1. Protocol Parsing

1.1 CAN Related Instructions

1. **CAN Baud Rate**:
    - Arbitration segment: 1 Mbps
    - Data segment: 1 Mbps
2. **ID**: Composed of 16 bits, where 0x7F is the broadcast address.
    - **High 8 bits**: Represent the source address:
        - The highest bit is 1: Requires a response.
        - The highest bit is 0: No response is required.
        - The remaining 7 bits: Signal source address.
    - **Low 8 bits**: Represent the destination address:
        - The highest bit is 0.
        - The remaining 7 bits: Destination address.

Example:

1. ID: 0x8001
    - Source address: 0.
    - Destination address: 1.
    - Highest bit is 1, requiring a response.
2. ID: 0x100
    - Source address: 1.
    - Destination address: 0.
    - Highest bit is 0, no response is required.

1.2 Mode Description

1.2.1 Normal Mode (Position and speed cannot be controlled simultaneously)

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

- The normal protocol consists of 8 bytes: command bits (2 bytes) + position (2 bytes) + speed (2 bytes) + torque (2 bytes).
- `0x07 0x07`: Normal mode, allowing control of speed and torque, or position and torque (see [[#2.1 Normal Mode]]).
- Position, speed, and torque data in the protocol use little-endian mode, i.e., the lower byte is sent first, followed by the higher byte.
  - For example, if `pos = 0x1234`, then `pos1 = 0x34` and `pos2 = 0x12`.
- This mode can be divided into two control methods:
  - Position and torque control (when `val = 0x8000`, indicating unlimited).
  - Speed and torque control (when `pos = 0x8000`, indicating unlimited).

1.2.2 Torque Mode

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

- The torque mode protocol consists of: command bits (2 bytes) + torque (2 bytes).
- `0x05 0x13`: Pure torque mode, followed by two bytes of torque data (see [[#2.3 Torque Mode]]).
- Torque data in the protocol uses little-Endian mode, i.e., the lower byte is sent first, followed by the higher byte.
  - For example, if `tqe = 0x1234`, then `tqe1 = 0x34` and `tqe2 = 0x12`.

1.2.3 Cooperative Control Mode (Position, speed, and torque can be controlled simultaneously)

```
uint8_t cmd[] = {0x07, 0x35, val1, val2, tqe1, teq2, pos1, pos2};
```

- The cooperative control mode protocol consists of 8 bytes: command bits (2 bytes) + speed (2 bytes) + torque (2 bytes) + position (2 bytes).
- `0x07 0x35`: Cooperative control mode, specifying rotation at a designated speed to a specified

position with a limited maximum torque.

- In this mode, the parameter `0x8000` indicates unlimited (unlimited speed and torque are the maximum values).
  - For example, `val = 5000`, `tqe = 1000`, `pos = 0x8000` means the motor rotates at 0.5 revolutions per second with a maximum torque of 0.1NM.
- Position, speed, and torque data in the protocol use little-Endian mode, i.e., the lower byte is sent first, followed by the higher byte.
  - For example, if `pos = 0x1234`, then `pos1 = 0x34` and `pos2 = 0x12`.

1.3 Motor Status Data Reading

1. The protocol for reading motor status is the same as that in CAN-FD, with the only difference being that CAN is limited by an 8-byte data segment.
2. For register addresses and function descriptions, please refer to the "Register Functions, Motor Operation Modes, and Error Code Description.xlsx" file.
3. Due to the 8-byte data segment limitation of CAN, a single CAN frame can return a limited amount of motor information:
   - One float-type or int32_t-type motor information from a register.
   - Three consecutive int16_t-type motor information from registers.
   - Six consecutive int8_t-type motor information from registers.
4. The example provides a C-language union-based sample function for querying motor position, speed, and torque information (int16_t type) and motor information parsing (which directly copies bytes 3 to 8 of CAN data).

1.3.1 Transmission Protocol Description

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

Rough meaning: Read `cmd[0, 1]` number of `cmd[3, 2]`-type data from `addr`.

- **cmd**:
  - High 4 bits [7, 4]: 0001 indicates reading.
  - Bits 2~3 [3, 2]: Indicate the type:

        - 00: int8_t type.
        - 01: int16_t type.
        - 10: int32_t type.
        - 11: float type.
    - Low 2 bits [1, 0]: Indicate the quantity:
        - 01: One data.
        - 10: Two data.
        - 11: Three data.
- **addr**: The starting address for acquisition.

Multiple `cmd` and `addr` can be concatenated to read discontinuous addresses and different types of data in one go.

1.3.2 Reception Protocol Description

Assuming the acquired data is uint16_t:

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

- **cmd**:
    - High 4 bits [7, 4]: 0010 indicates a response.
    - Bits 2~3 [3, 2]: Indicate the type (same as transmission).
        - 00: int8_t
        - 01: int16_t
        - 10: int32_t
        - 11: float
    - Low 2 bits [1, 0]: Indicate the quantity (same as transmission).
        - 01: One
        - 10: Two
        - 11: Three
- **addr**: The starting address for acquisition.

- a1, a2: Data 1, in little-Endian mode.
- b1, b2: Data 2, in little-Endian mode.

1.3.3 Example

1. Require reading position, speed, and torque data.
2. From the register Excel table, the data addresses for position, speed, and torque are 01, 02, and 03 , respectively.
3. Therefore, 3 data can be read consecutively starting from address 01. Considering that CAN can transmit a maximum of 8 bytes of data, and cmd + addr occupies 2 bytes, the data type can at most be int16_t.
4. Thus, the binary value of cmd is 0001 1011, and the hexadecimal value is 0x17.
5. Reading starts from address 01, so addr = 0x01.
6. The total data to be sent is uint8_t tdata[] = {0x17, 0x01}.

