关节模组通讯协议V2.0

更新记录

版本	修改内容	日期	作者
v1.0	首版拟定	2025.4.27	
v1.1	数据包从28减小到24个字节	2025.5.6	
v1.2	串口波特率由19200提高到115200	2025.5.9	
v1.3	指令增加速度设置支持	2025.5.19	
v1.4	增加电流信息读取	2025.5.20	
v1.5	指令增加电流设置支持	2025.5.28	
v1.6	1波特率由115200提高到256000 2去掉TAIL字段，数据包从24字节将到20个字节；更改标志头；增加6个广播报文； 3校验改成CRC16_CCITT_FALSE	2025.6.7	
v1.7	1增加PID、通讯波特率、控制模式设置 2增加反馈信息包3读取	2025.6.18	
v1.8	1增加电流保护限制参数配置 2默认波特率由256000改成230400	2025.6.26	
v1.9	1速度基准值由40krpm改成50krpm(自固件v124版本起) 2指令中电流字段改成保留，该字段值不起作用，无法再通过该字段动态设置电流(自固件版本v120起)	2025.7.7	
v2.0	控制指令更改：使能EN增加CMD_SET_REF	2025.7.28	
v2.0.1	1更新伪代码示例 2增加一些电流保护限制配置项文字说明 3增加Gaia Tools上位机使用说明	2025.8.1	

v2.0.2	增加瞬爆配置项说明	2025.8.11	

1. 功能列表

功能列表		
序号	功能	描述
1	设置角度(位置控制)	
2	点动控制	
3	读取状态信息	
4	配置参数	

2. 协议内容

串口数据传输，**波特率**230400，**停止位**1，**数据位**8，**奇偶校验**无。

包速率推算：

$$\text{传输时间} = \frac{\text{总数据位数}}{\text{波特率}} = \frac{N \times (D+S)}{B}$$

$$\text{总数据位数} = N \times (D+S) = 20 \times (8+1) = 20 \times 9 = 180\text{位}$$

$$\text{传输时间} = \frac{180}{256000} \approx 0.00078125\text{秒} \approx 0.781\text{毫秒}$$

使用二进制数据通信，数据包结构如下，多字节为Intel格式：

协议数据包结构

```
typedef struct {
  uint32_t head;              //协议头
  uint32_t can_id;            //功能ID
  uint8_t can_data[8];        //数据段
  uint32_t crc;               //CRC16_CCITT_FALSE
}UART_Frame_t;
```

字段说明：

字段	内容	备注
head	协议标志头，固定0x1400AA55，对应字节序为byte[3]=14,byte[2]=00,byte[1]=AA,byte[0]=55	

can_id	功能ID，见can_id字段说明	
can_data[8]	数据字段，含义见功能ID说明	
crc	CRC校验值(校验字段can_id,can_data[8])， CRC16_CCITT_FALSE标准，参考代码见附录	占位还是4个字节，高2字节为零。

can_id功能字段列表：

can_id	功能	data内容	主机方向	备注
0x00000000 ~0x0000009F, 0x000000FF	角度和速度指令	角度，速度，电流	TX	
0x000000A0	配置/状态读取	设备ID，数据索引，详见2.3节	TX	
0x000000F0	点动(零点重置)	设备ID，点动方向	TX	
0x00010000 ~0x009FFF00	读取返回信息	数据	RX	

2.1 控制指令(0x00~0x9F,0xFF)

一包数据中包含位置、速度和电流3个给定值，以及使能控制4个字段，在位置控制模式下，三个值都生效，在速度模式下，位置不生效。

模式	~~电流~~	速度	位置
位置	~~✔~~	✔	✔
速度	~~✔~~	✔	

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
dev_id	RSVD	EN	~~cur_ref~~		spd_ref		pos_ref	

字段	内容	备注

pos_ref	目标角度，Q7格式：要设定的值(度)x128，支持负角度。
	示例(Intel)：
	int16_t ref_q7 = -10.123(度)x128
	data[0] = (uint8_t)ref_q7;
	data[1] = (uint8_t)(ref_q7 >> 8);
		不溢出的范围是-255度~255度

spd_ref	速度设置，Q15格式，0~32767表示0%速度到100%基准速度。基准速度为50000rpm(未经减速比)	建议最大速度20krpm，对应Q15值为(32767*20/50)=13106
~~cur_ref~~	~~电流设置，Q15格式，0~32767表示电流0~5.625A~~ 保留	~~根据固件版本不同，在执行端会有不同限制，目前固件限制0.8A(Q15=4660)，参考：~~ ~~0.100A =582~~ ~~0.125A =728~~ ~~0.200A=1165~~ ~~0.250A=1456~~ ~~0.300A=1747~~ ~~0.500A=2912~~
EN	使能控制： 1: CMD_ENABLE 开启控制，电机上电锁定 2: CMD_DISABLE 关闭控制，释放电机 3: CMD_SET_REF 设置给定值 其他：无操作	1. 每次上电后，第一次开启控制，编码器需要定位(约1秒)，请保持空载状态 2. 只有EN=3(SET_REF)时，ref字段才会生效
RSVD	保留，请置0	
dev_id	设备ID，范围0x00000001~0x0000009F，0x000000FF为广播指令，所有设备同时响应	

2.2 JOG点动指令(0xF0)

需要先使能控制(见控制指令)以后，点动才会响应。点动会同时重置零点到点动后的位置。

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000F0	RSVD1					dir	RSVD0	dev_id

字段	内容	备注
dev_id	设备ID	支持0xFF广播ID
RSVD0	保留	
dir	0：无动作 1：CW顺时针，+5度 2：CCW逆时针，-3度	1.实际点动手指角度变化与安装方式有关，请结合实际使用。 2.可以通过前进后退组合出任意位置。
RSVD1	保留	

指令示例，ID2点动正转：

55 AA 00 14 F0 00 00 00 02 00 01 00 00 00 00 00 EE 60 00 00

ID2点动反转：

55 AA 00 14 F0 00 00 00 02 00 02 00 00 00 00 00 0E AE 00 00

参照以上字段不难解读。

55 AA 00 14 为标志头，0E AE 00 00 为CRC16，下文示例中还会再次出现，将不再重复说明，重点关注can_id和data字段。

2.3 状态读取&参数配置(0xA0)

2.3.1 发送指令格式

先看示例，读取设备2的状态信息(index=1)：

55 AA 00 14 A0 00 00 00 02 01 00 00 00 00 00 00 3F 77 00 00

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0	value				RSVD		index	dev_id

