



## 六叶树 USBCAN/FD

## 接口函数使用手册

| 类别  | 内容   |
|-----|--|
| 关键词 | USBCANFD、六叶树、接口函数使用手册                          |
| 摘要  | USBCANFD 接口函数库是提供给用户进行上位机二次开发,可以自行编程进行数据收发处理等。 |

库介绍:

| 类别  | 内容   |
|-----|--|
| 库文件 | Windows 平台:<br>ControlCAN.h、 ControlCAN.dll、 ControlCAN.lib<br>Linux 平台:<br>ControlCAN.h、 libControlCAN.a libControlCAN.so |
|     |  |

文档记录

| 版本    | 日期      | 说明   |
|-------|---------|------|
| V1.00 | 2022.01 | 创建文档 |
|       |         |      |



## 目录

|                              |    |
|------------------------------|----|
| 六叶树 USBCAN/FD .....          | 1  |
| 接口函数使用手册 .....               | 1  |
| 1.函数库简介 .....                | 3  |
| 1.1 功能说明 .....               | 3  |
| 1.2 文件清单 .....               | 3  |
| 1.3 调用流程(CAN 设备) .....       | 4  |
| 1.4 调用流程(CANFD 设备) .....     | 5  |
| 2.函数说明 .....                 | 6  |
| 2.1 LCAN_OpenDevice .....    | 6  |
| 2.2 LCAN_CloseDevice .....   | 6  |
| 2.3 LCAN_SetFliter .....     | 7  |
| 2.4 LCAN_InitCAN .....       | 8  |
| 2.5 LCAN_InitCANFD .....     | 9  |
| 2.6 LCAN_GetReceiveNum ..... | 11 |
| 2.7 LCAN_Receive .....       | 12 |
| 2.8 LCAN_ReceiveFD .....     | 13 |
| 2.9 LCAN_Transmit .....      | 14 |
| 2.10 LCAN_TransmitFD .....   | 16 |
| 2.11 LCAN_ClearBuffer .....  | 18 |
| 3.重要结构体 .....                | 18 |
| 3.1 LCAN_CAN_OBJ .....       | 18 |
| 3.2 LCAN_CANFD_OBJ .....     | 19 |
| 3.3 LCAN_INIT_CONFIG .....   | 20 |
| 3.4 LCAN_INITFD_CONFIG ..... | 21 |
| 3.5 LCAN_FLITER_CONFIG ..... | 24 |
| 4.免责声明 .....                 | 25 |
| 5.联系我们 .....                 | 25 |
| 附录一:(设备类型定义) .....           | 26 |
| 附录二:(常用宏定义) .....            | 26 |



## 1. 函数库简介

### 1.1 功能说明

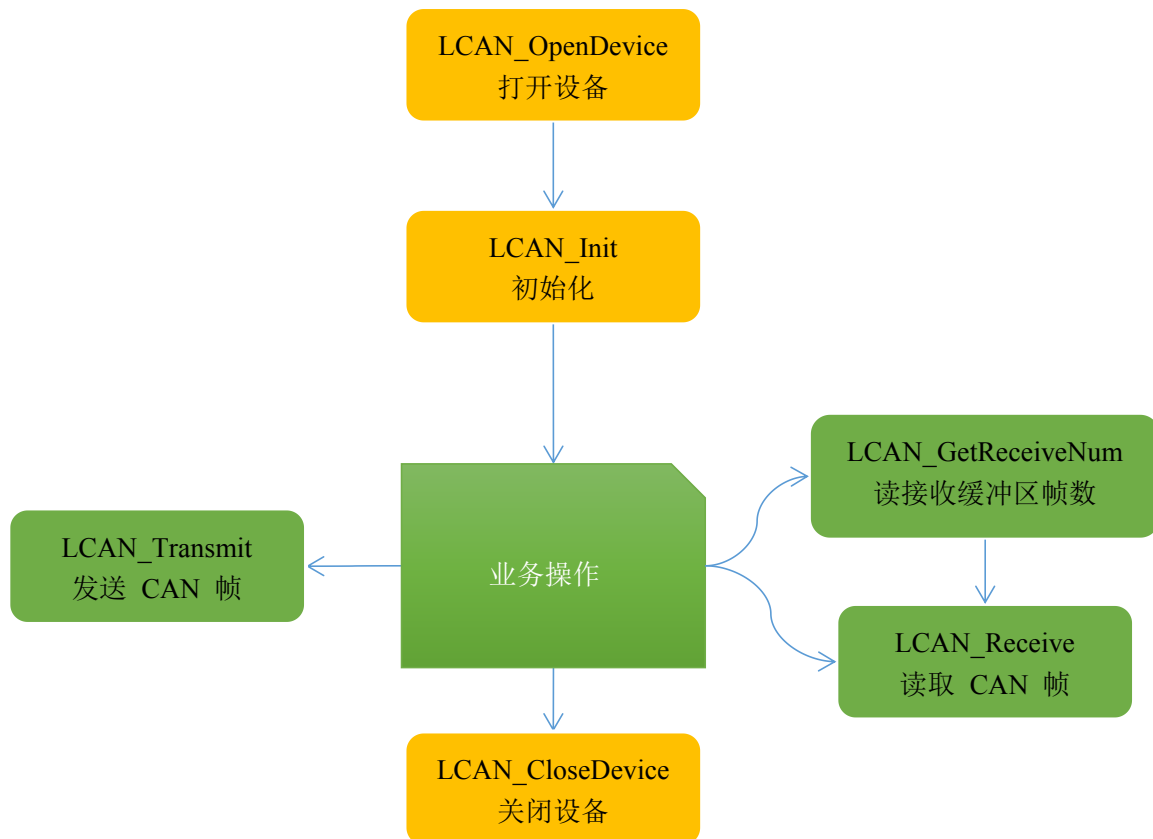
函数库是提供给用户进行上位机二次开发，可以自行编程进行数据收发处理等。不同操作系统及平台使用的函数接口是一致的，应用可以很容易从一个平台移植到另一个平台。

