МАРАФОН
CAN технология

CANopen библиотека

+

Адаптированный **слейв** для ОС Windows

Руководство программиста

Код проекта: **0002**$_h$

Москва, 2023

Оглавление

1. Общие положения

Документ является составной частью руководства по CANopen библиотеке (свидетельство о государственной регистрации программы для ЭВМ № 2022610093). В нем приводится описание адаптированной слейв версии библиотеки и приложений для ОС Windows. Адаптированная версия поддерживает протокол EN50325-5 (функционально безопасные коммуникации на основе CANopen, ведомый), который не включен в основной код CANopen библиотеки. Адаптированный слейв может использоваться в качестве прототипа для реализации CANopen устройств на микроконтроллерных платформах.

Программа выполнена в виде двух частей: библиотечной и прикладной. Общая библиотечная часть может использоваться для разработки различных проектов (устройств) на микроконтроллерных платформах. Прикладная реализует CANopen профиль устройства. Адаптированная версия поддерживает CANopen профиль CiA 401 (модули ввода-вывода общего назначения).
Библиотечные модули адаптированной версии размещаются в директории CANopen_WinLib_EN50325-5. Модули приложения размещены в директории CANopen_WinAppl_CiA401.
Адаптированная версия поддерживает только ускоренный и сегментированный SDO протоколы. При этом обеспечивается возможность передачи данных переменной длины (visible string, octet string). Формируется единственный обязательный SDO параметр сервера, который задается по умолчанию.
Для слейв устройств возможна работа по нескольким CAN шинам (до восьми) в режиме ”холодного” резервирования.
Поддерживаются только байтовые (группами по 8 бит) динамическое PDO и SRDO отображения. Поддерживается только little-endian формат, когда наименее значимый (младший) байт данных любого стандартного типа размещается по меньшему адресу.

Дополнением к данному описанию являются документы:
CANopen_CiA401_Framework.pdf – программная архитектура CANopen, профиль CiA 401;
CHAI_MCU_Driver.pdf – CAN драйвер CHAI для микроконтроллерных платформ.

Следующие модули адаптированной версии требуют доработки для целевой микроконтроллерной платформы:
- драйвер канального уровня CAN с базовым API CHAI;
- can_system_windows.c системно-зависимые функции (таймер, критические секции и др.)

1.1 Наименование основных типов данных

boolean	Логическое значение false / true.
integer8, int8	Целое 8 разрядов со знаком.
unsigned8	Без-знаковое целое 8 разрядов.
integer16, int16	Целое 16 бит со знаком.
unsigned16	Без-знаковое целое 16 бит.
integer32, int31	Целое 32 бита со знаком.
unsigned32	Без-знаковое целое 32 бита.
integer64, int64	Целое 64 бита со знаком.
unsigned64	Без-знаковое целое 64 бита.
real32	32-х разрядное с плавающей точкой.
real64	64-х разрядное с плавающей точкой.
vis-string	Строка видимых ASCII символов (коды 0 и 20_h..$7E_h$).
octet-string	Байтовая строка (коды 0..255).

1.2 Прочие соглашения

1. Размер байта данных составляет 8 (восемь) бит.
2. Наименее значимый (младший) байт данных любого стандартного типа размещается по меньшему адресу (little-endian).
3. Шестнадцатеричный формат данных всегда указывается явно (h, hex). При отсутствии указания hex число представлено в десятичном формате. Этот формат может быть также указан явно (d, dec).
4. Индексы и субиндексы объектного словаря CANopen указываются в шестнадцатеричном виде (hex).
5. Объекты CANopen записываются в формате $1234_h sub1_h$ или 1234_h с указанием индекса и субиндекса объектного словаря.

1.3 Обновление терминологии

Международные организации CAN in Automation и Society of Automotive Engineers приняли совместное решение использовать термины “commander” вместо “master” и “responder” вместо “slave”. Переход к обновленной терминологии будет осуществляться по мере внесения правок в документацию.

Оригинальное сообщение на английском языке, декабрь 2020 г:
«
CiA and SAE have decided to use “commander” and “responder” instead of “master” respectively ”slave” in combination with “network”, “device”, and “node”. Both organizations are committed to use inclusive language in their specifications.
»

2. Изменения в версиях программ

Версия CANopen библиотеки 2.2

В состав библиотеки включена адаптированная слейв версия для ОС Windows.

Версия CANopen библиотеки 2.3

Адаптированный CANopen слейв реализован на основе версии 2.3 CANopen библиотеки.

Версия CANopen библиотеки 3.0

Версия CANopen слейв сформирована в рамках реализации стандарта EN50325-5: функционально безопасные коммуникации на основе CANopen.

Версия приложения 1.2

Часть функций приложения перенесена в новый модуль DS401_control.c. Внесены изменения в код некоторых функций приложения.

Версия приложения 2.1

Реализует набор программ с поддержкой протокола EN50325-5 и прикладного CANopen профиля CiA 401. Поддерживается единый код приложения для программ-эмуляторов и встроенного ПО для микроконтроллерных платформ.

Версия приложения 3.0

Обновленное приложение с поддержкой эталонного профиля CiA 401.

2.1 Управление версиями адаптированных модулей

Каждый библиотечный модуль адаптированной версии заключается в условный макрос вида
```
#if CHECK_VERSION_CANLIB(3, 0, 1)
        код адаптированного модуля CANopen
#endif
```
Аргументы макроса фиксируют версию модуля CANopen библиотеки, на основе которого сформирован код адаптированной версии.

Аналогичные макросы используются для контроля версий приложения
```
#if CHECK_VERSION_APPL(3, 0, 0)
        код модуля приложения
#endif
```

3. Типы и структуры данных адаптированной версии

3.1 Типы данных CANopen библиотеки

Обозначение	Тип данных	Описание
canbyte	**unsigned8**	Без-знаковое целое 8 бит.
cannode	**unsigned8**	Без-знаковое целое 8 бит, идентификатор CAN узла.
canindex	**unsigned16**	Без-знаковое целое 16 бит, индекс объектного словаря.
cansubind	**unsigned8**	Без-знаковое целое 8 бит, субиндекс объектного словаря.
canlink	**unsigned16**	Без-знаковое целое 16 бит, CAN идентификатор канального уровня для 11 битового CAN–ID.

3.2 Структуры данных CANopen библиотеки

```
union cansdob0 {
        struct segm {
    unsigned8 ndata;    число байт в сегменте, которые не содержат данных.
    unsigned8 bit_0;    бит 0 нулевого байта сегмента.
    unsigned8 bit_1;    бит 1 нулевого байта сегмента.
    unsigned8 toggle;   значение мерцающего бита.
        } sg;
};
```

Структура **segm** объединения **cansdob0** заполняется по итогам разбора управляющего (нулевого) байта данных ускоренного и сегментированного SDO протоколов.

```
struct sdoixs {
    canindex index;     индекс прикладного объекта.
    cansubind subind;   субиндекс прикладного объекта.
};
```

Структура **sdoixs** определяет индекс и субиндекс прикладного CANopen объекта для SDO протокола (мультиплексор SDO протокола).

```
struct cansdo {
    unsigned8 cs;           команда SDO протокола.
    struct sdoixs si;       индекс и субиндекс прикладного объекта (мультиплексор SDO
                            протокола).
    union cansdob0 b0;      управляющий байт SDO протокола.
    canbyte bd[8];          прикладные данные CAN кадра SDO протокола.
};
```

Структура **cansdo** размещает информацию SDO кадра в разобранном виде.

```
struct sdosrvfull {
    int16 busy;             семафор занятия буфера (инкрементный).
    unsigned8 capture;      флаг захвата буфера.
    unsigned32 timeout;     таймаут операции обмена одним сегментом данных в рамках
                            SDO протокола (базовой транзакции).
    unsigned8 toggle;       ожидаемое значение мерцающего бита.
    struct sdoixs si;       индекс и субиндекс прикладного объекта (мультиплексор SDO
                            протокола).
```

canbyte *bufpnt; указатель на текущую позицию в буфере принимаемых или передаваемых данных.
unsigned32 rembytes; число оставшихся для передачи байт объекта.
};

Структура **sdosrvfull** размещает данные, необходимые для реализации SDO транзакции на стороне сервера.

struct cancache {
int16 busy; семафор занятия кэша (инкрементный).
unsigned8 capture; флаг захвата кэша и занесения в него данных.
canframe cf; CAN кадр канального уровня.
};

Структура **cancache** формирует кэш для размещения отправляемых CAN кадров.

struct canframe {
unsigned32 id; CAN–ID.
unsigned8 data[8]; поле данные CAN кадра.
unsigned8 len; фактическая длина данных (от 0 до 8).
unsigned16 flg; битовые флаги CAN кадра. Бит 0 – RTR, бит 2 – EFF.
unsigned32 ts; временн*а*я метка получения CAN кадра в микросекундах.
};

Структура **canframe** размещает CAN кадр канального уровня. Ее определение содержится в заголовочном файле CAN драйвера CHAI (структура **canmsg_t**).

struct pargroup {
unsigned16 index; Единый индекс прикладных объектов.
unsigned8 subfirst; Первый субиндекс прикладных объектов.
unsigned8 sublast; Последний субиндекс прикладных объектов.
};

