接口函数库（二次开发库）

使用说明书

说明书版本：V1.3

更新日期：2022.08.08

修订历史

版本	日期	说明
V1.0	2022.01.12	初版
V1.1	2022.03.23	新增加库函数（3.16~3.20 章节）
V1.2	2022.04.03	完善滤波配置；新增库函数（3.21~3.25 章节）
V1.3	2022.08.08	完善功能

目录

1 概述

用户如果只是利用 USBCANFD 设备进行 CAN/CANFD 总线调试，可以直接利用随机提供的 CANFD Tool 工具软件，进行收发数据的测试。

如果用户打算编写自己产品的软件程序。请认真阅读以下说明，并参考我们提供的 VC 示例代码。

开发用库文件：ControlCANFD.lib, ControlCANFD.dll

VC 平台函数声明文件：ControlCANFD.h,config.h

注意：ControlCANFD.lib, ControlCANFD.dll 依赖于 VC2008 运行库，该运行库一般的系统都会包含，只有在极个别的精减系统中没有，需要安装一下。

说明：本设备支持的二次开发接口函数及数据结构兼容 ZLG 的接口与数据结构。

2 数据结构定义

2.1 ZCAN_DEVICE_INFO

本结构体包含设备的一些基本信息，在函数 ZCAN_GetDeviceInf 中被填充。

成员

hw_Version

硬件版本号，16 进制，比如 0x0100 表示 V1.00。

fw_Version

固件版本号，16 进制。

dr_Version

驱动程序版本号，16 进制。

in_Version

接口库版本号，16 进制。

irq_Num

板卡所使用的中断号。

can_Num

表示有几路通道。

str_Serial_Num

此板卡的序列号，比如“USBCANFD0002”（注意：包括字符串结束符’\0’）。

str_hw_Type

硬件类型。

reserved

仅作保留，不设置。

2.2 ZCAN_CHANNEL_INIT_CONFIG

本结构体定义了通道初始化配置的参数，调用 ZCAN_InitCAN 之前，要先初始化该结构体。

成员

can_type

设备类型， =0 表示 CAN 设备，=1 表示 CANFD 设备。

- **CAN 设备**

  acc_code

  SJA1000 的帧过滤验收码，对经过屏蔽码过滤为“有关位”进行匹配，全部匹配成功后，此报文可以被接收，否则不接收。推荐设置为 0。

  acc_mask

  SJA1000 的帧过滤屏蔽码，对接收的 CAN 帧 ID 进行过滤，位为 0 的是“有关位”，8 位为 1 的是“无关位”。推荐设置为 0xFFFFFFFF，即全部接收。

reserved

仅作保留，不设置。

filter

滤波方式，=1 表示单滤波，=0 表示双滤波。

timing0

忽略，不设置。

timing1

忽略，不设置。

mode

工作模式，=0 表示正常模式（相当于正常节点），=1 表示只听模式（只接收，不影响总线）。

CANFD 设备

acc_code

验收码，同 CAN 设备。

acc_mask

屏蔽码，同 CAN 设备。

abit_timing

忽略，不设置。

dbit_timing

忽略，不设置。

brp

波特率预分频因子，设置为 0。

filter

滤波方式，同 CAN 设备。

mode

模式，同 CAN 设备。

pad

数据对齐，不设置。

reserved

仅作保留，不设置。

注意：设备的波特率 (abit_timing 和 dbit_timing)采用 GetIProperty 设置，详见 5.2。

2.3 can_frame

本结构体包含了 CAN 报文信息。

```
typedef struct {
    canid_t can_id;  /* 32 bit MAKE_CAN_ID + EFF/RTR/ERR flags */
    BYTE    can_dlc; /* frame payload length in byte (0 .. CAN_MAX_DLEN) */
    BYTE    __pad;   /* padding */
    BYTE    __res0;  /* reserved / padding */
    BYTE    __res1;  /* reserved / padding */
    BYTE    data[CAN_MAX_DLEN]/* __attribute__((aligned(8)))*/;
}can_frame;
```

成员

can_id

帧 ID，32 位，高 3 位属于标志位，标志位含义如下：
第 31 位(最高位)代表扩展帧标志，=0 表示标准帧，=1 代表扩展帧，宏 IS_EFF 可获取该标志；
第 30 位代表远程帧标志，=0 表示数据帧，=1 表示远程帧，宏 IS_RTR 可获取该标志；
第 29 位代表错误帧标准，=0 表示 CAN 帧，=1 表示错误帧，目前只能设置为 0；
其余位代表实际帧 ID 值，使用宏 MAKE_CAN_ID 构造 ID，使用宏 GET_ID 获取 ID。

can_dlc

数据长度。

__pad

对齐，忽略。

__res0

仅作保留，不设置。

__res1

仅作保留，不设置。

data

报文数据，有效长度为 can_dlc。

2.4 canfd_frame

本结构体包含了 CANFD 报文信息。

```
typedef struct {
    canid_t can_id;  /* 32 bit MAKE_CAN_ID + EFF/RTR/ERR flags */
    BYTE    len;     /* frame payload length in byte */
    BYTE    flags;   /* additional flags for CAN FD,i.e error code */
    BYTE    __res0;  /* reserved / padding */
    BYTE    __res1;  /* reserved / padding */
    BYTE    data[CANFD_MAX_DLEN]/* __attribute__((aligned(8)))*/;
}canfd frame;
```

