在 LabVIEW 下使用 ZLGCAN 接口函数库

广州周立功单片机发展有限公司
2005 年 3 月 18 日

目 录

第 1 章 概述 1
第 2 章 使用 VCI 函数 2
2.1 数据结构 2
2.1.1 VCI_BOARD_INFO 结构 2
2.1.2 VCI_CAN_OBJ 结构 3
2.1.3 VCI_CAN_STATUS 结构 3
2.1.4 VCI_ERR_INFO 结构 4
2.1.5 VCI_INIT_CONFIG 结构 4
2.2 调用 VCI 库函数 5
2.3 应用示例 8
第 3 章 VCI 函数调用参考 11
3.1 VCI_OpenDevice 11
3.2 VCI_CloseDevice 11
3.3 VCI_InitCan 11
3.4 VCI_ReadBoardInfo 12
3.5 VCI_ReadErrInfo 12
3.6 VCI_ReadCanStatus 13
3.7 VCI_GetReference 13
3.8 VCI_SetReference 14
3.9 VCI_GetReceiveNum 14
3.10 VCI_ClearBuffer 15
3.11 VCI_StartCAN 15
3.12 VCI_ResetCAN 15
3.13 VCI_Transmit 16
3.14 VCI_Receive 16

第1章　概述

Virtual CAN Interface (VCI) 函数库是专门为 ZLGCAN 设备在 PC 上使用而提供的应用程序接口。库里的函数从 ControlCAN.dll 中导出，在 LabVIEW7.0 中可以直接使用这些库函数而无需额外的操作。VCI 函数的使用流程如图 1.1 所示。

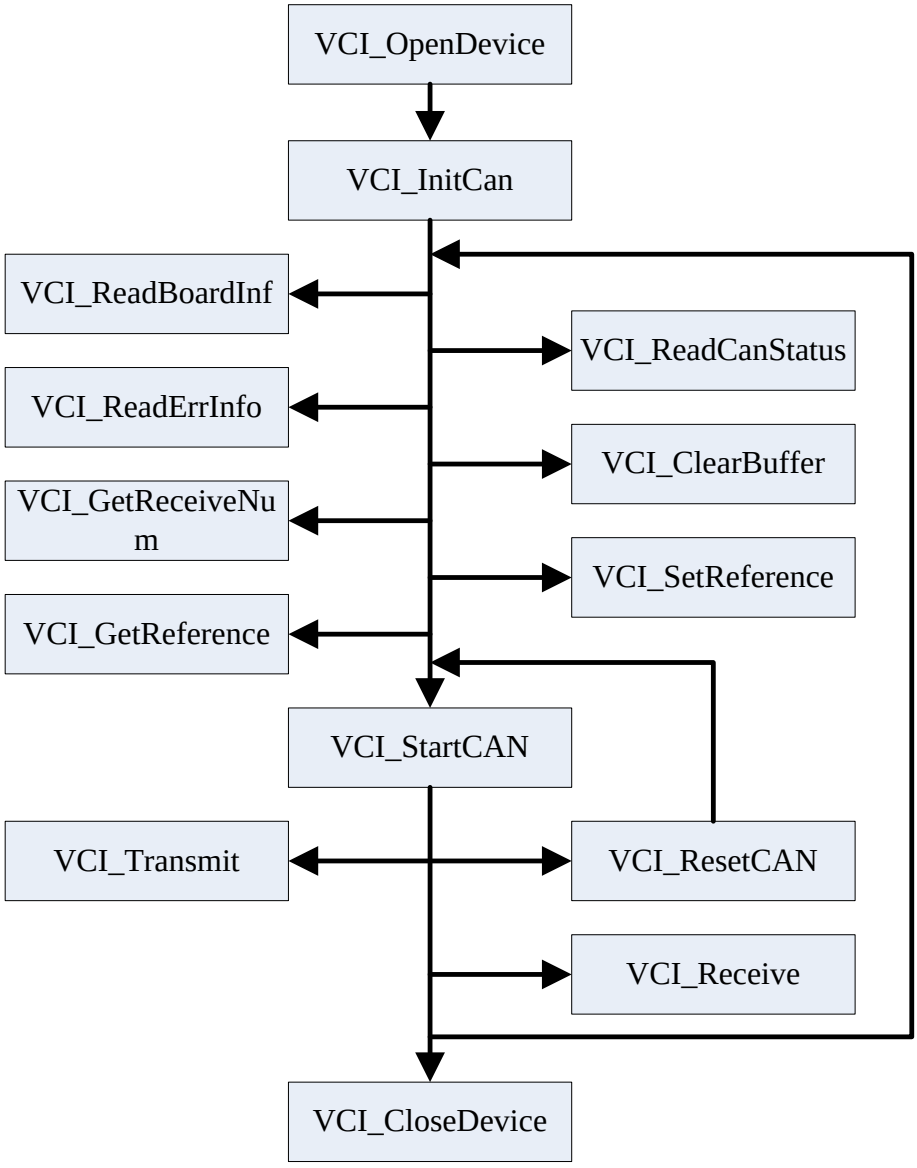

图 1.1　VCI 函数使用流程

第2章 使用 VCI 函数

2.1 数据结构

VCI 函数库中定义了一些数据结构用于数据交换，在使用 VCI 函数前应该先创建这些数据结构。在 LabVIEW 中创建这些结构时应该使用簇——Cluster。一个簇就是一个由若干不同的数据类型的成员组成的集合体，类似于 C 语言中的结构。其成员可以是任意的数据类型，但必须都是控件或都是显示件。成员的逻辑顺序是由它们被放入簇的先后顺序决定的。

