1.2 fdcan Protocol Analysis

1. Protocol Parsing

1.1 CAN-FD Related Instructions

1. CAN-FD Baud Rate:
    - Arbitration Segment: 1 Mbps
    - Data Segment: Supports up to 5 Mbps, and 1 Mbps (variable baud rate) is also available.
2. ID: Consists of 16 bits, where `0x7F` is the broadcast address.
    - High 8 bits: represents**source address**:
        - When the highest bit is 1: It needs to be replied, equivalent to a **master switch** , and turning it on without issuing a query command will return a frame with a data segment length of 0.
        - Bit 0 of the most significant bit: No reply required.
        - The remaining 7 bits: signal source address.
    - Lower 8 bits: represents **destination address**:
        - The maximum is 0.
        - The remaining 7 bits represent the destination address.

For example:

1. `ID: 0x8001`
    - The signal source address is 0.
    - The destination address is 1.
    - A value of 1 in the most significant bit indicates that a reply is required, i.e., turn on the reply **master switch** .
2. `ID: 0x100`
    - The signal source address is 1.
    - The destination address is 0.
    - The highest bit being 0 indicates no need for a reply, i.e., the reply is turned off **Master switch** .

1.2 Basic Description of the Agreement

1. The smallest unit is a subframe, and each subframe can write values to **one or more** registers or read data.
2. A CAN-FD frame can be composed of **one or more** subframes.
3. The implementation of each function of the motor is achieved by writing **one or more** values of the registers corresponding to the function in a single FDCAN frame.
4. can write to any register `int8_t, int16_t, int32_t, float` four basic data types.
5. The data types in the **same subframe** must be the same, and the data types in **different subframes** can be different.
6. All transmissions of basic types are in**little-endian mode**, which means sending low-byte data first and then high-byte data.
7. In a CAN-FD, **the trailing padding** ByteDance should use `0x50`, indicating no operation (NOP).
8. Unrestricted **for various data types** :
   - `int8_t` : `0x80`
   - `int16_t` : `0x8000`
   - `int32_t` : `0x80000000`
   - `float` : `NAN`
9. Mode 3 (One-to-Many Mode) is identified by the ID of FDCAN.
   - When the lower 8 bits of the ID are greater than or equal to `0x80`, it is Mode 3 (One-to-Many Mode).
   - When the lower 8 bits of the ID are less than `0x80`, it is Mode 1 or Mode 2 (Normal Mode).

1.3 Subframe Parsing

1.3.1 Transmission Protocol

1.3.1.1 Mode 1

```
1   uint8_t tdata[] = {cmd, addr, a1, a2, b1, b2...};
```

1. `cmd` : Indicates read/write, data type, and quantity.
   a. High four bits `cmd[7:4]` indicate read, write, and read-back.
      i. `0000` : Write
      ii. `0001` : Read

iii. `0010` : Reply

b. Lower four bits `cmd[3:0]` indicate the data type and number:

i. The upper two bits `cmd[3:2]` indicate the data type:

1. `00` : `int8_t`
2. `01` : `int16_t`
3. `10` : `int32_t`
4. `11` : `float`

ii. The lower two bits `cmd[1:0]` indicate the number of data items.

1. `01` : A piece of data.
2. `10` : Two data points.
3. `11` : Three data points.
4. `00` : **Mode 2** Flag.

2. `addr` : Starting register address.
3. `a1, a2, b1, b2...` : Data written to the register. Note: It must meet items 4 and 5 of the basic description of Protocol 1.2.

a. `a1, a2` : Data to be written into `addr` .

b. `b1, b2` : Data to be written into `addr + 1` .

c. `...` : Data to be written into `add + n` .

1.3.1.2 Mode 2

```
uint8_t tdata[] = {cmd, num, addr, a1, a2, b1, b2...};
```

1. `cmd` : Indicates read/write, data type, and quantity.

a. High four bits `cmd[7:4]` indicate read, write, and read-back.

i. `0000` : Write

ii. `0001` : Read

iii. `0010` : Reply

b. Lower four bits `cmd[3:0]` indicate the data type and number:

i. The upper two bits `cmd[3:2]` indicate the data type:

1. `00` : `int8_t`

2. `01` : `int16_t`
3. `10` : `int32_t`
4. `11` : `float`

ii. The lower two bits `cmd[1:0]` are fixed to `00` , indicating Mode 2, and the following byte represents the number of data items.

