广成科技 USB CANFD

接口函数库（ECanFDVci.dll）

使用手册

V2.0

修订历史

版本	日期	原因
V1.0	2019/07/22	创建文档
V1.1	2020/04/27	修改函数
V1.2	2020/08/11	增加函数
V2.0	2021/04/22	修改函数

目 录

1 接口函数库说明及其使用

1.1 接口卡设备类型定义

各个接口卡的类型定义如下：

设备名称	设备类型号
USBCANFD	6

1.2 错误码定义

名称	值	描述
ERR_CAN_NOINIT	0x00000001	CAN未初始化
ERR_CAN_DISABLE	0x00000002	CAN未使能
ERR_CAN_BUSOFF	0x00000004	CAN已经离线
	0x00000008	
ERR_DATA_LEN	0x00000010	数据长度错误
ERR_USB_WRITE	0x00000020	USB写数据失败
ERR_USB_READ	0x00000040	USB读数据失败
	0x00000080	
ERR_DEVICEOPENED	0x00000100	设备已经打开
ERR_DEVICEOPEN	0x00000200	打开设备错误
ERR_DEVICENOTOPEN	0x00000400	设备没有打开
ERR_BUFFEROVERFLOW	0x00000800	缓冲区溢出
ERR_DEVICENOTEXIST	0x00001000	此设备不存在
ERR_DEVICECLOSE	0x00002000	关闭设备失败
	0x00004000	
	0x00008000	

1.3 接口库函数结构体

1.3.1 CANFDFRAME_TYPE

描述

定义CANFD信息帧的类型，可以进行字节操作或位操作

```
typedef struct _CANFDFRAME_TYPE
{
    BYTE    proto          : 1;    // CAN/CANFD
    BYTE    format         : 1;    // STD/EXD
    BYTE    type           : 1;    // DATA/RTR
    BYTE    bitratemode    : 1;    // bitrateswitch on/off
    BYTE    reserved       : 4;
}CANFDFRAME_TYPE;
```

proto

是否是CANFD帧。=0时为标准CAN帧，=1时为CANFD帧。

format

是否是扩展帧。=0时为标准帧（11位帧ID），=1时为扩展帧（29位帧ID）。

type

是否是远程帧。=0时为数据帧，=1时为远程帧。

bitratemode

是否是可变数据波特率。=0时为数据波特率不可变，=1时为数据波特率可变。

1.3.2 TIMESTAMP_TYPE

描述

定义CANFD的时间戳，占8个字节

```
typedef struct _TIMESTAMP_TYPE
{
    BYTE    mday;
    BYTE    hour;
    BYTE    minute;
    BYTE    second;
    WORD    millisecond;
    WORD    microsecond;
} TIMESTAMP_TYPE;
```

1.3.3 CANFD_OBJ

描述

CANFD_OBJ结构体表示帧的数据结构。在发送函数TransmitFD和接收函数ReceiveFD中被用来传送CANFD信息帧。

```
//定义CANFD信息帧的数据类型
typedef  struct  _CANFD_OBJ{            // 80字节
    CANFDFRAME_TYPE   CanORCanfdType; // 定义帧的类型
    BYTE    DataLen;              // 有效数据长度
    BYTE    Reserved[2];          // 保留
    DWORD   ID;            // CANID
    TIMESTAMP_TYPE TimeStamp;       // 时间戳
    BYTE    Data[64];             // 数据字节
}CANFD_OBJ,*P_CANFD_OBJ;
```

成员

ID

报文帧ID。

TimeStamp

表示接收到信息帧时的时间标识，使用绝对时间戳。包含日、时、分、秒、毫秒、微妙。

DataLen

数据长度DLC(<=64)，即Data的长度。

Data

CAN报文的数据。空间受DataLen的约束。

Reserved

系统保留。

1.3.4 CANFD_ECR_TYPE

描述

定义错误计数寄存器，占4个字节。

```
typedef struct _CANFD_ECR_TYPE
{
    DWORD TEC          : 8;
    DWORD REC          : 7;
    DWORD RP           : 1;
    DWORD CEL          : 8;
    DWORD              : 8;
} CANFD_ECR_TYPE;
```

