

PCI-6000 双口智能 CAN 卡

使用说明书

北京华创至诚科技有限公司

北京华创至诚科技有限公司
邮政编码: 100086

地址: 北京市昌平区珠江摩尔国际中心7号1单元805
电话: 010-62979188 010-51662162 传真: 010-51662162

<http://www.ehcgk.com>

Sales E-mail: ehcgk@ehcgk.com

E-mail: 97101798@qq.com

PCI-6000 双口智能 CAN 卡使用说明书

1 综述

PCI-6000 智能卡是高性能的基于 PCI 总线的通讯卡,为你的 PC 和 CAN 总线连接提供了方便可靠的解决方案。PCI 总线符合 PCI2.1 标准, CAN 接口符合 CAN2.0B 协议。卡上集成两个独立且隔离的 CAN 总线数据通道,提高了在恶劣环境下工作的产品可靠性。支持 WIN2000、WINXP WIN 7/8/10多种操作系统,同时提供 VC++的程序举例,方便用户快速开发自己的应用程序。支持多卡工作,最多可支持 8 块卡在同一台 PC 上运行。

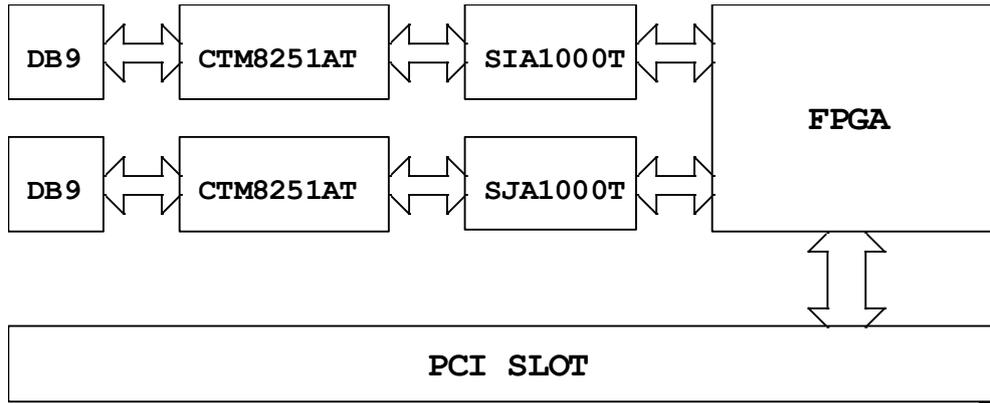
2 技术参数

- PCI 接口 : 32 位 33MHz PCI 数据总线,符合 PCI2.1 规范,支持 PNP;
- 数据存储器: FPGA 内部集成 8KB 双口 RAM 作为数据缓冲区;
- CAN 控制器: NXP SJA1000T;
- CAN 传输器: CTM8251AT;
- 数据传输速率: 可编程的传输速率,支持 5Kbps 到 1Mbps;
- CAN 通讯接口: 采用 DB9;
- CAN 通讯协议: CAN2.0B 协议(PeliCAN),兼容 CAN2.0A 协议;
- 隔离电压: 1000V;
- 工作温度: 0-70 度
- 板卡尺寸: 121mm X 87mm;
- 操作系统支持: WIN2000/XP/WIN7 /8/10;

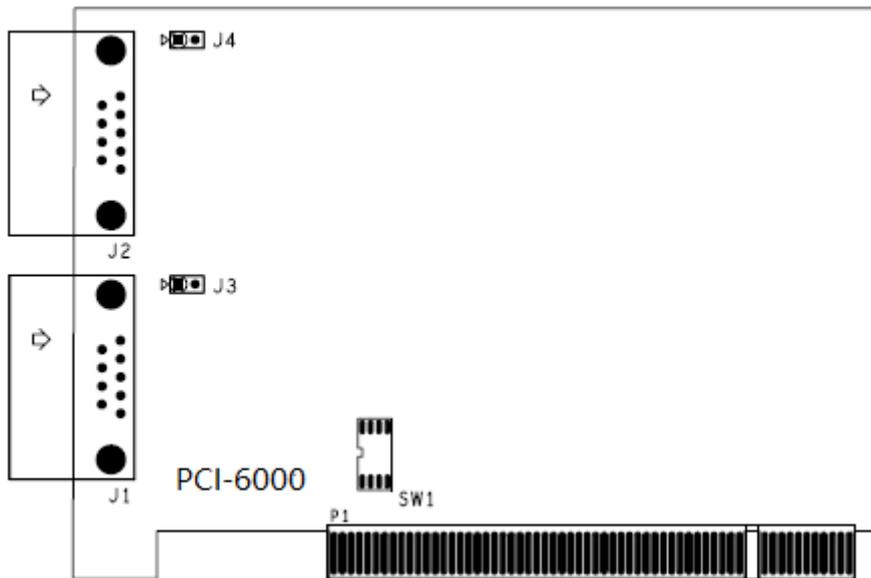
3 硬件原理

3.1 硬件原理框图

卡上采用单片 FPGA 集成 PCI9052 核,双口 RAM 和单片机的控制功能,高集成度的设计提高了产品的可靠性。



3.2 主要连接器位置图



J1 为 CAN0 连接器，J2 为 CAN1 连接器，SW1 为板卡 ID 选择，J3 为 CAN0 终端匹配电阻跳线器，J4 为 CAN1 口终端匹配电阻跳线器。

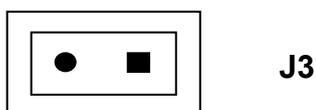
3.3 硬件连接器，跳线和拨码开关说明

3.3.1 DB9 连接器引脚定义

引脚号	信号	功能
2	CAN_L	CAN_L 信号
7	CAN_H	CAN_H 信号
3, 6	GND	信号地
5	GND_SHIELD	屏蔽地
1,4,8,9	NC	空脚

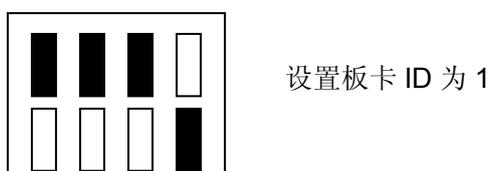
3.3.2 终端匹配电阻跳线器说明

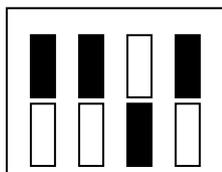
卡上可通过跳线选择是否采用终端匹配电阻，J3 为 CAN0 口的终端匹配电阻跳线器，J4 为 CAN1 口终端匹配电阻的跳线器。如果采用终端匹配电阻的话，须将加跳帽使能，如下图：