字段	内容	备注
dev_id	设备ID	
index	要读取或配置的项索引，取值与含义见索引列表	具体是读取或配置由索引含义决定
RSVD	保留，请置0	
value	要写入配置项的值，格式见对应索引描述	对于读取功能索引无需该值，置0

index	配置指令名	内容/含义	备注
1	CONFIG_INFO_01_R	读取数据包1(运行状态、版本信息)	
2	CONFIG_INFO_02_R	读取数据包2(角度位置，电流值)	
4	CONFIG_SET_ID	设置ID	有回复(成功/失败)
7	CONFIG_CTRL_MODE_SW	控制模式切换(速度/位置模式)	无回复(如无殊说明,下同)
8	CONFIG_SET_POS_KP	位置环KP设置	
9	CONFIG_SET_POS_KD	位置环KD设置	
10	CONFIG_SET_SPD_KP	速度环KP设置	
11	CONFIG_SET_SPD_KI	速度环KI设置	
12	CONFIG_INFO_03_R	读取信息3(速度和位置PID参数)	
13	CONFIG_WRITE_FLASH	PID参数写入FLASH	有回复(成功/失败)
15	CONFIG_BUAD_RATE	5档通讯波特率选择	有回复(成功/失败)
16	CONFIG_POS_LPF_LV	位置平滑力度等级设置	
17	CONFIG_INFO_04_R	读取配置信息4(位置平滑力度、波特率)	
18	CONFIG_MAX_CUR	最大输出电流	
19	CONFIG_PROTECT_CUR_LIMIT	保护限制后的最大电流	
20	CONFIG_PROTECT_CUR_LV1	电流保护等级1阈值设定	
21	CONFIG_CUR_LV1_DELAY	电流保护等级1持续时间设定	
22	CONFIG_RECV_CUR	电流恢复阈值	
23	CONFIG_CUR_RECV_WIN	恢复判定时间窗口ms	
24	CONFIG_INFO_05_R	读取配置信息5(电流保护等级1、恢复时间)	
25	CONFIG_INFO_06_R	读取配置信息6(瞬爆冷却,瞬爆时长)	
26	CONFIG_COLD_TIME	瞬间爆发冷却时长	
27	CONFIG_PLUS_TIME	瞬间爆发时长	

回复数据格式，先看一个示例，读取设备2的状态信息(索引1，CONFIG_INFO_01_R)：

55 AA 00 14 00 02 01 00 00 00 75 4F 00 00 FA 05 AC 95 00 00

这里主要理解can_id字段的结构：

can_id				data[7]~data[0]
byte[3](高位)	**byte[2]**	**byte[1]**	**byte[0](低位)**	
0	index	dev_id	0	DATAS

字段	含义	备注
dev_id	回复信息的设备ID	
index	配置索引，与指令索引一致	
DATAS	数据字段，内容详见各个指令描述	

2.3.2 读取/返回数据1(CONFIG_INFO_01_R)

包含运行状态，故障码，固件版本信息。

读取发送格式：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0							1	dev_id

返回数据：

设备上电时会自动发送一次该数据包，方便用于识别在线设备，以及通信线路检查。

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
sta_id	Vbus		angle_fb		temp	soft_ver	err_code	fsm

sta_id			
byte[3]	byte[2]	byte[1]	byte[0]

0	index=1	dev_id	0

字段	内容	备注

fsm	运行状态：
	mcReady = 0, // 就绪
	mcAlign = 5, // 预定位
	mcRun = 7, // 运行
	mcFault = 9, // 故障
	1~4，6，8，10：保留	

err_code	故障码：
	FaultNoSource = 0, // 无故障
	FaultHardOVCurrent = 1, // 硬件过流
	FaultSoftOVCurrent = 2, // 软件过流
	FaultHardOverVoltage = 3, // 硬件过压
	FaultSoftOverVoltage = 4, // 软件过压
	FaultHardUnderVoltage = 5, // 硬件欠压
	FaultSoftUnderVoltage = 6, // 软件欠压
	FaultPhaseLost = 7, // 缺相
	FaultStall = 8, // 堵转
	FaultSoftOTErr = 9, // 软件过温
	FaultHardOTErr = 9, // 硬件过温
	FaultUartLost = 10, // 通信丢失
	FaultPOST = 11, // FCT自检故障	

soft_ver	软件版本，如103，解释为v1.0.3	
temp	MCU温度，单位摄氏度，偏移量50	如收到temp=110,那么表示温度是110-50=60°C
angle_fb	手指角度，Q7，Intel小端示例： angle_fb= (data[5] << 8 | data[4]) * 0.0078125(1/128);	单位度
Vbus	母线电压，Q7，单位V	
sta_01_id	dev_id:设备ID index=1(CONFIG_INFO_01_R)，与读取指令的索引一致	

2.3.3 读取/返回数据2(CONFIG_INFO_02_R)

包含电流、母线电压，角度信息。

读取发送格式：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0							2	dev_id

返回数据：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
sta_id	angle_fb		speed		Q_cur		D_cur	

sta_id			
byte[3]	byte[2]	byte[1]	byte[0]
0	index	dev_id	0

字段	内容	备注
D_cur	D轴电流，Q15，0~32767表示电流0~5.625A Intel小端示例： Dcur= (data[1] << 8 | data[0]);	相电流幅值： Is=sqrtf(D*D+Q*Q)
Q_cur	Q轴电流，Q15，0~32767表示电流0~5.625A	
speed	速度，Q15，0~32767表示0~40k的转速	
angle_fb	末端角度，Q7，Intel小端示例： angle_fb= (data[5] << 8 | data[4]) * 0.0078125(1/128);	
sta_id	dev_id:设备ID index=2(CONFIG_INFO_02_R)，与读取指令的索引一致	

2.3.4 设置ID(CONFIG_SET_ID)

发送格式，如前所示，这里再次展示：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0			Value_H	Value_L			index	dev_id

字段	含义	备注
dev_id	目标设备ID	
index	CONFIG_SET_ID(4)	
Value_L	新设备ID低8位	Q7格式
Value_H	新设备ID高8位	

例如设置ID为10：

代码块

```
uint16_t id_set = 10 << 7;    //to Q7
uint8_t Value_L = id_set ;
uint8_t Value_H = id_set >> 8;
```

回复内容：

can_id				data[7]~data[1]	data[0]
byte[3](高位)	**byte[2]**	**byte[1]**	**byte[0](低位)**		
0	index	dev_id	0	0	res
	4	新设备ID			0:失败 1:成功