1.3.5 CANFD_PSR_TYPE

描述

定义总线状态寄存器，占4个字节。

```
typedef struct _CANFD_PSR_TYPE
{
    DWORD LEC          : 3;
    DWORD ACT          : 2;
    DWORD EP           : 1;
    DWORD EW           : 1;
    DWORD BO           : 1;
    DWORD DLEC         : 3;
    DWORD RESI         : 1;
    DWORD RBRS         : 1;
    DWORD RFDF         : 1;
    DWORD PXE          : 1;
    DWORD              : 1;
    DWORD TDCV         : 7;
    DWORD              : 9;
} CANFD_PSR_TYPE;
```

1.3.6 CAN_STATUS

描述

CAN_STATUS结构体包含CANFD设备所有通道的的状态信息。

```
typedef  struct  _CANFD_STATUS{
    WORD LeftSendBufferNum;     // 未使用的发送缓冲数，被动发送模式
                                // 使用发动缓冲，主动模式不使用
    TIMESTAMP_TYPE    can0_timestamp;      // 时间戳
    CANFD_ECR_TYPE    can0_ecr_register;   // 错误计数寄存器
    CANFD_PSR_TYPE    can0_psr_register;   // 协议状态寄存器
    DWORD  can0_RxLost_Cnt;                // 接收失败计数
    DWORD  can0_TxFail_Cnt;                // 发送失败计数
    FLOAT  can0_Load_Rate;                 // 负载率

    TIMESTAMP_TYPE    can1_timestamp;      // 时间戳
    CANFD_ECR_TYPE    can1_ecr_register;   // 错误计数寄存器
    CANFD_PSR_TYPE    can1_psr_register;   // 协议状态寄存器
    DWORD  can1_RxLost_Cnt;                // 接收失败计数
    DWORD  can1_TxFail_Cnt;                // 发送失败计数
    FLOAT  can1_Load_Rate;                 // 负载率
}CANFD_STATUS, *P_CANFD_STATUS;
```

1.3.7 ERR_FRAME

描述

ERR_FRAME定义了错误帧的数据结构

```
typedef  struct  _ERR_FRAME{                    // 错误帧
    TIMESTAMP_TYPE    can_timestamp;            // 时间戳
    CANFD_ECR_TYPE    can_ecr_register;         // 错误计数寄存器
    CANFD_PSR_TYPE    can_psr_register;         // 协议状态寄存器
}ERR_FRAME, *P_ERR_FRAME;
```

1.3.8 INIT_CONFIG

描述

INIT_CONFIG结构体定义了初始化CAN的配置。结构体将在InitCANFD函数中被填充。在初始化之前，要先填好这个结构体变量。

```
typedef struct _INIT_CONFIG{                    // 78字节
    BYTE    CanReceMode;                        // 接收模式设置
    BYTE    CanSendMode;                        // 发送模式设置
    DWORD   NominalBitRate;                     // 不需要用户设定   Nominal Bit Rate
    DWORD   DataBitRate;                        // 不需要用户设定   Data Bit Rate
                                                // 在数据波特率可变时才是有意义的
    BYTE_TYPE   FilterUsedBits;                 // 定义接收模式为 SPECIFIED_RECEIVE
                                                // 时，决定滤波器1~8的是否被使用
    BYTE_TYPE   StdOrExdBits;                   // 定义接收模式为 SPECIFIED_RECEIVE
                                                // 时，决定滤波器1~8属于标准还是扩展
    BYTE    NominalBitRateSelect;               // 需要用户设定
    BYTE    DataBitRateSelect;                  // 需要用户设定
    // 定义接收模式为 SPECIFIED_RECEIVE 时，下面的8个滤波器的配置
    // 才有意义，并且至少需要使用和配置一个

    // 滤波器1的CANID设置和屏蔽设置
    DWORD   StandardORExtendedfilter1;      // Standard Or Extended filter1
    DWORD   StandardORExtendedfilter1Mask;  // Standard Or Extended filter1 Mask
    // 滤波器2的CANID设置和屏蔽设置
    DWORD   StandardORExtendedfilter2;      // Extended Or Extended filter2
    DWORD   StandardORExtendedfilter2Mask;  // Extended Or Extended filter2 Mask
    // 滤波器3的CANID设置和屏蔽设置
    DWORD   StandardORExtendedfilter3;      // Standard Or Extended filter3
    DWORD   StandardORExtendedfilter3Mask;  // Standard Or Extended filter3 Mask
    // 滤波器4的CANID设置和屏蔽设置
    DWORD   StandardORExtendedfilter4;      // Extended Or Extended filter4
    DWORD   StandardORExtendedfilter4Mask;  // Extended Or Extended filter4 Mask
    // 滤波器5的CANID设置和屏蔽设置
```

```
    DWORD   StandardOrExtendedfilter5;      // Standard Or Extended filter5
    DWORD   StandardOrExtendedfilter5Mask;  // Standard Or Extended filter5 Mask
    // 滤波器6的CANID设置和屏蔽设置
    DWORD   StandardOrExtendedfilter6;      // Extended Or Extended filter6
    DWORD   StandardOrExtendedfilter6Mask;  // Extended Or Extended filter6 Mask
    // 滤波器7的CANID设置和屏蔽设置
    DWORD   StandardOrExtendedfilter7;      // Standard Or Extended filter5
    DWORD   StandardOrExtendedfilter7Mask;  // Standard Or Extended filter5 Mask
    // 滤波器8的CANID设置和屏蔽设置
    DWORD   StandardOrExtendedfilter8;      // Extended Or Extended filter6
    DWORD   StandardOrExtendedfilter8Mask;  // Extended Or Extended filter6 Mask
}INIT_CONFIG,*P_INIT_CONFIG;
```

