Title	**FLIPSKY FTESC UART Communication Protocol V1.4**	**Version**	V 1. 4	
		FT Firmware Version	V1.4	
		Issue Date	2025/04/22	
Number	FT-CM-UART-20250422-01			
Product Model	**FLIPSKY FT Series ESC**	**Total 21 pages**	**Level**	**Public**

FLIPSKY FTESC
UART Communication Protocol V1.4

Guangdong FLIPSKY Technology Co., Ltd.

Revision Control

Version	Revision Date	Revised by	Revision Summary
1. 4	2024-07-05	Lu	
	2025-04-22	Lu	

Catalog

1 Summary

This document describes the communication protocol between the Motor Control Unit (MCU) and the Electronic Control Unit(ECU). In this document, MCU refers to the FT series ESC developed by FLIPSKY only, the Flipsky ESC Tool, VX4 receiver, VX3Pro receiver or other user-developed control units are defined as ECUs. This document aims to assist customers to use the UART ports to access and control the FT series ESC.

2 Message format description

MCU are all equipped with UART ports. Users can use UART ports, such as electrical interface like RS232，RS485, to connect with MCU when developing their own ECU.

2.1 Communication parameters

Baud Rate ：115200 BPS
Start Bit : 1 bit
Data Bit : 8 bit
Stop Bit : 1 bit
Parity Bit: None
Mode: Half/Full Duplex
Flow Control: None

2.2 Message Format

STX	DLEN	CMD	DATA	CRC 16	ETX
Header	Data length	Instruction	Data	16-bit CRC check (CMD + DATA)	End of the report
1 byte	1~2 bytes	1 byte	N byte	2 bytes	1 byte

Communication Data Frame Format:

Name	Length (bytes)	Explanation
STX	1 byte	0xAA / 0xBB : (1) 0xAA : DLEN <= 255 bytes； (2) 0xBB : 255 bytes < DLEN <= 65535 bytes.

DLEN		1 ~ 2 bytes	Total length of INFO bytes（CMD+DATA）： (1) DLEN of INFO <= 255 bytes, the DLEN is 1 byte (2) 255 bytes < DLEN of INFO <= 65535 bytes, the DLEN is 2 bytes；
INFO	CMD	1 byte	(1) For the ECU sender , when sending relevant control instructions , this byte is used for concrete commands, like current control and speed control. (2) The MCU receiver will receive and parse the commands from ECU, and return response signals to it. See 2.5 for specific instructions.
	DATA	N	Data attached to the commands.
CRC16		2 bytes	CRC16 of 16 bits, read 2.4. for more details.
ETX		1 byte	0xDD, indicating the end of a frame of data.

2.3 Implementation Annotation

The communication between ECU and MCU adopts the command /response/ message system. ECU sends instructions /messages, and MCU needs to send a response to ECU after receiving instructions /messages from ECU .

Note: In the frame protocol of messages and responses,

(1) **uint16_t** data, **int16_t** data, **uint32_t** data, and **int32_t data** involved are all transmitted in Big-endian mode, i.e., the high-order byte data comes first (low address) and the low-order byte data comes second (high address).

(2) Single-byte data without special mark is **uint8_t** type data.

