

广成科技 USB CAN
接口函数库（ECanVci.dll）
使用手册
V5.5

修订历史

版本	日期	原因
V5.1	2013/06/16	创建文档
V5.2	2015/09/10	添加部分参数
V5.3	2017/05/19	修改部分参数
V5.31	2017/10/20	修改部分参数
V5.4	2022/05/07	新增滤波功能、修改函数
V5.5	2022/10/31	修改部分参数、增加函数用法样例、修改结构

目 录

1 接口函数库说明及其使用.....	1
1.1 接口卡设备类型定义.....	1
1.2 开发流程图.....	1
1.3 错误码定义.....	4
1.4 接口库函数结构体.....	5
1.4.1 BOARD_INFO	5
1.4.2 CAN_OBJ	6
1.4.3 CAN_STATUS	7
1.4.4 ERR_INFO.....	8
1.4.5 INIT_CONFIG	9
1.4.6 FILTER_RECORD	11
1.5 接口库函数说明.....	12
1.5.1 OpenDevice.....	12
1.5.2 CloseDevice	13
1.5.3 InitCan.....	14
1.5.4 ReadBoardInfo	15
1.5.5 ReadErrInfo.....	16
1.5.6 ReadCanStatus	20
1.5.7 GetReference.....	21
1.5.8 SetReference	22
1.5.9 GetReceiveNum	24
1.5.10 ClearBuffer.....	25
1.5.11 StartCAN	26
1.5.12 Transmit	27
1.5.13 Receive.....	28
1.5.14 ResetCAN	29
1.6 接口库函数使用方法.....	30
1.6.1 VC 调用动态库的方法	30
1.6.2 VB 调用动态库的方法	30
1.7 接口库函数.....	32
1.8 常用功能样例.....	33
1.8.1 滤波设置.....	33
1.8.2 读取设备 SN.....	34
1.8.3 批量读取数据.....	35

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表1.1所示。

表 1.1 接口卡的类型定义

设备名称	设备类型号
USBCAN-I	3
USBCAN-II	4

1.2 开发流程图

本开发流程适用于我司出品的 USBCAN 系列接口卡(USBCAN II FD 除外)，请按照开发流程进行二次开发，流程如图 1.1 所示，开发代码示例如程序清单 1.1 所示。

