Kvaser CANLIB API
杨勇 2010 年 11 月 29 日 21:24:36

Kvaser CANlib 是 Kvaser 公司的 CAN 测试设备提供的 API 函数，通过使用这些 API 可以方便的编写自己的 CAN 测试工具，对 CAN 网络的开发测试有很大的帮助。

Kvaser CANLIB 主要以下几类 API 函数：

1、信息服务 Information Services

信息服务类的 API 函数，主要用于查看 CAN 设备通道信息，驱动版本信息，CANLIB 库信息，硬件时钟信息等。使用这部分函数，可以有效的控制设备，及时发现设备和驱动等不兼容信息。

函数	功能
canGetChannelData	该函数用于接收指定 CAN 通道指定类别信息
canGetErrorText	该函数用于解释一个错误代码信息
canGetNumberOfChannels	该函数用于获取计算机可用的 CAN 通道数量，不是通道句柄
canGetVersion	该函数用于获取 CANLIB32.DLL 的版本信息
canGetVersionEx	该函数用于获取 Kvaser CAN 设备驱动程序的版本信息
canProbeVersion	该函数用于检测指定版本的 CANLIB 是否在系统上安装
canReadErrorCounters	该函数用于读取 CAN 通道上的错误计数器（接收错误、发送错误、错误溢出）内容
canReadStatus	该函数用于读取指定 CAN 通道的状态，返回特定特定参数组合
canReadTimer	该函数用于读取指定 CAN 通道的硬件时钟
canTranslateBaud	

2、通道开启和关闭 Channel Open and Close

通道开启和关闭类 API 函数，主要用于初始化 CANLIB、释放 CANLIB，开启和关闭 CAN 通道，使总线开启、关闭、复位等。这部分函数是 CANLIB 的核心函数，几乎任意一个 Kvaser 程序都会用到。

函数	功能
canBusOff	使指定 CAN 通道总线关闭
canBusOn	使指定 CAN 通道总线激活
canClose	关闭与处理相关的 CAN 通道
canInitializeLibrary	初始化驱动程序，必须在其他函数之前使用
canOpenChannel	开启一个 CAN 通道，并返回一个句柄供后续调用
canResetBus	使 CAN 控制器重启
canUnloadLibrary	如果动态调用 CANLIB32.DLL，则使用该函数进行释放

3、通道参数 Channel Parameters

通道参数类 API 主要用于设置或获取 CAN 通道的一些参数，如设定 CAN 滤波器参数、工作模式、帧格式、波特率、采样点等等。也是经常会用到的函数之一。

函数	功能
canAccept	用于设定指定通道的验收滤波器
canSetAcceptanceFilter	功能同上，但更好用。
canGetBusOutputControl	该函数用于读取 CAN 控制器类型
canGetBusParams	获取指定通道的参数
canGetDriverMode	获取指定通道的工作模式
canSetBusOutputControl	设定 CAN 控制器驱动类型
canSetBusParams	设定指定 CAN 通道的相关参数
canSetBusParamsC200	
canSetDriverMode	设定指定 CAN 通道的工作模式

4、接收消息 Receiving Messages

接收消息类 API 函数主要用于接收 CAN 通道中接受缓冲区的 CAN 消息。使用不同的接受函数，可以实现不同的接收方式。

函数	功能
canRead	从接收缓冲区接收一个 CAN 消息
canReadSpecific	从接收缓冲区中读取一个指定 ID 的 CAN 消息
canReadSpecificSkip	从接收缓冲区中读取一个指定 ID 的 CAN 消息，将 ID 不匹配的消息从缓冲区中删除
canReadSync	等待直到接收缓冲区接收到至少一条消息或等待超时
canReadSyncSpecific	等待直到接收缓冲区中包含至少对应 ID 的消息或等待超时
canReadWait	从接收缓冲区接收一条消息，若缓冲区没有消息，则等待直到至少有一条消息或等待超时

5、发送消息 Sending Messages

发送消息类 API 函数主要用于往 CAN 通道中的发送缓冲区发送消息

函数	功能
canWrite	发送一条 CAN 消息
canWriteSync	等待发送缓冲区中说有消息发送完毕或等待超时
canWriteWait	发送一条消息，若消息发送成功或超时则返回

6、通知和等待 Notification and Waiting

该来函数主要用于往通道上发送通知，或等待通道事件发生。

函数	功能
canSetNotify	
canWaitForEvent	等待指定 CAN 通道上有一个事件发生，如收到 CAN 消息或模式改变。

7、杂项 Miscellaneous

该类函数主要用于清空发送或接收缓冲区。

函数	功能
canIoCtl	执行一些特定功能
canFlushReceiveQueue	清空接收缓冲区
canFlushTransmitQueue	清空发送缓冲区