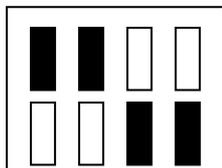
3.3.3 板卡拨码 ID 说明

如果需要在同一台 PC 上支持多卡，每块卡需要设置不同的 ID 号，一台机器最多可支持 8 块卡，ID 号从 0 到 7，通过板上 SW1 可设置板卡的 ID 号。出厂 ID 的默认设置为 0。

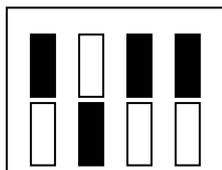




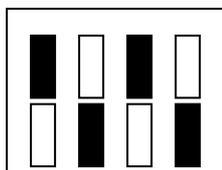
设置板卡 ID 为 2



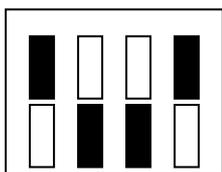
设置板卡 ID 为 3



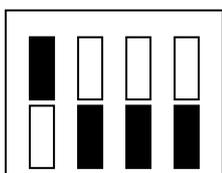
设置板卡 ID 为 4



设置板卡 ID 为 5

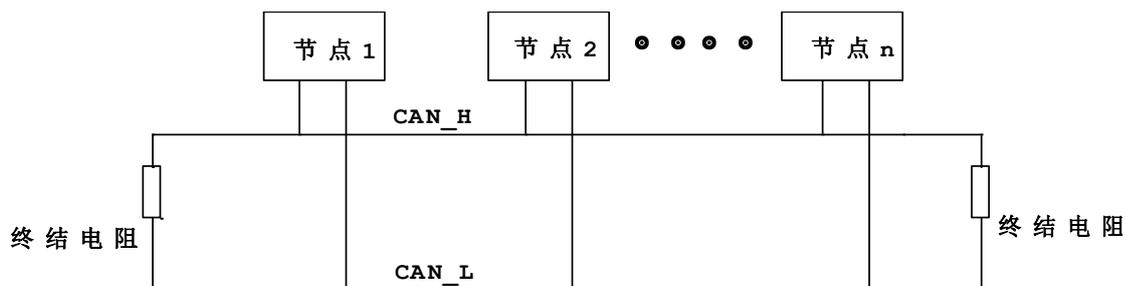


设置板卡 ID 为 6



设置板卡 ID 为 7

4 CAN 总线连接



为了提高 CAN 总线通讯的可靠性，每个 CAN 口都集成了一个 120 欧终端匹配电阻，终端匹配电阻的阻值是由 CAN 总线通讯电缆的阻抗来决定的，本卡的终端匹配电阻是按照双绞线的阻抗来决定的。可通过 J3 和 J4 的跳线来选择是否采用终端匹配电阻，在 CAN 通讯网络的首末节点时应采用终端匹配电阻来保证传输信号的可靠性。另外，请注意当传输距离超过 1 公里时，传输电缆的线径应大于 1MM。传输电缆可采用双绞线或带屏蔽的双绞线。

5 安装驱动程序

为了确保任何时候安装都可以正确指定相应的驱动程序，请严格按照以下步骤进行安装处理。

如果 PCI-6000 智能卡已经装好，则在重启操作系统后一般都会提示发现新硬件，如下图所示，此时应该选择“从列表或指定位置安装（高级）”，然后单击“下一步”，



图 5-1 欢迎使用找到新硬件向导

当出现如下所示的对话框，选择“不要搜索，我要自己选择要安装的驱动程序”项然后单击“下一步”，



图 5-2 请选择你的搜索和安装选项

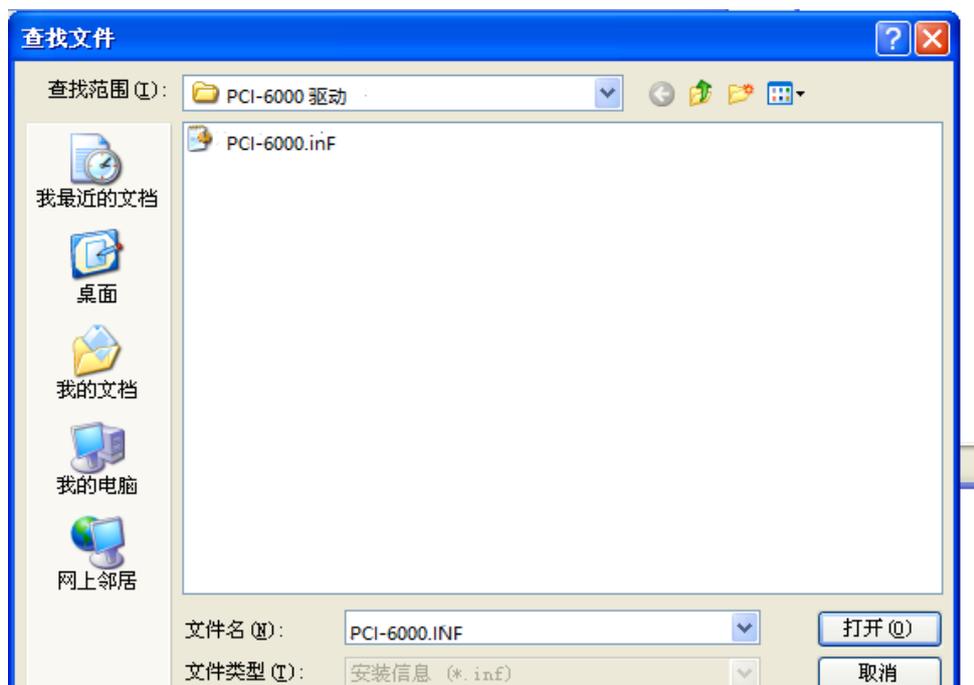
当出现“选择要为此硬件安装的设备驱动程序”对话框时，单击“型号”栏中的空白处，暂时不选中任何型号，接着单击“从磁盘安装”按钮以指定PCI-6000驱动文件目录位置。