1.3.9 BOARD_INFO

描述

BOARD_INFO结构体包含ECAN系列接口卡的设备信息。结构体将在GetReference函数中被填充。

```
typedef struct _BOARD_INFO {
    USHORT hw_Version;
    USHORT fw_Version;
    USHORT dr_Version;
    USHORT in_Version;
    USHORT irq_Num;
    BYTE   can_Num;
    CHAR   str_Serial_Num[20];
    CHAR   str_hw_Type[40];
    USHORT Reserved[4];
} BOARD_INFO, *P_BOARD_INFO;
```

成员

hw_Version

硬件版本号，用16进制表示。

fw_Version

固件版本号，用16进制表示。

dr_Version

驱动程序版本号，用16进制表示。

in_Version

接口库版本号，用16进制表示。

irq_Num

板卡所使用的中断号。

can_Num

表示有几路CAN通道。

str_Serial_Num

此板卡的序列号。

str_hw_Type

硬件类型。

Reserved

系统保留。

1.4 接口库函数说明

1.4.1 OpenDeviceFD

描述

此函数用以打开设备。

```
DWORD __stdcall OpenDeviceFD(DWORD DeviceType,DWORD DeviceInd)
```

参数

DeviceType

设备类型号。USBCANFD 选择6

DeviceInd

设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。

返回值

为STATUS_OK表示操作成功，其他参见错误码定义。

示例

```
#include "EcanFDVci.h"
DWORD status;
status = OpenDeviceFD(USBCANFD,0);
if(status == STATUS_OK)
printf("CANFD设备0打开成功\n");
```

1.4.2 CloseDeviceFD

描述

此函数用于关闭设备。

```
DWORD __stdcall CloseDeviceFD(DWORD DeviceType,DWORD DeviceInd)
```

参数

DeviceType

设备类型号。USBCANFD 选择6

DeviceInd

设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。

返回值

为STATUS_OK表示操作成功，其他参见错误码定义。

示例

```
#include "EcanFDVci.h"
DWORD status;
status = CloseDeviceFD (USBCANFD,0);
if(status == STATUS_OK)
```

```
printf("CANFD设备0关闭成功\n");
```

1.4.3 InitCANFD

描述

此函数用以初始化指定的CAN通道。

```
DWORD __stdcall InitCANFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd,
P_INIT_CONFIG pInitConfig)
```