8、常用函数说明

下边对一些常用到的函数进行详细说明：

1、canInitializeLibrary

a) 函数原型：void **canInitializeLibrary**(void);

2、canOpenChannel

a) 函数原型：int **canOpenChannel**(unsigned int channelNumber,unsigned int flags);

b) 参数说明：

i. channelNumber 通道编号，该编号需要硬件支持。第一个通道始终为数字 0

ii. Flags 通道开启标志位，用于设定开启模式，如下表所示：

canOPEN_EXCLUSIVE	独占方式
canOPEN_ACCEPT_VIRTUAL	打开已经以独占方式打开的通道
canOPEN_OVERRIDE_EXCLUSIVE	允许虚拟通道和物理通道同时开放

iii. 返回句柄或错误代码

3、canClose

a) 函数原型：canStatus **canClose**(int handle);
b) 参数说明　handle　需要关闭的通道句柄
c) 返回值　　关闭成功或错误代码

4、canSetBusParams

a) 函数原型：canStatus **canSetBusParams**(int handle,long freq,unsigned int tseg1,unsigned int tseg2,unsigned int sjw, unsigned int noSamp,unsigned int syncmode);
b) 参数说明
   i. Handle　待设定的通道的句柄
   ii. freq　速率
   iii. tseg1 同步采样点 1
   iv. tseg2 同步采样点 2
   v. sjw　采样点宽度 1,2,3, or 4.
   vi. noSamp 采样点数目 1or3
   vii. Syncmode 忽视
c) 返回值　　成功或错误代码

Constant	Bit rate	Sampling point	SJW	Quanta per bit
canBITRATE_1M *synonym:*BAUD_1M	1 Mbit/s	62.5%	1	8
canBITRATE_500K *synonym:*BAUD_500K	500 kbit/s	62.5%	1	8
canBITRATE_250K *synonym:*BAUD_250K	250 kbit/s	62.5%	1	8
canBITRATE_125K *synonym:*BAUD_125K	125 kbit/s	68.7%	1	16
canBITRATE_100K *synonym:*BAUD_100K	100 kbit/s	68.7%	1	16
canBITRATE_83K *synonym:*BAUD_83K	83.333 kbit/s	75%	2	8
canBITRATE_62K *synonym:*BAUD_62K	62.5 kbit/s	68.7%	1	16
canBITRATE_50K *synonym:*BAUD_50K	50 kbit/s	68.7%	1	16
canBITRATE_10K	10 kbit/s	75%	1	16

5、canBusOn

a) 函数原型：canStatus **canBusOn**(int handle);
b) 参数说明　handle　通道句柄
c) 返回值　　成功或错误代码

6、canBusOff

a) 函数原型：canStatus **canBusOff**(int handle);
b) 参数说明　handle　通道句柄
c) 返回值　　成功或错误代码

7、canRead

a) 函数原型：canStatus **canRead**(int handle, long* id, void* msg, unsigned int* dlc, unsigned int* flag, DWORD* time);

b) 参数说明

i. handle 通道句柄

ii. *id 接收到的 id

iii. *msg 接收到的数据

iv. *dlc 结束到的数据长度

v. *flag 接收数据标志

Flag	Value	Meaning
canMSG_RTR	**1**	Message is a remote request
canMSG_STD	**2**	Message has a standard (11-bit) identifier
canMSG_EXT	**4**	Message has a extended (29-bit) identifier
canMSG_WAKEUP	**8**	Message is a WAKEUP message (SWC hardware.)
canMSG_ERROR_FRAME	**32**	Message represents an error frame.
canMSG_NERR	**16**	NERR was active during the message (TJA1054 etc. hardware. See Note 4 below.)
canMSG_TXACK	**64**	Message is a TX ACK (meaning that the message was really sent)
canMSG_TXRQ	**128**	Message is a TX REQ (meaning that the message was transferred to the CAN controller)
canMSGERR_HW_OVERRUN	**512**	Hardware buffer overrun.
canMSGERR_SW_OVERRUN	**1024**	Software buffer overrun.