2. `num` : Number of data items.
3. `addr` : Starting register address.
4. `a1, a2, b1, b2...` : Data written to the register. Note: It must meet items 4 and 5 of the basic description of Protocol 1.2.
   a. `a1, a2` : Data to be written into `addr` .
   b. `b1, b2` : Data to be written into `addr + 1` .
   c. `...` : Data to be written into `add + n` .

Essence of Mode 2: When the number of data in `xxx` of Mode 1 is written as 0, the next byte is used to represent the number of data, and the remaining bytes are shifted one position backward.

1.3.1.3 Mode 3 (One-to-Many Mode)

- Mode 3 is determined by the **lower 8 bits** of FDCAN-ID. When FDCAN-ID is greater than **0x80** , it is **one-to-many mode** (see the example routine for the specific mode)
- The protocol format is: Motor 1 data, Motor 2 data, Motor 3 data... **The last 2 ByteDance** are the query motor status codes.
- Motor 1, Motor 2...: refers to the motor with id 1, the motor with id 2...
- The data types sent in the one-to-many mode are all **int16 type** .

Example 1: ID is 0x8090

```
1   uint8_t cmd[] = {pos11, pos12, val11, val12, tqe11, tqe12, pos21, pos22,
    val21, val22, tqe21, tqe22, Placeholder byte, 0x17, 0x01}
```

- When the ID is 0x8090, it is for one-to-many position, speed, and torque control.
- Byte 1 to Byte 6 are respectively for position, velocity, and torque control.
- Byte 7 to Byte 12 are respectively for position, velocity, and torque control.
- Byte 13 to Byte 18 correspond to position, velocity, and torque control respectively.
- Using this internal referral...

- The last 2 bytes are the instructions for querying the motor status.

Example 2: ID is 0x8093

```
uint8_t cmd[] = {pos11, pos12, val11, val12, tqe11, tqe12, rkp11, rkp12,
rkd11, rkd12, pos21, pos22, val21, val22, tqe21, tqe22, rkp21, rkp22, rkd21,
rkd22, Placeholder byte, 0x17, 0x01}
```

- When the ID is 0x8093, it represents one-to-many position, velocity, torque, and PD control.
- Byte 1 to Byte 10 are respectively position, velocity, torque, and PD control.
- Byte 11 to Byte 20 are respectively position, velocity, torque, and PD control.
- Byte 21 to Byte 30 are respectively position, velocity, torque, and PD control.
- Using this internal referral...
- The last 2 bytes are the instructions for querying the motor status.

1.3.2 Receiving Protocol

- **The acceptance protocol mode is determined by the mode of the sending protocol.**
- **The receiving protocol mode and the sending protocol mode are the same.**

1.3.2.1 Mode 1

Assume the acquired data is `int16_t`

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

- `cmd` :
  - High four bits `cmd[7, 4]` : `0010` indicates a reply.
  - 2~3 bits `cmd[3, 2]` : Indicates the type.
    - `00` : `int8_t` type.
    - `01` : `int16_t` type.
    - `10` : `int32_t` type.
    - `11` : `float` type.
  - Lower 2 bits `cmd[1, 0]` : Indicates the quantity.
    - `00` : Indicates Mode 2.

        - `01` : A piece of data.
        - `10` : Two data points.
        - `11` : Three data points.
- `addr` : The address to start fetching from.
- `a1, a2` : Data 1, Little Endian.
- `b1, b2` : Data 2, Little Endian.

1.3.2.2 Mode 2

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

- `cmd` :
    - High four bits `cmd[7, 4]` : `0010` indicates a reply.
    - 2~3 bits `cmd[3, 2]` : Indicates the type.
        - `00` : `int8_t` type.
        - `01` : `int16_t` type.
        - `10` : `int32_t` type.
        - `11` : `float` type.
    - Lower 2 bits `cmd[1, 0]` : Indicates the quantity.
        - `00` : Indicates Mode 2, and the subsequent ByteDance represents the data quantity.
- `num` : Number of data items.
- `addr` : The address to start fetching from.
- `a1, a2` : Data 1, Little Endian.
- `b1, b2` : Data 2, Little Endian.