成员

can_id

帧 ID，同 2.3。

len

数据长度。

flags

额外标志，比如使用 CANFD 加速，则设置为宏 CANFD_BRS。

__res0

仅作保留，不设置。

__res1

仅作保留，不设置。

data

报文数据，有效长度为 len。

2.5 ZCAN_Transmit_Data

包含发送的 CAN 报文信息，在函数 ZCAN_Transmit 中使用。

成员

frame

报文数据信息，详见 2.3。

transmit_type

发送方式，0=正常发送，1=单次发送，2=自发自收，3=单次自发自收。

发送方式说明如下：

- 正常发送：在 ID 仲裁丢失或发送出现错误时，CAN 控制器会自动重发，直到发送成功，或发送超时，或总线关闭。
- 单次发送：在一些应用中，允许部分数据丢失，但不能出现传输延迟时，自动重发就没有意义了。在这些应用中，一般会以固定的时间间隔发送数据，自动重发会导致后面的数据无法发送，出现传输延迟。使用单次发送，仲裁丢失或发送错误，CAN 控制器不会重发报文。
- 自发自收：产生一次带自接收特性的正常发送，在发送完成后，可以从接收缓冲区中读到已发送的报文。
- 单次自发自收：产生一次带自接收特性的单次发送，在发送出错或仲裁丢失不会执行重发。在发送完成后，可以从接收缓冲区中读到已发送的报文。

2.6 ZCAN_TransmitFD_Data

结构体包含发送的 CANFD 报文信息，在函数 ZCAN_TransmitFD 中使用。

成员

frame

报文数据信息，详见 2.4。

transmit_type

发送方式，同 2.5。

2.7 ZCAN_Receive_Data

结构体包含接收的 CAN 报文信息，在函数 ZCAN_Receive 中使用。

成员

frame

报文数据信息，详见 2.3。

timestamp

时间戳，单位微秒，基于设备启动时间。

2.8 ZCAN_ReceiveFD_Data

结构体包含接收的 CANFD 报文信息，在函数 ZCAN_ReceiveFD 中使用。

成员

frame

报文数据信息，详见 2.4。

timestamp

时间戳，单位微秒。

2.9 IProperty

结构体详情如下，用于获取/设置设备参数信息，示例代码参考程序清单 5.2。

成员

SetValue

设置设备属性值，详见第 4 章属性列表。

GetValue

获取属性值。

GetPropertys

用于返回设备包含的所有属性。

3 接口库函数说明

3.1 ZCAN_OpenDevice

该函数用于打开设备。一个设备只能被打开一次。

```
DEVICE_HANDLE  ZCAN_OpenDevice(UINT device_type, UINT device_index, UINT reserved);
```

参数

device_type

设备类型，详见头文件 zlgcan.h 中的宏定义。

device_index

设备索引号，比如当只有一个 USBCANFD 时，索引号为 0，这时再插入一个 USBCANFD，那 么后面插入的这个设备索引号就是 1，以此类推。

reserved

仅作保留。

返回值

为 INVALID_DEVICE_HANDLE 表示操作失败，否则表示操作成功，返回设备句柄值，请保存该句柄值，往后的操作需要使用。

3.2 ZCAN_CloseDevice

该函数用于关闭设备，关闭设备和打开设备一一对应。

```
UINT  ZCAN_CloseDevice(DEVICE_HANDLE device_handle);
```

参数

device_handle

需要关闭的设备的句柄值，即 ZCAN_OpenDevice 成功返回的值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.3 ZCAN_GetDeviceInf

该函数用于获取设备信息。

```
UINT  ZCAN_GetDeviceInf(DEVICE_HANDLE device_handle, ZCAN_DEVICE_INFO* pInfo);
```

参数

device_handle

设备句柄值。

pInfo

设备信息结构体，详见 2.1。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.4 ZCAN_IsDeviceOnLine

该函数用于检测设备是否在线，仅支持 USB 系列设备。

```
UINT  ZCAN_IsDeviceOnLine(DEVICE_HANDLE device_handle);
```

参数

device_handle

设备句柄值。

返回值

设备在线=STATUS_ONLINE，不在线=STATUS_OFFLINE。

3.5 ZCAN_InitCAN

该函数用于初始化 CAN。

```
CHANNEL_HANDLE  ZCAN_InitCAN(DEVICE_HANDLE device_handle, UINT can_index, ZCAN_CHANNEL_INIT_CONFIG* pInitConfig);
```