2.1.1 VCI_BOARD_INFO 结构

VCI_BOARD_INFO 结构体包含 ZLGCAN 系列接口卡的设备信息。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;
```

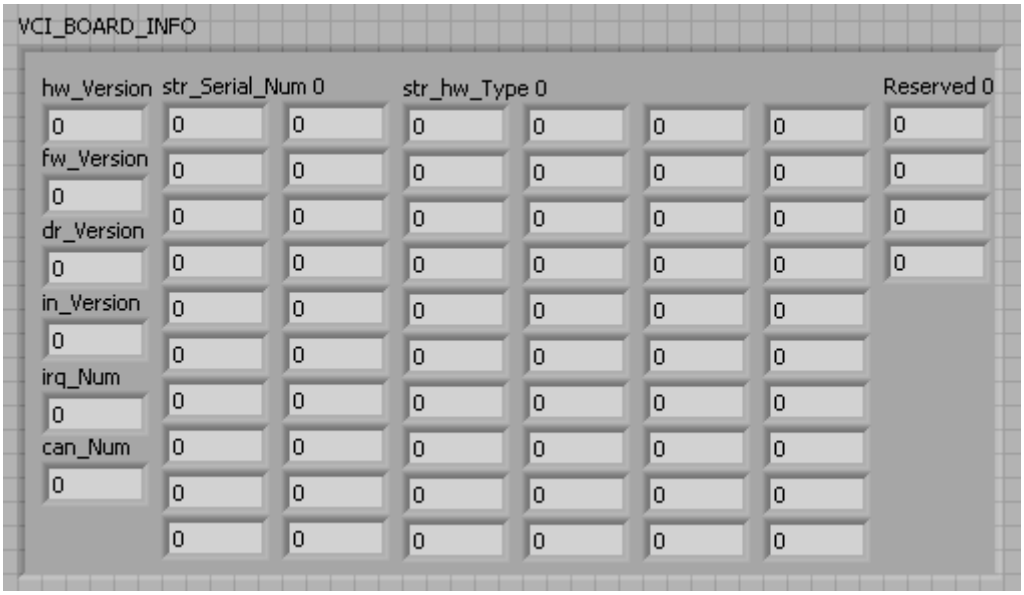

图 2.1 VCI_BOARD_INFO 结构

2.1.2 VCI_CAN_OBJ 结构

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;
```

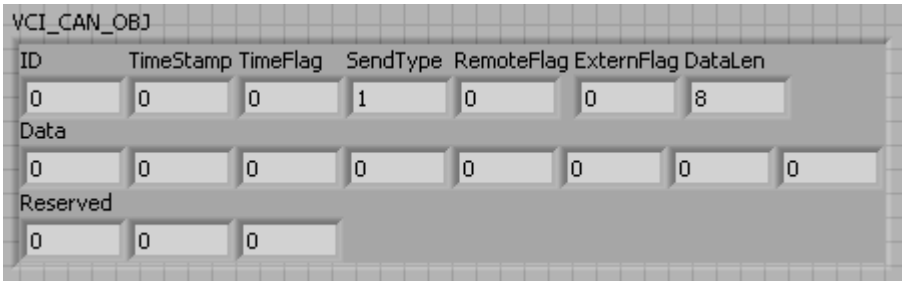

图 2.2 VCI_CAN_OBJ 结构

2.1.3 VCI_CAN_STATUS 结构

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;
```

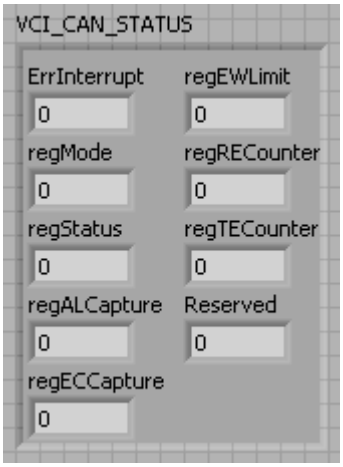

图 2.3 VCI_CAN_STATUS 结构

2.1.4 VCI_ERR_INFO 结构

VCI_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;
```

