

# CAN-bus 通用测试软件及接口函数库使用手册

## (LINUX)

### 一 . 驱动程序的安装

所有驱动都在 Linux 2.4.20-8 下测试通过。

#### 1.1 USBCAN 驱动的安装

把 driver 目录下的 usbcan.o 文件拷贝到 /lib/modules/(\*)/kernel/drivers/usb 目录下,就完成了驱动的安装(其中(\*)根据 Linux 版本的不同而不同,比如 Linux 版本为 2.4.20-8,则此目录的名称也为“2.4.20-8”,即跟 Linux 内核版本号相同)。

#### 1.2 PCI5121 驱动的安装

把 driver 目录下的 pci51xx.o 文件拷贝到 /lib/modules/(\*)/kernel/drivers/char 目录下,就完成了驱动的安装(其中(\*)根据 Linux 版本的不同而不同,比如 Linux 版本为 2.4.20-8,则此目录的名称也为“2.4.20-8”,即跟 Linux 内核版本号相同)。

#### 1.3 CAN232

CAN232 由于是串口设备,系统已经自带了串口的驱动程序,所以就无需安装了。**不过要注意的是,由于受到串口速度的限制,CAN232 只能用于低速通讯的场合。**

#### 1.4 PCI9820 驱动的安装

把 driver 目录下的 pci9820b.o 文件拷贝到 /lib/modules/(\*)/kernel/drivers/char 目录下,就完成了驱动的安装(其中(\*)根据 Linux 版本的不同而不同,比如 Linux 版本为 2.4.20-8,则此目录的名称也为“2.4.20-8”,即跟 Linux 内核版本号相同)。

### 二 . 动态库的安装

把 dll 文件夹中的 **libcontrolcan.so 文件和 kernelDlls 文件夹**一起拷贝到 /lib 目录,然后运行 ldconfig /lib 命令,就可以完成动态库的安装。

### 三 . 动态库的调用及编译

动态库的调用是非常简单的,只需要把 dll 文件夹中的 controlcan.h 文件拷贝到你的当前工程目录下,然后用#include “controlcan.h”把 controlcan.h 文件包含到你的源代码文件中,就可以使用动态库中的函数了。

在用 GCC 编译的时候只需要添加 -lcontrolcan 选项就可以了,比如:

```
gcc -lcontrolcan -g -o test test.c
```

### 四 . 动态库函数说明及其使用

#### 4.1 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
PCI5121	1
PCI9810	2
USBCAN1	3
USBCAN2	4
PCI9820	5
CAN232	6
PCI5110	7
CANlite(CANmini)	8
ISA9620	9
ISA5420	10

#### 4.2 错误码定义

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_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

#### 4.3 函数库中的数据结构定义

##### 4.3.1 存储 ZLGCAN 系列接口卡信息的数据结构

定义：

```
typedef struct _VCI_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] ;
}
```

```
} VCI_BOARD_INFO, *PVC_BOARD_INFO;
```

**参数：**

hw_Version	用 16 进制表示的硬件版本号，比如 0x0100 表示 V1.00；
fw_Version	用 16 进制表示的固件版本号；
dr_Version	用 16 进制表示的驱动程序版本号；
in_Version	用 16 进制表示的接口库版本号；
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，比如“USBCAN V1.00”(注意：包括字符串结束符‘\0’)；
Reserved	系统保留。

## 4.3.2 定义 CAN 信息帧的数据结构

**定义：**

```
typedef struct _VCI_CAN_OBJ{
    UINT    ID;
    UINT    TimeStamp;
    BYTE    TimeFlag;
    BYTE    SendType;
    BYTE    RemoteFlag;
    BYTE    ExternFlag;
    BYTE    DataLen;
    BYTE    Data[8];
    BYTE    Reserved[3];
}VCI_CAN_OBJ, *PVC_CAN_OBJ;
```

**参数：**

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；
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；
SendType	发送帧类型，=0时为正常发送，=1时为单次发送，=2时为自发自收，=3时为单次自发自收，只在此帧为发送帧时有意义。
RemoteFlag	是否是远程帧；
ExternFlag	是否是扩展帧；
ID	报文 ID；
DataLen	数据长度(<=8)，即 Data 的长度；
Data	报文的数据；
Reserved	系统保留。