2.4 CRC

```
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

```
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

```
typedef enum {
ERROR_NONE                      = 0,
ERROR_FOR_PHASE_A_SOFTWARE_OVER_CURRENT  = 1, //Phase A software ABS over current
ERROR_FOR_PHASE_B_SOFTWARE_OVER_CURRENT  = 2, //Phase B software ABS over current
ERROR_FOR_PHASE_C_SOFTWARE_OVER_CURRENT  = 3, //Phase C software ABS over current
ERROR_FOR_PHASE_A_CURRENT_SENSOR         = 4, //Phase A current sensor fault
ERROR_FOR_PHASE_B_CURRENT_SENSOR         = 5, //Phase B current sensor fault
ERROR_FOR_PHASE_C_CURRENT_SENSOR         = 6, //Phase C current sensor fault
ERROR_FOR_PHASE_CURRENTS_SUM_NOT_ZERO    = 7, //The sum of the three-phase current is not zero

ERROR_FOR_BUS_UNDER_VOLTAGE              = 8, //Bus under voltage
ERROR_FOR_BUS_OVER_VOLTAGE               = 9, //Bus over voltage
ERROR_FOR_MOSFET_OVER_HEAT               = 10, //MOSFET overheating
ERROR_FOR_MOTOR_OVER_HEAT                = 11, //Motor overheating
ERROR_FOR_MOSFET_TEMPERATURE_SENSOR      = 12, //MOSFET temperature sensor failure
ERROR_FOR_BOOTING_FROM_WATCHDOG_RESET    = 13, //Watchdog reset
ERROR_FOR_FLASH_CORRUPTION               = 14, //Flash memory failure
ERROR_FOR_MCU_UNDER_VOLTAGE              = 15, //MCU low voltage alarm
ERROR_FOR_MOTOR_TEMPERATURE_SENSOR       = 16, //Motor temperature sensor fault
ERROR_FOR_MOTOR_BLOCKING                 = 17, //Motor blocking fault
ERROR_FOR_DRIVER                         = 18, //Driver chip error
```

```
ERROR_FOR_MOTOR_OUT_OF_PHASE            = 19, //Phase loss protection fault
ERROR_FOR_BUS_OVER_CURRENT              = 20, //Bus over current
ERROR_END                               = 23,
} Motor_General_Error_Code;
```

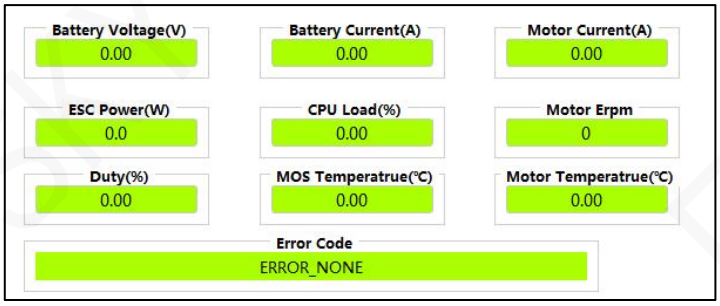
3 Command /Response Message Details

3.1 Instruction Parsing

3.1.1 UART_OBTAIN_DATA_ONCE (00H)

1) Function

Get real-time data once , as shown as the figure below.



2) Message

Item	Value	Explanation
DLEN	01H	
CMD	00H	
DATA	None	

3) Response

Item	Value	Explanation
DLEN	1DH	
CMD	00H	
DATA	D0	MCU ID
	D1	ERROR CODE
	D2 D3	Battery Voltage Note: Amplify 100 times (int16_t)
	D4 D5 D6 D7	Battery Current Note: Amplify 1000000 times (int32_t)
	D8 D9 D10 D11	Motor Current Note: Amplify 1000000 times (int32_t)
	D12 D13 D14 D15	Motor Erpm Note: (int32_t)
	D16 D17	Duty Note: Amplify 10000 times (int16_t)
	D18 D19	MOSFET Temp Note: Amplify 100 times (int16_t)

	D20 D21	Motor Temp Note: Amplify 100 times (int16_t)
	D22 D23	CPU Utilization Note: Amplify 10000 times (int16_t)
	D24 D25 D26 D27	Encoder Angle Note: Amplify 1000000 times (int32_t)

3.1.2 UART_CONTROL_AND_OBTAIN_DATA_ONCE (0 2 H)

1) Function

Send a UART control command once and return real-time data.

Note : This command is only valid when the Input Signal Type is set to UART mode. It is mainly used in the electric skateboards (ESK8) or electric surfboards (ESURF).

