CRC16校验码，具体解析见2.4。
帧结束符（ETX）		1byte	0xDD，代表一帧数据结束。

2.3 实现注解

ECU与 MCU 间通信采用指令/应答/消息机制，ECU发送指令/消息，MCU收到来自ECU的指令/消息后需要发送应答给ECU。

注意：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。

2.4 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);
}
```

2.5 UART COMMAND 指令码

```
typedef enum {
  UART_OBTAIN_DATA_ONCE                       = 0,
  UART_CONTROL_AND_OBTAIN_DATA_ONCE           = 2,
  UART_SET_DUTY                               = 3,
  UART_SET_CURRENT                            = 4,
  UART_SET_CURRENT_GEAR                       = 5,
  UART_SET_BRAKE_CURRENT                      = 6,
  UART_CAN_OVER                               = 16,
  UART_OBTAIN_FIRMWARE_VERSION                = 17,
  UART_KEEP_LIVE                              = 25,
  UART_SET_AUTO_OBTAIN_REALTIME_DATA          = 26,
  UART_RESET_AND_REBOOT_FTESC                 = 29,
  UART_OBTAIN_ALL_FTESC_ID                    = 30,
  UART_SET_CURRENT_GEAR_AND_OBTAIN_DATA       = 32,
  UART_REBOOT_FTESC                           = 35,
  UART_SET_ID_CURRENT                         = 37,
  UART_SET_POSITION                           = 38,
  UART_SET_SPEED                              = 39,
} UART_COMMAND;
```

2.6 MCU ERROR 故障码

```
typedef enum {
  ERROR_NONE                                  = 0,
  ERROR_FOR_PHASE_A_SOFTWARE_OVER_CURRENT     = 1,   //A相软件ABS过流
  ERROR_FOR_PHASE_B_SOFTWARE_OVER_CURRENT     = 2,   //B相软件ABS过流
  ERROR_FOR_PHASE_C_SOFTWARE_OVER_CURRENT     = 3,   //C相软件ABS过流
  ERROR_FOR_PHASE_A_CURRENT_SENSOR            = 4,   //A相电流传感器故障
  ERROR_FOR_PHASE_B_CURRENT_SENSOR            = 5,   //B相电流传感器故障
  ERROR_FOR_PHASE_C_CURRENT_SENSOR            = 6,   //C相电流传感器故障
  ERROR_FOR_PHASE_CURRENTS_SUM_NOT_ZERO       = 7,   //三相电流之和不为零
  ERROR_FOR_BUS_UNDER_VOLTAGE                 = 8,   //母线欠压
  ERROR_FOR_BUS_OVER_VOLTAGE                  = 9,   //母线过压
  ERROR_FOR_MOSFET_OVER_HEAT                  = 10,  //MOS管过热
  ERROR_FOR_MOTOR_OVER_HEAT                   = 11,  //电机过热
  ERROR_FOR_MOSFET_TEMPERATURE_SENSOR         = 12,  //MOS管温度传感器故障
  ERROR_FOR_BOOTING_FROM_WATCHDOG_RESET       = 13,  //看门狗复位
  ERROR_FOR_FLASH_CORRUPTION                  = 14,  //闪存故障
  ERROR_FOR_MCU_UNDER_VOLTAGE                 = 15,  //单片机低压报警
  ERROR_FOR_MOTOR_TEMPERATURE_SENSOR          = 16,  //电机温度传感器故障
  ERROR_FOR_MOTOR_BLOCKING                    = 17,  //电机堵转故障
  ERROR_FOR_DRIVER                            = 18,  //驱动芯片报错
  ERROR_FOR_MOTOR_OUT_OF_PHASE                = 19,  //缺相保护故障
  ERROR_FOR_BUS_OVER_CURRENT                  = 20,  //母线过流
```

