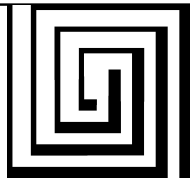

« МАРАФОН »

Библиотека CHAI 2.4.0

Руководство программиста

Версия документа 0.13

Марафон, Москва, 117330, Мосфильмовская ул., 17б
Тел.: +7 (495)-988-27-26, 939-13-24 Факс: +7 (495)-939-56-59

Гарантийные обязательства МАРАФОНА.

ГАРАНТИЙНЫЕ ОБЯЗАТЕЛЬСТВА
(Ограниченная гарантия на продукцию МАРАФОНА)

Программное обеспечение:

Гарантийное обслуживание по программному обеспечению можно получить, связавшись с офисом МАРАФОНА в оговоренный гарантийный период. Адрес офиса МАРАФОНА приведен на первой странице Руководства по эксплуатации устройства, а также приложен вместе с Регистрационной карточкой.

МАРАФОН гарантирует, что его программное обеспечение будет работать в строгом соответствии с прилагаемой к нему МАРАФОНОМ документацией в течении девяноста (90) дней с момента его приобретения у МАРАФОНА или Авторизованного Реселлера. МАРАФОН предоставляет гарантию на носитель, на котором поставляется программное обеспечение, в виде отсутствия потери им информации на тот же гарантийный срок. Данная гарантия имеет отношение только к приобретенному программному обеспечению или его замене по гарантии, и не касается любых обновлений или замен, которые получены через Internet или бесплатно.

Ответственность МАРАФОНА по обеспечению гарантии программного обеспечения состоит в замене его на новое, которое выполняет перечисленные в прилагаемой документации функции. Ответственность Заказчика состоит в выборе соответствующего приложения, программной платформы/системы и дополнительных материалов. МАРАФОН не отвечает за работоспособность программного обеспечения вместе с любыми аппаратными средствами и/или программными платформами/системами, которые поставляются третьими сторонами, если совместимость с ними не оговорена в прилагаемой к продукции МАРАФОН документации. Согласно данной гарантии, МАРАФОН старается обеспечить разумную совместимость своей продукции, но МАРАФОН не несет ответственность, если с аппаратными или программными средствами третьих фирм происходят сбои. МАРАФОН не гарантирует, что работа программного обеспечения будет непрерывна и в процессе не будут происходить ошибки, а также то, что все дефекты в программном продукте с или без учета документации на него, будут исправлены.

Ограничения гарантий

Вышеупомянутые гарантии и замечания являются исключительными и соответствуют всем прочим гарантиям, объявленным или подразумеваемым, которые даются в явном виде или в соответствии с законодательством, установленными законами или в другом виде, включая гарантии на сам товар и его пригодность для стандартных целей. МАРАФОН никогда не допускает и не принимает на себя прочую ответственность, связанную с продажами, поддержкой инсталляции или использования продукции МАРАФОНА

МАРАФОН никогда не несет ответственность по гарантии, если проводимое им тестирование и анализ определяет, что заявленный дефект в изделии не был обнаружен, или он был вызван неверным использованием заказчиком, или третьей стороной, невнимательной или неправильной инсталляцией или тестированием, попыткой ремонта неавторизованными лицами, или чем-либо еще, не

предусмотренным в назначении изделия, типа несчастного случая, огня, пожара и других бедствий.

Ограничения ответственности

Ни в каком случае МАРАФОН не несет ответственность за любые убытки, включая потерю данных, потерю прибыли, стоимости покрытия или других случайных, последовательных или непрямых убытков, являющихся следствием инсталляции, сопровождения, использования, производительности, неисправности или временной неработоспособности изделий производства МАРАФОНА. Эти ограничения действуют, даже если МАРАФОН был предупрежден о возможности такого убытка.

Регистрационная карточка, прилагаемая на обратной стороне Руководства, должна быть отправлена в офис МАРАФОН по факсу, электронной почте или почтовым отправлением. Список адресов/ телефонов/ факсов офисов МАРАФОНА содержится на первой странице данного Руководства.

Юр. адрес: 117330 Москва, ул. Мосфильмовская, дом 17Б.
Факт. адрес: 119899 Москва, Ленинске горы, МГУ, НИИЯФ, д.1. стр.5.
Тел. (495)-988-27-26, 939-56-59, 939-13-24
Факс. (495)-939-56-59
E-mail: support@marathon.ru
WEB: www.marathon.ru

По техническим вопросам звоните по тел. +7 (495)-988-27-26, 939-56-59, 939-13-24 или свяжитесь с нами по email support@marathon.ru.

Лицензионное соглашение на Программное обеспечение, поставляемое с CAN интерфейсами производства МАРАФОН

Все права на программное обеспечение, аппаратное обеспечение и данное руководство принадлежат фирме Марафон и защищены законодательством Российской Федерации.

ПЕРЕД ИСПОЛЬЗОВАНИЕМ ПРИЛАГАЕМОГО ИЗДЕЛИЯ ПОКУПАТЕЛЬ ДОЛЖЕН ВНИМАТЕЛЬНО ОЗНАКОМИТЬСЯ С УСЛОВИЯМИ НАСТОЯЩЕГО СОГЛАШЕНИЯ. ИСПОЛЬЗОВАНИЕ ДАННОГО ИЗДЕЛИЯ ПОДРАЗУМЕВАЕТ ПРИНЯТИЕ ЭТИХ ПОСТАНОВЛЕНИЙ И УСЛОВИЙ. ЕСЛИ ОГОВОРЕННЫЕ УСЛОВИЯ ЯВЛЯЮТСЯ ДЛЯ ПОКУПАТЕЛЯ НЕПРИЕМЛЕМЫМИ, ОН ДОЛЖЕН НЕЗАМЕДЛИТЕЛЬНО ВЕРНУТЬ НЕИСПОЛЬЗОВАННЫЙ КОМПЛЕКТ, ПРИ ЭТОМ ЗАТРАТЫ ПОКУПАТЕЛЯ БУДУТ ВОЗМЕЩЕНЫ.

ДАННЫЙ ДОКУМЕНТ ЯВЛЯЕТСЯ ЛИЦЕНЗИОННЫМ СОГЛАШЕНИЕМ, НО НЕ СОГЛАШЕНИЕМ О ПРОДАЖЕ. МАРАФОН ЯВЛЯЕТСЯ ВЛАДЕЛЬЦЕМ ИЛИ ИМЕЕТ ЛИЦЕНЗИОННЫЕ СОГЛАШЕНИЯ С ДРУГИМИ ВЛАДЕЛЬЦАМИ АВТОРСКИХ ПРАВ НА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ВХОДЯЩЕЕ В КОМПЛЕКТ ПОСТАВКИ СВОИХ ИЗДЕЛИЙ И ПО. ПОКУПАТЕЛЬ НЕ ПРИОБРЕТАЕТ НИКАКИХ ПРАВ НА ИНТЕЛЛЕКТУАЛЬНУЮ СОБСТВЕННОСТЬ, СОДЕРЖАЩУЮСЯ В ПРОГРАММНОМ ОБЕСПЕЧЕНИИ, ЗА ИСКЛЮЧЕНИЕМ ТЕХ, КОТОРЫЕ НАСТОЯЩЕЕ СОГЛАШЕНИЕ ПРЕДОСТАВЛЯЕТ ЕМУ В ОТНОШЕНИИ ЭТОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ПРАВО СОБСТВЕННОСТИ НА ПРИЛАГАЕМУЮ КОПИЮ УПОМЯНУТОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, А ТАКЖЕ НА ВСЕ КОПИИ, СДЕЛАННЫЕ С НЕЕ, СОХРАНЯЕТСЯ ЗА МАРАФОНОМ ИЛИ ДРУГИМИ ВЛАДЕЛЬЦАМИ АВТОРСКИХ ПРАВ. ПОКУПАТЕЛЬ ПРИНИМАЕТ НА СЕБЯ ВСЮ ОТВЕТСТВЕННОСТЬ В ОТНОШЕНИИ ВЫБОРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ДОСТИЖЕНИЯ СВОИХ ЦЕЛЕЙ, А ТАКЖЕ ЗА