当出现如下所示对话框时，我们可以通过“浏览”按钮找到驱动程序的 inf 文件。



图 5-3 从磁盘安装

查找文件结果如下图所示，选中相应文件“PCI-6000.inf”后单击“打开”。



当出现再次出现“从磁盘安装”对话框并确定（目录正确后目录不对的话必须单击“上一步”重新查找到对为止），单击“确定”键，

接着如下图所示，这时我们才选中相应的板卡型号，确定选中正确型号后单击“下一步”，

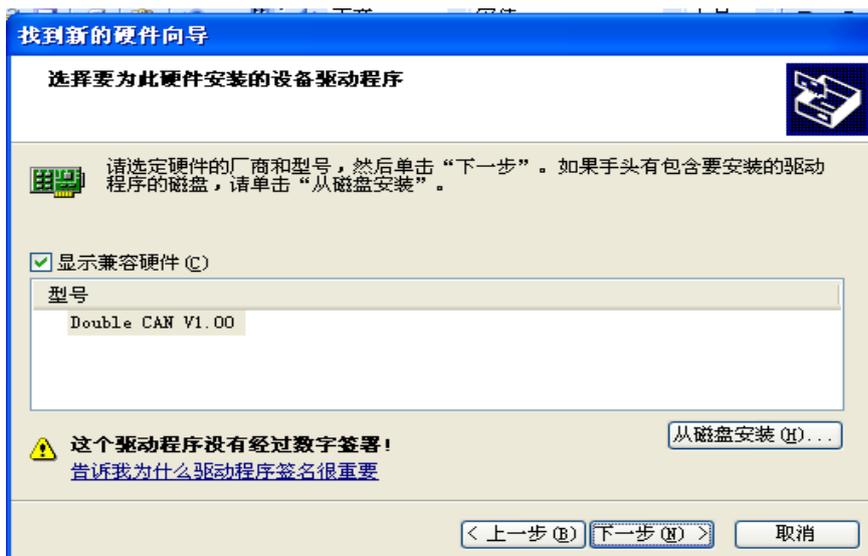
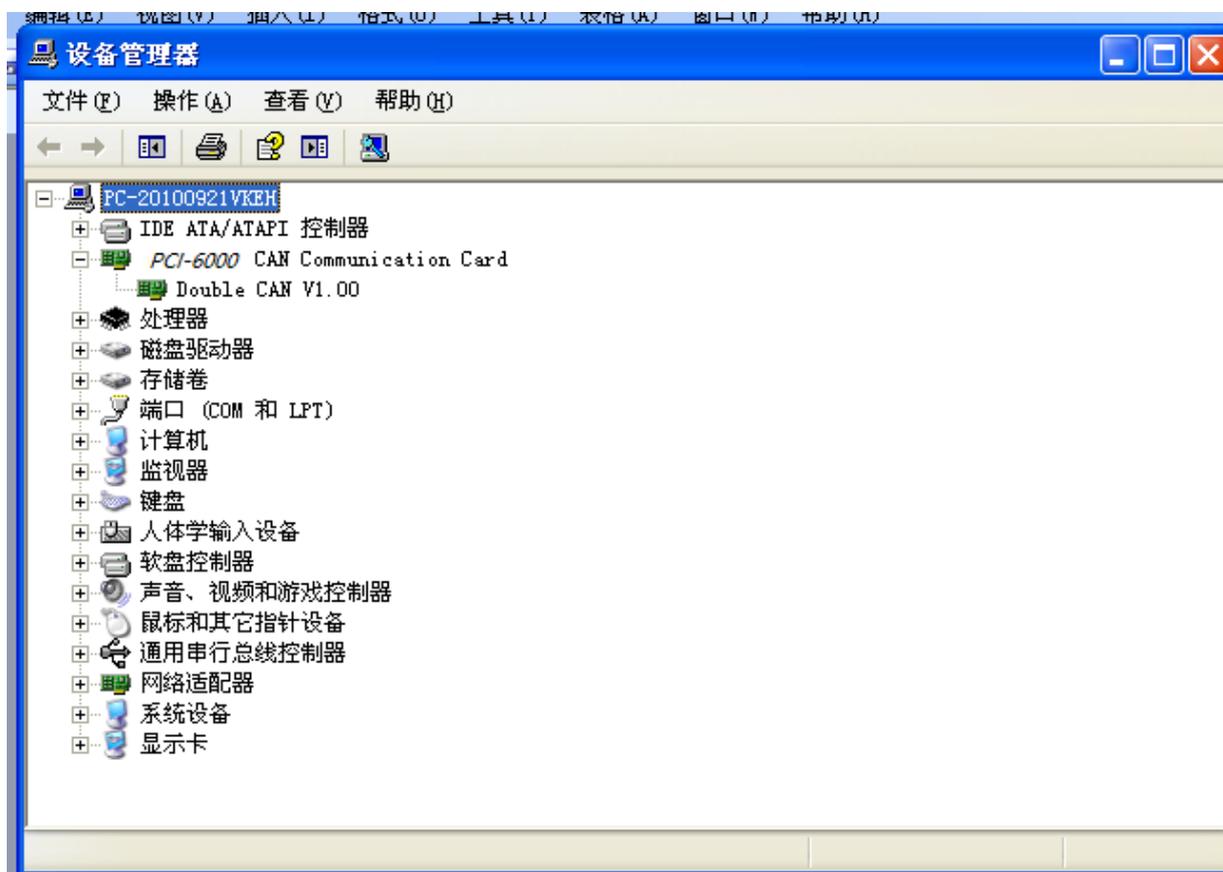


图 5-4 选择要为此硬件安装的设备驱动程序

如果安装过程没有出现任何错误，最后应该出现“完成找到新硬件向导”对话框，单击“完成”就可以了。安装成功后“设备管理器”中将列出所安装的PCI-6000。下图为安装完成1块 PCI-6000智能卡后的设备管理器界面。



6 接口函数

PCI-6000智能卡采用WDM驱动程序，支持WIN2000、WinXP/WIN7/8/10操作系统，支持一机多卡；同时提供完整VC++开发例程示范代码，用户进行开发。