```
  ERROR_END                                   = 23,
} Motor_General_Error_Code;
```

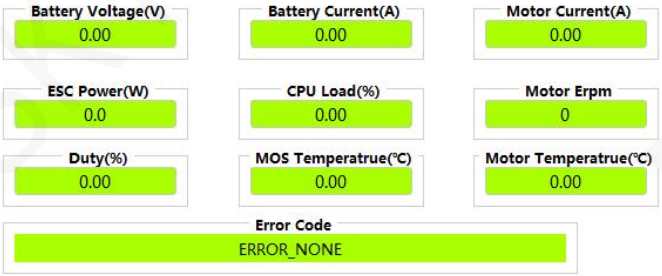
3 指令/应答 消息详解

3.1 指令解析

3.1.1 UART_OBTAIN_DATA_ONCE (00H)

1）功能

获取一次实时数据。如下图所示。



2）报文

项目	值	说明
长度（DLEN）	01H	
指令（CMD）	00H	
数据（DATA）	无	

3）应答

项目名称	值	说明
长度（DLEN）	1DH	
指令（CMD）	00H	
数据（DATA）	D0	MCU ID
	D1	ERROR CODE 故障代码
	D2 D3	Battery Voltage 电池电压 注：放大100倍（int16_t）
	D4 D5 D6 D7	Battery Voltage 电池电流 注：放大1000000倍（int32_t）
	D8 D9 D10 D11	Motor Current 电机电流 注：放大1000000倍（int32_t）
	D12 D13 D14 D15	Motor Erpm 电机电子转速 注：（int32_t）
	D16 D17	Duty 占空比 注：放大10000倍（int16_t）
	D18 D19	MOSFET Temp MOS管温度 注：放大100倍（int16_t）

	D20 D21	Motor Temp 电机温度 注：放大100倍（int16_t）
	D22 D23	CPU Utilization CPU 利用率 注：放大10000倍（int16_t）
	D24 D25 D26 D27	Encoder Angle 编码器角度 注：放大1000000倍（int32_t）

3.1.2 UART_CONTROL_AND_OBTAIN_DATA_ONCE (02H)

1）功能

发送一次UART控制指令并返回一次实时数据。

注意：该指令必须是在输入信号类型（Input Signal Type）设置为UART模式下才有效。主要应用在电动滑板（ESK8）或者电动冲浪板（ESURF）。

2）报文

项目	值	说明
长度（DLEN）	0FH	
指令（INFO）	02H	
数据（DATA）	D0 D1	范围： 0~1023 0~511: 刹车电流或者负向电流 512~1023: 驱动电流（uint16_t）
	D2 D3	保留，默认为0X0000
	D4	使能方向切换 00H: 禁止切换方向 01H: 使能切换方向
	D5	电机旋转方向 00H: 正向旋转 01H: 反向旋转 **注意：** 必须使能切换方向（D4置1），D5设置才有效
	D6	限速挡位设置 00H: 无挡位 01H: 低速挡 02H: 中速挡 03H: 高速挡 04H: 倒车挡

	D7	外置喇叭控制 00H: 关闭 01H: 开启
	D8	外置前照灯控制 00H: 关闭 01H: 开启
	D9	外置刹车灯控制 00H: 关闭 01H: 开启
	D10	使能巡航模式控制 00H: 禁止使用巡航模式 01H: 允许使用巡航模式
	D11	巡航模式开启 00H：退出巡航模式 01H：进入巡航模式
	D12	使能多模式控制 00H：禁止使用多模式控制 01H：允许使用多模式控制
	D13	多模式控制切换 00H：正常控制 01H：坦克掉头模式 02H：爬坡越野模式

3）应答

项目名称	值	说明
长度（DLEN）	1CH	
指令（CMD）	02H	
数据（DATA）	D0	ERROR CODE 故障代码
	D1 D2	Battery Voltage 电池电压 注：放大100倍（int16_t）
	D3 D4 D5 D6	Battery Current 电池电流 注：放大1000000倍（int32_t）
	D7 D8 D9 D10	Motor Current 电机电流 注：放大1000000倍（int32_t）
	D11 D12 D13 D14	Motor Erpm 电机电子转速 注：（int32_t）
	D15 D16	Duty 占空比 注：放大10000倍（int16_t）

	D17 D18	MOSFET Temp MOS管温度 注：放大100倍（int16_t）
	D19 D20	Motor Temp 电机温度 注：放大100倍（int16_t）
	D21 D22	CPU Utilization CPU 利用率 注：放大10000倍（int16_t）
	D23 D24 D25 D26	Encoder Angle 编码器角度（单位：°） 注：放大1000000倍（int32_t）

3.1.3 UART_SET_DUTY (03H)

1）功能

占空比控制指令。

注意：该指令必须是在输入信号类型（Input Signal Type）设置为OFF下才有效。

2）报文

项目	值	说明