Структура **pargroup** содержит описание прикладных объектов для сохранения в энергонезависимой памяти. Для каждого индекса прикладных объектов формируется отдельный блок данных. Массив структур **pargroup** задает группу объектов, которые сохраняется и восстанавливается совместно. Как правило, определяется с квалификатором const.

struct srdodata {
unsigned8 len; фактическая длина данных (от 0 до 8).
canbyte data[CAN_DATALEN_MAXIMUM]; поле данных CAN кадра.
};

Структура **srdodata** размещает данные CAN кадра протокола EN50325-5.

struct srdostruct {
unsigned8 dir; Направление передачи SRDO.
unsigned8 srvt; Контрольное время SRVT для принимаемого SRDO.
unsigned8 trtype; Тип передачи SRDO.
unsigned8 mask; Битовая маска принятых кадров SRDO.
unsigned16 sct; Период обновления передаваемого SRDO.
Охранное время для принимаемого SRDO.
unsigned32 id_odd; CAN-ID 1, нечетный (SRDO в основном формате).
unsigned32 id_even; CAN-ID 2, четный (SRDO в побитно-инвертированном формате).
unsigned16 signature; Подпись (CRC) SRDO параметров.
unsigned8 config; Конфигурация SRDO объекта достоверна / не достоверна.
unsigned8 nrec; Число объектов SRDO отображения.

unsigned32 map[...];	Объекты SRDO отображения в основном формате.
int32 srvt_cnt;	SRVT счетчик. Со знаком для асинхронного сброса.
int32 sct_cnt;	SCT счетчик. Со знаком для асинхронного сброса.
int32 discard_cnt;	Счетчик мертвого времени принимаемого SRDO. Со знаком для асинхронного сброса.
struct srdodata plain;	Данные принимаемого SRDO в основном формате.
struct srdodata bwinv;	Данные принимаемого SRDO в побитно-инвертированном формате.

};

Структура **srdostruct** размещает данные SRDO объекта, которые используются в протоколе EN50325-5.

3.3 Глобальные данные CANopen библиотеки

Обозначение	Тип данных	Описание
netmask_work	unsigned8	Битовая маска рабочих CAN сетей (объект $11F0_h sub3_h$).
can_netchan	unsigned8	Номер активной CAN сети (от 0 до 7, объект $11F0_h sub4_h$).
node_id	unsigned8	Номер CAN узла (от 1 до 127).
node_state	unsigned8	NMT состояние CAN узла.

3.4 Структуры данных приложения

struct object {

canindex index;	индекс объекта.
cansubind subind;	субиндекс объекта.
int32 size;	размер объекта в байтах (>= 0) или код ошибки (< 0).
canindex deftype;	индекс типа объекта.
int16 access;	маска доступа к объекту.
canbyte *pnt;	указатель на размещение объекта.

};

Структура **object** содержит обобщенное описание прикладного объекта.

union numbers {

unsigned64	**init**	служит для инициализации объединения и переноса данных.
unsigned32	**dt32[2]**	используются для unsigned32 доступа к данным.
unsigned16	**dt16[4]**	используются для unsigned16 (например ModBus) доступа.
unsigned8	**dt8[8]**	используются для CANopen доступа.
int8	**i8**	целое 8 бит со знаком.
unsigned8	**uns8**	без-знаковое целое 8 бит. Либо булево значение false / true.
int16	**i16**	целое 16 бит со знаком.
unsigned16	**uns16**	без-знаковое целое 16 бит.
int32	**i32**	целое 32 бита со знаком.
unsigned32	**uns32**	без-знаковое целое 32 бита.
int64	**i64**	целое 64 бита со знаком.
unsigned64	**uns64**	без-знаковое целое 64 бита.
real32	**re32**	с плавающей точкой одинарной точности (float).
real64	**re64**	с плавающей точкой двойной точности (double).

};

Объединение **numbers** служит для единого представления различных типов численных

данных.

Замечание.

Согласованность данных в объединении **numbers** обеспечивается только для little-endian порядка байт и лишь в случае, когда размер байта составляет 8 бит.

3.5 Глобальные данные приложения

Обозначение	Тип данных	Описание
error_state	boolean	Состояние ошибки устройства.
iwdt_tim_state	unsigned8	Статус CANopen таймера и трафика TIME для перезагрузки сторожевого таймера (используется в микроконтроллерных приложениях).
nodeid_eeprom	unsigned8	Номер CAN узла для сохранения в энергонезависимой памяти (объект 2110_h).
brindex_eeprom	unsigned8	Индекс битовой скорости для сохранения в энергонезависимой памяти (объект 2111_h).
digital_inblocks **digital_outblocks**	unsigned8	Число входных цифровых блоков по 8 бит (от 0 до 2). Число выходных цифровых блоков по 8 бит (от 0 до 2).
analog_inputs **analog_outputs**	unsigned8	Число аналоговых входов (от 0 до 8). Число аналоговых выходов (от 0 до 8).
diginp_tpdo_forced	boolean	Режим передачи TPDO от цифровых входов

4. Программные модули CANopen

4.1 Функции доступа к объектному словарю

Доступ к объектному словарю CANopen осуществляется с использованием функций, имеющих единый вид прикладного интерфейса. В API приведены два примера функций. Один вид (*_deftype_*) поддерживает объекты определения типов данных (индексы $0001_h..001F_h$). Другой (*_standev_*) используется для доступа к объектам стандартизированных устройств (индексы $6000_h..9FFF_h$). Для объектов переменного размера дополнительно применяется функция server_put_object_size(...).

int16 get_deftype_bytes_objsize(canindex index, cansubind subind);
int32 server_get_standev_objsize(canindex index, cansubind subind);

Запрос размера объекта в байтах.

Эта функция также определяет наличие соответствующего объекта в словаре. В случае ошибки, например, не существующего объекта, возвращается отрицательное значение. Для коммуникационных объектов может возвращать значения типа int16. Прикладные объекты и диспетчер доступа к компонентам объектного словаря возвращают значения типа int32. Особым случаем является возврат нулевого значения. Для данных переменной длины это одно из допустимых значений, а в случае фиксированного размера объекта является ошибкой.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.

Возвращаемые значения: размер объекта >= 0; ошибка < 0:

- >= 0 – размер объекта в байтах. Для объектов переменной длины может принимать нулевое значение.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int32 server_put_object_size(canindex index, cansubind subind, int32 size);

Запись размера объекта переменной длины.

Используется, когда размер объекта не известен заранее и передается клиентом при инициализации обмена данными.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- **size** – размер объекта в байтах.

Возвращаемые значения: размер объекта > 0; ошибка < 0:

- > 0 – записанный размер объекта в байтах.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.
- CAN_ERRET_OBD_DATAMISM – невозможно принять и записать размер объекта.
- CAN_ERRET_OBD_DATALOW – принятый размер объекта менее допустимого.
- CAN_ERRET_OBD_DATAHIGH – принятый размер объекта более допустимого.

int16 see_deftype_access(canindex index, cansubind subind);
int16 server_see_standev_access(canindex index, cansubind subind);

Запрос маски доступа к записи объектного словаря.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.

Возвращаемые значения: маска доступа > 0 (биты 0..14); ошибка = 0 или < 0 (установлен бит 15).

- CAN_MASK_ACCESS_PDO – флаг допустимости PDO отображения объекта (бит_0 = 1).
- CAN_MASK_ACCESS_RO – флаг доступа к объекту по чтению (бит_1 = 1).
- CAN_MASK_ACCESS_WO – флаг доступа к объекту по записи (бит_2 = 1).
- CAN_MASK_ACCESS_RW – доступ к объекту по чтению и записи (бит_1 = 1 и бит_2 = 1).
- = 0 – нет доступа к объекту.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int16 get_deftype_objtype(canindex index, cansubind subind);
int16 server_get_standev_objtype(canindex index, cansubind subind);

Запрос индекса типа объекта из словаря устройства.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.

Возвращаемые значения: индекс типа объекта > 0; ошибка <= 0:

- > 0 – индекс типа объекта (0001_h..$001F_h$: статические типы данных объектного словаря).
- = 0 – ошибка, объект со значением индекса 0000_h относится CiA 301 к не используемым.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int16 read_deftype_objdict(canindex index, cansubind subind, canbyte *data);
int16 server_read_standev_objdict(canindex index, cansubind subind, canbyte *data);

Чтение значения объекта из словаря устройства.

Объект преобразуется в байтовый вид и размещается по адресу ***data**. Приложение должно выделить буфер, достаточный для размещения всех данных. Размер объекта при необходимости может быть определен с помощью функции server_get_object_size(...).

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- ***data** – байтовый указатель на размещаемые данные.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.
- CAN_ERRET_OBD_WRITEONLY – попытка чтения объекта с доступом только по записи.
- CAN_ERRET_NULL_POINTER – указатель на объект — NULL. Может являться следствием ошибок при описании объекта.

int16 write_deftype_objdict(canindex index, cansubind subind, canbyte *data);
int16 server_write_standev_objdict(canindex index, cansubind subind, canbyte *data);

Запись значения объекта в словарь устройства.

Функция помещает объект, расположенный по адресу ***data**, в запись объектного словаря. До вызова функции приложение должно привести соответствующие данные к байтовому виду.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- ***data** – байтовый указатель на размещенные данные.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – не существует объекта с индексом **index**.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.
- CAN_ERRET_OBD_READONLY – попытка записи объекта с доступом только по чтению.