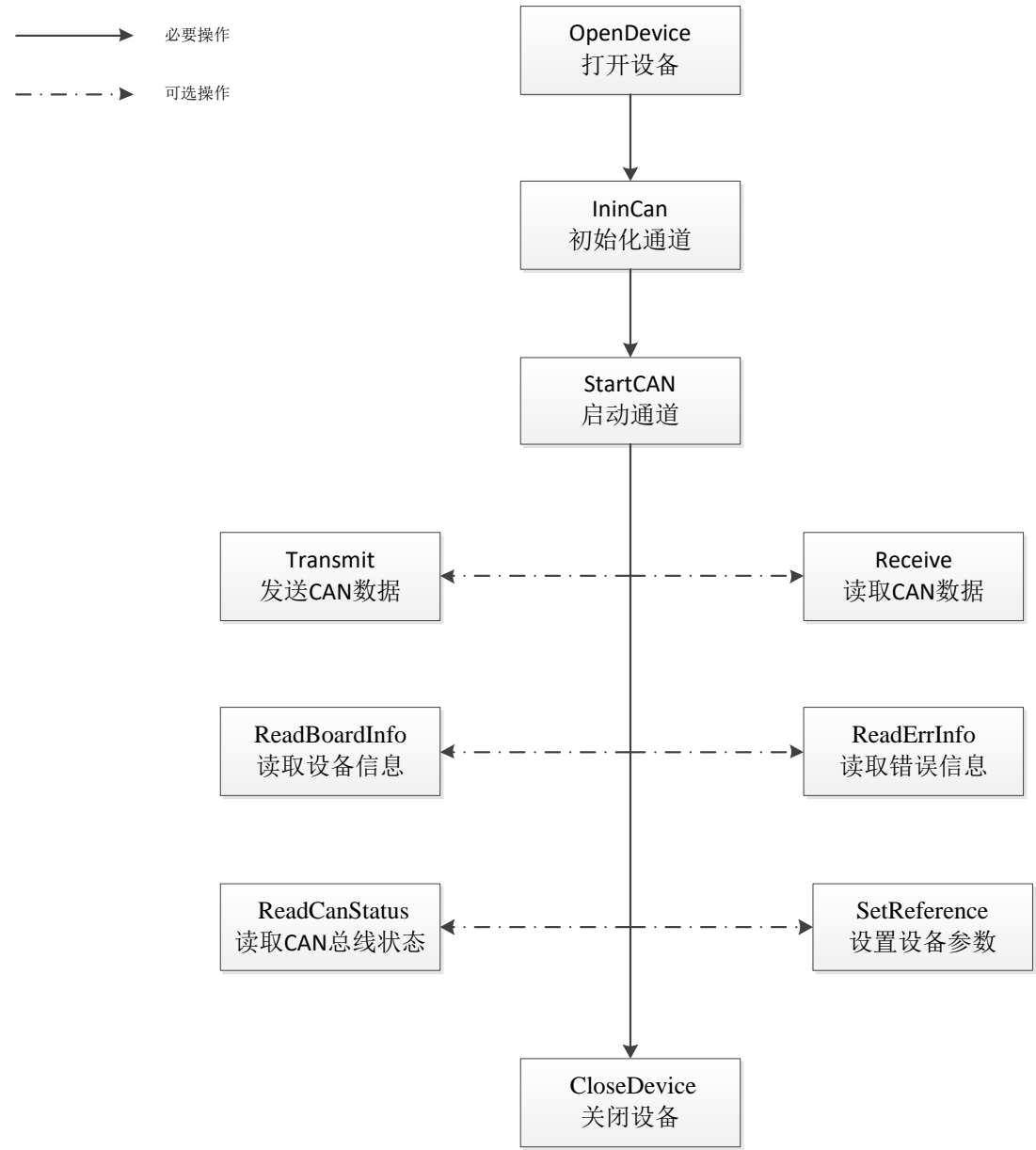


图 1.1 CAN 开发流程图

程序清单 1.1 开发代码示例

```

//接口卡类型定义
#define USBCAN1 3
#define USBCAN2 4
//函数调用返回状态值
#define STATUS_OK 1
#define STATUS_ERR 0
//通道号
#define CAN1 0
#define CAN2 1

bool m_connect=false; /*设备启动标致符 false: 表示设备未启动或者已经关闭 true: 表示设备已
经启动可以正常收发数据*/
DWORD DeviceType = USBCAN2; //设备类型 USBCAN2 双通道
DWORD DeviceInd = 0; //设备索引 0, 只有一个 USBCAN
if(OpenDevice(DeviceType, DeviceInd,0)!=STATUS_OK)
{
    Std::cout<<"打开设备失败"<<std::endl;
    return 0;
}
INIT_CONFIG init_config;
memset(&init_config,0,sizeof(init_config));
init_config.AccCode = 0;
init_config.AccMask =0xfffff; //不滤波
init_config.Filter = 0;
init_config.Timing0 = 0x01;
init_config.Timing1 = 0x1c; //波特率 250k
init_config.Mode = 0; //正常模式
if(InitCAN(DeviceType, DeviceInd, CAN1,&init_config)!=STATUS_OK)
{
    Std::cout<<"初始化通道失败"<<std::endl;
    CloseDevice(devtype DeviceInd);
    return;
}
if(StartCAN(DeviceType, DeviceInd, CAN1) !=STATUS_OK)
{
    Std::cout<<"打开通道失败"<<std::endl;
    CloseDevice(devtype DeviceInd);
    return;
}
if(InitCAN(DeviceType, DeviceInd, CAN2,&init_config)!=STATUS_OK)
{
    Std::cout<<"初始化通道失败"<<std::endl;
    CloseDevice(devtype DeviceInd);

```

```
        return;
    }
    if(StartCAN(DeviceType, DeviceInd, CAN2) !=STATUS_OK)
    {
        Std::cout<<"打开通道失败"<<std::endl;
        CloseDevice(devtype DeviceInd);
        return;
    }
    m_connect=true;
    CAN_OBJ frame;
    Memset(&frame,0,sizeof(frame));
    frame.ID=0x100;
    frame.DataLen=8;
    frame.SendType=0;
    frame.RemoteFlag=0;
    frame.ExternFlag=0
    Byte data[]={1,2,3,4,5,6,7,8};
    Memcpy(frame.data,data, frame.DataLen);
    If (Transmit(DeviceType, DeviceInd, CAN1,&frame,1) !=STATUS_OK)
    {
        Std::cout<<"发送数据失败"<<std::endl;
    }
    CloseDevice(devtype DeviceInd);
```

1.3 错误码定义

表 1.2 错误码

名称	值	描述
ERR_CAN_OVERFLOW	0x00000001	CAN控制器内部FIFO溢出
ERR_CAN_ERRALARM	0x00000002	CAN控制器错误报警
ERR_CAN_PASSIVE	0x00000004	CAN控制器消极错误
ERR_CAN_LOSE	0x00000008	CAN控制器仲裁丢失
ERR_CAN_BUSERR	0x00000010	CAN控制器总线错误
ERR_CAN_REG_FULL	0x00000020	CAN接收寄存器满
ERR_CAN_REC_OVER	0x00000040	CAN接收寄存器溢出
ERR_CAN_ACTIVE	0x00000080	CAN控制器主动错误
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_LOADKERNELDLL	0x00002000	装载动态库失败
ERR_CMDFAILED	0x00004000	执行命令失败错误码
ERR_BUFFERCREATE	0x00008000	内存不足

1.4 接口库函数结构体

1.4.1 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ReadBoardInfo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。

1.4.2 CAN_OBJ

描述

CAN_OBJ结构体表示帧的数据结构。在发送函数Transmit和接收函数Receive中被用来传送CAN信息帧。

```
typedef struct _CAN_OBJ {  
    UINT ID;  
    UINT TimeStamp;  
    BYTE TimeFlag;  
    BYTE SendType;  
    BYTE RemoteFlag;  
    BYTE ExternFlag;  
    BYTE DataLen;  
    BYTE Data[8];  
    BYTE Reserved[3];  
} CAN_OBJ, *P_CAN_OBJ;
```

