CANopen协议移植及测试

1. CANopen移植前准备与学习资料

强烈推荐 ZLG 的 **CANopen入门教程**：https://n09fqw94uwx.feishu.cn/wiki/CqeZwvHMHicbEyks41KcryATnif?from=from_copylink

ZLG 的 **CANopen备忘录**：https://n09fqw94uwx.feishu.cn/wiki/H24Ew3OnqiaEggkLlcQcuvaMnMc?from=from_copylink

CANopen协议栈源码：https://github.com/CANopenNode/CANopenNode

CANopen协议栈基于STM32开发的demo：https://github.com/CANopenNode/CanOpenSTM32

CANopenNode协议栈的字典配置工具：https://github.com/CANopenNode/CANopenEditor

2. 在DM_MC02(H7)主控板上移植

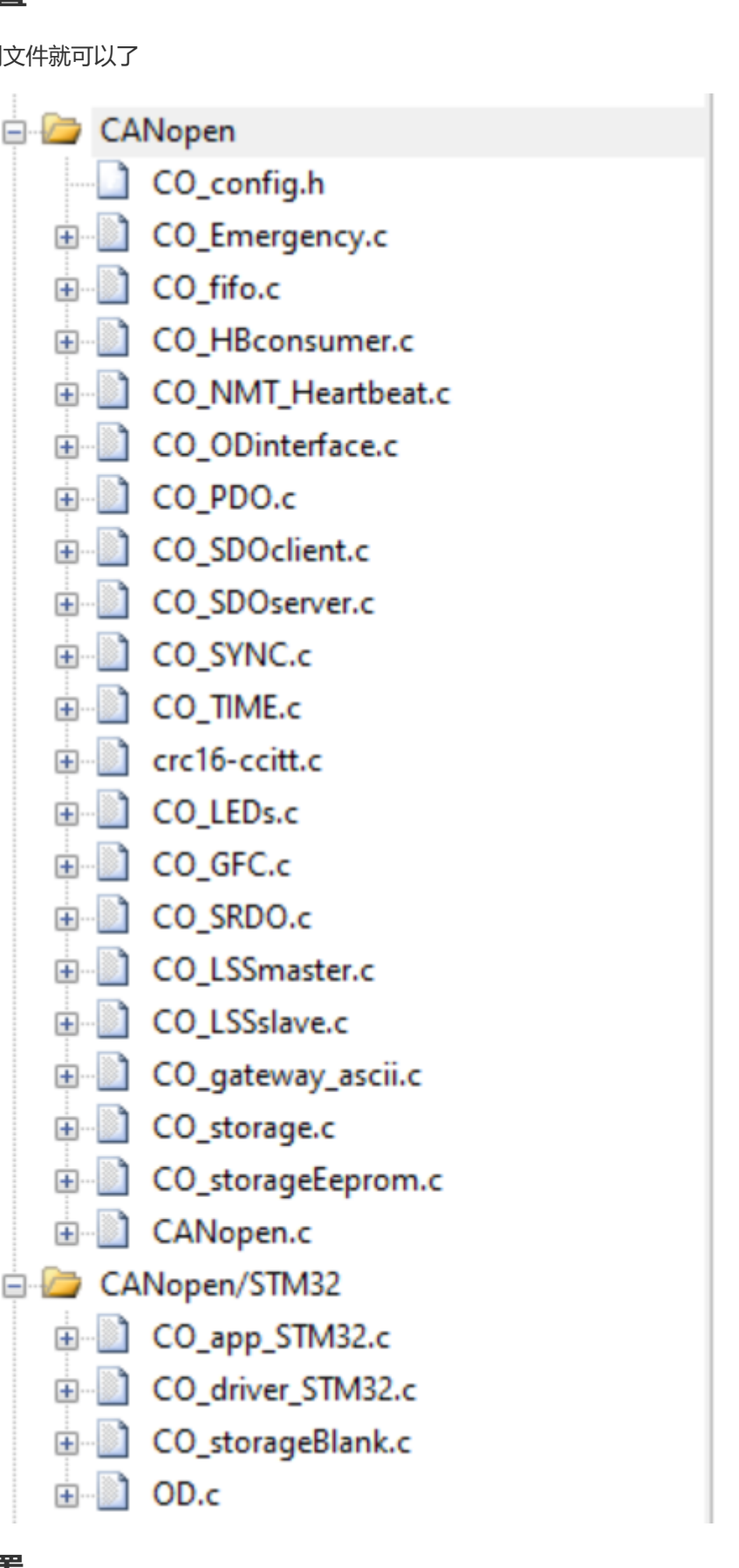
2.1 STM32CubeMX 配置

在STM32CubeMX中创建一个新项目，主要配置下面四步就可以了

- 将CAN/FDCAN配置为您想要的比特率，并将其映射到相关的tx/rx引脚 - 确保激活自动总线恢复（bxCAN）/协议异常处理（FDCAN）
- 激活CAN外设上的RX和TX中断
- 为 1ms 溢出中断启用定时器，并为该定时器激活中断
- 修改CubeMx初始配置的堆栈大小