参数

DeviceType

设备类型号。USBCANFD 选择6

DeviceInd

设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。

CANInd

第几路CAN。即对应CANFD设备的通道号，0为通道1，1为通道2。

pInitConfig

初始化参数结构

返回值

为STATUS_OK表示操作成功，其他参见错误码定义。

示例

```
#include "EcanFDVci.h"
DWORD status;
INIT_CONFIG Canfd_InitConfig;
// CanReceMode即CANFD的接收模式可以有4种选择：
// (1) SPECIFIED_RECEIVE : 特定接收模式，最多可以使用8个滤波器
// (2) GLOBAL_STANDARD_RECEIVE : 标准帧接收模式
// (3) GLOBAL_EXTENDED_RECEIVE : 扩展帧接收模式
// (4) GLOBAL_STANDARD_AND_EXTENDED_RECEIVE : 标准帧和扩展帧都接收模式
Canfd_InitConfig.CanReceMode = GLOBAL_STANDARD_AND_EXTENDED_RECEIVE;
// CanSendMode即CANFD的发送模式可以有2种选择：
// (1) POSITIVE_SEND : 主动发送模式，不用发送缓冲，需要等数据都发完才能再次发送数据
// (2) PASSIVE_SEND : 被动发送模式，使用发送缓冲，只要发送缓冲没用完就能发送数据
Canfd_InitConfig.CanSendMode = PASSIVE_SEND;
Canfd_InitConfig.NominalBitRateSelect = BAUDRATE_1M; // 标准波特率
Canfd_InitConfig.DataBitRateSelect = DATARATE_5M;    // 数据波特率
// 配置为特定接收模式时，才需要对滤波器进行详细的设置
if(Canfd_InitConfig.CanReceMode == SPECIFIED_RECEIVE) // 配置为特定接收模式时，
                                                      // 才需要对滤波器进行详细的设置
{
//  Canfd_InitConfig.FilterUsedBits.as_byte = 0xff; // 8个滤波器全部使用
    Canfd_InitConfig.FilterUsedBits.Bit0 = USED;
    Canfd_InitConfig.FilterUsedBits.Bit1 = USED;
    Canfd_InitConfig.FilterUsedBits.Bit2 = USED;
    Canfd_InitConfig.FilterUsedBits.Bit3 = USED;
    Canfd_InitConfig.FilterUsedBits.Bit4 = USED;
```

```
    Canfd_InitConfig.FilterUsedBits.Bit5 = USED;
    Canfd_InitConfig.FilterUsedBits.Bit6 = USED;
    Canfd_InitConfig.FilterUsedBits.Bit7 = USED;

//  Canfd_InitConfig.StdOrExdBits.as_byte = 0xf0;    // 前4个是标准，后4个是扩展
    Canfd_InitConfig.StdOrExdBits.Bit0 = STANDARD_FORMAT;
    Canfd_InitConfig.StdOrExdBits.Bit1 = STANDARD_FORMAT;
    Canfd_InitConfig.StdOrExdBits.Bit2 = STANDARD_FORMAT;
    Canfd_InitConfig.StdOrExdBits.Bit3 = STANDARD_FORMAT;
    Canfd_InitConfig.StdOrExdBits.Bit4 = EXTENDED_FORMAT;
    Canfd_InitConfig.StdOrExdBits.Bit5 = EXTENDED_FORMAT;
    Canfd_InitConfig.StdOrExdBits.Bit6 = EXTENDED_FORMAT;
    Canfd_InitConfig.StdOrExdBits.Bit7 = EXTENDED_FORMAT;

    Canfd_InitConfig.StandardORExtendedfilter1 = 0x120;          // 0x120
    Canfd_InitConfig.StandardORExtendedfilter1Mask = 0x7ff;
    Canfd_InitConfig.StandardORExtendedfilter2 = 0x121;          // 0x121
    Canfd_InitConfig.StandardORExtendedfilter2Mask = 0x7ff;
    Canfd_InitConfig.StandardORExtendedfilter3 = 0x122;          // 0x122
    Canfd_InitConfig.StandardORExtendedfilter3Mask = 0x7ff;
    Canfd_InitConfig.StandardORExtendedfilter4 = 0x130;          // 0x130~0x13f
    Canfd_InitConfig.StandardORExtendedfilter4Mask = 0x7f0;
    Canfd_InitConfig.StandardORExtendedfilter5 = 0x10000120;     // 0x100000120
    Canfd_InitConfig.StandardORExtendedfilter5Mask = 0x1fffffff;
    Canfd_InitConfig.StandardORExtendedfilter6 = 0x10000121;     // 0x100000121
    Canfd_InitConfig.StandardORExtendedfilter6Mask = 0x1fffffff;
    Canfd_InitConfig.StandardORExtendedfilter7 = 0x10000122;     // 0x100000122
    Canfd_InitConfig.StandardORExtendedfilter7Mask = 0x1fffffff;
    Canfd_InitConfig.StandardORExtendedfilter8 = 0x10000130; // 0x10000130~0x1000013f
    Canfd_InitConfig.StandardORExtendedfilter8Mask = 0x1ffffff0;
}
status = InitCANFD(USBCANFD,0,0,&Canfd_InitConfig);
if(status == STATUS_OK)
printf("CANFD设备0通道1初始化成功\n");
```