成员

ID

报文帧ID。

TimeStamp

接收到信息帧时的时间标识，从CAN控制器初始化开始计时，单位微秒。

TimeFlag

是否使用时间标识，为1时TimeStamp有效，TimeFlag和TimeStamp只在此帧为接收帧时有意义。

SendType

发送帧类型。=0时为正常发送，=1时为单次发送（不自动重发），=2时为自发自收（用于测试CAN卡是否损坏），=3时为单次自发自收（只发送一次，用于自测试），只在此帧为发送帧时有意义。

RemoteFlag

是否是远程帧。=0时为数据帧，=1时为远程帧。

ExternFlag

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）。

DataLen

数据长度DLC(<=8)，即Data的长度。

Data

CAN报文的数据。空间受DataLen的约束。

Reserved

系统保留。

1.4.3 CAN_STATUS

描述

CAN_STATUS结构体包含CAN控制器状态信息。结构体将在ReadCanStatus函数中被填充。

```
typedef struct _CAN_STATUS {  
    UCHAR ErrInterrupt;  
    UCHAR regMode;  
    UCHAR regStatus;  
    UCHAR regALCapture;  
    UCHAR regECCapture;  
    UCHAR regEWLimit;  
    UCHAR regRECounter;  
    UCHAR regTECounter;  
    DWORD Reserved;  
} CAN_STATUS, *P_CAN_STATUS;
```

成员

ErrInterrupt

中断记录，读操作会清除。

regMode

CAN控制器模式寄存器。

regStatus

CAN控制器状态寄存器。

regALCapture

CAN控制器仲裁丢失寄存器。

regECCapture

CAN控制器错误寄存器。

regEWLimit

CAN控制器错误警告限制寄存器。

regRECounter

CAN控制器接收错误寄存器。

regTECounter

CAN控制器发送错误寄存器。

Reserved

系统保留。

1.4.4 ERR_INFO

描述

ERR_INFO 结构体用于装载 VCI 库运行时产生的错误信息。结构体将在 ReadErrInfo 函数中被填充。

```
typedef struct _ERR_INFO {  
    UINT ErrCode;  
    BYTE Passive_ErrData[3];  
    BYTE ArLost_ErrData;  
} ERR_INFO, *P_ERR_INFO;
```

成员

ErrCode

错误码。对应 1.2 中的错误码定义。

Passive_ErrData

当产生的错误中有消极错误时表示为消极错误的错误标识数据。

ArLost_ErrData

当产生的错误中有仲裁丢失错误时表示为仲裁丢失错误的错误标识数据。

1.4.5 INIT_CONFIG

描述

INIT_CONFIG结构体定义了初始化CAN的配置。结构体将在InitCan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 {  
    DWORD AccCode;  
    DWORD AccMask;  
    DWORD Reserved;  
    UCHAR Filter;  
    UCHAR Timing0;  
    UCHAR Timing1;  
    UCHAR Mode;  
} INIT_CONFIG, *P_INIT_CONFIG;
```

成员

AccCode

验收码。SJA1000的帧过滤验收码。

AccMask

屏蔽码。SJA1000的帧过滤屏蔽码。屏蔽码推荐设置为0xFFFF FFFF，即全部接收。

Reserved

保留。

Filter

滤波使能。0=不使能，1=使能。使能时，请参照SJA1000验收滤波器设置验收码和屏蔽码。

Timing0

波特率定时器0（BTR0）。设置值见下表。

Timing1

波特率定时器1（BTR1）。设置值见下表。

Mode

模式。=0为正常模式（相当于正常节点），=1为只听模式（只接受，不发送，不会响应CAN总线ACK应答），=2为自发自收模式。（自动将接收到的数据返回到CAN总线）

如果您需要进行滤波设置，您可以对AccCode和AccMask进行填充。填充值需要使用CANTest软件进行计算。请注意，本结构体仅接受双滤波的计算值，单滤波无效。

备注

Timing0和Timing1用来设置CAN波特率，以下是几种常见的波特率。如果要使用表中没有列出的波特率，请联系广成科技技术支持。

CAN波特率	定时器0	定时器1
5Kbps	0xBF	0xFF
10Kbps	0x31	0x1C
20Kbps	0x18	0x1C
40Kbps	0x87	0xFF
50Kbps	0x09	0x1C
80Kbps	0x83	0xFF
100Kbps	0x04	0x1C
125Kbps	0x03	0x1C
200Kbps	0x81	0xFA
250Kbps	0x01	0x1C
400Kbps	0x80	0xFA
500Kbps	0x00	0x1C
666Kbps	0x80	0xB6
800Kbps	0x00	0x16
1000Kbps	0x00	0x14

1.4.6 FILTER_RECORD

描述

FILTER_RECORD 结构体定义了 CAN 滤波器的滤波范围。结构体将在 SetReference 函数中被填充。

```
typedef struct _FILTER_RECORD{  
    DWORD ExtFrame;  
    DWORD Start;  
    DWORD End;  
} FILTER_RECORD, *P_FILTER_RECORD;
```

成员

ExtFrame

过滤的帧类型标志，为1代表要过滤的为扩展帧，为0代表要过滤的为标准帧。

Start

滤波范围的起始帧ID。

End

滤波范围的结束帧ID。

1.5 接口库函数说明

1.5.1 OpenDevice

描述

此函数用以打开设备。一个设备只能被打开一次，如需再次将已经打开的设备关闭。