参数立刻写入FLASH，无需再次发送CONFIG_WRITE_FLASH指令。

2.3.5 切换控制模式(CONFIG_CTRL_MODE_SW)

发送格式：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0				mode			index	dev_id

字段	内容	备注
dev_id	目标设备ID	
index	CONFIG_CTRL_MODE_SW(7)	
mode	0：无操作 1：速度模式 2：位置模式	

2.3.6 设置位置/速度PID(CONFIG_SET_POS_Kx)

发送格式：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0			Value_H	Value_L			index	dev_id

配置名	index	备注
CONFIG_SET_POS_KP	8	value格式Q12,范围0~32767(对应0~8.000)
CONFIG_SET_POS_KD	9	value格式Q12,范围0~32767(对应0~8.000)
CONFIG_SET_SPD_KP	10	value格式Q12,范围0~32767(对应0~8.000)
CONFIG_SET_SPD_KI	11	value格式Q15,范围0~32767(对应0~1.000)

更新之后立刻生效，无回复。

仅改变内存数据，下电丢失，如需永久保存请调试好后使用CONFIG_WRITE_FLASH指令。

2.3.7 读取/返回数据3(CONFIG_INFO_03_R)

数据包3的内容是位置的KP,KD,速度的KP,KI。

发送格式：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0							12	dev_id

返回数据：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
sta_id	SPD_KI(Q15)		SPD_KP(Q12)		POS_KD(Q12)		POS_KP(Q12)	

sta_id			
byte[3]	byte[2]	byte[1]	byte[0]
0	12	dev_id	0

返回数据示例：

55 AA 00 14 00 02 0C 00 FF 03 00 00 99 01 99 01 73 3F 00 00

POS_KP(Q12) = 0xFF | (0x03 << 8) = 0x03FF = 1023 (/ 4096 = 0.245)

2.3.8 参数保存(CONFIG_WRITE_FLASH)

发送格式：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0				KEY			index	dev_id

字段	内容	备注
dev_id	目标设备ID	
index	CONFIG_WRITE_FLASH(13)	
KEY	0xA5	

回复内容：

can_id				data[7]~data[1]	data[0]
byte[3](高位)	**byte[2]**	**byte[1]**	**byte[0](低位)**		
0	index	dev_id	0	0	res
	13	设备ID			0:失败 1:成功

2.3.9 设置通讯波特率(CONFIG_BUAD_RATE)

发送格式：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0				level			index	dev_id

字段	内容	备注
dev_id	目标设备ID	
index	CONFIG_BUAD_RATE(15)	
level	0: 500000 1: 230400 2: 115200 3: 38400 4: 19200	默认波特率为230400，多机控制时不建议降低的波特率。

回复内容：

can_id				data[7]~data[1]	data[0]
byte[3](高位)	**byte[2]**	**byte[1]**	**byte[0](低位)**		
0	index	dev_id	0	0	res
	15	设备ID			0:失败 1:成功

2.3.10 位置平滑力度等级(CONFIG_POS_LPF_LV)

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0				level			index	dev_id

字段	内容	备注
Level	0~5	数值越大平滑效果越大

2.3.11 读取配置信息4(CONFIG_INFO_04_R)

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0							index=17	dev_id

返回数据：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
sta_id	protect_limit		max_current		保留	param_ver	baud_level	lpf_level

sta_id			
byte[3]	byte[2]	byte[1]	byte[0]
0	17	dev_id	0

字段	内容	备注
lpf_level	0~5，当前位置平滑等级	数值越大平滑效果越大
baud_level	0~4，当前波特率等级	

param_ver	参数版本号	
max_current	最大输出电流，0~32767，Q15格式	电流基准值5.625A，下同
protect_limit	最大输出电流，0~32767，Q15格式	

2.3.12 最大输出电流设定(CONFIG_MAX_CUR)

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0			valueQ				index=18	dev_id

字段	内容	备注
index	CONFIG_MAX_CUR(18)	
valueQ	设定的电流值，0~32767，Q15格式。	基准值为5.625A。例如要设定为0.5A，则 valueQ=0.5/5.625*32767

2.3.13 保护限制电流设定(CONFIG_PROTECT_CUR_LIMIT)

触发保护阈值后限制输出不超过该电流。

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0			valueQ				index	dev_id

字段	内容	备注
index	CONFIG_MAX_CUR(19)	
valueQ	设定的电流值，0~32767，Q15格式。	

2.3.14 保护电流阈值设定(CONFIG_PROTECT_CUR_LV1)

当电流超过该值并持续一定时间后，触发限制电流保护。

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0			valueQ				index	dev_id

字段	内容	备注
index	CONFIG_PROTECT_CUR_LV1(20)	
valueQ	设定的电流值，0~32767，Q15格式。	

2.3.15 保护电流允许时间设定(CONFIG_CUR_LV1_DELAY)

允许电流超过保护阈值后的持续时间，超过该时间后触发电流限制保护。

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0			value				index	dev_id

字段	内容	备注
index	CONFIG_CUR_LV1_DELAY(21)	
valueQ	设定的时间，0~65000毫秒	

2.3.16 保护恢复阈值设定(CONFIG_RECV_CUR)

当电流小于该值且持续一定**时间(恢复时间窗)**，自动退出电流保护限制。

注：退出保护还有一种方式，给定新的指令后立即恢复。

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0			valueQ				index	dev_id

字段	内容	备注
index	CONFIG_RECV_CUR(22)	
valueQ	设定的电流值，0~32767，Q15格式。	

2.3.17 保护恢复窗口时间设定(CONFIG_CUR_RECV_WIN)

当电流小于**恢复值**且持续该时间，自动退出电流保护限制。

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0			valueQ				index	dev_id

字段	内容	备注
index	CONFIG_RECV_CUR(22)	
valueQ	设定的时间，0~65000毫秒	

2.3.18 读取配置信息5(CONFIG_INFO_05_R)

保护阈值相关：电流保护等级1、恢复时间。

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0							index=24	dev_id

返回数据：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]

sta_id	recover_delay	recv_cur	protect_delay_lv1	protect_cur_lv1

sta_id			
byte[3]	byte[2]	byte[1]	byte[0]
0	index=24	dev_id	0

字段	内容	备注
protect_cur_lv1	电流保护等级1阈值Q15	例如byte[0]=0x60，byte[1]=0x0B，protect_cur_lv1 = 0x0B60 = 2912(/32767*5.625=0.5A)
protect_delay_lv1	电流保护等级1持续时间	
recv_cur	电流恢复阈值Q15	
recover_delay	恢复判定时间窗口ms	