6.1 接口卡设备类型定义

接口卡的类型定义如下：设备名称: PCI_6000 设备类型号: 1

6.2 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

6.3 函数库中的数据结构定义

VCI_BOARD_INFO

描述

VCI_BOARD_INFO结构体包含接口卡的设备信息。结构体将在VCI_ReadBoardInfo函数中被填充。

```
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, *PVCI_BOARD_INFO;
```

成员

hw_Version

硬件版本号，用16进制表示。比如0x0100表示V1.00。

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，比如“Double CAN V1.00”（注意：包括字符串结束符'\0'）。

Reserved

系统保留。

VCI_CAN_OBJ

描述

VCI_CAN_OBJ结构体在VCI_Transmit和VCI_Receive函数中被用来传送CAN信息帧。

```
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, *PVCI_CAN_OBJ;
```

成员

ID

报文ID。

TimeStamp

接收到信息帧时的时间标识，用于区别相同的帧。

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，=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。

DataLen

数据长度(<=8)，即Data的长度。

Data

报文的数据。

Reserved

系统保留。

VCI_CAN_STATUS

描述

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

```
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;
```

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。

VCI_ERR_INFO

描述

VCI_ERR_INFO 结构体用于装载PCI-6000库运行时产生的错误信息。结构体将在VCI_ReadErrInfo函数中被填充。

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

成员

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。

VCI_INIT_CONFIG

描述

VCI_INIT_CONFIG结构体定义了初始化CAN的配置。结构体将在VCI_InitCan函数中被填充。

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

成员

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。

备注

Timing0和Timing1用来设置CAN波特率，几种常见的波特率设置如下：

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

6.4 接口库函数说明

VCI_OpenDevice

描述

此函数用以打开设备。

DWORD __stdcall VCI_OpenDevice(DWORD DevType, DWORD DevIndex, DWORD Reserved);

参数

DevType

设备类型号。

DevIndex

拨码ID。

Reserved

保留,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 "ControlCan.h"
int nDeviceType = 1; /* PCI-6000 */
int nDeviceInd = 0; /* CAN1 */
DWORD dwRel;
dwRel = VCI_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;
}
```

VCI_CloseDevice

描述

此函数用以关闭设备。

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

参数

DevType

设备类型号。

DevIndex

拨码ID。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ControlCan.h"
int nDeviceType = 1;
int nDeviceInd = 0;
BOOL bRel;
bRel = VCI_CloseDevice(nDeviceType, nDeviceIndex);
```

VCI_InitCan

描述

此函数用以初始化指定的CAN。

```
DWORD __stdcall VCI_InitCan(DWORD DevType, DWORD DevIndex, DWORD CANIndex,
PVCV_INIT_CONFIG pInitConfig);
```

参数

DevType

设备类型号。

DevIndex

拨码ID。

CANIndex

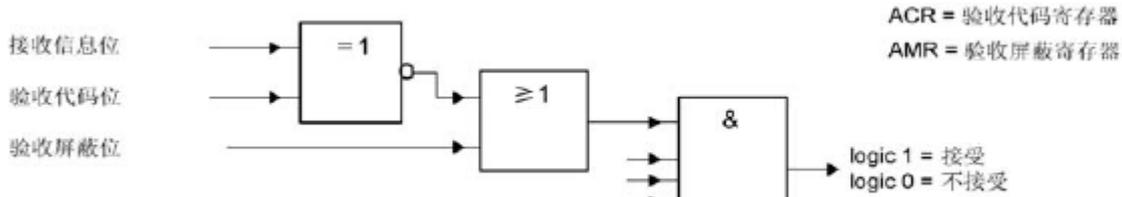
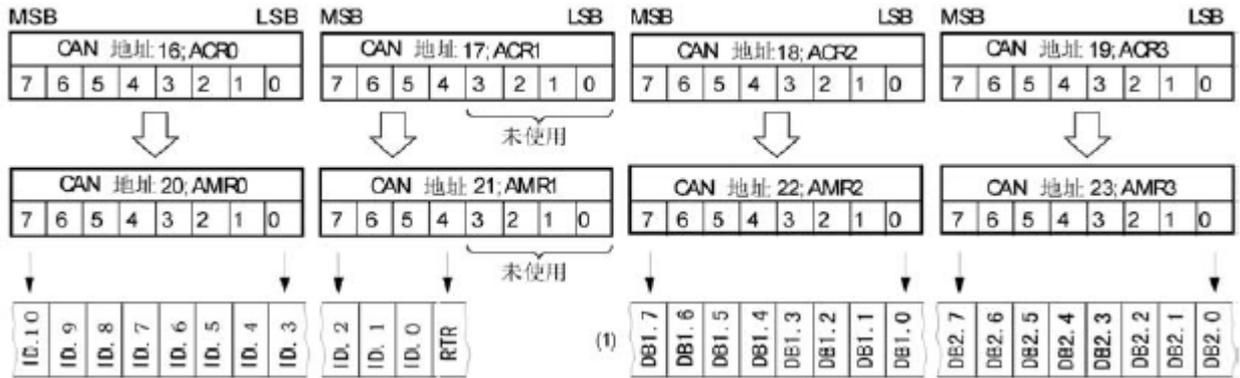
第几路CAN。

pInitConfig

初始化参数结构。成员	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;
pInitConfig->AccMask	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)

plnitConfig->Reserved	保留
plnitConfig->Filter	滤波方式, 1表示单滤波, 0表示双滤波
plnitConfig->Timing0	定时器0
plnitConfig->Timing1	定时器1
plnitConfig->Mode	模式, 0表示正常模式, 1表示只听模式