```
DWORD __stdcall OpenDevice(DWORD DevType, DWORD DevIndex, DWORD Reserved);
```

参数

DevType

设备类型号。USBCAN I 选择3，USBCAN II 选择4。

DevIndex

设备索引号，根据设备SN号码从小到大分配索引号。

Reserved

参数无意义。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ECanVci.h"
int nDeviceType = 3; /* USBCAN-I */
int nDeviceInd = 0; /* 设备索引号*/
int nReserved = 0;
DWORD dwRel;

dwRel = OpenDevice(nDeviceType, nDeviceInd, nReserved);

if (dwRel != 1)
{
    MessageBox(_T("打开设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

1.5.2 CloseDevice

描述

此函数用于关闭设备，关闭设备和打开设备一一对应。

```
DWORD __stdcall CloseDevice(DWORD DevType, DWORD DevIndex);
```

参数

DevType

设备类型号。

DevIndex

设备索引号。对应已将打开的设备。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ECanVci.h"  
int nDeviceType = 3; /* USBCAN-I */  
int nDeviceInd = 0; /* 设备索引号*/  
CloseDevice(nDeviceType, nDeviceInd);
```


1.5.3 InitCan

描述

此函数用以初始化指定的CAN通道。有多个CAN通道时，需要多次调用。用于设备波特率、硬件滤波、CAN工作模式。

DWORD __stdcall InitCan(DWORD DevType, DWORD DevIndex, DWORD CANIndex, P_INIT_CONFIG *pInitConfig);

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。即对应卡的CAN通道号，CAN0为0，CAN1为1。

pInitConfig

初始化参数结构，详见1.4.5。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ECANVci.h"
int nDeviceType = 3; // USBCAN-I
int nDeviceInd = 0; // 索引号为0
int nCANInd = 0;
int nReserved = 0;
DWORD dwRel;

INIT_CONFIG init_config;
init_config.AccCode = 0;
init_config.AccMask = 0xffffffff; // 不滤波
init_config.Filter = 0;
//500k
init_config.Timing0 = 0;
init_config.Timing1 = 0x1c; //500k
init_config.Mode = 0;

dwRel = InitCAN(nDeviceType, nDeviceInd, nCANInd, &init_config)
if (dwRel != STATUS_OK)
{
    MessageBox(_T("设备初始化失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
}
```

1.5.4 ReadBoardInfo

描述

此函数用以获取设备信息：设备SN、软件版本、硬件版本、驱动版本、通道数。

```
DWORD __stdcall ReadBoardInfo(DWORD DevType, DWORD DevIndex,  
P_BOARD_INFO *pInfo);
```

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。

pInfo

用来存储设备信息的BOARD_INFO结构指针，详见1.4.1

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ECanVci.h"  
int nDeviceType = 3; // USBCAN-I  
int nDeviceInd = 0;  
int nCANInd = 0;  
INIT_CONFIG vic;  
BOARD_INFO vbi;  
DWORD dwRel;  
String SN="";  
dwRel = ReadBoardInfo(nDeviceType, nDeviceInd, &vbi);  
if (dwRel != STATUS_OK)  
{  
    MessageBox(_T("读取设备信息失败!"), _T("警告"),  
    MB_OK|MB_ICONQUESTION);  
}  
Else  
{  
    For (int i=0;i<11;i++)  
    {  
        SN=SN+(char)vbi.str_Serial_Num[i]; //获取设备SN  
    }  
}
```

1.5.5 ReadErrInfo

描述

此函数用以获取USBCAN分析仪最后一次错误信息。

```
DWORD __stdcall ReadErrInfo(DWORD DevType, DWORD DevIndex, DWORD
CANIndex, P_ERR_INFO *pErrInfo);
```

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通道。

pErrInfo

用来存储错误信息的ERR_INFO结构指针，详见1.4.4。

pErrInfo->ErrCode可能为下列各个错误码的多种组合之一：

ErrCode	Passive_ErrData	ArLost_ErrData	错误描述
0x0100	无	无	设备已经打开
0x0200	无	无	打开设备错误
0x0400	无	无	设备没有打开
0x0800	无	无	缓冲区溢出
0x1000	无	无	此设备不存在
0x2000	无	无	装载动态库失败
0x4000	无	无	表示为执行命令失败错误
0x8000	无	无	内存不足
0x0001	无	无	CAN控制器内部FIFO溢出
0x0002	无	无	CAN控制器错误报警
0x0004	有，具体值见表后	无	CAN控制器消极错误
0x0008	无	有，具体值见表后	CAN控制器仲裁丢失
0x0010	无	无	CAN控制器总线错误
0x0020			CAN接收寄存器满
0x0040			CAN接收寄存器溢出
0x0080		有，具体值见表后	CAN控制器主动错误

