

3.2 FDCAN 例程详细说明

1. 介绍

这是一个用于控制高擎机电电机的软件接口包。

通过这个接口包，用户可以方便地与高擎机电的电机进行通信，获取电机的状态信息，并控制电机的动作。

2. 软件结构介绍

- **App: 硬件**

- `led`：负责 LED 灯的控制。
- `my_fdcan`：与 FDCAN 硬件相关的功能实现。

- **Src: 应用**

- `convert`：负责单位转换，包括电机的位置、速度、加速度、力矩、电压、电流、PID 以及转 (r)、弧度 (rad)、度 (°) 等单位的转换。
- `livelybot_fdcan`：实现电机底层协议，处理原始数据。
- `motor_control`：用于单电机控制，支持 16 种电机控制模式。
- `motor_many`：用于一拖多模式的电机控制，支持 12 种电机控制模式。
- `motor_config`：负责电机设置相关功能，包括带反馈的电机零位重置和电机设置保存。
- `motor`：包含 FDCAN 接口通道映射、电机类型定义以及电机返回信息解析（包括 FDCAN 中断函数 `HAL_FDCAN_RxFifo0Callback` 的实现），**移植和使用的時候主要就是修改这个部分。**

- **test: 测试**

- `test_motor`：包含单电机函数使用例程和一些简单的控制。
- `test_motor_many`：包含一拖多模式的函数使用例程和一些简单的控制。

3. 移植和配置说明

- 如果没有移植需求可直接跳到 3.2 配置 部分。

- 操作如下

- 将 `App` 文件夹下的 `my_fdcan` 文件夹和 `Src` 文件夹下的 `convert`、`livelybot_fdcan`、`motor_control`、`motor_many`、`motor_config` 和 `motor` 5 个文件夹复制到新工程。

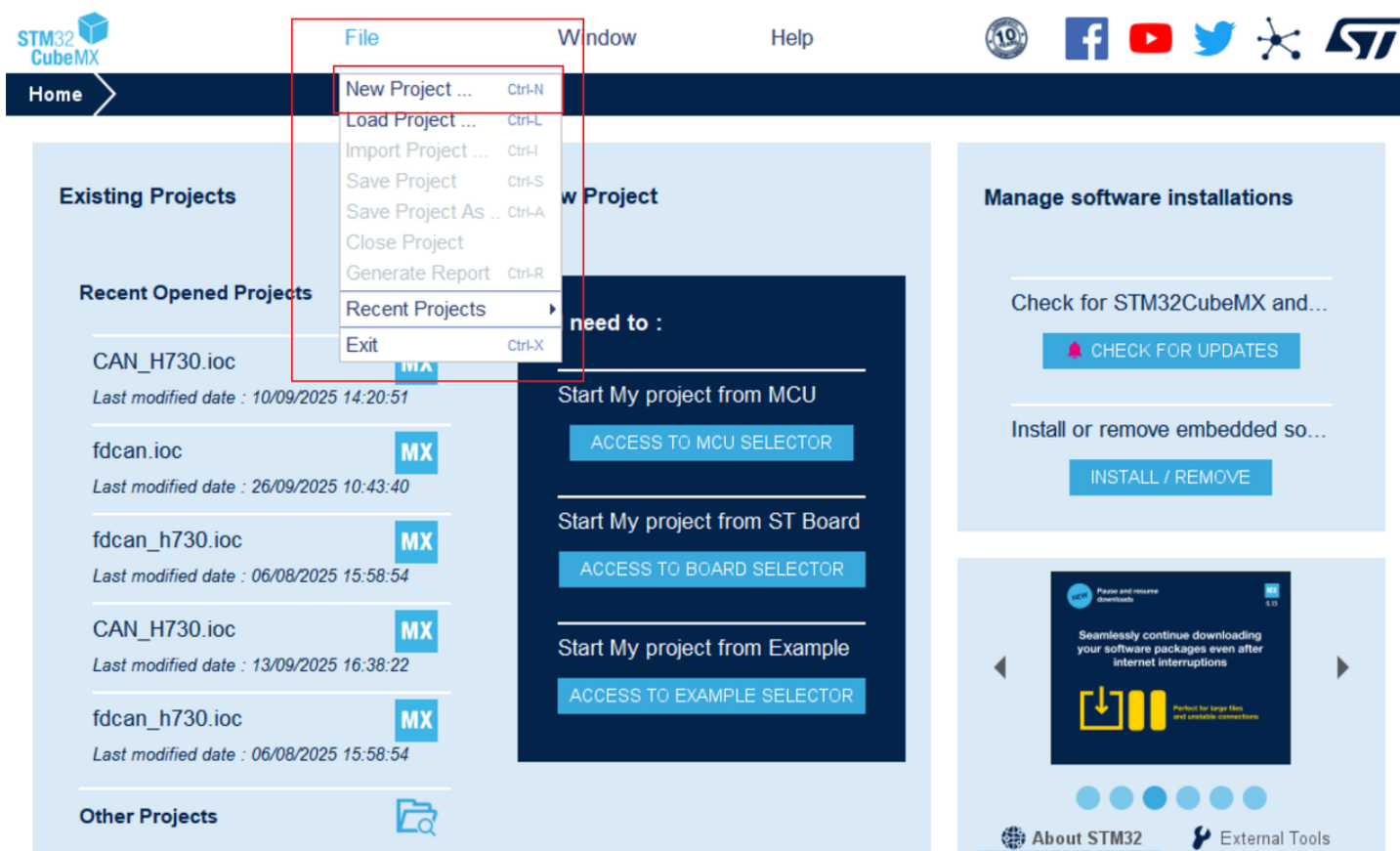
- 将 `my_fdcan`、`convert`、`livelybot_fdcan`、`motor_control`、`motor_many`、`motor_config` 和 `motor` 6 个文件夹都加入到头文件路径。

3.1 移植指南

具体移植操作如下

3.1.1 新建工程

1. 创建项目：打开 STM32CubeMX，点击 `New Project`。
2. 选择芯片：在芯片选择器中，输入您使用的单片机型号（例程使用 STM32H730VBT6），双击选中该型号。
3. 关键系统配置：在 `Pinout & Configuration` 选项卡中，进入 `System Core` -> `SYS` 菜单：
 - 将 `Debug` 选项设置为 `Serial Wire`。此处是为了通过 ST-Link 等进行调试和烧录。



MCU/MPU Selector Board Selector Example Selector Cross Selector

MCU/MPU Filters

Commercial Part Number: STM32H730VBT6

PRODUCT INFO

- Segment
- Series
- Line
- Marketing Status
- Price
- Package
- Core
- Coprocessor

MEMORY

- Flash = 128 (kBytes)
- EEPROM = 0 (Bytes)

STM32H7 Series

STM32H730VBT6

High-performance and DSP with DP-FPU, Arm Cortex-M7 MCU with 128 Kbytes Flash, 564 Kbytes RAM, 550 MHz CPU, L1 cache, external memory interface, subset of peripherals including a Crypto accelerator

Unit Price for 10kU (US\$): 4.5658

LQFP 100 14x14x1.4 mm

STM32H730xB devices are based on the high-performance Arm® Cortex®-M7 32-bit RISC core operating at up to 550 MHz. The Cortex®-M7 core features a floating-point unit (FPU) which supports Arm® double-precision (IEEE 754

MCUs/MPUs List: 1 item

Commercial...	Part No	Reference	Marketin...	Unit Price for ...	Bo...	Package	Flash	RAM	I/O	Frequ...
★	STM32H730V...	STM32H7...	STM32H730...	Active	4.5658	LQFP 100 14x14x1.4 ...	128 kB...	564 kB...	82	550 M...

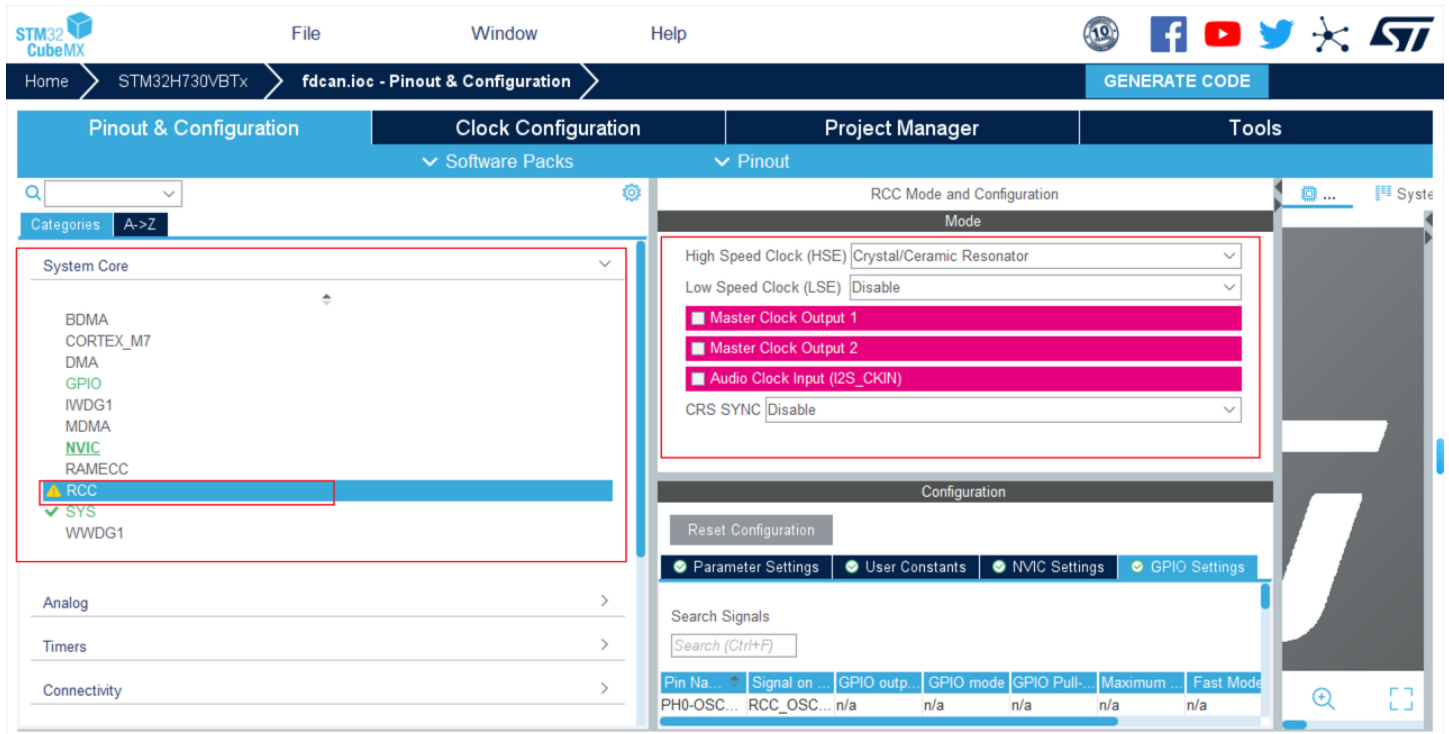
3.1.2 时钟与 FDCAN 配置

本部分是移植成功的关键，请严格按照步骤操作。

3.1.2.1 配置外部高速时钟 (HSE)