Sample code:

```
/**
* @brief Read motor status
* @param id Motor 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}:

Read 3 int16_t registers starting from address 0x01 (as per the table, registers at addresses 0x01~0 x03 represent position, speed, and torque), so this command queries the motor's position, speed, and torque information.

- 0x17:
  - 0x17[7:4] in binary is 0001: Indicates reading.
  - 0x17[3:2] in binary is 01: Indicates int16_t data type.
  - 0x17[1:0] in binary is 11: Indicates 3 data points.
- 0x01: Starts from address 0x01.

Corresponding reception data example:

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

- 0x27: Corresponding to the transmitted 0x17.
- 0x01: Starts from address 0x01.
- 0x38 0xf6: Position data: 0xf638, i.e., -2505.
- 0x09 0x00: Speed data: 0x0009, i.e., 9.
- 0x00 0x00: Torque data: 0x0000, i.e., 0.

1.4 Motor Stop

- **Description**:
  1. Stop the motor.
  2. Corresponding to the host computer command d stop.

Code:

```
/**
* @brief Stop the motor
*/
void motor_stop(uint8_t id)
{
    uint8_t tdata[] = {0x01, 0x00, 0x00};
    CAN_Send_Msg(0x8000 | id, tdata, sizeof(tdata));
}
```

1.5 Reset Motor Zero Position

- **Description**:
  1. Set the current position as the motor's zero position.
  2. This command only modifies the RAM; it needs to be used with the conf write command to save to flash.
  3. It is recommended to send the conf write command about 1 second after using this command .

Code:

```
```

```
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); // Recommend a 1s delay
    conf_write(id); // Save the settings
}
```

1.6 Save Motor Settings (conf write)

- **Description**:
  1. Save the motor settings in RAM to flash.
  2. It is recommended to power cycle the motor after using this command.

Code:

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

1.7 Periodic Return of Motor Status Data

- **Description**:
  1. Periodically return motor position, speed, and torque data.
  2. The return data format is the same as that obtained using the `0x17, 0x01` command.
  3. The period unit is ms.
  4. The minimum period is 1ms.
  5. To stop periodic data return, set the period to 0 or power off the motor.

Code:

```
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. Sample Functions

2.1 Normal Mode

2.1.1 Position Control

```
/**
* @brief Position control
* @param id Motor ID
* @param pos Position: Unit 0.0001 revolutions, e.g., pos = 5000 means rotating
 to 0.5 revolutions.
* @param tqe Torque
*/
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 Speed Control

```
/**
* @brief Speed control
* @param id Motor ID
* @param vel Speed: Unit 0.00025 revolutions per second, e.g., val = 1000 means
 0.25 revolutions per second.
* @param tqe Torque
*/
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);
}
```

2.2 Torque Mode

```
/**
* @brief Torque mode
* @param id Motor ID
* @param tqe Torque
*/
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);
}
```

2.3 Cooperative Control Mode

```
/**
* @brief Motor position-speed-feedforward torque (maximum torque) control, int
16 type
* @param id Motor ID
* @param pos Position: Unit 0.0001 revolutions, e.g., pos = 5000 means rotating
 to 0.5 revolutions.
* @param val Speed: Unit 0.00025 revolutions per second, e.g., val = 1000 means
 0.25 revolutions per second.
* @param tqe Maximum torque
*/
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);
}
```

2.4 Brake

- **Description**:
  1. Apply motor brake, maintaining position-holding force.

Code:

```
/**
* @brief Apply motor brake
* @param fdcanHandle &hfdcanx
* @param id Motor 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));
}
```

2.5 Stop

- **Description**:
  1. Stop the motor, losing position-holding force.
  2. Note: The motor must be stopped before resetting the zero position; otherwise, it will be invalid .

Code:

```
/**
* @brief Stop the motor (note: reset zero position after stopping, or it will
 be invalid)
* @param fdcanHandle &hfdcanx
* @param id Motor 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. Common Type (Unit) Descriptions

3.1 Current (A)

Data Type	Actual (A)

int8	1
int16	0.1
int32	0.001
float	1

3.2 Voltage (V)

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

3.3 Torque (Nm)

- **Real torque = k * tqe + d**

3.3.1 5046 Torque (Nm)

Data Type	Slope (k)	Offset (d)
int16	0.005397	-0.455107
int32	0.000528	-0.414526
float	0.533654	-0.519366

3.3.2 4538 Torque (Nm)

Data Type	Slope (k)	Offset (d)
int16	0.004587	-0.290788
int32	0.000445	-0.234668

float	0.493835	-0.473398

3.3.3 5047 (Bipolar, 36 Gear Ratio) Torque (Nm)

Data Type	Slope (k)	Offset (d)
int16	0.004563	-0.493257
int32	0.000462	-0.512253
float	0.465293	-0.554848

3.3.4 5047 (Unipolar, 9 Gear Ratio) Torque (Nm)

Data Type	Slope (k)	Offset (d)
int16	0.005332	-0.072956
int32	0.000533	-0.034809
float	0.547474	-0.150232

3.4 Temperature (°C)

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

3.5 Time (s)

LSB	Actual (s)
1	0.01

1	0.001
1	0.000001
1	1

3.6 Position (Revolutions)

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

3.7 Speed (Revolutions/Sec)

Data Type	LSB	Actual (Revolutions/Sec)
int8	1	0.01
int16	1	0.00025
int32	1	0.00001
float	1	1

3.8 Acceleration (Revolutions/Sec^2)

Data Type	LSB	Actual (Revolutions/Sec^2)
int8	1	0.05
int16	1	0.001
int32	1	0.00001

float	1	1

3.9 PWM Scale (No Unit)

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

3.10 Kp, Kd Scale (No Unit)

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