2) Message

Item	Value	Explanation
DLEN	0FH	
INFO	02H	
DATA	D0 D1	Range : 0~1023 0~511: Braking current/Negative current 512~1023: Driving current (uint16_t)
	D2 D3	Reserved , default as 0X0000
	D4	Enable Direction Switching 00H: Direction switching disabled 01H: Direction switching enabled
	D5	Motor Rotation Direction 00H: Forward 01H: Reverse **Note:** D5 setting is only valid when the direction switching enabled (D4 set to 1)

	D6	Speed Limit Gear Settings 00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse gear
	D7	External Horn Control 00H: Off 01H: On
	D8	External Headlight Control 00H: Off 01H: On
	D9	External Brake Light Control 00H: Off 01H: On
	D10	Enable Cruise Mode Control 00H: Disable cruise mode 01H: Enable cruise mode
	D11	Cruise Mode On 00H: Exit cruise mode 01H: Enter cruise mode
	D12	Enable Multi-mode Control 00H: Disable multi-mode control 01H: Enable multi-mode control
	D13	Multi-mode Control Switching 00H: Normal control 01H: Tank U-turn mode 02H: Hill climbing off-road mode

3) Response

Item	Value	Explanation
DLEN	1CH	
CMD	02H	
DATA	D0	ERROR CODE
	D1 D2	Battery Voltage Note: Amplify 100 times (int16_t)
	D3 D4 D5 D6	Battery Current Note: Amplify 1000000 times (int32_t)
	D7 D8 D9 D10	Motor Current Note: Amplify 1000000 times (int32_t)

	D11 D12 D13 D14	Motor Erpm Note: (int32_t)
	D15 D16	Duty Note: Amplify 10000 times (int16_t)
	D17 D18	MOSFET Temp MOS tube temperature Note: Amplify 100 times (int16_t)
	D19 D20	Motor Temp Note: Amplify 100 times (int16_t)
	D21 D22	CPU Utilization Note: Amplify 10000 times (int16_t)
	D23 D24 D25 D26	Encoder Angle (unit : °) Note: Amplify 1000000 times (int32_t)

3.1.3 UART_SET_DUTY (0 3 H)

1) Function

Duty cycle control command.

Note : This command is only valid when the Input Signal Type is set to OFF.

2) Message

Item	Value	Explanation
DLEN	05H	
CMD	03H	
DATA	D0 D1 D2 D3	Duty Note : Amplify 100000 times. For example, if the duty cycle is 0.215, the sending data is 21500. (int32_t)

3) Response

Note : This command is of no response.

3.1.4 UART_SET_CURRENT (0 4 H)

1) Function

Motor current control command.

Note : This command is only valid when the Input Signal Type is set to OFF.

2) Message

Item	Value	Explanation
DLEN	05H	
CMD	04H	

DATA	D0 D1 D2 D3	Current Motor current Note : Amplify 1000 times. For example, if the current is 50.45, the sending data is 50450. (int32_t)

3) Response

Note : This command is of no response.

3.1.5 UART_SET_CURRENT_GEAR (0 5H)

1) Function

Motor current control command with adjustable gear .

Note : This command is only valid when the Input Signal Type is set to OFF.

2) Message

Item	Value	Explanation
DLEN	06H	
CMD	05H	
DATA	D0 D1 D2 D3	Current - Note : Amplify 1000 times. For example, if the current is 50.45, the sending data is 50450. (int32_t)
	D4	GEAR 00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse

3) Response

Note : This command is of no response.

3.1.6 UART_SET_BRAKE_CURRENT (06H)

1) Function

Motor brake current control command .

Note : This command is only valid when the Input Signal Type is set to OFF.

2) Message

Item	Value	Explanation
DLEN	05H	
CMD	06H	

DATA	D0 D1 D2 D3	Brake Current Note : Amplify 1000 times. For example, if the current is 50.45, the sending data is 50450. (int32_t)

3) Response

Note : This command is of no response.

3.1.7 UART_CAN_OVER (10H)

1) Function

When the ECU communicates with the Master MCU through the UART interface, this instruction allows the ECU to access all the Slave MCU connected to the CAN bus .