1.4.4 StartCANFD

描述

此函数用以启动USBCANFD设备的某一个通道。如有多个N通道时，需要多次调用。在执行StartCANFD函数后，需要延迟10ms执行TransmitFD函数。

DWORD __stdcall StartCANFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd)

参数

DeviceType
设备类型号。USBCANFD 选择6
DeviceInd
设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。
CANInd
第几路CAN。即对应CANFD设备的通道号，0为通道1，1为通道2。

返回值

为STATUS_OK表示操作成功，其他参见错误码定义。

示例

```
#include "EcanFDVci.h"
DWORD status;
status = StartCANFD(USBCANFD,0,0);
if(status == STATUS_OK)
printf("CANFD设备0通道1启动成功\n");
```

1.4.5 StopCANFD

描述

此函数用以停止USBCANFD设备的某一个通道。如有多个N通道时，需要多次调用。

DWORD __stdcall StopCANFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd)

参数

DeviceType
设备类型号。USBCANFD 选择6
DeviceInd
设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。
CANInd
第几路CAN。即对应CANFD设备的通道号，0为通道1，1为通道2。

返回值

为STATUS_OK表示操作成功，其他参见错误码定义。

示例

```
#include "EcanFDVci.h"
DWORD status;
status = StopCANFD(USBCANFD,0,0);
if(status == STATUS_OK)
printf("CANFD设备0通道1停止成功\n");
```

1.4.6 ResetCANFD

描述

此函数用以复位CANFD的一个通道。如当USBCANFD设备的一个通道进入总线关闭状态时，可以调用这个函数。

```
DWORD __stdcall ResetCANFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd)
```

参数

DeviceType

设备类型号。USBCANFD 选择6

DeviceInd

设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。

CANInd

第几路CAN。即对应CANFD设备的通道号，0为通道1，1为通道2。

返回值

为STATUS_OK表示操作成功，其他参见错误码定义。

示例

```
#include "EcanFDVci.h"
DWORD status;
status = ResetCANFD(USBCANFD,0,0);
if(status == STATUS_OK)
printf("CANFD设备0通道1复位成功\n");
```

1.4.7 TransmitFD

描述

```
DWORD __stdcall TransmitFD(DWORD DeviceType,DWORD DeviceInd,BYTE CANInd,P_CANFD_OBJ pCanfdMQ,DWORD Len)
```

参数

DeviceType

设备类型号。USBCANFD 选择6

DeviceInd

设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。

CANIndex

第几路CAN。即对应CANFD设备的通道号，0为通道1，1为通道2。

pSend

要发送的数据帧数组的首指针。

Len

要发送的数据帧数组的长度。

返回值

为STATUS_OK表示操作成功，其他参见错误码定义。

示例

```
#include "EcanFDVci.h"
DWORD status;
CANFD_OBJ Canfd0_TXMessageQueue[10];    // CAN发送消息队列深度为10
DWORD Canfd0_TxMQ_Num = 10; // 发送10帧数据
status = TransmitFD(USBCANFD, 0, 0, Canfd0_TXMessageQueue, Canfd0_TxMQ_Num);
if(status == STATUS_OK)
printf("CANFD设备0通道1成功收到要发送的CANFD数据\n");
```

1.4.8 ReceiveFD

描述

此函数从指定的CANFD设备通道的缓冲区里读取数据。

```
DWORD __stdcall ReceiveFD(DWORD DeviceType,DWORD DeviceInd,BYTE CANInd,
P_CANFD_OBJ pCanfdMQ,DWORD *Len)
```

参数

DeviceType

设备类型号。USBCANFD 选择6

DeviceInd

设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。

CANInd

第几路CAN。即对应CANFD设备的通道号，0为通道1，1为通道2。

pCanfdMQ

用来接收的数据帧数组的首指针。

Len

用来接收的数据帧数组的长度。

返回值

为STATUS_OK表示操作成功，其他参见错误码定义。

示例

```
#include "EcanFDVci.h"
DWORD status;
CANFD_OBJ Canfd0_RXMessageQueue[CANFD_RECEFBUFFER_MAX_NUMBER];
DWORD Canfd0_RxMQ_Num = 0;
status = ReceiveFD(USBCANFD, 0, 0, Canfd0_RXMessageQueue, &Canfd0_RxMQ_Num);
if(status == STATUS_OK)
printf("CANFD设备0通道1收到%d个数据帧\n",Canfd0_RxMQ_Num);
```