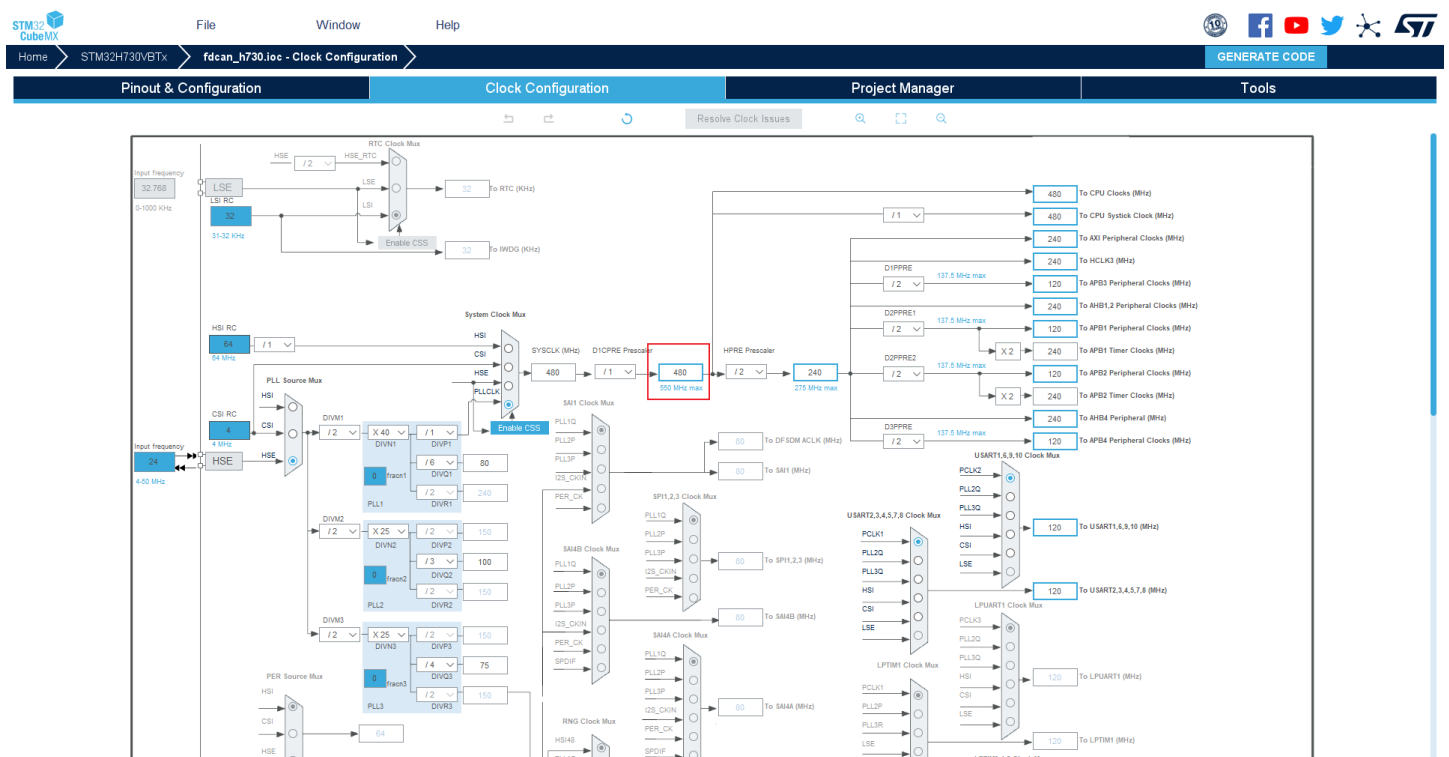
为了确保系统时钟准确稳定，需要配置外部高速时钟源。

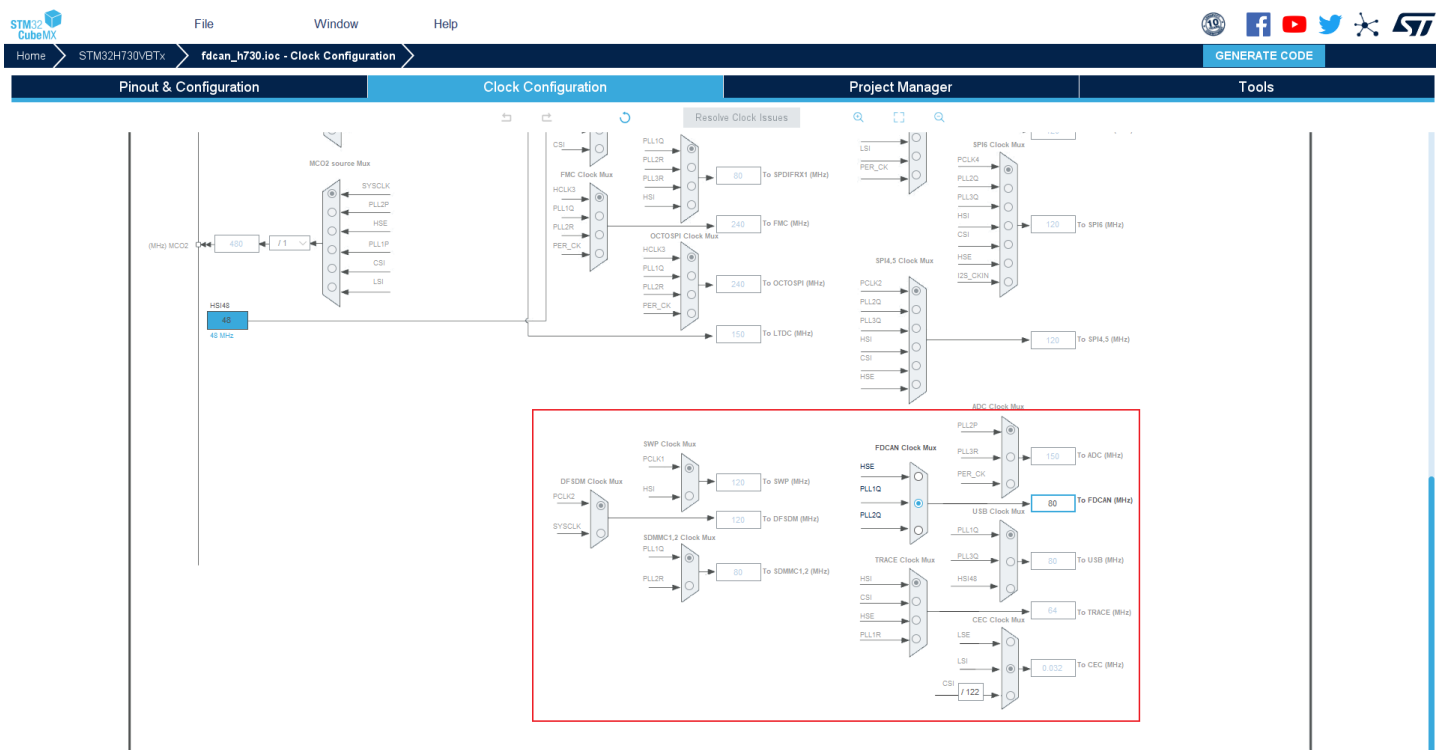
1. 在左侧分类中找到 System Core > RCC（复位和时钟控制）。
2. 在右侧的 RCC Mode and Configuration 模式下，找到 High Speed Clock (HSE) 选项。
3. 将其从默认的 "Disable" 修改为 Crystal/Ceramic Resonator。
 - 作用：这告诉单片机，我们将使用外接的晶振作为高速时钟源，而不是使用芯片内部的 RC 振荡器。外部晶振能提供更精确的时钟频率，是整个系统（包括 FDCAN 通信波特率）稳定的基础。



3.1.2.2 配置时钟树

1. 进入 **Clock Configuration** 选项卡。
2. 根据芯片情况设置主频率，此处配置为 480MHz。
3. 找到并确认 FDCAN 的时钟源频率为 80MHz。此频率是计算 FDCAN 通信波特率的基础，必须相同。





3.1.2.3 配置 FDCAN

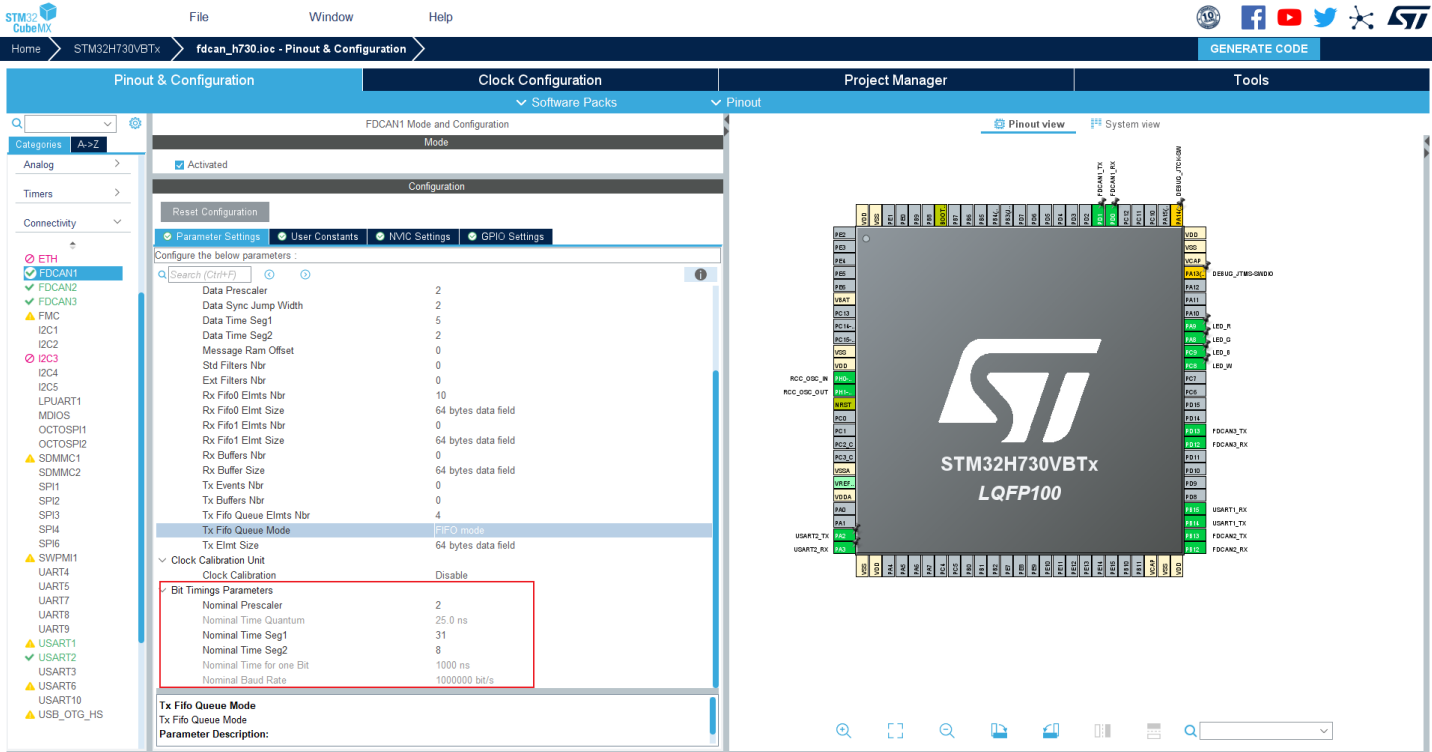
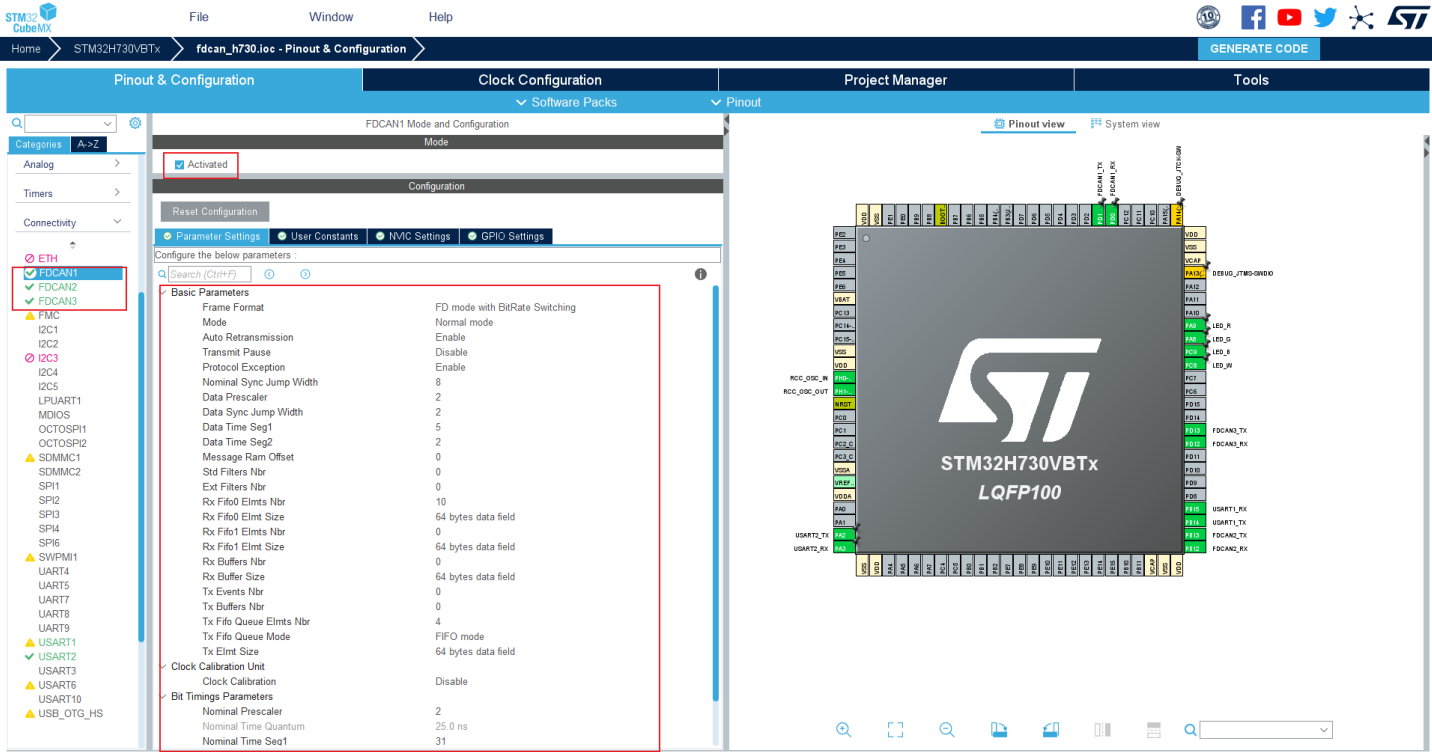
1. 在 **Connectivity** 菜单下，启用您计划使用的所有 FDCAN 模块（如 FDCAN1, FDCAN2, FDCAN3）。
2. 对每个启用的 FDCAN 模块，根据下表进行参数配置（此配置基于 80MHz 时钟，可实现仲裁段 1Mbps，数据段 5Mbps）：