参数

device_handle

设备句柄值。

can_index

通道索引号，通道 0 的索引号为 0，通道 1 的索引号为 1，以此类推。

pInitConfig

初始化结构，详见 2.2。

返回值

为 INVALID_CHANNEL_HANDLE 表示操作失败，否则表示操作成功，返回通道句柄值，请保存该句柄值，往后的操作需要使用。

3.6 ZCAN_StartCAN

该函数用于启动 CAN 通道。

```
UINT  ZCAN_StartCAN(CHANNEL_HANDLE channel_handle);
```

参数

channel_handle

通道句柄值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.7 ZCAN_ResetCAN

该函数用于复位 CAN 通道，可通过 ZCAN_StartCAN 恢复。

```
UINT  ZCAN_ResetCAN(CHANNEL_HANDLE channel_handle);
```

参数

channel_handle

通道句柄值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败

3.8 ZCAN_ClearBuffer

该函数用于清除库接收缓冲区。

```
UINT  ZCAN_ClearBuffer(CHANNEL_HANDLE channel_handle);
```

参数

channel_handle

通道句柄值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.9 ZCAN_Transmit

该函数用于发送 CAN 报文。

```
UINT  ZCAN_Transmit(CHANNEL_HANDLE channel_handle, ZCAN_Transmit_Data* pTransmit, UINT len);
```

参数

channel_handle

通道句柄值。

pTransmit

结构体 ZCAN_Transmit_Data 数组的首指针。

len

报文数目

返回值

返回实际发送成功的报文数目。

3.10 ZCAN_TransmitFD

该函数用于发送 CANFD 报文。

```
UINT  ZCAN_TransmitFD(CHANNEL_HANDLE channel_handle, ZCAN_TransmitFD_Data* pTransmit, UINT len);
```

参数

channel_handle

通道句柄值。

pTransmit

结构体 ZCAN_TransmitFD_Data 数组的首指针。

len

报文数目

返回值

返回实际发送成功的报文数目。

3.11 ZCAN_GetReceiveNum

获取缓冲区中 CAN 或 CANFD 报文数目。

```
UINT  ZCAN_GetReceiveNum(CHANNEL_HANDLE channel_handle, BYTE type);
```

参数

channel_handle

通道句柄值。

type

获取 CAN 或 CANFD 报文，0=CAN，1=CANFD。

返回值

返回报文数目。

3.12 ZCAN_Receive

该函数用于接收 CAN 报文，建议使用 ZCAN_GetReceiveNum 确保缓冲区有数据再使用。

```
UINT  ZCAN_Receive(CHANNEL_HANDLE channel_handle, ZCAN_Receive_Data* pReceive, UINT len, int wait_time DEF(-1));
```

参数

channel_handle

通道句柄值。

pReceive

结构体 ZCAN_Receive_Data 数组的首指针。

len

数组长度（本次接收的最大报文数目，实际返回值小于等于这个值）。

wait_time

缓冲区无数据，函数阻塞等待时间，单位毫秒。若为-1 则表示无超时，一直等待，默认值为-1。

返回值

返回实际接收的报文数目。

3.13 ZCAN_ReceiveFD

该函数用于接收 CANFD 数据，建议使用 ZCAN_GetReceiveNum 确保缓冲区有数据再使用。

```
UINT  ZCAN_ReceiveFD(CHANNEL_HANDLE channel_handle, ZCAN_ReceiveFD_Data* pReceive, UINT len, int wait_time DEF(-1));
```

参数

channel_handle

通道句柄值。

pReceive

结构体 ZCAN_ReceiveFD_Data 数组的首指针。

len

数组长度（本次接收的最大报文数目，实际返回值小于等于这个值）。

wait_time

缓冲区无数据，函数阻塞等待时间，单位毫秒。若为-1 则表示无超时，一直等待，默认值为-1。

返回值

返回实际接收的报文数目。

3.14 GetIProperty

该函数返回属性配置接口。

```
IProperty*  GetIProperty(DEVICE_HANDLE device_handle);
```

参数

device_handle

设备句柄值。

返回值

返回属性配置接口指针，详见 2.9，空则表示操作失败。

3.15 ReleaseIProperty

释放属性接口，与 GetIProperty 结对使用。

```
UINT  ReleaseIProperty(IProperty * pIProperty);
```

参数

pIProperty

GetIProperty 的返回值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.16 ZCAN_SetAbitBaud

该函数用于设置 CANFD 仲裁域波特率。当使用属性” n/canfd_abit_baud_rate”设置波特率失败时可调用此函数设置。如当开发环境为 LabView 时可以调用此函数接口设置 CANFD 仲裁波特率。

```
UINT FUNC_CALL ZCAN_SetAbitBaud(DEVICE_HANDLE device_handle, UINT can_index, UINT abitbaud);
```

参数

device_handle

设备句柄值。

can_index

通道索引号，通道 0 的索引号为 0，通道 1 的索引号为 1，以此类推。

abitbaud

仲裁域波特率值，见属性列表仲裁域波特率 value。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.17 ZCAN_SetDbitBaud