1.3.3 Example

1.3.3.1 Example of Sending Protocol

1.3.3.1.1 Mode 1

```
uint8_t cmd[] = {0x01, 0x00, 0x0A, 0x0A, 0x20, 0x00, 0x00, 0x00, 0x80, 0x10, 0x27, 0x00, 0x00, 0x50, 0x50, 0x50};
```

This CAN-FD frame consists of two subframes:

1. Subframe 1: The overall meaning is that the motor enters the position mode.
   - `0x01` : Beginning of the first subframe
     - The high four bits are `0000` , indicating a write operation.
     - Lower four bits:
       - The high 2 bits are: `00` , indicating `int8_t` type.
       - The lower 2 bits are: `01` , indicating 1 data item.
   - `0x00` : Starting register address: As shown in the lookup table, `0x00` register represents motor mode setting.
   - `0x0A` : Write to `0x00` register `0x0A` .
2. Subframe 2: The overall meaning is that the position is unrestricted and the speed is 0.1 revolutions per second
   - `0x0A` : Beginning of the 2nd subframe
     - The high 8 bits are `0x0` , indicating a write operation.
     - Lower 8 bits:
       - The upper 2 bits are: `10` , indicating `int32_t` type.
       - The lower 2 bits are: `10` , indicating 2 data items.
   - `0x20` : Starting register address: As shown in the lookup table, `0x20` register represents position, `0x21` register represents speed.
   - `0x00, 0x00, 0x00, 0x80` : Little-endian mode, i.e., `0x80000000` written to `0x20` register, which means the motor position is unrestricted.
   - `0x10, 0x27, 0x00, 0x00` : Little-endian mode, i.e., `0x2710` written to `0x21` register, indicating that the motor speed is set to 0.1 revolutions per second.
3. `0x50, 0x50, 0x50` : Since the two closest numbers of bytes sent by CAN-FD are 12 and 16, 3 placeholder bytes need to be added.

1.3.3.1.2 Mode 2

```
1   uint8_t cmd[] = {0x01, 0x00, 0x0A, 0x08, 0x02, 0x20, 0x00, 0x00, 0x00, 0x80,
    0x10, 0x27, 0x00, 0x00, 0x50, 0x50};
```

This CAN-FD frame consists of two subframes:

1. Subframe 1: The overall meaning is that the motor enters the position mode.

- `0x01` : Beginning of the first subframe
  - The high 8 bits are `0001` , indicating a write operation.
  - Lower 8 bits:
    - The high 2 bits are: `00` , indicating `int8_t` type.
    - The lower 2 bits are: `01` , indicating 1 data item.
- `0x00` : Starting register address: As shown in the lookup table, `0x00` register represents motor mode setting.
- `0x0A` : Write to `0x00` register `0x0A` .

2. Subframe 2: The overall meaning is that the position is unrestricted and the speed is 0.1 revolutions per second
   - `0x08` : Beginning of the 2nd subframe
     - The high 8 bits are `0x0` , indicating a write operation.
     - Lower 8 bits:
       - The upper 2 bits are: `10` , indicating `int32_t` type.
       - The lower 2 bits are: `00` , indicating Mode 2, and the following ByteDance represents the number of data.
   - `0x02` : 2 data items.
   - `0x20` : Starting register address: As shown in the lookup table, `0x20` register represents position, `0x21` register represents speed.
   - `0x00, 0x00, 0x00, 0x80` : Little-endian mode, i.e., `0x80000000` written to `0x20` register, which means the motor position is unrestricted.
   - `0x10, 0x27, 0x00, 0x00` : Little-endian mode, i.e., `0x2710` written to `0x21` register, indicating that the motor speed is set to 0.1 revolutions per second.
3. `0x50, 0x50` : Since the two closest numbers of CAN-FD transmitted bytes are 12 and 16, 2 placeholder bytes need to be added.