### 1.2 文件清单

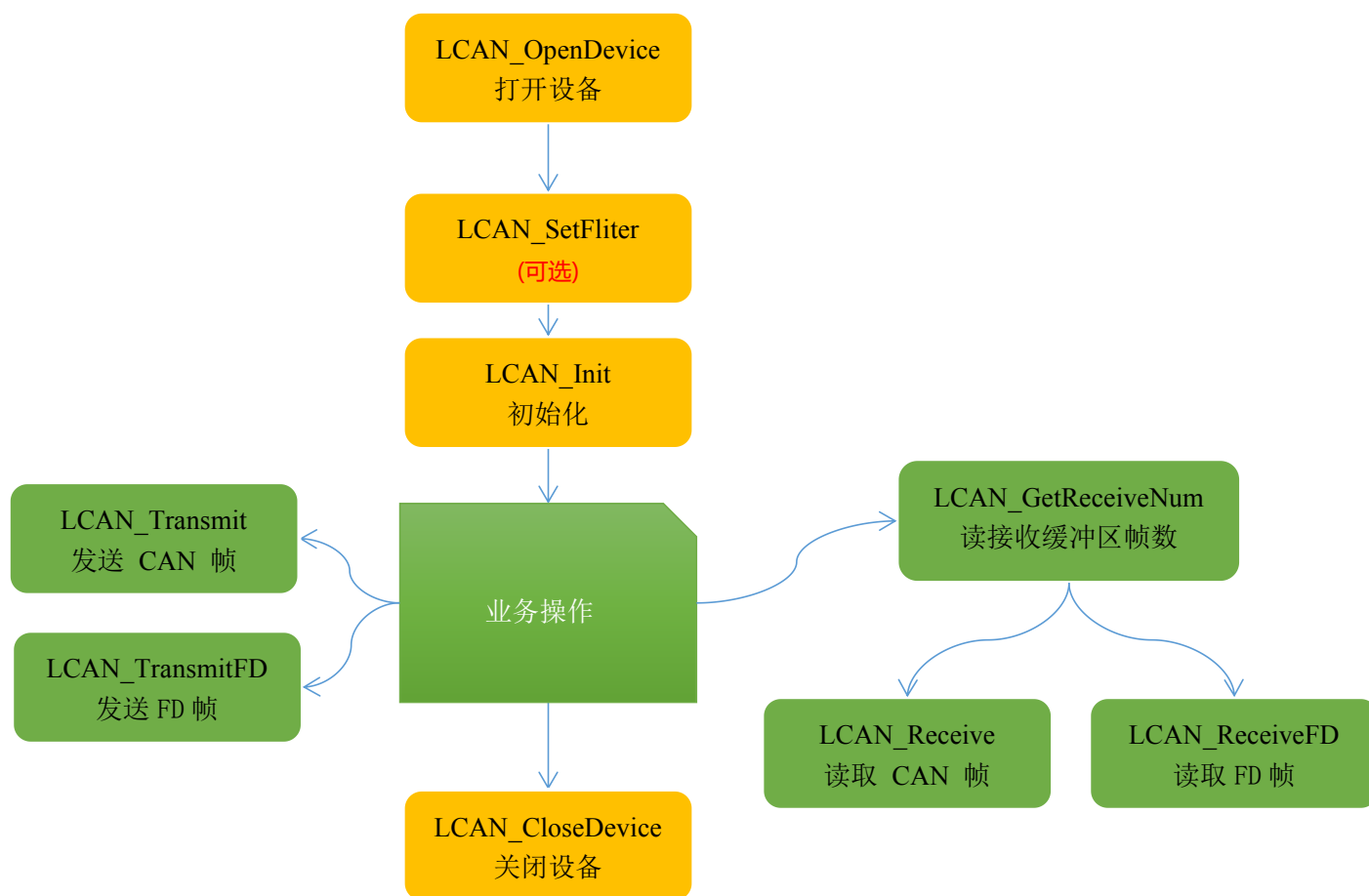
| 平台      | 文件               | 文件说明                                      |
|---------|------------------|---|
| Windows | ControlCAN.h     | 函数接口头文件，应用需要的函数都在该文件中已声明，应用调用库函数，包含该文件即可。 |
|         | ControlCAN.dll   | 动态库                                       |
|         | ControlCAN.lib   | 静态库                                       |
| Linux   | ControlCAN.h     | 函数接口头文件，应用需要的函数都在该文件中已声明，应用调用库函数，包含该文件即可。 |
|         | libControlCAN.so | 动态库                                       |
|         | libControlCAN.a  | 静态库                                       |



## 1.3 调用流程(CAN 设备)



## 1.4 调用流程(CANFD 设备)





## 2. 函数说明

### 2.1 LCAN\_OpenDevice

| 函数       |   |
|----------|---|
| 函数原型     | UINT LCAN_OpenDevice(UINT devType, UINT devIndex);                      |
| 描述       | 设备打开函数，注意一个设备只能打开一次。  |
| 参数/返回值   |   |
| devType  | 设备类型号。对应不同的产品型号，见附录一。   |
| devIndex | 设备索引号。比如当只有一个 USBCAN2 时，索引号为 0，这时再插入一个 USBCAN2，那么后面插入的这个设备索引号就是 1，以此类推。 |
| 返回值      | LCAN_STATUS_OK 表示设置成功，STATUS_ERR 表示设置失败。                                |

