2011

CAN-bus 接口函数库

（LINUX）

使用手册 V1.0

--北京爱泰电子出品

目　录

1. 驱动程序的安装

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）。

2. 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。

3. 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

4. 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
USBCAN1	3
USBCAN2	4

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存储接口卡信息的数据结构

```
typedef    struct_VCI_BOARD_INFO{
            USHORT    hw_Version；
            USHORT    fw_Version；
            USHORT    dr_Version；
            USHORT    in_Version；
            USHORT    irq_Num；
            USHORT    can_Num；
            USHORT    str_Serial_Num[20]；
            USHORT    str_hw_Type[40]；
            USHORT    Reserved[4]；
} VCI_BOARD_INFO，*PVCI_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，*PVCI_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USBCAN 时，索引号为0，有两个时可以为0 或1。

Reserved：

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USBCAN 时，索引号为0，有两个时可以为0 或1。

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USBCAN 时，索引号为0，有两个时可以为0 或1。

CANIndex：

第几路CAN。

pInitConfig：

初始化参数结构。

函数功能：此函数用以初始化指定的CAN。

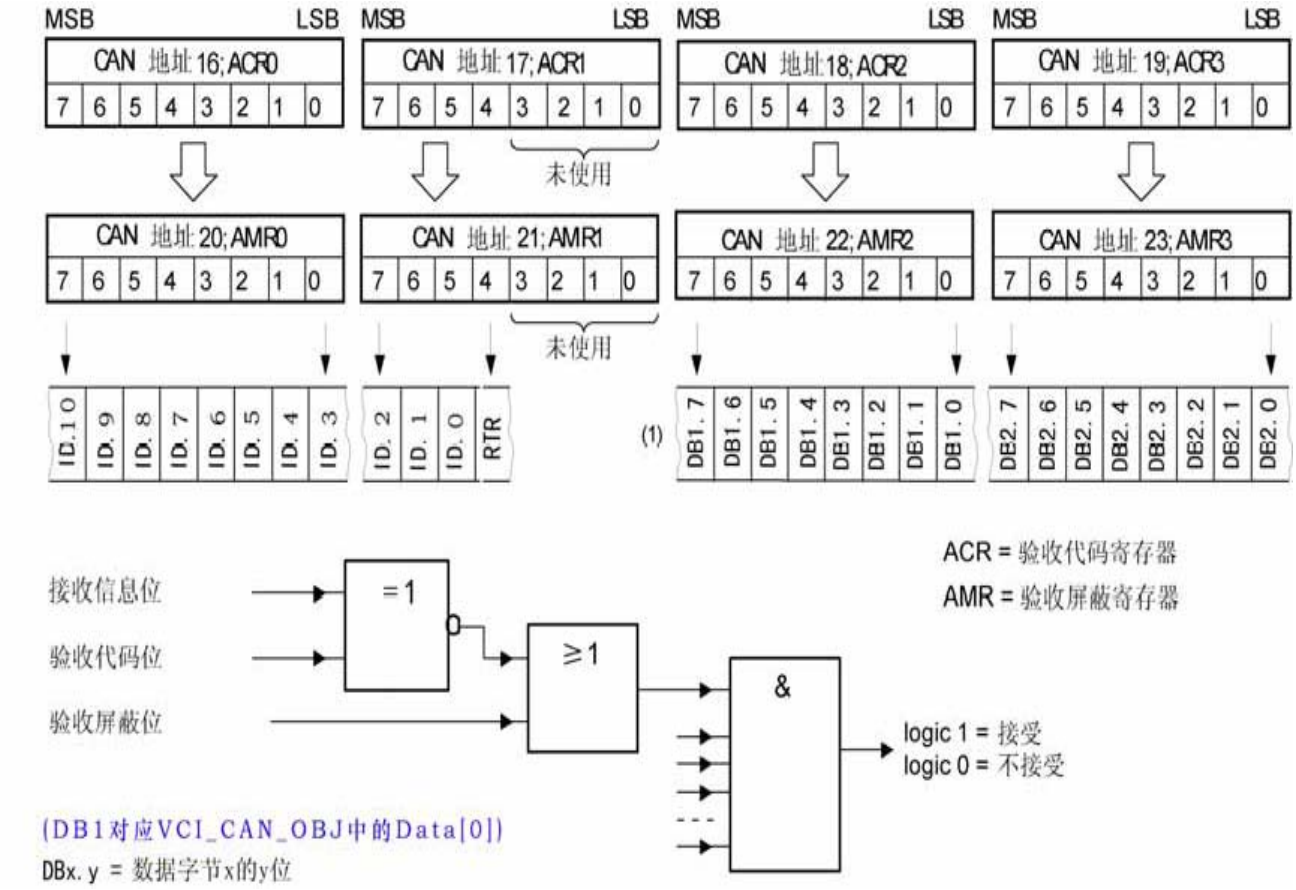
返回值：为1 表示操作成功，0 表示操作失败。

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
pInitConfig->AccMask	

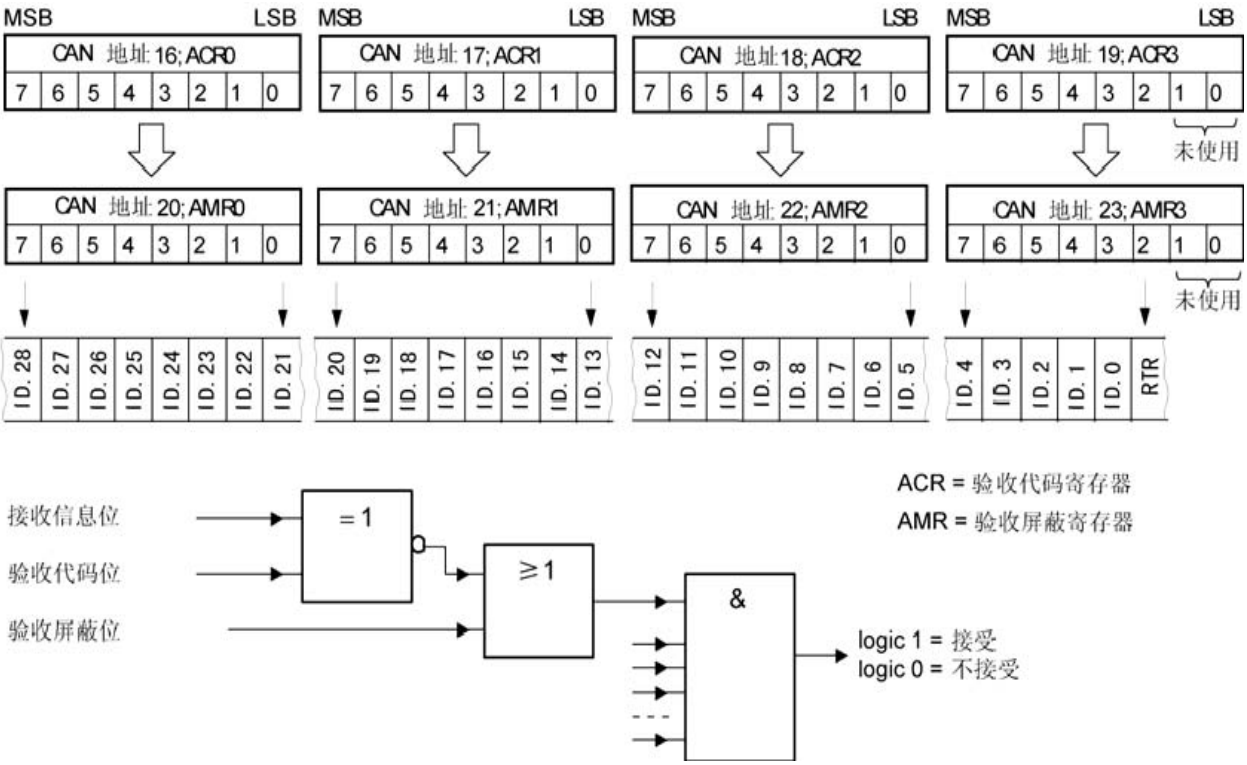
	器AMR0，AMR1，AMR2，AMR3，其中高字节对应AMR0，低字节对应AMR3。（请看表后说明）
pInitConfig->Reserved	保留
pInitConfig->Filter	滤波方式，1 表示单滤波，0 表示双滤波
pInitConfig->Timing0	定时器0
