以太网系列2

本系列相对于系列 1 增加了定时发送、队列发送等功能，支持的型号包括：CANFDNET-TCP、CANFDNET-400U-TCP、CANFDNET-UDP、CANFDNET-400U-UDP。

1.工作模式

连接的工作模式，如设备作为服务器，二次开发时应设为客户端模式。

项	值	说明
path	n/work_mode	n 代表通道号
value	0-客户端 1-服务器	
Get/Set	Set	

注意：

1. 需在 ZCAN_StartCAN 之前设置
2. UDP 系列不设置此参数

2.本地端口

UDP 模式和 TCP Server 模式下的本地监听端口。

项	值	说明
path	n/local_port	n 代表通道号
value	自定义	如 “4001”
Get/Set	Set	

注意：

1. 需在 ZCAN_StartCAN 之前设置

3.IP地址

目标设备的IP

项	值	说明
path	n/ip	n 代表通道号
value	自定义	如 “192.168.0.178”
Get/Set	Set	

注意：

1. 需在 ZCAN_StartCAN 之前设置

4.工作端口

目标设备监听的端口

项	值	说明
path	n/work_port	n 代表通道号
value	自定义	如 “4001”
Get/Set	Set	

注意：

1. 需在 ZCAN_StartCAN 之前设置

5.定时发送

定时发送CAN帧

项	值	说明
path	n/auto_send	n 代表通道号
value	ZCAN_AUTO_TRANSMIT_OBJ指针	需将指针强制转换为char*
Get/Set	Set	

注意点：需要在ZCAN_StartCAN之后设置

定时发送CANFD帧

项	值	说明
path	n/auto_send_canfd	n 代表通道号
value	ZCANFD_AUTO_TRANSMIT_OBJ指针	需将指针强制转换为char*
Get/Set	Set	

注意点：需要在ZCAN_StartCAN之后设置

开始定时发送

项	值	说明
path	n/apply_auto_send	n 代表通道号
value	0	固定值
Get/Set	Set	

注意点：需要在ZCAN_StartCAN之后设置

清空定时发送

项	值	说明
path	n/clear_auto_send	n 代表通道号
value	0	固定值
Get/Set	Set	

注意点：需要在ZCAN_StartCAN之后设置

查询定时发送CAN帧数量

项	值	说明
path	n/get_auto_send_can_count/1	n 代表通道号,最后的数字“1”只是内部标志，可以是任意数字，返回值 char*转换为 int
value	函数返回指向int的指针表示定时发送CAN的数量	intnCount = *(int*)pRet; pRet 表示 GetValue 返回值，nCount 表示定时发送 CAN 数量
Get/Set	Get	

注意点：需要在ZCAN_StartCAN之后设置

查询定时发送CAN帧信息

项	值	说明
path	n/get_auto_send_can_data/1	n 代表通道号,最后的数字“1”只 char*转换为 ZCAN_AUTO_T
value	函数返回指向ZCAN_AUTO_TRANSMIT_OBJ*的指针表示定时发送CAN的数据首地址	返回值类型转换后指向定时发
Get/Set	Get	

注意点：需要在ZCAN_StartCAN之后设置

查询定时发送CANFD帧数量

项	值	说明
path	n/get_auto_send_canfd_count/1	n 代表通道号,最后的数字“1”只是内部标志，可以是任意数字，返 char*转换为 int
value	函数返回指向int的指针表示定时发送CANFD的数量	intnCount = *(int*)pRet; pRet 表示 GetValue 返回值，nCount 表示定时发送 CANFD 数
Get/Set	Get	

注意点：需要在ZCAN_StartCAN之后设置

查询定时发送CANFD帧信息

项	值	说明
path	n/get_auto_send_can_data/1	n 代表通道号,最后的数字“1 char*转换为 ZCANFD_AU
value	函数返回指向ZCANFD_AUTO_TRANSMIT_OBJ*的指针表示定时发送CAN的数据首地址	返回值类型转换后指向定
Get/Set	Get	