示例:

```
#include "ControlCan.h"
UINT deviceType = 4; /* USBCAN2 */
UINT deviceInd = 0; /* 索引号 0 */
UINT ret;
ret = LCAN_OpenDevice(deviceType, deviceInd);
if (ret != LCAN_STATUS_OK)
{
    //失败
    //...
}
```

### 2.2 LCAN\_CloseDevice

| 函数      |   |
|---------|---|
| 函数原型    | UINT LCAN_CloseDevice(UINT devType, UINT devIndex); |
| 描述      | 此函数用以关闭设备。  |
| 参数/返回值  |   |
| devType | 设备类型号。  |



## USBCAN 接口函数使用手册

|          |  |
|----------|--|
| devIndex | 设备索引号。比如当只有一个 USBCAN2 时，索引号为 0，这时再插入一个 USBCAN2，那么后面插入的这个设备索引号就是 1，以此类推。需对应已经打开的设备。 |
| 返回值      | LCAN_STATUS_OK 表示设置成功，LCAN_STATUS_ERR 表示设置失败。                                      |

示例:

```
#include "ControlCan.h"
UINT deviceType = 4; /* USBCAN2 */
UINT deviceInd = 0; /* 索引号 0 */
UINT ret;
ret = LCAN_CloseDevice(deviceType, deviceInd);
if (ret != LCAN_STATUS_OK)
{
    //失败...
}
```

### 2.3 LCAN\_SetFliter

| 函数            |   |
|---------------|---|
| 函数原型          | UINT LCAN_SetFliter(UINT devType, UINT devIndex, UINT canIndex, PLCAN_FLITER_CONFIG pFliterConfig); |
| 描述            | ID 过滤设置。  |
| 参数/返回值        |   |
| devType       | 设备类型号。  |
| devIndex      | 设备索引号。  |
| canIndex      | 第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。   |
| pFliterConfig | 过滤规则参数，具体见 PLCAN_FLITER_CONFIG。   |
| 返回值           | LCAN_STATUS_OK 表示设置成功，LCAN_STATUS_ERR 表示设置失败。   |
| 注意            | <i>1.目前只针对 FD 设备有效。<br/>2.必须先于 LCAN_InitCANFD 执行。<br/>3.如果应用无 ID 过滤要求，可以不执行该函数，设备默认接收所有 ID 数据。</i>  |



### 2.4 LCAN\_InitCAN

| 函数          |   |
|-------------|---|
| 函数原型        | UINT LCAN_InitCAN(UINT devType, UINT devIndex, UINT canIndex, PLCAN_INIT_CONFIG pInitConfig); |
| 描述          | 此函数用以初始化 CAN 设备，一般用于配置波特率、过滤器。  |
| 参数/返回值      |   |
| devType     | 设备类型号。  |
| devIndex    | 设备索引号。  |
| canIndex    | 第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。   |
| pInitConfig | 初始化参数结构，为一个 LCAN_INIT_CONFIG 结构体变量。   |
| 返回值         | LCAN_STATUS_OK 表示设置成功，LCAN_STATUS_ERR 表示设置失败。   |





示例:

```
#include "ControlCan.h"
UINT deviceType = 4; /* USBCAN2 */
UINT deviceInd = 0; /* 索引号 0 */
UINT canInd = 0; /* 第 0 路 CAN */
LCAN_INIT_CONFIG cfg;
UINT ret;

//clr
memset(&cfg, 0, sizeof(LCAN_INIT_CONFIG));
cfg.accCode = 0x00000000;
//屏蔽码->全部接收
cfg.accMask = 0xFFFFFFFF;

//滤波方式->单滤波
cfg.filter = 1;

//波特率->500K
cfg.timing0 = 0x00;
cfg.timing1 = 0x1C;

//模式->正常工作模式
cfg.mode = 0;

//初始化
ret = LCAN_InitCAN(deviceType, deviceInd, canInd, &cfg);
if (ret != LCAN_STATUS_OK)
{
    //失败...
}
```

## 2.5 LCAN\_InitCANFD

| 函数   |   |
|------|---|
| 函数原型 | UINT LCAN_InitCAN(UINT devType, UINT devIndex, UINT canIndex, PLCAN_INITFD_CONFIG pInitConfig); |



## USBCAN 接口函数使用手册

|             |   |
|-------------|---|
| 描述          | 此函数用以初始话 FD 设备，一般用于配置波特率、工作模式。                |
| 参数/返回值      |   |
| devType     | 设备类型号。  |
| devIndex    | 设备索引号。  |
| canIndex    | 第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。 |
| pInitConfig | 初始化参数结构，为一个 LCAN_INITFD_CONFIG 结构体变量。         |
| 返回值         | LCAN_STATUS_OK 表示设置成功，LCAN_STATUS_ERR 表示设置失败。 |