该函数用于设置 CANFD 数据域波特率。当使用属性” n/ canfd_dbit_baud_rate”设置波特率失败时可调用此函数设置。如当开发环境为 LabView 时可以调用此函数接口设置 CANFD 数据波特率。

```
UINT FUNC_CALL ZCAN_SetDbitBaud(DEVICE_HANDLE device_handle, UINT can_index, UINT dbitbaud);
```

参数

device_handle

设备句柄值。

can_index

通道索引号，通道 0 的索引号为 0，通道 1 的索引号为 1，以此类推。

dbitbaud

仲裁域波特率值，见属性列表数据域波特率 value。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.18 ZCAN_SetBaudRateCustom

该函数用于设置 CANFD 自定义波特率。当使用属性” n/baud_rate_custom”设置自定义波特率失败时可调用此函数设置。如当开发环境为 LabView 时可以调用此函数接口设置 CANFD 自定义波特率。

```
UINT FUNC_CALL ZCAN_SetBaudRateCustom(DEVICE_HANDLE device_handle, UINT can_index, char * RateCustom);
```

参数

device_handle

设备句柄值。

can_index

通道索引号，通道 0 的索引号为 0，通道 1 的索引号为 1，以此类推。

RateCustom

自定义波特率字符串，见属性列表自定义波特率 value。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.19 ZCAN_SetCANFDStandard

该函数用于设置 CANFD 标准类型。当使用属性” n/canfd_standard”设置 CANFD 标准失败时可调用此函数设置。如当开发环境为 LabView 时可以调用此函数接口设置 CANFD 标准。

```
UINT FUNC_CALL ZCAN_SetCANFDStandard(DEVICE_HANDLE device_handle, UINT can_index, UINT canfd_standard);
```

参数

device_handle

设备句柄值。

can_index

通道索引号，通道 0 的索引号为 0，通道 1 的索引号为 1，以此类推。

canfd_standard

CANFD 标准类型，0 为 CANFD ISO，1 为 CANFD BOSCH。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.20 ZCAN_SetResistanceEnable

该函数用于设置 CANFD 标准类型。当使用属性” n/initenal_resistance”设置 CANFD 标准失败时可调用此函数设置。如当开发环境为 LabView 时可以调用此函数接口设置 CANFD 标准。

```
UINT FUNC_CALL ZCAN_SetResistanceEnable(DEVICE_HANDLE device_handle, UINT can_index, UINT enable);
```

参数

device_handle

设备句柄值。

can_index

通道索引号，通道 0 的索引号为 0，通道 1 的索引号为 1，以此类推。

enable

0 为禁止终端电阻，1 为使能终端电阻。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.21 ZCAN_ClearFilter

该函数用于清除通道滤波设置。当使用属性” n/filter_clear”清除滤波失败时可调用此函数。如当开发环境为 LabView 时可以调用此函数接口清除滤波设置。此函数不单独调用，每次配置时，按照清除滤波设置、配置模式、配置起始 ID、配置结束 ID、滤波生效的顺序进行；如果要设置多条滤波，在清除滤波和滤波生效之间设置多条滤波即可。

```
UINT FUNC_CALL ZCAN_ClearFilter(CHANNEL_HANDLE channel_handle);
```

参数

channel_handle

通道句柄值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.22 ZCAN_SetFilterMode

该函数用于配置通道滤波模式。当使用属性” n/filter_mode”设置滤波模式失败时可调用此函数。如当开

发环境为 LabView 时可以调用此函数接口设置。此函数不单独调用，每次配置时，按照清除滤波设置、配置模式、配置起始 ID、配置结束 ID、滤波生效的顺序进行；如果要设置多条滤波，在清除滤波和滤波生效之间设置多条滤波即可。

```
UINT FUNC_CALL ZCAN_SetFilterMode(CHANNEL_HANDLE channel_handle, UINT mode);
```

参数

channel_handle

通道句柄值。

mode

模式，0 表示标准帧，1 表示扩展帧。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.23 ZCAN_SetFilterStartID

该函数用于配置通道滤波起始 ID。当使用属性” n/filter_start”设置起始 ID 失败时可调用此函数。如当开发环境为 LabView 时可以调用此函数接口设置。此函数不单独调用，每次配置时，按照清除滤波设置、配置模式、配置起始 ID、配置结束 ID、滤波生效的顺序进行；如果要设置多条滤波，在清除滤波和滤波生效之间设置多条滤波即可。

```
UINT FUNC_CALL ZCAN_SetFilterStartID(CHANNEL_HANDLE channel_handle, UINT startID);
```

参数

channel_handle

通道句柄值。

startID

起始 ID 值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.24 ZCAN_SetFilterEndID

该函数用于配置通道滤波结束 ID。当使用属性” n/filter_end”设置结束 ID 失败时可调用此函数。如当开发环境为 LabView 时可以调用此函数接口设置。此函数不单独调用，每次配置时，按照清除滤波设置、配置模式、配置起始 ID、配置结束 ID、滤波生效的顺序进行；如果要设置多条滤波，在清除滤波和滤波生效之间设置多条滤波即可。

```
UINT FUNC_CALL ZCAN_SetFilterEndID(CHANNEL_HANDLE channel_handle, UINT EndID);
```