注意点：需要在ZCAN_StartCAN之后设置

6.队列发送

CANFDNET 系列设备支持普通发送、队列发送。普通发送指用户发送多帧数据时，设备会尽力使用最快的速度进行发送，帧之间的时间间隔无法控制。队列发送指用户可以设定帧之间的发送间隔，设备会按照帧顺序，使用指定的间隔将数据发送到总线。
通过队列发送，用户可以提前准备好多帧报文，以及报文之间的间隔，将准备好的报文批量发送给设备，设备按照预定义的帧间隔进行精准发送，通过此方式可提高发送帧之间的帧间隔精度。区别与定时发送，队列发送每帧只发送一次，需由用户不断准备报文并批量发送到设备。CANFDNET 设备可以同时使用队列发送，定时发送，普通发送。

- 队列发送数据的发送使用 ZCAN_Transmit/ZCAN_TransmitFD 接口，返回值表示有多少帧已经加入到设备的发送队列中。
- 可以通过接口获取设备端可用的队列空间。
- 模式发送帧间隔时间单位为毫秒(ms)，长度 2 字节需要分别填入can/canfd 帧中的**res0(帧间隔低 8 位)**和res1(帧间隔高 8 位)字段中，设备支持最大帧间隔时间为65535ms，同时需要设置延时发送标志位(TX_DELAY_SEND_FLAG)为 1 标识使用队列发送。
- 队列发送时，CAN 帧的 TX_DELAY_SEND_FLAG 标志位在 frame.__pad 字段的Bit7 位，CANFD 帧的 TX_DELAY_SEND_FLAG 标志位在 frame.flags 字段的 Bit7位。标志位为 1 表示使用队列顺序发送数据。标志为 0 表示是普通发送，会直接发送数据到总线。
- 设备发送当前帧的同时会启动计时器按照当前帧设定的时间进行计时，计时时间结束会从队列取下一帧进行发送并重新开始计时。

获取发送队列可用缓存数量（仅队列模式）

项	值	说明
path	n/get_device_available_tx_count/1	n 代表通道号,最后的数字“1”只是内部标志，可以是任意数字，返回值 char*转换为 int*
value	无	
Get/Set	Get	

清空发送缓存（仅队列模式，缓存中未发送的帧将被清空，停止时使用）

项	值	说明
path	n/clear_delay_send_queue	n 代表通道号
value	0	固定值
Get/Set	Set	

7.总线利用率

CANFDNET 设备如果开启了总线利用率上报，则可以通过接口获取总线利用率信息

项	值	说明
path	n/get_bus_usage/1	n 代表通道号，最后的数字“1”只是内部标志，可以是任意数字，返回值 char*转换为 BusUsage*
value	0	
Get/Set	Get	