示例:

```
#include "ControlCan.h"
UINT deviceType = 4; /* USBCAN2 */
UINT deviceInd = 0; /* 索引号 0 */
UINT canInd = 0; /* 第 0 路 CAN */
LCAN_INITFD_CONFIG cfg;
UINT ret;

//clr
memset(&cfg, 0, sizeof(LCAN_INITFD_CONFIG));
//仲裁域波特率->1M
cfg.abit_timing = 0x00018B2E;
//仲裁域波特率->5M
cfg.dbit_timing = 0x00010207;
//模式->正常工作模式
cfg.mode = 0;

//初始化
ret = LCAN_InitCAN(deviceType, deviceInd, canInd, &cfg);
if (ret != LCAN_STATUS_OK)
{
    //失败...
}
```



## 2.6 LKAN\_GetReceiveNum

| 函数       |  |
|----------|--|
| 函数原型     | UINT LKAN_GetReceiveNum(UINT devType, UINT devIndex, UINT canIndex, UCHAR type); |
| 描述       | 获取缓冲区中 CAN , CANFD 或者合并接收报文数目。   |
| 参数/返回值   |  |
| devType  | 设备类型号。   |
| devIndex | 设备索引号。   |
| canIndex | 第几路 CAN。即对应卡的 CAN 通道号, CAN0 为 0, CAN1 为 1, 以此类推。                                 |
| type     | 获取 CAN , CANFD 或者合并接收报文, 0=CAN, 1=CANFD, 2=合并接收。                                 |
| 返回值      | 返回未被读取的帧数, 没有数据返回 0。   |

示例:

```
#include "ControlCan.h"
UINT deviceType = 4; /* USBCAN2 */
UINT deviceInd = 0; /* 索引号 0 */
UINT canInd = 0; /* CAN0 */
UCHAR type = 2;
UINT ret;

//type =2, 查询 CAN 和 FD 的总帧数
ret = LKAN_GetReceiveNum(deviceType, deviceInd, canInd, type);
if (0 == ret)
{
    //没有数据
    return ;
}
//有数据未读取, 读数据
LKAN_Receive(...)
LKAN_ReceiveFD(...)
```



## 2.7 LCAN\_Receive

| 函数       |   |
|----------|---|
| 函数原型     | UINT LCAN_Receive(UINT devType, UINT devIndex, UINT canIndex, PLCAN_CAN_OBJ pReceive, UINT len, INT waitTime= - 1); |
| 描述       | 接收函数。此函数从指定的设备 CAN 通道的接收缓冲区中读取数据。建议在调用之前，先调用 LCAN_GetReceiveNum。函数获知缓冲区中有多少帧，然后对应地去接收。                              |
| 参数/返回值   |   |
| devType  | 设备类型号。  |
| devIndex | 设备索引号，从 0 开始。   |
| canIndex | 第几路 CAN，从 0 开始。   |
| pReceive | 用来接收的帧结构体 LCAN_CAN_OBJ 数组的首指针。  |
| len      | 用来接收的帧结构体数组的长度（本次接收的最大帧数，实际返回值小于等于这个值）。   |
| waitTime | 缓冲区无数据，函数阻塞等待时间，以毫秒为单位。若为-1 则表示无超时，一直等待。  |
| 返回值      | 返回实际读取到的帧数，实际有效数据按顺序存储在 <i>pReceive</i> 指定的数组里。   |

示例:

```
#include "ControlCan.h"
UINT deviceType = 4; /* USBCAN2 */
UINT deviceInd = 0; /* 索引号 0 */
UINT canInd = 0; /* CAN0 */
UINT ret;
LCAN_CAN_OBJ recObjBuf[100];

//一次最多能接收 100 帧，如果缓冲区无数据，接收函数等待 500 毫秒后退出，返回 0
ret = LCAN_Receive(deviceType, deviceInd, canInd, recObjBuf, 100, 500);
if (0 == ret)
{
    //没有数据
    return ;
}
```



## 2.8 LCAN\_ReceiveFD

| 函数       |   |
|----------|---|
| 函数原型     | UINT LCAN_ReceiveFD(UINT devType, UINT devIndex, UINT canIndex, PLCAN_CANFD_OBJ pReceive, UINT len, INT waitTime= - 1); |
| 描述       | 接收函数。此函数从指定的设备 CAN 通道的接收缓冲区中读取数据。建议在调用之前，先调用 LCAN_GetReceiveNum。函数获知缓冲区中有多少帧，然后对应地去接收。                                  |
| 参数/返回值   |   |
| devType  | 设备类型号。  |
| devIndex | 设备索引号，从 0 开始。   |
| canIndex | 第几路 CAN，从 0 开始。   |
| pReceive | 用来接收的帧结构体 LCAN_CANFD_OBJ 数组的首指针。  |
| len      | 用来接收的帧结构体数组的长度（本次接收的最大帧数，实际返回值小于等于这个值）。   |
| waitTime | 缓冲区无数据，函数阻塞等待时间，以毫秒为单位。若为-1 则表示无超时，一直等待。  |
| 返回值      | 返回实际读取到的帧数，实际有效数据按顺序存储在 <i>pReceive</i> 指定的数组里。   |

示例:

```
#include "ControlCan.h"
UINT deviceType = 4; /* USBCAN2 */
UINT deviceInd = 0; /* 索引号 0 */
UINT canInd = 0; /* CAN0 */
UINT ret;
LCAN_CANFD_OBJ recObjBuf[100];

//一次最多能接收 100 帧，如果缓冲区无数据，接收函数等待 500 毫秒后退出，返回 0
ret = LCAN_ReceiveFD(deviceType, deviceInd, canInd, recObjBuf, 100, 500);
if (0 == ret)
{
    //没有数据
    return ;
}
```