## 4.3.3 存储 CAN 控制器状态的数据结构

**定义：**

```
typedef struct _VCI_CAN_STATUS{
    UCHAR    ErrInterrupt;
    UCHAR    regMode;
    UCHAR    regStatus;
```

```

        UCHAR    regALCapture ;
        UCHAR    regECCapture ;
        UCHAR    regEWLimit ;
        UCHAR    regRECounter ;
        UCHAR    regTECounter ;
        DWORD    Reserved ;
}VCI_CAN_STATUS, *PVCI_CAN_STATUS ;

```

**参数：**

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。

## 4.3.4 存储错误信息的数据结构

**定义：**

```

typedef struct _ERR_INFO{
        UINT      ErrCode ;
        BYTE      Passive_ErrData[3] ;
        BYTE      ArLost_ErrData ;
} VCI_ERR_INFO , *PVCI_ERR_INFO ;

```

**参数：**

ErrCode 错误码；

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。

## 4.3.5 定义初始化 CAN 的数据结构

**定义：**

```

typedef struct _INIT_CONFIG{
        DWORD     AccCode ;
        DWORD     AccMask ;
        DWORD     Reserved ;
        UCHAR     Filter ;
        UCHAR     Timing0 ;
        UCHAR     Timing1 ;
        UCHAR     Mode ;
}VCI_INIT_CONFIG, *PVCI_INIT_CONFIG ;

```

**参数：**

AccCode 验收码；

AccMask	屏蔽码；
Reserved	保留；
Filter	滤波方式；
Timing0	定时器 0 ( BTR0 )；
Timing1	定时器 1 ( BTR1 )；
Mode	模式。

注：Timing0 和 Timing1 用来设置 CAN 波特率，几种常见的波特率设置如下（SJA1000 晶振频率 = 16MHz）：

CAN 波特率	定时器 0	定时器 1
5KBPS	0xBF	0xFF
10KBPS	0xFF	0xFF
20KBPS	0x53	0x2F
40KBPS	0x87	0xFF
50KBPS	0x47	0x2F
80KBPS	0x83	0xFF
100KBPS	0x43	0x2F
125KBPS	0x03	0x1C
200KBPS	0x81	0xFA
250KBPS	0x01	0x1C
400KBPS	0x80	0xFA
500KBPS	0x00	0x1C
666KBPS	0x80	0xB6
800KBPS	0x00	0x16
1000KBPS	0x00	0x14

#### 4.4 接口库函数说明

##### 4.4.1 **DWORD \_\_stdcall VCI\_OpenDevice(DWORD DevType,DWORD DevIndex, DWORD Reserved);**

入口参数：

DevType：

设备类型号。

DevIndex：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

Reserved：

当设备为 CAN232 时，此参数表示为用以打开串口的波特率，可以为 2400，4800，9600，14400，19200，28800，57600。当为其他设备时此参数无意义。

函数功能：

此函数用以打开设备。

返回值：

为 1 表示操作成功，0 表示操作失败。

##### 4.4.2 **DWORD \_\_stdcall VCI\_CloseDevice(DWORD DevType,DWORD DevIndex);**

入口参数：

DevType：

设备类型号。

DevIndex :

设备索引号, 比如当只有一个 PCI5121 时, 索引号为 0, 有两个时可以为 0 或 1。 (注: 当为 CAN232 时, 0 表示要打开的是 COM1, 1 表示要打开的是 COM2。)

函数功能:

此函数用以关闭设备。

返回值:

为 1 表示操作成功, 0 表示操作失败。

#### 4.4.3 **DWORD \_\_stdcall VCI\_InitCan(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI\_INIT\_CONFIG pInitConfig);**

入口参数:

DevType :

设备类型号。

DevIndex :

设备索引号, 比如当只有一个 PCI5121 时, 索引号为 0, 有两个时可以为 0 或 1。 (注: 当为 CAN232 时, 0 表示要打开的是 COM1, 1 表示要打开的是 COM2。)

CANIndex :

第几路 CAN。

pInitConfig :

初始化参数结构 (注: 当为 CAN232 时, 忽略此参数, 其值设为 NULL)。

函数功能:

此函数用以初始化指定的 CAN。

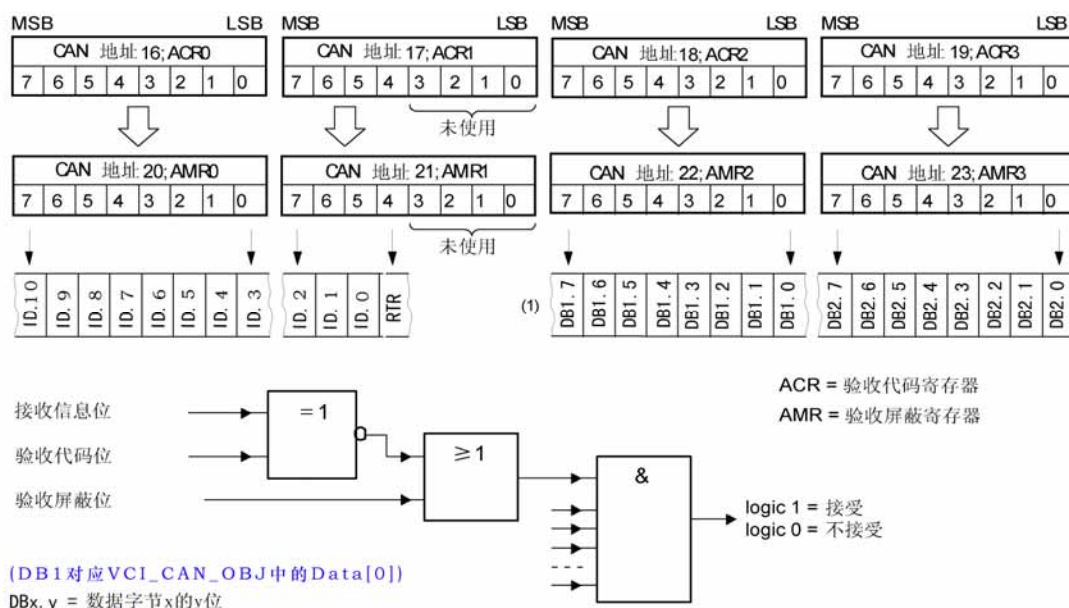
返回值:

为 1 表示操作成功, 0 表示操作失败。

参数表:

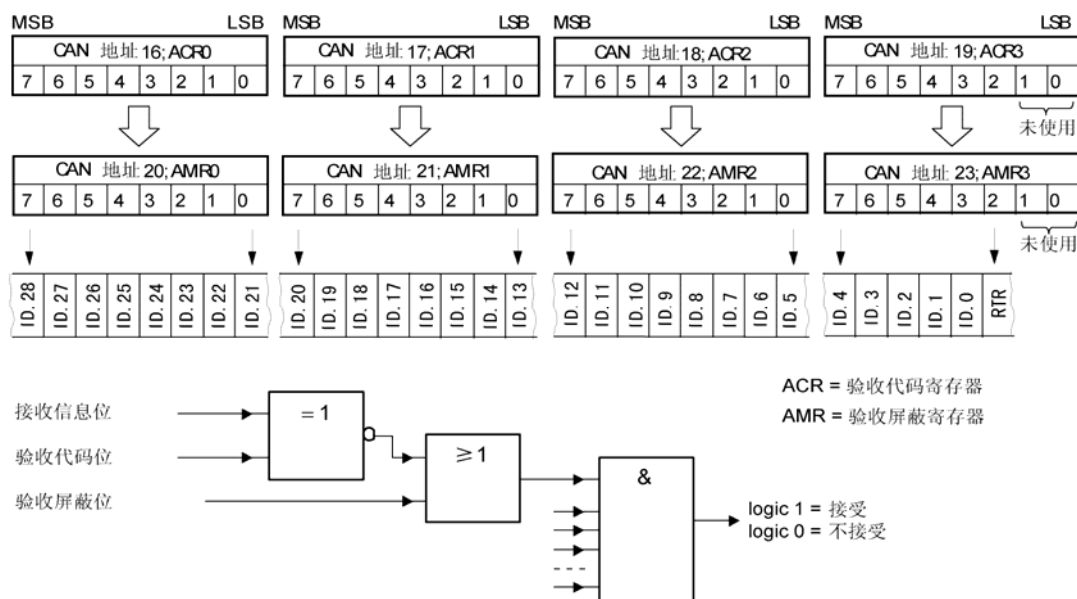
pInitConfig	功能描述
pInitConfig->AccCode	AccCode 对应 SJA1000 中的四个寄存器 ACR0, ACR1, ACR2, ACR3, 其中高字节对应 ACR0, 低字节对应 ACR3; AccMask 对应 SJA1000 中的四个寄存器 AMR0, AMR1, AMR2, AMR3, 其中高字节对应 AMR0, 低字节对应 AMR3。 (请看表后说明)
pInitConfig->AccMask	
pInitConfig->Reserved	保留
pInitConfig->Filter	滤波方式, 1 表示单滤波, 0 表示双滤波
pInitConfig->Timing0	定时器 0
pInitConfig->Timing1	定时器 1
pInitConfig->Mode	模式, 0 表示正常模式, 1 表示只听模式