8.示例代码

```
// 以下代码以 CANFDNET-400 为例，对于本系列，每个连接均可操作全部通道
// 如设备做服务器，端口 8000，程序做客户端，只需在 start 任意通道之前设置 0 通道的工作模式、端口、IP
#include "stdafx.h"
#include "zlgcan.h"
#include <iostream>
#include <windows.h>
#include <thread>
#define CH_COUNT 4
bool g_thd_run = 1;
// 此函数仅用于构造示例 CAN 报文
void get_can_frame(ZCAN_Transmit_Data& can_data, canid_t id)
{
memset(&can_data, 0, sizeof(can_data));
can_data.frame.can_id = id; // CAN ID
can_data.frame.can_dlc = 8; // CAN 数据长度 8
can_data.transmit_type = 0; // 正常发送
for (int i = 0; i < 8; ++i) { // 填充 CAN 报文 DATA
can_data.frame.data[i] = i;
} }// 此函数仅用于构造示例 CANFD 报文
void get_canfd_frame(ZCAN_TransmitFD_Data& canfd_data, canid_t id)
{
memset(&canfd_data, 0, sizeof(canfd_data));
canfd_data.frame.can_id = id; // CANFD ID
canfd_data.frame.len = 64; // CANFD 数据长度 64
canfd_data.transmit_type = 0; // 正常发送
for (int i = 0; i < 64; ++i) { // 填充 CANFD 报文 DATA
canfd_data.frame.data[i] = i;
} }
// 此函数仅用于构造示例队列发送 CAN 报文
void get_can_frame_queue(ZCAN_Transmit_Data& can_data, canid_t id, UINT delay)
{
memset(&can_data, 0, sizeof(can_data));
can_data.frame.can_id = id; // CAN ID
can_data.frame.can_dlc = 8; // CAN 数据长度 8
can_data.frame.__pad |= TX_DELAY_SEND_FLAG; // 设置延时标志位
can_data.frame.__res0 = LOBYTE(delay); // 帧间隔低字节
can_data.frame.__res1 = HIBYTE(delay); // 帧间隔高字节
can_data.transmit_type = 0; // 正常发送
for (int i = 0; i < 8; ++i) { // 填充 CAN 报文 DATA
can_data.frame.data[i] = i;
} }
void thread_task(CHANNEL_HANDLE handle, int ch)
{
std::cout << "chnl: " << std::dec << ch << " thread run, handle:0x" << std::hex << handle
<< std::endl;
ZCAN_Receive_Data data[100] = {};
ZCAN_ReceiveFD_Data fd_data[100] = {};
while (g_thd_run)
{
int count = ZCAN_GetReceiveNum(handle, 0); // 获取 CAN 报文（参数 2：0 - CAN, 1 - CANFD）数量
while (g_thd_run && count > 0)
{
int rcount = ZCAN_Receive(handle, data, 100, 10);
for (int i = 0; i < rcount; ++i)
{
std::cout << "CHNL: " << std::dec << ch << " recv can ID: 0x" << std::hex <<
data[i].frame.can_id << std::endl;
}
count -= rcount;
}
count = ZCAN_GetReceiveNum(handle, 1); // 获取 CANFD 报文（参数 2：0 - CAN, 1 - CANFD）数量
while (g_thd_run && count > 0)
{
int rcount = ZCAN_ReceiveFD(handle, fd_data, 100, 10);
for (int i = 0; i < rcount; ++i)
{
std::cout << "CHNL: " << std::dec << ch << " recv canfd ID: 0x" << std::hex <<
fd_data[i].frame.can_id << std::endl;
}
count -= rcount;
}
Sleep(100);
}
std::cout << "chnl: " << std::dec << ch << " thread exit" << std::endl;
}
int _tmain(int argc, _TCHAR* argv[])
{
UINT dev_type = ZCAN_CANFDNET_400U_TCP;
std::thread thd_handle[CH_COUNT];
CHANNEL_HANDLE ch[CH_COUNT] = {};
DEVICE_HANDLE device = INVALID_DEVICE_HANDLE;
IProperty* prop = {};
// 打开设备
device = ZCAN_OpenDevice(dev_type, 0, 0);
if (INVALID_DEVICE_HANDLE == device) {
std::cout << "open device failed!" << std::endl;
goto end;
}
// 获取 IProperty 指针，用于配置参数
prop = GetIProperty(device);
if (NULL == prop) {
std::cout << "get property failed" << std::endl;
goto end;
}
// 本系列同一设备多通道只需设置一次 工作模式、IP、端口，并需在全部通道 start 之前设置
// 设置工作模式为客户端
if (0 == prop->SetValue("0/work_mode", "0")) {
std::cout << "set work mode failed" << std::endl;
goto end;
}
// 设置目标 IP 地址
if (0 == prop->SetValue("0/ip", "172.16.9.221")) {
std::cout << "set ip failed" << std::endl;
goto end;
}
// 设置目标通道端口
if (0 == prop->SetValue("0/work_port", "8000")) {
std::cout << "set port failed" << std::endl;
goto end;
}
// 循环初始化、启动每个通道
for (int i = 0; i < CH_COUNT; ++i) {