1.3.3.2 Example of Receiving Protocol

- **The acceptance protocol mode is determined by the mode of the sending protocol.**
- **The receiving protocol mode and the sending protocol mode are the same.**

1.3.3.2.1 Mode 1

```
uint8_t rdata[] = {0x2B, 0x01, 0x96, 0x1D, 0x04, 0x00, 0x54, 0x40, 0x02,
0x00, 0x86, 0x01, 0x00, 0x00, 0x50, 0x50};
```

This frame consists of a subframe:

1. Subframe 1: The overall meaning is that the motor enters the position mode.
   - `0x2B` :
     - The highest 8 bits are `0x2` , indicating a reply operation.
     - Lower 8 bits:
       - The upper 2 bits are: `10` , indicating `int32_t` type.
       - The lower 2 bits are: `11` , indicating 3 data.
   - `0x01` : Starting register address
   - `0x96, 0x1D, 0x04, 0x00` : Little-endian mode, decimal value is 269718, which means the current position of the motor is at 2.69718 revolutions.
   - `0x54, 0x40, 0x02, 0x00` : Little-endian mode, decimal value is 147540, which means the current motor speed is 1.4754 revolutions per second.
   - `0x86, 0x01, 0x00, 0x00` : Little-endian mode, decimal value is 390, which means the current actual output torque is: 0.0039 NM.
   - `0x50, 0x50` : Placeholder.

From the first three characters, we know that: This CAN-FD is a response to `0x1B, 0x01` .

1.3.3.2.2 Mode 2

```
uint8_t rdata[] = {0x28, 0x04, 0x00, 0x0A, 0x00, 0x00, 0x00, 0x96, 0x1D,
0x04, 0x00, 0x54, 0x40, 0x02, 0x00, 0x86, 0x01, 0x00, 0x00, 0x50};
```

This frame consists of a subframe:

1. Subframe 1: The overall meaning is that the motor enters the position mode.
   - `0x28` :
     - The high 8 bits are `0010` , indicating a reply operation.
     - Lower 8 bits:
       - The upper 2 bits are: `10` , indicating `int32_t` type.
       - The lower 2 bits are: `00` , indicating Mode 2, and the following ByteDance represents the number of data.
   - `0x04` : 4 data items.
   - `0x00` : Starting register address

- `0x0A, 0x00, 0x00, 0x00` : `0x00` The value of the register, which can be found from the table to be the motor mode, is 10 in decimal and can be found from the table to be the position mode.
- `0x96, 0x1D, 0x04, 0x00` : Little-endian mode, decimal value is 269718, which means the current position of the motor is at 2.69718 revolutions.
- `0x54, 0x40, 0x02, 0x00` : Little-endian mode, decimal value is 147540, which means the current motor speed is 1.4754 revolutions per second.
- `0x86, 0x01, 0x00, 0x00` : Little-endian mode, decimal value is 390, which means the current actual output torque is: 0.0039 NM.
- `0x50` : Placeholder.

From the first three characters, we know that: This CAN-FD is a response to `0x18, 0x04, 0x00` .

2. Explanation of Common Types (Units)

2.1 Current (A)

Data Type	LSB	Actual (A)
`int8`	1	1
`int16`	1	0.1
`int32`	1	0.001
`float`	1	1

2.2 Voltage (V)

Data Type	LSB	Actual (V))
`int8`	1	0.5
`int16`	1	0.1
`int32`	1	0.001
`float`	1	1

2.3 Torque (Nm)

- **True Torque = k * tqe + d**
- **Item d:** Equivalent to the friction of the motor itself, it can be omitted for applications with low precision requirements. Starting from v3.0.5, the fdcan_h730 project no longer includes this item.
- Torque coefficient is subject to `motor_tqe_adj` in fdcan_h730.

Motor Model	Torque Coefficient
M3536_32	0.458105
M4438_30	0.5256
M4438_32	0.485565
M4538_19	0.493835
M5043_20	0.966
M5046_20	0.533654
M5047_09	0.547474
M5047_36	0.803
M6056_36	0.677
M7256_35	0.676524
M60SG_35	0.7942
M60BM_35	0.7942

2.4 Temperature (°C)

Data Type	LSB	Actual (°C)
`int8`	1	1
`int16`	1	0.1
`int32`	1	0.001
`float`	1	1

2.5 Time (s)

Data Type	LSB	Actual (s)
`int8`	1	0.01
`int16`	1	0.001
`int32`	1	0.000001
`float`	1	1

2.6 Position (rotation)

Data Type	LSB	Actual (Turn)	Actual (°)
`int8`	1	0.01	3.6
`int16`	1	0.0001	0.036
`int32`	1	0.00001	0.0036
`float`	1	1	360

2.7 Speed (rev/s)

Data Type	LSB	Actual (rps)
`int8`	1	0.01
`int16`	1	0.00025
`int32`	1	0.00001
`float`	1	1