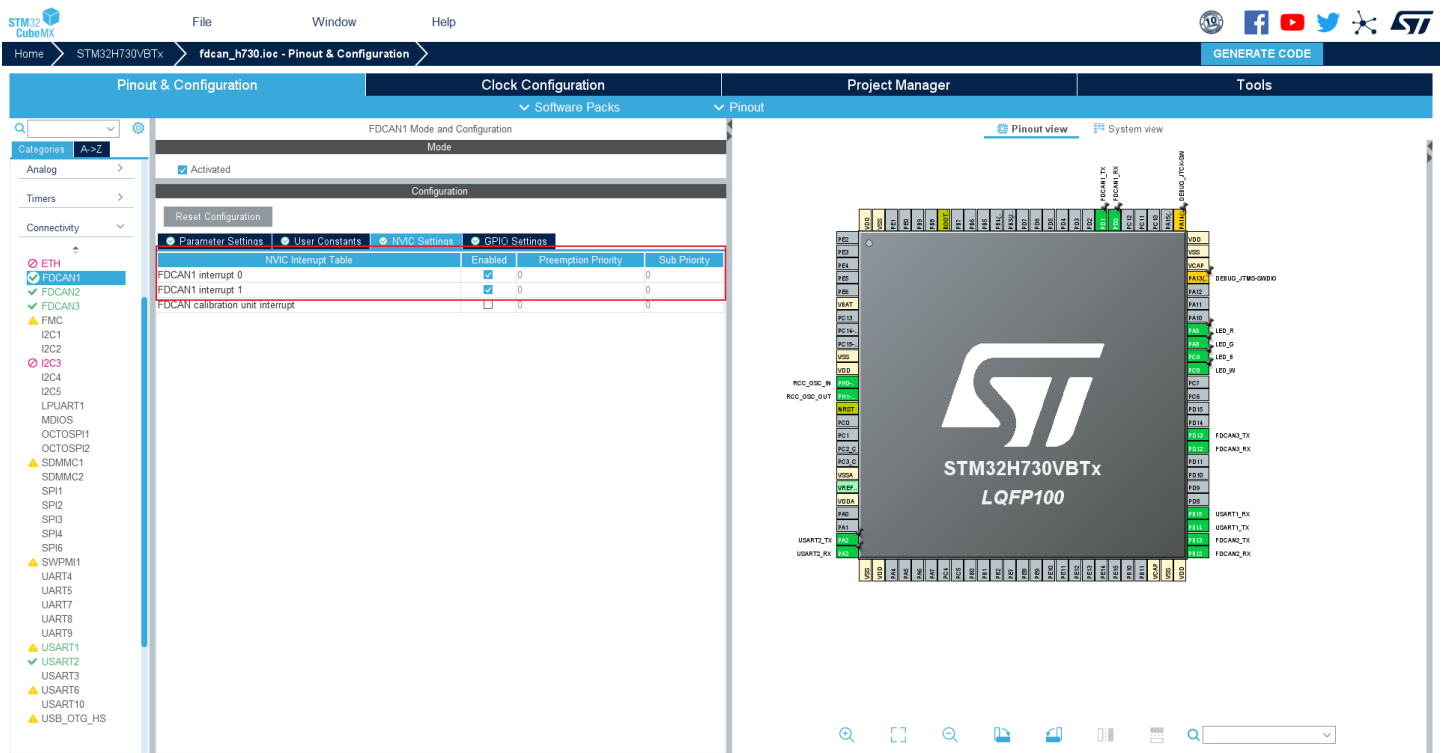
参数分类	参数名称	当前值	说明
	Mode	Normal mode	模式：Normal / FD
	Auto Retransmission	Enable	自动重传
	Transmit Pause	Disable	发送暂停
	Protocol Exception	Enable	协议异常处理
	Nominal Sync Jump Width	8	仲裁段同步跳转宽度
	Data Prescaler	2	数据段预分频
	Data Sync Jump Width	2	数据段同步跳转宽度
	Data Time Seq1	5	数据段相位缓冲段1

基本参数	~~~~~		
	Data Time Seq2	2	数据段相位缓冲段2
	Std Filters Nbr	0	标准ID过滤器数量
	Ext Filters Nbr	10	扩展ID过滤器数量
	Rx Fifo0 Elmts Nbr	0	Rx FIFO0 元素数量
	Rx Fifo0 Elmt Size	64 bytes	FIFO0 元素大小
	Rx Fifo1 Elmts Nbr	0	Rx FIFO1 元素数量
	Rx Fifo1 Elmt Size	64 bytes	FIFO1 元素大小
	Rx Buffers Nbr	0	Rx 缓冲区数量
	Rx Buffer Size	64 bytes	Rx 缓冲区大小
	Tx Events Nbr	0	Tx 事件数量
	Tx Buffers Nbr	4	Tx 缓冲区数量
	Tx Fifo Queue Elmts Nbr	0	Tx FIFO 队列元素数量
	Tx Fifo Queue Mode	FIFO mode	Tx 队列模式
	Tx Elmt Size	64 bytes	Tx 元素大小
时钟校准	Clock Calibration	Disable	时钟校准开关
位定时参数	Nominal Prescaler	2	仲裁段预分频
	Nominal Time Quantum	25.0 ns	仲裁段时间量子
	Nominal Time Seq1	31	仲裁段相位缓冲段1
	Nominal Time Seq2	8	仲裁段相位缓冲段2

	Time Seq2		
Nominal Time for one Bit	4000 ns	仲裁段每比特时间	

3. 开启中断：对每个FDCAN 模块，进入 **NVIC Settings** 标签页，勾选使能 **FDCANx Interrupt 0**（x 为 CAN 编号）。这是接收电机数据的必要条件，至少要开启两个FDCAN通道的中断。





