

1. 协议解析

1.1 CAN 相关说明

1. CAN 波特率:

- 仲裁段: 1 Mbps
- 数据段: 1Mbps。

2. ID: 由 16 位构成, 其中 0x7F 是广播地址。

- 高 8 位: 表示**源地址**:
 - 最高位为 1: 需要回复, 相当于一个**总开关**, 开启但不发生查询指令会返回一个数据段长度为 0 的帧。
 - 最高位为 0: 无需回复。
 - 其余 7 位: 信号源地址。
- 低 8 位: 表示**目的地址**:
 - 最高为 0。
 - 其余 7 位表示目的地址。

例如:

1. ID: 0x8001

- 信号源地址为 0。
- 目的地址为 1。
- 最高位为 1, 表示需要回复, 即打开回复**总开关**。

2. ID: 0x100

- 信号源地址为 1。
- 目的地址为 0。
- 最高位为 0, 表示无需回复, 即关闭回复**总开关**。

1.2 模式说明

1.2.1 普通模式 (位置和速度不能同时控制)

```
uint8_t cmd[] = {0x07, 0x07, pos1, pos2, val1, val2, tqe1, tqe2};
```

- 普通协议由: 指令位 (2 字节) + 位置 (2 字节) + 速度 (2 字节) + 力矩 (2 字节) 共 8 字节构成。
- 0x07 0x07: 普通模式, 可控制速度和力矩、位置和力矩 (见[[#2.1 普通模式]])。
- 协议中**位置、速度、力矩**数据都为**小端模式**, 即低字节先发, 高字节后发,
 - 如 pos = 0x1234 中, pos1 = 0x34, pos2 = 0x12。
- 此模式可分为**两种**控制方式:
 - 位置、力矩控制 (此时 val=0x8000, 表示无限制)。
 - 速度、力矩控制 (此时 pos=0x8000, 表示无限制)。

1.2.2 力矩模式

```
uint8_t cmd[] = {0x05, 0x13, tqe1, tqe2};
```

- 力矩模式协议由：指令位（2 字节）+力矩（2 字节）。
- 0x05 0x13：纯力矩模式，后面接两字节的力矩数据。（见 [[#2.3 力矩模式]]）。
- 协议中力矩数据为小端模式，即低字节先发，高字节后发。
 - 如 tqe = 0x1234 中，tqe1 = 0x34，tqe2 = 0x12。

1.2.3 协同控制模式（位置、速度、力矩可以同时控制）

```
uint8_t cmd[] = {0x07, 0x35, val1, val2, tqe1, tqe2, pos1, pos2};
```

- 协同控制模式协议：指令位（2 字节）+速度（2 字节）+力矩（2 字节）+位置（2 字节）共 8 字节构成。
- 0x07 0x35：协同控制模式，已指定速度转动到指定位置，并限制最大力矩。
- 此模式中给如参数 0x8000 表示**无限制**（无限制的速度和力矩即为最大值）。
 - 如 val = 5000, tqe = 1000, pos = 0x8000：表示电机以 0.5 转/秒的转速一直转动，最大力矩为 0.1NM。
- 协议中位置、速度、力矩数据都为小端模式，即低字节先发，高字节后发，
 - 如 pos = 0x1234 中，pos1 = 0x34，pos2 = 0x12。

1.3 电机状态数据读取

1. 读取电机状态部分的协议和 CAN-FD 中的协议是一样的，唯一的区别是 CAN 受到 8 字节数据段的限制。
2. 寄存器地址和功能说明请查看**寄存器功能、电机运行模式、报错代码说明.xlsx**文件。
3. 由于 CAN 受 8 字节数据段限制，一帧 CAN 最多返回的电机信息有限：
 1. 一个寄存器的 float 类型或 int32_t 的电机信息。
 2. 3 个地址连续的 int16_t 类型电机信息。
 3. 6 个地址连续的 int8_t 类型的电机信息。
4. 例程中提供了 int16_t 的查询电机位置、速度、力矩信息的示例函数和电机信息解析（例程中使用的是 C 语言的共用体直接复制了 CAN 中第 3 到第 8 字节的数据）。