2.3.19 读取配置信息6(CONFIG_INFO_06_R)

读取瞬间爆发相关参数：瞬爆冷却时长，瞬爆时长。

瞬间爆发设计用于堵转之后的尝试恢复，处于堵转状态下在指令改变时触发爆发电流。

发送格式

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x000000A0							index=25	dev_id

返回数据：

can_id	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
sta_id	保留(0)						plus_t	cold_dt

sta_id			

byte[3]	byte[2]	byte[1]	byte[0]
0	index=25	dev_id	0

字段	内容	备注
cold_dt	瞬间爆发冷却时长	单位25ms
plus_t	瞬间爆发时长	单位25ms

2.4 特殊广播

特殊广播设计用于快速同步控制ID1~ID16的设备。

特殊广播数据结构

```
typedef struct {
  uint16_t head2;              //标志头
  int16_t value[8];           //数据段
  uint16_t crc16;              //CRC16_CCITT_FALSE
}UART_Frame2_t;
```

字段	内容	备注

head2	标志头：
	BOARDCASE_POS01_08_HEAD1 = 0xAB55;//角度
	BOARDCASE_POS09_15_HEAD2 = 0xAC55;//角度
	BOARDCASE_SPD01_08_HEAD3 = 0xAD55;//速度
	BOARDCASE_SPD09_16_HEAD4 = 0xAE55;//速度
	BOARDCASE_EN01_16_HEAD7 = 0xB155;//使能	

value[x]	指令数据。根据标志头区分，x=[0,7]对应设备ID1~8或9~16的指令数据。数据格式如下：	1. 角度范围-255度~255度

	角度设置：Q7格式：要设定的值(度)x128，支持负角度。 速度设置：Q15格式，0~32767表示0%速度到100%基准速度。	2. 基准速度参见控制指令章节
value[y]	对于使能广播(0xB155)，value解释为uint8_t value[16]，y=[0,15]，对应设备ID1~16。 value[y]=1: 使能控制，电机上电锁定 value[y]=2: 关闭控制，释放电机 value[y]=其他: 无操作	每次上电后，第一次开启控制，编码器需要定位(约2秒)，请保持空载状态
crc16	CRC16_CCITT_FALSE	