返回值

为1表示操作成功，0表示操作失败。

备注

当(PErrInfo->ErrCode&0x0004)==0x0004时，存在CAN控制器消极错误。

PErrInfo->Passive_ErrData[0] 错误代码捕捉位功能表示

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
错误代码类型		错误属性	错误段表示				

错误代码类型功能说明

位 ECC.7	位 ECC.6	功能
0	0	位错
0	1	格式错
1	0	填充错
1	1	其它错误

错误属性

bit5 =0; 表示发送时发生的错误。

bit5 =1; 表示接收时发生的错误。

错误段表示功能说明

bit4	bit 3	bit 2	bit 1	bit 0	功能
0	0	0	1	1	帧开始
0	0	0	1	0	ID.28-ID.21
0	0	1	1	0	ID.20-ID.18
0	0	1	0	0	SRTR 位
0	0	1	0	1	IDE 位
0	0	1	1	1	ID.17-ID.13
0	1	1	1	1	ID.12-ID.5
0	1	1	1	0	ID.4-ID.0
0	1	1	0	0	RTR 位
0	1	1	0	1	保留位 1
0	1	0	0	1	保留位 0
0	1	0	1	1	数据长度代码
0	1	0	1	0	数据区
0	1	0	0	0	CRC 序列
1	1	0	0	0	CRC 定义符
1	1	0	0	1	应答通道
1	1	0	1	1	应答定义符
1	1	0	1	0	帧结束
1	0	0	1	0	中止
1	0	0	0	1	活动错误标志
1	0	1	1	0	消极错误标志
1	0	0	1	1	支配（控制）位误差
1	0	1	1	1	错误定义符
1	1	1	0	0	溢出标志

PErrInfo->Passive_ErrData[1] 表示接收错误计数器

PErrInfo->Passive_ErrData[2] 表示发送错误计数器

当 (PErrInfo->ErrCode&0x0008)==0x0008时, 表示存在CAN控制器仲裁丢失错误。

PErrInfo->ArLost_ErrData 仲裁丢失代码捕捉位功能表示

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
_____	_____	_____	错误段表示				

位					十进制值	位功能
ALC. 4	ALC. 3	ALC. 2	ALC. 1	ALC. 0		
0	0	0	0	0	0	仲裁丢失在识别码的bit1
0	0	0	0	1	1	仲裁丢失在识别码的bit2
0	0	0	1	0	2	仲裁丢失在识别码的bit3
0	0	0	1	1	3	仲裁丢失在识别码的bit4
0	0	1	0	0	4	仲裁丢失在识别码的bit5
0	0	1	0	1	5	仲裁丢失在识别码的bit6
0	0	1	1	0	6	仲裁丢失在识别码的bit7
0	0	1	1	1	7	仲裁丢失在识别码的bit8
0	1	0	0	0	8	仲裁丢失在识别码的bit9
0	1	0	0	1	9	仲裁丢失在识别码的bit10
0	1	0	1	0	10	仲裁丢失在识别码的bit11
0	1	0	1	1	11	仲裁丢失在SRTR位
0	1	1	0	0	12	仲裁丢失在IDE位
0	1	1	0	1	13	仲裁丢失在识别码的bit12
0	1	1	1	0	14	仲裁丢失在识别码的bit13
0	1	1	1	1	15	仲裁丢失在识别码的bit14
1	0	0	0	0	16	仲裁丢失在识别码的bit15
1	0	0	0	1	17	仲裁丢失在识别码的bit16
1	0	0	1	0	18	仲裁丢失在识别码的bit17
1	0	0	1	1	19	仲裁丢失在识别码的bit18
1	0	1	0	0	20	仲裁丢失在识别码的bit19
1	0	1	0	1	21	仲裁丢失在识别码的bit20
1	0	1	1	0	22	仲裁丢失在识别码的bit21
1	0	1	1	1	23	仲裁丢失在识别码的bit22
1	1	0	0	0	24	仲裁丢失在识别码的bit23
1	1	0	0	1	25	仲裁丢失在识别码的bit24
1	1	0	1	0	26	仲裁丢失在识别码的bit25
1	1	0	1	1	27	仲裁丢失在识别码的bit26
1	1	1	0	0	28	仲裁丢失在识别码的bit27
1	1	1	0	1	29	仲裁丢失在识别码的bit28
1	1	1	1	0	30	仲裁丢失在识别码的bit29
1	1	1	1	1	31	仲裁丢失在ERTR位

示例

```
#include "ECanVci.h"
int nDeviceType =3; // USBCAN-I
int nDeviceInd = 0;
int nCANInd = 0;
ERR_INFO vei;
DWORD dwRel;
dwRel = ReadErrInfo(nDeviceType, nDeviceInd, nCANInd, &vei);
```

1.5.6 ReadCanStatus

描述

此函数用以获取CAN状态。

```
DWORD __stdcall ReadCanStatus(DWORD DevType, DWORD DevIndex, DWORD  
CANIndex, P_CAN_STATUS *pCANStatus);
```

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。

pCANStatus

用来存储CAN状态的CAN_STATUS结构指针，详见1.4.3。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ECanVci.h"  
int nDeviceType = 4; // USBCAN-II  
int nDeviceInd = 0;  
int nCANInd = 0;  
INIT_CONFIG vic;  
CAN_STATUS vcs;  
DWORD dwRel;  
dwRel = ReadCanStatus(nDeviceType, nDeviceInd, nCANInd, &vcs);
```

1.5.7 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 "ECanVci.h"  
int nDeviceType = 4; // USBCAN-II  
int nDeviceInd = 0;  
int nCANInd = 0;  
BYTE info[14];  
DWORD dwRel;  
info[0] = 1;  
dwRel = GetReference(nDeviceType, nDeviceInd, nCANInd, 1, (PVOID)info);
```