4. 可在程序中的 `fdcan.c` 查看配置情况

代码块

```

1      hfdcan1.Instance = FDCAN1;
2      hfdcan1.Init.FrameFormat = FDCAN_FRAME_FD_BRS;
3      hfdcan1.Init.Mode = FDCAN_MODE_NORMAL;
4      hfdcan1.Init.AutoRetransmission = ENABLE;
5      hfdcan1.Init.TransmitPause = DISABLE;
6      hfdcan1.Init.ProtocolException = ENABLE;
7      hfdcan1.Init.NominalPrescaler = 2;
8      hfdcan1.Init.NominalSyncJumpWidth = 8;
9      hfdcan1.Init.NominalTimeSeg1 = 31;
10     hfdcan1.Init.NominalTimeSeg2 = 8;
11     hfdcan1.Init.DataPrescaler = 2;
12     hfdcan1.Init.DataSyncJumpWidth = 2;
13     hfdcan1.Init.DataTimeSeg1 = 5;
14     hfdcan1.Init.DataTimeSeg2 = 2;
15     hfdcan1.Init.MessageRAMOffset = 0;
16     hfdcan1.Init.StdFiltersNbr = 0;
17     hfdcan1.Init.ExtFiltersNbr = 0;
18     hfdcan1.Init.RxFifo0ElmtsNbr = 10;
19     hfdcan1.Init.RxFifo0ElmtSize = FDCAN_DATA_BYTES_64;
20     hfdcan1.Init.RxFifo1ElmtsNbr = 0;
21     hfdcan1.Init.RxFifo1ElmtSize = FDCAN_DATA_BYTES_64;
22     hfdcan1.Init.RxBuffersNbr = 0;
23     hfdcan1.Init.RxBufferSize = FDCAN_DATA_BYTES_64;
24     hfdcan1.Init.TxEventsNbr = 0;
25     hfdcan1.Init.TxBuffersNbr = 0;

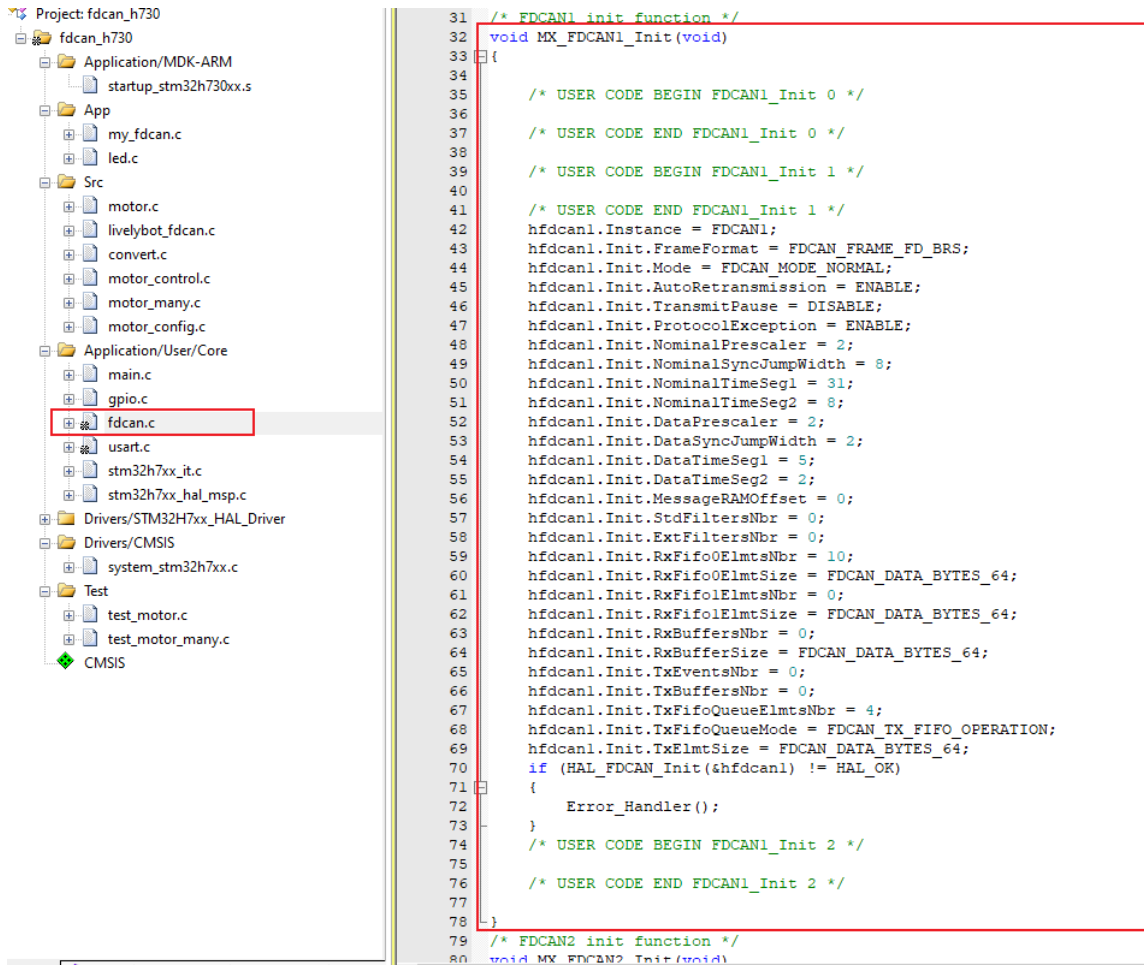
```



```

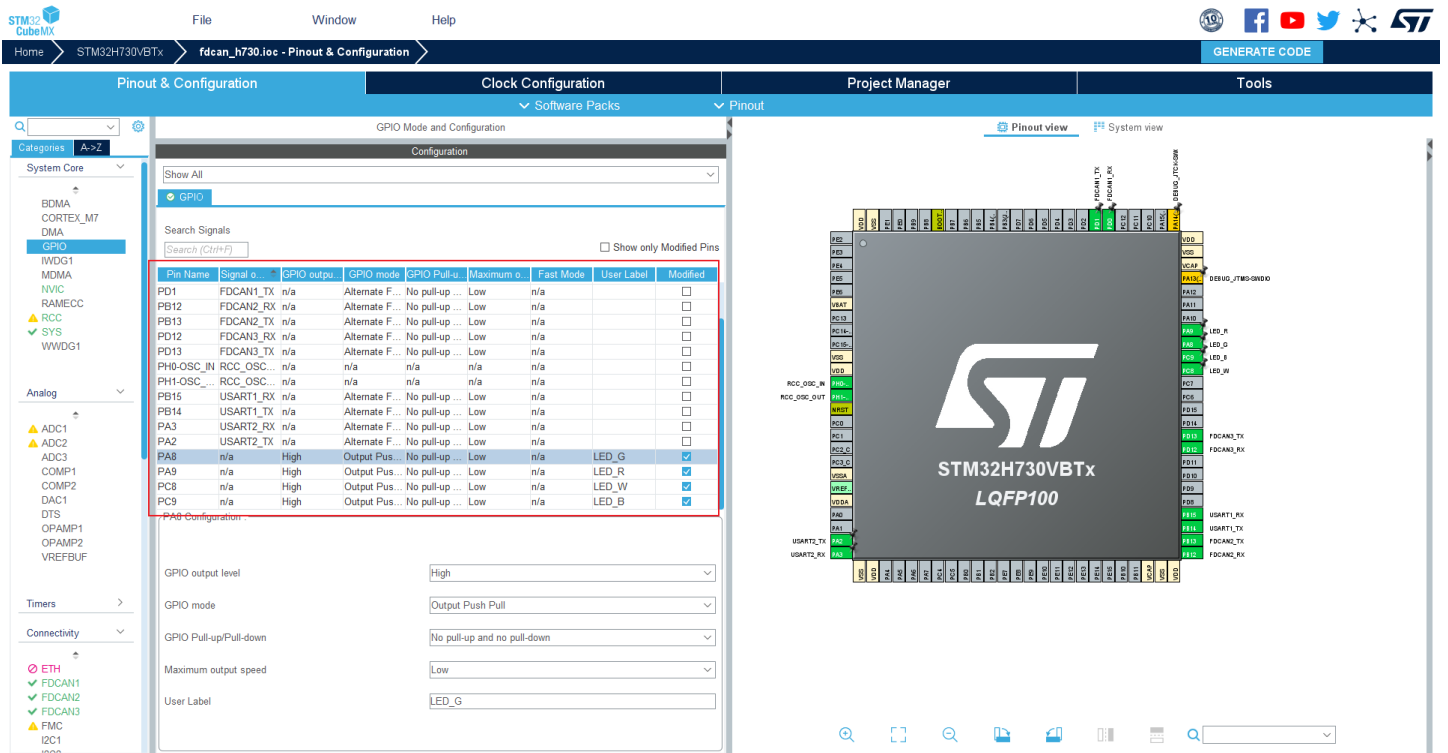
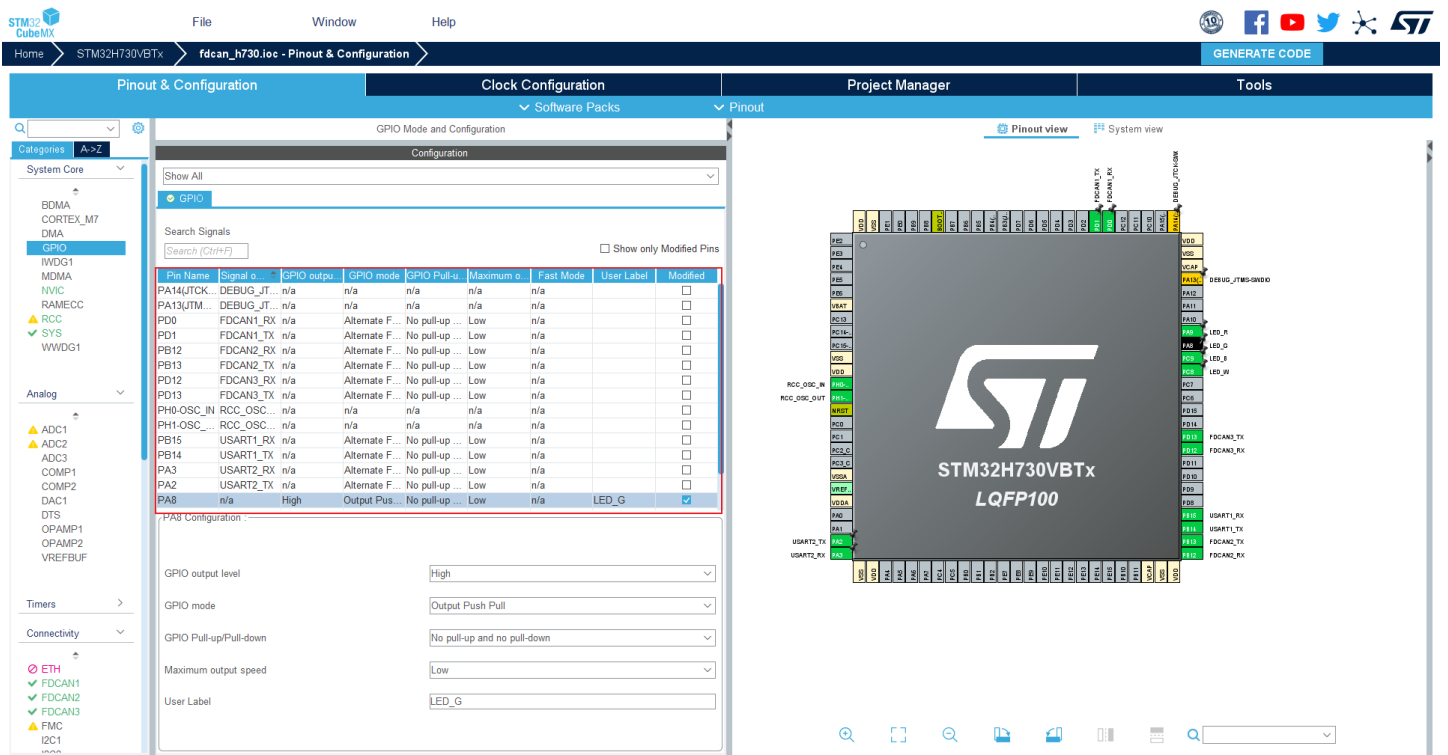
26     hfdcan1.Init.TxFifoQueueElmtsNbr = 4;
27     hfdcan1.Init.TxFifoQueueMode = FDCAN_TX_FIFO_OPERATION;
28     hfdcan1.Init.TxEltSize = FDCAN_DATA_BYTES_64;

```



3.1.2.4 配置其他外设（可选）

可根据您的项目需求，配置串口、LED GPIO 等外设。例程中开启了相关功能供参考。



3.1.3 生成代码与文件移植

3.1.3.1 生成代码：

1. 配置生成选项

- 进入 **Project Manager** -> **Code Generator** 页面进行关键设置：
 - 取消勾选 **Generate peripheral initialization as a pair of '.c/.h' files per peripheral**。此设置能将所有外设初始化代码整合到 **main.c** 中，极大简化项目结构，避免产生过多零散文件。

- 务必勾选 **Keep User Code when re-generating**。这是最重要的设置，可以保护您在特定注释标签（`/* USER CODE BEGIN */`）内编写的代码在重新生成时不被覆盖。
- 进入 **Project Manager** -> **Project** 选项卡，为项目命名并选择保存路径。
 - 在 **Toolchain / IDE** 选项中选择您使用的 IDE（如 **MDK-ARM**）。

2. 生成项目文件

完成设置后，回到 **Project** 页面，检查项目路径和 IDE 选项无误，点击 **GENERATE CODE** 按钮生成完整的工程代码。

The screenshot displays the STM32CubeMX Project Manager interface. The top navigation bar includes 'File', 'Window', and 'Help' menus, along with social media icons and the ST logo. The main window is titled 'fdcan.ioc - Project Manager' and features a 'GENERATE CODE' button in the top right corner. The interface is divided into four tabs: 'Pinout & Configuration', 'Clock Configuration', 'Project Manager', and 'Tools'. The 'Project Manager' tab is active, showing a sidebar with 'Project', 'Code Generator', and 'Advanced Settings'. The 'Project' section is expanded, revealing the 'Project Settings' form. This form includes fields for 'Project Name' (fdcan), 'Project Location' (F:\1\fdcan), 'Application Structure' (Advanced), 'Toolchain Folder Location' (F:\1\fdcan\fdcan\), and 'Toolchain / IDE' (MDK-ARM). It also has checkboxes for 'Do not generate the main()' and 'Generate Under Root', and a 'Min Version' dropdown set to V5.32. Below this, the 'Linker Settings' section shows 'Minimum Heap Size' (0x200) and 'Minimum Stack Size' (0x400). The 'Thread-safe Settings' section includes a checkbox for 'Enable multi-threaded support' and a 'Thread-safe Locking Strategy' dropdown. The 'Mcu and Firmware Package' section shows 'Mcu Reference' (STM32H730VBTx), 'Firmware Package Name and Version' (STM32Cube FW_H7 V1.11.2), and a checked checkbox for 'Use latest available version'. A 'Use Default Firmware Location' checkbox is also present at the bottom.