图 2.4 VCI_ERR_INFO 结构

2.1.5 VCI_INIT_CONFIG 结构

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;
```

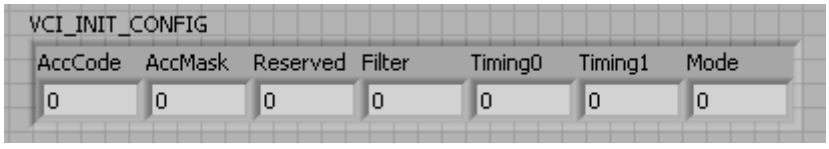

图 2.5 VCI_INIT_CONFIG 结构

下面以 VCI_CAN_OBJ 结构为例，在 LabVIEW7.0 中创建 VCI_CAN_OBJ 结构。先在 Array&Cluster 控件子模板选择一个簇的空壳放到前面板上，将其命名为 VCI_CAN_OBJ，然后根据需要放置的控件多少用定位工具调整簇空壳的大小；按照 VCI_CAN_OBJ 结构成员的顺序，从 Numeric 控件子模板中取 Numeric Indicator 控件或从前面板上移动控件到簇的空壳中，并按图 2.6 将各 Numeric Indicator 控件重命名。

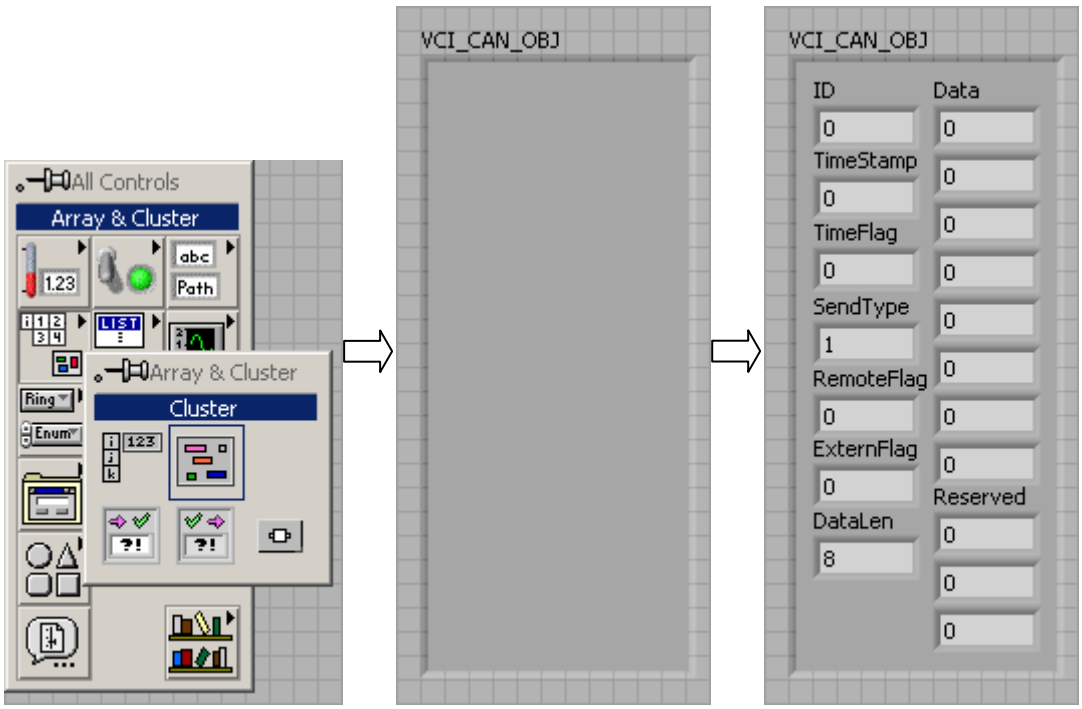

图 2.6　在前面板上创建 VCI_CAN_OBJ 结构的簇

此时，簇壳内的成员的数据类型都为默认的 Double 类型。在簇壳内的边框上弹出快捷菜单，选择 Representation。在下一级子菜单中选择与 VCI_CAN_OBJ 结构成员类型一致的类型。

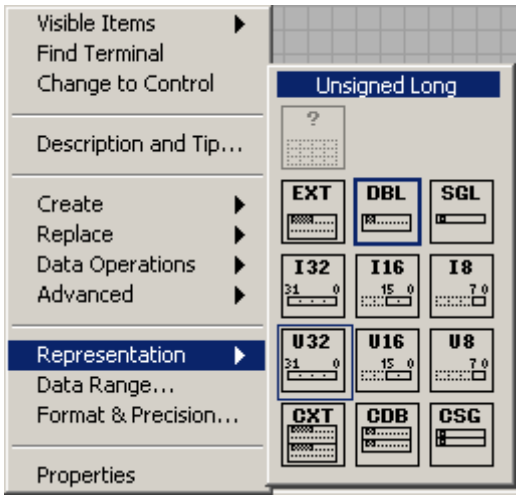

图 2.7　设置成员类型

2.2　调用 VCI 库函数

在 LabVIEW 中调用 VCI 库函数的过程比较简单。LabVIEW 在 Advanced 函数子模板中提供了 Calling Library Function Node，只要知道动态连接库里被导出的函数名称及其参数，就可以通过 Calling Library Function Node 调用。ZLG VCI 函数库已经提供了库里的函数声明，因此，在 LabVIEW 中使用 VCI 函数库将通过 Calling Library Function Node 来实现。

以调用 VCI_OpenDevice 函数为例。在 LabVIEW 图形代码窗口中放上调用库函数节点，用鼠标双击节点或使用快捷菜单命令 Configure 弹出如图 2.8 所示的对话框。

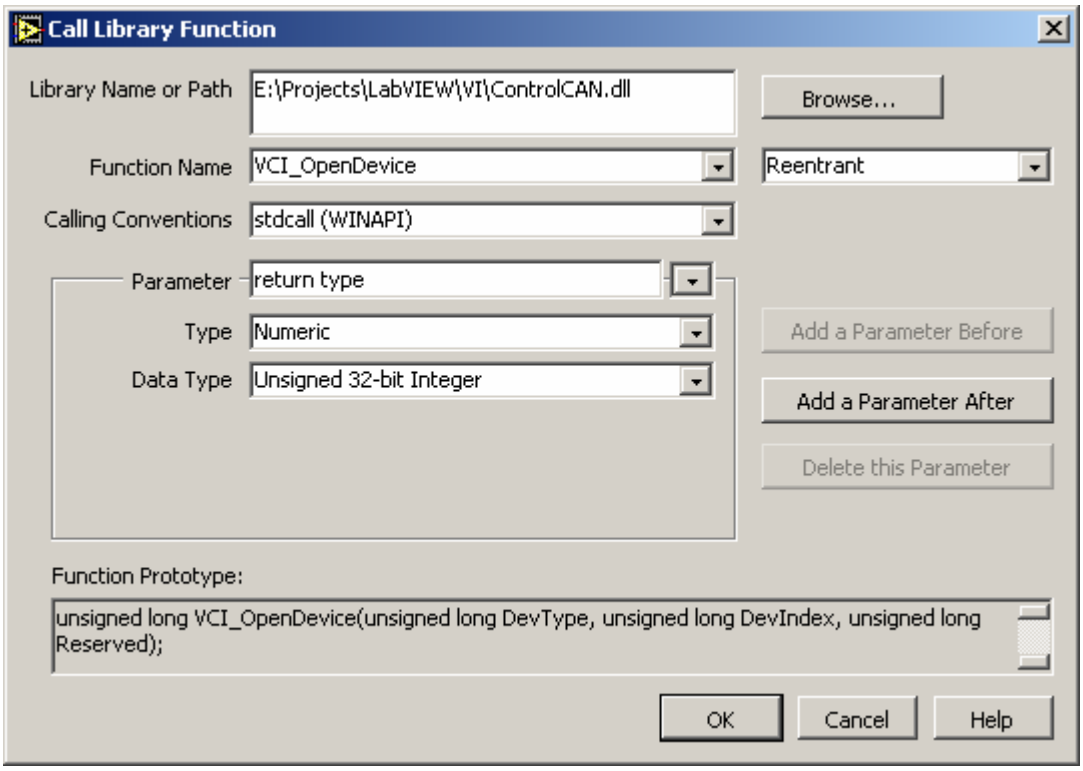

图 2.8 调用库函数 1

单击 Browse…按钮，打开一个文件对话框，找到 ControlCAN.dll 文件。或者直接输入库文件路径和名称。

在 Function Name 下拉列表框中照到 VCI_OpenDevice 函数。或直接输入函数名。

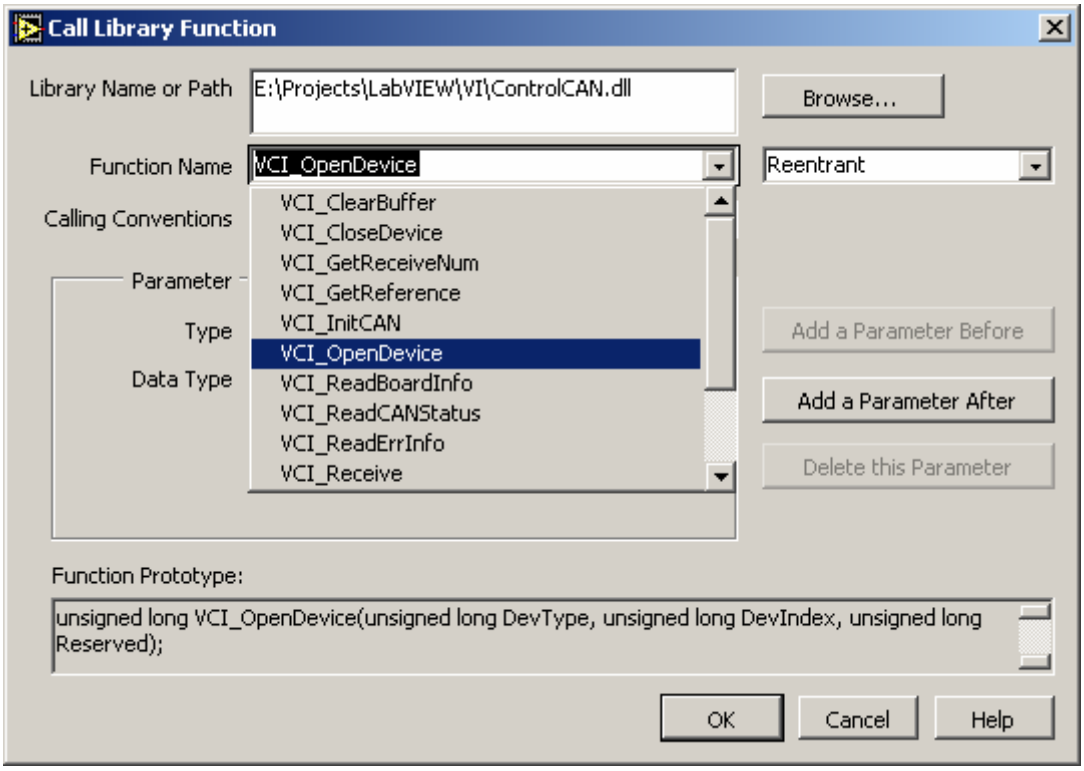