2.2 keil 配置

在keil中添加下列文件就可以了



2.3 裸机配置

添加如下代码就行

```
/* USER CODE BEGIN 2 */
CANopenNodeSTM32 canOpenNodeSTM32;
canOpenNodeSTM32.CANHandle = &hfdcan1;
canOpenNodeSTM32.HWInitFunction = MX_FDCAN1_Init;
canOpenNodeSTM32.timerHandle = &htim3;
canOpenNodeSTM32.desiredNodeID = 0x1A;
canOpenNodeSTM32.baudrate = 1000;
canopen_app_init(&canOpenNodeSTM32);
/* USER CODE END 2 */
/* USER CODE BEGIN WHILE */
while (1)
{
    canopen_app_process();
  /* USER CODE END WHILE */
}
```

添加定时器回调代码

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  // Handle CANOpen app interrupts
  if (htim == canopenNodeSTM32->timerHandle) {
      canopen_app_interrupt();
  }
}
```

2.4 FreeRTOS 配置

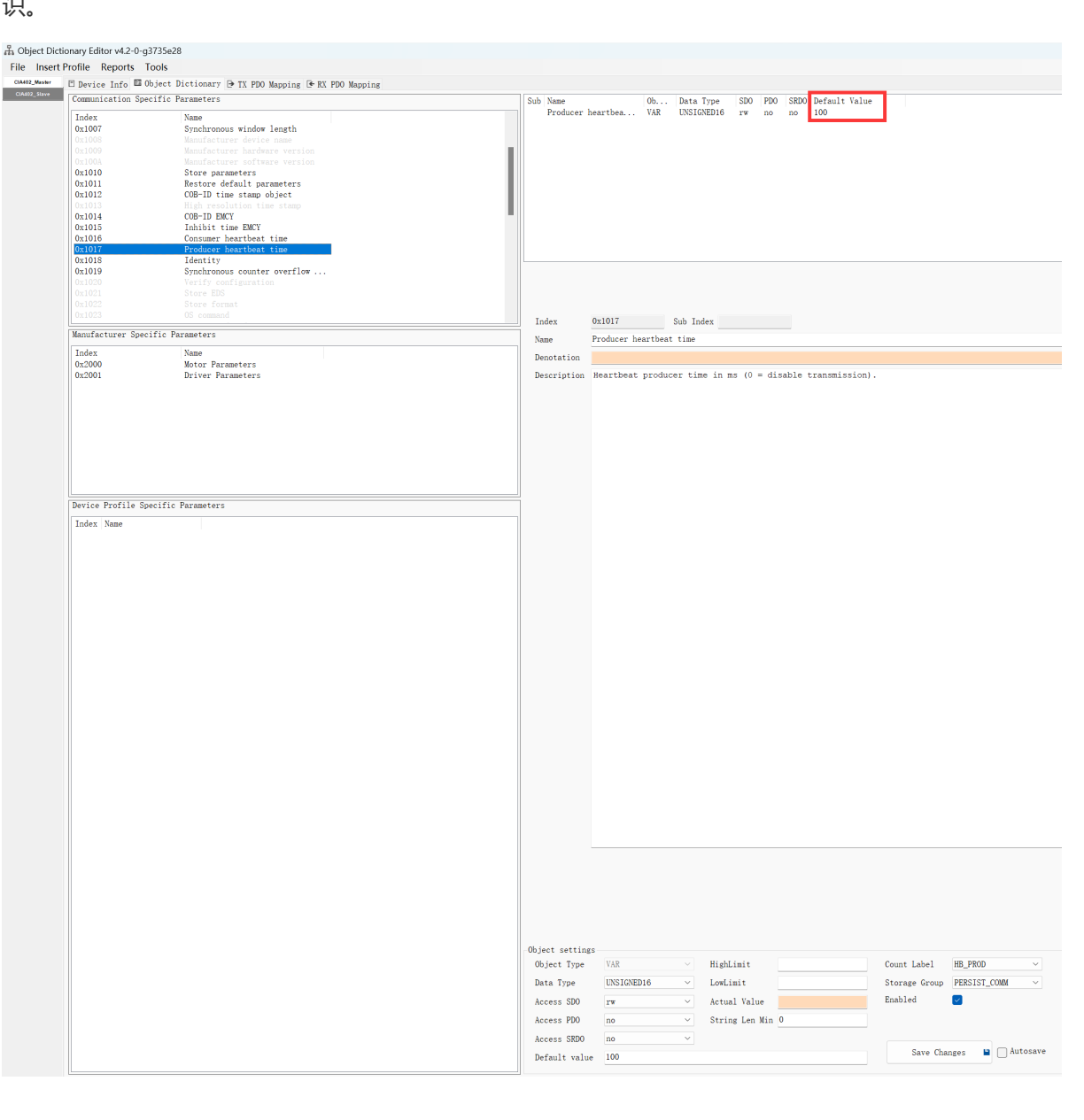
外卖需要为 CANOpen 创建一个任务，并且以高优先级去调用它，并在该任务中使用以下代码：`canopen_task`