1.5.8 SetReference

描述

此函数用以设置设备的相应参数，主要处理不同设备的特定操作。（可用于设备滤波）

```
DWORD __stdcall SetReference(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

///滤波演示

```
#include "ECANVci.h"  
int nDeviceType = 3; // USBCAN-I  
int nDeviceInd = 0;  
int nCANInd = 0;  
INIT_CONFIG init_config;  
init_config.AccCode = 0;  
init_config.AccMask = 0xffffffff;  
init_config.Filter = 0;  
init_config.Timing0 = 0;///1000k  
init_config.Timing1 = 0x14;  
init_config.Mode = 0;  
if (OpenDevice(nDeviceType, nDeviceInd, 0) != STATUS_OK)  
{  
    MessageBox(_T("打开设备失败!"), _T("警告"), MB_OK | MB_ICONQUESTION);  
    return;  
}  
ShowInfo("Open device Success", 0);  
if (InitCAN(nDeviceType, nDeviceInd, nCANInd, &init_config) != STATUS_OK)  
{
```

```
MessageBox(_T("设备初始化失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
return;
}
DWORD dwRel;
FILTER_RECORD filter_record;
filter_record.ExtFrame = 1;///只接收标准扩展帧
filter_record.Start = 0x000;///起始ID为0x000
filter_record.End = 0x001;///结束ID为0x001
///只接收ID是0x000和0x001的扩展帧
SetReference(nDeviceType, nDeviceInd, nCANInd, 1, &filter_record);
if(StartCAN(nDeviceType, nDeviceInd, nCANInd)==1)
{
    ShowInfo("Start Success",0);
}
```

1.5.9 GetReceiveNum

描述

此函数用以获取指定接收缓冲区中接收到但尚未被读取的帧数量。

```
ULONG __stdcall GetReceiveNum(DWORD DevType, DWORD DevIndex, DWORD  
CANIndex);
```

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。

返回值

返回尚未被读取的帧数。

示例

```
#include "ECanVci.h"  
int nDeviceType =3; // USBCAN-I  
int nDeviceInd = 0;  
int nCANInd = 0;  
DWORD dwRel;  
dwRel = GetReceiveNum(nDeviceType, nDeviceInd, nCANInd);
```

1.5.10 ClearBuffer

描述

此函数用以清空指定CAN通道的缓冲区。

```
DWORD __stdcall ClearBuffer(DWORD DevType,  DWORD DevIndex,  DWORD  
CANIndex);
```

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。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ECANVci.h"  
int nDeviceType = 3; // USBCAN-I  
int nDeviceInd = 0;  
int nCANInd = 0;  
DWORD dwRel;  
dwRel = ClearBuffer(nDeviceType, nDeviceInd, nCANInd);
```

1.5.11 StartCAN

描述

此函数用以启动USBCAN设备的某一个CAN通道。如有多个CAN通道时，需要多次调用。在执行StartCAN函数后，需要延迟10ms执行Transmit函数。

DWORD __stdcall StartCAN(DWORD DevType, DWORD DevIndex, DWORD CANIndex);

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。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ECanVci.h"
int nDeviceType =3; // USBCAN-I
int nDeviceInd = 0;
int nCANInd = 0;
int nReserved = 0;
INIT_CONFIG vic;
DWORD dwRel;
dwRel = OpenDevice(nDeviceType, nDeviceInd, nReserved);
if (dwRel != STATUS_OK)
{
    MessageBox(_T("打开设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
dwRel = InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic);
if (dwRel == STATUS_ERR)
{
    CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("初始化设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
dwRel = StartCAN(nDeviceType, nDeviceInd, nCANInd);
if (dwRel == STATUS_ERR)
{
    CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("启动设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

1.5.12 Transmit

描述

返回实际发送成功的帧数量。

```
ULONG __stdcall Transmit(DWORD DevType, DWORD DevIndex, DWORD CANIndex,  
P_CAN_OBJ *pSend, ULONG Len);
```

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。

pSend

结构体CAN_OBJ数组的首指针，详见1.4.2。

Len

报文数目。

返回值

返回实际发送的帧数。

示例

```
#include "ECANVci.h"  
#include <string.h>  
int nDeviceType = 3;           // USBCAN-I  
int nDeviceInd = 0;            // 索引1  
int nCANInd = 0;               // CAN1通道  
DWORD dwRel;  
CAN_OBJ vco;                  //需要发送数据的结构体  
ZeroMemory(&vco, sizeof (CAN_OBJ));  
vco.ID = 0x00000000;           //ID=0x000  
vco.SendType = 0;              //正常发送  
vco.RemoteFlag = 0;           //数据帧  
vco.ExternFlag = 0;           //标准帧  
vco.DataLen = 8;               //数据长度为8  
for (int i=0;i<7;i++)  
{  
    Vco.Data[i]=i;             //数据  
}  
dwRel = Transmit(nDeviceType, nDeviceInd, nCANInd, &vco, 1);  
//单次执行发送一条
```