1.3.1 发送协议说明

```
uint8_t tdata[] = {cmd, addr, cmd1, addr1, cmd2, add2};
```

大致含义为：从 addr 读取 cmd[0, 1] 个 cmd[3, 2] 类型的数据。

- cmd：
 - 高四位 [7, 4]：0001 表示读取。
 - 2~3 位 [3, 2]：表示类型。
 - 00：int8_t 类型。
 - 01：int16_t 类型。
 - 10：int32_t 类型。

- 11: float 类型。
 - 低 2 位 [1, 0]: 表示数量。
 - 01: 一个数据。
 - 10: 两个数据。
 - 11: 三个数据。
 - addr: 开始获取的地址。
- 可以将多个 cmd , addr 拼接在一起, 一次性读取地址不连续和不同类型的数据。

1.3.2 接受协议说明

- 假设获取的数据是 uint16_t。

```
uint8_t rdata[] = {cmd, addr, a1, a2, b1, b2, ..., cmd1, addr1, c1, c2, c3, c4}
```

- cmd:
 - 高四位 [7, 4]: 0010 表示回复。
 - 2~3 位 [3, 2]: 表示类型。
 - 00: int8_t 类型。
 - 01: int16_t 类型。
 - 10: int32_t 类型。
 - 11: float 类型。
 - 低 2 位 [1, 0]: 表示数量。
 - 01: 一个数据。
 - 10: 两个数据。
 - 11: 三个数据。
- addr: 开始获取的地址。
- a1, a2: 数据 1, 小端模式。
- b1, b2: 数据 2, 小端模式。

1.3.3 示例

- 我们需要读取位置、速度和扭矩数据。
- 从寄存器 excel 表中可知: 位置、速度和转矩的数据地址分别为: 01, 02, 03。
- 由此可知, 我们可以从地址 01 开始连读 3 个数据, 考虑到 CAN 一次最大传输 8 字节的数据, 而 cmd + addr 占两个字节, 所以数据类型最多可以选择 int16_t 类型。
- 由上可知 cmd 的二进制为: 0001 0111, 十六进制为: 0x17。
- 需要从地址 01 开始读取, 故 addr 为 0x01。
- 需要发送的总数据为 uint8_t tdata[] = {0x17, 0x01}。

示例代码如下:

```

/**
 * @brief 读取电机
 * @param id
 */
void motor_read(uint8_t id)
{
    static uint8_t tdata[8] = {0x17, 0x01};

    CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
}

```

```
uint8_t cmd[] = {0x17, 0x01};
```

整体含义是：从地址 0x01 处开始，读取 3 个 int16_t 的寄存器（查表可知，地址 0x01~0x03 的寄存器分别表示位置、速度和力矩），故此命令为查询电机的位置、速度、力矩信息。

- 0x17：
 - 0x17[7:4] 的二进制为 0001：表示读。
 - 0x17[3:2] 的二进制为 01：表示数据类型为 int16_t。
 - 0x17[1:0] 的二进制为 11：表示数据个数为 3。
- 0x01：
 - 从 0x01 地址开始。

相应接受数据示例：

```
uint8_t rdata[] = {0x27, 0x01, 0x38, 0xf6, 0x09, 0x00, 0x00, 0x00};
```

- 0x27：对应发送的 0x17。
- 0x01：从地址 0x01 开始。
- 0x38 0xf6：位置数据：0xf638，即 -2505。
- 0x09 0x00：速度数据：0x0009，即 9。
- 0x00 0x00：力矩数据：0x0000，即 0。

1.4 电机停止

说明：

1. 使电机停止。
2. 对上位机指令 d stop

```

/**
 * @brief 电机停止
 */
void motor_stop(uint8_t id)
{
    uint8_t tdata[] = {0x01, 0x00, 0x00};

    CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
}

```

1.5 重置电机零位

说明:

1. 将当前位置设为电机零位。
2. 此指令只是在 RAM 中修改，还需配合 `conf write` 指令保存到 flash 中。
3. 建议使用此指令后 1s 左右再发送 `conf write` 指令。

```
void rezero_pos(uint8_t id)
{
    uint8_t tdata[] = {0x40, 0x01, 0x04, 0x64, 0x20, 0x63, 0x0a};

    CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
    HAL_Delay(1000); // 建议延时1s

    conf_write(id); // 保存设置
}
```

1.6 保存电机设置 (conf write)

说明:

1. 将电机 RAM 中设置保存到 flash 中。
2. 使用此指令后建议给电机重新上电。

```
void conf_write(uint8_t id)
{
    uint8_t tdata[] = {0x05, 0xb3, 0x02, 0x00, 0x00};

    CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
}
```

1.7 周期返回电机状态数据

说明:

1. 周期返回电机位置、速度、力矩数据。
2. 返回数据格式和使用 `0x17, 0x01` 指令获取的格式一样
3. 周期单位为 ms。
4. 最小周期为 1ms。
5. 如需停止周期返回数据，将周期给 0 即可，或者给电机断电。

```
void timed_return_motor_status(uint8_t id, int16_t t_ms)
{
    uint8_t tdata[] = {0x05, 0xb4, 0x02, 0x00, 0x00};

    *(int16_t *)&tdata[3] = t_ms;

    CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
}
```

2. 示例函数

2.1 普通模式

2.1.1 位置控制

```
/**
 * @brief 位置控制
 * @param id 电机ID
 * @param pos 位置: 单位 0.0001 圈, 如 pos = 5000 表示转到 0.5 圈的位置。
 * @param tqe 力矩
 */
void motor_control_Pos(uint8_t id,int32_t pos,int16_t tqe)
{
    uint8_t tdata[8] = {0x07, 0x07, 0x0A, 0x05, 0x00, 0x00, 0x80, 0x00};

    *(int16_t *)&tdata[2] = pos;
    *(int16_t *)&tdata[6] = tqe;

    uint32_t ext_id = (0x8000 | id);
    CAN_Send_Msg(ext_id, tdata, 8);
}
```

2.1.2 速度控制

```
/**
 * @brief 速度控制
 * @param id 电机ID
 * @param vel 速度: 单位 0.00025 转/秒, 如 val = 1000 表示 0.25 转/秒
 * @param tqe 力矩
 */
void motor_control_Vel(uint8_t id,int16_t vel,int16_t tqe)
{
    uint8_t tdata[8] = {0x07, 0x07, 0x00, 0x80, 0x20, 0x00, 0x80, 0x00};

    *(int16_t *)&tdata[4] = vel;
    *(int16_t *)&tdata[6] = tqe;

    uint32_t ext_id = (0x8000 | id);
    CAN_Send_Msg(ext_id, tdata, 8);
}
```

2.3 力矩模式

```

/**
 * @brief 力矩模式
 * @param id 电机ID
 * @param tqe 力矩
 */
void motor_control_tqe(uint8_t id,int32_t tqe)
{
    uint8_t tdata[8] = {0x05, 0x13, 0x00, 0x80, 0x20, 0x00, 0x80, 0x00};

    *(int16_t *)&tdata[2] = tqe;

    CAN_Send_Msg(0x8000 | id, tdata, 4);
}

```

2.4 协同控制模式

```

/**
 * @brief 电机位置-速度-前馈力矩(最大力矩)控制，int16型
 * @param id 电机ID
 * @param pos 位置：单位 0.0001 圈，如 pos = 5000 表示转到 0.5 圈的位置。
 * @param val 速度：单位 0.00025 转/秒，如 val = 1000 表示 0.25 转/秒
 * @param tqe 最大力矩
 */
void motor_control_pos_val_tqe(uint8_t id, int16_t pos, int16_t val, int16_t tqe)
{
    static uint8_t tdata[8] = {0x07, 0x35, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

    *(int16_t *)&tdata[2] = val;
    *(int16_t *)&tdata[4] = tqe;
    *(int16_t *)&tdata[6] = pos;

    CAN_Send_Msg(0x8000 | id, tdata, 8);
}

```

2.5 DQ 电压模式

说明：

1. 可控制 Q 相电压，单位：0.1v，如 vol = 10 表示 Q 相电压为 1V。

```

void motor_control_volt(FDCAN_HandleTypeDef *hfdcanx, uint8_t id, int16_t vol)
{
    static uint8_t tdata[7] = {0x01, 0x00, 0x08, 0x05, 0x1b, 0x00, 0x00};

    *(int16_t *)&tdata[5] = vol;

    can_send(hfdcanx, 0x8000 | id, tdata, sizeof(tdata));
}

```

2.6 DQ 电流模式

说明:

1. 可控制 Q 相电流, 单位: 0.1A, 如 cur = 10 表示 Q 相电压为 1A。

```
void motor_control_cur(FDCAN_HandleTypeDef *hfdcanx, uint8_t id, int16_t cur)
{
    static uint8_t tdata[7] = {0x01, 0x00, 0x09, 0x05, 0x1c, 0x00, 0x00};

    *(int16_t *)&tdata[5] = cur;

    can_send(hfdcanx, 0x8000 | id, tdata, sizeof(tdata));
}
```

2.7 刹车

说明:

1. 电机刹车, 转动电机会有阻尼。

```
/**
 * @brief 电机刹车
 * @param fdcanHandle &hfdcanx
 * @param motor id 电机ID
 */
void set_motor_brake(FDCAN_HandleTypeDef *fdcanHandle, uint8_t id)
{
    static uint8_t cmd[] = {0x01, 0x00, 0x0f};

    can_send(fdcanHandle, 0x8000 | id, cmd, sizeof(cmd));
}
```

2.8 停止

说明:

1. 电机停止, 并失去保持位置的力。

```
/**
 * @brief 电机停止, 注意: 需让电机停止后再重置零位, 否则无效
 * @param fdcanHandle &hfdcanx
 * @param motor id 电机ID
 */
void set_motor_stop(FDCAN_HandleTypeDef *fdcanHandle, uint8_t id)
{
    static uint8_t cmd[] = {0x01, 0x00, 0x00};

    can_send(fdcanHandle, 0x8000 | id, cmd, sizeof(cmd));
}
```

3. 常用类型 (单位) 说明

3.1 电流 (A)

数据类型	LSB	实际 (A)
int8	1	1
int16	1	0.1
int32	1	0.001
float	1	1

3.2 电压 (V)

数据类型	LSB	实际 (V)
int8	1	0.5
int16	1	0.1
int32	1	0.001
float	1	1

3.3 扭矩 (Nm)

- 真实扭矩 = $k * tqe + d$

3.3.1 5046 扭矩 (Nm)

数据类型	斜率 (k)	偏移量 (d)
int16	0.005397	-0.455107
int32	0.000528	-0.414526
float	0.533654	-0.519366

3.3.2 4538 扭矩 (Nm)

数据类型	斜率 (k)	偏移量 (d)
int16	0.004587	-0.290788
int32	0.000445	-0.234668
float	0.493835	-0.473398

3.3.2 5047/6056 (双极, 36 减速比) 扭矩 (Nm)

数据类型	斜率 (k)	偏移量 (d)
int16	0.004563	-0.493257
int32	0.000462	-0.512253
float	0.465293	-0.554848

3.3.3 5047 (单极, 9 减速比) 扭矩 (Nm)

数据类型	斜率 (k)	偏移量 (d)
int16	0.005332	-0.072956
int32	0.000533	-0.034809
float	0.547474	-0.150232

3.4 温度 (°C)

数据类型	LSB	实际 (°C)
int8	1	1
int16	1	0.1
int32	1	0.001
float	1	1

3.5 时间 (s)

数据类型	LSB	实际 (s)
int8	1	0.01
int16	1	0.001
int32	1	0.000001
float	1	1

3.6 位置 (转)

数据类型	LSB	实际 (转)	实际 (°)
int8	1	0.01	3.6
int16	1	0.0001	0.036
int32	1	0.00001	0.0036

数据类型	LSB	实际 (转)	实际 (°)
float	1	1	360

3.7 速度 (转/秒)

数据类型	LSB	实际 (转/秒)
int8	1	0.01
int16	1	0.00025
int32	1	0.00001
float	1	1

3.8 加速度 (转/秒^2)

数据类型	LSB	实际 (转/秒^2)
int8	1	0.05
int16	1	0.001
int32	1	0.00001
float	1	1

3.9 PWM 标度 (无单位)

数据类型	LSB	实际
int8	1	1/127 - 0.007874
int16	1	1/32767 - 0.000030519
int32	1	(1/2147483647) - 4.657^10
float	1	1

3.10 Kp、Kd 标度 (无单位)

数据类型	LSB	实际
int8	1	1/127 - 0.007874
int16	1	1/32767 - 0.000030519
int32	1	(1/2147483647) - 4.657^10
float	1	1