数据示例，广播设置ID1~ID8位置90度：

55 AB 00 2D 00 2D 00 2D 00 2D 00 2D 00 2D 00 2D B6 67

90 * 128 = 11520 = 0x2D00

广播设置ID9~ID16位置-90度：

55 AC 00 D3 00 D3 00 D3 00 D3 00 D3 00 D3 00 D3 B0 76

-90 * 128 = -11520 = 0xD300

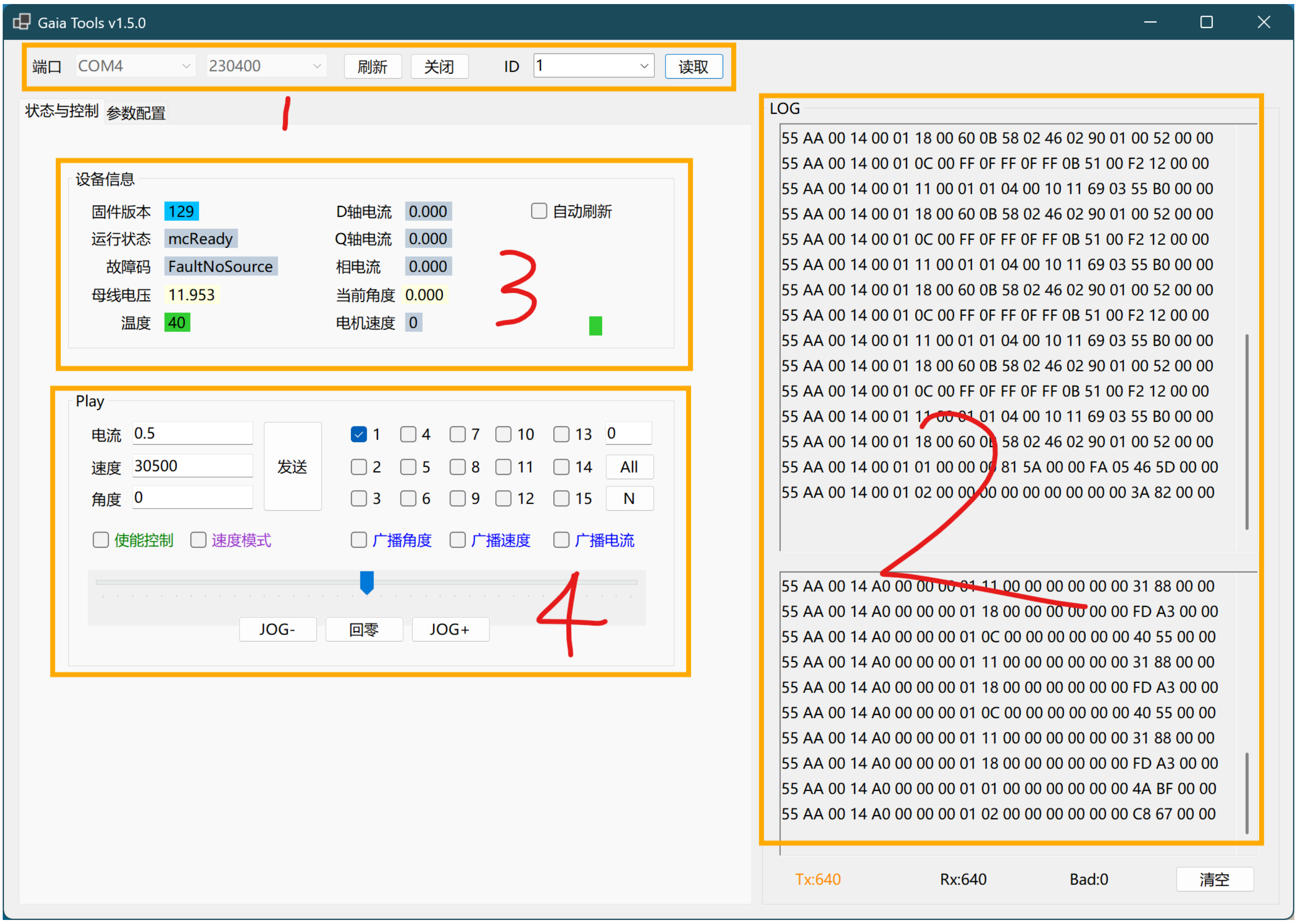
广播使能：

55 B1 00 14 B1 00 00 00 FF 01 00 00 00 00 00 00 F3 58 00 00

3. 上位机说明与指令样例

GaiaTools_v151.zip

3.1 UI介绍



1区：串口连接。默认波特率230400。读取按钮用于确认设备连接状态，返回的数据显示在3区中。

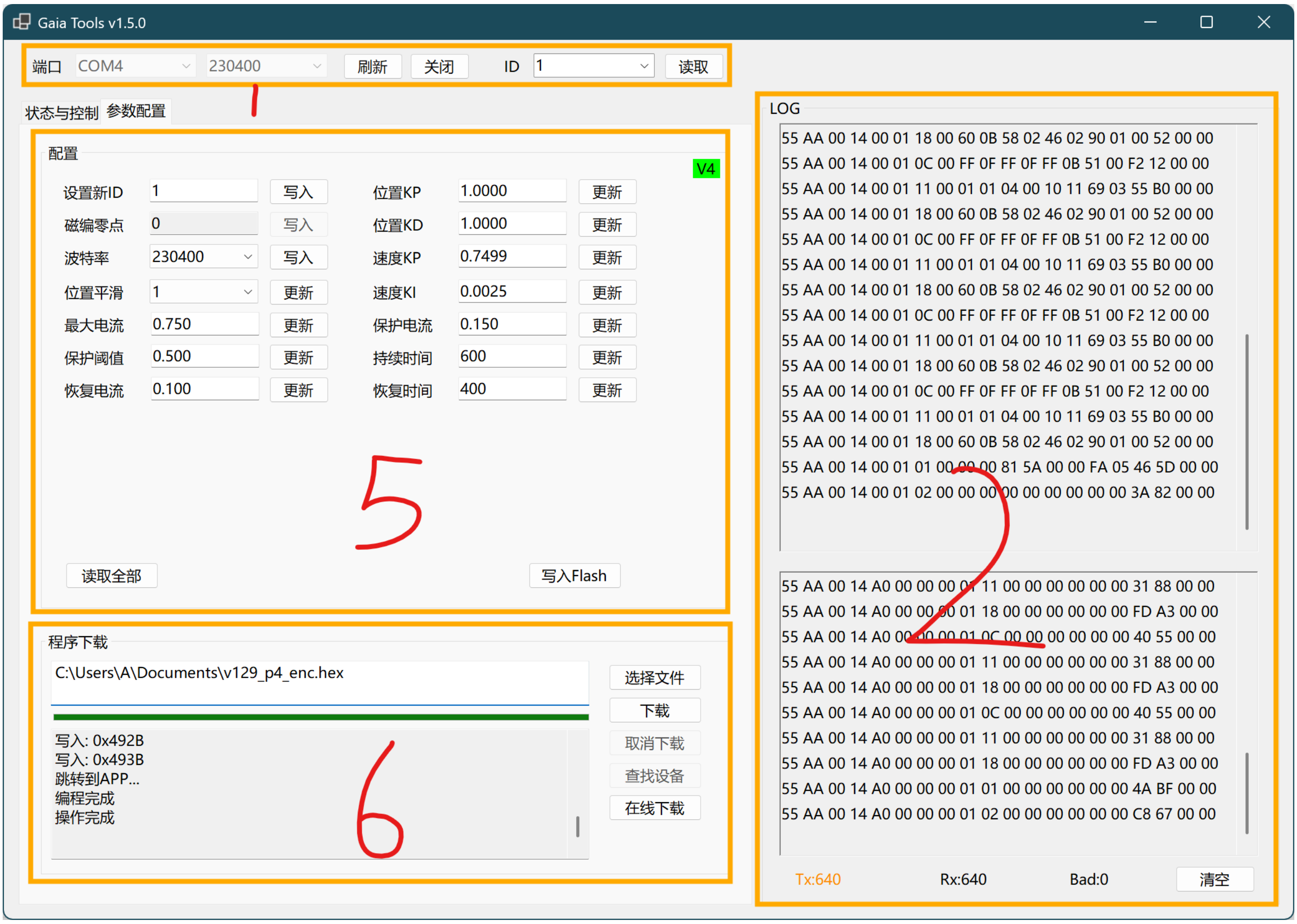
2区：收发通讯数据包原始数据。上方为接收，下方为发送(若串口未打开时显示为红字)。可以作为样例参考。

3区：状态信息显示。手动点击“读取”可以获取最新信息，或者勾选自动刷新，以3Hz的频率自动读取。(对应的设备ID为1区中的ID)