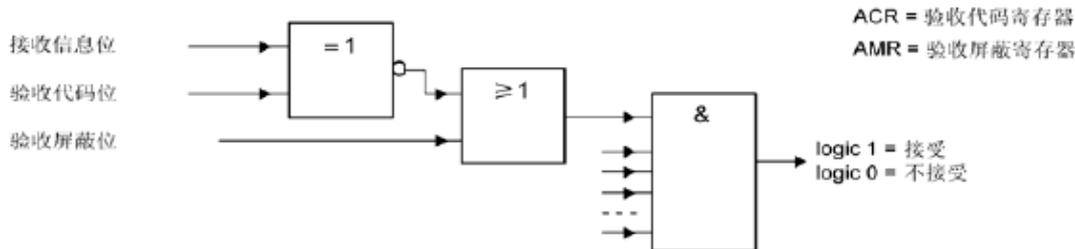
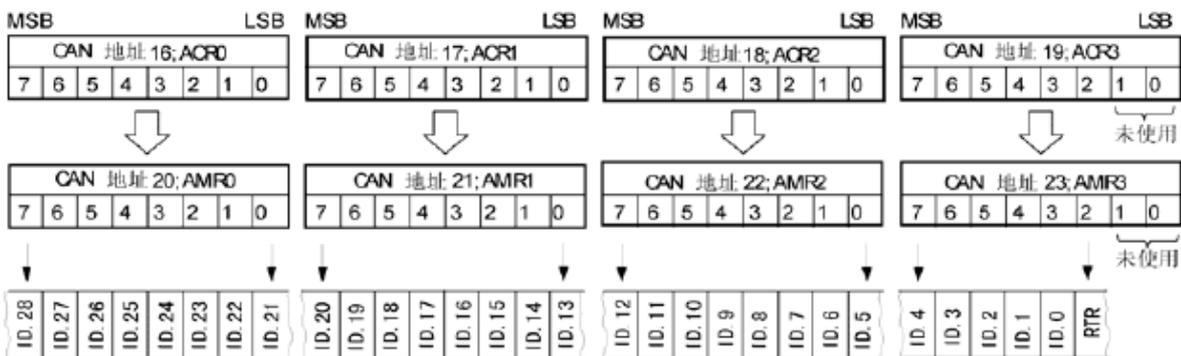
当滤波方式为单滤波, 接收帧为:



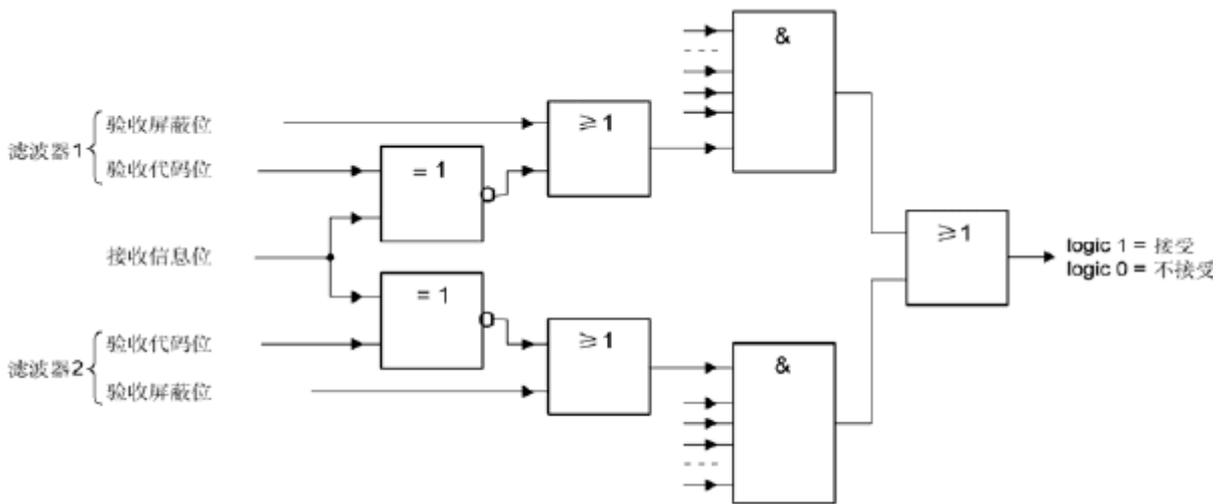
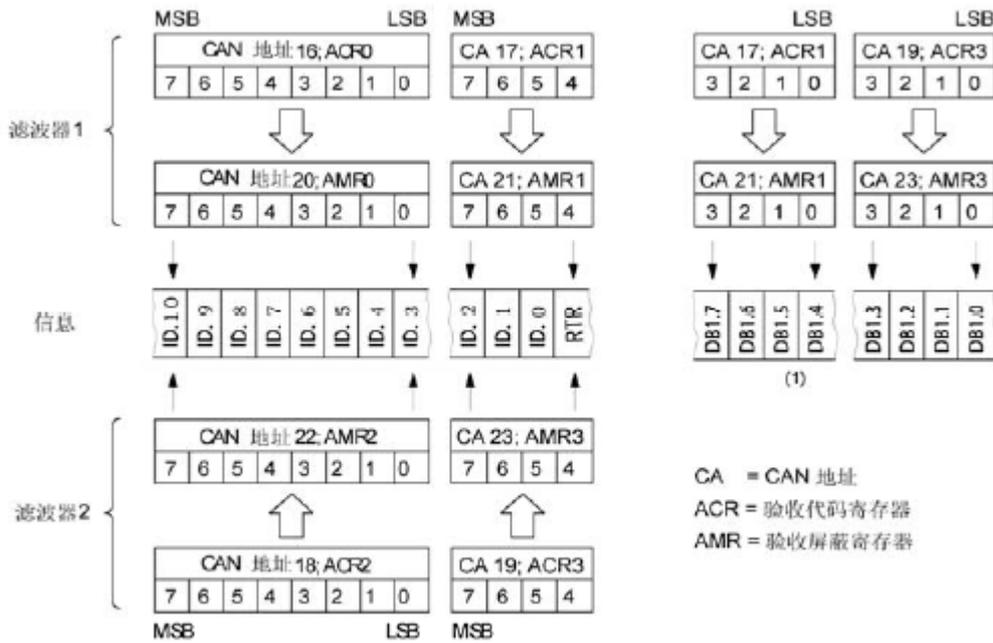
[DB1对应VCI_CAN_OBJ中的Data[0]]
DBx.y = 数据字节x的y位

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:

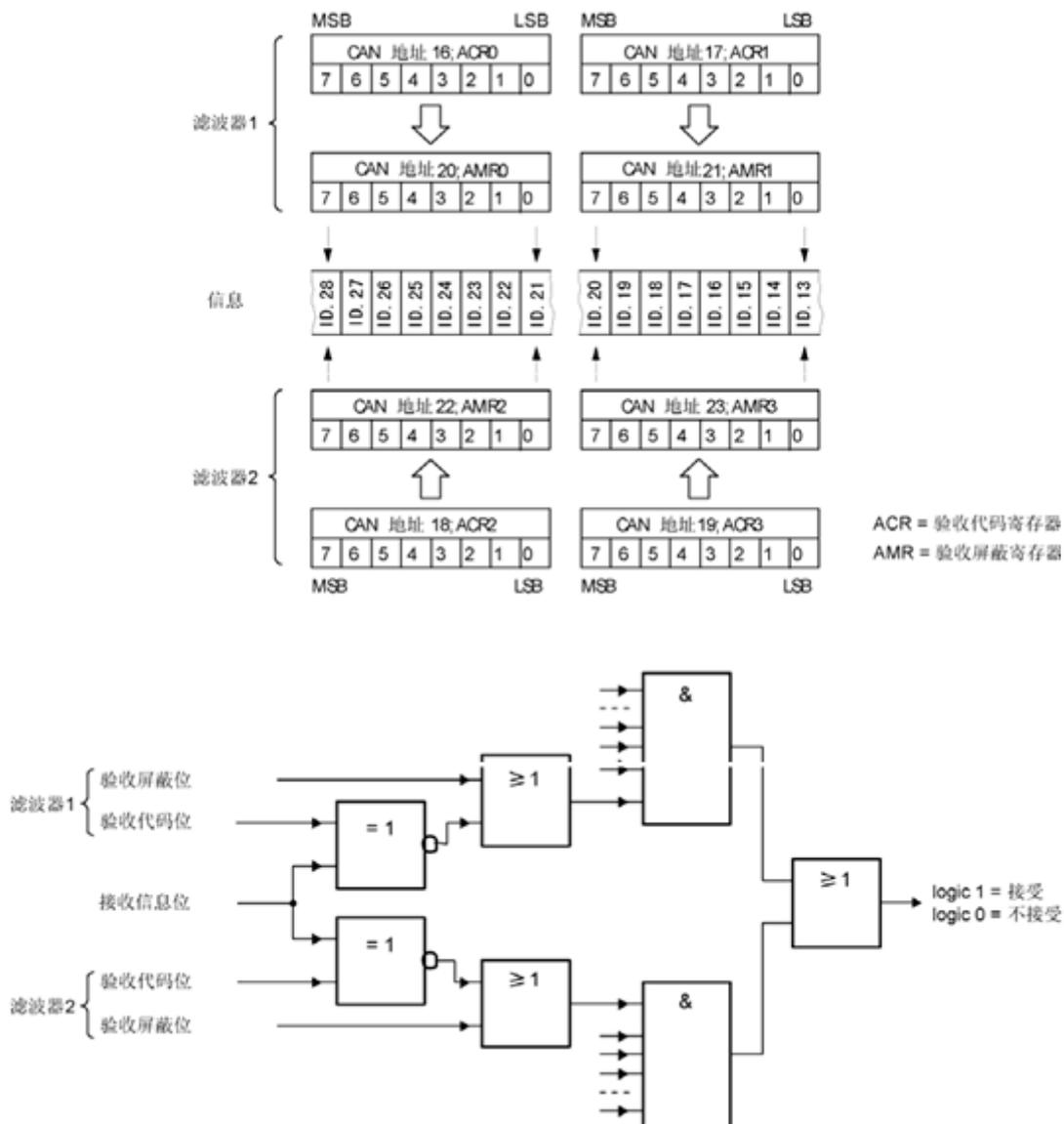


当滤波方式为双滤波，接收帧为标准帧时：



DBx.y = 数据字节x位y

当滤波方式为双滤波，接收帧为扩展帧时：



返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ControlCan.h"
int nDeviceType = 1;
int nDeviceInd = 0;
int nCANInd = 0;
VCI_INIT_CONFIG vic;
DWORD dwRel;
dwRel = VCI_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 = VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic);
if (dwRel == STATUS_ERR)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("初始化设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

VCI_ReadBoardInfo

描述

此函数用以获取设备信息。

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

参数

DevType

设备类型号。

DevIndex

拨码ID。

pInfo

用来存储设备信息的VCI_BOARD_INFO结构指针。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ControlCan.h"
int nDeviceType = 1;
int nDeviceInd = 0;
int nCANInd = 0;
VCI_INIT_CONFIG vic;
VCI_BOARD_INFO vbi;
DWORD dwRel;
bRel = VCI_ReadBoardInfo(nDeviceType, nDeviceInd, nCANInd, &vbi);
```

VCI_ReadErrInfo

描述

此函数用以获取最后一次错误信息。

```
DWORD __stdcall VCI_ReadErrInfo(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PPCI_ERR_INFO pErrInfo);
```

参数

DevType

设备类型号。

DevIndex

拨码ID。

CANIndex

第几路CAN。

pErrInfo

用来存储错误信息的VCI_ERR_INFO结构指针。

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
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

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。

=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, 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				

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

示例

```
DWORD dwRel;
```

```
bRel = VCI_ReadErrInfo(nDeviceType, nDeviceInd, nCANInd, &vei);
```

VCI_ReadCanStatus

描述

此函数用以获取CAN状态。

```
DWORD __stdcall VCI_ReadCanStatus(DWORD DevType, DWORD DevIndex, DWORD
CANIndex, P VCI_CAN_STATUS pCANStatus);
```

参数

DevType

设备类型号。

DevIndex

拨码ID。

CANIndex

第几路CAN。

pCANStatus

用来存储CAN状态的VCI_CAN_STATUS结构指针。

返回值

为1表示操作成功，0表示操作失败。

示例

```
#include "ControlCan.h"
```

```
int nDeviceType = 1;
```

```
int nDeviceInd = 0;
```

```
int nCANInd = 0;
```

```
VCI_INIT_CONFIG vic;
```

```
VCI_CAN_STATUS vcs;
```

```
DWORD dwRel;
```

```
bRel = VCI_ReadCANStatus(nDeviceType, nDeviceInd, nCANInd, &vcs);
```

VCI_GetReference

描述

此函数用以获取设备的相应参数。

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

参数

DevType

设备类型号。

DevIndex

拨码ID。

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。

备注

(1) 当设备类型为 KPCI_8210: 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。	读取CAN芯片某个寄存器的值。 例如要读取地址为9的寄存器值： UCHAR pData[2] = {9,0}; VCI_GetReference(DeviceType,D eviceInd,CANInd,1,pData); 如果调用成功，pData[1]将存放读出的 值。

示例