图 2.9 调用库函数 2

在 Calling Conventions 下拉列表框中选中 stdcall(WINAPI)，因为 VCI 库函数使用的是 stdcall 调用约定。

Parameter 框中的 return type 不变。Type 框中选 Numeric。Data Type 框中选 Unsigned 32-bit Interger。即指定返回 32 位整形数。

单击 Add a Parameter After 按钮，Parameter 框中的选项变为图 2.10 所示。将缺省值 agr1 改为 DevType，因为在 VCI_OpenDevice 函数声明中定义了参数 DevType。

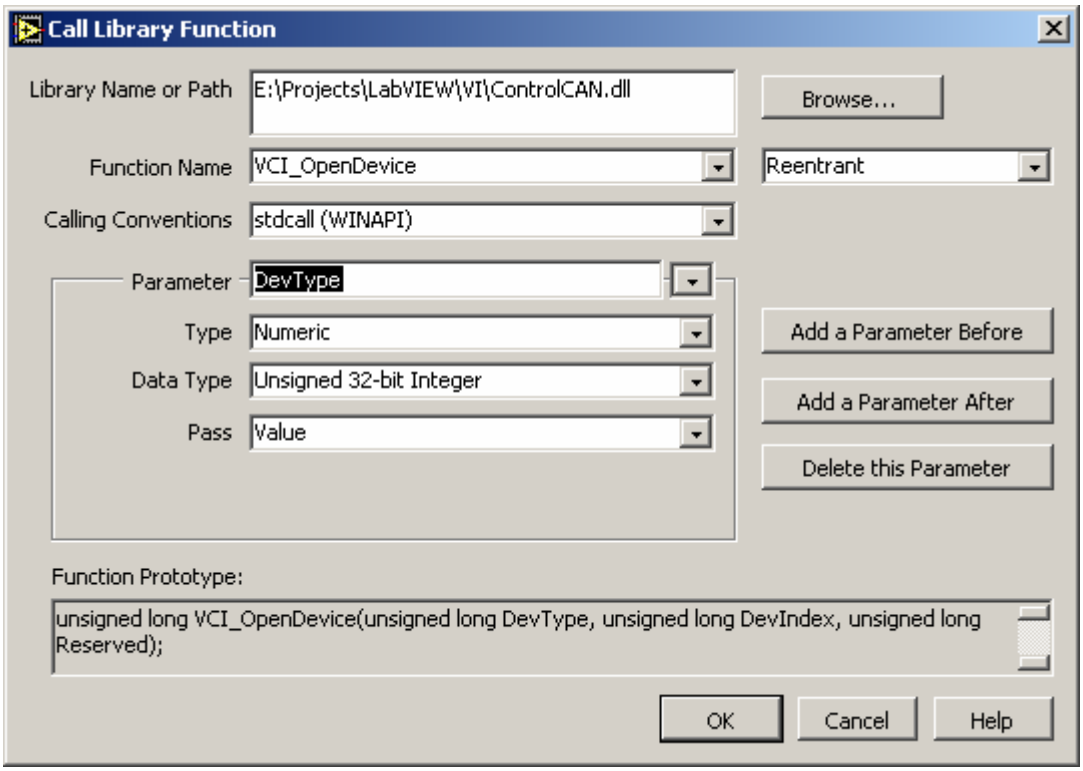

图 2.10 调用库函数 3

Type 框中选 Numeric，并在 Data Type 框中选 Unsigned 32-bit Integer。表示将编程时指定的 LabVIEW 数据类型为 32 位无符号整型。Pass 框中选择 Value。

同样，按以上步骤添加 DevIndex、Reserved 参数。

单击 OK 按钮退出这个对话框。调用库函数节点变为图 2.11 所示中的情况。图中的参数端口由上到下分别为 return type、DevType、DevIndex 和 Reserved。每个端口均有一个输入端和一个输出端，左边的端口为输入端，右边的端口位输出端。因为第一个端口是函数的返回值，所以没有输入端，在图中可以看到其输入端为填充的蓝色。其他端口则是函数的参数，如果参数的类型是指针的话，可以通过参数的输出端输出数据。在其他情况下，不需要使用参数的输出端。

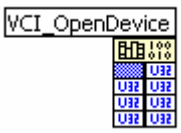

图 2.11 调用库函数程序框图

2.3 应用示例

应用示例 Demo 演示了在 LabVIEW7.0 下如何使用 VCI 库函数。其界面如图 2.12 所示。

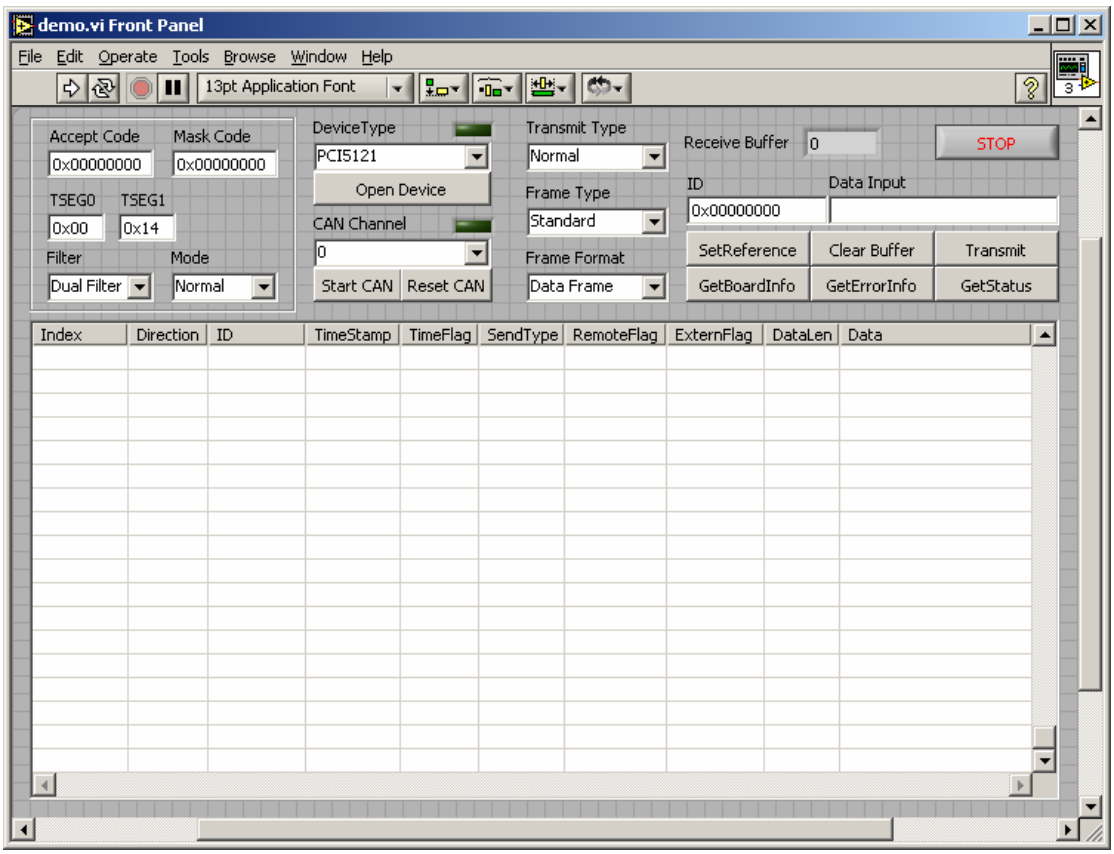

图 2.12 demo 界面

在 Demo 中实现了数据的收发，并将在 CAN 总线上收发的数据在列表示图中显示。Demo 程序当中有 3 个主要的 While 循环：主循环、发送数据循环和接收数据循环。这三个循环是并行运行的。其中，主循环处理与用户交互的界面，并通过用户事件 TREvent 与发送数据循环和接收数据循环通信。

在主循环中使用事件驱动机制处理用户在前面板的操作。打开设备的程序框图如图 2.13 所示。在图中调用 VCI_OpenDevice 函数打开设备，如果打开成功，则调用 VCI_InitCAN 函数初始化设备，成功的话就处理一下前面板控件的状态。

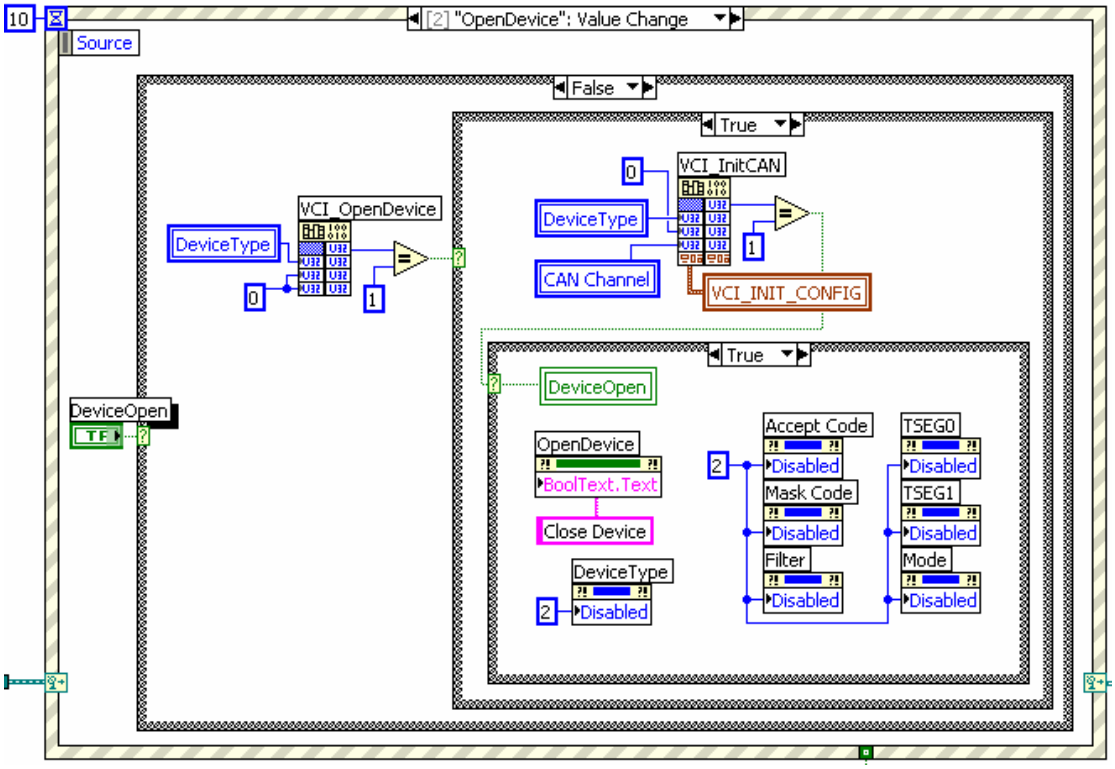