```
void canopen_task(void *argument)
{
  /* USER CODE BEGIN canopen_task */
  CANopenNodeSTM32 canOpenNodeSTM32;
  canOpenNodeSTM32.CANHandle = &hfdcan1;
  canOpenNodeSTM32.HWInitFunction = MX_FDCAN1_Init;
  canOpenNodeSTM32.timerHandle = &htim3;
  canOpenNodeSTM32.desiredNodeID = 0x1A;
  canOpenNodeSTM32.baudrate = 1000;
  canopen_app_init(&canOpenNodeSTM32);
  /* Infinite loop */
  for(;;)
  {
    canopen_app_process();
    // Sleep for 1ms, you can decrease it if required, in the canopen_app_process
we will       double check to make sure 1ms passed
    vTaskDelay(pdMS_TO_TICKS(1));
  }
  /* USER CODE END canopen_task */
}
```

3. 字典配置

ZLG 的 **CANopen备忘录** 里面有详细的对象字典

参照 CIA301 协议去配置初始化字典，这里我添加了0x2000和0x2001的索引号，用来配置电机和驱动板参数，并且将从机心跳上报时间设置为100ms。这里先按照工程模板去使用，后面再去讲解这方面的知识。



4. 从机测试

4.1 添加从机代码，将从机ID设置为0x18

```
void canopen_demo(void)
{
    uint32_t time;
    /* 初始化CANopen */
    canOpenNodeSTM32.CANHandle = &hfdcan1;                  /* 使用CANFD1接口 */
    canOpenNodeSTM32.HWInitFunction = MX_FDCAN1_Init;       /* 初始化CANFD1 */
    canOpenNodeSTM32.timerHandle = &htim3;                  /* 使用的定时器句柄 */
    canOpenNodeSTM32.desiredNodeID = 0x18;                  /* Node-ID */
    canOpenNodeSTM32.baudrate = 1000;                       /* 波特率，单位KHz */
    canopen_app_init(&canOpenNodeSTM32);

    /* 设置年月日，时分秒，毫秒和时间戳上传周期 */
    CANopen_WriteClock(2024,5,8,10,10,10,0,1000);