```
#include "ControlCan.h"
int nDeviceType = 1;
int nDeviceInd = 0;
int nCANInd = 0;
UCHAR info[2];
DWORD dwRel;
info[0] = 1;
bRel = VCI_GetReference(nDeviceType, nDeviceInd, nCANInd, 1, (PVOID)info);
```

VCI_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。

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

参数

DevType

设备类型号。

DevIndex

拨码ID。

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。

备注

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。

当设备类型为KPCI_8210,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

示例

```
#include "ControlCan.h"
int nDeviceType = 1;
int nDeviceInd = 0;
int nCANInd = 0;
UCHAR info[2];
DWORD dwRel;
info[0] = 1;
info[1] = 2;
bRel = VCI_SetReference(nDeviceType, nDeviceInd, nCANInd, 1, (PVOID)baud);
```

VCI_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。

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

参数

DevType

设备类型号。

DevIndex

拨码ID。

CANIndex

第几路CAN。

返回值

返回尚未被读取的帧数。

示例

```
#include "ControlCan.h"  
int nDeviceType = 1;  
int nDeviceInd = 0;  
int nCANInd = 0;  
DWORD dwRel;  
bRel = VCI_GetReceiveNum(nDeviceType, nDeviceInd, nCANInd);
```

VCI_ClearBuffer

描述

此函数用以清空指定缓冲区。

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

参数

DevType

设备类型号。

DevIndex

拨码ID。

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 "ControlCan.h"
int nDeviceType = 1;
int nDeviceInd = 0;
int nCANInd = 0;
DWORD dwRel;
bRel = VCI_ClearBuffer(nDeviceType, nDeviceInd, nCANInd);
```

VCI_StartCAN

描述

此函数用以启动CAN。

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

参数

DevType

设备类型号。

DevIndex

拨码ID。

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 "ControlCan.h"
int nDeviceType = 1;
int nDeviceInd = 0;
int nCANInd = 0;
VCI_INIT_CONFIG vic;
DWORD dwRel;
dwRel = VCI_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 = VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic);
if (dwRel == STATUS_ERR)
{
```

```
VCI_CloseDevice(nDeviceType, nDeviceInd);
MessageBox(_T("初始化设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
return FALSE;
}
dwRel = VCI_StartCAN(nDeviceType, nDeviceInd, nCANInd);
if (dwRel == STATUS_ERR)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("启动设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

VCI_ResetCAN

描述

此函数用以复位CAN。

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

参数

DevType

设备类型号。

DevIndex

拨码ID。

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 "ControlCan.h"
int nDeviceType = 1;
int nDeviceInd = 0;
int nCANInd = 0;
DWORD dwRel;
bRel = VCI_ResetCAN(nDeviceType, nDeviceInd, nCANInd);
```

VCI_Transmit

描述

返回实际发送的帧数。

```
ULONG __stdcall VCI_Transmit(DWORD DevType, DWORD DevIndex, DWORD  
CANIndex, PVCI_CAN_OBJ pSend, ULONG Len);
```

参数

DevType

设备类型号。

DevIndex

拨码ID。

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。

返回值

返回实际发送的帧数。

示例

```
#include "ControlCan.h"  
#include <string.h>  
int nDeviceType = 1;  
int nDeviceInd = 0;  
int nCANInd = 0;  
DWORD dwRel;  
VCI_CAN_OBJ vco;  
ZeroMemory(&vco, sizeof (VCI_CAN_OBJ));  
vco.ID = 0x00000000;  
vco.SendType = 0;  
vco.RemoteFlag = 0;  
vco.ExternFlag = 0;  
vco.DataLen = 8;  
IRet = VCI_Transmit(nDeviceType, nDeviceInd, nCANInd, &vco, i);
```

VCI_Receive

描述

此函数从指定的设备读取数据。

```
ULONG __stdcall VCI_Receive(DWORD DevType, DWORD DevIndex, DWORD CANIndex,  
PVCAN_OBJ pReceive, ULONG Len, INT WaitTime=-1);
```

参数

DevType

设备类型号。

DevIndex

拨码ID。

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。

返回值

返回实际读取到的帧数。

示例