### 2.9 LCAN\_Transmit

| 函数       |  |
|----------|--|
| 函数原型     | UINT LCAN_Transmit(UINT devType, UINT devIndex, UINT canIndex, PLCAN_CAN_OBJ pSend, UINT len); |
| 描述       | 发送函数。返回值为实际发送成功的帧数。  |
| 参数/返回值   |  |
| devType  | 设备类型号。   |
| devIndex | 设备索引号，从 0 开始。  |
| canIndex | 第几路 CAN，从 0 开始。  |
| pSend    | 要发送的帧结构体 LCAN_CAN_OBJ 数组的首指针，实际有效数据按顺序存储在这里。   |
| len      | 要发送的帧结构体数组的长度（发送的帧数量）。   |
| 返回值      | 返回实际发送成功的帧数  |



示例:

```
#include "ControlCan.h"
UINT deviceType = 4; /* USBCAN2 */
UINT deviceInd = 0; /* 索引号 0 */
UINT canInd = 0; /* CAN0 */
UINT ret;
LCAN_CAN_OBJ sendObjBuf[2]; // 定义两帧的结构体数组

//填充发送数据
sendObjBuf[0].id = 0x00000001; // 填写第一帧的 ID
sendObjBuf[0].sendType = 0; // 正常发送
sendObjBuf[0].remoteFlag = 0; // 数据帧
sendObjBuf[0].externFlag = 0; // 标准帧
sendObjBuf[0].dataLen = 1; // 数据长度 1 个字节
sendObjBuf[0].data[0] = 0x66; // 数据 0 为 0x66
sendObjBuf[1].id = 0x00000002; // 填写第二帧的 ID
sendObjBuf[1].sendType = 0; // 正常发送
sendObjBuf[1].remoteFlag = 0; // 数据帧
sendObjBuf[1].externFlag = 0; // 标准帧
sendObjBuf[1].dataLen = 1; // 数据长度 1 个字节
sendObjBuf[1].data[0] = 0x55; // 数据 0 为 0x55

//发送 2 帧数据
ret = LCAN_Transmit(deviceType, deviceInd, canInd, sendObjBuf, 2);
if (2 != ret)
{
    //发送失败
    return ;
}
```



### 2.10 LCAN\_TransmitFD

| 函数       |  |
|----------|--|
| 函数原型     | UINT LCAN_TransmitFD(UINT devType, UINT devIndex, UINT canIndex, PLCAN_CANFD_OBJ pSend, UINT len); |
| 描述       | 发送函数。返回值为实际发送成功的帧数。  |
| 参数/返回值   |  |
| devType  | 设备类型号。   |
| devIndex | 设备索引号，从 0 开始。  |
| canIndex | 第几路 CAN，从 0 开始。  |
| pSend    | 要发送的帧结构体 LCAN_CANFD_OBJ 数组的首指针，实际有效数据按顺序存储在这里。   |
| len      | 要发送的帧结构体数组的长度（发送的帧数量）。   |
| 返回值      | 返回实际发送成功的帧数  |





示例:

```
#include "ControlCan.h"
UINT deviceType = 4; /* USBCAN2 */
UINT deviceInd = 0; /* 索引号 0 */
UINT canInd = 0; /* CAN0 */
UINT ret;
LCAN_CANFD_OBJ sendObjBuf[2]; // 定义两帧的结构体数组

//填充发送数据
sendObjBuf[0].id = 0x00000001; // 填写第一帧的 ID
sendObjBuf[0].sendType = 0; // 正常发送
sendObjBuf[0].remoteFlag = 0; // 数据帧
sendObjBuf[0].externFlag = 0; // 标准帧
//FD 长度取值范围:0-8/12/16/20/24/32/48/64
sendObjBuf[0].dataLen = 1; // 数据长度 1 个字节
sendObjBuf[0].data[0] = 0x66; // 数据 0 为 0x66
sendObjBuf[1].id = 0x00000002; // 填写第二帧的 ID
sendObjBuf[1].sendType = 0; // 正常发送
sendObjBuf[1].remoteFlag = 0; // 数据帧
sendObjBuf[1].externFlag = 0; // 标准帧
//FD 长度取值范围:0-8/12/16/20/24/32/48/64
sendObjBuf[1].dataLen = 1; // 数据长度 1 个字节
sendObjBuf[1].data[0] = 0x55; // 数据 0 为 0x55

//发送 2 帧数据
ret = LCAN_TransmitFD(deviceType, deviceInd, canInd, sendObjBuf, 2);
if (2 != ret)
{
    //发送失败
    return ;
}
```



## 2.11 LCAN\_ClearBuffer

| 函数       |   |
|----------|---|
| 函数原型     | UINT LCAN_ClearBuffer(UINT devType, UINT devIndex,UINT canIndex); |
| 描述       | 清除指定通道的接收缓冲区。   |
| 参数/返回值   |   |
| devType  | 设备类型号。  |
| devIndex | 设备索引号，从 0 开始。   |
| canIndex | 第几路 CAN，从 0 开始。   |
| 返回值      | LCAN_STATUS_OK 表示设置成功，LCAN_STATUS_ERR 表示设置失败。                     |