*time 接收到的时间戳

c) 返回值 成功或错误代码

8、canReadWait

a) 函数原型：canStatus **canReadWait**(int handle, long* id,void* msg, unsigned int* dlc, unsigned int* flag, DWORD* time,DWORD timeout);

b) 参数说明 timeout 等待时间，单位为毫秒，其他同上

c) 返回值 成功或错误代码

9、canWrite

a) 函数原型：canStatus **canWrite**(int handle, long id,void* msg, unsigned int dlc, unsigned int flag);

b) 参数说明 同上

c) 返回值 成功或错误代码

10、canWriteSync

a) 函数原型：canStatus **canWriteSync**(int handle, unsigned long timeout);

b) 参数说明 handle 需要关闭的通道句柄 timeout 等待的时间

c) 返回值 成功或错误代码

9、使用说明

1、使用 Kvaser CANLIB 时，在文件开头，必须包含 #include <canlib.h>

2、简单操作流程

- **a) 初始化驱动程序**
- **b) 读 CAN 通道数目，获取通道句柄**
- **c) 选择开启指定通道**
- **d) 设定通道参数**
- **e) 设定通道工作类型**
- **f) 打开总线**
- **g) 读 CAN 消息或写 CAN 消息**
- **h) 关闭总线**
- **i) 关闭通道**

例如：

```
#include <canlib.h>
#include <stdio.h>

void main(void)
{
  canHandle h;

  canInitializeLibrary();
  h = canOpenChannel(0, canWANT_EXCLUSIVE);
  if (h != canOK) {
    char msg[64];
    canGetErrorText((canStatus)h, msg, sizeof(msg));
    fprintf(stderr, "canOpenChannel failed (%s)\n", msg);
    exit(1);
  }
  canSetBusParams(h, BAUD_250K, 0, 0, 0, 0, 0);
  canSetBusOutputControl(h, canDRIVER_NORMAL);
  canBusOn(h);
  canWrite(h, 123, "HELLO!", 6, 0);
  canWriteSync(h, 500);
  canBusOff(h);
  canClose(h);
}
```

10、附录

```
//存在以下几类函数：
//发送消息类函数
//接收消息类函数
//查看驱动版本类函数
//CAN 通道控制函数

#include <canlib.h>

canStatus canAccept(int handle, long envelope, int flag);
canStatus canSetAcceptanceFilter(int handle,
                                 unsigned int code,
                                 unsigned int mask,
                                 int is_extended);
//以上两个函数用于设置 CAN 接收滤波器的。有两种形式进行设置，验收屏蔽（MASK）和验收代码（CODE）。
//验收屏蔽用于设置某一位的屏蔽与验收代码有关（1）或无关（0）。
//验收代码用于设置某一位的屏蔽，通过验收屏蔽设置有关的地址对应位与目标地址对应位必须相同。
//使用 canAccept 函数，需要设置两次，第一次先设置验收屏蔽位，第二次设置代码位。
//使用 canSetAcceptanceFilter 则只需设置一次，验收屏蔽盒验收代码同时设置。其参数 is_extended 表示帧格式，1 为标准帧，0 为扩展帧。

canStatus canBusOff(int handle);
canStatus canBusOn(int handle);
//以上两个函数用于设置指定的通道（handle）开启或关闭

canStatus canClose(int handle);
//关闭指定通道。如果没有其他线程正在使用的 CAN 电路，它将关闭总线。该函数用于 canBusOff 之后

canStatus canFlushReceiveQueue(int handle);
//该函数将移除通道中接收通道中的所有接收到的信息
//这个函数与 canIoCtl 函数的 canIOCTL_FLUSH_RX_BUFFER 参数功能相同
canStatus canFlushTransmitQueue(int handle);
//清除发送缓冲区中的信息
//功能和 canIoCtl 函数的 canIOCTL_FLUSH_TX_BUFFER 参数功能相同

canStatus canGetBusOutputControl(
    const int hnd,
    unsigned int * drivertype
);
canStatus canSetBusOutputControl(int handle, unsigned int drivertype);
```

//设定指定 CAN 通道的模式，其中有正常模式（canDRIVER_NORMAL）和静默模式（canDRIVER_SILENT）两种

```
canStatus canGetBusParams(
    int handle,
    long* freq,
    unsigned int* tseg1,
    unsigned int* tseg2,
    unsigned int* sjw,
    unsigned int* noSamp,
    unsigned int* syncmode
);
```

//这个函数取回指定 CAN 通道的相关参数

```
/*
Input Parameters
handle
An open handle to a CAN controller.
Output Parameters
freq
Bit rate (bits per second).
tseg1
Time segment 1, that is, the number of quanta from (but not including) the Sync Segment to the sampling point.
tseg2
Time segment 2, that is, the number of quanta from the sampling point to the end of the bit.
sjw
The Synchronization Jump Width; can be 1,2,3, or 4.
noSamp
The number of sampling points; can be 1 or 3.
syncmode
Unsupported, always read as zero.
*/
```

```
canStatus CANLIBAPI canGetBusStatistics(int hnd, canBusStatistics *stat, size_t bufsiz);
```