  Функция может также возвращать коды ошибок, вызванные превышением диапазона допустимых значений объекта, отказом в доступе из-за текущего состояния устройства и другими причинами, например:
- CAN_ERRET_OBD_VALRANGE – ошибка диапазона записываемого значения.
- CAN_ERRET_OBD_OBJACCESS – в текущем состоянии объект не может быть изменен.

Доступ к данным, формирующим записи объектного словаря, может осуществляться асинхронно из различных прикладных программ. Функции CANopen производят запись и чтение этих данных отдельными байтами. Таким образом, записи объектного словаря становятся разделяемым ресурсом, требующим дополнительных мер по обеспечению их состоятельности. Для защиты таких данных используется дополнительная буферизация и обновление их значений с использованием операции присвоения данных требуемого типа. В функциях чтения объектного словаря буферизация обеспечивает состоятельность данных при асинхронной записи со стороны приложения и побайтном чтении из CANopen. Для функций записи буферизация дополнительно позволяет осуществлять проверку данных до их занесения в объектный словарь.

4.2 Библиотечные модули адаптированной версии

Библиотечные модули размещаются в директории CANopen_WinLib_EN50325-5. Расположение файлов приводится относительно этой директории. Для удобства разработки приложений приводятся API всех видимых функций библиотеки. Их полный список размещен в файле \include\lib_defunc.h

4.2.1 Модули определений и прототипов

Размещаются в директории \include.

- lib_commonhead.h – заголовочный файл, используемый совместно CANopen библиотекой и приложением.
- lib_defines.h – определение параметров (констант), специфичных для CANopen.
- lib_defunc.h – прототипы функций. Включают функции CANopen библиотеки и обязательные функции (заглушки) приложения.
- lib_genhead.h – основной файл заголовков и подключений библиотеки. Содержит определение номера версии CANopen библиотеки. Размещает ссылку на заголовочный файл приложения.
- lib_globals.h – список внешних (глобальных) переменных библиотеки.
- lib_header.h – базовый заголовочный файл для модулей CANopen библиотеки.
- lib_macros.h – определение макросов CANopen библиотеки.
- lib_structures.h – определение структур данных CANopen библиотеки.
- lib_typedefs.h – определение типов данных CANopen библиотеки.

4.2.2 Системный модуль can_system_windows.c

Размещается в корневой директории CANopen_WinLib_EN50325-5.
Содержит системно-зависимые функции. Требует доработки для целевой платформы микроконтроллера.

void can_sleep(int32 microseconds);

Функция временн*о*й задержки.

Параметры:

- **microseconds** – временн*ая* задержка в микросекундах. Точное время задержки определяется разрешением соответствующего таймера системы. Любое положительное значение аргумента функции должно обеспечивать отличную от нуля задержку.

void can_init_system_timer(void (*handler)(void));

Инициализация CANopen таймера.

Период таймера в микросекундах задается константой CAN_TIMERUSEC. Сигнал или поток таймера должен обладать более высоким приоритетом, чем сигнал (поток) обработчика CAN кадров и ошибок CAN контроллера. CANopen таймер может быть не самоблокирующим, то есть возможны повторно-входимые вызовы обработчика таймера. Это дает возможность контролировать наложение тиков таймера при высокой загрузке системы. Обработчик ***handler** является сигнало-безопасным и может быть назначен непосредственно на аппаратные прерывания, в том числе не самоблокирующие.

Если таймер исполняется как отдельный поток операционной системы, метод работы диспетчера ОС может не гарантировать непрерывного выполнения этого потока. В таком случае рекомендуется формировать код обработчика таймера как единую критическую секцию.

Параметры:

- **handler** – функция обработчика таймера, имеет прототип void canopen_timer(void).

void can_cancel_system_timer(void);

Отмена CANopen таймера. Прекращает либо завершает работу таймера.

void init_critical(void);

Функция инициализации критической секции.

Внедряется в код с помощью макроса CAN_CRITICAL_INIT, определенного в модуле lib_macros.h.

void enter_critical(void);
void leave_critical(void);

Функции входа и выхода из критической секции.

Служат для обеспечения атомарности семафорных операций и непрерывности сегментов кода при использовании библиотеки в многопоточной среде, когда CANopen таймер и обработчик CAN кадров запускаются как отдельные потоки (нити). Функции должны обеспечивать многократный (вложенный) вход и выход из критической секции. Функции внедряются в код библиотеки с помощью макросов CAN_CRITICAL_BEGIN и CAN_CRITICAL_END, определенных в модуле lib_macros.h. Для однопоточных приложений (вложенные прерывания, операционные системы с поддержкой сигналов) код библиотеки обеспечивает возможность работы с не атомарными семафорами. Таким образом, эти макросы могут оставаться пустыми.

void enable_can_transmitter(void);
void disable_can_transmitter(void);

Функции разрешения работы и блокировки передающего CAN трансивера.

Служат для исключения выдачи CAN контроллером в сеть ложных сигналов при включении питания устройства. Работа трансивера аппаратно блокируется при включении питания и разрешается при инициализации CAN подсистемы (модуль lib_backinit.c).

void green_led_on(void);
void green_led_off(void);

Физическое включение и отключение зеленого светодиода.

void red_led_on(void);
void red_led_off(void);

Физическое включение и отключение красного светодиода.

4.2.3 Модуль __lib_main.c

Размещается в корневой директории CANopen_WinLib_EN50325-5.

Содержит запускаемую на выполнение функцию программы main(...) и монитор (главный цикл) программы. Монитор демонстрирует способы взаимодействия библиотеки и CAN драйвера. Так, если драйвер не осуществляет асинхронную доставку входящих сигналов (прием CAN кадра, возникновение ошибки), в главный цикл программы должен быть включен пропагатор сигналов драйвера ci_propagate_sigs().

4.2.4 Модуль __lib_events.c

Размещается в директории \CANopen.

Реализует обработчики CANopen событий.

void consume_sync(unsigned8 sc);

Вызывается при получении объекта синхронизации SYNC.

Параметры:
- **sc** – текущее значение SYNC счетчика (диапазон от 1 до 240).

void no_sync_event(void);

Потребитель SYNC не получил объекта синхронизации в течение промежутка времени, определяемого объектом 1006_h (период SYNC).

void consume_controller_error(canev ev);

Обрабатывает сигнал ошибки от CAN контроллера. Ошибка bus off отключения узла от CAN шины обрабатывается согласно настройкам объекта 1029_h – поведение устройства при возникновении серьезных ошибок. Обработка остальных ошибок предусматривает передачу соответствующего сообщения EMCY и светодиодную индикацию. Коды ошибок определены в заголовочном файле CAN драйвера CHAI.
Параметры:
- **ev** (тип int16) – код ошибки:
  CIEV_BOFF – bus off,
  CIEV_EWL – error warning limit,
  CIEV_HOVR – hardware overrun,
  CIEV_SOVR – software overrun.
  CIEV_WTOUT – CAN write timeout.

void life_guarding_event(void);

Обработка события life guarding event со статусом ”occurred”.
Вызывается при отсутствии запроса в протоколе охраны узла.

void life_guarding_resolved(void);

Обработка события life guarding event со статусом ”resolved”.
Вызывается при возобновлении запросов в протоколе охраны узла.

void pdo_activated_slave(canindex index, cansubind subind);
void srdo_activated_slave(canindex index, cansubind subind);

Сообщение об активировании соответственно PDO или SRDO.
Информирует приложение о том, что отображенный в PDO или SRDO объект был успешно записан в объектный словарь устройства.
Параметры:
- **index** – индекс прикладного объекта.
- **subind** – субиндекс прикладного объекта.

void no_pdo_event(canindex index);

Не получено RPDO до истечения его таймера события.
Параметры:
- **index** – индекс коммуникационного объекта RPDO.

void can_timer_overlap(void);

Зарегистрировано наложение тиков CANopen таймера.

void can_cache_overflow(canbyte state);

Переполнен выходной CANopen кэш.
Параметры:
- **state** – NMT состояние CAN узла.

4.2.5 Модуль lib_backinit.c

Размещается в директории \CANopen.
Реализует функции (пере)инициализации CANopen устройства. Формирует и поддерживает

диспетчер таймера и CANopen монитор.

unsigned8 get_netmask_free(void);

Битовая маска свободных CAN сетей.

Возвращаемые значения:

• Маска CAN сетей из числа физических CAN_MASK_NETWORKS, которые не заняты другими приложениями.

int16 cifunc_start(void);

Переводит в активное состояние все рабочие CAN контроллеры (CAN сети).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

• CAN_RETOK – все рабочие CAN контроллеры запущены успешно.
• CAN_ERRET_CI_NETWORKS – нет рабочих CAN контроллеров.
• CAN_ERRET_CI_START – не удалось запустить хотя бы один рабочий контроллер.

int16 cifunc_stop(void);

Переводит все рабочие CAN контроллеры в состояние останова.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

• CAN_RETOK – все рабочие CAN контроллеры остановлены успешно.
• CAN_ERRET_CI_NETWORKS – нет рабочих CAN контроллеров.
• CAN_ERRET_CI_STOP – не удалось остановить хотя бы один рабочий контроллер.

int16 can_reset_communication(void);

Отрабатывает NMT команду Reset Communication.