УСТАНОВКУ, ИСПОЛЬЗОВАНИЕ И РЕЗУЛЬТАТЫ ИСПОЛЬЗОВАНИЯ ДАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

Покупатель имеет право:

1. копировать Программное обеспечение исключительно для создания резервных копий или при установке для подразумеваемого обычного использования Программного обеспечения при условии, что в любой копии упомянутого Программного обеспечения будут воспроизведены все уведомления об авторских правах и торговых марках, содержащиеся в данном Программном обеспечении;
2. передавать право владения копиями Программного обеспечения другому юридическому или физическому лицу путем передачи данной копии настоящего Соглашения и всей прочей документации, а также по меньшей мере одной полной и не претерпевший изменений копии Программного обеспечения, при условии, что (1) все сделанные Покупателем копии Программного обеспечения будут переданы означенному лицу или уничтожены, (2) такая передача права владения прекращает лицензионное соглашение Покупателя с МАРАФОНОМ, и (3) означенное лицо примет на себя и будет соблюдать постановления данного лицензионного соглашения с момента начала пользования Программным обеспечением; и
3. использовать торговые марки, связанные с Программным обеспечением, исключительно в соответствии с существующей практикой использования торговых марок, включая ссылки на имена владельцев торговых марок.

Без письменного согласия МАРАФОНА запрещается:

1. использовать, копировать, изменять, объединять или передавать копии данного Программного обеспечения при условиях, отличных от оговоренных в данном соглашении;
2. деассемблировать или декомпилировать Программное обеспечение;
3. выдавать сублицензию, сдавать в аренду и лизинг, передавать в пользование данное Программное обеспечение или любую его копию.

Содержание

1. ВВЕДЕНИЕ

CHAI (CAN Hardware Abstraction Interface) представляет собой библиотеку, реализующую программный интерфейс доступа к сети CAN на канальном уровне (Data Link Layer) эталонной модели ISO/OSI. Программный интерфейс, предоставляемый библиотекой CHAI, не зависит от используемой аппаратуры CAN и операционной системы. Библиотека разработана как для применения во встраиваемых приложениях, так и для задач, работающих под управлением операционных систем общего назначения: Windows XP/Vista/7, Linux и других.

1.1. Возможности CHAI

CHAI поддерживает:

- стандартный и расширенный протокол CAN (11 и 29-битовый идентификаторы кадров),
- скорость передачи до 1000 Kbaud,
- настройку аппаратного фильтра CAN контроллера,
- прием CAN кадров с использованием очереди сообщений,
- регистрацию времени приема кадра (отметка времени),
- обработку событий в CAN контроллере через механизм функций обратного вызова (callbacks),
- несколько CAN-контроллеров в одном устройстве,

1.2. Управление CAN-кадрами

CHAI предоставляет свои собственные способы управления CAN-кадрами, независимые от типа используемого CAN-контроллера.

Полученный из сети CAN-кадр сохраняется в приемной очереди. В очереди сообщения сохраняются последовательно одно за другим по мере их получения, с сохранением последовательности и времени получения (по принципу FIFO). К каждому кадру, помещаемому в очередь, добавляется время его получения (timestamp).

Отправляемые CAN-кадры помещаются в регистры CAN контроллера и отправляются в сеть.

RTR-кадры обрабатываются одинаковым образом с кадрами данных, то есть при приеме помещаются в приемную очередь, при отправке записываются в регистры CAN контроллера.

Приложение может зарегистрировать функцию, которая будет вызываться библиотекой при наступлении определенного события (функции обратного вызова - callbacks). Возможные события:

- получен новый кадр из сети,
- произошла ошибка (Bus-off, Error Warning Limit, аппаратное переполнение контроллера, программное переполнение приемной очереди, тайм-аут при передаче).

2. ОПИСАНИЕ API

Каждый CAN контроллер идентифицируется целым числом (каналом ввода-вывода) начиная с 0. Каждый CAN контроллер (канал ввода-вывода) рассматривается библиотекой как отдельное устройство, вне зависимости от того, что те или иные CAN контроллеры могут быть сгруппированы в одной плате расширения. Например, плата CAN-bus-PCI имеет в своем составе два CAN-контроллера, каждый из которых рассматривается библиотекой CHAI независимо друг от друга. С каждым контроллером (каналом ввода-вывода) ассоциирована одна очередь на прием. Существует аппаратный приемный фильтр, относящийся к CAN-контроллеру в целом, этот фильтр является частью CAN-контроллера. Приемный фильтр позволяет выбирать в CAN-контроллере принимаемые CAN кадры на основе значений их идентификаторов, разгружая таким образом центральный процессор.

Программный интерфейс библиотеки CHAI состоит из трех групп функций:

- Базовые функции, обязательно присутствующие во всех версиях CHAI. Эти функции позволяют настраивать CAN-контроллер и отправлять/принимать CAN-кадры.

- Функции статуса предназначены для определения состояния CAN-контроллера и CAN- адаптера. Эти функции могут отсутствовать в редакции библиотеки для микроконтроллеров из-за ограничений на производительность и память.

- Дополнительные функции, зависящие от типа применяемого CAN-контроллера и платформы, могут быть непереносимы с одной платформы на другую.

Структура данных для представления CAN-кадра имеет следующий прототип:

```
typedef struct {
  _u32 id;          /* идентификатор кадра */
  _u8 data[8];      /* данные */
  _u8 len;          /* фактическая длина поля данных,
                       от 0 до 8 байт */
  _u16 flags;       /* bit 0 - RTR,
                       bit 2 - EFF */
  _u32 ts;          /* отметка времени получения
                     (timestamp) в микросекундах */
} canmsg_t;


Примечания:

если бит 0 поля flags выставлен в 1 – кадр помечен как
RTR

если бит 2 поля flags выставлен в 1 – кадр помечен как
EFF (Extended Frame Format, идентификатор - 29 бит)

время, указываемое в поле ts, измеряется от момента
открытия канала в микросекундах,  переполнение этого
поля наступает примерно через 71 минуту после открытия
канала; после переполнения время вновь отсчитывается от
нуля.
```

Для работы с типом данных canmsg_t определены следующие функции:

- `void msg_zero (canmsg_t *msg);` - обнуляет кадр msg. После вызова кадр msg представляет собой кадр стандартного формата (SFF - standart frame format, идентификатор - 11 бит), с длиной поля данных ноль, данные и все остальные поля выставлены в ноль.

- `_s16 msg_isrtr (canmsg_t *msg);` - возвращает не ноль (true), если msg - RTR кадр.

- `void msg_setrtr (canmsg_t *msg);` - помечает msg как RTR кадр.