    while (1)
    {
        canopen_app_process();
    }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */
    if (htim == canopenNodeSTM32->timerHandle) {
        canopen_app_interrupt();
    }
}
```

4.2 NMT 节点测试

4.2.1 NMT 节点上线与心跳测试

任何一个 CANopen 从站上线后，为了提示主站它已经加入网络（便于热插拔），或者避免与其他从站 Node-ID 冲突。这个从站必须发出**节点上线报文（boot-up）**，节点上线报文的 **ID为700h+Node-ID，数据为 1个字节 0**。生产者为 CANopen 从站。

为了监控 CANopen 节点是否在线与目前的节点状态。CANopen 应用中通常都要求在线，上电的从站定时发送状态报文（心跳报文），以便于主站确认从站是否故障、是否脱离网络。为心跳报文发送的格式，CANID与节点上线报文相同**为700h+Node-ID，数据为1个字节，代表节点目前的状态，04h为停止状态，05h为操作状态，7Fh为预操作状态。**

上述发现 NMT 节点在上线之后发送了一帧数据，之后按每帧100ms的时间发送心跳报文。

4.2.2 NMT 节点状态测试

NMT 节点状态切换命令

命令	帧ID	bit1	bit2
NMT启动命令	0x00	0x01	0-0x7F
NMT停止命令	0x00	0x02	0-0x7F
NMT预操作命令	0x00	0x80	0-0x7F
NMT复位应用层命令	0x00	0x81	0-0x7F
NMT复位节点通讯命令	0x00	0x82	0-0x7F

第 1 个字节代表命令类型，第2个字节代表被控制的节点 **Node-ID**，如果要对整个网络所有节点同时进行控制，则这个数值为 **0** 即可。

NMT 节点状态数据

数据值	状态
0x04	停止状态
0x05	操作状态
0x7F	预操作状态

实际测试

4.3 快速 SDO 协议测试

4.3.1 快速 SDO CS 命令

SDO 写对象字典:

	数据1	数据2-3	数据4	数据5-8	
客户端	**CS命令符**	**索引(3为高位)**	**子索引**	**数据(高位在后)**	**>>服务器**

SDO 写对象响应:

	数据1	数据2-3	数据4	数据5-8	
客户端<<	**CS命令符**	**索引(3为高位)**	**子索引**	**补0**	**服务器**

SDO 读对象字典:

	数据1	数据2-3	数据4	数据5-8	
客户端	**CS命令符**	**索引(3为高位)**	**子索引**	**补0**	**>>服务器**

SDO 读对象响应:

	数据1	数据2-3	数据4	数据5-8	
客户端<<	**CS命令符**	**索引(3为高位)**	**子索引**	**数据(高位在后)**	**服务器**

CS 命令符:

0x2F	**写一个字节**	**0x40**	**读取**
0x2B	**写两个字节**	**0x4F**	**读响应一个字节**
0x27	**写三个字节**	**0x4B**	**读响应两个字节**
0x23	**写四个字节**	**0x47**	**读响应三个字节**
0x60	**写成功应答**	**0x43**	**读响应四个字节**
0x80	**异常响应**		

4.3.2 SDO 命令测试

命令	帧ID	bit0	bit1	bit2	bit3	bit4	bit5	bit6	bit7
设置心跳周期5s	0x618	2B	17	10	00	88	13	00	00
关闭心跳周期	0x618	2B	17	10	00	00	00	00	00
读取心跳包周期	0x618	40	17	10	00	00	00	00	00
开启时间戳	0x618	23	12	10	00	00	00	00	C0
关闭时间戳	0x618	23	12	10	00	00	00	00	00

测试结果:

4.4 RPDO 测试

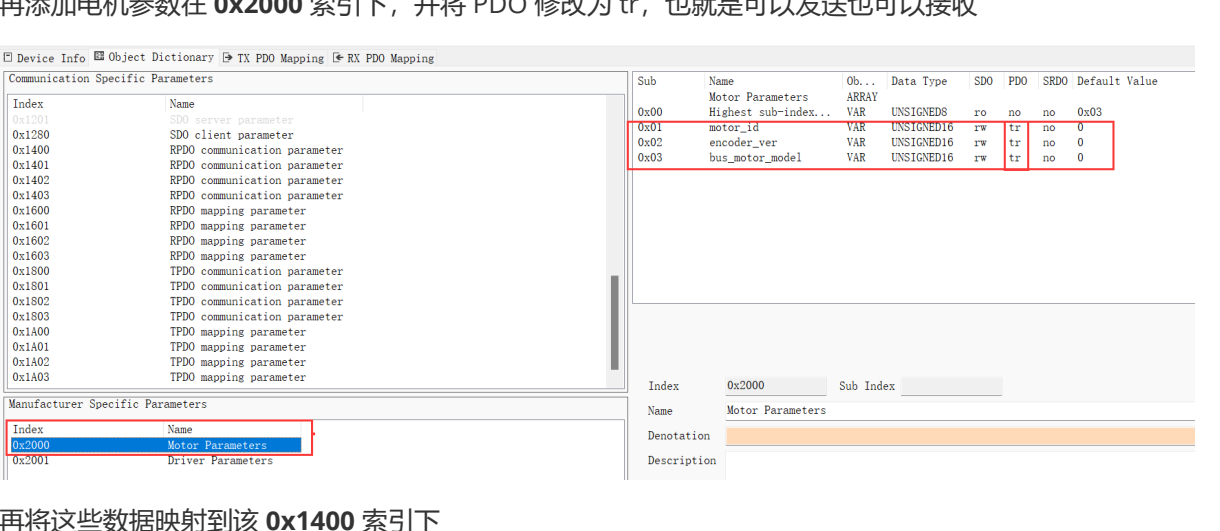
PDO 通信比较灵活，广义上只要符合 PDO 范围内的所有 CANID 都可以作为节点自身的 TPDO 或者 RPDO 使用，也称为 COB-ID，不受功能码和 Node-ID 限制。

object 对象	Specification 规范	CAN_ID(COB-ID)
TPDO1 发送过程数据对象 1	CiA301	181h to 1FFh (180h +node-ID)
RPDO1 接收过程数据对象 1	CiA301	201h to 27Fh (200h +node-ID)
TPDO2 发送过程数据对象 2	CiA301	281h to 2FFh (280h +node-ID)
RPDO2 接收过程数据对象 2	CiA301	301h to 37Fh (300h +node-ID)
TPDO3 发送过程数据对象 3	CiA301	381h to 3FFh (380h +node-ID)
RPDO3 接收过程数据对象 3	CiA301	401h to 47Fh (400h +node-ID)
TPDO4 发送过程数据对象 4	CiA301	481h to 4FFh (480h +node-ID)