1.5.13 Receive

描述

此函数从指定的设备CAN通道的缓冲区里读取数据。

```
ULONG __stdcall Receive(DWORD DevType, DWORD DevIndex, DWORD CANIndex,  
P_CAN_OBJ *pReceive, ULONG Len, INT WaitTime=-1);
```

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。

pReceive

结构体CAN_OBJ数组的首指针，详见1.4.2。

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）。。

WaitTime

缓存区无数据，函数阻塞等待时间等，以毫秒为单位，推荐设置成0，即不等待。

返回值

返回实际读取到的帧数。如果返回值为0xFFFFFFFF，则表示读取数据失败，有错误发生，请调用ReadErrInfo函数来获取错误码。

示例

```
#include "ECanVci.h"  
#include <string.h>  
int nDeviceType = 3; // USBCAN-I  
int nDeviceInd = 0;  
int nCANInd = 0;  
DWORD dwRel;  
CAN_OBJ vco[100];  
dwRel = Receive(nDeviceType, nDeviceInd, nCANInd, vco, 100, 0); // 单次  
执行函数读取最多100条数据
```

1.5.14 ResetCAN

描述

此函数用以复位CAN。如当USBCAN分析仪进入总线关闭状态时，可以调用这个函数。

```
DWORD __stdcall ResetCAN(DWORD DevType, DWORD DevIndex, DWORD CANIndex);
```

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。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ECanVci.h"
int nDeviceType = 4; // USBCAN-II
int nDeviceInd = 0;
int nCANInd = 0;
DWORD dwRel;
dwRel = ResetCAN(nDeviceType, nDeviceInd, nCANInd);
```


1.6 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。库函数文件总共有三个文件：ECanVci.h、ECanVci.lib、ECanVci.dll。

1.6.1 VC 调用动态库的方法

(1) 在扩展名为.CPP 的文件中包含 ECANVCI.h 头文件。

如：#include "ECANVci.h"

(2) 在工程的连接器设置中连接到 ECANVCI.lib 文件。

如：在 VC7 环境下，在项目属性页里的配置属性→连接器→输入→附加依赖项中添加 ECANVCI.lib

在 VC6 环境下，在 CPP 文件中加入静态库调用：

#pragma comment(lib,"ECANVCI")

1.6.2 VB 调用动态库的方法

通过以下方法进行声明后就可以调用了。

语法：

```
[Public | Private] Declare Function name Lib "libname" [Alias "aliasname"]  
[([arglist])] [As type]
```

Declare 语句的语法包含下面部分：

Public（可选）

用于声明在所有模块中的所有过程都可以使用的函数。

Private（可选）

用于声明只能在包含该声明的模块中使用的函数。

Name（必选）

任何合法的函数名。动态链接库的入口处（entry points）区分大小写。

Libname（必选）

包含所声明的函数动态链接库名或代码资源名。

Alias（可选）

表示将被调用的函数在动态链接库（DLL）中还有另外的名称。当外部函数名与某个函数重名时，就可以使用这个参数。当动态链接库的函数与同一范围内的公用变量、常数或任何其它过程的名称相同时，也可以使用 Alias。如果该动态链接库函数中的某个字符不符合动态链接库的命名约定时，也可以使用 Alias。

Aliasname（可选）

动态链接库。如果首字符不是数字符号（#），则 aliasname 是动态链接库中该函数入口处的名称。如果首字符是（#），则随后的字符必须指定该函数入口处的顺序号。

Arglist（可选）

代表调用该函数时需要传递参数的变量表。

Type（可选）

Function返回值的数据类型；可以是 Byte、Boolean、Integer、Long、Currency、

Single、Double、Decimal（目前尚不支持）、Date、String（只支持变长）或 Variant，用户定义类型，或对象类型。

arglist 参数的语法如下：

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type]
```

部分描述：

Optional（可选）

表示参数不是必需的。如果使用该选项，则 arglist 中的后续参数都必需是可选的，而且必须都使用 Optional 关键字声明。如果使用了 ParamArray，则任何参数都不能使用 Optional。

ByVal（可选）

表示该参数按值传递。

ByRef（可选）

表示该参数按地址传递。

例如：

```
Public Declare Function OpenDevice Lib "ECANVCI" (ByVal devicetype As Long, ByVal deviceind As Long, ByVal reserved As Long) As Long
```

1.7 接口库函数

```
OpenDevice(DWORD DeviceType,DWORD DeviceInd,DWORD Reserved);
CloseDevice(DWORD DeviceType,DWORD DeviceInd);
InitCAN(DWORD DeviceType, DWORD DeviceInd, DWORD CANInd, P_INIT_CONFIG *pInitConfig);
ReadBoardInfo(DWORD DeviceType,DWORD DeviceInd,P_BOARD_INFO* pInfo);
ReadErrInfo(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,P_ERR_INFO* pErrInfo);
ReadCANStatus(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,P_CAN_STATUS *pCANStatus);
GetReference(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,DWORD RefType,PVOID *pData);
SetReference(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,DWORD RefType,PVOID* pData);
GetReceiveNum(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd);
ClearBuffer(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd);
StartCAN(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd);
ResetCAN(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd);
Transmit(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,P_CAN_OBJ *pSend,ULONG Len);
Receive(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,P_CAN_OBJ *pReceive,ULONG Len,INT WaitTime);
```