- `_s16 msg_iseff (canmsg_t *msg);` - возвращает не ноль (true), если msg - кадр расширенного формата (EFF - extended frame format, идентификатор - 29 бит).

- `void msg_seteff (canmsg_t *msg);` - помечает msg как кадр расширенного формата.

В целях переносимости в заголовочном файле chai.h определены следующие типы данных:

- _u8 - беззнаковое целое длины 8 бит (1 байт)
- _s8 - знаковое целое длины 8 бит (1 байт)
- _u16 - беззнаковое целое длины 16 бит (2 байта)
- _s16 - знаковое целое длины 16 бит (2 байта)
- _u32 - беззнаковое целое длины 32 бит (4 байта)
- _s32 - знаковое целое длины 32 бит (4 байта)

Список функций CHAI:

- Базовые функции, обязательно присутствующие во всех версиях CHAI.

```
o  _s16 CiInit(void);
o  _s16 CiOpen(_u8 chan, _u8 flags);
o  _s16 CiClose(_u8 chan);
o  _s16 CiStart(_u8 chan);
o  _s16 CiStop(_u8 chan);
o  _s16 CiSetFilter(_u8 chan, _u32 acode,
   _u32 amask);
o  _s16 CiSetBaud(_u8 chan, _u8 bt0, _u8
   bt1);
o  _s16 CiWrite(_u8 chan, canmsg_t *mbuf,
   _s16 cnt);
```

- `_s16 CiRead(_u8 chan, canmsg_t *mbuf, _s16 cnt);`
- `_s16 CiSetCB(_u8 chan, _u8 ev, void (*ci_handler) (_s16));`

- Функции статуса. Эти функции могут отсутствовать в реализации для микроконтроллеров из-за ограничений на производительность и память. Если эти функции есть в библиотеке, то в заголовочном файле chai.h определен символ CHAI_STATUS (#define CHAI_STATUS), который можно проверить директивой препроцессора #ifdef CHAI_STATUS.
  - `_u32 CiGetLibVer(void);`
  - `_u32 CiGetDrvVer(void);`
  - `_s16 CiChipStat(_u8 chan, chipstat_t *stat);`
  - `_s16 CiChipStatToStr(chipstat_t * status, chstat_desc_t * desc);`
  - `_s16 CiBoardInfo(canboard_t *binfo);`

- Дополнительные функции, зависят от типа применяемого микроконтроллера и платформы. Дополнительные функции могут быть непереносимы с одной платформы на другую (их состав может меняться). Эти функции могут отсутствовать в конкретной реализации из-за ограничений на производительность и память или из-за свойств CAN-контроллера. Если эти функции есть в библиотеке, то в заголовочном файле chai.h определен символ CHAI_EXTRA (#define CHAI_EXTRA), который можно проверить директивой препроцессора #ifdef CHAI_EXTRA.
  - `_s16 CiWaitEvent(canwait_t * cw, int cwcount, int tout);`
  - `_s16 CiErrsGetClear(_u8 chan, canerrs_t * errs);`
  - `_s16 CiQueResize(_u8 chan, _u16 size);`
  - `_s16 CiRcQueEmpty(_u8 chan);`
  - `_s32 CiRcGetCnt(_u8 chan);`
  - `_s16 CiHwReset(_u8 chan);`
  - `_s16 CiSetLom(_u8 chan, _u8 mode);`

- `_s16 CiSetWriteTout(_u8 chan, _u16 msec);`
- `_s16 CiGetWriteTout(_u8 chan, _u16 * msec);`
- `void CiStrError(_s16 cierrno, char *buf, _s16 n);`
- `void CiPerror(_s16 cierrno, const char *s);`

Функции в случае успешного выполнения возвращают ноль или положительное целое, в случае ошибки отрицательное целое, модуль которого равен коду ошибки определенной в заголовочном файле chai.h:

- `ECIGEN - generic (not specified) error`
- `ECIBUSY - device or resource busy`
- `ECIMFAULT - memory fault`
- `ECISTATE - function can't be called for object in current state`
- `ECIINCALL - invalid call, function can't be called for this object`
- `ECIINVAL - invalid parameter`
- `ECIACCES - can not access resource`
- `ECINORES - no resources`
- `ECINOSYS - function or feature not implemented`
- `ECIIO - input/output error`
- `ECINODEV - no such device`
- `ECIINTR - call was interrupted by event`
- `ECITOUT - time out occurred`

3. РАБОТА С БИБЛИОТЕКОЙ (ПРИМЕР)

Каждая программа, работающая с CHAI должна включать заголовочный файл chai.h, который содержит необходимые для работы с библиотекой, объявления типов данных и прототипов функций.

Типичная последовательность работы с библиотекой выглядит следующим образом:

- инициализация библиотеки (CiInit)
- открытие канала ввода/вывода CAN (CiOpen)
- конфигурирование: CAN-контроллера (CiSetBaud, CiSetFilter), обработчиков событий (CiSetCB).
- запуск CAN-контроллера (CiStart)
- работа с сетью через операции ввода/вывода (CiWrite, CiRead)
- останов CAN-контроллера (CiStop)
- закрытие канала ввода/вывода CAN (CiClose)

Если вам необходимо создать программу, которая работает как на PC, так и в микроконтроллерах, то вы должны ограничиться вызовом функций из первой базовой группы API CHAI, или использовать директивы условной компиляции.

