

# 1.3 CAN协议解析

## 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)

```
代码块 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

代码块

```
1 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）

代码块

```
1 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

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。

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。

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：

一个寄存器的 `float` 类型或 `int32_t` 的电机信息。

3 个地址连续的 `int16_t` 类型电机信息。

6 个地址连续的 `int8_t` 类型的电机信息。

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

代码块

```
1  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`。

代码块

```
1  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：

代码块

```
1  /**
2  @brief 读取电机
```

```

3  @param id
4  */
5  void motor_read(uint8_t id)
6  {
7      static uint8_t tdata[8] = {0x17, 0x01};
8
9      CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
10 }
11 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：

代码块

```
1  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。

代码块

```

1  /**
2  @brief 电机停止

```

```
3  */
4  void motor_stop(uint8_t id)
5  {
6      uint8_t tdata[] = {0x01, 0x00, 0x00};
7
8      CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
9  }
```

## 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。

代码块

```
1  void rezero_pos(uint8_t id)
2  {
3      uint8_t tdata[] = {0x40, 0x01, 0x04, 0x64, 0x20, 0x63, 0x0a};
4
5      CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
6      HAL_Delay(1000); // 建议延时1s
7
8      conf_write(id); // 保存设置
9  }
```

## 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。

代码块

```
1  void conf_write(uint8_t id)
2  {
3      uint8_t tdata[] = {0x05, 0xb3, 0x02, 0x00, 0x00};
4
5      CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
6  }
```

## 1.7 读取电机状态

说明：

1. 单次读取电机位置、速度、力矩数据。
2. 电机反馈的状态信息数据解析参考例程中的中断函数 HAL\_FDCAN\_RxFifo0Callback 内的代码。

代码块

```
1  /**
2   @brief 读取电机位置、速度、力矩指令
3   @param id 电机ID
4   */
5   void motor_read(CAN_HandleTypeDef *hcan, uint8_t id)
6   {
7       static uint8_t tdata[8] = {0x17, 0x01};
8
9       can_send(hcan, 0x8000 | id, tdata, sizeof(tdata));
10  }
```

## 1.8 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（即 1.7 读取位置状态）。
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。
6. 电机反馈的状态信息数据解析请参考例程中的中断函数 HAL\_FDCAN\_RxFifo0Callback 内的代码。

代码块

```
1  void timed_return_motor_status(uint8_t id, int16_t t_ms)
2  {
3      uint8_t tdata[] = {0x05, 0xb4, 0x02, 0x00, 0x00};
4
5      *(int16_t *)&tdata[3] = t_ms;
6
7      CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
8  }
```

## 2. 示例函数

### 2.1 DQ 电压模式

说明：

1. 通过给定的 Q 相电压控制电机运行，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `id` : 电机 ID
  - `volt` : Q 相电压，单位：伏特 (V) 。

代码块

```
1 void motor_set_dq_vlot(port_t portx, const uint8_t id, const float volt);
```

### 2.2 DQ 电流模式

说明：

1. 通过给定的 Q 相电流控制电机运行，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `id` : 电机 ID
  - `cur` : Q 相电流，单位：安培 (A) 。

代码块

```
1 void motor_set_dq_current(port_t portx, const uint8_t id, const float cur);
```

### 2.3 位置模式

说明：

1. 电机将以最大速度和最大加速度运动到指定目标位置，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `id` : 电机 ID



- `pos`：目标位置，单位：转（r）。

#### 注意：

1. 该模式下速度与力矩均为最大，运动过程较为激烈。
2. 瞬时电流可能会飙到 5A~10A，若电源响应不够快，或电源电流限制过小，可能导致电机在瞬间获得的电流不足，从而报错。
3. 此模式适用于对响应速度有极端要求的场合，一般不建议使用，如需进行位置控制，推荐使用**梯形控制**模式。

#### 代码块

```
1 void motor_set_pos(port_t portx, const uint8_t id, const float pos);
```

## 2.4 速度模式

#### 说明：

1. 电机以最大加速度加速到指定目标速度，并让电机返回状态信息。
2. 参数解析：
  - `portx`：CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `id`：电机 ID
  - `vel`：目标速度，单位：转/秒（r/s）

#### 代码块