## 3.重要结构体

### 3.1 LCAN\_CAN\_OBJ

| 结构体       |   |
|-----------|---|
| 定义        | <pre>typedef struct _LCAN_CAN_OBJ {<br/>    UINT id;<br/>    UINT timeStamp;<br/>    UCHAR timeFlag;<br/>    UCHAR sendType;<br/>    UCHAR remoteFlag;<br/>    UCHAR externFlag;<br/>    UCHAR dataLen;<br/>    UCHAR data[8];<br/>    UCHAR reserved[3];<br/>} LCAN_CAN_OBJ, *PLCAN_CAN_OBJ;</pre> |
| 描述        | LCAN_CAN_OBJ 结构体是 CAN 帧结构体，即 1 个结构体表示一个帧的数据结构。在发送函数 LCAN_Transmit 和接收函数 LCAN_Receive 中，被用来传送 CAN 信息帧。   |
| 说明        |   |
| id        | 帧 id，32 位变量，数据格式为靠右对齐。  |
| timeStamp | 设备接收到某一帧的时间标识。  |
| timeFlag  | 保留。   |
| sendType  | 发送帧类型。<br>=0 时为正常发送（发送失败会自动重发，重发最长时间为 1.5-3 秒）；<br>=1 时为单次发送（只发送一次，不自动重发）；  |



## USBCAN 接口函数使用手册

|            |   |
|------------|---|
|            | =2 时为自发自收（自测试模式，用于测试 CAN 卡是否损坏）；<br>=3 时为单次自发自收（单次自测试模式，只发送一次）。<br>只在此帧为发送帧时有意义。                          |
| remoteFlag | 是否是远程帧。=0 时为数据帧，=1 时为远程帧（数据段空）。   |
| externFlag | 是否是扩展帧。=0 时为标准帧（11 位 ID），=1 时为扩展帧（29 位 ID）  |
| dataLen    | 数据长度 DLC (<=8)，即 CAN 帧 Data 有几个字节。约束了后面 Data[8]中的有效字节。  |
| data[8]    | CAN 帧的数据。由于 CAN 规定了最大是 8 个字节，所以这里预留了 8 个字节的空间，受 DataLen 约束。如 DataLen 定义为 3，即 Data[0]、Data[1]、Data[2]是有效的。 |
| reserved   | 系统保留。   |

### 3.2 LCAN\_CANFD\_OBJ

| 结构体       |  |
|-----------|--|
| 定义        | <pre>typedef struct _LCAN_CANFD_OBJ {<br/>    UINT id;<br/>    UINT timeStamp;<br/>    UCHAR flag;<br/>    UCHAR sendType;<br/>    UCHAR remoteFlag;<br/>    UCHAR externFlag;<br/>    UCHAR dataLen;<br/>    UCHAR data[64];<br/>    UCHAR reserved[3];<br/>} LCAN_CANFD_OBJ, *PLCAN_CANFD_OBJ;</pre> |
| 描述        | LCAN_CANFD_OBJ 结构体是 FD 帧结构体，即 1 个结构体表示一个帧的数据结构。在发送函数 LCAN_TransmitFD 和接收函数 LCAN_ReceiveFD 中，被用来传送 FD 信息帧。  |
| 说明        |  |
| id        | 帧 id，32 位变量，数据格式为靠右对齐。   |
| timeStamp | 设备接收到某一帧的时间标识，接收时有效。   |
| flag      | 0x08:CANFD_BRS<br>0x10:CANFD_ESI   |
| sendType  | 发送帧类型。（发送时有效）<br>=0 时为正常发送（发送失败会自动重发，重发最长时间为 1.5-3 秒）；<br>=1 时为单次发送（只发送一次，不自动重发）；  |



## USBCAN 接口函数使用手册

|            |   |
|------------|---|
|            | =2 时为自发自收（自测试模式，用于测试 CAN 卡是否损坏）；<br>=3 时为单次自发自收（单次自测试模式，只发送一次）。<br>只在此帧为发送帧时有意义。                          |
| remoteFlag | 是否是远程帧。=0 时为数据帧，=1 时为远程帧（数据段空）。   |
| externFlag | 是否是扩展帧。=0 时为标准帧（11 位 ID），=1 时为扩展帧（29 位 ID）  |
| dataLen    | 数据长度 DLC (<=8)，即 CAN 帧 Data 有几个字节。约束了后面 Data[8]中的有效字节。  |
| data[8]    | CAN 帧的数据。由于 CAN 规定了最大是 8 个字节，所以这里预留了 8 个字节的空间，受 DataLen 约束。如 DataLen 定义为 3，即 Data[0]、Data[1]、Data[2]是有效的。 |
| reserved   | 系统保留。   |

### 3.3 LCAN\_INIT\_CONFIG

| 结构体      |   |
|----------|---|
| 定义       | <pre>typedef struct _INIT_CONFIG {<br/>    UINT    accCode;<br/>    UINT    accMask;<br/>    UINT    reserved;<br/>    UCHAR   filter;<br/>    UCHAR   timing0;<br/>    UCHAR   timing1;<br/>    UCHAR   mode;<br/>}LCAN_INIT_CONFIG, *PLCAN_INIT_CONFIG;</pre> |
| 描述       | 结构体定义了初始化 CAN 的配置。结构体将在 LCAN_InitCAN 函数中被填充，即初始化之前，要先填好这个结构体变量。   |
| 说明       |   |
| accCode  | 验收码。SJA1000 的帧过滤验收码。对经过屏蔽码过滤为“有关位”进行匹配，全部匹配成功后，此帧可以被接收。否则不接收。   |
| accMask  | 屏蔽码。SJA1000 的帧过滤屏蔽码。对接收的 CAN 帧 ID 进行过滤，对应位为 0 的是“有关位”，对应位为 1 的是“无关位”。屏蔽码推荐设置为 0xFFFFFFFF，即全部接收。   |
| reserved | 保留。   |
| filter   | 滤波方式。=1 表示单滤波，=0 表示双滤波  |
| timing0  | 波特率定时器 0。设置值见下表。  |
| timing1  | 波特率定时器 1。设置值见下表。  |
| mode     | 模式。=0 表示正常模式（相当于正常节点），=1 表示只听模式（只接收，不影响总线）。   |