RPDO4 接收过程数据对象 4	CiA301	501h to 57Fh (500h +node-ID)

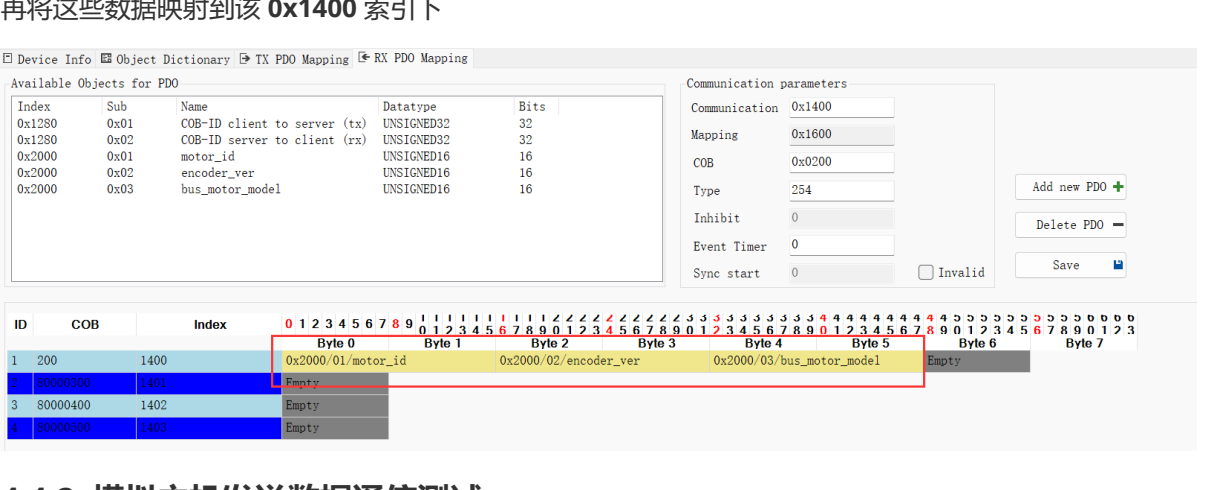
4.4.1 设置 RPDO 字典

设置索引 **0x1400** 下子索引的值，将 **COB-ID** 设置为主机发送数据的ID，将 **Transmit type** 的类型配置为254，即用户自定义的发送类型

再添加电机参数在 **0x2000** 索引下，并将 PDO 修改为 tr，也就是可以发送也可以接收



再将这些数据映射到该 **0x1400** 索引下

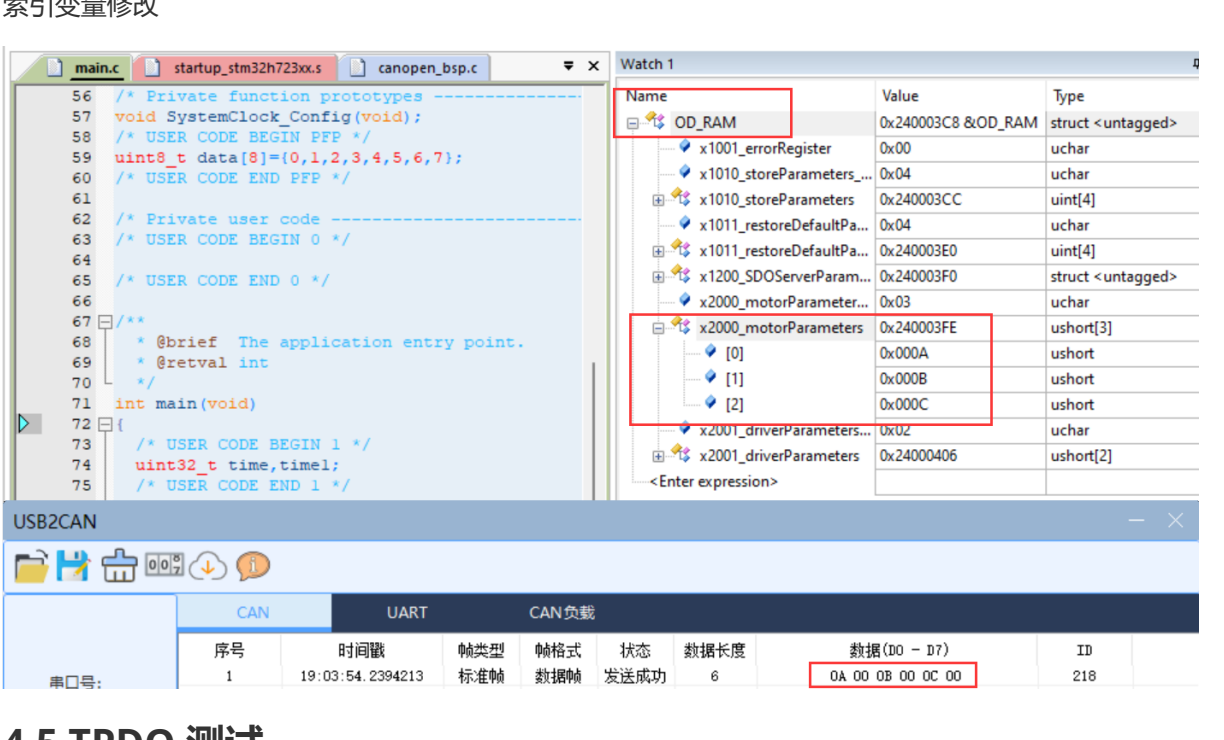


4.4.2 模拟主机发送数据通信测试

使用 DM 的 **USB2CAN** 模拟主机发送信号

命令	帧ID	bit0	bit1	bit2	bit3	bit4	bit5	bit6	bit7
设置映射在RPDO1参数	0x218	0x0A	0x00	0x0B	0x00	0x0C	0x00	0x00	0x00

格式需要注意大小端，打开 **keil debug** 界面查看 **OD_RAM** 变量，发送成功后会将 **0x2000** 索引下的子索引变量修改



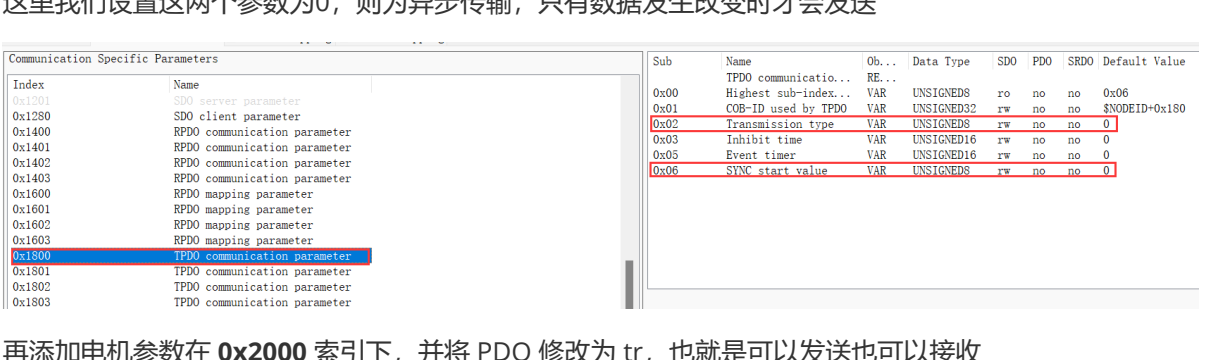
4.5 TPDO 测试

4.5.1 TPDO 字典注释

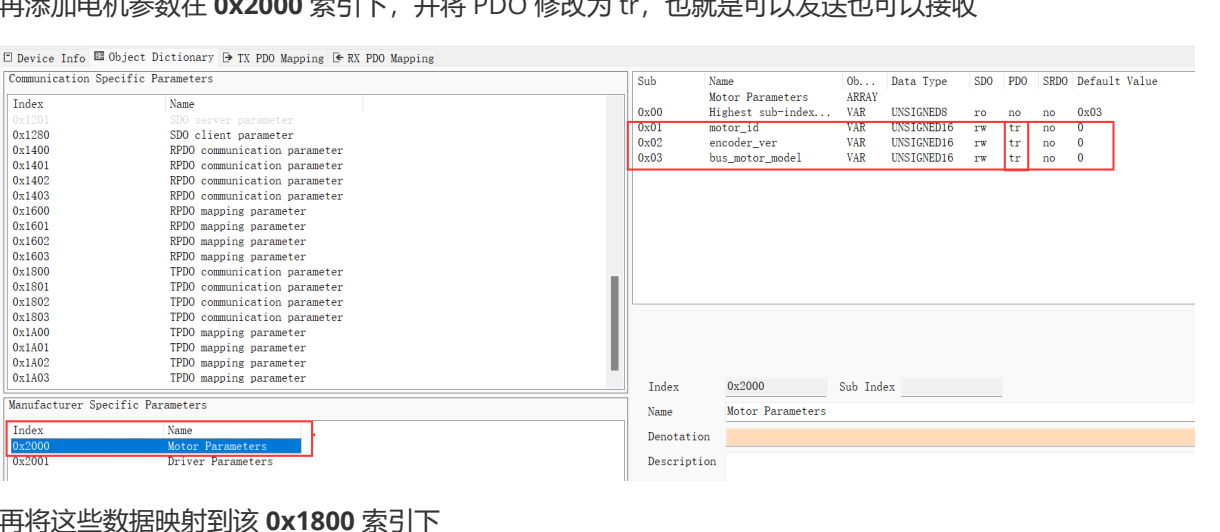
COB-ID used by TPDO：TPDO反馈时的数据ID，即节点ID+0X180；
Transmission type：每收到多少个同步报文反馈一次数据；
Inhibit time：定义了对该数据对象的传输服务的两个连续调用之间必须经过的最小时间
Event timer：这个待补充；
SYNC start value：同步开始的计数，这里说的是开始进行数据同步时，0x80同步报文的计数值；

4.5.2 配置 TPDO 异步传输

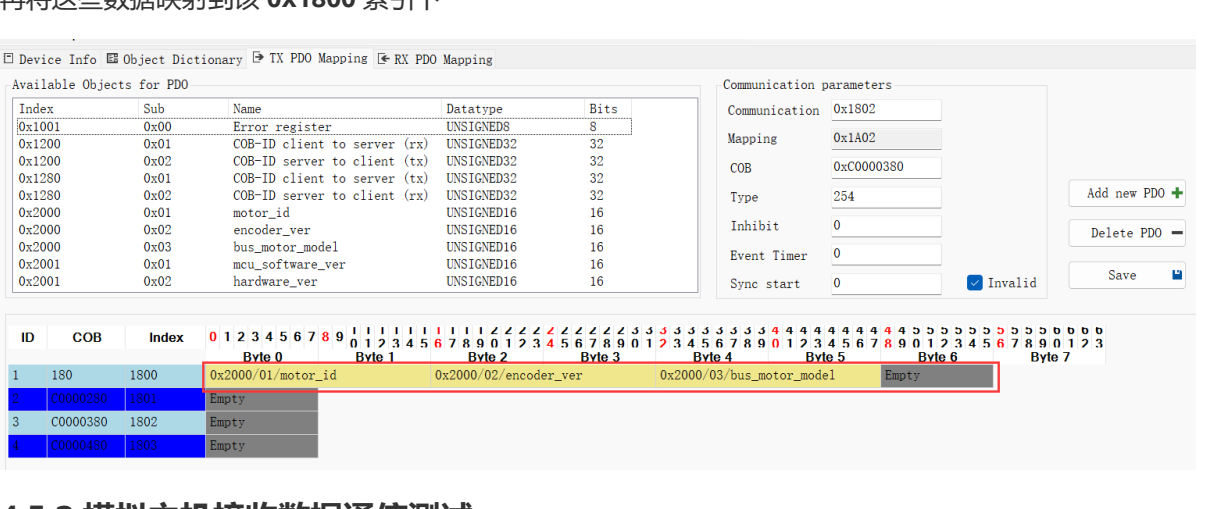
这里我们设置这两个参数为0，则为异步传输，只有数据发生改变时才会发送



再添加电机参数在 **0x2000** 索引下，并将 PDO 修改为 tr，也就是可以发送也可以接收



再将这些数据映射到该 **0x1800** 索引下



4.5.3 模拟主机接收数据通信测试

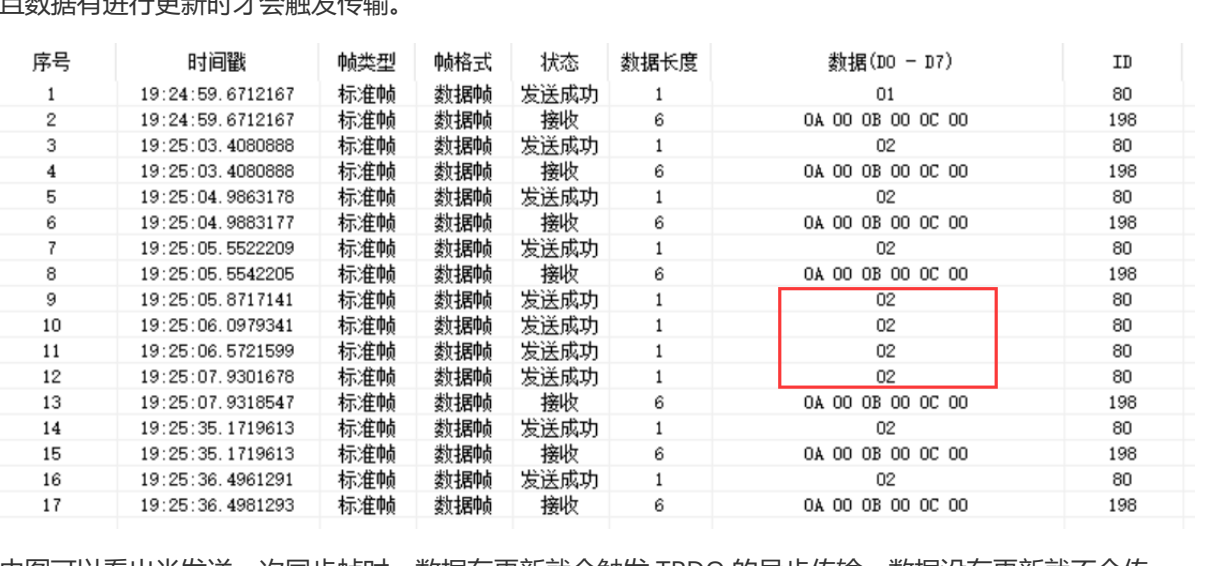
在 keil 中添加代码：