图 2.13 打开设备框图

图 2.14 所示是接收数据的程序框图。接收数据的过程是在一个 while 循环中，这个循环在程序已开始时就一直运行，直到前面板上的 stop 按钮被按下并在其 Value Change 事件中使 stop2 的值变为 False 时才停止。在循环当中，只有设备已经启动时才会进行读操作。在 Demo 中 VCI_Reveive 函数一次只读取一帧，输出的数据保存到 VCI_CAN_OBJ_R 结构中，如果 VCI_Reveive 函数执行成功的话，就把接收到的数据通过事件传递给主循环处理。

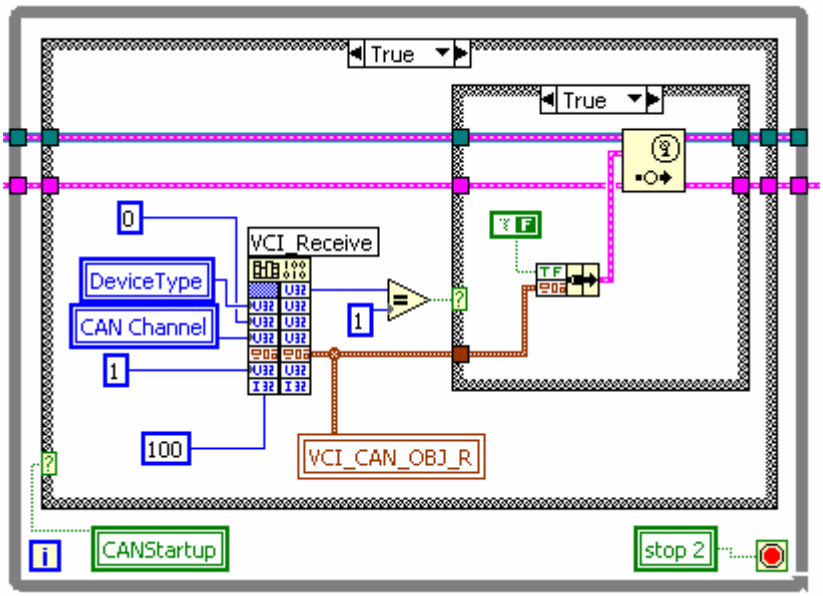

图 2.14 接收数据

发送数据的过程与接收数据的过程相似。当前面板上的 Transmit 按钮被按下时，才会

把 VCI_CAN_OBJ_T 结构中的数据通过 VCI_Transmit 函数发送到 CAN 总线上。发送成功后，生成一个 TREvent 事件，并通过这个事件把 VCI_CAN_OBJ_T 结构的内容传递给主循环显示。

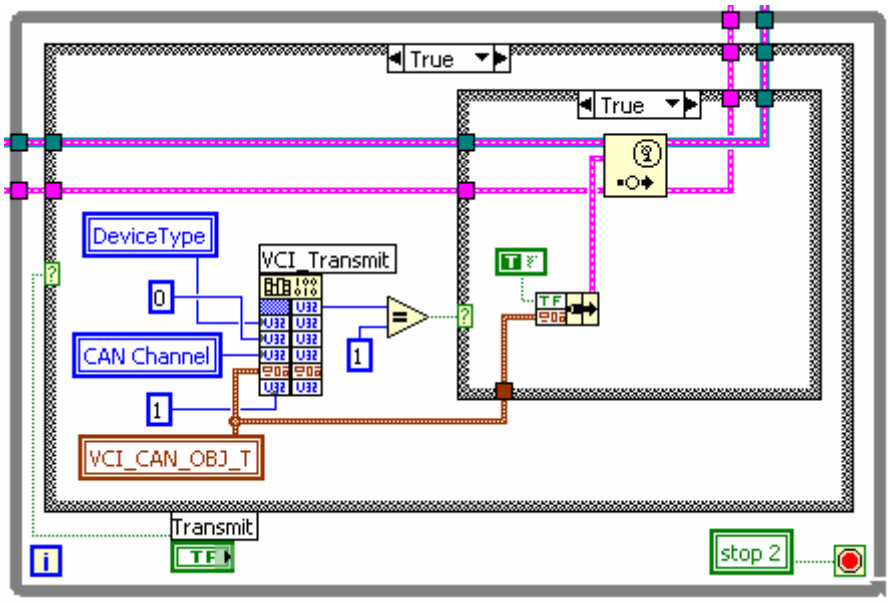

图 2.15 发送数据

第3章 VCI 函数调用参考

在 LabVIEW 中使用 Calling Library Function Node 调用 VCI 库函数的配置如下各表所示。

3.1 VCI_OpenDevice

Function name	VCI_OpenDevice				
Prototype	DWORD __stdcall VCI_OpenDevice(DWORD DeviceType, DWORD DeviceIndex, DWORD Reserved);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value

3.2 VCI_CloseDevice

Function name	VCI_CloseDevice				
Prototype	DWORD __stdcall VCI_CloseDevice(DWORD DeviceType, DWORD DeviceIndex);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value

3.3 VCI_InitCan

Function name	VCI_InitCan
Prototype	DWORD __stdcall VCI_InitCan(DWORD DeviceType, DWORD DeviceIndex, DWORD CANIndex, PVCI_INIT_CONFIG pInitConfig);
Calling Conventions	stdcall(WINAPI)