当滤波方式为单滤波, 接收帧为标准帧时:

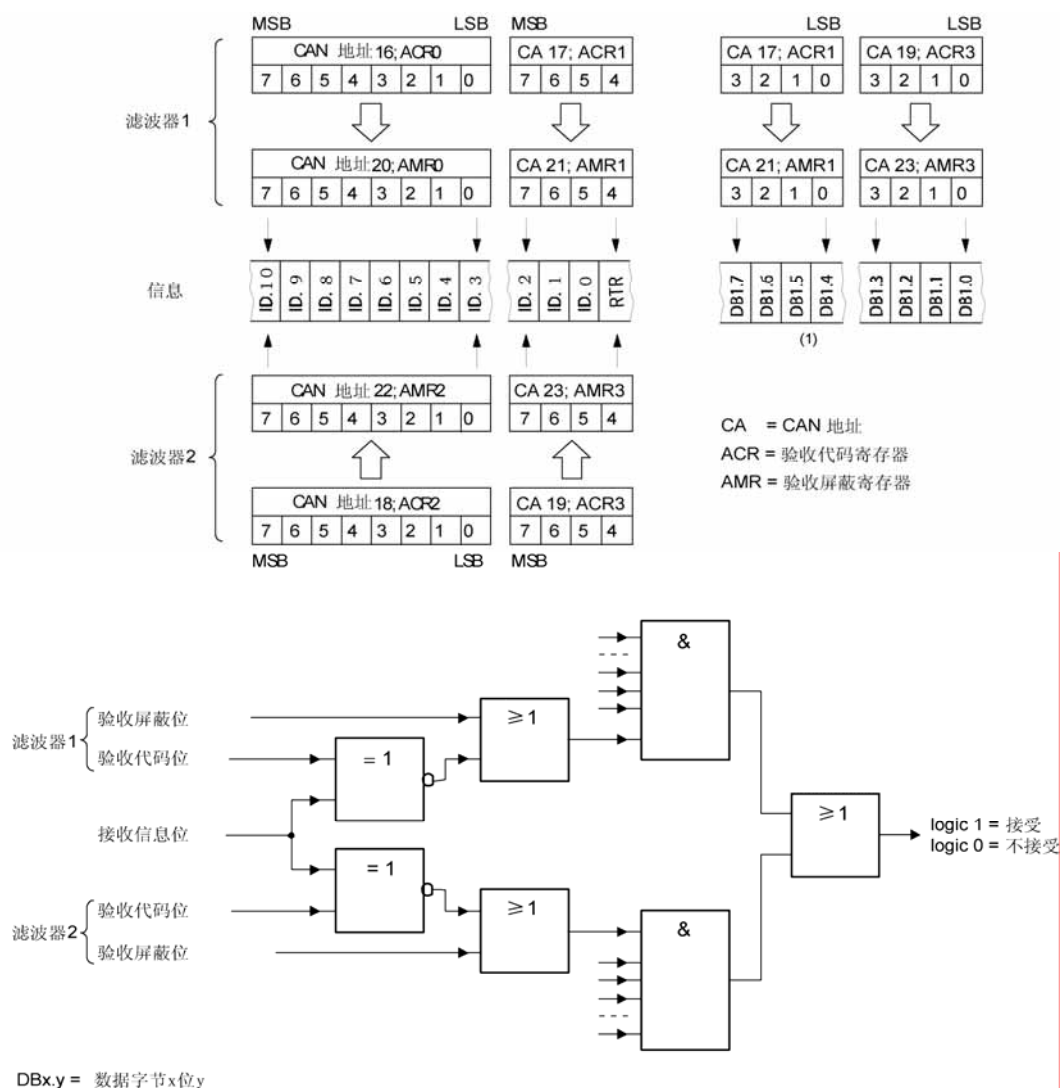


RTR 对应 VCI\_CAN\_OBJ 中的 RemoteFlag

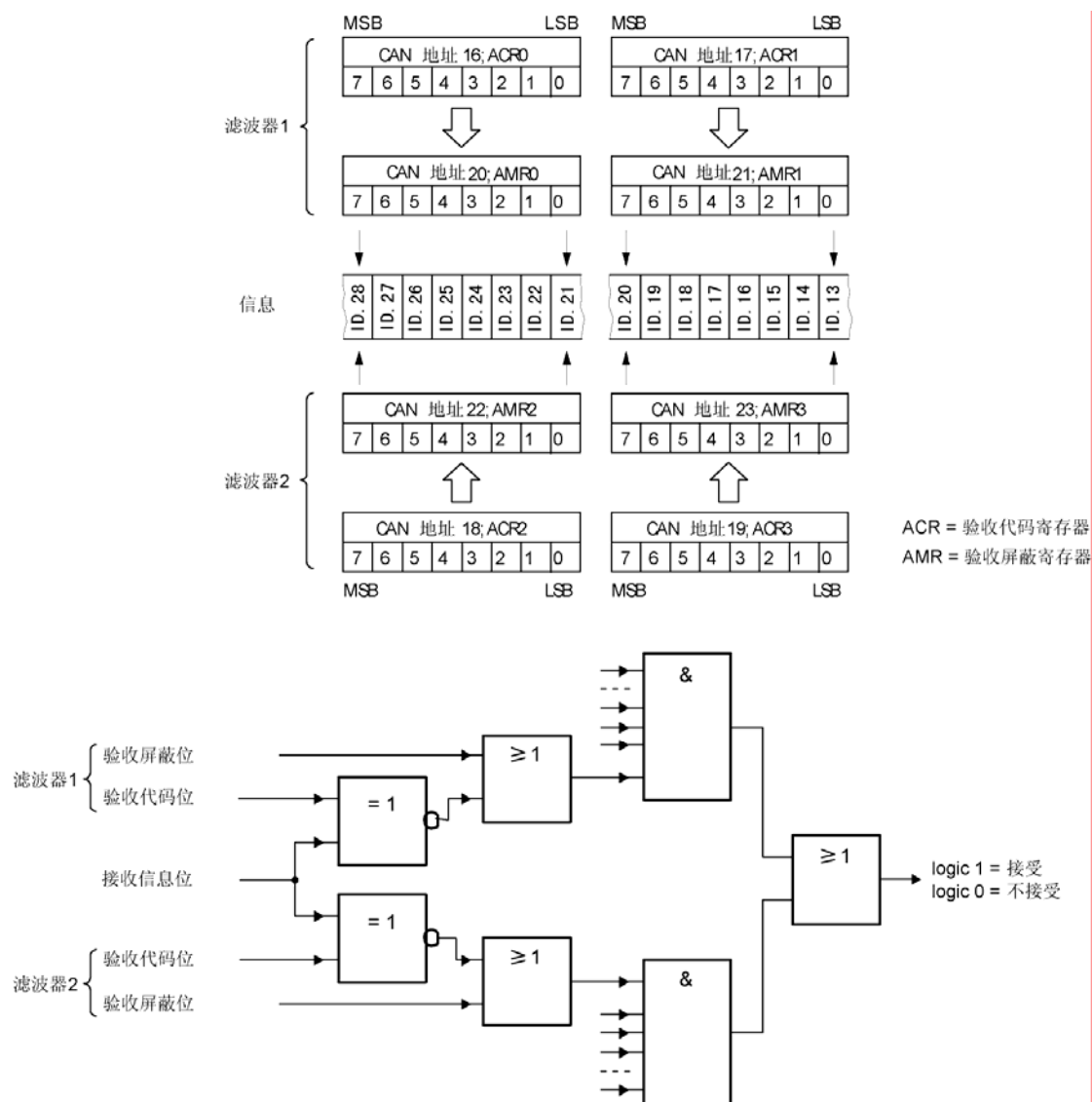
当滤波方式为单滤波，接收帧为扩展帧时：



当滤波方式为双滤波，接收帧为标准帧时：







4.4.4 **DWORD** `__stdcall VCI_ReadBoardInfo(DWORD DevType, DWORD DevIndex, PVCI_BOARD_INFO pInfo);`

入口参数：

**DevType**：

设备类型号。

**DevIndex**：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

**pInfo**：

用来存储设备信息的 **VCI\_BOARD\_INFO** 结构指针。

函数功能：

此函数用以获取设备信息。

返回值：

为 1 表示操作成功，0 表示操作失败。

#### 4.4.5 DWORD \_\_stdcall VCI\_ReadErrInfo(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI\_ERR\_INFO pErrInfo);