2) Message

Item		Value	Explanation
DLEN		XXH	Determined by different commands.
Master CMD		10H	Command sent to Master MCU
DATA	Slave ID	D0	
	Slave CMD	D1	Command sent to Slave MCU
	Slave DATA	D2~Dn	Corresponding data sent to Slave MCU 's

3) Response

Item	Value	Explanation
DLEN	XXH	Determined by different instructions .
Slave CMD	D0	The command returned from Slave MCU to Master ECU. This value shall be the same as the Slave CMD in the message 3.1.7.
Data (Slave DATA)	D1~Dn	Slave Data returned from MCU to ECU

3.1.8 UART_OBTAIN_FIRMWARE_VERSION (11H)

1) Function

Get the firmware version of MCU, hardware name, hardware serial number and other information.

2) Message

Item	Value	Explanation
DLEN	01H	
CMD	11H	

DATA	None	

3) Response

Item	Value	Explanation
DLEN	XXH	
CMD	11H	
DATA	D0	Version number: FIRST
	D1	Version number: SECOND
	D2	0XAC: Bootloader Mode 0XEF : APP Mode
	D3~D(3+n)	Hardware name ASCII code value
	D(3+n+1)	Hardware serial number

3.1.9 UART_KEEP_LIVE (19H)

1) Function

After sending the relevant control command (such as duty cycle control, current control), send the command (19H) to the MCU at least every 500ms to make the last command continue to be valid.

2) Message

Item	Value	Explanation
DLEN	01H	
CMD	19H	
DATA	None	

3) Response

Item	Value	Explanation
DLEN	01H	
CMD	19H	
DATA	None	

3.1.10 UART_SET_AUTO_OBTAIN_REALTIME_DATA (1A H)

1) Function

Set to return the real-time data automatically.

Note: After turning on automatic return of real-time data, it is necessary to ensure that the command is sent to the MCU at least every 800ms, otherwise the MCU will stop returning

real-time data when detect the timeout. At the same time, the real-time return data format will be transmitted back according to the response format of 3.1.1 after the command (1AH) is replied.

2) Message

Item	Value	Explanation
DLEN	04H	
CMD	1AH	
DATA	D0	Whether to enable automatic return of real-time data 0: Off 1: On
	D1 D2	Automatically return data frequency: Setting range: 1 ~ 1000Hz (uint16_t)

3) Response

Item	Value	Explanation
DLEN	08H	
CMD	1AH	
DATA	D0	Automatic return of real-time data marks
	D1 D2	Automatic return data frequency
	D3 D4 D5 D6	Automatic return function address

3.1.11 UART_RESET_AND_REBOOT_FTESC (1DH)

1) Function

Reset the MCU to its default factory settings and reboot. The MCU start to reset and restart after 50ms of the ECU receiving the response.

2) Message

Item	Value	Explanation
DLEN	02 H	
CMD	1D H	
DATA	D0	Reserved , default as 0X00

3) Response

Item	Value	Explanation
DLEN	01H	
CMD	1DH	
DATA	None	

3.1.12 UART_OBTAIN_ALL_FTESC_ID (1E H)

1) Function

Get the ID of all MCU , including the ID of the Master MCU .

2) Message

Item	Value	Explanation
DLEN	01H	
CMD	1EH	
DATA	None	

3) Response

Item	Value	Explanation
DLEN	XXH	Determined by the total number of MCU
CMD	1EH	
DATA	D0	Master MCU ID
	D1~Dn	Slave MCU ID in turn

3.1.13 UART_SET_CURRENT_GEAR_AND_OBTAIN_DATA (20 H)

1) Function

Motor current control command with speed limit gear. Return the real-time data at the same time.

Note : This command is only valid when the Input Signal Type is set to OFF mode.

2) Message

Item	Value	Explanation
DLEN	06H	
CMD	20H	
DATA	D0 D1 D2 D3	Current Note : Amplify 1000 times. For example, if the current is 50.45, the sending data is 50450. (int32_t)
	D4	GEAR Speed limit gear 00H: No gear 01H: Low speed 02H: Medium speed 03H: High speed 04H: Reverse