(Пере)инициализирует коммуникационную подсистему CANopen устройства. Формирует маску рабочих CAN сетей из числа свободных.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

см. функцию can_reset_node(void).

int16 can_reset_node(void);

Отрабатывает NMT команду Reset Node.

(Пере)инициализирует CANopen устройство. Формирует маску рабочих CAN сетей из числа свободных.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

• CAN_RETOK – запуск CANopen устройства выполнен успешно.
• CAN_ERRET_CI_STOP – не удалось остановить хотя бы один рабочий контроллер.
• CAN_ERRET_NODEID – ошибочный номер CAN узла устройства.
• CAN_ERRET_CI_NETWORKS – все CAN сети заняты либо инициализация ни одной из них не удалась.

int16 start_can_device(void);

Осуществляет первоначальный запуск устройства после включения питания. Выполняет однократные операции, которые не требуются при программной переинициализации CANopen устройства.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

• CAN_ERRET_CI_INIT – ошибка начальной инициализации CAN драйвера.

см. функцию can_reset_node(void).

int16 stop_can_device(void);

Осуществляет полный останов устройства.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

• CAN_RETOK – останов CAN устройства выполнен успешно.

void canopen_monitor(void);

CANopen монитор.

Вызывается из главного цикла программы.

4.2.6 Модуль lib_can_networks.c

Размещается в директории \CANopen.
Формирует объектный словарь конфигурации CAN сетей (контроллеров).

int16 get_can_networks_bytes_objsize(canindex index, cansubind subind);
int16 see_can_networks_access(canindex index, cansubind subind);
int16 get_can_networks_objtype(canindex index, cansubind subind);
int16 read_can_networks_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_can_networks_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к объектному словарю конфигурации CAN сетей. Назначение и параметры функций приведены в разделе 4.1.

Поддерживают коммуникационные объекты, заданные индексами:
$11F0_h$ - Параметры CAN сетей.

4.2.7 Модуль lib_canid.c

Размещается в директории \CANopen.
Формирует таблицу соответствия индексов словаря CANopen (коммуникационные объекты) и CAN идентификаторов канального уровня. Осуществляет проверку идентификаторов ограниченного использования.
Устанавливает аппаратный фильтр принимаемых кадров CAN контроллера. В зависимости от поддержки CAN драйвером используется одно-, двух- либо трех-уровневый масочный фильтр входящих CAN кадров. Двух-уровневый фильтр обеспечивает полную фильтрацию для предопределенного распределения CAN идентификаторов согласно CiA 301. Трех-уровневый позволяет отдельно селектировать кадры протокола EN50325-5.

int16 correct_recv_canid(canindex index, canlink canid);

Формирует таблицу соответствия индексов коммуникационного объектного словаря CANopen и CAN идентификаторов канального уровня. Устанавливает аппаратный входной фильтр CAN контроллера для всех рабочих CAN сетей.
Параметры:
- **index** – индекс коммуникационного объекта.
- **canid** – CAN идентификатор канального уровня.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – успешное завершение.
- CAN_ERRET_CI_STOP – не удалось остановить хотя бы один рабочий CAN контроллер.
- CAN_ERRET_CI_FILTER – ошибка установа аппаратного фильтра хотя бы одного рабочего CAN контроллера.
- CAN_ERRET_CI_START – не удалось запустить хотя бы один рабочий CAN контроллер.
- CAN_ERRET_OUTOFMEM – нет места для размещения записи таблицы.

int16 correct_double_canid(canindex index, canlink canid_1, canlink canid_2);

Формирует таблицу соответствия индексов коммуникационного объектного словаря CANopen и CAN идентификаторов канального уровня для протокола EN50325-5. В этом протоколе каждому SRDO объекту сопоставлено два CAN идентификатора. Устанавливает аппаратный входной фильтр CAN контроллера для всех рабочих CAN сетей.
Параметры:
- **index** – индекс коммуникационного SRDO объекта.
- **canid_1** – первый CAN идентификатор для SRDO объекта.

• **canid_2** – второй CAN идентификатор для SRDO объекта.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
см. функцию correct_recv_canid(...).

canindex find_comm_recv_index(canlink canid);

Возвращает индекс коммуникационного объекта CANopen, который соответствует CAN идентификатору канального уровня. При отсутствии соответствующего объекта возвращается значение CAN_INDEX_DUMMY.

Параметры:

• **canid** – CAN идентификатор канального уровня.

Возвращаемые значения:

• индекс коммуникационного CANopen объекта.

unsigned8 check_canid_restricted(canlink canid);

Определяет принадлежность CAN-ID к идентификаторам ограниченного использования.

Параметры:

• **canid** – проверяемый CAN идентификатор канального уровня.

Возвращаемые значения:

• RESTRICTED – **canid** является идентификатором ограниченного использования.
• UN_RESTRICTED – **canid** не относится к идентификаторам ограниченного использования.

void can_init_recv_canids(void);

Инициализирует таблицу соответствия индексов коммуникационного объектного словаря CANopen и CAN идентификаторов канального уровня.

4.2.8 Модуль lib_globals.c

Размещается в директории \CANopen.
Определяет внешние (глобальные) переменные и структуры данных библиотеки.

4.2.9 Модуль lib_inout.c

Размещается в директории \CANopen.
Осуществляет прием и передачу CAN кадров канального уровня. Производит первичный разбор идентификаторов принимаемых кадров.

void push_all_can_data(void);

Пересылает CAN драйверу накопленные в CANopen кэше кадры канального уровня с целью дальнейшего вывода в рабочую CAN сеть. Для гарантированного вывода всех данных из кэша также вызывается из CANopen таймера. Функция является сигналобезопасной.

Замечание.

Использование кэша может привести к тому, что кадры будут выводиться в CAN сеть в последовательности, отличной от очередности их записи со стороны приложения. Для протокола EN50325-5 используется выделенный кэш-буфер, поскольку важна строгая последовательность передачи CAN кадров.

int16 send_can_data(canframe *cf);

Размещает в CANopen кэше кадр канального уровня. Осуществляет пересылку CAN драйверу всех накопленных в кэше кадров. Программный приоритет кадра устанавливается на основе значения его CAN-ID. Функция является сигналобезопасной.

Параметры:

- ***cf** – кадр, предназначенный для отправки CAN драйверу.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – данные размещены в CANopen кэше. Для вызывающих функций означает успешное завершение отправки CAN кадра.
- CAN_ERRET_NODE_STATE – CAN узел находится в состоянии останова или инициализации.
- CAN_ERRET_COMM_SEND – не удалось разместить кадр в CANopen кэше.

void can_read_handler_0(canev ev);
void can_read_handler_1(canev ev);
void can_read_handler_2(canev ev);
void can_read_handler_3(canev ev);
void can_read_handler_4(canev ev);
void can_read_handler_5(canev ev);
void can_read_handler_6(canev ev);
void can_read_handler_7(canev ev);

Обработчики сигнала приема кадров канального уровня для CAN контроллеров 0..7. Функции являются сигналобезопасными и обеспечивают в том числе чтение кадров, принятых во время обработки текущего кадра. Параметр функции **ev** не используется.

void can_rchd_direct(unsigned8 chan, canframe *cf);

Обработчик сигналов приема кадров канального уровня непосредственно от CAN контроллера, без дополнительной буферизации в драйвере.

Параметры:

- **chan** – номер CAN контроллера от 0 до 7.
- ***cf** – принятый CAN кадр.

void can_set_datalink_layer(unsigned8 mode);

Управляет логическим доступом к канальному уровню рабочих CAN сетей по записи.

Параметры:

- **mode** – режим логического доступа.
  ON – штатный режим работы: все передаваемые кадры отправляются в CAN сеть.
  OFF – все передаваемые CAN кадры аннулируются.

void can_init_io(void);

Инициализирует семафоры и другие данные модуля.

4.2.10 Модуль lib_led_indicator.c

Размещается в директории \CANopen.
Поддерживает светодиодную индикацию состояния CANopen устройства согласно CiA 303 ч. 3.

void led_control(void);

Управляет переключением светодиодов в различных режимах. Вызывается из CANopen таймера.

void set_led_green_on(void);
void set_led_green_off(void);

Включение и отключение зеленого светодиода в непрерывный режим.

void set_led_red_on(void);
void set_led_red_off(void);

Включение и отключение красного светодиода в непрерывный режим.

void set_leds_flickering(void);

Включение светодиодов в режим мерцания с частотой 10 Гц. Мерцание красного и зеленого светодиодов осуществляется в противофазе.

void set_led_green_blinking(void);

Включение зеленого светодиода в режим мигания с частотой 2.5 Гц.

void set_led_red_blinking(void);

Включение красного светодиода в режим мигания с частотой 2.5 Гц.

void set_led_green_single_flash(void);
void set_led_green_double_flash(void);
void set_led_green_triple_flash(void);
void set_led_green_quadruple_flash(void);

Включение зеленого светодиода в режим соответственно одной, двух, трех и четырех вспышек длительностью 200 мс с интервалом 200 мс и паузой 1 с.

void set_led_red_single_flash(void);
void set_led_red_double_flash(void);
void set_led_red_triple_flash(void);
void set_led_red_quadruple_flash(void);

Включение красного светодиода в режим соответственно одной, двух, трех и четырех вспышек длительностью 200 мс с интервалом 200 мс и паузой 1 с.

void can_init_led_indication(void);

Инициализирует данные модуля индикации.