4区：Play试玩。

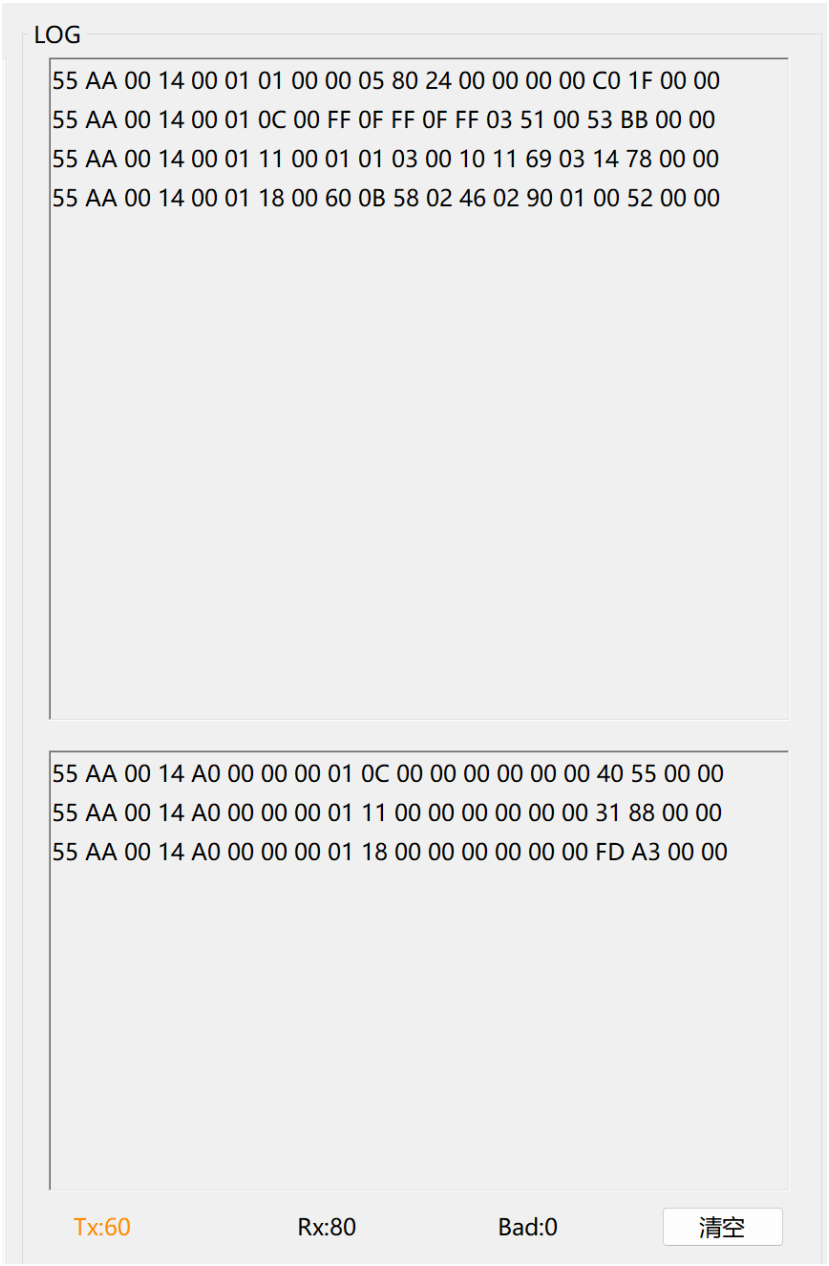
5区：参数配置区，对应的设备ID为1区中的ID。

6区：程序升级。选择好HEX文件后，确认ID无误，点击"在线下载"。



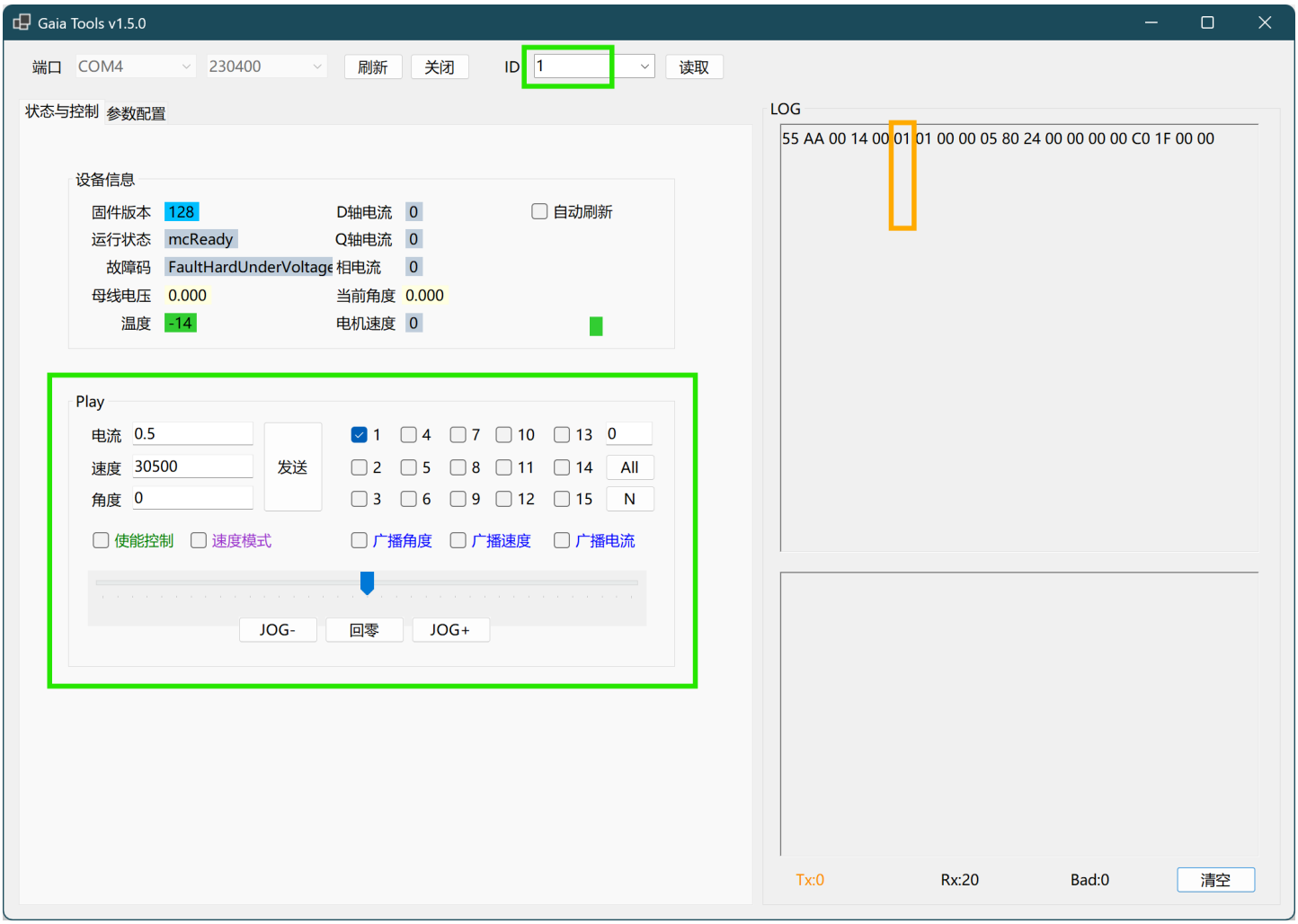
3.2 指令样例

可以通过操作上位机生成指令，无需打开串口也可以生成指令，上位机的每一次操作都会有输出，**2区**的LOG窗口上方为接收，下方为发送，方便参考。



（注：每包数据均为20个字节，如果收发字节统计不是20的整数倍，说明通讯有受到过干扰。）

3.3 基本控制



步骤1，先确定在线设备ID和通讯状态，在连接好串口后，上位机先打开串口，然后再上电，设备在上电后会发出一帧数据，如上图黄框中的0x01即是设备ID，ID框选择对应ID之后，点读取按钮，如果设备有回复，那说明通讯正常。(如果上电后没有报文数据，请检查串口线和供电是否正常)