```
canopen_app_process();
//1.5s 进行一次数据赋值，达到数据更新的目的
if(HAL_GetTick() - time > 1500) {
    OD_RAM.x2000_motorParameters[0]=0x0A;
    OD_RAM.x2000_motorParameters[1]=0x0B;
    OD_RAM.x2000_motorParameters[2]=0x0C;
    CO_TPDOsendRequest(&CO->TPDO[0]);
    time = HAL_GetTick();
}
```

使用 DM 的 **USB2CAN** 模拟主机发送同步信号

命令	帧ID	bit0	bit1	bit2	bit3	bit4	bit5	bit6	bit7
同步信号	0x80	0x01 递增							

同步帧的时间也可以在主机字典里面进行设置，这里我们模拟一下，因为异步传输只有在接收到同步帧且数据有进行更新时才会触发传输。

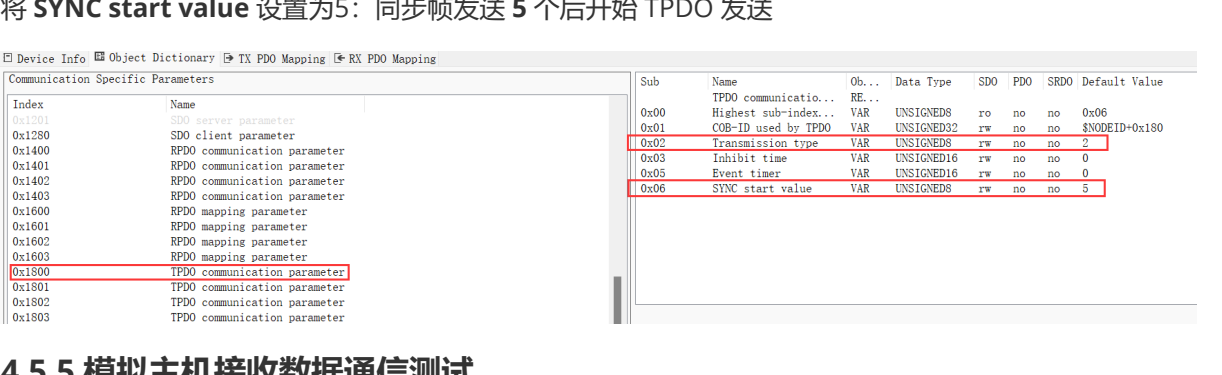


由图可以看出当发送一次同步帧时，数据有更新就会触发 TPDO 的异步传输，数据没有更新就不会传输。

4.5.4 配置 TPDO 同步传输

将 **Transmission type** 设置为2：每接收到 **2** 个同步帧时，TPDO 发送

将 **SYNC start value** 设置为5：同步帧发送 **5** 个后开始 TPDO 发送

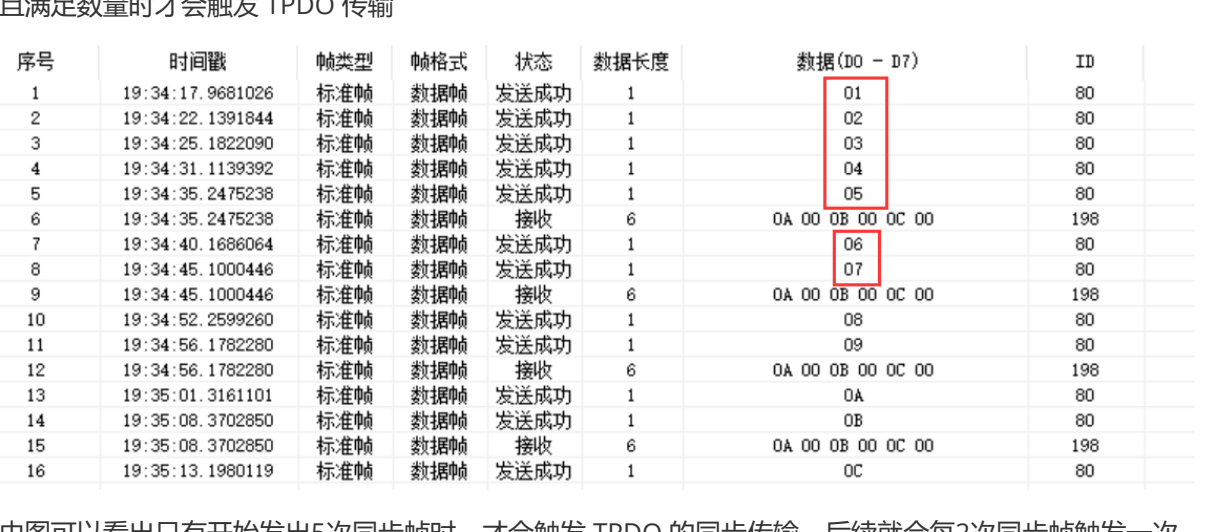


4.5.5 模拟主机接收数据通信测试

使用 DM 的 **USB2CAN** 模拟主机发送同步信号

命令	帧ID	bit0	bit1	bit2	bit3	bit4	bit5	bit6	bit7
同步信号	0x80	0x01 递增							

同步帧的时间也可以在主机字典里面进行设置，这里我们模拟一下，因为同步传输只有在接收到同步帧且满足数量时才会触发 TPDO 传输



由图可以看出只有开始发出5次同步帧时，才会触发 TPDO 的同步传输，后续就会每2次同步帧触发一次 TPDO 传输。

5. 主机测试

主机测试与从机测试差不多一样，就是字典设置上有区别，且主机测试需要发送同步帧。因为之前都是用上位机去模拟主机，其实大部分操作都已经做完，只要将索引号和ID一一对应起来就好。

5.1 主机通信测试代码

5.1.1 初始化代码