4.2.11 Модуль lib_lib.c

Размещается в директории \CANopen.
Функции общего назначения: подсчет CRC, преобразование и проверка данных.

void can_start_crc(unsigned16 val);

Инициализация расчета 16-разрядного CRC.

Параметры:

- **val** – инициализирующее значение. Для CANopen равно нулю.

unsigned16 can_calc_crc(unsigned8 data);

Подсчет CRC-16 для очередного байта данных.

Параметры:

- **data** – байт данных для включения в расчет CRC.

Возвращаемое значение:

- результат подсчета CRC-16.

void can_init_crc(void);

Расчет таблицы для байт-ориентированного подсчета CRC-16.

int16 check_node_id(cannode node);

Проверка номера CAN узла.

Параметры:

- **node** – проверяемый номер CAN узла.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – допустимый номер CAN узла от 1 до 127.
- CAN_ERRET_NODEID – ошибочный номер CAN узла.

int16 check_bitrate_index(unsigned8 br);

Проверка индекса битовой скорости CAN сети.

Параметры:

- **br** – проверяемый индекс битовой скорости .

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – допустимый индекс битовой скорости.
- CAN_ERRET_BITRATE – ошибочный индекс битовой скорости.

cannode get_actual_node_id(void);

Возвращает фактический номер CAN узла устройства.

Возвращаемое значение:

- фактический номер CAN узла.

unsigned8 get_actual_node_state(void);

Возвращает фактическое NMT состояние CAN узла.

Возвращаемое значение:

- фактическое NMT состояние CAN узла.

void clear_can_data(canbyte *data);

Очистка поля данных CAN кадра.

Параметры:

- **data** – поле данных CAN кадра (8 байт).

void u16_to_canframe(unsigned16 ud, canbyte *data);

Преобразование данных типа unsigned16 в байтовый (сетевой) формат. Поддерживает только little-endian порядок 8-битовых байт.

Параметры:

- **ud** – преобразуемый параметр.
- ***data** – байтовый указатель на преобразованные данные.

unsigned16 canframe_to_u16(canbyte *data);

Преобразование данных из байтового (сетевого) формата в тип unsigned16. Поддерживает только little-endian порядок 8-битовых байт.

Параметры:

- ***data** – байтовый указатель на преобразуемые данные.

Возвращаемое значение:

- данные типа unsigned16.

void u32_to_canframe(unsigned32 ud, canbyte *data);

Преобразование данных типа unsigned32 в байтовый (сетевой) формат. Поддерживает только little-endian порядок 8-битовых байт.

Параметры:

- **ud** – преобразуемый параметр.
- ***data** – байтовый указатель на преобразованные данные.

unsigned32 canframe_to_u32(canbyte *data);

Преобразование данных из байтового (сетевого) формата в тип unsigned32. Поддерживает только little-endian порядок 8-битовых байт.

Параметры:

- ***data** – байтовый указатель на преобразуемые данные.

Возвращаемое значение:

- данные типа unsigned32.

4.2.12 Модуль lib_nmt_responder.c

Размещается в директории \CANopen.
Поддерживает диаграмму конечного автомата NMT responder устройства. Реализует протоколы контроля ошибок (сердцебиение и охрана узла).

void start_node(void);
Переводит узел в операционное состояние. Запускает работу безопасного протокола EN50325-5, если он сконфигурирован.

void stop_node(void);
Переводит узел в состояние останова.

void enter_pre_operational(void);
Переводит узел в пред-операционное состояние. Реализует протокол загрузки узла Boot-up.

void nmt_slave_process(canframe *cf);
Обрабатывает принятые NMT команды управления сетью.
Параметры:
- ***cf** – CAN кадр с нулевым значением CAN-ID, который содержит NMT команду.

void nmt_slave_control(void);
Инициализирует выполнение NMT команд перезапуска узла (Reset Node) или его коммуникационной подсистемы (Reset Communication). Вызывается из главного цикла программы.

int16 get_ecpslave_bytes_objsize(canindex index, cansubind subind);
int16 see_ecpslave_access(canindex index, cansubind subind);
int16 get_ecpslave_objtype(canindex index, cansubind subind);
int16 read_ecpslave_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_ecpslave_objdict(canindex index, cansubind subind, canbyte *data);
Функции доступа к объектному словарю протоколов контроля ошибок. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают коммуникационные объекты, заданные индексами:
$100C_h$ – охранное время CAN узла.
$100D_h$ – множитель времени жизни.
1017_h – период сердцебиения.

void node_guard_slave(void);
Обрабатывает принятый RTR кадр охраны узла.

void manage_slave_ecp(void);
Управляет задержкой запуска узла (Boot-up, Start) и протоколами контроля ошибок (сердцебиение, охрана узла). Вызывается из CANopen таймера.

void can_init_ecp(void);
Инициализирует данные модуля.

4.2.13 Модуль lib_obdsdo_server .c

Размещается в директории \CANopen.
Формирует и поддерживает объектный словарь единственного SDO параметра сервера, который используется по умолчанию.

int16 find_sdo_server_send_canid(canlink *canid);

Возвращает CAN-ID SDO протокола, который передается от сервера клиенту.

Параметры:

- ***canid** – CAN идентификатор канального уровня SDO протокола.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – SDO действителен.

int16 get_sdo_server_bytes_objsize(canindex index, cansubind subind);
int16 see_sdo_server_access(canindex index, cansubind subind);
int16 get_sdo_server_objtype(canindex index, cansubind subind);
int16 read_sdo_server_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_sdo_server_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к объектному словарю SDO параметра сервера по умолчанию. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают коммуникационные объекты, заданные индексами:
1200_h – SDO параметр сервера по умолчанию.

void can_init_sdo_server(void);

Инициализирует данные модуля.

4.2.14 Модуль lib_obdsrv .c

Размещается в директории \CANopen.
Организует двухуровневый доступ к записям объектного словаря сервера.

int32 server_get_object_size(canindex index, cansubind subind);
int32 server_put_object_size(canindex index, cansubind subind, int32 size);
int16 server_see_access(canindex index, cansubind subind);
int16 server_get_object_type(canindex index, cansubind subind);
int16 server_read_object_dictionary(canindex index, cansubind subind, canbyte *data);
int16 server_write_object_dictionary(canindex index, cansubind subind, canbyte *data);

Функции доступа к объектному словарю слейв устройства (сервера). Назначение и параметры функций приведены в разделе 4.1.

int16 server_read_obd_u32(canindex index, cansubind subind, unsigned32 *du32);

Чтение объекта размером не более 32 бит.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- ***du32** – указатель на размещаемые данные, 32 бита.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
см. раздел 4.1, функции доступа к объектному словарю.

int16 server_write_obd_u32(canindex index, cansubind subind, unsigned32 du32);

Запись объекта размером не более 32 бит.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- **du32** – записываемый в словарь объект размером до 32 бит.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
см. раздел 4.1, функции доступа к объектному словарю.

4.2.15 Модуль lib_obj_deftype .c

Размещается в директории \CANopen.
Формирует словарь объектов определения типов данных.

int16 get_deftype_bytes_objsize(canindex index, cansubind subind);
int16 see_deftype_access(canindex index, cansubind subind);
int16 get_deftype_objtype(canindex index, cansubind subind);
int16 read_deftype_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_deftype_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю объектов определения типов данных. Назначение и параметры функций приведены в разделе 4.1.
Размер объекта определяется его типом. Запись любого значения завершается успешно без каких-либо последствий, а по чтению всегда возвращается ноль.
Поддерживают объекты, заданные индексами:
0002_h - INTEGER8; 0005_h - UNSIGNED8.
0003_h - INTEGER16; 0006_h - UNSIGNED16.
0004_h - INTEGER32; 0007_h - UNSIGNED32.

4.2.16 Модуль lib_obj_device.c

Размещается в директории \CANopen.
Формирует объектный словарь описания устройства. Используется совместно с модулем приложения \CANopen__CiA401_devices.c, где определяются значения соответствующих объектов.

int16 get_dev_bytes_objsize(canindex index, cansubind subind);
int16 see_dev_access(canindex index, cansubind subind);
int16 get_dev_objtype(canindex index, cansubind subind);
int16 read_dev_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_dev_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к объектному словарю описания устройства. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают коммуникационные объекты, заданные индексами:
1000_h – тип устройства.
1002_h – регистр статуса от производителя.
1008_h – название устройства от производителя.
1009_h – версия «железа» устройства от производителя.
$100A_h$ – версия программного обеспечения устройства от производителя.
1018_h – объект идентификации устройства.