## USBCAN 接口函数使用手册

波特率:(常见的波特率 (采样点 87.5%, SJW 为 0) )

| 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  |
| 250Kbps  | 0x01  | 0x1C  |
| 400Kbps  | 0x80  | 0xFA  |
| 500Kbps  | 0x00  | 0x1C  |
| 666Kbps  | 0x80  | 0xB6  |
| 800Kbps  | 0x00  | 0x16  |
| 1000Kbps | 0x00  | 0x14  |

可以通过波特率计算器计算:

波特率计算器

USBCAN1/2

PEAK

| 序号 | 定时器0 | 定时器1 | 采样点   | 实际波特率      | 误差    |
|----|------|------|-------|------------|-------|
| 0  | 0x01 | 0x76 | 50.0% | 250.000KHz | 0.00% |
| 1  | 0x01 | 0x67 | 56.0% | 250.000KHz | 0.00% |
| 2  | 0x01 | 0x58 | 62.0% | 250.000KHz | 0.00% |
| 3  | 0x01 | 0x49 | 68.0% | 250.000KHz | 0.00% |
| 4  | 0x01 | 0x3A | 75.0% | 250.000KHz | 0.00% |
| 5  | 0x01 | 0x2B | 81.0% | 250.000KHz | 0.00% |
| 6  | 0x01 | 0x1C | 87.0% | 250.000KHz | 0.00% |

六叶树教育科技

计算

退出

波特率: 250 Kbps

时钟: 16000 KHz

误差: <=1%

采样点: >=50%

### 3.4 LCNAN\_INITFD\_CONFIG

| 结构体 |  |
|-----|--|
| 定义  | <pre>typedef struct _INITFD_CONFIG {<br/>    UINT    abitBaudHz;</pre> |



## USBCAN 接口函数使用手册

|             |   |
|-------------|---|
|             | UINT dbitBaudHz;<br>UINT abit_timing;<br>UINT dbit_timing;<br>UCHAR mode;<br>UCHAR fdEn;<br>UCHAR isoEn;<br>UCHAR rev1;<br>}LCAN_INITFD_CONFIG, *PLCAN_INITFD_CONFIG; |
| 描述          | 结构体定义了初始化 CANFD 的配置。结构体将在 LCAN_InitCANFD 函数中被填充，即初始化之前，要先填好这个结构体变量。   |
| 说明          |   |
| abitBaudHz  | 仲裁域波特率，单位:Hz<br>取值范围：【1000000，800000，500000，250000，125000，100000，50000】   |
| dbitBaudHz  | 数据域波特率，单位:Hz<br>取值范围：【5000000，4000000，2000000，1000000，800000，500000，250000，125000，100000】<br><b>fdEn 为 1 时有效。</b>   |
| abit_timing | 仲裁域波特率配置字   |
| dbit_timing | 数据域波特率配置字<br><b>fdEn 为 1 时有效。</b>   |
| fdEn        | 是否使能 FD。<br>0:工作于 CAN 模式<br>1:工作于 FD 模式   |
| isoEn       | 是否使能 ISO。<br>0:NONE<br>1:ISO  |
| rev1        | 保留,设置为 0。   |
| mode        | 模式。=0 表示正常模式（相当于正常节点），=1 表示只听模式（只接收，不影响总线）。   |

温馨提示:

接口里包含了两组波特率配置参数，第一组是 <abitBaudHz,dbitBaudHz>，第二组 <abit\_timing,dbit\_timing>，使用规则如下：

**fdEn 使能时(工作于 FD 模式):**

1.<abitBaudHz,dbitBaudHz>的值非 0，代表通过使用该参数初始化波特率，这种方式简单易用，适用大部分应用场景，只要需要的波特率属于表格里的取值。

2.<abit\_timing,dbit\_timing> 主要用于波特率自定义，可以精确的细化波特率参数，在 <abitBaudHz,dbitBaudHz>的值为 0 时才有效，该参数可以通过六叶树 USBCAN 调试助手的波特率计算软件计算获取。

**fdEn 非使能时(工作于 CAN 模式):**

1.<abitBaudHz>的值非 0，代表通过使用该参数初始化波特率，这种方式简单易用，适用大部分应用场



## USBCAN 接口函数使用手册

景，只要需要的波特率属于表格里的取值。

2.<abit\_timing>主要用于波特率自定义，可以精确的细化波特率参数，在<abitBaudHz>的值为0时才有效，该参数可以通过六叶树 USBCAN 调试助手的波特率计算软件计算获取。

CANFD 仲裁域波特率:

| 波特率     | abit_timing |
|---------|-------------|
| 1Mbps   | 0x00018B2E  |
| 800kbps | 0x00018E3A  |
| 500kbps | 0x0001975E  |
| 250kbps | 0x0001AFBE  |
| 125kbps | 0x0041AFBE  |
| 100kbps | 0x0041BBEE  |
| 50kbps  | 0x00C1BBEE  |

CANFD 数据域波特率:

| 波特率     | abit_timing |
|---------|-------------|
| 5Mbps   | 0x00010207  |
| 4Mbps   | 0x0001020A  |
| 2Mbps   | 0x0041020A  |
| 1Mbps   | 0x0081830E  |
| 800kbps | 0x00018E3A  |
| 500kbps | 0x0001975E  |
| 250kbps | 0x0001AFBE  |
| 125kbps | 0x0001AFBE  |
| 100kbps | 0x0041BBEE  |
| 50kbps  | 0x00C1BBEE  |