```
#include "ControlCan.h"  
#include <string.h>  
int nDeviceType = 1;  
int nDeviceInd = 0;  
int nCANInd = 0;  
DWORD dwRel;  
VCI_CAN_OBJ vco[100];  
IRet = VCI_Receive(nDeviceType, nDeviceInd, nCANInd, vco, 100, 0);
```

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

VC调用动态库的方法

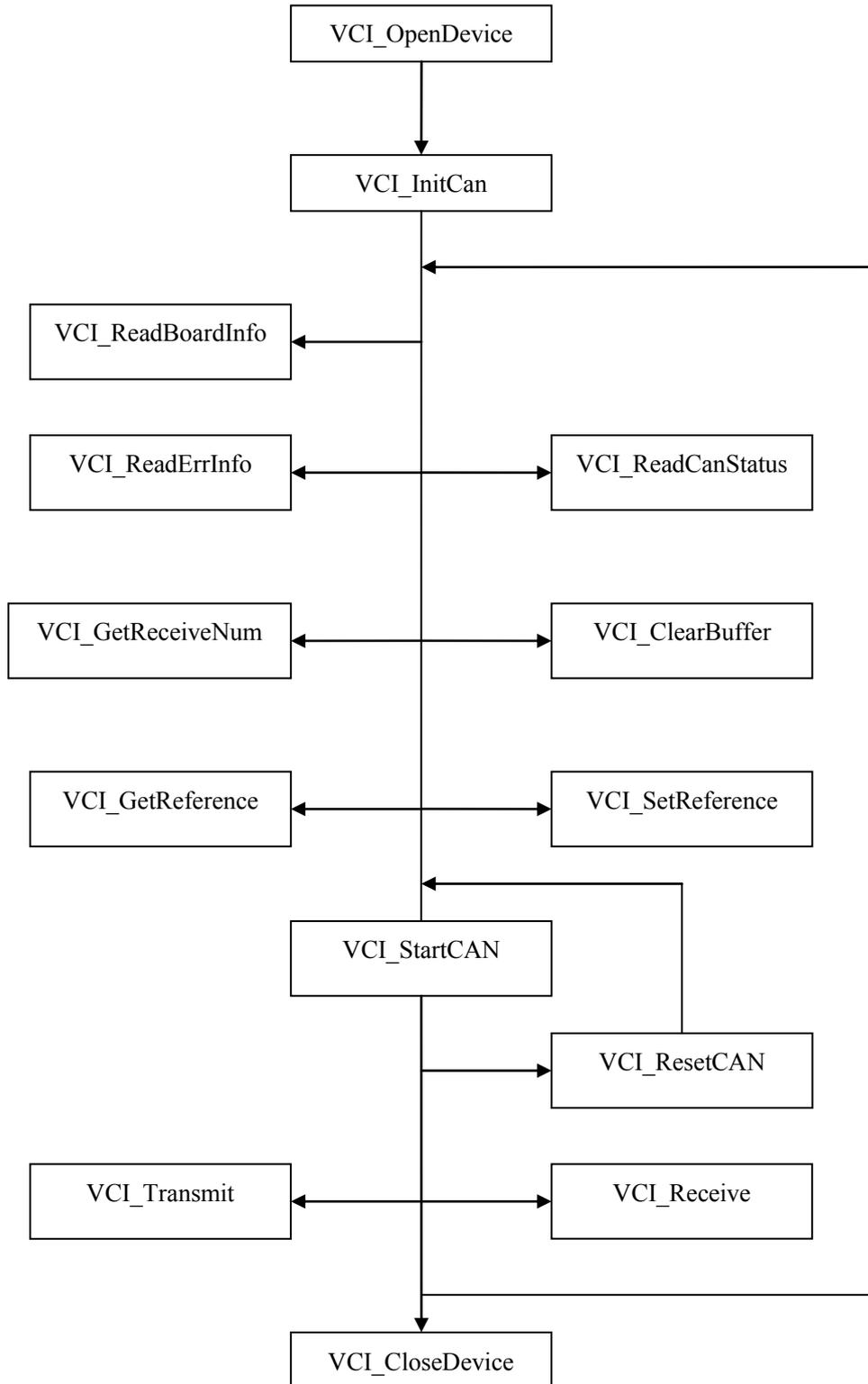
(1) 在扩展名为.CPP的文件中包含ControlCAN.h头文件。

如：`#include "ControlCAN.h"`

(2) 在工程的连接器设置中连接到ControlCAN.lib文件。

如：在VC6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ControlCAN.lib

6.6 接口库函数使用流程



7、常见问题

在使用本系统开发项目之前,最好能先看看相应操作系统的“系统”下“设备属性”中是否列出了PCI-6000的驱动选项,并且该选项里应该没有感叹号或问号。若出现问号说明驱动没有被正确安装,若出现感叹号则说明板卡硬件有问题,比如接触不良,这时应该采取措施确保硬件接触良好。

8、保修:

本产品自售出之日起两年内,凡用户遵守贮存、运输及使用要求,而产品质量低于技术指标的,凭保修单免费维修。因违反操作规定和要求而造成损坏的,需交纳器件和维修费。

9、产品成套性:

- 9.1 PCI-6000 双口智能 CAN 卡壹块。
- 9.2 光盘壹张。
- 9.3 9 芯 D 型插头两套。

附录 A CAN2.0B 协议帧格式

A.1 CAN2.0B 标准帧

CAN标准帧信息为11个字节包括两部分信息和数据部分前3个字节为信息部分。

	7	6	5	4	3	2	1	0	
字节 1	FF	RTR	X	X		DLC (数据长度)			
字节 2				(报文识别码) ID.10-ID.3					
字节 3		ID.2-ID.0		X	X	X	X	X	
字节 4				数据 1					
字节 5				数据 2					
字节 6				数据 3					
字节 7				数据 4					
字节 8				数据 5					
字节 9				数据 6					
字节 10				数据 7					
字节 11				数据 8					

◆ 字节1为帧信息第7位（FF）表示帧格式，在标准帧中，FF=0；第6位（RTR）表示帧的类型，RTR=0表示为数据帧，RTR=1表示为远程帧；DLC表示在数据帧时实际的数据长度。

◆ 字节2、3 为报文识别码，11位有效。

◆ 字节 4~11 为数据帧的实际数据，远程帧时无效。

A.2 CAN2.0B 扩展帧

CAN扩展帧信息为13个字节,包括两部分,信息和数据部分.前5个字节为信息部分。

	7	6	5	4	3	2	1	0
字节 1	FF	RTR	X	X		DLC (数据长度)		
字节 2				(报文识别码) ID.28-ID.21				
字节 3				ID.20-ID.13				
字节 4				ID.12-ID.5				
字节 5		ID.4-ID.0				X	X	X
字节 6				数据 1				
字节 7				数据 2				
字节 8				数据 3				
字节 9				数据 4				
字节 10				数据 5				
字节 11				数据 6				
字节 12				数据 7				
字节 13				数据 8				

- ◆ 字节1为帧信息。第7位（FF）表示帧格式，在扩展帧中，FF=1；第6位（RTR）表示帧的类型，RTR=0表示为数据帧，RTR=1表示为远程帧；DLC表示在数据帧时实际的数据长度。
- ◆ 字节2~5 为报文识别码，其高29位有效。
- ◆ 字节 6~13 为数据帧的实际数据，远程帧时无效。

附录 B SJA1000 标准波特率

SJA1000独立CAN控制器的CAN通讯波特率由寄存器BTR0、BTR1、晶振等参数共同决定。下表列出了一组推荐的BTR0、BTR1设置值。

表 SJA1000 标准波特率

序号	Baudrate (Kbps)	晶振频率 = 16MHz	
		BTR0 (Hex)	BTR1(Hex)
1	5	BF	FF
2	10	31	1C
3	20	18	1C
4	40	87	FF
5	50	09	1C
6	80	83	FF
7	100	04	1C
8	125	03	1C
9	200	81	FA
10	250	01	1C
11	400	80	FA
12	500	00	1C
13	666	80	B6
14	800	00	16
15	1000	00	14

参考资料

- <<SJA1000 独立的CAN 控制器>>
- <<SJA1000 独立的CAN 控制器应用指南>>
- <<确定 SJA1000 CAN 控制器的位定时参数>>