4.2.17 Модуль lib_obj_emcy .c

Размещается в директории \CANopen.
Формирует и поддерживает объект срочного сообщения EMCY (Emergency).

int16 get_emcy_bytes_objsize(canindex index, cansubind subind);
int16 see_emcy_access(canindex index, cansubind subind);
int16 get_emcy_objtype(canindex index, cansubind subind);
int16 read_emcy_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_emcy_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю объекта EMCY. Назначение и параметры функций

приведены в разделе 4.1.
Поддерживают коммуникационные объекты, заданные индексами:
1014_h – COB-ID объекта EMCY.
1015_h – время подавления посылок EMCY.

void control_emcy_inhibit(void);

Осуществляет контроль времени подавления EMCY. Вызывается из CANopen таймера.

int16 produce_emcy(unsigned16 errorcode, unsigned16 addinf, canbyte *mserr);

Создает сообщене EMCY с полной информацией об ошибке. Заносит в список предопределенных ошибок (объект 1003_h) код **errorcode** совместно с дополнительной информацией **addinf**. Затем формирует и отсылает EMCY с кодом **errorcode**, текущим состоянием регистра ошибок и полем ошибки производителя устройства ***mserr (**используются первые пять байт). EMCY должен быть действительным, а время подавления его посылок должно истечь.
Параметры:
- **errorcode** – код ошибки и EMCY.
- **addinf** – дополнительная информация об ошибке.
- ***mserr** – поле ошибки производителя устройства (5 байт).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – нормальное завершение.
- CAN_ERRET_EMCY_INVALID – объект EMCY не действителен.
- CAN_ERRET_EMCY_INHIBIT – объект EMCY находится в состоянии подавления.
- CAN_ERRET_NODE_STATE – CAN узел находится состоянии останова или инициализации.
- CAN_ERRET_COMM_SEND – не удалось разместить кадр в CANopen кэше.

int16 produce_emcy_default(unsigned16 errorcode);

Создает сообщение EMCY с минимальной информацией об ошибке. Используется только код ошибки **errorcode**. Дополнительная информация в списке предопределенных ошибок и поле ошибки производителя устройства отсутствуют (сбрасываются в ноль).
Параметры:
- **errorcode** – код ошибки и сообщения EMCY.

Возвращаемые значения:
см. функцию produce_emcy(...).

void can_init_emcy(void);

Инициализирует данные модуля.

4.2.18 Модуль lib_obj_err_behaviour.c

Размещается в директории \CANopen.
Задает коммуникационные режимы устройства при возникновении серьезных ошибок и сбоев. Обычно такие ошибки рассматриваются как отказ устройства.

int16 get_err_behaviour_bytes_objsize(canindex index, cansubind subind);
int16 see_err_behaviour_access(canindex index, cansubind subind);
int16 get_err_behaviour_objtype(canindex index, cansubind subind);
int16 read_err_behaviour_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_err_behaviour_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к объекту поведения при ошибках. Назначение и параметры функций приведены в разделе 4.1.

Поддерживают коммуникационные объекты, заданные индексами:
1029_h – поведение устройства при возникновении серьезных ошибок и сбоев.

void process_serious_error(cansubind subind);

Исполняет алгоритм поведения устройства при возникновении серьезных ошибок.

Параметры:

- **subind** – субиндекс объекта 1029_h (поведение устройства при возникновении ошибок).

void can_init_err_behaviour(void);

Инициализирует данные модуля.

4.2.19 Модуль lib_obj_errors.c

Размещается в директории \CANopen.
Формирует и поддерживает объект ошибок.

int16 get_err_bytes_objsize(canindex index, cansubind subind);
int16 see_err_access(canindex index, cansubind subind);
int16 get_err_objtype(canindex index, cansubind subind);
int16 read_err_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_err_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю объекта ошибок. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают коммуникационные объекты, заданные индексами:
1001_h – регистр ошибок.
1003_h – список предопределенных ошибок.

void set_error_field(unsigned16 errorcode, unsigned16 addinf);

Заносит в список предопределенных ошибок код ошибки **errorcode** совместно с дополнительной информацией **addinf**. Устанавливает соответствующий бит регистра ошибок, а также бит общей ошибки. Коды ошибок в диапазоне $1000_h..10FF_h$ устанавливают только бит общей ошибки.
Функция является сигналобезопасной. При наложении обращений к функции возможна потеря записи в списке предопределенных ошибок, но информация в любом случае сохраняется в регистре ошибок.

Параметры:

- **errorcode** – код ошибки.
- **addinf** – дополнительная информация об ошибке.

void clear_error_register(unsigned8 mask);

Производит побитовую очистку регистра ошибок. Нулевой бит регистра (общая ошибка) сбрасывается лишь при условии очистки всех остальных бит. При этом выдается сообщение EMCY с нулевым значением кода ошибки (сброс ошибки). Коды ошибок в диапазоне $1000_h..10FF_h$ устанавливают только бит общей ошибки, который, в отсутствии других ошибок, будет сброшен при любом значении **mask**.

Параметры:

- **mask** – битовая маска. Очищаются биты, для которых в маске установлено значение 1.

unsigned8 read_error_register(void);

Возвращает значение регистра ошибок.

Возвращаемое значение:

- регистр ошибок.

void can_init_error(void);
Инициализирует данные модуля.

4.2.20 Модуль lib_obj_sync.c

Размещается в директории \CANopen.
Формирует и поддерживает объекты синхронизации SYNC.

int16 get_sync_bytes_objsize(canindex index, cansubind subind);
int16 see_sync_access(canindex index, cansubind subind);
int16 get_sync_objtype(canindex index, cansubind subind);
int16 read_sync_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_sync_objdict(canindex index, cansubind subind, canbyte *data);
Функции доступа к словарю объектов синхронизации SYNC. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают коммуникационные объекты, заданные индексами:
1005_h – COB-ID объекта синхронизации.
1006_h – период объекта синхронизации.
1007_h – длительность окна синхронизации.
1019_h – значение переполнения для SYNC счетчика.

void reset_sync_counter(void);
Устанавливает SYNC счетчик в минимальное (единичное) значение.

unsigned8 sync_window_expired(void);
Определяет состояние (истечение времени) окна синхронизации.
Возвращаемые значения:
- FALSE – окно синхронизации открыто, можно проводить синхронные операции.
- TRUE – окно синхронизации истекло, синхронные операции запрещены.

void sync_received(canframe *cf);
Производит обработку принятого из сети объекта синхронизации SYNC. Если устройство является источником SYNC, функция вызывается при каждой передаче SYNC кадра.
Параметры:
- ***cf** – принятый или переданный CAN кадр, содержащий объект синхронизации SYNC.

void control_sync(void);
Осуществляет управление SYNC объектом. Вызывается из CANopen таймера.

void can_init_sync(void);
Инициализирует данные модуля.

4.2.21 Модуль lib_obj_time.c

Размещается в директории \CANopen.
Формирует и поддерживает объект временн*о*й метки TIME.

int16 get_time_bytes_objsize(canindex index, cansubind subind);
int16 see_time_access(canindex index, cansubind subind);
int16 get_time_objtype(canindex index, cansubind subind);
int16 read_time_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_time_objdict(canindex index, cansubind subind, canbyte *data);
Функции доступа к словарю объекта временн*о*й метки TIME. Назначение и параметры

функций приведены в разделе 4.1.
Поддерживают коммуникационные объекты, заданные индексами:
1012_h – COB-ID объекта временной метки.

void can_init_time(void);

Инициализирует данные модуля.

4.2.22 Модуль lib_pdo_map.c

Размещается в директории \CANopen.
Формирует и поддерживает динамическое байт-ориентированное PDO отображение.

int16 check_pdo_map_object(canindex index);

Осуществляет проверку наличия и состояния объекта PDO отображения.
Параметры:
- **index** – индекс объекта PDO отображения.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – объект PDO отображения существует и активирован.
- CAN_ERRET_OBD_NOOBJECT – не существует PDO отображения с индексом **index**.
- CAN_ERRET_PDO_MAP_DEACT – PDO отображение не активировано.

int16 get_pdo_map_bytes_objsize(canindex index, cansubind subind);
int16 see_pdo_map_access(canindex index, cansubind subind);
int16 get_pdo_map_objtype(canindex index, cansubind subind);
int16 read_pdo_map_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_pdo_map_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю объектов PDO отображения. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают объекты, заданные индексами (для четырех PDO):
$1600_h..1603_h$ – параметры отображения принимаемых PDO (RPDO1 - RPDO4).
$1A00_h..1A03_h$ – параметры отображения передаваемых PDO (TPDO1 - TPDO4).

int16 map_pdo(canindex index, canframe *cf);

Формирует PDO отображение, соответствующее коммуникационному PDO объекту и заносит его в поле данных CAN кадра. Определяет и устанавливает длину поля данных.
Параметры:
- **index** – индекс коммуникационного объекта PDO (как правило, передаваемого TPDO).
- ***cf** – CAN кадр, в который заносится PDO отображение.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – PDO сформирован успешно.
- CAN_ERRET_OBD_NOOBJECT – не существует коммуникационного объекта с индексом **index**, либо соответствующего ему PDO отображения.
- CAN_ERRET_PDO_INVALID – коммуникационный PDO объект не действителен.
- CAN_ERRET_PDO_MAP_DEACT – PDO отображение не активировано.
- CAN_ERRET_PDO_ERRMAP – размер данных PDO отображения превышает максимальную длину CAN кадра.
- Функция также может возвращать ошибки чтения из словаря значений отображаемых объектов.

int16 activate_pdo(canindex index, canframe *cf);

Осуществляет разбор CAN кадра, руководствуясь PDO отображением, которое соответствует коммуникационному PDO объекту. Заносит извлеченные из кадра данные в объектный словарь.