pInitConfig->Timing1	定时器1
pInitConfig->Mode	模式，0 表示正常模式，1 表示只听模式

当滤波方式为单滤波，接收帧为标准帧时：

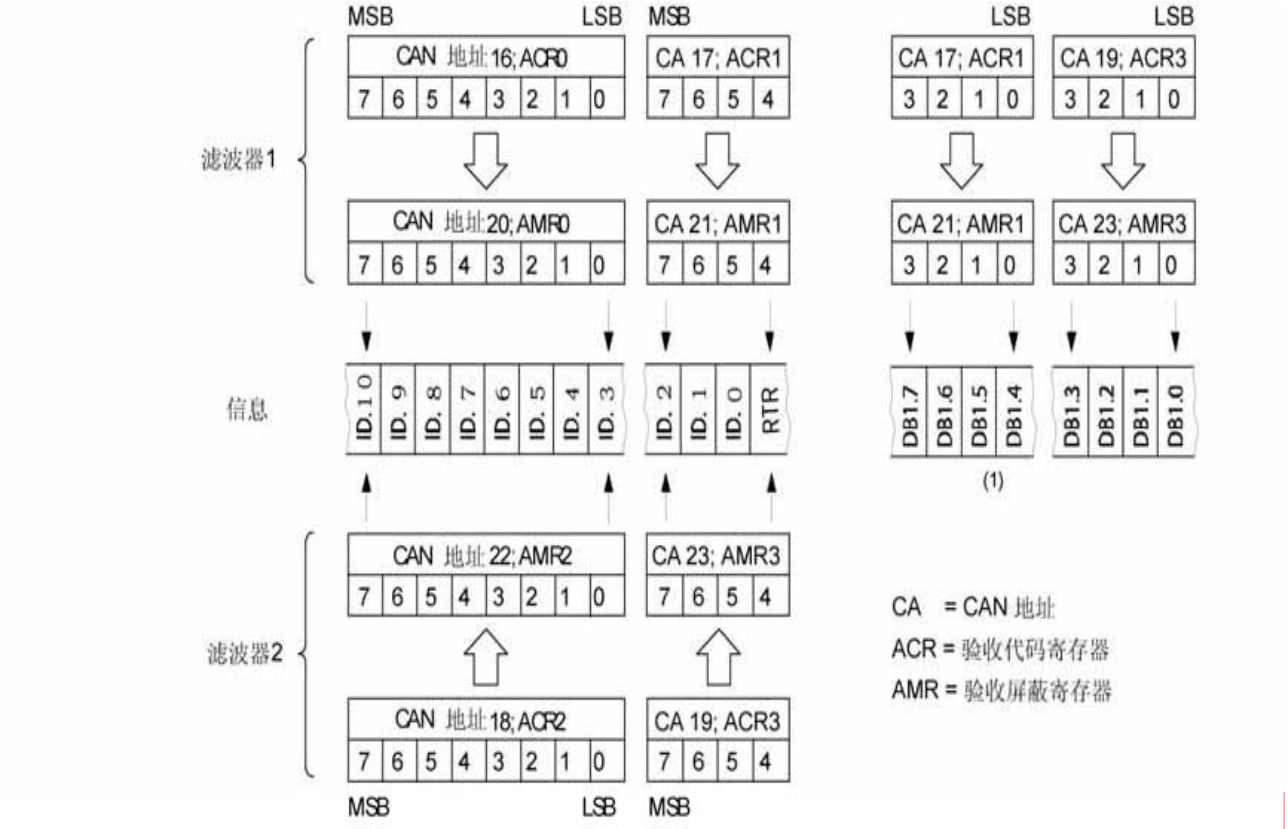


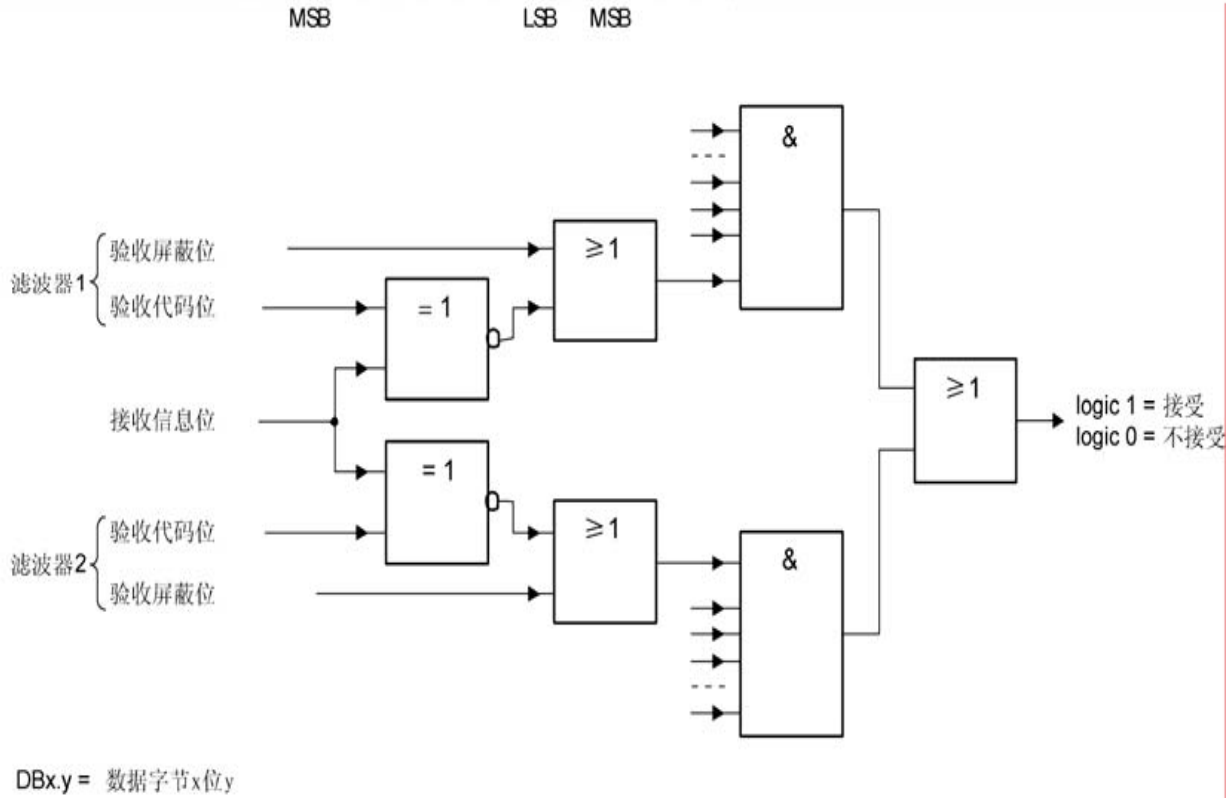
RTR 对应VCI_CAN_OBJ 中的RemoteFlag

当滤波方式为单滤波，接收帧为扩展帧时：

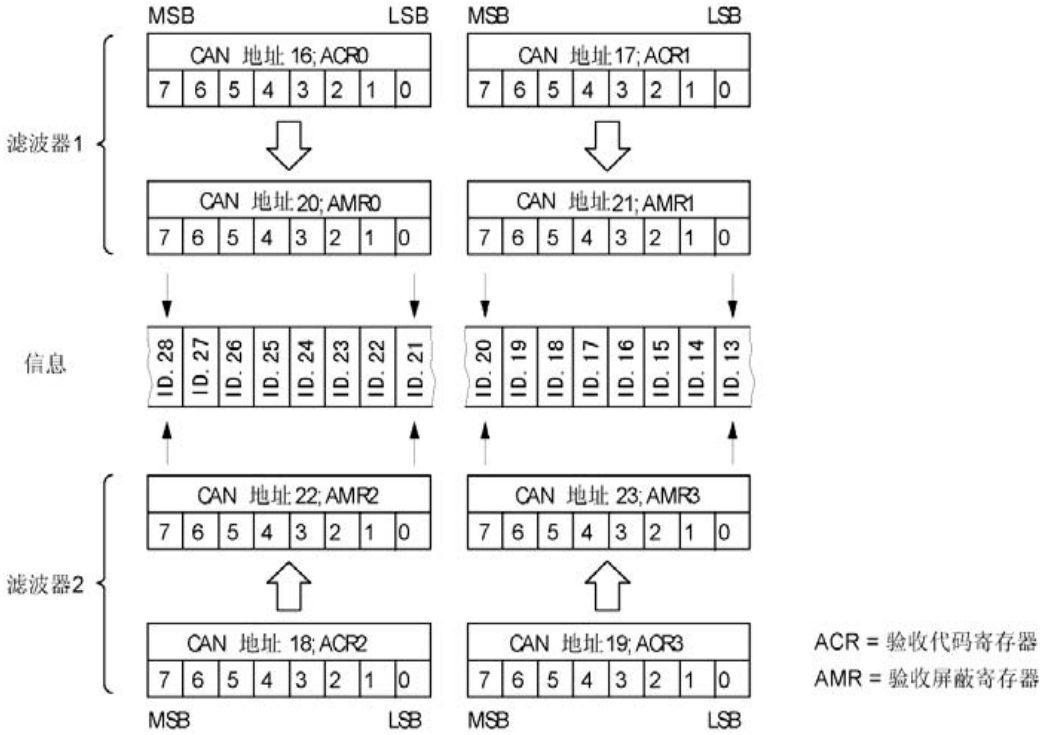


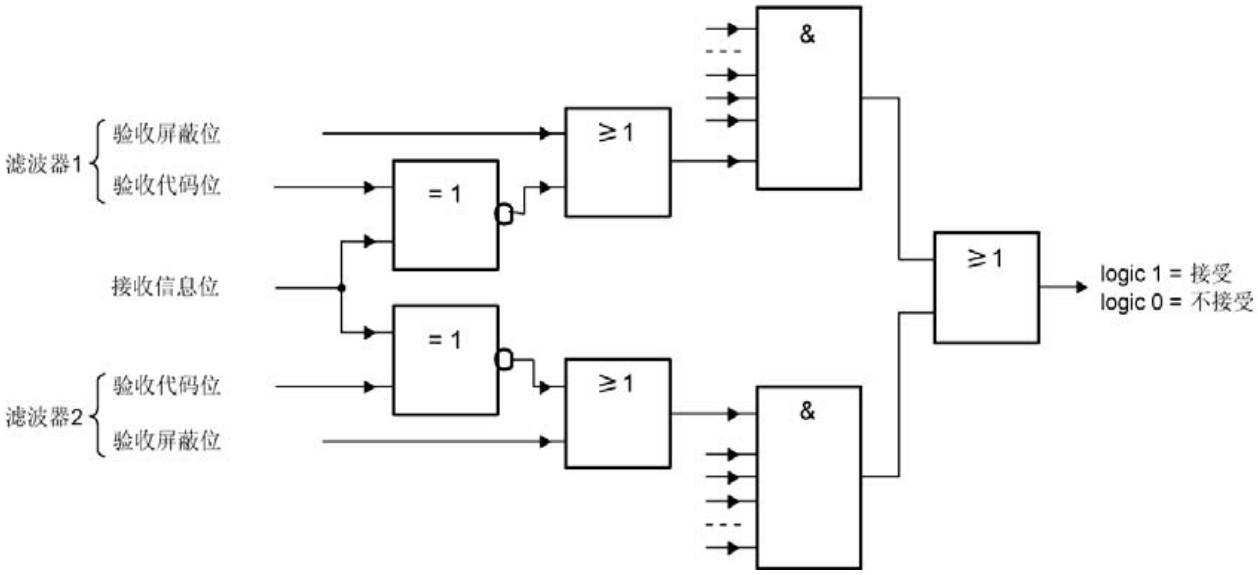
当滤波方式为双滤波，接收帧为标准帧时：





当滤波方式为双滤波，接收帧为扩展帧时：





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USBCAN 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USBCAN 时，索引号为0，有两个时可以为0 或1。

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。

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->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

4.4.6 DWORD __stdcall VCI_ReadCanStatus(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI_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USBCAN 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USBCAN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。

函数功能：此函数用以获取设备的相应参数。

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**USBCAN1**，**USBCAN2** 时：

RefType	pData	功能描述

	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。 当作为输出参数时，表示为 CAN 控制器的控制寄存器的值。	读CAN 控制器的指定控制寄存器的值， 例如对 USBCAN1： BYTE val=0; VCI_GetReference(VCI_USBCAN1,0,0,1,(PVOID)&val); 如果此函数调用成功,则在val 中返回寄存器的值。

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USBCAN 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。

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。

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。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**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。	CAN 控制器的指定控制寄存器

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USBCAN 时，索引号为0，有两个时可以为0 或1。

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USBCAN 时，索引号为0，有两个时可以为0 或1。

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USBCAN 时，索引号为0，有两个时可以为0 或1。

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USBCAN 时，索引号为0，有两个时可以为0 或1。

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USBCAN 时，索引号为0，有两个时可以为0 或1。

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, PVCI_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USBCAN 时，索引号为0，有两个时可以为0 或1。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