入口参数：

DevType：

设备类型号。

DevIndex：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex：

第几路 CAN。（注：当要读取设备错误的时候，此参数应该设为 - 1。比如当调用 VCI\_OpenDevice、VCI\_CloseDevice 和 VCI\_ReadBoardInfo 这些与特定的第几路 CAN 操作无关的操作函数失败后，调用此函数来获取失败错误码的时候应该把 CANIndex 设为 - 1。）

pErrInfo：

用来存储错误信息的 VCI\_ERR\_INFO 结构指针。

函数功能：

此函数用以获取最后一次错误信息。（注：在 PCI9820 中不支持此函数）

返回值：

为 1 表示操作成功，0 表示操作失败。

参数表：

（注：pErrInfo->ErrCode 可能为下列各个错误码的多种组合之一）

pErrInfo->ErrCode	pErrInfo->Passive_ErrData	pErrInfo->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	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

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

=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-&gt;Passive\_ErrData[1] 表示接收错误计数器

PErrInfo-&gt;Passive\_ErrData[2] 表示发送错误计数器

当(PErrInfo-&gt;ErrCode&amp;0x0008)==0x0008 时，存在 CAN 控制器仲裁丢失错误

PErrInfo-&gt;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				

错误段表示功能表示

位					十进制值	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

4.4.6 **DWORD \_\_stdcall VCI\_ReadCanStatus(DWORD DevType, DWORD DevIndex, DWORD CANIndex, P VCI\_CAN\_STATUS pCANStatus);**

入口参数：

DevType：

设备类型号。

DevIndex：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex：

第几路 CAN。

pCANStatus：

用来存储 CAN 状态的 VCI\_CAN\_STATUS 结构指针。

函数功能：

此函数用以获取 CAN 状态。

返回值：

为 1 表示操作成功，0 表示操作失败。

#### 4.4.7 DWORD \_\_stdcall VCI\_GetReference(DWORD DevType,DWORD DevIndex,DWORD CANIndex,DWORD RefType,PVOID pData);

入口参数：

DevType：

设备类型号。

DevIndex：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

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。

函数功能：

此函数用以获取设备的相应参数。（注：在 PCI9820 中不支持此函数）

返回值：

为 1 表示操作成功，0 表示操作失败。

参数表：

(1) 当设备类型为 PCI5121，PCI5110 或 ISA5420 时：

RefType	pData	功能描述
1	总长度为 2 个字节，pData[0] 表示 CAN 控制器的控制寄存器的地址，pData[1] 表示要读出 CAN 控制器的控制寄存器的值。	<p>读取 CAN 芯片某个寄存器的值</p> <p>例如要读取地址为 9 的寄存器值：</p> <p>UCHAR pData[2] = {9,0};</p> <p>VCI_GetReference(DeviceType,DeviceInd,CANInd,1,pData);</p> <p>如果调用成功，pData[1]将存放读出的值。</p>

(2) 当设备类型为 USBCAN1，USBCAN2 时：

RefType	pData	功能描述
---------	-------	------

1	<p>总长度 1 个字节, 当作为输入参数时, 表示为所要读取的 CAN 控制器的控制寄存器的地址。</p> <p>当作为输出参数时, 表示为 CAN 控制器的控制寄存器的值。</p>	<p>读 CAN 控制器的指定控制寄存器的值, 例如对 USBCAN1:</p> <p>BYTE val=0;</p> <p>VCI_GetReference(VCI_USBCAN1,0,0,1,(PVOID)&amp;val);</p> <p>如果此函数调用成功, 则在 val 中返回寄存器的值。</p>
---	--	--

## (3) 当设备类型为 CAN232 时:

RefType	pData	功能描述
1	<p>总长度 14 个字节, 当作为输入参数时, 只有第一个字节有效, 值为所读取的滤波器的序号, 以为 1, 2, 3, 4。</p> <p>当作为输出参数时, 具体各个字节所代表的意义见表后。</p>	<p>取得指定的滤波器参数, 例如要读取第一个滤波器的参数, 可以这样:</p> <p>BYTE info[14];</p> <p>info[0]=1;</p> <p>VCI_GetReference(VCI_CAN232,0,0,1,(PVOID)info);</p> <p>如果此函数调用成功, 则在 info 中返回 14 个字节的第一个滤波器的参数。</p>
2	<p>总长度 1 个字节, 当作为输入参数时, 表示为所要读取的 CAN 控制器的控制寄存器的地址。</p> <p>当作为输出参数时, 表示为 CAN 控制器的控制寄存器的值。</p>	<p>读 CAN 控制器的指定控制寄存器的值, 例如:</p> <p>BYTE val=0;</p> <p>VCI_GetReference(VCI_CAN232,0,0,2,(PVOID)&amp;val);</p> <p>如果此函数调用成功, 则在 val 中返回寄存器的值。</p>