2.8 Acceleration (rev/s^2)

Data Type	LSB	Actual (rev/s^2)
`int8`	1	0.05
`int16`	1	0.001
`int32`	1	0.00001
`float`	1	1

2.9 PWM Scale (unitless)

Data Type	LSB	Actual
`int8`	1	1/127 - 0.007874
`int16`	1	1/32767 - 0.000030519
`int32`	1	(1/2147483647) - 4.657^10
`float`	1	1

2.10 rKp, rKd Scaling (Unitless, for Registers 0x23 and 0x24)

Data Type	LSB	Actual
`int8`	1	1/127 - 0.007874
`int16`	1	1/32767 - 0.000030519
`int32`	1	(1/2147483647) - 4.657^10
`float`	1	1

2.11 Kp, Kd Scaling (unitless, used in the example routine)

Data Type	LSB	Actual
`int8`	1	1
`int16`	1	0.1
`int32`	1	0.001
`float`	1	1

3. Function Example

Description:

- **The following provides a brief description of the motor control modes provided in the example program.**
- **For details, please refer to the provided sample project.**

- **The functions that this motor can achieve are not limited to this. If you need to customize special functions, you can use your imagination based on the register table.**
- **The following are encapsulated functions that can be directly called**

3.1 Normal Mode

3.1.1 DQ Voltage Mode

Description:

1. Control the motor operation through the given Q-phase voltage and make the motor return status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `volt` : Q-phase voltage, unit: volt (V).

代码块

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

3.1.2 DQ Current Mode

Description:

1. Control the motor operation through the given Q-phase current and have the motor return status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `cur` : Q-phase current, unit: Ampere (A).

代码块

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

3.1.3 Position Mode

Description:

1. The motor will move to the specified target position at maximum speed and maximum acceleration, and return status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).

Note:

1. In this mode, both speed and torque are at their maximum, and the motion process is relatively intense.
2. The instantaneous current may surge to 5A - 10A. If the power supply response is not fast enough or the power supply current limit is too small, it may cause the motor to receive insufficient current instantaneously, resulting in an error.
3. This mode is suitable for situations with extreme requirements for response speed, and is generally not recommended. If position control is required, it is recommended to use **Trapezoidal Control** mode.

代码块

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

3.1.4 Speed Mode

Description:

1. The motor accelerates to the specified target speed at maximum acceleration and returns status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID

- `vel` : Target speed, unit: revolutions per second (r/s)

代码块

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

3.1.5 Torque Mode

Description:

1. The motor rotates according to the set target torque and returns status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `tqe` : Target torque, unit: Newton meter (N·m).

代码块

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

3.1.6 Position, Velocity Mode

Description:

1. The motor moves to the specified target position at the target speed and returns status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).
   - `vel` : Target speed, unit: revolutions per second (r/s)

代码块

```
1   void motor_set_pos_vel(port_t portx, const data_type_t type, const uint8_t id,
```

```
2   const float pos, const float vel);
```

3.1.7 Position, Velocity, Maximum Torque Mode

Description:

1. The motor moves to the specified target position at the target speed, while limiting the maximum output torque and allowing the motor to return status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).
   - `vel` : Target speed, unit: revolutions per second (r/s).
   - `tqe` : Target torque, unit: Newton meter (N·m).

Note:

1. This mode has a limit on the output torque. If the set maximum torque is too small, the motor may not be able to reach the target speed.

代码块

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

3.1.8 Position, Velocity, Acceleration Mode (Trapezoidal Control)

Description:

1. The motor moves at a constant acceleration, achieving a trapezoidal speed control of acceleration → constant speed → deceleration, and returns status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).

- `vel` : Target speed, unit: revolutions per second (r/s).
- `acc` : Acceleration, Unit: revolutions per second squared (rps^2).

代码块

```
1   void motor_set_pos_velmax_acc(port_t portx, const data_type_t type, const
    uint8_t id,
2   const float pos, const float vel, const float acc);
```

3.1.9 Speed and Acceleration Mode

Description:

1. Accelerate to the target speed at the target acceleration and have the motor return status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `vel` : Target speed, unit: revolutions per second (r/s).
   - `acc` : Acceleration, Unit: revolutions per second squared (rps^2).