Pinout & Configuration	Clock Configuration	Project Manager	Tools
Project	STM32Cube MCU packages and embedded software packs— <input type="radio"/> Copy all used libraries into the project folder <input checked="" type="radio"/> Copy only the necessary library files <input type="radio"/> Add necessary library files as reference in the toolchain project configuration file		
Code Generator	Generated files— <input checked="" type="checkbox"/> Generate peripheral initialization as a pair of '.c/.h' files per peripheral <input type="checkbox"/> Backup previously generated files when re-generating <input checked="" type="checkbox"/> Keep User Code when re-generating <input checked="" type="checkbox"/> Delete previously generated files when not re-generated		
Advanced Settings	HAL Settings— <input type="checkbox"/> Set all free pins as analog (to optimize the power consumption) <input type="checkbox"/> Enable Full Assert		
	Template Settings— Select a template to generate customized code Settings...		

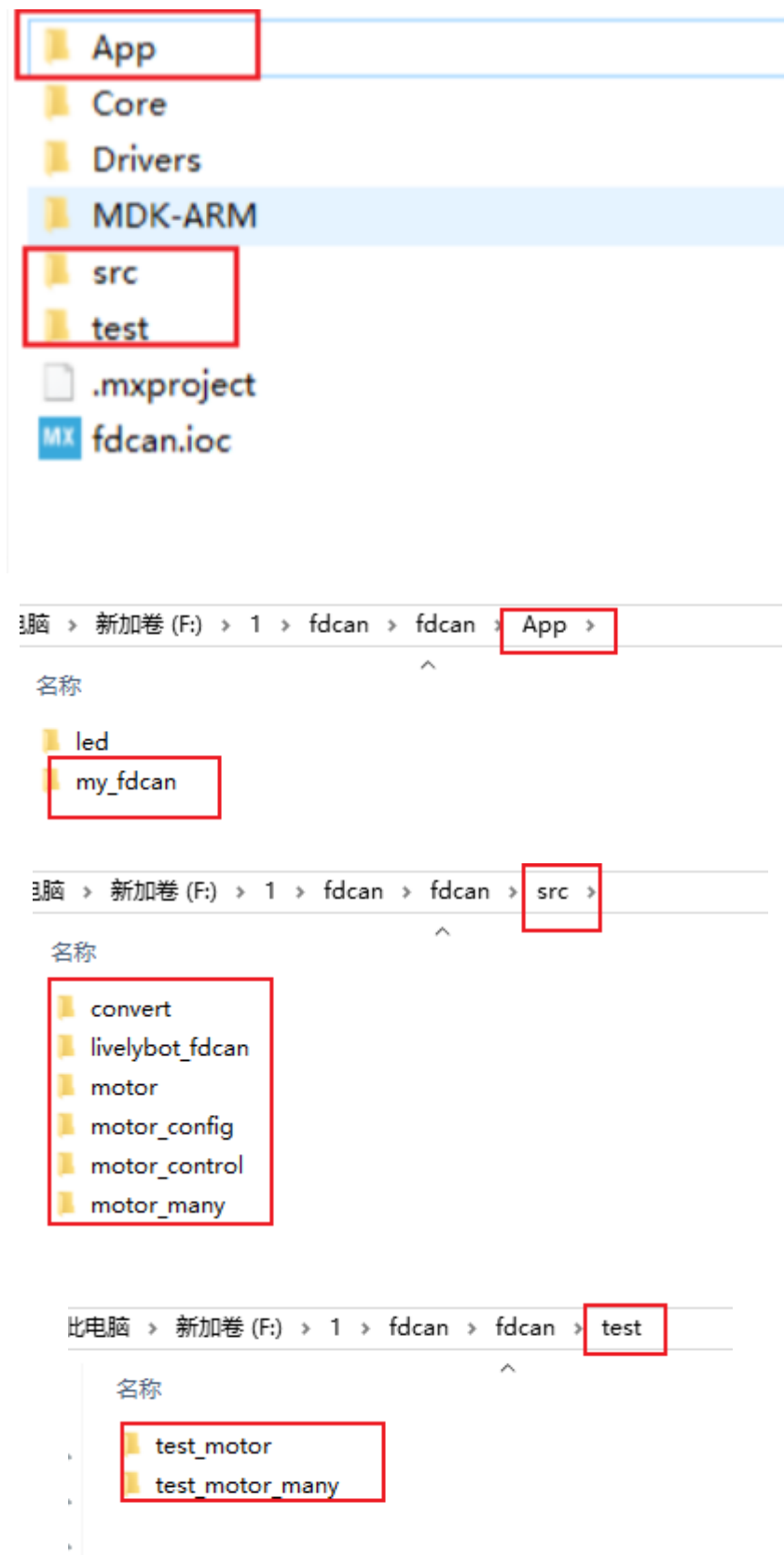
3.1.3.2 文件移植

- 将以下文件夹从原始软件包目录完整复制到您新生成的 STM32 工程根目录下：

代码块

```
1  |— App/
2  |   |— my_fdcan/           # 硬件抽象层 (HAL)
3  |— Src/
4  |   |— convert/           # 单位转换
5  |   |— livelybot_fdcan/   # 底层协议处理
6  |   |— motor_control/     # 单电机控制
7  |   |— motor_many/        # 一拖多控制
8  |   |— motor_config/      # 电机设置 (如零位重置)
9  |   |— motor/             # 【核心】电机类型定义、通信映射、数据解析
10 |   |— test/               # (建议复制, 供参考)
11 |       |— test_motor/     # 单电机使用例程
12 |       |— test_motor_many/ # 一拖多使用例程
```

- (建议复制) 将 `test/test_motor` 和 `test/test_motor_many` 文件夹也复制到工程中，作为使用范例参考。



3.1.3.3 添加头文件路径：

- 在 IDE（如 Keil MDK）中打开工程，进入 Options for Target -> C/C++ 选项卡。
- 在 Include Paths 中，添加以下所有路径（请仔细核对，确保路径正确）：

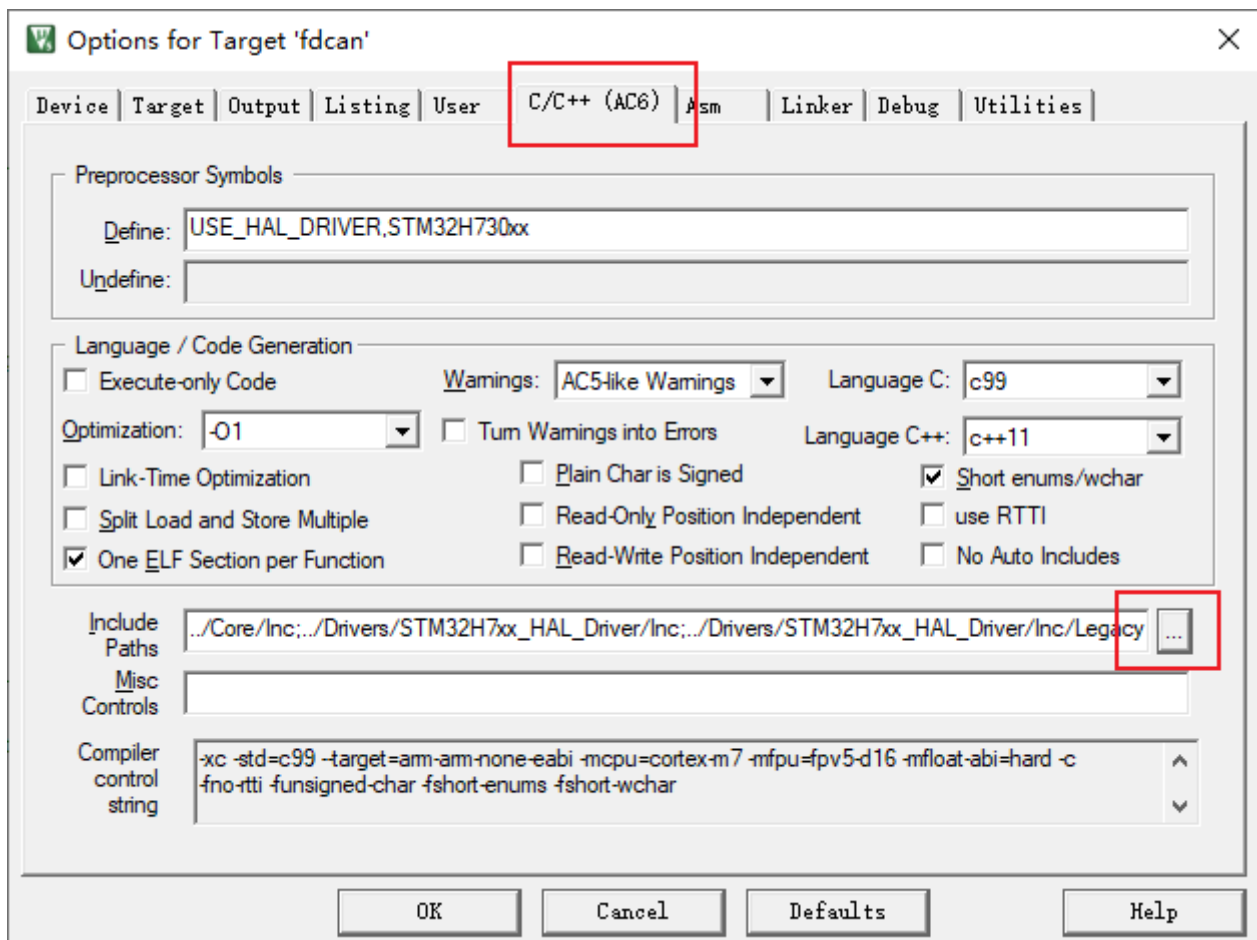
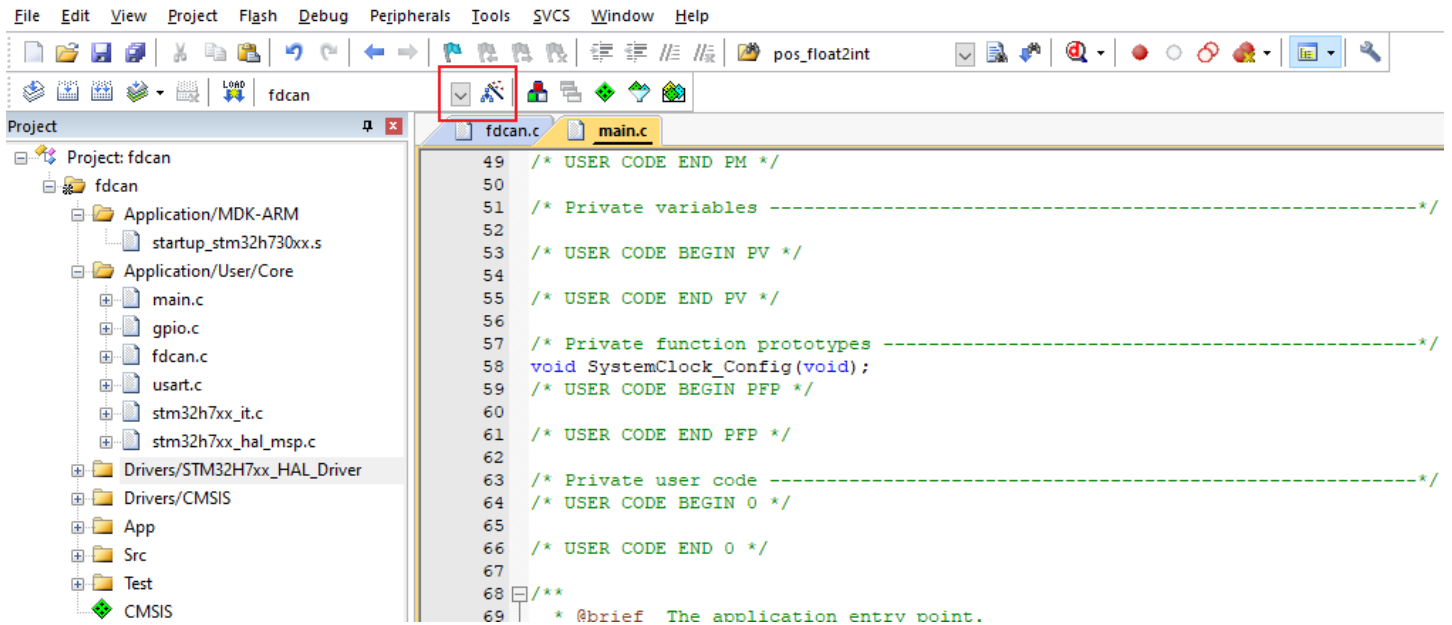
代码块