1.4.9 GetErrFrame

描述

此函数从指定的CANFD设备通道的缓冲区里读取错误帧数据。

```
DWORD __stdcall GetErrFrame(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd, P_ERR_FRAME pCanfdErrbuffer, DWORD *Len)
```

参数

DeviceType

设备类型号。USBCANFD 选择6

DeviceInd

设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。

CANInd

第几路CAN。即对应CANFD设备的通道号，0为通道1，1为通道2。

pCanfdErrbuffer

用来接收的错误帧数组的首指针。

Len

用来接收的错误帧数组的长度。

返回值

为STATUS_OK表示操作成功，其他参见错误码定义。

示例

```
#include "EcanFDVci.h"
DWORD status;
ERR_FRAME Canfd0_ErrFrameQueue[CANFD_ERRFRAME_MAX_NUMBER];
DWORD Canfd0_ErrFrame_Num = 0;
status = GetErrFrame(USBCANFD, 0, 0, Canfd0_ErrFrameQueue, &Canfd0_ErrFrame_Num);
if(status == STATUS_OK)
printf("CANFD设备0通道1收到%d个错误帧\n",Canfd0_ErrFrame_Num);
```

1.4.10 GetCanfdBusStatus

描述

此函数从指定的CANFD设备获取通道1和通道2的总线状态信息。

```
DWORD __stdcall GetCanfdBusStatus(DWORD DeviceType, DWORD DeviceInd, P_CANFD_STATUS p_canfd_status)
```

参数

DeviceType

设备类型号。USBCANFD 选择6

DeviceInd

设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。
p_canfd_status

返回值

为CANFD_STATUS的结构体。

示例

```
#include "EcanFDVci.h"
CANFD_STATUS device0_canfd_status;
GetCanfdBusStatus(USBCANFD, 0, &device0_canfd_status);
printf("CANFD设备0通道1发送失败计数：%d\n", device0_canfd_status.can0_TxFail_Cnt);
printf("CANFD设备0通道2发送失败计数：%d\n", device0_canfd_status.can1_TxFail_Cnt);
printf("CANFD设备0通道1接收失败计数：%d\n", device0_canfd_status.can0_RxLost_Cnt);
printf("CANFD设备0通道2接收失败计数：%d\n", device0_canfd_status.can1_RxLost_Cnt);
```

1.4.11 GetReference

描述

此函数用以获取设备的相应参数。

```
DWORD __stdcall GetReference(DWORD DevType, DWORD DevIndex, DWORD CANIndex, DWORD RefType, PVOID pData);
```

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为0，有两个时可以为0或1。

CANIndex

第几路CAN。

RefType

参数类型。

pData

用来存储参数有关数据缓冲区地址首指针。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ECanFDVci.h"
int nDeviceType = 6; //  USBCAN FD
int nDeviceInd = 0;
int nCANInd = 0;
BOARD_INFO info;
```

```
DWORD dwRel;
dwRel = GetReference(nDeviceType, nDeviceInd, nCANInd, 1, (PVOID)info);

//reftype=1,读取设备信息
```

1.5 接口库函数使用方法

首先，把库函数文件都放在工作目录下。库函数文件总共有5个文件：
32位系统需要ECanFDVci.h、ECanFDVci.lib、ECanFDVci.dll这3个文件
64位系统需要ECanFDVci.h、ECanFDVci64.lib、ECanFDVci64.dll这3个文件

其次，在扩展名为.CPP 的文件中包含 ECanFDVci.h 头文件。
如：#include “ECanFDVci.h”
最后，在工程的连接器设置中连接到 ECanFDVci.lib 或 ECanFDVci64.lib 文件。
在 CPP 文件中加入静态库调用：