3) Response

Item	Value	Explanation
DLEN	1CH	
CMD	20H	
DATA	D0	ERROR CODE
	D1 D2	Battery Voltage Note: Enlarge 100 times (int16_t)
	D3 D4 D5 D6	Battery Current Note: Enlarge 1000000 times (int32_t)
	D7 D8 D9 D10	Motor Current Note: Enlarge 1000000 times (int32_t)
	D11 D12 D13 D14	Motor Erpm Note: (int32_t)
	D15 D16	Duty Note: Enlarge 10000 times (int16_t)
	D17 D18	MOSFET Temp Note: Enlarge 100 times (int16_t)
	D19 D20	Motor Temp Note: Enlarge 100 times (int16_t)
	D21 D22	CPU Utilization Note: Enlarge 10000 times (int16_t)
	D23 D24 D25 D26	Encoder Angle (unit : °) Note: Enlarge 1000000 times (int32_t)

3.1.14 UART_REBOOT_FTESC (23H)

1) Function

Reset the MCU.

2) Message

Item	Value	Explanation
DLEN	01H	
CMD	23H	
DATA	None	

3) Response

Note : This command is of no response.

3.1.15 UART_SET_ID_CURRENT (25H)

1) Function

Motor D-axis current control command.This command can lock the motor at zero phase.

Note : This command is only valid when the Input Signal Type is set to OFF mode.

2) Message

Item	Value	Explanation
DLEN	05H	
CMD	25H	
DATA	D0 D1 D2 D3	Motor D-axis current Note : Amplify 1000 times. For example, if the D axis current is 50.45, the sending data is 50450. (int32_t)

3) Response

Note : This command is of no response.

3.1.16 UART_SET_POSITION (26 H)

1) Function

Position control command.

Note : This command is only valid when the Input Signal Type is set to OFF mode. It can only be used in **Encoder Mode**.

2) Message

Item	Value	Explanation
DLEN	05H	
CMD	26H	
DATA	D0 D1 D2 D3	Position Note : Amplify 1000 times . (int32_t)

3) Response

Note : This command is of no response.

3.1.17 UART_SET_ SPEED (27H)

1) Function

Motor electronic speed control command.

Note : This command is only valid when the Input Signal Type is set to OFF mode.

2) Message

Item	Value	Explanation
DLEN	05H	
CMD	27H	
DATA	D0 D1 D2 D3	Erpm Control Note : (int32_t)

3) Response

Note : This command is of no response.

3. 2 Command/Response Operation Example (C language)

The following code uses the instruction UART_OBTAIN_DATA_ONCE (00H) as an example to illustrate the entire operation process.

3.2.1 ECU Sends Command (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 <= 25 5 ) {
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);
}

// Execute this function and send commands to the MCU
void Ftesc_Obtain_Realtime_Data(void)
{
```

```
uint8_t payload[1];
payload[0] = UART_OBTAIN_DATA_ONCE;
FTESC_Uart_packSendPayload(payload, 1); //Serial ports send function
}
```

3.2.2 ECU Receives the Response (00H) Data Packet Returned from MCU

```
// Split the signed short integer data (int16_t) into two bytes for transmission, with the high-
//order digit first and the low-order digit second
void valueDecompose_I16(uint8_t* dataNum, int16_t value, int 16 _t *numInd) {
dataNum[(*numInd)++] = value >> 8;
dataNum[(*numInd)++] = value;
}

// Split the unsigned short integer data (uint16_t) into two bytes for transmission, with the high
//-order digit first and the low-order digit second
void valueDecompose_U16(uint8_t* dataNum, uint16_t value, int 16 _t *numInd) {
dataNum[(*numInd)++] = value >> 8;
dataNum[(*numInd)++] = value;
}

// Split the signed long integer data (int32_t) into four bytes for transmission, with the high-
//order digit first and the low-order digit second
void valueDecompose_I32(uint8_t* dataNum, int32_t value, int 16 _t *numInd) {
dataNum[(*numInd)++] = value >> 24;
dataNum[(*numInd)++] = value >> 16;
dataNum[(*numInd)++] = value >> 8;
dataNum[(*numInd)++] = value;
}

// Split the unsigned long integer data (uint32_t) into four bytes for transmission, with the high
//-order digit first and the low-order digit second

void valueDecompose_U32(uint8_t* dataNum, uint32_t value, int 16 _t *numInd) {
dataNum[(*numInd)++] = value >> 24;
dataNum[(*numInd)++] = value >> 16;
dataNum[(*numInd)++] = value >> 8;
dataNum[(*numInd)++] = value;
}