参数

channel_handle

通道句柄值。

EndID

结束 ID 值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

3.25 ZCAN_AckFilter

该函数用于生效通道滤波设置。当使用属性” n/filter_ack”设置生效失败时可调用此函数。如当开发环境为 LabView 时可以调用此函数接口设置。此函数不单独调用，每次配置时，按照清除滤波设置、配置模式、配置起始 ID、配置结束 ID、滤波生效的顺序进行；如果要设置多条滤波，在清除滤波和滤波生效之间设置多条滤波即可。

```
UINT FUNC_CALL ZCAN_AckFilter(CHANNEL_HANDLE channel_handle);
```

参数

channel_handle

通道句柄值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

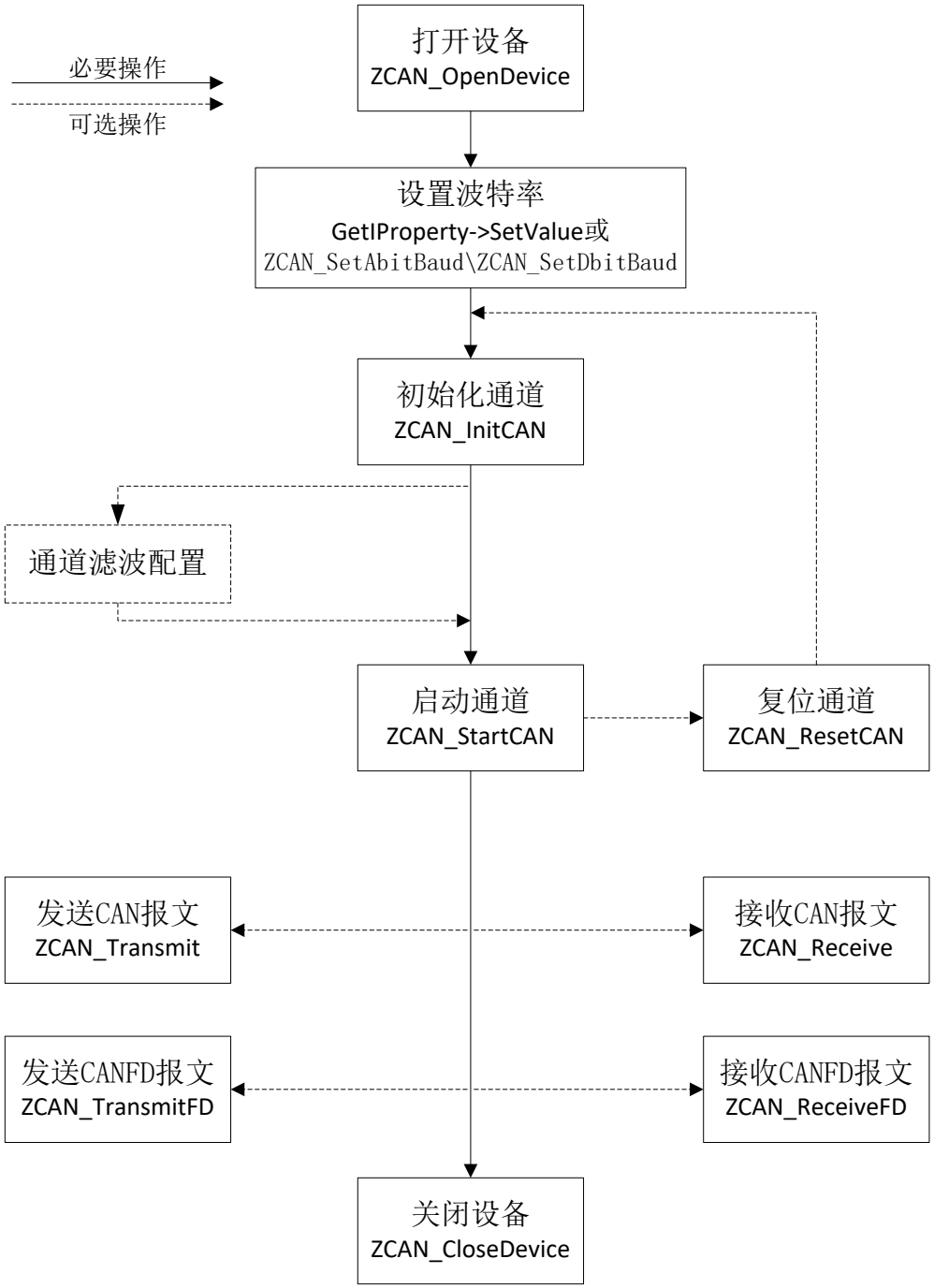
4 属性列表

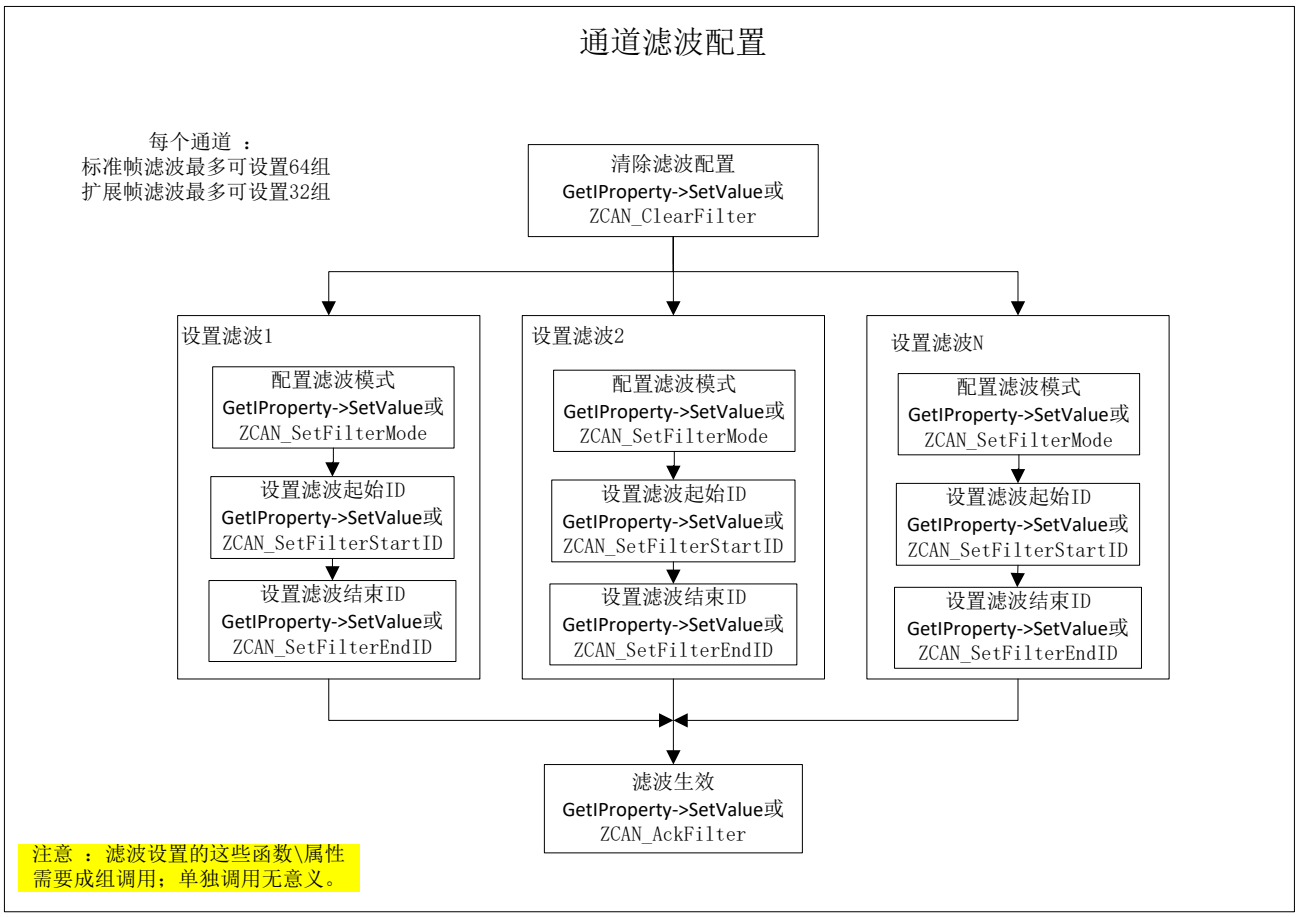
本设备支持的属性列表如下表所示。

参数	Path	value
仲裁域波特率	n/ canfd_abit_baud_rate n 表示通道号，0 代表通道 1,1 代表通道 2	1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 50000:50kbps 注：在 ZCAN_InitCAN 之前设置