```
#ifdef _WIN64
    #pragma comment(lib, "ECANFDVCI64.lib")
#else
    #pragma comment(lib, "ECANFDVCI.lib")
#endif
```

1.6 接口库函数

OpenDeviceFD(DWORD DeviceType, DWORD DeviceInd);

CloseDeviceFD(DWORD DeviceType, DWORD DeviceInd);

InitCANFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd, P_INIT_CONFIG pInitConfig);

TransmitFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd, P_CANFD_OBJ pCanfdMQ, DWORD Len);

Receive_buffer_thread(DWORD DeviceType, DWORD DeviceInd, DWORD WaitTime);

ReceiveFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd, P_CANFD_OBJ pCanfdMQ, DWORD *Len);

GetErrFrame(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd, P_ERR_FRAME pCanfdErrbuffer, DWORD *Len);

ResetCANFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd);

StartCANFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd);

StopCANFD(DWORD DeviceType, DWORD DeviceInd, BYTE CANInd);

GetCanfdBusStatus(DWORD DeviceType, DWORD DeviceInd, P_CANFD_STATUS p_canfd_status);

GetReference(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,DWORD RefType,PVOID pData);

SetReference(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,DWORD RefType,PVOID pData);

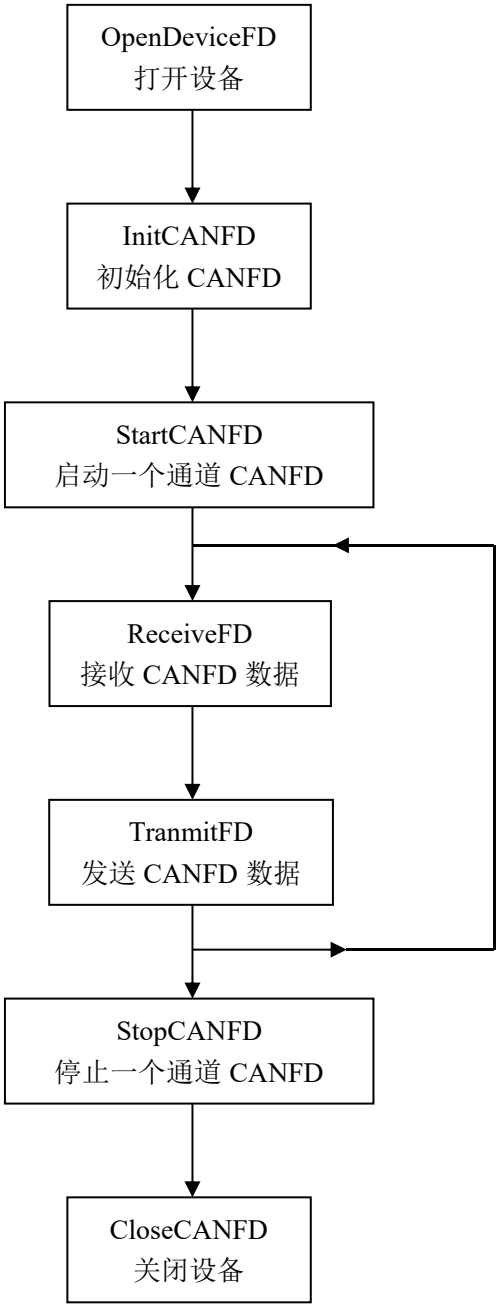
1.7 二次开发最简流程图

参数结构体封装

INIT_CONFIG
板卡初始化结构体

对 INIT_CONFIG 结构体填充

函数库调用流程



2. 免责声明

感谢您购买广成科技的 GCAN 系列软硬件产品。GCAN 是沈阳广成科技有限公司的注册商标。本产品及手册为广成科技版权所有。未经许可，不得以任何形式复制翻印。在使用之前，请仔细阅读本声明，一旦使用，即被视为对本声明全部内容的认可和接受。请严格遵守手册、产品说明和相关的法律法规、政策、准则安装和使用该产品。在使用产品过程中，用户承诺对自己的行为及因此而产生的所有后果负责。因用户不当使用、安装、改装造成的任何损失，广成科技将不承担法律责任。

关于免责声明的最终解释权归广成科技所有。

销售与服务

沈阳广成科技有限公司

地址：辽宁省沈阳市浑南区长青南街 135-21 号 5 楼

邮编：110000

网址：www.gcgd.net

全国销售与服务电话：400-6655-220

售前服务电话与微信号：13889110770

售前服务电话与微信号：18309815706

售后服务电话与微信号：13840170070

售后服务电话与微信号：17602468871