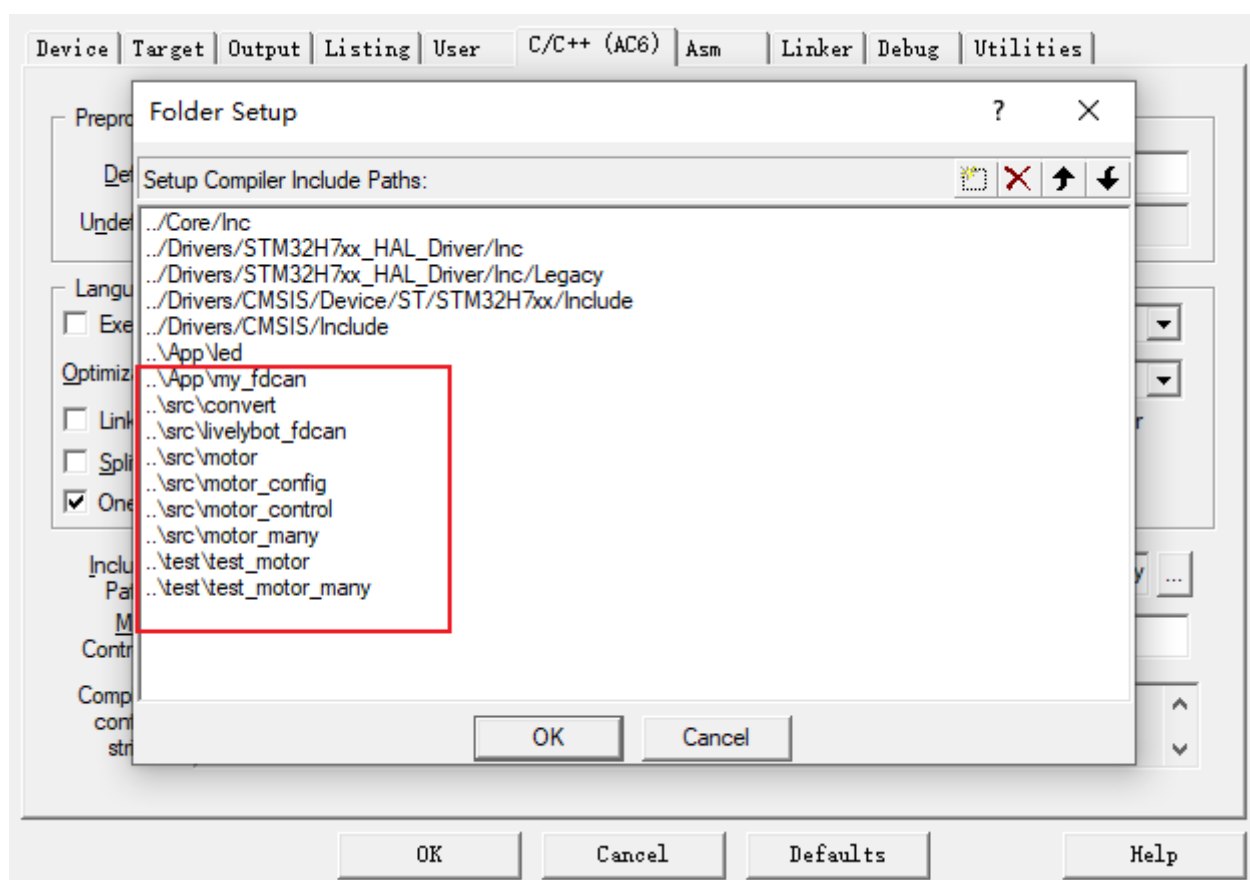
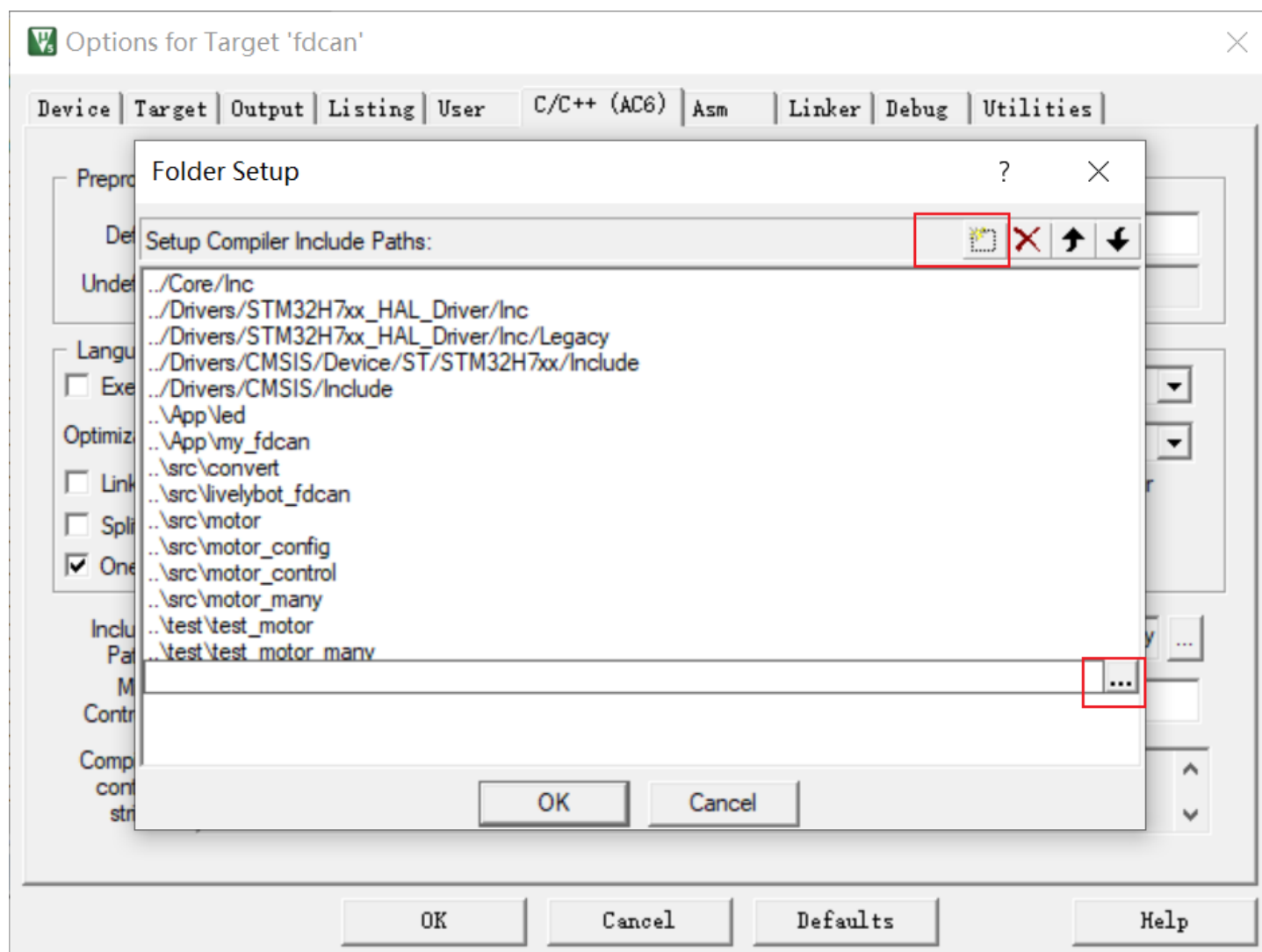
```
1 .\App\my_fdcan
2 .\Src\convert
```

```

3  .\Src\livelybot_fdcan
4  .\Src\motor_config
5  .\Src\motor_control
6  .\Src\motor_many
7  .\Src\motor
8  .\test\test_motor
9  .\test\test_motor_many