注：如果同时连接多个设备，请确保ID不重复，否则需要先单独改ID再连接到一起。

步骤2，勾选需要控制的ID。点击“使能控制”，之后拖动滑块即可控制，也可手动输入角度，点击发送。（上电默认位置控制，勾选“速度模式”可以切换到速度控制模式，模式不保存，下电恢复）

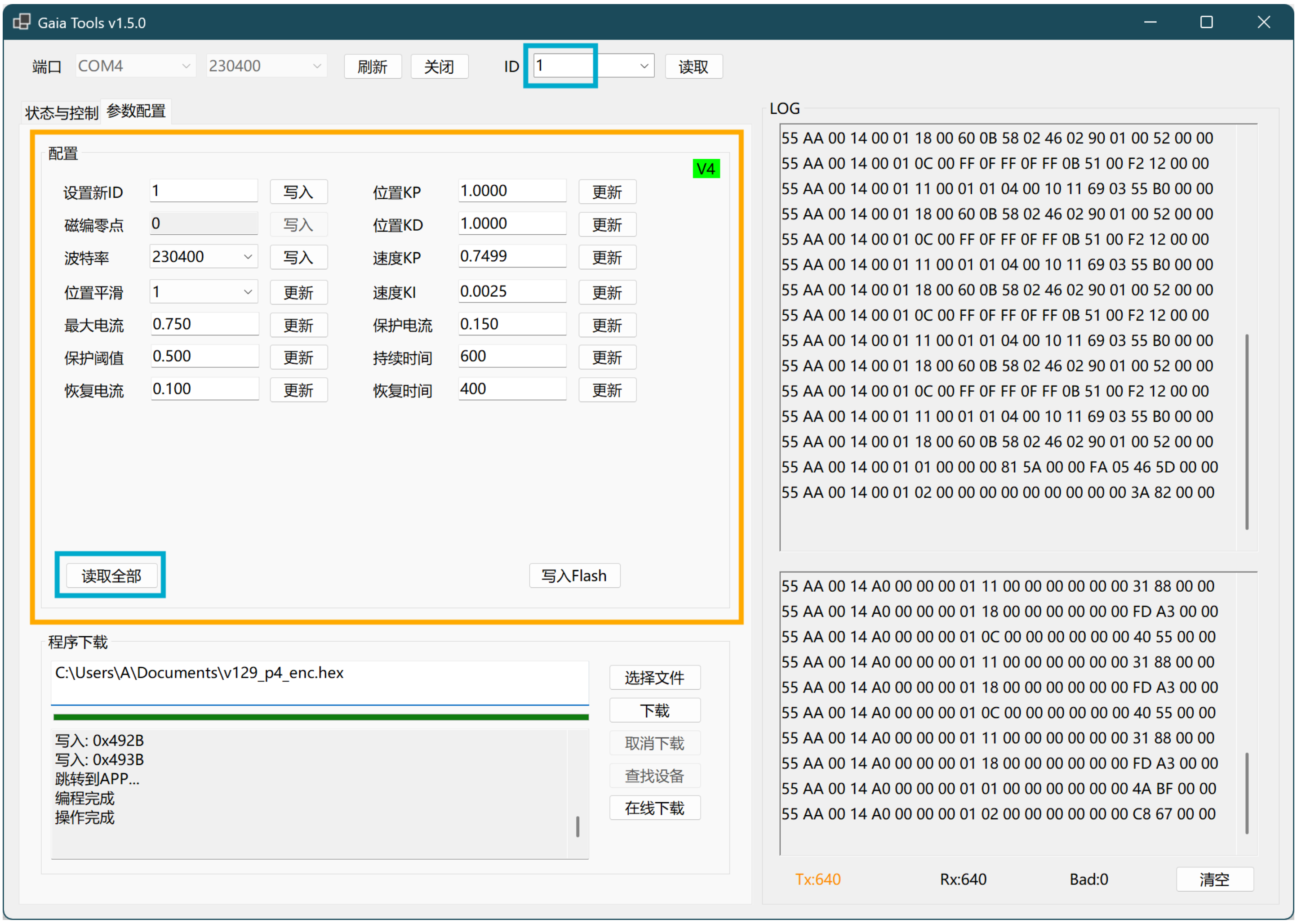
点动：位置模式下，JOG-和JOG+，微调位置同时使零点更新到当前位置。

3.4 参数配置-ID更改

对应的设备ID为1区中的ID。

”写入“字样按钮的点击后立即写入Flash，永久保存。

”更新“字样按钮点击后参数立即生效，下电丢失，如需永久保存，请在设置完所有参数后点击"写入Flash”按钮。



ID更改：在“设置新ID”框中输入新ID(十进制)，点击写入，即可完成ID修改。立即生效，掉电保存。(新ID不能与原ID相同)