//该函数用于接收最近一次的统计信息，必须使用函数 canRequestBusStatistics 从设备请求统计信息。

//bufsiz　信息宽度

//stat　　统计信息存储指针

```
canStatus canGetChannelData(int channel, int item,
                            void *buffer, size_t bufsize);
```

//该函数用于检索某通道的相关信息

channel

The number of the channel you are interested in. Channel numbers are integers in the interval beginning at 0 (zero) and ending at the value returned by canGetNumberOfChannels minus 1. Note: you must pass a channel number and not a channel handle.
想要读取信息的通道。必须是通道号，而不是通道句柄。通道号可以通过 canGetNumberOfChannels 函数返回的值减 1.
item
此参数指定那些通道获取那些参数，有特定的参数共选择
This parameter specifies what data to obtain for the specified channel. The value is one of the constants canCHANNELDATA_xxx; see below.
buffer
The address of a buffer which is to receive the data.
接收数据缓冲区的地址
bufsize
The size of the buffer to which the buffer parameter points.
接收数据缓冲区的大小
*/

```
canStatus canGetDriverMode(
    const int hnd,
    int *lineMode,
    int *resNet);
);
//指定设备模式     sleep, wakeup, standby, fast mode, etc

canStatus canGetErrorText(canStatus err, char* buf, int bufsiz);
//翻译错误代码

canStatus canGetNumberOfChannels(int * channelCount);
//这个函数返回计算机可用的 CAN 通道数
//参数为通道计数器

unsigned short canGetVersion(void);
//这个函数返回 CANLIB32.DLL 文件的版本号

unsigned int canGetVersionEx(unsigned int itemCode);
//这个函数返回从驱动程序的各种版本号。

void canInitializeLibrary(void);
//设备初始化函数
//必须在其他函数之前调用，可以调用多次，但只初始化一次
//如果在初始化过程中，出现错误，则进入静默状态，并在调用 canOpenChannel 函数后，返回错误代码

canStatus canIoCtl(int handle,
```

```
    unsigned int func,
    void* buf,
    unsigned int buflen);
//处理特别事件
//有详细的参数说明

int canOpenChannel(
    unsigned int channelNumber,
    unsigned int flags);
//打开一个通道，返回一个通道句柄共随后的函数使用
//channelNumber
//通道的数量。通道的数量依赖于特定的硬件
//优先使用该参数 canOPEN_EXCLUSIVE（独占），或使用

BOOL canProbeVersion(
    int hnd,
    int major,
    int minor,
    int oem_id,
    unsigned int flags);
//检查是否是指定驱动版本

canStatus canRead(
    int handle, long* id,
    void* msg, unsigned int* dlc,
    unsigned int* flag, DWORD* time);
//从接受缓冲区中接收一个信息，如果接收缓冲区没有信息，则立即返回错误代码
canERR_NOMSG
//time 指向一个缓冲区，接收消息的时间戳

canStatus canReadErrorCounters(
    int handle,
    unsigned int* txErr,
    unsigned int* rxErr,
    unsigned int* ovErr);
//从 CAN 控制器中读错误计数器的值，是最近一次的计数值

canStatus canReadSpecific(
    int handle, long id,
    void* msg, unsigned int* dlc,
    unsigned int* flag, unsigned long* time);
//从接收缓冲区中接收一个明确 ID 的消息，任何不和该 ID 不匹配的消息，继续保持在接受
缓冲区中。若没有消息，则立即返回错误信息
```

canStatus canReadSpecificSkip(
int handle, long id,
void* msg, unsigned int* dlc,
unsigned int* flag, unsigned long* time);
//从接收缓冲区中接收一个明确 ID 的消息，任何不和该 ID 不匹配的消息，将一处接收缓冲区。若没有消息，则立即返回错误信息