数据域波特率	n/ canfd_dbit_baud_rate n 表示通道号，0 代表通道 1,1 代表通道 2	5000000:5Mbps 4000000:4Mbps 2000000:2Mbps 1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 注：在 ZCAN_InitCAN 之前设置
自定义波特率	n/baud_rate_custom n 表示通道号，0 代表通道 1,1 代表通道 2	使用波特率计算工具计算，详见《自定义波特率使用说明书》 注：在 ZCAN_InitCAN 之前设置
滤波模式	n/filter_mode n 表示通道号，0 代表通道 1,1 代表通道 2	“0”=标准帧 “1”=扩展帧 注：在 ZCAN_InitCAN 之后设置
滤波起始帧	n/filter_start n 表示通道号，0 代表通道 1,1 代表通道 2	“0x00000000”，16 进制字符 注：在 ZCAN_InitCAN 之后设置
滤波结束帧	n/filter_end n 表示通道号，0 代表通道 1,1 代表通道 2	“0x00000000”，16 进制字符 注：在 ZCAN_InitCAN 之后设置
清除滤波	n/filter_clear n 表示通道号，0 代表通道 1,1 代表通道 2	“0” 注：在 ZCAN_InitCAN 之后设置
滤波生效	n/filter_ack n 表示通道号，0 代表通道 1,1 代表通道 2	“0” 注：在 ZCAN_InitCAN 之后设置
终端电阻	n/initenal_resistance n 表示通道号，0 代表通道 1,1 代表通道 2	“0”=禁能 “1”=使能 注：在 ZCAN_InitCAN 之前设置
CANFD 标准	n/canfd_standard n 表示通道号，0 代表通道 1,1 代表通道 2	“0”=CANFD ISO “1”=CANFD BOSCH 注：在 ZCAN_InitCAN 之前设置