长度（DLEN）	05H	
指令（CMD）	03H	
数据（DATA）	D0 D1 D2 D3	Duty 占空比 注： 放大100000倍。比如，占空比为0.215，则需要发送21500。（int32_t）

3）应答

注意： 该指令无应答。

3.1.4 UART_SET_CURRENT (04H)

1）功能

电机电流控制指令。

注意：同3.1.3。

2）报文

项目	值	说明
长度（DLEN）	05H	
指令（CMD）	04H	
数据（DATA）	D0 D1 D2 D3	Current 电机电流 注： 放大1000倍。比如，电流为50.45，则需要发送50450。（int32_t）

3）应答

注意： 该指令无应答。

3.1.5 UART_SET_CURRENT_GEAR (05H)

1）功能

电机电流控制指令，并且带有档位可设置。

注意：同3.1.3。

2）报文

项目	值	说明
长度（DLEN）	06H	
指令（CMD）	05H	
数据（DATA）	D0 D1 D2 D3	Current 电机电流 注： 放大1000倍。比如，电流为50.45，则需要发送50450。（int32_t）
	D4	GEAR 限速挡位 00H: 无挡位 01H: 低速挡 02H: 中速挡 03H: 高速挡 04H: 倒车挡

3）应答

注意： 该指令无应答。

3.1.6 UART_SET_BRAKE_CURRENT (06H)

1）功能

电机刹车电流控制指令。

注意：同3.1.3。

2）报文

项目	值	说明
长度（DLEN）	05H	
指令（CMD）	06H	
数据（DATA）	D0 D1 D2 D3	Brake Current 电机刹车电流 注： 放大1000倍。比如，电流为50.45，则需要发送50450。（int32_t）

3）应答

注意： 该指令无应答。

3.1.7 UART_CAN_OVER (10H)

1）功能

当ECU与Master MCU通过UART接口通信时，通过该指令，可以让ECU访问挂接在CAN总线上的所有Slave MCU。

2）报文

项目		值	说明
长度（DLEN）		XXH	根据不同指令而确定。
指令（Master CMD）		10H	发送到Master MCU的指令
数据 DATA	从机ID Slave ID	D0	Slave ID，即需要被发送到的指定从机ID
	从机指令 Slave CMD	D1	发送到Slave MCU的指令
	从机数据 Slave DATA	D2~Dn	发送到Slave MCU的数据

3）应答

项目	值	说明
长度（DLEN）	XXH	根据不同指令而确定。
指令（Slave CMD）	D0	Slave MCU返回到Master ECU的指令，此值应与3.1.7报文中的Slave CMD一致
数据（Slave DATA）	D1~Dn	Slave MCU返回到ECU的数据

3.1.8 UART_OBTAIN_FIRMWARE_VERSION (11H)

1）功能

获取MCU的固件版本、硬件名字以及硬件序号等信息。

2）报文

项目	值	说明
长度（DLEN）	01H	
指令（CMD）	11H	
数据（DATA）	无	

3）应答

项目	值	说明
长度（DLEN）	XXH	
指令（CMD）	11H	
数据（DATA）	D0	版本号FIRST
	D1	版本号SECOND
	D2	0XAC: Bootloader 模式

		0XEF : APP模式
	D3~D(3+n)	硬件名称ASCII码值
	D(3+n+1)	硬件序号

3.1.9 UART_KEEP_LIVE (19H)

1）功能

在发送相关控制指令后（如占空比控制、电流控制），至少每隔500ms发送该指令（19H）到MCU，可令上次指令持续有效。

2）报文

项目	值	说明
长度（DLEN）	01H	
指令（CMD）	19H	
数据（DATA）	无	

3）应答

项目	值	说明
长度（DLEN）	01H	
指令（CMD）	19H	
数据（DATA）	无	

3.1.10 UART_SET_AUTO_OBTAIN_REALTIME_DATA (1AH)

1）功能

设置自动返回实时数据。

注意：开启自动返回实时数据后，需要确保该指令至少每隔800ms被发送到MCU一次，否则MCU将会监测到超时后，停止返回实时数据。同时，在该指令（1AH）应答完成后，实时返回数据格式将按照3.1.1应答格式进行回传。

2）报文

项目	值	说明
长度（DLEN）	04H	
指令（CMD）	1AH	
数据（DATA）	D0	是否开启自动返回实时数据 0：关闭 1：开启
	D1 D2	自动返回数据频率： 设置范围：1～1000Hz（uint16_t）

3）应答

项目	值	说明
长度（DLEN）	08H	
指令（CMD）	1AH	
数据（DATA）	D0	自动返回实时数据标记