4. 示例代码片段

```c
#define TARGET_DEVICE_ID        (0x01)          //设备ID(也即指令功能ID)
#define DEVICE_BOARDCASE_ID     (0xFF)          //广播ID(所有设备对该ID均响应)
#define JOG_ID                  (0xF0)          //点动功能ID
#define CONFIG_FUNC_ID          (0xA0)          //信息查询功能ID

/*
 * BYTE[6]CMD列表
 */
#define CMD_NONE                (0)             //空操作
#define CMD_ENABLE              (1)             //使能
#define CMD_DISABLE             (2)             //下使能
#define CMD_SET_REF             (3)             //设置目标值

/*
 * 配置与状态读取功能ID(0xA0)支持的配置项
 */
#define CONFIG_REBOOT           (0)             //复位请求
#define CONFIG_INFO_01_R        (1)             //读取状态信息1
#define CONFIG_INFO_02_R        (2)             //读取状态信息2(位置、电流、母线电压)
#define CONFIG_UPDATE_FIN_ZERO  (3)             //末端角度编码器零点
#define CONFIG_SET_ID           (4)             //设置设备ID
#define CONFIG_UPDATE_ENC_ZERO  (5)             //转子角度编码器零点
#define CONFIG_COMM_SILENT      (6)             //通讯静默(除本指令外不响应任何指令)
#define CONFIG_CTRL_MODE_SW     (7)             //控制模式切换(2位置/1速度)
#define CONFIG_SET_POS_KP       (8)             //位置环KP
#define CONFIG_SET_POS_KD       (9)             //位置环KD
#define CONFIG_SET_SPD_KP       (10)            //速度环KP
#define CONFIG_SET_SPD_KI       (11)            //速度环KI
#define CONFIG_INFO_03_R        (12)            //读取配置信息3(速度PI,位置PD参数)
#define CONFIG_WRITE_FLASH      (13)            //参数写入FLASH(参数存储)
#define CONFIG_READ_FLASH       (14)            //参数读取FLASH(参数存储)
#define CONFIG_BUAD_RATE        (15)            //波特率设置(0:256000,1:115200,2:38400,3:19200,4:9600)
#define CONFIG_POS_LPF_LV       (16)            //位置指令平滑力度0~5
#define CONFIG_INFO_04_R        (17)            //读取配置信息4(位置平滑力度、波特率)
#define CONFIG_MAX_CUR          (18)            //最大输出电流
#define CONFIG_PROTECT_CUR_LIMIT        (19)    //保护限制后的最大电流
#define CONFIG_PROTECT_CUR_LV1  (20)            //电流保护等级1阈值设定
#define CONFIG_CUR_LV1_DELAY    (21)            //电流保护等级1持续时间设定
#define CONFIG_RECV_CUR         (22)            //电流恢复阈值
#define CONFIG_CUR_RECV_WIN     (23)            //恢复判定时间窗口ms
```

```c
#define CONFIG_INFO_05_R        (24)                    //读取配置信息5(电流保护等级1、恢复时间)
#define CONFIG_INFO_06_R        (25)                    //读取配置信息6(瞬爆冷却,瞬爆时长)
#define CONFIG_COLD_TIME        (26)                    //瞬爆冷却
#define CONFIG_PLUS_TIME        (27)                    //瞬爆时长

//JOG dir
#define JOG_CW                  (1)                 //顺时针
#define JOG_CCW                 (2)                 //逆时针

/*
 * 协议标志头尾(注意大小端)
 */
#define UART_PACKGE_HEAD                (0x1400AA55)

typedef struct {
  uint32_t head;              //固定0x1400AA55(byte[0]=0x55)
  uint32_t can_id;            //功能ID
  uint8_t can_data[8];        //数据段
  uint32_t crc;               //CRC16_CCITT_FALSE
}UART_Frame_t;

UART_Frame_t tx_frame = {
        .head = UART_PACKGE_HEAD,
};

UART_Frame_t rx_frame;

void set_enable(UART_Frame_t* p, bool en)
{
    int16_t pos_ref = 0;
    uint16_t spd_ref = 0;
    uint16_t cur_limit = 0;     //保留
    uint8_t cmd = en ? CMD_ENABLE : CMD_DISABLE;

    p->can_id = TARGET_DEVICE_ID;
    p->can_data[0] = 0;
    p->can_data[1] = 0;
    p->can_data[2] = 0;
    p->can_data[3] = 0;
    p->can_data[4] = 0;
    p->can_data[5] = 0;
    p->can_data[6] = cmd;
    p->crc = CRC16_CCITT_FALSE((uint8_t*)&p->can_id,
        sizeof(UART_Frame_t) - sizeof(p->head) - sizeof(p->crc));
    UART_Send((uint8_t*)p, sizeof(UART_Frame_t));
```

```c
}

/*
 * 设置角度
 */
void set_angle(UART_Frame_t* p)
{
    int16_t pos_ref;
    uint16_t spd_ref = 16384;    //Q15,0~32767对应0%~100%基准速度(50000rpm)
    uint16_t cur_limit = 0;     //保留
    uint8_t cmd = CMD_SET_REF;

    pos_ref = 90.0f * 128;
    p->can_id = TARGET_DEVICE_ID;
    p->can_data[0] = (uint8_t)pos_ref;
    p->can_data[1] = (uint8_t)(pos_ref >> 8);
    p->can_data[2] = (uint8_t)spd_ref;
    p->can_data[3] = (uint8_t)(spd_ref >> 8);
    p->can_data[4] = (uint8_t)cur_limit;
    p->can_data[5] = (uint8_t)(cur_limit >> 8);
    p->can_data[6] = cmd;
    p->crc = CRC16_CCITT_FALSE((uint8_t*)&p->can_id,
        sizeof(UART_Frame_t) - sizeof(p->head) - sizeof(p->crc));
    UART_Send((uint8_t*)p, sizeof(UART_Frame_t));
}

/*
 * 读取状态信息
 */
void get_status(UART_Frame_t* p)
{
    p->can_id = CONFIG_FUNC_ID;
    p->can_data[0] = TARGET_DEVICE_ID;
    p->can_data[1] = CONFIG_INFO_01_R;      //index=1
    p->crc = CRC16_CCITT_FALSE((uint8_t*)&p->can_id,
        sizeof(UART_Frame_t) - sizeof(p->head) - sizeof(p->crc));
    UART_Send((uint8_t*)p, sizeof(UART_Frame_t));

    //串口接收数据
    UART_Recv((uint8_t*)&rx_frame, sizeof(UART_Frame_t));
    int angle = (rx_frame->can_data[5] << 8 | rx_frame->can_data[4]) >> 7;
    printf("DEV:%2d,FSM:%3d,ERROR_CODE:%2d,ANGLE:%5.2d\n",
            rx_frame->can_id;
            rx_frame->can_data[0],
            rx_frame->can_data[1],
            angle);
}
```

```
133
134 /*
135  * 点动
136  */
137 void do_jog(UART_Frame_t* p, bool dir)
138 {
139     p->can_id = JOG_ID;
140     p->can_data[0] = TARGET_DEVICE_ID;
141     p->can_data[2] = dir ? JOG_CW : JOG_CCW;
142     p->crc = CRC16_CCITT_FALSE((uint8_t*)&p->can_id,
143         sizeof(UART_Frame_t) - sizeof(p->head) - sizeof(p->crc));
144     UART_Send((uint8_t*)p, sizeof(UART_Frame_t));
145 }
146
147 // CRC16_CCITT_FALSE
148 uint16_t CRC16_CCITT_FALSE(uint8_t data[], int offset, int length)
149 {
150     const uint16_t polynomial = 0x1021;
151     uint16_t crc = 0xFFFF;
152     for (int i = offset; i < offset + length; i++)
153     {
154         crc ^= (uint16_t)(data[i] << 8);
155         for (int j = 0; j < 8; j++)
156         {
157             if ((crc & 0x8000) != 0)
158                 crc = (uint16_t )((crc << 1) ^ polynomial);
159             else
160                 crc <<= 1;
161         }
162     }
163     return crc;
164 }
165
166
167 void task_test()
168 {
169     //get_status(&tx_frame);
170     set_enable(&tx_frame, 1);
171     sleep(3000);
172     set_angle(&tx_frame);
173     //do_jog(&tx_frame, 1);
174 }
```

5. END