1.8 常用功能样例

1.8.1 滤波设置

方法一（使用 SetReference）

```
#include "ECanVci.h"
int nDeviceType = 3; // USBCAN-I
int nDeviceInd = 0;
int nCANInd = 0;
INIT_CONFIG init_config;
init_config.AccCode = 0;
init_config.AccMask = 0xffffffff;
init_config.Filter = 0;
init_config.Timing0 = 0;
init_config.Timing1 = 0x14; //1000k
init_config.Mode = 0;
if(OpenDevice(nDeviceType, nDeviceInd, 0) != STATUS_OK)
{
    MessageBox(_T("打开设备失败!"), _T("警告"),
    MB_OK|MB_ICONQUESTION);
    return;
}
ShowInfo("Open device Success", 0);
if(InitCAN(nDeviceType, nDeviceInd, nCANInd, &init_config) != STATUS_OK)
{
    MessageBox(_T("设备初始化失败!"), _T("警告"),
    MB_OK|MB_ICONQUESTION);
    return;
}
DWORD dwRel;
FILTER_RECORD filter_record;
filter_record.ExtFrame = 1; ///只接收扩展帧
filter_record.Start = 0x000; ///起始ID为0x000
filter_record.End = 0x001; ///结束ID为0x001
///只接收ID是0x000和0x001的扩展帧
SetReference(nDeviceType, nDeviceInd, nCANInd, 1, &filter_record);
if(StartCAN(nDeviceType, nDeviceInd, nCANInd) == 1)
{
    ShowInfo("Start Success", 0);
}
方法二（使用屏蔽码和验收码）
#include "ECanVci.h"
int nDeviceType = 3; // USBCAN-I
int nDeviceInd = 0;
int nCANInd = 0;
```

```

INIT_CONFIG init_config;
init_config.AccCode = 0x00200000;
init_config.AccMask = 0x001FFFFF; //只接受ID是0x001的标准帧数据
init_config.Filter = 0;
init_config.Timing0 = 0;
init_config.Timing1 = 0x14; //1000k
init_config.Mode = 0;
if (OpenDevice(nDeviceType, nDeviceInd, 0) != STATUS_OK)
{
    MessageBox(_T("打开设备失败!"), _T("警告"),
    MB_OK|MB_ICONQUESTION);
    return;
}
ShowInfo("Open device Success", 0);
if (InitCAN(nDeviceType, nDeviceInd, nCANInd, &init_config) != STATUS_OK)
{
    MessageBox(_T("设备初始化失败!"), _T("警告"),
    MB_OK|MB_ICONQUESTION);
    return;
}
SetReference(nDeviceType, nDeviceInd, nCANInd, 1, &filter_record);
if (StartCAN(nDeviceType, nDeviceInd, nCANInd) == 1)
{
    ShowInfo("Start Success", 0);
}

```

1.8.2 读取设备 SN

```

#include "ECANvci.h"
int nDeviceType = 3; // USBCAN-I
int nDeviceInd = 0;
int nCANInd = 0;
INIT_CONFIG vic;
BOARD_INFO vbi;
DWORD dwRel;
String SN="";
dwRel = ReadBoardInfo(nDeviceType, nDeviceInd, &vbi);
if (dwRel != STATUS_OK)
{
    MessageBox(_T("读取设备信息失败!"), _T("警告"),
    MB_OK|MB_ICONQUESTION);
}
Else
{
    For (int i=0; i<11; i++)
    {

```

```

        SN+=(char)vbi.str_Serial_Num[i]; //获取设备SN
    }
    Std::cout<<SN <<std::endl;

}

```

1.8.3 批量读取数据

```

#include "ECanVci.h"
#include <string.h>
int nDeviceType = 3; // USBCAN-I
int nDeviceInd = 0;
int nCANInd = 0;
int len =100
DWORD dwRel;
CAN_OBJ frameinfo [100];
dwRel = Receive(nDeviceType, nDeviceInd, nCANInd, frameinfo, len, 0);
//单次执行函数读取最多100条数据
If (dwRel>0)
{
    For (int i=0;i<len;)
    {
        String str=" Rec:";
        if(frameinfo[i].TimeFlag==0)
            tmpstr="Time: ";
        else
            tmpstr.Format(_T("Time:%08x"),frameinfo[i].TimeStamp);
        str+=tmpstr;
        tmpstr.Format(_T("ID:%08x "),frameinfo[i].ID);
        str+=tmpstr;
        str+="Format:";
        if(frameinfo[i].RemoteFlag==0)
            tmpstr="Data ";
        else
            tmpstr="Romte ";
        str+=tmpstr;
        str+="Type:";
        if(frameinfo[i].ExternFlag==0)
            tmpstr="Stand ";
        else
            tmpstr="Exten ";
        str+=tmpstr;
        if(frameinfo[i].RemoteFlag==0)
        {
            Str+="Data:";
            if(frameinfo[i].DataLen>8)

```

```
        frameinfo[i].DataLen=8;
        for(int j=0;j<frameinfo[i].DataLen;j++)
        {
            tmpstr.Format(_T("%02x "), frameinfo[i].Data[j]);
            str+=tmpstr;
        }
    }
    Std::cout<<str<<std::endl;
}
}
```

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