代码块

```
1   void motor_set_vel_acc(port_t portx, const data_type_t type, const uint8_t id,
2   const float vel, const float acc);
```

3.1.10 Motion Control Mode

Description:

1. The formula for calculating the output torque of the motor is:
   - Output torque = (target position - current position) * kp + (target velocity - current velocity) * kd + torque.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID

- `pos` : Target position, unit: revolution (r).
- `vel` : Target speed, unit: revolutions per second (r/s).
- `tqe` : Target torque, unit: Newton meter (N·m).
- `kp` : Position proportional coefficient.
- `kd` : Velocity proportional coefficient.

Note:

1. Appropriate `kp` and `kd` need to be set; otherwise, the control effect may be poor.
2. `The position of the motor_set_pos_vel_tqe_kp_kd function is obtained through continuous integration, so unexpected situations may occur under low-frequency control.`
3. It is recommended to use motion control mode 2 `motor_set_pos_vel_tqe_kp_kd_2` .

代码块

```
/* 不建议使用 */
void motor_set_pos_vel_tqe_kp_kd(port_t portx, const data_type_t type, const uint8_t id,
const float pos, const float vel, const float tqe, const float kp, const float kd);

/* 建议使用 */
void motor_set_pos_vel_tqe_kp_kd_2(port_t portx, const data_type_t type, const uint8_t id,
const float pos, const float vel, const float tqe, const float kp, const float kd);
```

3.1.11 Motion Control Mode 2

Description:

1. The formula for calculating the output torque of the motor is:
   - Output torque = (target position - current position) * kp + (target velocity - current velocity) * kd + torque.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID

- `pos` : Target position, unit: revolution (r).
- `vel` : Target speed, unit: revolutions per second (r/s).
- `tqe` : Target torque, unit: Newton meter (N·m).
- `kp` : Position proportional coefficient.
- `kd` : Velocity proportional coefficient.

Note:

1. The real motion control mode, and `motor_set_pos_vel_tqe_kp_kd` difference is:
   - `motor_set_pos_vel_tqe_kp_kd` is obtained through continuous integration, and unexpected situations are likely to occur under low-frequency control.
   - `motor_set_pos_vel_tqe_kp_kd_2` avoids this problem and provides more stable control.
2. Appropriate `kp` and `kd` need to be set; otherwise, the control effect may be poor.

代码块

```
/* 不建议使用 */
void motor_set_pos_vel_tqe_kp_kd(port_t portx, const data_type_t type, const uint8_t id,
const float pos, const float vel, const float tqe, const float kp, const float kd);

/* 建议使用 */
void motor_set_pos_vel_tqe_kp_kd_2(port_t portx, const data_type_t type, const uint8_t id,
const float pos, const float vel, const float tqe, const float kp, const float kd);
```

3.1.12 Query motor status information

Description:

1. Send a command to query motor status information.
2. The status information returned by the motor includes mode, error code, position, speed, and torque.
3. Motor status information parsing is in the `motor.c` file's `motor_process_state` function.
4. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3

- `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
- `id` : Motor ID

```
代码块
1   void motor_get_state_send(port_t portx, const data_type_t type, const uint8_t
    id);
```

3.1.13 Query motor firmware version

Description:

1. Send the command to query the motor firmware version number.
2. Motor firmware version parsing is in the `motor.c` file's `motor_process_state` function.
3. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `id` : Motor ID

```
代码块
1   void motor_get_version(port_t portx, const uint8_t id);
```

3.1.14 Stop Mode

Description:

1. The motor enters the stop mode, with all three phases of the motor floating, allowing the motor to rotate freely.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `id` : Motor ID

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

3.1.15 Brake Mode

Description:

1. Short all phases of the motor to ground to achieve the "damping brake" effect.

2. Brake resistance is positively correlated with motor speed.
3. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `id` : Motor ID

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

3.1.16 Motor Soft Restart

Description:

1. Perform a software restart on the motor. After the restart, it enters the stop mode.
2. There will be no feedback.
3. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `id` : Motor ID

```
代码块
1   void motor_set_reset(port_t portx, const uint8_t id);
```