Reentrant	

		Type	Data Type	Data Format	Pass
Parameter	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value
	pInitConfig	Adapt to Type		Handle by Value	

3.4 VCI_ReadBoardInfo

Function name	VCI_ReadBoardInfo				
Prototype	DWORD __stdcall VCI_ReadBoardInfo(DWORD DeviceType, DWORD DeviceIndex, PVCI_BOARD_INFO pInfo);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	pInfo	Adapt to Type		Handle by Value	

3.5 VCI_ReadErrInfo

Function name	VCI_ReadErrInfo				
Prototype	DWORD __stdcall VCI_ReadErrInfo(DWORD DeviceType, DWORD DeviceIndex, DWORD CANIndex, PVCI_ERR_INFO pErrInfo);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value

	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value
	pErrInfo	Adapt to Type		Handle by Value	

3.6 VCI_ReadCanStatus

Function name	VCI_ReadCanStatus				
Prototype	DWORD __stdcall VCI_ReadCanStatus(DWORD VCI_ReadCanStatus, DWORD DeviceIndex, DWORD CANIndex, PVCI_CAN_STATUS pCANStatus);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value
	pCANStatus	Adapt to Type		Handle by Value	

3.7 VCI_GetReference

Function name	VCI_GetReference				
Prototype	DWORD __stdcall VCI_GetReference(DWORD DeviceType, DWORD DevIndex, DWORD DeviceIndex, DWORD RefType, PVOID pData);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	RefType	Numeric	Unsigned 32-bit Integer		Value