当 RefType=1 时, 此时返回的 pData 各个字节所代表的意义如下:

pData[0]信息保留

pData[1]表示CAN控制器BTR0的值;

pData[2]表示CAN控制器BTR1的值;

pData[3]读取该组验收滤波器模式, 位功能

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
-----						MFORMATB	AMODEB

MFORMATB=1; 验收滤波器该组仅用于扩展帧信息。标准帧信息被忽略。

=0; 验收滤波器该组仅用于标准帧信息。扩展帧信息被忽略。

AMODEB =1; 单验收滤波器选项使能 - 长滤波器有效

=0; 双验收滤波器选项使能 - 短滤波器有效

pData[4]读取该组验收滤波器的使能, 位功能

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
-----						BF2EN	BF1EN

BF2EN=1; 该组滤波器2使能, 不能对相应的屏蔽和代码寄存器进行写操作

=0; 该组滤波器2 禁止, 可以改变相应的屏蔽和代码寄存器

BF1EN=1; 该组滤波器1使能, 不能对相应的屏蔽和代码寄存器进行写操作

=0；该组滤波器1禁止，可以改变相应的屏蔽和代码寄存器

注：如果选择单滤波器模式，该单滤波器与对应的滤波器1 使能位相关。滤波器2 使能位在单滤波器模式中不起作用。

pData[5]读取该组验收滤波器的优先级,位功能

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
-----						BF2PRIO	BF1PRIO

BF2PRIO=1；该组滤波器2优先级高，如果信息通过该组滤波器2 ，立即产生接收中断

= 0；该组滤波器2优先级低，如果FIFO 级超过接收中断级滤波器，产生接收中断

BF1PRIO=1；该组滤波器1优先级高，如果信息通过该组滤波器1 ，立即产生接收中断

= 0；该组滤波器1优先级低，如果FIFO 级超过接收中断级滤波器，产生接收中断

pData[6---9]表示该组滤波器ACR的值；

pData[a---d]表示该组滤波器 AMR 的值；

4.4.8    **DWORD    \_\_stdcall    VCI\_SetReference(DWORD    DevType,DWORD    DevIndex,DWORD    CANIndex,DWORD RefType,PVOID pData);**

入口参数：

DevType：

设备类型号。

DevIndex：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

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。

函数功能：

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。（注：在 PCI9820 中不支持此函数）

返回值：

为 1 表示操作成功，0 表示操作失败。

注：VCI\_SetReference 和 VCI\_GetReference 这两个函数是用来针对各个不同设备的一些特定操作的。比如 CAN232 的更改波特率，设置报文滤波等等。函数中的 PVOID 型参数 pData 随不同设备的不同操作而具有不同的意义。

参数表：

(1) 当设备类型为 PCI5121，PCI5110 或 ISA5420 时：

RefType	pData	功能描述
1	总长度为 2 个字节， pData[0] 表示 CAN 控制器的控制寄存器的地址， pData[1] 表示要写入的数值。	写 CAN 控制器的指定控制寄存器

(2) 当设备类型为 USBCAN1，USBCAN2 时：

RefType	pData	功能描述
1	总长度为 2 个字节， pData[0] 表示 CAN 控制器的控制寄存器的地址， pData[1] 表示要写入的数值。	写 CAN 控制器的指定控制寄存器

## (3) 当设备类型为 CAN232 时：

RefType	pData	功能描述
1	总长度为 1 个字节 = 0 ; 10Kbps = 1 ; 20Kbps = 2 ; 50Kbps = 3 ; 125Kbps = 4 ; 250Kbps = 5 ; 500Kbps = 6 ; 800Kbps = 7 ; 1000Kbps	更改 CAN 波特率，例如要设置 CAN 的波特率为 10Kbps : BYTE baud=0; VCI_SetReference(VCI_CAN232,0,0,1,(PVOID)&baud);
2	总长度为 12 个字节， 其各个字节所代表的意义见表后。	设置滤波器参数
3	总长度为 1 个字节 = 1 ; 2.4Kbps = 2 ; 4.8Kbps = 3 ; 9.6Kbps = 4 ; 14.4Kbps = 5 ; 19.2Kbps = 6 ; 28.8Kbps = 7 ; 57.6Kbps	更改 232 波特率
4	总长度为 2 个字节， pData[0] 表示 CAN 控制器的控制寄存器的地址， pData[1] 表示要写入的数值。	写 CAN 控制器的指定控制寄存器
5	总长度为 1 个字节， = 0xAA ; 使用时间标识 = 其他 ; 不使用时间标识	设置时间标识