char path[64] = {};
// 初始化通道，在 CANFDET 系列中初始化不做实际操作，仅用于获取通道句柄
ZCAN_CHANNEL_INIT_CONFIG config;
ch[i] = ZCAN_InitCAN(device, i, &config);
if (INVALID_CHANNEL_HANDLE == ch[i]) {
std::cout << "init channel failed!" << std::endl;
goto end;
}
// 启动 CAN 通道
if (0 == ZCAN_StartCAN(ch[i])) {
std::cout << "start channel failed" << std::endl;
goto end;
}
// 启动 CAN 通道的接收线程
thd_handle[i] = std::thread(thread_task, ch[i], i);
}
// 通道 0 发送 10 帧报文
ZCAN_Transmit_Data trans_data[10] = {};
for (int i = 0; i < 10; ++i){
get_can_frame(trans_data[i], i);
}
int send_count = ZCAN_Transmit(ch[0], trans_data, 10);
std::cout << "send frame: " << std::dec << send_count << std::endl;
// 通道 0 定时 100ms 发送 ID 为 0x100 CAN 报文，通道 1 定时 200ms 发送 ID 为 0x200 CANFD 报文
ZCAN_AUTO_TRANSMIT_OBJ auto_can;
ZCANFD_AUTO_TRANSMIT_OBJ auto_canfd;
memset(&auto_can, 0, sizeof(auto_can));
auto_can.index = 0; // 定时列表索引 0
auto_can.enable = 1; // 使能此索引，每条可单独设置
auto_can.interval = 100; // 定时发送间隔 100ms
get_can_frame(auto_can.obj, 0x100); // 构造 CAN 报文
prop->SetValue("0/auto_send", (const char*)&auto_can); // 设置定时发送
memset(&auto_canfd, 0, sizeof(auto_canfd));
auto_canfd.index = 1; // 定时列表索引 1
auto_canfd.enable = 1; // 使能此索引，每条可单独设置
auto_canfd.interval = 200; // 定时发送间隔 500ms
get_canfd_frame(auto_canfd.obj, 0x200); // 构造 CANFD 报文
prop->SetValue("1/auto_send_canfd", (const char*)&auto_canfd); // 设置定时发送
prop->SetValue("0/apply_auto_send", "0"); // 使能通道 0 定时发送
prop->SetValue("1/apply_auto_send", "0"); // 使能通道 1 定时发送
// 获取 0 通道定时发送 CAN 报文列表
int auto_count = *(prop->GetValue("0/get_auto_send_can_count/1"));
if (auto_count > 0)
{
ZCAN_AUTO_TRANSMIT_OBJ* pAuto;
pAuto = (ZCAN_AUTO_TRANSMIT_OBJ*)prop->GetValue("0/get_auto_send_can_data/1");
std::cout << "ch0 auto send list count: " << auto_count << std::endl;
for (int i = 0; i < auto_count; ++i,pAuto++)
{
std::cout << "index " << pAuto->index << "\tID " << pAuto->obj.frame.can_id << std::endl;
} }
// 获取 1 通道定时发送 CANFD 报文列表
auto_count = *(prop->GetValue("1/get_auto_send_canfd_count/1"));
if (auto_count > 0)
{
ZCANFD_AUTO_TRANSMIT_OBJ* pAuto;
pAuto = (ZCANFD_AUTO_TRANSMIT_OBJ*)prop->GetValue("1/get_auto_send_canfd_data/1");
std::cout << "ch1 auto send list count: " << auto_count << std::endl;
for (int i = 0; i < auto_count; ++i, pAuto++)
{
std::cout << "index " << pAuto->index << "\tID " << pAuto->obj.frame.can_id << std::endl;
} }
Sleep(5000); // 等待发送 5s
prop->SetValue("0/clear_auto_send", "0"); // 清除通道 0 定时发送
prop->SetValue("1/clear_auto_send", "0"); // 清除通道 1 定时发送
// 队列发送及获取总线利用率
// 获取队列发送可用缓冲
int free_count = *((int*)prop->GetValue("0/get_device_available_tx_count/1"));
// 构造 50 条报文，报文 ID 从 0 递增，帧间隔 10ms 递增
if (free_count > 50) {
ZCAN_Transmit_Data tran_data[50] = {};
for (int i = 0; i < 50; ++i) {
get_can_frame_queue(tran_data[i], i, i * 10);
}
int ret_count = ZCAN_Transmit(ch[0], tran_data, 50);
}
// 获取通道 0 总线利用率，5 次，1000ms 一次
for (int i = 0; i < 5; ++i) {
BusUsage* pUse = (BusUsage*)prop->GetValue("0/get_bus_usage/1");
if (NULL != pUse) {
std::cout << "busload: " << pUse->nBusUsage << "%" << std::endl;
}
Sleep(1000);
}
// 清空队列发送
prop->SetValue("0/clear_delay_send_queue", "0");
system("pause");
end:
g_thd_run = 0;
for (int i = 0; i < CH_COUNT; ++i) {
if (thd_handle[i].joinable())
thd_handle[i].join();
std::cout << "thread exit, close device" << std::endl;
}
ReleaseIProperty(prop);
ZCAN_CloseDevice(device);
system("pause");
return 0;
}
```