3.2 One-to-many control mode `motor_many.c`

- Within this file, only `motor_many_send` is used to send instructions, while other functions are used to write instructions to the cache.
- In one-to-many mode, the motor modes on the same CAN channel are the same.
- The general principle of the one-to-many mode is to send a universal FDCAN frame, where different ByteDances on the frame control different motors, with the motor mode determined by the ID. For detailed instructions, please refer to the 02-fdcan protocol analysis.
- In the one-to-many mode, when the arbitration segment is 1M and the data segment is 5M, a CAN bus can control up to 10 motors within 1ms, that is, up to 10 motors at a control frequency of 1KHz. If more motors need to be controlled, the control frequency needs to be reduced.
- `TINT16` corresponds to `the int16` data type, and the control can only reach 3.2 turns.
- One-to-Many Mode **Instructions for Use** :

代码块

```
/* Write to cache */
motor_many_vel(PORT1, 1, 0.1);
motor_many_vel(PORT1, 2, 0.1);
motor_many_vel(PORT1, 3, 0.1);

/* Send the command */
motor_many_send(PORT1);
```

3.2.1 DQ Voltage Mode

Description:

1. Control the motor operation through the given Q-phase voltage and make the motor return status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `volt` : Q-phase voltage, unit: volt (V).

Note:

1. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

代码块

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

3.2.2 DQ Current Mode

Description:

1. Control the motor operation through the given Q-phase current and have the motor return status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`

- `id` : Motor ID
- `cur` : Q-phase current, unit: Ampere (A).

Note:

1. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

代码块

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

3.2.3 Position Mode

Description:

1. The motor will move to the specified target position at maximum speed and maximum acceleration, and return status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).

Note:

1. In this mode, both speed and torque are at their maximum, and the motion process is relatively intense.
2. The instantaneous current may surge to 5A - 10A. If the power supply response is not fast enough or the power supply current limit is too small, it may cause the motor to receive insufficient current instantaneously, resulting in an error.
3. This mode is suitable for situations with extreme requirements for response speed, and is generally not recommended. If position control is required, it is recommended to use **Trapezoidal Control** mode.
4. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

代码块

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

3.2.4 Speed Mode

Description:

1. The motor accelerates to the specified target speed at maximum acceleration and returns status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `vel` : Target speed, unit: revolutions per second (r/s)

Note:

1. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

```
代码块
1   void motor_many_vel(port_t portx, const uint8_t id, const float vel);
```

3.2.5 Torque Mode

Description:

1. The motor rotates according to the set target torque and returns status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `tqe` : Target torque, unit: Newton meter (N·m).

Note:

1. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

```
代码块
1   void motor_many_tqe(port_t portx, const uint8_t id, const float tqe);
```

3.2.6 Position, Velocity Mode

Description:

1. The motor moves to the specified target position at the target speed and returns status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).
   - `vel` : Target speed, unit: revolutions per second (r/s)

Note:

1. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

代码块

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

3.2.7 Position, Velocity, Maximum Torque Mode

Description:

1. The motor moves to the specified target position at the target speed, while limiting the maximum output torque and allowing the motor to return status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).
   - `vel` : Target speed, unit: revolutions per second (r/s).
   - `tqe` : Target torque, unit: Newton meter (N·m).

   Note:

3. This mode has a limit on the output torque. If the set maximum torque is too small, the motor may not be able to reach the target speed.
4. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

代码块

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

3.2.8 Position, Velocity, Acceleration Mode (Trapezoidal Control)

Description:

1. The motor moves at a constant acceleration, achieving a trapezoidal speed control of acceleration → constant speed → deceleration, and returns status information.
2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).
   - `vel` : Target speed, unit: revolutions per second (r/s).
   - `acc` : Acceleration, Unit: revolutions per second squared (rps^2).

Note:

1. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

代码块

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

3.2.9 Motion Control Mode

Description:

1. The formula for calculating the output torque of the motor is:
   - Output torque = (target position - current position) * kp + (target velocity - current velocity) * kd + torque.

2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).
   - `vel` : Target speed, unit: revolutions per second (r/s).
   - `tqe` : Target torque, unit: Newton meter (N·m).
   - `kp` : Position proportional coefficient.
   - `kd` : Velocity proportional coefficient.