Для иллюстрации работы с библиотекой напишем первую программу - can-echo, которая будет принимать CAN-кадры из сети, и тут же отправлять их обратно в сеть, вычитая из значения идентификатора кадра единицу. Кроме того, программа распечатывает на экране все полученные кадры.

```
/*
 *  can-echocb.c - example program to work with CHAI in callbacks mode.
 *  Program recieve frames from CAN-network (baudrate = 500K)
 *  and send it back.
 *
 *  Copyright (C) 2002-2008 Marathon Ltd. Moscow
 *
 */

#include <stdio.h>
#include <chai.h>     /* includes declarations to work with CHAI */

#ifdef WIN32
#include <windows.h> // for Sleep()
#endif

_u8 chan;

/* callbacks forward definitions */
void cbrcv(_s16 sig);
void cberr(_s16 sig);

/* sleep function forward definitions */
void mysleep(unsigned int millisec);

int main(int argc, char **argv)
{
  int ret;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s X \n", argv[0]);
    fprintf(stderr, "       where X=0,1,2 ...\n");
    exit(1);
  }
  chan = atoi(argv[1]);

  if ((ret = CiInit()) < 0) {
    CiPerror(ret, "CHAI library initialization failed");
    exit(1);
  }

  /*  open channel
   *  After operation complete we have one RC queues (with size
   *  CIQUE_DEFSIZE_RC of frames) with filter that accepts all frames.
   *  CAN controller will accept both 11bit and 29bit identifiers.
   */
  if ((ret = CiOpen(chan, CIO_CAN11 | CIO_CAN29)) < 0) {
    fprintf(stderr, "Error opening CAN channel %s\n", argv[1]);
    CiPerror(ret, "CiOpen");
    exit(1);
  }

  /* set baudrate to 500Kbit */
  if ((ret = CiSetBaud(chan, BCI_500K) < 0)) {
    CiPerror(ret, "can't set baud");
         CiClose(chan);
    exit(1);
  }

  /*  Set callback cbrcv() on CIEV_RC (recieve frame from CAN) event.
```

```
   */
  if ((ret = CiSetCB(chan, CIEV_RC, cbrcv)) < 0) {
    CiPerror(ret, "can't set callback on CIEVRC");
         CiClose(chan);
    exit(1);
  }
  /*  Set callback cberr() on CIEV_CANERR (error occured) event.
   */
  if ((ret = CiSetCB(chan, CIEV_CANERR, cberr)) < 0) {
    CiPerror(ret, "can't set callback on CIEV_CANERR");
         CiClose(chan);
    exit(1);
  }

  /*  start CAN-controller
   *  After operation CAN-controller will be in RUNNING mode,
   *  i.e. it will send ACK and ERROR frames on the bus, so it will be
   *  "visible" by other nodes on the bus. Events start here also.
   */
  CiStart(chan);

  printf("ready to recive/send CAN-frames:\n");

  /* enter to main loop */
  while (1) {
    /* do nothing */
    mysleep(1);
  }
  /* never reach this point */
  exit(0);
}

/*  this callback will be called by CHAI each time
 *  CAN-frame rised from the bus.
 */
void cbrcv(_s16 sig)
{
  canmsg_t rx;
  int ret;
  int i;

  ret = CiRead(chan, &rx, 1);  /* read one frame from channel (RC
queue)*/
  if (ret > 0) {
    if (msg_iseff(&rx))
      printf("EFF ");
    else
      printf("SFF ");                  /* check frame format */
    printf("id=%#lx data:", rx.id);    /* print frame identifier */
    if (msg_isrtr(&rx))                /* check frame is RTR */
      printf(" RTR ");
    else
      for (i = 0; i < rx.len; i++) /* print data field of the frame */
         printf(" %#x", rx.data[i]);
    printf("\n");
    rx.id = rx.id - 1;
    if (CiWrite(chan, &rx, 1) < 1) { /* write modified frame back to
CAN */
      CiPerror(ret, "can't send frame back to CAN");
    }
```

```
  } else if (ret < 0) {
    CiPerror(ret, "error recieving frame from CAN");
  }
}

/*  this callback will be called by CHAI each time
 *  error occured. Callback will recieve event code via
 *  sig argument. Event codes for CIEV_CANERR are
 *  defined in chai.h: CIEV_EWL, CIEV_BOFF, CIEV_HOVR,
 *  CIEV_WTOUT, CIEV_SOVR
 */
void cberr(_s16 sig)
{
  if (sig == CIEV_WTOUT) {
    printf("\n write timeout occured\n");
  } else if (sig == CIEV_EWL) {
    printf("\n Error Warning Limit occured\n");
  } else if (sig == CIEV_BOFF) {
    printf("\n Bus Off occured\n");
  } else if (sig == CIEV_HOVR) {
    printf("\n hardware overrun occured\n");
  } else if (sig == CIEV_SOVR) {
    printf("\n software overrun occured\n");
  }
}

void mysleep(unsigned int millisec)
{
#ifdef WIN32
    Sleep(millisec);
#else // LINUX
    usleep(1000*millisec);
#endif
}
```

В программе мы инициализируем библиотеку (CiInit), затем открываем канал ввода-вывода (CiOpen), номер которого передается программе в качестве параметра. Канал в нашей программе открывается для работы как с 11-битовыми идентификаторами, так и с 29-битовыми идентификаторами (по умолчанию - только 11-битовые идентификаторы, флаг CIO_CAN11). При открытии канала библиотека автоматически создает очередь на прием (длиной CIQUE_DEFSIZE_RC кадров). Функция чтения кадров CiRead() всегда работает в неблокирующем режиме, то есть сразу возвращает столько кадров сколько есть в приемной очереди, но не больше запрошенного количества. Поведение функции отправки кадров CiWrite() зависит от значения таймаута задаваемого при помощи функции CiSetWriteTout(), по умолчанию это

значение задается макроопределением CI_WRITE_TIMEOUT_DEF в файле chai.h. После открытия канала выставляется скорость передачи (CiSetBaud), при этом используется макрос BCI_500K одной из стандартных (стандарт международной организации CAN in Automation) скоростей передачи определенный в chai.h. При помощи функции CiSetCB() в библиотеке регистрируются две функции обратного вызова: void cbrcv(_s16 sig) - будет вызываться при получении кадра из сети, void cberr(_s16 sig) - будет вызываться при возникновении асинхронных ошибок. Затем программа запускает CAN-контроллер (CiStart). Запуск контроллера означает, что он начинает посылать в сеть кадры подтверждения и ошибок и таким образом становится видимым для остальных узлов сети. Затем программа входит в бесконечный цикл, в котором ничего не делается, все остальная работа с сетью происходит в функциях обратного вызова.

Другой пример работы с библиотекой без использования функций обратного вызова находится в файле can-echowt.c в директории ex дистрибутива. Этот пример использует функцию CiWaitEvent() для ожидания события (прием кадра или ошибка сети) и функцию CiErrsGetClear() для чтения ошибок CAN-контроллера.

4. ДЕТАЛЬНОЕ ОПИСАНИЕ ФУНКЦИЙ

4.1. _s16 CiInit(void);

Описание:

Эта функция инициализирует внутренние структуры данных библиотеки, и должна вызываться **только один единственный раз** в самом начале пользовательского приложения до вызова любой другой функции библиотеки CHAI.

Параметры:

Нет.

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

4.2. _s16 CiOpen(_u8 chan, _u8 flags);

Описание:

Открывает (инициализирует) канал ввода-вывода CAN для дальнейшего использования. Эта функция должна вызываться в самом начале работы с каналом. После успешного выполнения (в случае если параметр flags равен нулю) CAN-контроллер канала находится в следующем **начальном состоянии**: контроллер - в CAN_INIT режиме, скорость передачи - 500 Kbit, формат CAN-кадра - SFF (11 бит), значение таймаута на передачу CI_WRITE_TIMEOUT_DEF (chai.h), фильтр контроллера - настроен на прием всех возможных кадров.

Параметры:

- chan - номер открываемого канала
- flags - флаги, Возможные значения: CIO_CAN11 - стандартный формат кадра (идентификатор - 11 бит, по умолчанию), CIO_CAN29 - расширенный формат кадра (идентификатор - 29 бит). Значения флагов можно комбинировать побитовым ИЛИ, например: CIO_CAN11|CIO_CAN29 - это сочетание откроет канал для работы с обоими форматами кадра.

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов;

- ECINODEV - номеру канала, переданному в качестве параметра, не соответствует ни один из обнаруженных CAN-контроллеров;
- ECIBUSY - канал уже открыт (занят), либо не удается установить обработчик аппаратного прерывания;
- ECIMFAULT - не удается выделить память для структур данных канала;

4.3. _s16 CiClose(_u8 chan);

Описание:

Закрывает канал ввода-вывода. После этого вызова уничтожаются все структуры данных ассоциированные с этим каналом и аннулируются все функции - обработчики событий этого канала.

Параметры:

- chan - номер закрываемого канала

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал уже закрыт (не был открыт);
- ECINODEV - номеру канала, переданному в качестве параметра, не соответствует ни один из обнаруженных CAN-контроллеров;

4.4. _s16 CiStart(_u8 chan);

Описание:

Переводит CAN-контроллер канала в рабочее состояние CAN_RUNNING (можно посылать и принимать кадры).