5.接口库函数使用流程

5.1 使用流程图




通道滤波配置
每个通道 ：
标准帧滤波最多可设置64组
扩展帧滤波最多可设置32组
清除滤波配置
GetIProperty->SetValue或
ZCAN_ClearFilter
设置滤波1
配置滤波模式
GetIProperty->SetValue或
ZCAN_SetFilterMode
设置滤波起始ID
GetIProperty->SetValue或
ZCAN_SetFilterStartID
设置滤波结束ID
GetIProperty->SetValue或
ZCAN_SetFilterEndID
设置滤波2
配置滤波模式
GetIProperty->SetValue或
ZCAN_SetFilterMode
设置滤波起始ID
GetIProperty->SetValue或
ZCAN_SetFilterStartID
设置滤波结束ID
GetIProperty->SetValue或
ZCAN_SetFilterEndID
设置滤波N
配置滤波模式
GetIProperty->SetValue或
ZCAN_SetFilterMode
设置滤波起始ID
GetIProperty->SetValue或
ZCAN_SetFilterStartID
设置滤波结束ID
GetIProperty->SetValue或
ZCAN_SetFilterEndID
滤波生效
GetIProperty->SetValue或
ZCAN_AckFilter
注意 ：滤波设置的这些函数\属性
需要成组调用；单独调用无意义。

5.2 示例代码

打开设备

```
m_DevType = ZCAN_USBCANFD_200U;
m_DevIndex=0;
DWORD Reserved=0;

//打开设备
m_dev = ZCAN_OpenDevice(m_DevType,m_DevIndex,Reserved);
if(INVALID_DEVICE_HANDLE == m_dev)
{
    MessageBox("open failed");
    return;
}
```

关闭设备

```
//关闭设备
if(STATUS_OK != ZCAN_CloseDevice(m_dev))
{
    MessageBox("Close failed! ");
    return;

}
MessageBox("Close successful!");
```

设置波特率

```
IProperty * _pPro = GetIProperty(m_dev);
const char * str;
if(_pPro == NULL)
{
    MessageBox("Property's NULL!");
    return;
}
//设置通道1 仲裁域波特率 500kbps
if( STATUS_OK != _pPro->SetValue("0/canfd_abit_baud_rate","500000") )
{
    MessageBox("Set ch0 rateA failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 数据域波特率 1Mbps
if( STATUS_OK != _pPro->SetValue("0/canfd_dbit_baud_rate","1000000") )
{
    MessageBox("Set ch0 rateD failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道2 仲裁域波特率 500kbps
if( STATUS_OK != _pPro->SetValue("1/canfd_abit_baud_rate","500000") )
{
    MessageBox("Set ch1 rateA failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道2 数据域波特率 1Mbps
if( STATUS_OK != _pPro->SetValue("1/canfd_dbit_baud_rate","10000000") )
{
    MessageBox("Set ch1 rateD failed!");
    ReleaseIProperty(_pPro);
    return;
}
```

设置波特率 2

```
//设置通道1 仲裁域波特率 500kbps
if( STATUS_OK != ZCAN_SetAbitBaud(m_dev,0,500000) )
{
    MessageBox("Set ch0 rateA failed!");
    return;
}
//设置通道1 数据域波特率 1Mbps
if( STATUS_OK != ZCAN_SetDbitBaud(m_dev,0,1000000) )
{
    MessageBox("Set ch0 rateD failed!");
    return;
}
//设置通道2 仲裁域波特率 500kbps
if( STATUS_OK != ZCAN_SetAbitBaud(m_dev,1,500000) )
{
    MessageBox("Set ch1 rateA failed!");
    return;
}
//设置通道2 数据域波特率 1Mbps
if( STATUS_OK != ZCAN_SetDbitBaud(m_dev,1,1000000) )
{
    MessageBox("Set ch1 rateD failed!");
    return;
}
```