当 RefType=2 时,此时的 pData 各个字节所代表的意义如下：

pData[0]设置哪一组验收滤波器;共有 4 组,

=1:设置第1组;

=2:设置第2组;

=3:设置第3组;



=4:设置第4组;

pData[1]设置该组验收滤波器模式,位功能

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
-----						MFORMATB	AMODEB

MFORMATB=1; 验收滤波器该组仅用于扩展帧信息。标准帧信息被忽略。

=0; 验收滤波器该组仅用于标准帧信息。扩展帧信息被忽略。

AMODEB =1; 单验收滤波器选项使能 - 长滤波器有效

=0; 双验收滤波器选项使能 - 短滤波器有效

pData[2]设置该组验收滤波器的使能,位功能

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
-----						BF2EN	BF1EN

BF2EN=1; 该组滤波器2使能，不能对相应的屏蔽和代码寄存器进行写操作

=0; 该组滤波器2 禁止，可以改变相应的屏蔽和代码寄存器

BF1EN=1; 该组滤波器1使能，不能对相应的屏蔽和代码寄存器进行写操作

=0; 该组滤波器1禁止，可以改变相应的屏蔽和代码寄存器

注：如果选择单滤波器模式，该单滤波器与对应的滤波器1 使能位相关。滤波器2 使能位在单滤波器模式中不起作用。

pData[3]设置该组验收滤波器的优先级,位功能

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
-----						BF2PRIO	BF1PRIO

BF2PRIO =1; 该组滤波器2优先级高，如果信息通过该组滤波器2，立即产生接收中断

=0; 该组滤波器2优先级低，如果FIFO 级超过接收中断级滤波器，产生接收中断

BF1PRIO =1; 该组滤波器1优先级高，如果信息通过该组滤波器1，立即产生接收中断

=0; 该组滤波器1优先级低，如果FIFO 级超过接收中断级滤波器，产生接收中断

pData[4---7] 分别对应要设置的SJA1000的ACR0---ACR3的值；

pData[8---b] 分别对应要设置的SJA1000的AMR0---AMR3的值；

**4.4.9 ULONG \_\_stdcall VCI\_GetReceiveNum(DWORD DevType,DWORD DevIndex,DWORD CANIndex);**

入口参数：

DevType：

设备类型号。

DevIndex：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex：

第几路 CAN。

函数功能：

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。

返回值：

返回尚未被读取的帧数。

**4.4.10 DWORD \_\_stdcall VCI\_ClearBuffer(DWORD DevType,DWORD DevIndex,DWORD CANIndex);**

入口参数：

DevType：

设备类型号。

DevIndex：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex：

第几路 CAN。

函数功能：

此函数用以清空指定缓冲区。

返回值：

为 1 表示操作成功，0 表示操作失败。

#### 4.4.11 **DWORD \_\_stdcall VCI\_StartCAN(DWORD DevType,DWORD DevIndex,DWORD CANIndex);**

入口参数：

DevType：

设备类型号。

DevIndex：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex：

第几路 CAN。

函数功能：

此函数用以启动 CAN。

返回值：

为 1 表示操作成功，0 表示操作失败。

#### 4.4.12 **DWORD \_\_stdcall VCI\_ResetCAN(DWORD DevType,DWORD DevIndex,DWORD CANIndex);**

入口参数：

DevType：

设备类型号。

DevIndex：

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex：

第几路 CAN。

函数功能：

此函数用以复位 CAN。

返回值：

为 1 表示操作成功，0 表示操作失败。

#### 4.4.13 **ULONG \_\_stdcall VCI\_Transmit(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI\_CAN\_OBJ pSend, ULONG Len);**

入口参数：

DevType :

设备类型号。

DevIndex :

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex :

第几路 CAN。

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。

函数功能：

此函数向指定的设备发送数据。

返回值：

返回实际发送的帧数。

**4.4.14 ULONG \_\_stdcall VCI\_Receive(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PPCI\_CAN\_OBJ pReceive, ULONG Len, INT WaitTime= - 1);**

入口参数：

DevType :

设备类型号。

DevIndex :

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex :

第几路 CAN。

pReceive :

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。

WaitTime :

等待超时时间，以毫秒为单位。

函数功能：

此函数从指定的设备读取数据。

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 VCI\_ReadErrInfo 函数来获取错误码。

#### 4.5 接口库函数使用流程