```
1 void motor_set_vel(port_t portx, const uint8_t id, const float vel);
```

## 2.5 力矩模式

#### 说明：

1. 电机按照设定的目标力矩进行转动，并让电机返回状态信息。
2. 参数解析：
  - `portx`：CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `id`：电机 ID
  - `tqe`：目标力矩，单位：牛·米（N·m）。

#### 代码块

```
1 void motor_set_tqe(port_t portx, const uint8_t id, const float tqe);
```

## 2.6 位置、速度模式

说明：

1. 电机以目标速度运动至指定的目标位置，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道， `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `id` : 电机 ID
  - `pos` : 目标位置，单位：转（r）。
  - `vel` : 目标速度，单位：转/秒（r/s）

代码块

```
1 void motor_set_pos_vel(port_t portx, const uint8_t id, const float pos, const float vel);
```

## 2.7 位置、速度、最大力矩模式

说明：

1. 电机以目标速度运动至指定的目标位置，同时限制最大输出力矩，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道， `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `id` : 电机 ID
  - `pos` : 目标位置，单位：转（r）。
  - `vel` : 目标速度，单位：转/秒（r/s）。
  - `tqe` : 目标力矩，单位：牛·米（N·m）。

注意：

1. 该模式对输出力矩有限制，若设置的最大力矩过小，电机可能无法达到目标速度。

代码块

```
1 void motor_set_pos_vel_MAXtqe(port_t portx, const uint8_t id, const float pos, const float vel, const float tqe);
```

## 2.8 位置、速度、加速度模式（梯形控制）

### 说明：

1. 电机按照恒定加速度运动，实现先加速 → 匀速 → 减速的梯形速度控制，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
  - `id`：电机 ID
  - `pos`：目标位置，单位：转（r）。
  - `vel`：目标速度，单位：转/秒（r/s）。
  - `acc`：加速度，单位：转每秒平方（rps<sup>2</sup>）。

### 注意：

1. 电机固件 v4.6.0 开始支持

#### 代码块

```
1 void motor_set_pos_vel_acc(port_t portx, const uint8_t id, const float pos,
    const float vel, const float acc);
```

## 2.9 速度、加速度模式

### 说明：

1. 以目标加速度加速到目标速度，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
  - `id`：电机 ID
  - `vel`：目标速度，单位：转/秒（r/s）。
  - `acc`：加速度，单位：转每秒平方（rps<sup>2</sup>）。

## 2.10 运控模式

### 说明：

1. 电机输出力矩的计算公式为：
  - 输出力矩 = （目标位置-当前位置） \* `kp` + （目标速度-当前速度） \* `kd` + `torque`。

## 2. 参数解析：

- `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `id` : 电机 ID
- `pos` : 目标位置，单位：转 (r) 。
- `vel` : 目标速度，单位：转/秒 (r/s) 。
- `tqe` : 目标力矩，单位：牛·米 (N·m) 。
- `kp` : 位置比例系数。
- `kd` : 速度比例系数。

### 注意：

1. 需要设置合适的 `kp` 和 `kd` ，否则控制效果可能较差。
2. 电机固件 v4.6.0 开始支持

代码块

```
1 void motor_set_pos_vel_tqe_kp_kd(port_t portx, const uint8_t id, const float pos, const float vel, float tqe, float kp, float kd)
```

## 2.11 停止模式

### 说明：

1. 电机进入停止模式，电机三相都悬空，使电机可以自由转动。
2. 参数解析：

- `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `id` : 电机 ID

代码块

```
1 void motor_set_stop(port_t portx, const uint8_t id);
```

## 2.12 刹车模式

### 说明：

1. 将电机所有相短接到地，实现“阻尼刹车”效果。
2. 刹车阻力与电机转速成正相关。
3. 参数解析：

- `portx`: CAN 通道, `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `id`: 电机 ID

代码块

```
1 void motor_set_brake(port_t portx, const uint8_t id);
```