Параметры:

- chan - номер канала

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;

4.5. _s16 CiStop(_u8 chan);

Описание:

Переводит CAN-контроллер канала в CAN_INIT состояние (невозможно посылать и принимать кадры, CAN-контроллер не участвует в обмене данными сети).

Параметры:

- chan - номер канала

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;

4.6. _s16 CiSetFilter(_u8 chan, _u32 acode, _u32 amask);

Описание:

Устанавливает аппаратный фильтр (acceptance filter) CAN-контроллера. Аппаратный фильтр представляется двумя значениями: acode и amask, которые накладываются на идентификатор принимаемого кадра. В amask единицами помечены те позиции, в которых биты идентификатора кадра должны равняться битам acode. Установка фильтра возможна только, когда CAN-контроллер находится в состоянии CAN_INIT (см. вызов CiStop()).

Действие функции CiSetFilter() зависит от того, в каком режиме был открыт канал, а именно от значения параметра flags передаваемого в функцию CiOpen(). **Если канал открыт в режиме CIO_CAN11 или CIO_CAN11|CIO_CAN29 (только 11-битовые идентификаторы или совместно 11-битовые и 29-битовые идентификаторы), тогда CiSetFilter устанавливает значение фильтра только для 11-битовых идентификаторов, при этом действие фильтра на кадры с 29-битовыми идентификаторами не специфицируется. Если необходимо устанавливать фильтр для 29-битовых идентификаторов, необходимо открывать канал в режиме только CIO_CAN29.**

Параметры:

- acode - значение фильтра
- amask - маска фильтра

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECISTATE - канал не находится в состоянии CAN_INIT;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);

4.7. _s16 CiSetBaud(_u8 chan, _u8 bt0, _u8 bt1);

Описание:

Устанавливает скорость передачи кадров CAN-контроллера канала. Установка скорости передачи возможна только, когда CAN-контроллер находится в состоянии CAN_INIT (см. вызов CiStop()).

Параметры:

- bt0 - значение регистра bt0 CAN-контроллера
- bt1 - значение регистра bt1 CAN-контроллера

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECISTATE - канал не находится в состоянии CAN_INIT;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);

Примечание:

В заголовочном файле chai.h определены стандартные (рекомендованные CiA) скорости передач: BCI_1M, BCI_800K, BCI_500K, BCI_250K, BCI_125K, BCI_50K, BCI_20K, BCI_10K.

Эти мнемонические значения содержат в себе сразу значения bt0 и bt1. Поэтому при их использовании вместо двух параметров bt0 и bt1 указывается одно лишь мнемоническое значение скорости. Например, выставить сокрость передачи канала 1 в 500 Kbaud: `CiSetBaud(1, CI_500K);`

4.8. _s16 CiWrite(_u8 chan, canmsg_t *mbuf, _s16 cnt);

Описание:

Отправляет один CAN-кадр в сеть.

Параметры:

- chan - номер канала
- mbuf - указатель на буфер в котором находится CAN-кадр
- cnt – должно равняться единице начиная с версии библиотеки 2.0.0 (этот параметр оставлен для совместимости со старыми версиями)

Возвращаемое значение:

1 - успешное выполнение, (один отправленный кадр)

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECISTATE - канал не находится в состоянии CAN_RUNNING, или попытка отправки кадра неподходящего формата (например, канал был открыт в режиме только CIO_CAN11 и произошла попытка отправки кадра расширенного формата);

- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);
- ECIIO - ошибка ввода-вывода, регисты CAN-контроллера на отправку кадра не освобождаются за положенное время (timeout);

Примечание:

Алгоритм отправки кадра выглядит следующим образом. В самом начале в цикле в течении времени задаваемом значением таймаута на отправку (см. макрос CI_WRITE_TIMEOUT_DEF в заголовочном файле chai.h) проверяется свободен ли аппаратный буфер на отправку. Если таймаут выставлен в ноль, то проверка производится один раз. Если буфер свободен, то кадр загружается в регистры CAN контроллера и выставляется запрос на передачу кадра. Если буфер занят возвращается ошибка ECIIO. Затем если таймаут выставлен в ноль возвращается код успеха (1), в противном случае в течении времени таймаута проверяется успешное окончание отправки. Если отправка успешна возвращается код успеха (1), в противном случае попытки передачи прекращаются (сброс запроса на передачу в контроллере) и возвращается ошибка ECIIO. Из приведенного алгоритма видно, что поведение драйвера несколько отличается в двух случаях:

- таймаут **не равен** нулю: результат реальной отправки или не отправки кадра драйвер возвращает сразу в качестве кода возврата CiWrite(); функция тратит время на ожидание конца операции (busy wait);
- таймаут **равен** нулю: результат реальной отправки или не отправки кадра драйвер возвращает при следующем вызове CiWrite(), то есть если проблемы с сетью начались при отправке кадра N, заметно это станет только при отправке кадра N+1; функция не тратит

время на ожидание конца операции (busy wait отсутствует);

Значение таймаута на отправку может быть изменено при помощи функции CiSetWriteTout() (см. ниже)

4.9. _s16 CiRead(_u8 chan, canmsg_t *mbuf, _s16 cnt);

Описание:

Записывает cnt кадров находящихся в очереди на прием в буфер mbuf. Если в приемной очереди кадров меньше чем cnt, записывает столько кадров сколько есть в очереди. Функция возвращает управление сразу (не ожидает приема из сети запрошенного количества кадров cnt).

Параметры:

- chan - номер канала
- mbuf - указатель на буфер в который будут скопированы кадры
- cnt – запрашиваемое количество CAN-кадров

Возвращаемое значение:

>=0 - успешное выполнение, количество прочитанных кадров

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECISTATE - канал не находится в состоянии CAN_RUNNING;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);

4.10. _s16 CiSetCB(_u8 chan, _u8 ev, void (*ci_handler) (_s16));

Описание:

Регистрирует обработчик события.

Установка обработчика возможна только, когда CAN-контроллер находится в состоянии CAN_INIT (см. вызов CiStop()).

Параметры:

- chan - номер канала
- ev - события, возможные значения определены в заголовочном файле chai.h: CIEV_RC - получен CAN-кадр из сети, CIEV_CANERR - произошла ошибка CAN (в обработчик дополнительно передается тип ошибки в качестве единственного параметра: CIEV_EWL - error warning level, CIEV_BOFF - bus off, CIEV_HOVR - hardware overrun, CIEV_SOVR – software overrun, CIEV_WTOUT – write timeout).
- ci_handler - обработчик события, имеет следующий прототип: void ci_handler(_s16 ev)

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не

был открыт, либо неизвестный тип события передан в качестве параметра;

- ECISTATE - канал не находится в состоянии CAN_INIT;

Примечание: ниже приведено краткое описание ошибок которые могут возникнуть при работе с сетью:

- CIEV_EWL - error warning limit – Error Warning Level, один из аппаратных счетчиков ошибок CAN-контроллера превысил уровень предупреждения (по умолчанию 96); ошибка может возникнуть если нет соединения с сетью (например, обрыв кабеля, нет других контроллеров в сети, несоответствие выставленных скоростей передачи у контроллеров в сети), причиной также могут быть помехи в сети (согласно спецификации Bosch CAN 2.0);

- CIEV_BOFF - bus off – контроллер отключен от сети из-за ошибок (один из счетчиков ошибок достиг предельного значения 255), при возникновении этого события CAN-контроллер автоматически переходит в состояние CAN_INIT;