```



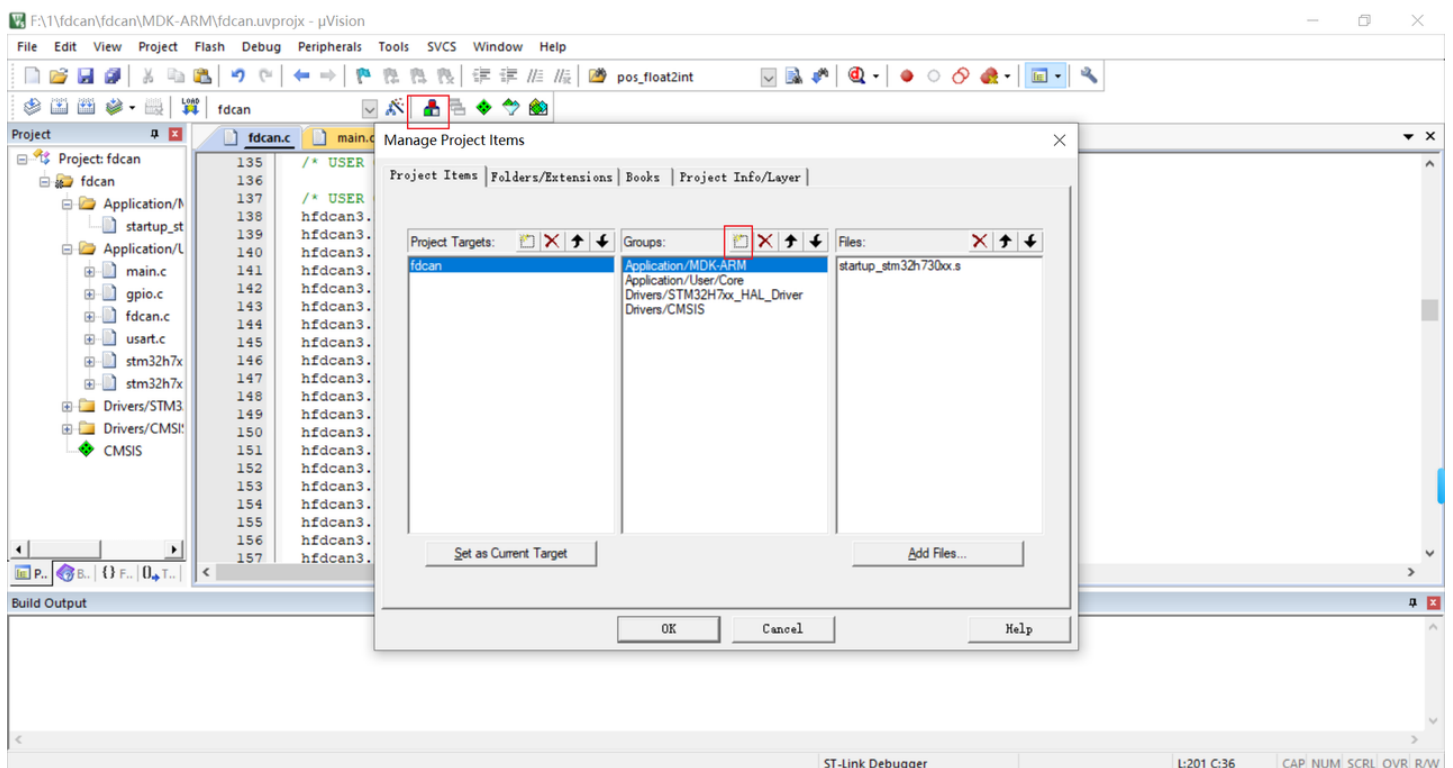


3.1.3.4 添加源文件到工程

1. 在 IDE 的项目管理窗口中，创建三个新的文件组（Group），分别命名为 `App`，`Src`，和 `Test`。
2. 将对应的源文件（`.c` 文件）拖放或添加到相应的组中：
 - `App` 组：添加 `my_fdcan.c`、`led.c`。
 - `Src` 组：添加 `motor.c`，`livelybot_fdcan.c`，`convert.c`，`motor_control.c`，`motor_many.c`，`motor_config.c`。
 - `Test` 组（可选）：添加 `test_motor.c`，`test_motor_many.c` 作为参考例程。
3. 将对应的源文件添加到各自的组中。添加成功后，您可以在 IDE 的项目管理器界面中清晰地看到完整的、按功能模块分组的工程文件结构。

代码块




```
1 my_fdcan.c
2 convert.c
3 livelybot_fdcan.c
4 motor_control.c
5 motor_many.c
6 motor_config.c
7 motor.c
8 （可选）test_motor.c 和 test_motor_many.c
```



Manage Project Items



Project Items | Folders/Extensions | Books | Project Info/Layer




Project Targets:    

fdcan

Set as Current Target

Groups:    

Application/MDK-ARM
Application/User/Core
Drivers/STM32H7xx_HAL_Driver
Drivers/CMSIS
App
Src
Test

Files:   





led.c
my_fdcan.c

Add Files...





OK

Cancel




Help

Project Targets:    

fdcan

Groups:    

Application/MDK-ARM
Application/User/Core
Drivers/STM32H7xx_HAL_Driver
Drivers/CMSIS
App
Src
Test

Files:   

convert.c
livelybot_fdcan.c
motor.c
motor_config.c
motor_control.c
motor_many.c

Set as Current Target

Add Files...

OK





Cancel

Help





Manage Project Items






Project Items | Folders/Extensions | Books | Project Info/Layer

Project Targets:    

fdcan

Groups:    

Application/MDK-ARM
Application/User/Core
Drivers/STM32H7xx_HAL_Driver
Drivers/CMSIS
App
Src
Test

Files:   

test_motor_many.c
test_motor.c

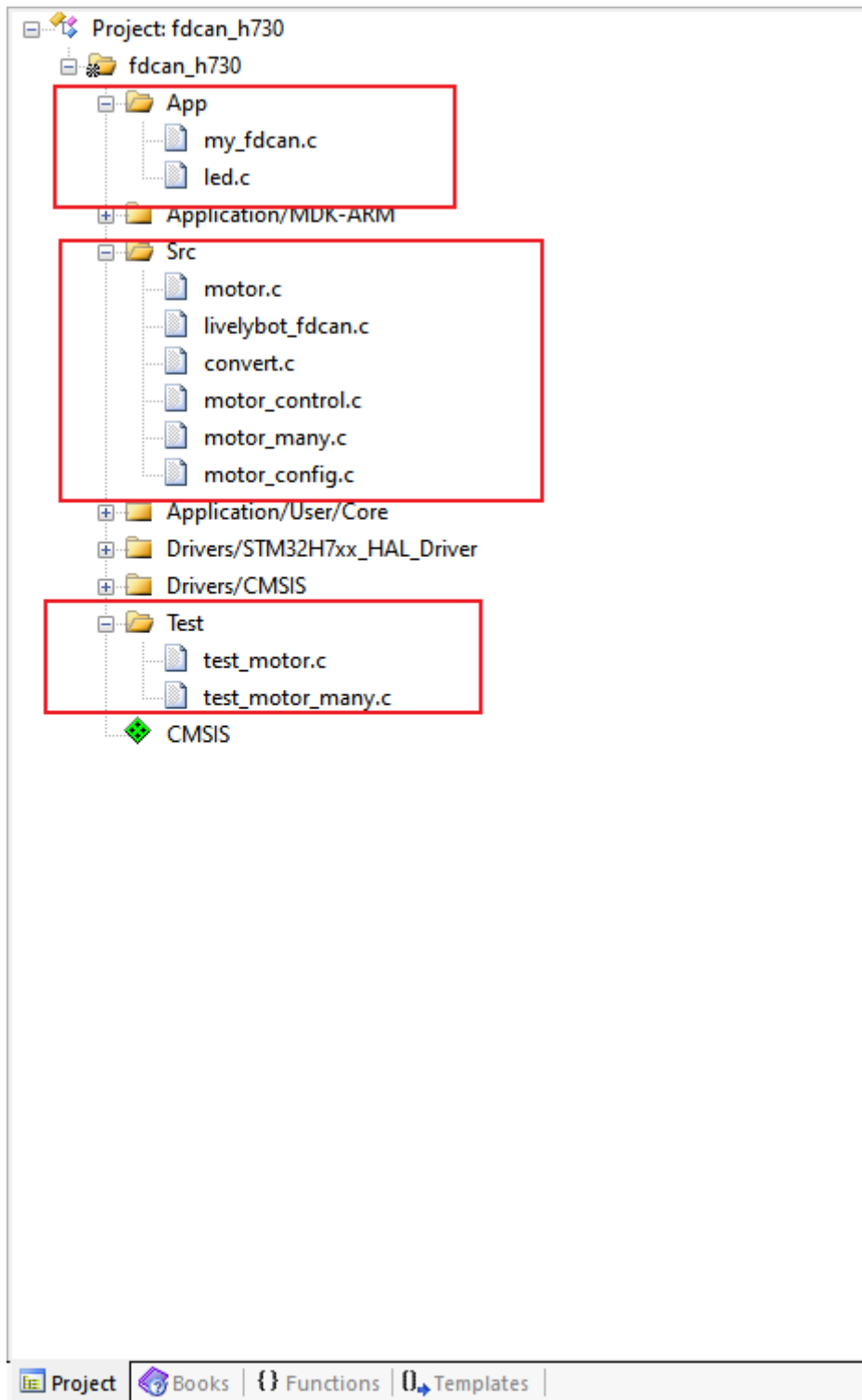
Set as Current Target

Add Files...

OK

Cancel

Help



3.1.3.5 在项目中包含头文件

在您的主程序文件（如 `main.c`）中，包含必要的头文件：

代码块

```
1  #include "motor.h"
2  #include "motor_control.h"
3  #include "motor_many.h"
4  #include "motor_config.h"
5
6  #include "test_motor.h"
7  #include "test_motor_many.h"
```

3.1.3.6 修改报错宏：

1. 在 `convert.h` 文件中，有用于检查某些配置或使用错误的宏，默认使用 `led_toggle_err` 所有 LED 同时闪烁报错，可根据需要自行修改。
2. `#include "led.h"` 这里引入 led 头文件只为使用 `led_toggle_err` 函数，如不需要直接删除即可。

代码块

```
1  #include "led.h"
2  #ifdef LED_ERR_FLAG // 这个宏定义在 led.h 中
3  #define MOTOR_ERR led_toggle_err // 所有 led 闪烁
4  #else
5  static inline void MOTOR_ERR(void) {}
6  #endif
```

3. 编辑成功，移植完成。

3.2 配置（第一次使用时必须配置）

此步骤根据您的实际硬件连接进行配置。

3.2.1 修改 `motor.h` 中的宏定义

1. 修改宏定义 `MOTOR_MAX_NUM`，根据单个 CAN 通道需连接的最大电机数量修改。
 - 假设需要使用三个 CAN 通道，其中 CAN1 接 1 个电机，CAN2 接 2 个电机，CAN3 接 5 个电机，宏定义 `MOTOR_MAX_NUM` 修改为 5（以最大使用电机数设置）。

代码块

```
1  #define MOTOR_MAX_NUM 5
```

2. 修改宏定义 `MOTOR_PORT_NUM`，需要使用个 CAN 通道用于电机通讯就改成多少。
 - 假设 STM32G474 一共有三个 CAN 通道，其中 CAN1 和 CAN3 两个 CAN 通道用于连接电机，宏定义 `MOTOR_PORT_NUM` 修改为 2。

代码块

```
1  #define MOTOR_PORT_NUM 2 // 通道数量
```

3.2.2 修改 `motor_state_port` 结构体数组（配置电机型号）

1. 在 `motor.c` 文件中，定义了 `motor_state_port[MOTOR_PORT_NUM][MOTOR_MAX_NUM]` 结构体数组，需要根据电机型号进行修改，以进行正确的力矩修正。
2. 假设我们一共要连接四个电机，每个通道连接两个电机，分布如下：

1.PORT1:

1.4438_30

2.5047_36

2.PORT2:

1.6056_36

2.5046_20

则 `motor_state_port` 配置如下

代码块

```
1  static motor_state_s motor_state_port[MOTOR_PORT_NUM][MOTOR_MAX_NUM] = // 下标
   + 1 = 电机 ID
2  {
3      { // CAN 通道 PORT1
4          { // ID = 1
5              .model = M4438_30,
6          },
7      }
8      { // ID = 2
9          .model = M5047_36,
10     },
11 },
12
13 { // CAN 通道 PORT2
14     { // ID = 1
15         .model = M6056_36,
16     },
17
18     { // ID = 2
19         .model = M5046_20,
20     }
21 },
22 };
```