	D1 D2	自动返回数据频率
	D3 D4 D5 D6	自动返回函数地址

3.1.11 UART_RESET_AND_REBOOT_FTESC (1DH)

1）功能

使MCU恢复默认出厂设置并重启。ECU收到应答50毫秒后，MCU开始复位重启。

2）报文

项目	值	说明
长度（DLEN）	02H	
指令（CMD）	1DH	
数据（DATA）	D0	保留，默认为0X00

3）应答

项目	值	说明
长度（DLEN）	01H	
指令（CMD）	1DH	
数据（DATA）	无	

3.1.12 UART_OBTAIN_ALL_FTESC_ID (1EH)

1）功能

获取所有MCU的ID，包含Master MCU的ID。

2）报文

项目	值	说明
长度（DLEN）	01H	
指令（CMD）	1EH	
数据（DATA）	无	

3）应答

项目	值	说明
长度（DLEN）	XXH	由总MCU个数确定
指令（CMD）	1EH	

数据（DATA）	D0	Master MCU ID
	D1~Dn	依次为各个Slave MCU ID

3.1.13 UART_SET_CURRENT_GEAR_AND_OBTAIN_DATA (20H)

1）功能

带有限速挡位的电机电流控制指令，并同时返回一次实时数据。

注意：同3.1.3。

2）报文

项目	值	说明
长度（DLEN）	06H	
指令（CMD）	20H	
数据（DATA）	D0 D1 D2 D3	Current 电机电流 注： 放大1000倍。比如，电流为50.45，则需要发送50450。（int32_t）
	D4	GEAR 限速挡位 00H: 无挡位 01H: 低速挡 02H: 中速挡 03H: 高速挡 04H: 倒车挡

3）应答

项目名称	值	说明
长度（DLEN）	1CH	
指令（CMD）	20H	
数据（DATA）	D0	ERROR CODE 故障代码
	D1 D2	Battery Voltage 电池电压 注：放大100倍（int16_t）
	D3 D4 D5 D6	Battery Voltage 电池电流 注：放大1000000倍（int32_t）
	D7 D8 D9 D10	Motor Current 电机电流 注：放大1000000倍（int32_t）
	D11 D12 D13 D14	Motor Erpm 电机电子转速 注：（int32_t）
	D15 D16	Duty 占空比 注：放大10000倍（int16_t）

	D17 D18	MOSFET Temp MOS管温度 注：放大100倍（int16_t）
	D19 D20	Motor Temp 电机温度 注：放大100倍（int16_t）
	D21 D22	CPU Utilization CPU 利用率 注：放大10000倍（int16_t）
	D23 D24 D25 D26	Encoder Angle 编码器角度（单位：°） 注：放大1000000倍（int32_t）

3.1.14 UART_REBOOT_FTESC (23H)

1）功能

复位MCU。

2）报文

项目	值	说明
长度（DLEN）	01H	
指令（CMD）	23H	
数据（DATA）	无	

3）应答

注意： 该指令无应答。

3.1.15 UART_SET_ID_CURRENT (25H)

1）功能

电机D轴电流控制指令。该指令可以让电机锁定在零相位处。

注意：同3.1.3。

2）报文

项目	值	说明
长度（DLEN）	05H	
指令（CMD）	25H	
数据（DATA）	D0 D1 D2 D3	D-axis Current 电机D轴电流 注：放大1000倍。比如，D轴电流为50.45，则需要发送50450。（int32_t）

3）应答

注意： 该指令无应答。

3.1.16 UART_SET_POSITION (26H)

1）功能

位置控制指令。

注意：

① 仅在编码器模式下才能使用；

② 同3.1.3。

2）报文

项目	值	说明
长度（DLEN）	05H	
指令（CMD）	26H	
数据（DATA）	D0 D1 D2 D3	Position 位置控制 注： 放大1000倍。（int32_t）

3）应答

注意： 该指令无应答。

3.1.17 UART_SET_SPEED (27H)

1）功能

电机电子转速控制指令。

注意：该指令必须是在输入信号类型（Input Signal Type）设置为OFF模式下才有效。

2）报文

项目	值	说明
长度（DLEN）	05H	
指令（CMD）	27H	
数据（DATA）	D0 D1 D2 D3	电机电子转速控制 注：（int32_t）

3）应答

注意： 该指令无应答。

3.2 指令/应答 操作举例（C语言）

下面代码以指令UART_OBTAIN_DATA_ONCE (00H)举例来阐述整个操作流程。

3.2.1 ECU发送指令 (00H)