- CIEV_HOVR - hardware overrun – произошло переполнение аппаратной приемной очереди CAN-контроллера (безвозвратно потерян один или несколько полученных кадров), причина – большой поток кадров в сети, ЦПУ не успевает вынимать кадры из аппаратной очереди контроллера;

- CIEV_SOVR - software overrun – произошло переполнение программной приемной очереди драйвера (безвозратно потерян один или несколько полученных кадров), причина – большой поток кадров в сети, пользовательское приложение (ЦПУ) не

успевает вынимать кадры из программной очереди драйвера;

- CIEV_WTOUT - write timeout – кадр не был отослан в течении стандартного временного интервала (см. функции CiSetWriteTout() и CiWrite()), ошибка может возникнуть если нет соединения с сетью (например, обрыв кабеля, нет других контроллеров в сети, несоответствие выставленных скоростей передачи у контроллеров в сети);

Подробно все ошибки протокола CAN, а также алгоритм обнаружения ошибок в сети CAN описаны в спецификации Bosch (http://can.marathon.ru/files/can2spec.pdf)

Кроме того, необходимо ознакомится со спецификацией используемого контроллера CAN (Phillips SJA1000 - http://can.marathon.ru/files/SJA1000.pdf)

4.11. _u32 CiGetLibVer(void);

Описание:

Возвращает версию библиотеки CHAI как беззнаковое четырехбайтное целое число. Номер версии состоит из трех цифр: major (второй байт) - меняется только при внесении существенных изменений в архитектуру библиотеки, minor (первый байт) - меняется при внесении существенных изменений в реализацию, subminor (нулевой байт). Для получения из 4-байтного числа версии билиотеки major, minor и subminor чисел используйте соответствующие макросы определенные в chai.h: VERMAJ(ver), VERMIN(ver), VERSUB(ver). Эти макросы возвращают знаковое 4-байтное целое.

Параметры:

Нет.

Возвращаемое значение:

>0 - успех, версия библиотеки

0 - ошибка

4.12. _u32 CiGetDrvVer(void);

Описание:

Возвращает версию драйвера unican как беззнаковое четырехбайтное целое число. Номер версии состоит из трех цифр: major (второй байт) - меняется только при внесении существенных изменений в архитектуру драйвера, minor (первый байт) - меняется при внесении существенных изменений в реализацию, subminor (нулевой байт). Для получения из 4-байтного числа версии major, minor и subminor чисел используйте соответствующие макросы определенные в chai.h: VERMAJ(ver), VERMIN(ver), VERSUB(ver). Эти макросы возвращают знаковое 4-байтное целое.

Параметры:

Нет.

Возвращаемое значение:

> 0 - успешное выполнение, версия драйвера

0 - ошибка

4.13. _s16 CiChipStat(_u8 chan, chipstat_t *stat);

Описание:

Возвращает текущее состояние CAN-контроллера. Структура chipstat_t имеет следующий вид:

```
typedef struct {
  int type;               /* тип контроллера */
  int brdnum;             /* номер платы на котором
                             находится контроллер */
  int irq;                /* линия прерывания
                          используемая контроллером */
  unsigned long baddr;    /* базовый адрес
                             контроллера */
  unsigned long hovr_cnt; /* счетчик аппаратных
                             переполнений */
  unsigned long sovr_cnt; /* счетчик программных
                             переполнений */
  char _pad[32];          /* padding для приведения
                       к типу текущего контроллера */
} chipstat_t;
```

В действительности используются другие структуры соответствующие данному типу контроллера, с возможностью вывода значений регистров этого контроллера. Например, для контроллера SJA1000 используется структура sja1000stat_t:

```
Typedef struct {
  int type;
  int brdnum;
  int irq;
  unsigned long baddr;
  unsigned long hovr_cnt;
  unsigned long sovr_cnt;

  /* this registers are readable in all mode */
  unsigned char mode;
  unsigned char stat;
  unsigned char inten;
  unsigned char clkdiv;
  unsigned char ecc;
  unsigned char ewl;
  unsigned char rxec;
  unsigned char txec;
  unsigned char rxmc;
```

```
  /* this registers are readable in init(reset) mode
only */
  unsigned int acode;
  unsigned int amask;
  unsigned char btr0;
  unsigned char btr1;
  unsigned char outctl;
  char _pad[8];
} sja1000stat_t;
```

При использовании функции CiChipStat() необходимо использовать структуру конкретного типа, и при вызове CiChipStat() приводить ее к типу chipstat_t. Например, для SJA1000:

```
_u8 chan = 0;
sja1000stat_t stat;

... /* CiOpen() CiRead() ... etc. */

CiChipStat(chan, (chipstat_t *) &stat);

/* теперь в stat находятся текущие данные по нашему
контроллеру (с значениями регистров) */
...
```

Имейте ввиду, что значения регистров зависят от состояния в котором находится контроллер (CAN_INIT или CAN_RUNNING). Например, для SJA1000 значения полей acode и amask отражают значение фильтра, только если контроллер находится в состоянии CAN_INIT (см. вызов CiStop()).

Параметры:

- chan - номер канала
- stat - указатель на структуру chipstat_t, куда будут записаны данные.

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);

4.14. _s16 CiChipStatToStr(chipstat_t * status, chstat_desc_t * desc);

Описание:

Преобразует бинарное значение структуры данных статуса контроллера в текстовое описание статуса в формате ASCII. Структура данных chstat_desc_t имеет вид:

```
typedef struct {
  char name[CI_CHSTAT_STRNUM][CI_CHSTAT_MAXLEN];
  char val[CI_CHSTAT_STRNUM][CI_CHSTAT_MAXLEN];
} chstat_desc_t;
```

Функция заполняет эту структуру размещая в полях name название соответствующего поля структуры chipstat_t, а в полях val соответствующее значение. Пример использования:

```
chstat_desc_t desc;
chipstat_t st;
int i;
...
CiChipStat(chan, &st);
CiChipStatToStr(&st, &desc);
i = 0;
while (desc.name[i][0] != '\0') {
  printf("%-12s: %s\n", desc.name[i], desc.val[i]);
  i++;
}
```

Параметры:

- status – входной параметр, указатель на структуру статуса CAN контроллера заполненную функцией CiChipStat();
- desc – выходной параметр, указатель на структуру данных chstat_desc_t;

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

4.15. _s16 CiBoardInfo(canboard_t *binfo);

Описание:

Выдает информацию о плате. Структура canboard_t имеет следующий вид:

```
typedef struct {
  _u8 brdnum;        /* номер платы (от 0 до CI_BRD_NUMS-1)*/
  _u32 hwver;        /* номер версии железа (аналогичен по структуре
номеру версии библиотеки */
  _s16 chip[4];      /* массив номеров каналов (например chip[0]
содержит номер канала к которому
                      привязан первый чип платы, если номер <0 -
чип отсутствует)*/
  char name[64];     /* текстовая строка названия платы */
  char manufact[64]; /* текстовая строка - имя производителя */
} canboard_t;
```

Поле brdnum является входным параметром и определяет для какой платы (номер) выдать информацию. Пример:

```
canboard_t binfo;
...
binfo.brdnum = 1;
CiBoardInfo(&binfo);
...
```

Параметры:

- binfo - указатель на структуру canboard_t

Возвращаемое значение:

>=0 - успешное выполнение (кол-во обнаруженных каналов ввода-вывода)

< 0 – ошибка

возможные ошибки:

- ECIINVAL - номер платы, переданный в качестве параметра, выходит за пределы поддерживаемого числа плат;

- ECINODEV - номеру платы, переданному в качестве параметра, не соответствует ни одна из обнаруженных плат;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);
- ECIBUSY - вызов не может быть выполнен, поскольку все каналы заняты другими процессами;

Примечание: данная функция может использоваться для определения доступных системе аппаратных CAN-интерфейсов и соответствующих им каналов ввода-вывода CAN. Например:

```
int print_binfo(void)
{
  _s16 i, j, ret;
  canboard_t binfo;
  int cnt = 0;

  for (i = 0; i < CI_BRD_NUMS; i++) {
    binfo.brdnum = (_u8) i;
    ret = CiBoardInfo(&binfo);
    if (ret < 0)
      continue;
    printf("%s (%s): \n", binfo.name, binfo.manufact);
    for (j = 0; j < 4; j++) {
      if (binfo.chip[j] >= 0) {
         printf("   channel %d\n", binfo.chip[j]);
         cnt++;
      }
    }
  }
  return cnt;
}
```

Приведенная выше функция print_binfo() распечатывает на экране все обнаруженные библиотекой CHAI аппаратные CAN-интерфейсы и каналы ввода-вывода им соответствующие. Количество обнаруженных каналов ввода-вывода возвращается в качестве результата работы функции.

4.16. _s16 CiBoardGetSerial(_u8 brdnum, char *sbuf, _u16 bufsize)

Описание:

Выдает серийный номер платы в виде строки. *Примечание:* в данный момент серийные номера поддерживаются только для устройств CAN-bus-USBnp (CAN-bus-USBnps), для всех остальных типов устройств вызов возвращает ошибку ECINOSYS.

Параметры:

- brdnum — номер платы (аналогично binfo.brdnum вызова CiBoardInfo);
- sbuf – указатель на массив пользователя, в который будет скопирован серийный номер;
- bufsize – длина массива sbuf;

Возвращаемое значение:

0 - успешное выполнение

< 0 – ошибка

возможные ошибки:

- ECIINVAL - номер платы, переданный в качестве параметра, выходит за пределы поддерживаемого числа плат;
- ECINODEV - номеру платы, переданному в качестве параметра, не соответствует ни одна из обнаруженных плат;
- ECINOSYS — данный тип платы не поддерживает серийный номер;

- ECIBUSY - вызов не может быть выполнен, поскольку все каналы заняты другими процессами;

4.17. _s16 CiErrsGetClear(_u8 chan, canerrs_t * errs);

Описание:

Возвращает значение программных счетчиков ошибок CHAI и сбрасывает их в ноль. Счетчики ошибок возвращаются в структуре canerrs_t которая имеет следующий вид:

```
typedef struct {
  _u16 ewl;    // кол-во ошибок EWL
  _u16 boff;   // кол-во ошибок BOFF
  _u16 hwovr;  // кол-во ошибок HOVR
  _u16 swovr;  // кол-во ошибок SOVR
  _u16 wtout;  // кол-во ошибок WTOUT
} canerrs_t;
```

При открытии канала счетчики равны нулю и непрерывно увеличиваются на 1 при возникновении соответствующей ошибки.

Параметры:

- chan - номер канала
- errs - указатель на структуру canerrs_t, куда будут записаны данные.

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);

4.18. _s16 CiWaitEvent(canwait_t * cw, int cwcount, int tout);

Описание:

Блокирует выполнение программы (потока выполнения) до наступления заданного события (получен CAN-кадр, ошибка) или до наступления таймаута в одном из указанных каналов ввода-вывода CAN. Каналы ввода-вывода и интересующие в них события задаются с помощью массива структур canwait_t, которая определена следующим образом:

```
typedef struct {
  _u8 chan;   // номер открытого канала
  _u8 wflags; // флаги интересующих нас событий
  _u8 rflags; // флаги наступивших событий (результат выполнения)
} canwait_t;
```

Пример использования:

```
canwait_t cw[2];

/* открываем и инициализируем два канала */
...
cw[0].chan = chan1;
cw[0].wflags = CI_WAIT_RC | CI_WAIT_ER;
cw[1].chan = chan2;
cw[1].wflags = CI_WAIT_RC | CI_WAIT_ER;

CiStart(chan1);
CiStart(chan2);

while (1) {
  ret = CiWaitEvent(cw, 2, 1000); // timeout = 1000 миллисекунд
  if (ret < 0) {
     // ошибка CiWaitEvent()
  }
  if (ret == 0) { // timeout (nothing happen)
    continue;
  }
  for (i=0; i<2; i++) {
    if (cw[i].rflags & CI_WAIT_RC) {
      // пришел кадр в канал cw[i].chan
      // читаем кадры с помощью CiRead();
    }
    if (cw[i].rflags & CI_WAIT_ER) {
      // в канале cw[i].chan произошла ошибка
      // читаем ошибки с помощью CiErrsGetClear()
    }
  }
}
```

Параметры:

- cw – указатель на массив структур canwait_t пользователя;
- cwcount – количество элементов в массиве;
- tout – таймаут в миллисекундах, если равен -1 (минус единица), то таймаут принимается равным бесконечности;

Возвращаемое значение:

> 0 - успешное выполнение: в каких либо из запрошенных каналах произошли какие-либо из указанных событий

0 - таймаут

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);
- ECIGEN – ошибка операционной системы;

4.19. _s16 CiHwReset(_u8 chan);

Описание:

Выполняет аппаратный сброс CAN-контроллера. При этом сбрасываются счетчики ошибок CAN-контроллера, обнуляются программные счетчики аппаратных и программных переполнений. Значения аппаратного фильтра и скорости передачи сохраняются. Состояние CAN-контроллера CAN_INIT или CAN_RUNNING также сохраняется.

Параметры:

- chan - номер канала

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);

4.20. _s16 CiSetWriteTout(_u8 chan, _u16 msec);

Описание:

Устанавливает таймаут в миллисекундах на отправку кадра в сеть через канал chan. Описание алгоритма оправки кадра смотрите в описании функции CiWrite().

Вызов этой функции возможен только, когда CAN-контроллер находится в состоянии CAN_INIT (см. вызов CiStop()).

Параметры:

- chan - номер канала
- msec – значение таймаута в миллисекундах, значение должно быть в пределах 0 - CI_WRITE_TIMEOUT_MAX миллисекунд, по умолчанию при открытии канала таймаут на отправку равен CI_WRITE_TIMEOUT_DEF миллисекунд (CI_WRITE_TIMEOUT_DEF и CI_WRITE_TIMEOUT_MAX определены в chai.h).

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов; канал не был открыт; запрошенное значение таймаута выходит за допустимые пределы;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);

- ECISTATE - контроллер не находится в состоянии CAN_INIT;

4.21. _s16 CiGetWriteTout(_u8 chan, _u16 * msec);

Описание:

Возвращает текущее значение таймаута в миллисекундах на отправку кадра в сеть через канал chan.

Вызов этой функции возможен только, когда CAN-контроллер находится в состоянии CAN_INIT (см. вызов CiStop()).

Параметры:

- chan - номер канала
- msec – указатель на переменную пользователя куда будет сохранено знаяение в миллисекундах.