3.2.3 修改 PORT、FDCAN、STATE 的映射关系

1. 在 `motor.c` 中，定义了结构体数组 `port_mapping[MOTOR_PORT_NUM]` 用于定义 PORT、FDCAN、STATE 之间的映射关系。
2. 假设我们用到了 CAN2 和 CAN3 两个通道，我们希望定义

- CAN2 映射到 PORT1，其状态数据存储在 motor_state_port[0]。
- CAN3 映射到 PORT2，其状态数据存储在 motor_state_port[1]。

3. 则对应修改如下：

代码块

```
1  const port_mapping_s port_maping[MOTOR_PORT_NUM] = // 通道映射表
2  {
3      {
4          .port = PORT1,
5          .fdcan = &hfdcan2,
6          .state = motor_state_port[0],
7      },
8
9      {
10         .port = PORT2,
11         .fdcan = &hfdcan3,
12         .state = motor_state_port[1],
13     },
14 };
```

3.2.4 修改位置速度的单位

- 位置和速度单位支持：**转 (r)**、**弧度 (rad)**、**或度 (°)**，通过 convert.h 中的宏定义 MOTOR_DATA_TYPE_FLAG 进行切换，默认为**转 (r)**。

1. 如果想使用**转 (r)** 做为位置速度的单位：

代码块

```
1  #define MOTOR_DATA_TYPE_FLAG  TURNS
```

2. 如果想使用**弧度(rad)** 做为位置速度的单位：

代码块

```
1  #define MOTOR_DATA_TYPE_FLAG  RADIAN_2PI
```

3. 如果想使用**角度(°)** 做为位置速度的单位：

代码块

```
1  #define MOTOR_DATA_TYPE_FLAG  ANGLE_360
```

4. 函数示例

4.1 单电机控制模式 `motor_control.c`

- 此文件内，除电机软重启函数 `motor_set_reset` 外，都带有查询电机状态信息功能（解析函数位于 `motor.c` 文件）。
- 此文件内的所有函数都是调用后立即发送对应的 FDCAN 帧，无软件缓存。
- 此文件内的位置速度单位由宏定义 `MOTOR_DATA_TYPE_FLAG` 决定，详情请看修改位置速度的单位。
- 此文件内的控制方式控制频率不能过高，在仲裁段为 1M，数据段为 5M 的情况下，一个 CAN 总线 1ms 内最多控制 3 个电机，即 1KHz 控制频率下最多控制 3 个电机，如需控制更多电机，需降低控制频率或使用一拖多控制模式（一个控制包括发送指令和接收电机状态信息）。
- 各个参数的单位在代码里的函数注释上都有详细说明。
- `TINT16` 对应的是 `int16` 的数据类型，控制只能到 3.2 圈。

4.1.1 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
 - `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
 - `volt` : Q 相电压，单位：伏特 (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);
```

4.1.2 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

- `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
- `cur`：Q 相电流，单位：安培（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);
```

4.1.3 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
 - `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）。

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 data_type_t type, const uint8_t id, const float pos);
```

4.1.4 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
- `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)

代码块

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

4.1.5 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
- `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
- `tqe`: 目标力矩, 单位: 牛·米 (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);
```

4.1.6 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
- `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）

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);
```

4.1.7 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
 - `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）。
 - `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 data_type_t type, const  
   uint8_t id,  
2   const float pos, const float vel, const float tqe);
```

4.1.8 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（rps²）。

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);
```

4.1.9 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（rps²）。

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);
```

4.1.10 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
- `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) 。
- `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. `motor_set_pos_vel_tqe_kp_kd` 函数的位置是不断积分得到的, 所以在低频控制下会有意外情况。
3. 推荐使用运控模式 2 `motor_set_pos_vel_tqe_kp_kd_2` 。

代码块

```
1  /* 不建议使用 */
2  void motor_set_pos_vel_tqe_kp_kd(port_t portx, const data_type_t type, const
   uint8_t id,
3  const float pos, const float vel, const float tqe, const float kp, const float
   kd);
4
5  /* 建议使用 */
6  void motor_set_pos_vel_tqe_kp_kd_2(port_t portx, const data_type_t type, const
   uint8_t id,
7  const float pos, const float vel, const float tqe, const float kp, const float
   kd);
```

4.1.11 运控模式 2

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
 - `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) 。

- `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. 真正的运控模式, 和 `motor_set_pos_vel_tqe_kp_kd` 区别是:
 - `motor_set_pos_vel_tqe_kp_kd` 的位置是不断积分得到的, 在低频控制下容易出现意外情况。
 - `motor_set_pos_vel_tqe_kp_kd_2` 则避免了此问题, 控制更稳定。
2. 该需要设置合适的 `kp` 和 `kd` , 否则控制效果可能较差。

代码块

```
1  /* 不建议使用 */
2  void motor_set_pos_vel_tqe_kp_kd(port_t portx, const data_type_t type, const
   uint8_t id,
3  const float pos, const float vel, const float tqe, const float kp, const float
   kd);
4
5  /* 建议使用 */
6  void motor_set_pos_vel_tqe_kp_kd_2(port_t portx, const data_type_t type, const
   uint8_t id,
7  const float pos, const float vel, const float tqe, const float kp, const float
   kd);
```

4.1.12 查询电机状态信息

说明:

1. 发送查询电机状态信息的指令。
2. 电机返回的状态信息包括, 模式、错误码、位置、速度、力矩。
3. 电机状态信息解析在 `motor.c` 的 `motor_process_state` 函数。
4. 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

代码块

```
1  void motor_get_state_send(port_t portx, const data_type_t type, const uint8_t
```

```
id);
```

4.1.13 查询电机固件版本

说明：

1. 发送查询电机固件版本号指令。
2. 电机固件版本解析在 `motor.c` 的 `motor_process_state` 函数。
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_get_version(port_t portx, const uint8_t id);
```

4.1.14 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);
```

4.1.15 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);
```

4.1.16 电机软重启

说明：

1. 对电机执行软件重启。，重启后进入停止模式。
2. 不会有任何反馈。
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_reset(port_t portx, const uint8_t id);
```

4.2 一拖多控制模式 `motor_many.c`

- 此文件内，只有 `motor_many_send` 是用于发送指令的，其他的函数都是向缓存写入指令。
- 一拖多模式下，同一条 CAN 通道上的电机模式是相同的。
- 一拖多模式大致原理，发送一条通用的 FDCAN 帧，帧上不同的字节控制不同电机，其中由 ID 确定电机模式，详细说明请看 02-fdcan 协议解析。
- 一拖多模式下，在仲裁段为 1M，数据段为 5M 的情况下，一个 CAN 总线 1ms 内最多控制 10 个电机，即 1KHz 控制频率下最多控制 10 个电机，如需控制更多电机，需降低控制频率。
- `TINT16` 对应的是 `int16` 的数据类型，控制只能到 3.2 圈。
- 一拖多模式使用说明：

代码块

```
1  /* 写入缓存 */
2  motor_many_vel(PORT1, 1, 0.1);
3  motor_many_vel(PORT1, 2, 0.1);
4  motor_many_vel(PORT1, 3, 0.1);
5
6  /* 发送指令 */
7  motor_many_send(PORT1);
```

4.2.1 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
- `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
- `volt`：Q 相电压，单位：伏特（V）。

注意：

1. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

代码块

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

4.2.2 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
- `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
- `cur`：Q 相电流，单位：安培（A）。

注意：

1. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

代码块

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

4.2.3 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
- `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) 。

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**。
4. 此函数仅将指令写入缓冲区, 不会立即发送, 需要调用 `motor_many_send` 才会生效。

代码块

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

4.2.4 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
 - `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)

注意:

1. 此函数仅将指令写入缓冲区, 不会立即发送, 需要调用 `motor_many_send` 才会生效。

代码块

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

4.2.5 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
 - `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
 - `tqe` : 目标力矩，单位：牛·米 (N·m) 。

注意：

1. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

代码块

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

4.2.6 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
 - `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)

注意：

1. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

代码块

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

4.2.7 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
 - `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) 。
 - `tqe` : 目标力矩，单位：牛·米 (N·m) 。

注意：

3. 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。
4. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

代码块

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

4.2.8 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 (rps²) 。

注意：

1. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

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

4.2.9 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
- `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) 。
- `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：

- 需要设置合适的 `kp` 和 `kd` ，否则控制效果可能较差。
- `motor_many_pos_vel_tqe_kp_kd` 函数的位置是不断积分得到的，所以在低频控制下会有意外情况。
- 推荐使用运控模式 2 `motor_many_pos_vel_tqe_kp_kd_2`。
- 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

代码块

```
1  /* 不建议使用 */  
2  void motor_many_pos_vel_tqe_kp_kd(port_t portx, const uint8_t id,  
3  const float pos, const float vel, const float tqe, const float kp, const float  
4  kd);  
5  /* 建议使用 */  
6  void motor_many_pos_vel_tqe_kp_kd_2(port_t portx, const uint8_t id,  
7  const float pos, const float vel, const float tqe, const float kp, const float  
8  kd);
```

4.2.10 运控模式 2

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
- 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) 。
- 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：

3. 真正的运控模式，和 motor_many_pos_vel_tqe_kp_kd 区别是：

- motor_many_pos_vel_tqe_kp_kd 的位置是不断积分得到的，在低频控制下容易出现意外情况。
- motor_many_pos_vel_tqe_kp_kd_2 则避免了此问题，控制更稳定。

4. 该需要设置合适的 kp 和 kd ，否则控制效果可能较差。

5. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 motor_many_send 才会生效。

代码块

```
1  /* 不建议使用 */
2  void motor_many_pos_vel_tqe_kp_kd(port_t portx, const uint8_t id,
3  const float pos, const float vel, const float tqe, const float kp, const float
4  kd);
5  /* 建议使用 */
6  void motor_many_pos_vel_tqe_kp_kd_2(port_t portx, const uint8_t id,
7  const float pos, const float vel, const float tqe, const float kp, const float
8  kd);
```

4.3 电机设置 motor_config

- 此文件内的函数都是用于电机设置相关。

4.3.1 重置电机零位

说明：

1. 功能：

- 将电机当前位置设置成零位。
- 重置一个电机的零位需要要 300ms 以上，且有失败可能。
- 返回值：0-成功，1-失败。

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 uint8_t motor_pos_reset(port_t portx, const uint8_t id);
```

4.3.2 保存电机设置

说明：

1. 功能

- 保存电机设置。
- 一般配合其他指令运行。
- 返回值：0-成功，1-失败。

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 uint8_t motor_conf_write(port_t portx, const uint8_t id);
```

4.4 电机配置和协议解析 `motor`

- 此文件中定义了两个重要的结构体数组：详见 3.2配置
 - `motor_state_port[MOTOR_PORT_NUM][MOTOR_MAX_NUM]`：用于配置电机型号和储存电机返回的信息。

- `port_maping[MOTOR_PORT_NUM]`：用于配置 PORT、FDCAN 和 STATE 的映射关系。

4.4.1 电机信息解析

说明：

- 解析所有 CAN 通道 FIFO 中的电机状态数据。
- 此函数中调用 `motor_process_state` 进行解析。
- **此函数需要放在主循环中不断调用，也可根据需求定时调用。**

代码块

```
1 void motor_process_state_all();
```