   Note:
3. Appropriate `kp` and `kd` need to be set; otherwise, the control effect may be poor.
4. `The position of the motor_many_pos_vel_tqe_kp_kd function is obtained through continuous integration, so unexpected situations may occur under low-frequency control.`
5. It is recommended to use motion control mode 2 motor_many_pos_vel_tqe_kp_kd_2.
6. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

代码块

```
/* 不建议使用 */
void motor_many_pos_vel_tqe_kp_kd(port_t portx, const uint8_t id,
const float pos, const float vel, const float tqe, const float kp, const float kd);

/* 建议使用 */
void motor_many_pos_vel_tqe_kp_kd_2(port_t portx, const uint8_t id,
const float pos, const float vel, const float tqe, const float kp, const float kd);
```

3.2.10 Motion Control Mode 2

Description:

1. The formula for calculating the output torque of the motor is:
   - Output torque = (target position - current position) * kp + (target velocity - current velocity) * kd + torque.

2. Parameter Parsing:
   - `portx` : CAN channel, `PORT1, PORT2, PORT3` correspond to channels 1, 2, 3
   - `type` : The data type of the Communication Protocol, which affects the precision and range of data, `TFLOAT, TINT32, TINT16` correspond to `float, int32, int16`
   - `id` : Motor ID
   - `pos` : Target position, unit: revolution (r).
   - `vel` : Target speed, unit: revolutions per second (r/s).
   - `tqe` : Target torque, unit: Newton meter (N·m).
   - `kp` : Position proportional coefficient.
   - `kd` : Velocity proportional coefficient.

   Note:
3. The real motion control mode, and `motor_many_pos_vel_tqe_kp_kd` difference is:
   - `motor_many_pos_vel_tqe_kp_kd` is obtained through continuous integration, and unexpected situations are likely to occur under low-frequency control.
   - `motor_many_pos_vel_tqe_kp_kd_2` avoids this problem and provides more stable control.
4. Appropriate `kp` and `kd` need to be set; otherwise, the control effect may be poor.
5. This function only writes the instruction to the buffer and does not send it immediately. It needs to call `motor_many_send` to take effect.

代码块

```
/* Not recommended */
void motor_many_pos_vel_tqe_kp_kd(port_t portx, const uint8_t id,
const float pos, const float vel, const float tqe, const float kp, const float kd);

/* Recommended to use */
void motor_many_pos_vel_tqe_kp_kd_2(port_t portx, const uint8_t id,
const float pos, const float vel, const float tqe, const float kp, const float kd);
```

4. Example Program Instructions

- **All motor example functions are located within the livelybot_fdcan.c and livelybot_fdcan.h files.**

- **The sample program does not run any control code by default. To test a specific function, please uncomment or write your own code in the main.c file.**

Macro definitions that need attention:

1. `MOTOR_TORQUE_RATIO` : Used to select the motor model and correct the input torque.
2. `POS_FLAG` : Used for **testing position modes** , with a total of three modes. The three modes differ only in data type, and their effects are all -0.5 turns to 0.5 turns of back-and-forth rotation.
    a. `POS_FLAG = 1` : float
    b. `POS_FLAG = 2` : int32
    c. `POS_FLAG = 3` : int16
3. `READ_MOTOR_FLAG` : Used to change the data type for reading **motor status**.
    a. `READ_MOTOR_FLAG = 1` : float
    b. `READ_MOTOR_FLAG = 2` : int 32
    c. `READ_MOTOR_FLAG = 3` : int 16
4. `data_type_t` : Used to change the data type for reading **motor status**.
    a. `TFLOAT` : float
    b. `TINT32` : int 32
    c. `TINT16` : int 16
5. `port_t` : Used to select**motor channel**
    a. `PNULL` : 0
    b. `PORT1` : 1
    c. `PORT2` : 2
    d. `PORT3` : 3
6. `POS_REZERO` : Used to test the **reset zero point** function of the motor,
    a. The effect is that the current position is set to zero 3 seconds after the motor is powered on.
    b. Note: The motor must be stopped before resetting the zero position; otherwise, it will be invalid.
    c. To test this function, simply uncomment POS_REZERO.
7. `MOTOR_STOP` : Used to test the motor **stop** function.
    a. To test this function, simply uncomment MOTOR_STOP.

    b. The effect is that the motor will stop 3 seconds after powering on.
    c. Note: It needs to be used in conjunction with the motor control function. Enabling the MOTOR_STOP macro will not change any motor control function.
8. `MOTOR_BRAKE` : Used to test the motor **brake** function.
    a. To test this function, simply uncomment MOTOR_BRAKE. ****