可以通过波特率计算器计算:



## USBCAN 接口函数使用手册

FD波特率计算器

设备类型:

输出结果:

主时钟:  MHz

仲裁域

波特率:  kbps 允许误差:  采样点:  同步跳转宽度:  +1

| 序号 | 配置字      | BRP | TSEG1 | TSEG2 | 采样点   | 实际波特率        | 误差    |
|----|----------|-----|-------|-------|-------|--------------|-------|
| 0  | 0080050C | 2   | 12    | 5     | 70.0% | 1000.000Kbps | 0.00% |
| 1  | 0080040D | 2   | 13    | 4     | 75.0% | 1000.000Kbps | 0.00% |
| 2  | 0080030E | 2   | 14    | 3     | 80.0% | 1000.000Kbps | 0.00% |

数据域

波特率:  kbps 允许误差:  采样点:  同步跳转宽度:  +1

| 序号 | 配置字      | BRP | TSEG1 | TSEG2 | 采样点   | 实际波特率        | 误差    |
|----|----------|-----|-------|-------|-------|--------------|-------|
| 0  | 00000207 | 0   | 7     | 2     | 75.0% | 5000.000Kbps | 0.00% |
| 1  | 00000108 | 0   | 8     | 1     | 83.3% | 5000.000Kbps | 0.00% |
| 2  | 00000009 | 0   | 9     | 0     | 91.7% | 5000.000Kbps | 0.00% |

步骤: 填写期望波特率->点击[计算]->表格(仲裁域和数据域)里选择结果->点击[保存并退出]

### 3.5 LCAN\_FLITER\_CONFIG

| 结构体 |   |
|-----|---|
| 定义  | <pre>typedef struct _FLITER_OBJ {     UCHAR cmd;     UCHAR fliterMode;     UCHAR idType;     UCHAR rev1;     UINT fliterStardId;     UINT filterEndId; }LCAN_FLITER_CONFIG, *PLCAN_FLITER_CONFIG;</pre> |
| 描述  | 滤波参数配置结构体, 通过 cmd 来识别滤波参数的解析。   |
| 说明  |   |
| cmd | 0x00:清空滤波参数<br>0x01:新增滤波范围(只有属于 fliterStardId 和 fliterEndId 范围以内的帧才能被设备接收)<br>0x02:使能滤波   |





|               |  |
|---------------|--|
| idType        | 帧 ID 类型:(cmd 为新增滤波指令时有效)<br>0x00:标准帧<br>0x01:扩展帧 |
| reserved      | 保留。  |
| fliterStardId | 滤波起始帧 ID   |
| fliterEndId   | 滤波结束帧 ID   |

温馨提示:

- 1.目前只针对 FD 设备有效。
- 2.必须先于 `LCAN_InitCANFD` 执行。
- 3.如果应用无 ID 过滤要求,可以不执行该函数,设备默认接收所有 ID 数据。
- 4.常见执行顺序,<清空滤波参数>-><新增滤波范围 1>-><新增滤波范围 2>-><使能滤波>。多个滤波范围,多次执行<新增滤波范围>即可。

## 4.免责声明

本文档提供有关长沙六叶树教育科技有限公司产品的信息。本文档并未授予任何知识产权的许可,并未以明示或暗示,或以禁止发言或其它方式授予任何知识产权许可。除六叶树教育科技在其产品的销售条款和条件中声明的责任之外,六叶树概不承担任何其它责任。并且,六叶树电子产品的销售使用不作任何明示或暗示的担保,包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等,均不作担保。六叶树可能随时对产品规格及产品描述做出修改,不另行通知。

## 5.联系我们



电话:15211065817(业务合作咨询)



邮箱:798746621@qq.com(业务咨询+技术支持)



微信:18673379565(技术支持)



官网:[www.liuyeshu.cn](http://www.liuyeshu.cn)(资料下载)



网上商城:<https://shop112408209.taobao.com>(产品购买)

淘宝店铺搜索:“六叶树教育科技”



## 附录一:(设备类型定义)

六叶树 USBCAN 卡设备类型定义如下:

| 产品型号        | 动态库中的设备名称        | 类型号(十进制) |
|-------------|------------------|----------|
| USBCAN1     | LCAN_USBCAN1     | 3        |
| USBCAN2     | LCAN_USBCAN2     | 4        |
| USBCAN-2E-U | LCAN_USBCAN-2E-U | 21       |

六叶树 USBCANFD 卡设备类型定义如下:

| 产品型号         | 动态库中的设备名称         | 类型号(十进制) |
|--------------|-------------------|----------|
| USBCANFDMini | LCAN_USBCANFDMini | 43       |
| USBCANFD1    | LCAN_USBCANFD1    | 42       |
| USBCANFD2    | LCAN_USBCANFD2    | 41       |

## 附录二:(常用宏定义)

| 项目                 | 相关宏  |  |
|--------------------|--|--|
| LCAN_CANFD_OBJ     | 0x08:LCAN_CANFD_BRS<br>0x10:LCAN_CANFD_ESI                           |  |
| 结果码                | 0:LCAN_STATUS_ERR<br>1:LCAN_STATUS_OK                                |  |
| LCAN_FLITER_CONFIG | 0x00:FLITER_CMD_CLR<br>0x01:FLITER_CMD_ADD<br>0x02:FLITER_CMD_ENABLE |  |
|                    |  |  |
|                    |  |  |