```
canOpenNodeSTM32.CANHandle = &hfdcan1;              /* 使用CANFD1接口 */
canOpenNodeSTM32.HWInitFunction = MX_FDCAN1_Init;   /* 初始化CANFD1 */
canOpenNodeSTM32.timerHandle = &htim3;              /* 使用的定时器句柄 */
canOpenNodeSTM32.desiredNodeID = 0x1A;              /* Node-ID */
canOpenNodeSTM32.baudrate = 1000;                   /* 波特率，单位KHz */
canopen_app_init(&canOpenNodeSTM32);

/* 设置年月日，时分秒，毫秒和时间戳上传周期 */
CANopen_WriteClock(2024,5,8,10,10,10,0,1000);
```

5.1.2 写SDO

```
// 对从机ID 0x18 下的 0x2000 索引下的子索引0x01写入2个字节数据
write_SDO(CO->SDOclient, CANopenSlaveID1, 0x2000, 0x01, write_sdo_buf, 2);
```

5.1.2 读SDO

```
// 读取从机ID 0x18 下的 0x2000 索引下的子索引0x01下的数据
read_SDO(CO->SDOclient, CANopenSlaveID1, 0x2000, 0x01, read_sdo_buf,
sizeof(read_sdo_buf), &read_sdo_size);
```

6 主从通信实验代码

链接：