	pData	Adapt to Type		Handle by Value	

3.8 VCI_SetReference

Function name	VCI_SetReference				
Prototype	DWORD __stdcall VCI_SetReference(DWORD DeviceType, DWORD DeviceIndex, DWORD CANIndex, DWORD RefType, PVOID pData);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value
	RefType	Numeric	Unsigned 32-bit Integer		Value
	pData	Adapt to Type		Handle by Value	

3.9 VCI_GetReceiveNum

Function name	VCI_GetReceiveNum				
Prototype	ULONG __stdcall VCI_GetReceiveNum(DWORD DeviceType, DWORD DeviceIndex, DWORD CANIndex);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value

3.10 VCI_ClearBuffer

Function name	VCI_ClearBuffer				
Prototype	DWORD __stdcall VCI_ClearBuffer(DWORD DeviceType, DWORD DeviceIndex, DWORD CANIndex);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value

3.11 VCI_StartCAN

Function name	VCI_StartCAN				
Prototype	DWORD __stdcall VCI_StartCAN(DWORD DeviceType, DWORD DeviceIndex, DWORD CANIndex);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value

3.12 VCI_ResetCAN

Function name	VCI_ResetCAN
Prototype	DWORD __stdcall VCI_ResetCAN(DWORD DeviceType, DWORD DeviceIndex, DWORD CANIndex);