Параметры:
- **index** – индекс коммуникационного объекта PDO (как правило, принимаемого RPDO).
- ***cf** – CAN кадр, из которого извлекаются прикладные данные PDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – Все отображенные данные успешно занесены в объектный словарь.
- CAN_ERRET_OBD_NOOBJECT – не существует коммуникационного объекта с индексом **index**, либо соответствующего ему PDO отображения.
- CAN_ERRET_PDO_INVALID – коммуникационный PDO объект не действителен.
- CAN_ERRET_PDO_MAP_DEACT – PDO отображение не активировано.
- CAN_ERRET_PDO_ERRMAP – размер данных PDO отображения превышает длину принятого CAN кадра.
- Функция также может возвращать ошибки записи в словарь значений отображаемых объектов.

void can_init_pdo_map(void);

Инициализирует данные модуля.

4.2.23 Модуль lib_pdo_obd.c

Размещается в директории \CANopen.
Формирует и поддерживает коммуникационные PDO объекты.

int16 check_pdo_comm_object(canindex index);

Определяет наличие и состояние коммуникационного PDO объекта.

Параметры:
- **index** – индекс коммуникационного PDO объекта.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – PDO существует и действителен.
- CAN_ERRET_OBD_NOOBJECT – PDO с индексом **index** не существует.
- CAN_ERRET_NODE_STATE – CAN узел находится в не операционном состоянии.
- CAN_ERRET_PDO_INVALID – PDO объект не действителен.

int16 find_pdo_recv_canid(canindex index, canlink *canid);

Выдает идентификатор CAN кадра принимаемого PDO. Используется для формирования удаленного запроса RPDO.

Параметры:
- **index** – индекс коммуникационного объекта RPDO.
- ***canid** – CAN идентификатор канального уровня RPDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.
- CAN_RETOK – RPDO существует и действителен.
- CAN_ERRET_OBD_NOOBJECT – RPDO с индексом **index** не существует.
- CAN_ERRET_NODE_STATE – CAN узел находится в не операционном состоянии.
- CAN_ERRET_PDO_INVALID – RPDO не действителен.
- CAN_ERRET_PDO_NORTR – удаленный запрос для RPDO запрещен.

Замечание.

В последних версиях стандарта CiA 301 (4.2.0.xx) бит удаленного запроса для RPDO не поддерживается (зарезервирован). Это не позволяет локально определить возможность формирования удаленного запроса для RPDO.

int16 find_pdo_tran_canid(canindex index, canlink *canid);

Выдает идентификатор CAN кадра передаваемого PDO.

Параметры:
- **index** – индекс коммуникационного объекта TPDO.

- ***canid** – CAN идентификатор канального уровня TPDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – TPDO существует и действителен.
- CAN_ERRET_OBD_NOOBJECT – TPDO с индексом **index** не существует.
- CAN_ERRET_NODE_STATE – CAN узел находится в не операционном состоянии.
- CAN_ERRET_PDO_INVALID – TPDO не действителен.

int16 find_pdo_recv_trantype(canindex index, unsigned8 *trtype);

Выдает тип передачи принимаемого PDO (субиндекс 2 коммуникационного объекта).

Параметры:

- **index** – индекс коммуникационного объекта RPDO.
- ***trtype** – тип передачи RPDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – RPDO существует и действителен.
- CAN_ERRET_OBD_NOOBJECT – RPDO с индексом **index** не существует.
- CAN_ERRET_NODE_STATE – CAN узел находится в не операционном состоянии.
- CAN_ERRET_PDO_INVALID – RPDO не действителен.

int16 find_pdo_tran_trantype(canindex index, unsigned8 *trtype);

Выдает тип передачи передаваемого PDO (субиндекс 2 коммуникационного объекта).

Параметры:

- **index** – индекс коммуникационного объекта TPDO.
- ***trtype** – тип передачи TPDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – TPDO существует и действителен.
- CAN_ERRET_OBD_NOOBJECT – TPDO с индексом **index** не существует.
- CAN_ERRET_NODE_STATE – CAN узел находится в не операционном состоянии.
- CAN_ERRET_PDO_INVALID – TPDO не действителен.

void find_pdo_rtr_tran_index(canlink canid, canindex *index);

Выдает индекс коммуникационного PDO объекта для CAN идентификатора **canid**. Используется для поиска TPDO, соответствующего удаленному запросу.

Параметры:

- **canid** – CAN идентификатор канального уровня TPDO.
- ***index** – индекс коммуникационного объекта TPDO. CAN_INDEX_DUMMY, если соответствующий TPDO не обнаружен или не действителен или удаленный запрос для него запрещен.

int16 set_pdo_state(canindex index, unsigned8 state);

Устанавливает состояние PDO действителен / не действителен. Перестраивает аппаратный фильтр входящих CAN кадров.

Параметры:

- **index** – индекс коммуникационного объекта PDO.
- **state** – новое состояние PDO (VALID / NOT_VALID).

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – установлено новое состояние PDO.
- CAN_ERRET_OBD_NOOBJECT – PDO с индексом **index** не существует.
- CAN_ERRET_PDO_MAP_DEACT – PDO отображение не активировано (если **state** = VALID).
- CAN_ERRET_CI_STOP – не удалось остановить хотя бы один рабочий CAN контроллер.
- CAN_ERRET_CI_FILTER – ошибка установа аппаратного фильтра хотя бы одного рабочего CAN контроллера.
- CAN_ERRET_CI_START – не удалось запустить хотя бы один рабочий CAN контроллер.

void set_pdo_recv_event_timer(canindex index);

Установ таймера события принимаемого PDO.

Параметры:

- **index** – индекс коммуникационного объекта RPDO.

void set_pdo_tran_event_timer(canindex index);

Установ таймера события передаваемого PDO.

Параметры:

- **index** – индекс коммуникационного объекта TPDO.

int16 test_cyclic_tpdo(canindex index, unsigned8 sc);

Определяет состояние циклических синхронных передаваемых PDO.

Параметры:

- **index** – индекс коммуникационного объекта TPDO.
- **sc** – текущее значение SYNC счетчика.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – циклическое синхронное TPDO готово к передаче.
- CAN_ERRET_OBD_NOOBJECT – TPDO с индексом **index** не существует.
- CAN_ERRET_NODE_STATE – CAN узел находится в не операционном состоянии.
- CAN_ERRET_PDO_INVALID – TPDO не действителен.
- CAN_ERRET_PDO_TRTYPE – неподходящий тип передачи TPDO.
- CAN_ERRET_PDO_TRIGGER – момент передачи TPDO не наступил.

int16 test_tpdo_inhibit(canindex index);

Определяет состояние подавления передаваемого PDO.

Параметры:

- **index** – индекс коммуникационного объекта TPDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0:

- CAN_RETOK – время подавления TPDO истекло.
- CAN_ERRET_OBD_NOOBJECT – TPDO с индексом **index** не существует.
- CAN_ERRET_NODE_STATE – CAN узел находится в не операционном состоянии.
- CAN_ERRET_PDO_INVALID – TPDO не действителен.
- CAN_ERRET_PDO_INHIBIT – TPDO находится в состоянии подавления.

void control_pdo(void);

Осуществляет контроль таймеров события RPDO и TPDO, а также времени подавления TPDO. Вызывается из CANopen таймера.

int16 get_pdo_comm_bytes_objsize(canindex index, cansubind subind);
int16 see_pdo_comm_access(canindex index, cansubind subind);
int16 get_pdo_comm_objtype(canindex index, cansubind subind);
int16 read_pdo_comm_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_pdo_comm_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю коммуникационных PDO объектов. Назначение и параметры функций приведены в разделе 4.1.

Поддерживают объекты, заданные индексами (для четырех PDO):

1400_h..1403_h – коммуникационные параметры принимаемых PDO (RPDO1 - RPDO4).

1800_h..1803_h – коммуникационные параметры передаваемых PDO (TPDO1 - TPDO4).

void can_init_pdo(void);

Инициализирует данные модулей lib_pdo_obd.c, lib_pdo_map.c и lib_pdo_proc.c.

4.2.24 Модуль lib_pdo_proc.c

Размещается в директории \CANopen.
Производит обработку принимаемых и передаваемых PDO, а также удаленных PDO запросов.

int16 pdo_remote_transmit_request(canindex index);

Формирует удаленный запрос принимаемого PDO.

Параметры:

- **index** – индекс коммуникационного объекта запрашиваемого RPDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0.

- CAN_RETOK – RTR запрос размещен в CANopen кэше.
- CAN_ERRET_OBD_NOOBJECT – RPDO с индексом **index** не существует.
- CAN_ERRET_PDO_INVALID – RPDO не действителен.
- CAN_ERRET_PDO_MAP_DEACT – PDO отображение не активировано.
- CAN_ERRET_PDO_ERRMAP – размер данных PDO отображения превышает максимальную длину CAN кадра.
- CAN_ERRET_NODE_STATE – CAN узел находится в не операционном состоянии.
- CAN_ERRET_COMM_SEND – не удалось разместить кадр в CANopen кэше.
- CAN_ERRET_PDO_NORTR – для данного PDO RTR запрещен.

Замечание.