```
static uint8_t messageSend[1024];

void FTESC_Uart_packSendPayload(uint8_t * payload, int Dlen)
{
  uint16_t crcPayload;
  int count = 0;

  crcPayload= crc16(payload, Dlen);
  if (Dlen <= 255) {
    messageSend[count++] = 0xAA;
    messageSend[count++] = Dlen;
  } else if (Dlen <= 65535) {
    messageSend[count++] = 0xBB;
    messageSend[count++] = (Dlen>>8)&0X00FF;
    messageSend[count++] = Dlen&0X00FF;
  }
  memcpy(&messageSend[count], payload, Dlen);
  count += Dlen;
  messageSend[count++] = (uint8_t)(crcPayload >> 8);
  messageSend[count++] = (uint8_t)(crcPayload & 0xFF);
  messageSend[count++] = 0xDD;
  messageSend[count] = '\0';

  USART_Sendstring(messageSend, count);
}

// 执行该函数，发送指令到MCU端
void Ftesc_Obtain_Realtime_Data(void)
{
  uint8_t payload[1];
  payload[0] = UART_OBTAIN_DATA_ONCE;
  FTESC_Uart_packSendPayload(payload, 1);  //串口发送函数
}
```

3.2.2 ECU接收到MCU返回的应答(00H)数据包

```
// 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;
}

typedef struct
{
  uint8_t controller_id;
  uint8_t fault;
  float inpVoltage;
  float inputCurrent;
  float motorCurrent;
  float rpm;
  float dutyCycleNow;
  float temp_fet;
  float temp_motor;
  float cpu_load;
  float encoder_angle;
}ftesc_realtime_data;
```

```
ftesc_realtime_data esc_data;

// 一般使用空闲中断触发串口接收完成，缓冲区uart_rx_buff通过DMA传输。
// 空闲中断触发后，即可以对缓冲区里的数据进行解析
int8_t Ftesc_Uart_processReadPacket(uint8_t * uart_rx_buff)
{
  uint16 data_len = 0;
  uint8_t * data = 0;

  // 判断帧头是否正确
  if (uart_rx_buff[0] == 0xAA) {
    data_len = uart_rx_buff[1];
    data = uart_rx_buff + 2;
  } else if (uart_rx_buff[0] == 0xBB) {
    data_len |= (uint16_t)uart_rx_buff[1] << 8; // 高8位
    data_len |= (uint16_t)uart_rx_buff[2];      // 低8位
    data = uart_rx_buff + 3;
  } else {
    return -1;    // 数据包格式异常，必须是以0XAA或者0XBB开头
  }

  // 计算链路层数据crc16校验码以及获取接收到的crc16校验码
  uint16 data_crc16_calc = crc16(data, data_len);
  uint16 data_crc16_rece = (uint16)data[data_len] << 8 | (uint16)data[data_len + 1];

  // 判断CRC16校验是否正确
  if (data_crc16_calc != data_crc16_rece) {
    return -2;            // 数据包CRC16校验失败
  }

  // 判断帧尾是否正确
  if (data[data_len + 2] != 0XDD) {  //必须是以0XDD结尾
    return -3;
  }

  UART_COMMAND uart_cmd = (UART_COMMAND)data[0];   // 获取指令
  uint16_t index = 0;
  data++;  // 偏移至DATA区域
  switch (uart_cmd)
  {
    case UART_OBTAIN_DATA_ONCE:{
      esc_data.controller_id  = data[index++];
      esc_data.fault          = data[index++];
      esc_data.inpVoltage     = valueCompose_F16( data, 100.0f, &index );
      esc_data.inputCurrent   = valueCompose_F32( data, 1000000.0f, &index );
      esc_data.motorCurrent   = valueCompose_F32( data, 1000000.0f, &index );
      esc_data.rpm            = valueCompose_F32( data, 1.0f, &index );
      esc_data.dutyCycleNow   = valueCompose_F16( data, 10000.0f, &index );
      esc_data.temp_fet       = valueCompose_F16( data, 100.0f, &index );
      esc_data.temp_motor     = valueCompose_F16( data, 100.0f, &index );
      esc_data.cpu_load       = valueCompose_F16( data, 10000.0f, &index );
```

```
      esc_data.encoder_angle  = valueCompose_F32( data, 1000000.0f, &index );
    }
    break;

    default:
      break;
  }

  return 0;   //帧解析完成
}
```