Calling Conventions	stdcall(WINAPI)
Reentrant	

		Type	Data Type	Data Format	Pass
Parameter	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value

3.13 VCI_Transmit

Function name	VCI_Transmit				
Prototype	ULONG __stdcall VCI_Transmit(DWORD DeviceType, DWORD DeviceIndex, DWORD CANIndex, PVCI_CAN_OBJ pSend, ULONG Length);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value
	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value
	pSend	Adapt to Type		Handle by Value	
	Length	Numeric	Unsigned 32-bit Integer		Value

3.14 VCI_Receive

Function name	VCI_Receive				
Prototype	ULONG __stdcall VCI_Receive(DWORD DeviceType, DWORD DeviceIndex, DWORD CANIndex, PVCI_CAN_OBJ pReceive, ULONG Length, INT WaitTime=－1);				
Calling Conventions	stdcall(WINAPI)				
Reentrant					
Parameter		**Type**	**Data Type**	**Data Format**	**Pass**
	return type	Numeric	Unsigned 32-bit Integer		Value

	DeviceType	Numeric	Unsigned 32-bit Integer		Value
	DeviceIndex	Numeric	Unsigned 32-bit Integer		Value
	CANIndex	Numeric	Unsigned 32-bit Integer		Value
	pReceive	Adapt to Type		Handle by Value	
	Length	Numeric	Unsigned 32-bit Integer		Value
	WaitTime	Numeric	Signed 32-bit Integer		Value

ZLGCAN
2005 年 3 月