通道滤波设置【通道初始化之后、启动之前设置】

```
//清除通道1 filter
if( STATUS_OK != _pPro->SetValue("0/filter_clear","0") )
{
    MessageBox("clear ch0 filter failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 模式：标准帧滤波
if( STATUS_OK != _pPro->SetValue("0/filter_mode","0") )
{
    MessageBox("set ch0 filter  mode failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 起始ID：0x100
if( STATUS_OK != _pPro->SetValue("0/filter_start","0x000100") )
{
    MessageBox("set ch0 filter  start failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 结束ID：0x200
if( STATUS_OK != _pPro->SetValue("0/filter_end","0x000200") )
{
    MessageBox("set ch0 filter  end failed!");
    ReleaseIProperty(_pPro);
    return;
}
//生效通道1 filter
if( STATUS_OK != _pPro->SetValue("0/filter_ack","0") )
{
    MessageBox("set ch0 filter  ack failed!");
    ReleaseIProperty(_pPro);
    return;
}
```

通道滤波设置 2【通道初始化之后、启动之前设置】

```
//清除通道1 filter
if( STATUS_OK != ZCAN_ClearFilter(dev_ch1) )
{
    MessageBox("clear ch0 filter failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 模式：标准帧滤波
if( STATUS_OK != ZCAN_SetFilterMode(dev_ch1,0) )
{
    MessageBox("set ch0 filter  mode failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 起始ID：0x100
if( STATUS_OK != ZCAN_SetFilterStartID(dev_ch1,0x100) )
{
    MessageBox("set ch0 filter  start failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 结束ID：0x200
if( STATUS_OK != ZCAN_SetFilterEndID(dev_ch1,0x200) )
{
    MessageBox("set ch0 filter  end failed!");
    ReleaseIProperty(_pPro);
    return;
}
//生效通道1 filter
if( STATUS_OK != ZCAN_AckFilter(dev_ch1) )
{
    MessageBox("set ch0 filter  ack failed!");
    ReleaseIProperty(_pPro);
    return;
}
```

初始化并启动通道

```
ZCAN_CHANNEL_INIT_CONFIG cfg;
memset(&cfg, 0, sizeof(cfg));
cfg.can_type = TYPE_CANFD;  // FD设备
cfg.canfd.mode = 0;         //正常模式
cfg.canfd.filter = 0;
cfg.canfd.pad = 0;
cfg.canfd.brp = 0;
//cfg.canfd.abit_timing = 0;
//cfg.canfd.dbit_timing = 0;
cfg.canfd.acc_code = 0;
cfg.canfd.acc_mask = 0xffffffff;
cfg.canfd.reserved = 0;

//初始化通道1
dev_ch1 = ZCAN_InitCAN(m_dev,0,&cfg);
if(INVALID_CHANNEL_HANDLE == dev_ch1)
{
    MessageBox("Init-CAN0 failed!");
    ReleaseIProperty(_pPro);
    return;
}
//启动通道1
if(STATUS_ERR == ZCAN_StartCAN(dev_ch1))
{
    MessageBox("Start-CAN0 failed!");
    ReleaseIProperty(_pPro);
    return;
}
```

发送数据

```
//向通道1发送CAN帧
ZCAN_Transmit_Data can_data;
can_data.frame.can_id = MAKE_CAN_ID(0x100, 0, 0, 0);
can_data.frame.can_dlc = 8;
for(i=0;i<can_data.frame.can_dlc;i++)
    can_data.frame.data[i]=i;
can_data.transmit_type = 0; //正常发送

if( 1 != ZCAN_Transmit(dev_ch1, &can_data, 1) )
{
    MessageBox("send failed\n");
    return;
}

//向通道1发送CANFD帧
ZCAN_TransmitFD_Data canfd_data;
canfd_data.frame.can_id = MAKE_CAN_ID(0x200, 0, 0, 0);
canfd_data.frame.len = 64;
for(i=0;i<canfd_data.frame.len;i++)
    canfd_data.frame.data[i]=i;
canfd_data.transmit_type = 0; //正常发送

if( 1 != ZCAN_TransmitFD(dev_ch1, &canfd_data, 1) )
{
    MessageBox("sendFD failed\n");
    return;
}
```

接收数据

```
ZCAN_Receive_Data   pCanObj0[2500];
ZCAN_ReceiveFD_Data pCanObjFD0[2500];

//获取通道1缓冲区CAN报文数目
can0_num=ZCAN_GetReceiveNum(dev_ch1,0);
if(can0_num)
{
    UINT ReadLen=0;
    //如果缓冲区有数据就读取
    ReadLen = ZCAN_Receive(dev_ch1, pCanObj0, can0_num, 50);
    RV_CAN0_NUMS += ReadLen;
    can0_num = 0;
    dlg->SetDlgItemInt(IDC_RECV_NUM, RV_CAN0_NUMS, TRUE);
}

//获取通道1缓冲区CANFD报文数目
can0fd_num=ZCAN_GetReceiveNum(dev_ch1,1);
if(can0fd_num)
{
    UINT ReadLen=0;
    //如果缓冲区有数据就读取
    ReadLen = ZCAN_ReceiveFD(dev_ch1, pCanObjFD0, can0fd_num, 50);
    RV_CANFD0_NUMS += ReadLen;
    can0fd_num = 0;
    dlg->SetDlgItemInt(IDC_RECVFD_NUM, RV_CANFD0_NUMS, TRUE);
}
```