canStatus canReadStatus(int handle, unsigned long* flags);
//读状态，有特定的状态指示：
/*
Flag Value Meaning
canSTAT_ERROR_PASSIVE 0x1 The circuit is "error passive".
canSTAT_BUS_OFF 0x2 The circuit is OffBus.
canSTAT_ERROR_WARNING 0x4 At least one error counter > 96.
canSTAT_ERROR_ACTIVE 0x8 The circuit is "error active" (this means that both error counters are below 128 and that the circuit will transmit active error frames when errors occur. See the CAN specification for details.
canSTAT_TX_PENDING 0x10 There are messages pending transmission.
canSTAT_RX_PENDING 0x20 There are messages in the receive buffer.
canSTAT_TXERR 0x80 There has been at least one TX error.
canSTAT_RXERR 0x100 There has been at least one RX error of some sort.
canSTAT_HW_OVERRUN 0x200 There has been at least one hardware buffer overflow.
canSTAT_SW_OVERRUN 0x400 There has been at least one software buffer overflow.
canSTAT_OVERRUN 0x600 For convenience. This flag is the logical bit wise or between canSTAT_SW_OVERRUN and canSTAT_HW_OVERRUN.
*/

canStatus canReadSync(int handle, unsigned long timeout);
//等待直到接收缓冲区至少有一个消息或等待超时
/*
handle
A handle to an open circuit.
timeout
The timeout in milliseconds 毫秒
*/

canStatus canReadSyncSpecific(int handle, long id, unsigned long timeout);
//等待消息队列中有指定 ID 的消息，或等待超时

unsigned long canReadTimer(int handle);
//从时钟电路中读取到当前时间戳 毫秒

canStatus canReadWait(
int handle, long* id,

```
        void* msg, unsigned int* dlc,
        unsigned int* flag, DWORD* time,
        DWORD timeout);
```
//从接收缓冲区中读取一个消息，若缓冲区中没有消息，则等待直到有一个消息为止，或者等待超时。

```
canStatus CANLIBAPI canRequestBusStatistics(int hnd);
```
//CAN 统计请求，常与 canGetBusStatistics 连用

```
canStatus CANLIBAPI canRequestChipStatus(int hnd);
```
//CAN 芯片状态请求，然后通过 canReadStatus 读取状态

```
canStatus canResetBus(int handle);
```
//对指定 CAN 通道复位

```
canStatus CANLIBAPI canSetBitrate(int hnd, int bitrate);
```
//设定指定 CAN 通道的通信速率，波特率

```
/*
hnd
An open handle to a CAN channel.
bitrate
The new bit rate, in bits/second.
*/
```

```
canStatus canSetBusParams(int handle,
        long freq,
        unsigned int tseg1,
        unsigned int tseg2,
        unsigned int sjw,
        unsigned int noSamp,
        unsigned int syncmode);
```
//设定 CAN 控制器的时序参数

```
canStatus canSetDriverMode(
    const int hnd,
    int lineMode,
    int resNet
);
```
//设置总线驱动模式 sleep, wakeup, standby, fast mode, etc.

```
canStatus canTranslateBaud(
        long *freq, unsigned int *tseg1,
        unsigned int *tseg2, unsigned int *sjw,
        unsigned int *nosamp, unsigned int *syncMode);
```

//翻译到对应的参数值

canStatus canUnloadLibrary(void)
//释放动态链接库，和初始化对应

canStatus canWaitForEvent(int hnd, DWORD timeout);
//等待事件发生或等待超时，比如：接收到一个数据，总线状态改变等

canStatus canWrite(
    int handle, long id,
    void* msg, unsigned int dlc,
    unsigned int flag);//该标志位用于选择发送帧类型
//发送一个 CAN 信号

canStatus canWriteSync(int handle, unsigned long timeout);
//等待知道指定句柄发送完所有消息，或超时。
//可以将该函数防御 canWrite 前，保证消息不溢出

canStatus canWriteWait(
    int handle, long id,
    void* msg, unsigned int dlc,
    unsigned int flag,
    unsigned long timeout);
//发送一个消息，返回消息发送成功或超时标志，为了保证消息发送成功，可以经常使用该函数。

canStatus CANLIBAPI kvBeep(int hnd, int freq, unsigned int duration);
//该函数用于驱动设备上的扬声器以特定的频率发出持续多长时间的声音
/*
hnd
An open handle to a CAN channel.
freq
The frequency (in Herz) of the sound
duration
The duration of the sound, in milliseconds.
*/

canStatus CANLIBAPI kvFlashLeds(int hnd, int action, int timeout);
//这个函数控制 LED 显示开启或关闭，开启时间为 timeout 单位是毫秒

kvStatus CANLIBAPI kvGetSupportedInterfaceInfo(int index, char *hwName, size_t nameLen, int *hwType, int *hwBusType);
//该函数用于参看驱动库是否支持该硬件

kvStatus CANLIBAPI kvReadTimer(int hnd, unsigned int *time);
//该函数用于查看指定通道的硬件时钟

canStatus CANLIBAPI kvSelfTest(int hnd, unsigned long *presults);
//用于内建自我测试