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов; канал не был открыт; запрошенное значение таймаута выходит за допустимые пределы;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);
- ECISTATE - контроллер не находится в состоянии CAN_INIT;

4.22. _s32 CiRcGetCnt(_u8 chan);

Описание:

Возвращает количество кадров находящихся в приемной очереди драйвера.

Параметры:

- chan - номер канала

Возвращаемое значение:

>= 0 – количество кадров в приемной очереди

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);

4.23. _s16 CiRcQueEmpty(_u8 chan);

Описание:

Принудительно очищает (стирает) содержимое приемной очереди канала.

Примечание: данная функция не должна применяться при “нормальной” работе с сетью CAN, так как функция CiRead() вынимает полученные кадры из приемной очереди. CiRcQueEmpty() должна использоваться только в случае обработки возможных “аварийных” ситуаций, когда необходимо очистить приемную очередь от неинтересующего уже содержимого.

Параметры:

- chan - номер канала

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;

4.24. _s16 CiSetLom(_u8 chan, _u8 mode);

Описание:

Переключает CAN-контроллер в режим Listen Only Mode и обратно. В режиме Listen Only Mode контроллер принимает все кадры из сети, но не посылает в сеть подтверждения, ошибок и т.п. Таким образом в этом режиме контроллер невидим для остальных узлов сети и никак не влияет на работу сети в целом. Режим Listen Only Mode используется для работы различного рода тестирующих и следящих за работой сети приложений. Посылка кадров в этом режиме невозможна.

Вызов этой функции возможен только, когда CAN-контроллер находится в состоянии CAN_INIT (см. вызов CiStop()).

В версиях библиотеки 1.x.x вместо этой функции использовались функции CiSJA1000SetLom() и CiSJA1000ClearLom(); эти функции сохранены для обратной совместимости, однако в новых версиях программ следует использовать CiSetLom().

Параметры:

- chan - номер канала
- mode – возможные значения: CI_LOM_ON включает режим, CI_LOM_OFF выключает режим; CI_LOM_ON/ CI_LOM_OFF определены в заголовочном файле chai.h

Возвращаемое значение:

0 - успешное выполнение

< 0 - ошибка

возможные ошибки:

- ECIINVAL - номер канала, переданный в качестве параметра, выходит за пределы поддерживаемого числа каналов, либо канал не был открыт;
- ECIMFAULT - системная ошибка памяти (не удается скопировать параметры или результаты);
- ECISTATE - контроллер не находится в состоянии CAN_INIT;

4.25. void CiStrError(_s16 cierrno, char *buf, _s16 n);

Описание:

Возвращает в буфере, на который указывает указатель buf, текстовое описание ошибки, номер которой содержит параметр cierrno.

Параметры:

- cierrno - номер ошибки CHAI (может быть отрицательным числом)
- buf - указатель на буфер в котором будет сохранено текстовое описание ошибки
- n - длина буфера в байтах (символах)

Возвращаемое значение:

нет

4.26. void CiPerror(_s16 cierrno, const char *s);

Описание:

Печатает в стандартный поток вывода ошибок сообщение s, а затем текстовое описание ошибки cierrno.

Параметры:

- cierrno - номер ошибки CHAI (может быть отрицательным числом)
- s - сообщение

Возвращаемое значение:

нет

5. ИЗМЕНЕНИЯ В API CHAI 2.X.X

В версиях 2.x.x библиотеки CHAI внесены следующие основные изменения во внешний интерфейс по сравнению с версиями 1.x.x:

- удален блокирующий режим работы с каналом ввода-ввода; флаг CIO_BLOCK вызова CiOpen() оставлен для совместимости, но полностью игнорируется;
- Функция CiWrite() отправляет только один кадр, параметр cnt вызова CiWrite() сохранен для совместимости, но должен быть всегда равен 1;
- Функция CiRead() работает в режиме совместимом с неблокирующим режимом версии 1.x.x, то есть возвращает столько кадров сколько есть в очереди но не больше запрошенного количества;
- флаги MSG_HOVR, MSG_SOVR поля flags структуры данных кадра canmsg_t полностью удалены; функции msg_ishovr() и msg_issovr() оставлены для совместимости, но всегда возвращают FALSE;
- добавлена функция CiErrsGetClear(), которая позволяет читать ошибки CAN в режиме опроса без использования функций обратного вызова;
- добавлена функция CiWaitEvent(), которая позволяет блокировать работу потока выполнения до наступления события CAN (получен кадр, произошла ошибка) без использования режима занятого ожидания (busy wait);
- добавлена функция CiSetLom() заменяющая фукнциональность двух функций CiSJA1000SetLom() и CiSJA1000ClearLom(), которые полностью сохранены для обратной совместимости;

- добавлены функции CiChipStatToStr(), CiGetWriteTout();

- в версии CHAI 2.3.0 добавлена функция CiBoardGetSerial().

6. ОСОБЕННОСТИ РЕАЛИЗАЦИИ БИБЛИОТЕКИ

6.1. ОС Linux

В ОС Linux для передачи событий CIEV_RC, CIEV_EWL, CIEV_BOFF, CIEV_HOVR, CIEV_WTOUT, CIEV_SOVR используются сигналы реального времени. Система гарантирует “надежную” доставку этих сигналов и поддерживает очередь сигналов (если сигнал возникает в момент работы обработчика, то сигнал не пропадает а запоминается в очереди). Функции обратного вызова обработки событий библиотеки CHAI (см. функцию CiSetCB()) выполняются в контексте обработчика сигналов реального времени ОС Linux. Используется одна общая функция обработчика для всех вышеуказанных сигналов, на время работы обработчика получение остальных сигналов блокируется (возникшие сигналы запоминаются в очереди операционной системой, и доставляются после окончания работы обработчика). Таким образом, в каждый момент времени может выполнятся только одна функция обратного вызова, или, говоря другими словами, функции обратного вызова не обязательно должны быть повторно-входимыми. Очевидно, что при такой схеме функции обратного вызова не должны блокировать свою работу на продолжительное время, поскольку любая блокировка или ожидание в функции обратного вызова приводит к блокированию получения других событий библиотеки.

Для реализации функции CiWaitEvent() в Linux используется системный вызов poll().

6.2. ОС Windows

Начиная с CHAI версии 1.5.0 драйвер для CAN-интерфейсов шин ISA и PCI в ОС Windows написан с использованием среды разработки KMDF (Kernel Mode Driver Foundation) фирмы Microsoft и работает в пространстве ядра. Для передачи событий CIEV_RC, CIEV_EWL, CIEV_BOFF, CIEV_HOVR, CIEV_WTOUT, CIEV_SOVR используется схема с отдельным потоком выполнения (thread). Для каждого канала в рамках процесса запускается отдельный поток выполнения в контексте которого выполняются функции обратного вызова библиотеки CHAI (см. функцию CiSetCB()) для вышеуказанных событий. Используется один поток выполнения для всех вышеуказанных сигналов. Таким образом, в каждый момент времени может выполнятся только одна функция обратного вызова, или, говоря другими словами, функции обратного вызова не обязательно должны быть повторно-входимыми. Очевидно, что при такой схеме функции обратного вызова не должны блокировать свою работу на продолжительное время, поскольку любая блокировка или ожидание в функции обратного вызова приводит к блокированию получения других событий библиотеки.

Для реализации функции CiWaitEvent() в Windows используется системный вызов WaitForMultipleObjects().