// Combine two bytes into a signed short integer data (int16_t), with the high-order digit first
//and the low-order digit second

int16_t valueCompose_I16(const uint8_t *dataNum, int 16 _t *numInd) {
  int16_t retval = ((uint16_t) dataNum[*numInd]) << 8 |
```

```
        ((uint16_t) dataNum[*numInd + 1]);
  *numInd += 2;
  return retval;
}

// Combine two bytes into unsigned short integer data (uint16_t), with the high-order digit first
//and the low-order digit second
uint16_t valueCompose_U16(const uint8_t *dataNum, int 16 _t *numInd) {
  uint16_t retval =    ((uint16_t) dataNum[*numInd]) << 8 |
           ((uint16_t) dataNum[*numInd + 1]);
  *numInd += 2;
  return retval;
}

// Combine four bytes into a signed long integer data (int32_t), with the high-order digit first
and the low-order digit second
  int32_t valueCompose_I32(const uint8_t *dataNum, int 16 _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;
}

// Combine four bytes into unsigned long integer data (uint32_t), with the high-order digit first
//and the low-order digit second
uint32_t valueCompose_U32(const uint8_t *dataNum, int 16 _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;
}

// Convert float data into int16_t data by magnification factor and split it into two bytes for
transmission, with the high-order digit first and the low-order digit second
void valueDecompose_F16(uint8_t* dataNum, float value, float multiple, int32_t *numInd) {
    valueDecompose_I16(dataNum, (int16_t)(value * multiple), numInd);
}

// Convert float data into int32_t data by magnification factor and split it into four bytes for
transmission, with the high-order digit first and the low-order digit second
void valueDecompose_F32(uint8_t* dataNum, float value, float multiple, int32_t *numInd) {
    valueDecompose_I32(dataNum, (int32_t)(value * multiple), numInd);
}

// Combine two bytes into signed short integer data (int16_t), with the high-order digit first
and the low-order digit second, and convert them into float data by magnification factor
float valueCompose_F16(const uint8_t *dataNum, float multiple, int 16 _t *numInd) {
```

```
return (float)valueCompose_I16(dataNum, numInd) / multiple;
}

// Combine four bytes into signed short integer data (int32_t), with the high-order data first and
the low-order data second, and convert them into float data by magnification factor
float valueCompose_F32(const uint8_t *dataNum, float multiple, int 16 _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;

// The UART receive completion is triggered by an idle interrupt serial interface normally. The
uart_rx_buff is transferred via DMA.
// Once the idle interrupt is triggered, the data in the buffer can be parsed.
int8_t Ftesc_Uart_processReadPacket(uint8_t * uart_rx_buff)
{
  uint16 data_len = 0;
  uint8_t * data = 0;

  // Check if the frame header is correct
  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; // high 8 bits
    data_len |= (uint16_t)uart_rx_buff[2]; // low 8 bits
    data = uart_rx_buff + 3;
  } else {
    return -1; // The data packet format is abnormal, it must start with 0XAA or 0XBB
  }

  // Calculate the link layer data crc16 checksum and obtain the receiving crc16 checksum
  uint16 data_crc16_calc = crc16(data, data_len);
  uint16 data_crc16_rece = (uint16)data[data_len] << 8 | (uint16)data[data_len + 1];
```

```
    // Determine whether the CRC16 check is correct
    if (data_crc16_calc != data_crc16_rece) {
      return -2;            // Data packet CRC16 check failed
  }

// Check if the frame end is correct
if (data[data_len + 2] != 0XDD) { //Must end with 0XDD
    return -3;
  }

UART_COMMAND uart_cmd = (UART_COMMAND)data[0]; // Obtain command
uint16_t index = 0;
  data++; // offset to DATA area
  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, 10 000 .0f, &index );
     esc_data.encoder_angle = valueCompose_F32( data, 1000000.0f, &index );
    }
    break;

    default:
      break;
    }

    return 0; //Frame parsing completed
}
```