В последних версиях стандарта CiA 301 (4.2.0.xx) бит удаленного запроса для RPDO не поддерживается (зарезервирован). Это не позволяет локально определить возможность формирования удаленного запроса для RPDO.

void receive_can_pdo(canindex index, canframe *cf);

Принимает и обрабатывает RPDO. Активирует асинхронные RPDO (записывает данные в объектный словарь). Синхронные RPDO заносятся в FIFO для последующей активации при поступлении объекта синхронизации SYNC.

Параметры:

- **index** – индекс коммуникационного объекта RPDO.
- ***cf** – принятый CAN кадр, содержащий RPDO.

int16 transmit_can_pdo(canindex index);

Формирует TPDO с типом передачи:

- 0 – ациклические синхронные, заносятся в FIFO для синхронной отправки;
- 254, 255 – асинхронные, отсылаются немедленно;

Параметры:

- **index** – индекс коммуникационного объекта TPDO.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0:

- CAN_RETOK – PDO успешно отправлено в CANopen кэш (типы 254, 255) или размещено в очереди на синхронную отправку (тип 0).
- CAN_ERRET_OBD_NOOBJECT – TPDO с индексом **index** не существует.
- CAN_ERRET_PDO_INVALID – TPDO не действителен.
- CAN_ERRET_PDO_TRTYPE – неподходящий тип передачи TPDO.
- CAN_ERRET_PDO_INHIBIT – TPDO находится в состоянии подавления.
- CAN_ERRET_PDO_MAP_DEACT – PDO отображение не активировано.
- CAN_ERRET_PDO_ERRMAP – размер данных PDO отображения превышает максимальную длину CAN кадра.
- CAN_ERRET_NODE_STATE – CAN узел находится в не операционном состоянии.
- CAN_ERRET_COMM_SEND – не удалось разместить кадр в CANopen кэше.

void transmit_rtr_pdo(canindex index);

Формирует TPDO, для которого получен удаленный запрос. В зависимости от типа передачи PDO заносится в FIFO для синхронной отправки (тип <= 252, синхронное) или отсылается немедленно (тип >= 253, асинхронное).

Параметры:

- **index** – индекс коммуникационного объекта TPDO.

void process_sync_pdo(unsigned8 sc);

Обрабатывает синхронные RPDO и TPDO. Вызывается из функции обработчика объекта синхронизации SYNC.

Параметры:

- **sc** – текущее значение SYNC счетчика.

void can_init_pdo_proc(void);

Инициализирует данные модуля.

4.2.25 Модуль lib_sdo_proc.c

Размещается в директории \CANopen.
Осуществляет разборку, а также сборку и отправку SDO кадров.

void parse_sdo(struct cansdo *sd, canbyte *data);

Производит разборку поля данных CAN кадра SDO протокола.

Параметры:

- ***sd** – информация о принятом SDO кадре в разобранном виде.
- ***data** – указатель на поле данных принятого кадра SDO протокола.

int16 send_can_sdo(struct cansdo *sd);

Осуществляет сборку и отправку CAN кадра SDO протокола.

Параметры:

- ***sd** – информация об отсылаемом SDO кадре в разобранном виде.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0:

- CAN_RETOK – SDO кадр размещен в CANopen кэше.
- CAN_ERRET_NODE_STATE – CAN узел находится состоянии останова или инициализации.
- CAN_ERRET_COMM_SEND – не удалось разместить кадр в CANopen кэше.

void abort_can_sdo(struct sdoixs *si, unsigned32 abortcode);

Производит отправку кадра «Abort SDO Transfer» протокола.

Параметры:

- ***si** – индекс и субиндекс прикладного CANopen объекта.
- **abortcode** – значение Abort кода.

4.2.26 Модуль lib_server.c

Размещается в директории \CANopen.
Выполняет серверные транзакции обмена данными для SDO протокола.

void receive_can_sdo(canframe *cf);

Принимает и обрабатывает CAN кадр SDO протокола.

Параметры:

- ***cf** – принятый кадр SDO протокола.

void can_server_control(void);

Контролирует таймаут операции обмена данными SDO протокола. Вызывается из CANopen таймера.

void can_init_server(void);

Инициализирует данные модуля.

4.2.27 Модуль lib_srdo_object.c

Размещается в директории \CANopen.
Поддерживает объектный словарь и программные алгоритмы безопасного протокола EN50325-5. Вариант модуля lib_srdo_dummy_secure.c поддерживает только безопасный NMT режим.

int16 get_srdo_comm_bytes_objsize(canindex index, cansubind subind);
int16 see_srdo_comm_access(canindex index, cansubind subind);
int16 get_srdo_comm_objtype(canindex index, cansubind subind);
int16 read_srdo_comm_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_srdo_comm_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю коммуникационных SRDO объектов. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают объекты, заданные индексами (для двух SRDO):
1301_h, 1302_h – коммуникационные параметры SRDO.

int16 get_srdo_map_bytes_objsize(canindex index, cansubind subind);
int16 see_srdo_map_access(canindex index, cansubind subind);
int16 get_srdo_map_objtype(canindex index, cansubind subind);
int16 read_srdo_map_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_srdo_map_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю объектов SRDO отображения. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают объекты, заданные индексами (для двух SRDO):
1381_h, 1382_h – параметры отображения SRDO.

int16 get_sr_config_bytes_objsize(canindex index, cansubind subind);
int16 see_sr_config_access(canindex index, cansubind subind);
int16 get_sr_config_objtype(canindex index, cansubind subind);
int16 read_sr_config_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_sr_config_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю конфигурационных SRDO объектов. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают объекты, заданные индексами:
1300_h – Статус широковещательной GFC команды.
$13E0_h$ – Дополнительные параметры GFC команды.
$13E1_h$ – Рабочий статус безопасного протокола.
$13FE_h$ – Конфигурация SR устройства достоверна.
$13FF_h$ – Подписи SRDO параметров.

void produce_sr_gfc(void);

Формирует и отправляет в CAN сеть GFC команду.

void consume_sr_gfc(void);

Принимает GFC команду.

void control_srdo(void);

Осуществляет управление работой безопасного протокола. Контролирует прием SRDO. Управляет передачей SRDO объектов. Вызывается из CANopen таймера.

void receive_can_srdo(canindex index, canframe *cf);

Принимает и обрабатывает SRDO.

Параметры:

- **index** – индекс коммуникационного объекта SRDO.
- ***cf** – принятый CAN кадр, содержащий SRDO.

void run_sr_operations(void);

Запускает работу безопасного протокола. Вызывается из функции start_node(), которая переводит CANopen узел в операционное состояние.

unsigned8 get_sr_runstatus(void);

Возвращает статус безопасного протокола: ON/OFF.

void can_init_srdo(void);

Инициализирует данные безопасного протокола.

4.2.28 Модули re_store_oneK_segm.c и re_store_twoK_segm.c

Размещаются в директории _No_optimization.
Реализуют сохранение параметров устройства и CANopen объектов в энергонезависимой памяти микроконтроллера. Для ОС Windows страницы энергонезависимой памяти эмулируются в статических массивах. При сохранении используется буфер, который размещается в стеке. Для микроконтроллерных платформ необходимо обеспечить соответствующий размер стека, а также отключить любую оптимизацию программного кода модуля.
Модуль re_store_oneK_segm.c использует для сохранения четыре страницы флэш памяти размером 1 Килобайт каждая.
Модуль re_store_twoK_segm.c использует для сохранения две страницы флэш памяти размером 2 Килобайта каждая либо четыре попарно сгруппированных страницы размером 1 Килобайт каждая.

int16 get_re_store_bytes_objsize(canindex index, cansubind subind);
int16 see_re_store_access(canindex index, cansubind subind);
int16 get_re_store_objtype(canindex index, cansubind subind);
int16 read_re_store_objdict(canindex index, cansubind subind, canbyte *data);
int16 write_re_store_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю объектов сохранения/восстановления. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают коммуникационные объекты, заданные индексами:
1010_h – сохранение параметров в энергонезависимой памяти.
1011_h – восстановление значений параметров по умолчанию либо загрузка параметров.

void load_group_stored(cansubind subind);

Производит загрузку группы параметров из энергонезависимой памяти.

Параметры:

- **subind** – субиндекс, который задает группу загружаемых параметров CAN_SUBIND_STORE_GROUP_[1..4].

void save_group_stored(cansubind subind);

Производит сохранение группы параметров в энергонезависимой памяти.

Параметры:
- **subind** – субиндекс, который задает группу сохраняемых параметров CAN_SUBIND_STORE_GROUP_[1..4].

int16 can_process_saved(unsigned16 mask);

Производит чтение из энергонезависимой памяти и запись в объектный словарь устройства сохраненных значений параметров.

Параметры:
- **mask** – битовая маска. Биты могут объединяться по "логическому или".
  CAN_MASK_LOAD_COMM_1 – загружаются коммуникационные параметры, которые не зависят от номера CAN узла.
  CAN_MASK_LOAD_COMM_2 – загружаются коммуникационные параметры, зависимые от номера CAN узла (CAN идентификаторы).
  CAN_MASK_LOAD_APPL – загружаются параметры приложения.

Возвращаемые значения: нормальное завершение = 0;
- CAN_RETOK – нормальное завершение.

int16 get_flash_nodeid(void);

Производит чтение из энергонезависимой памяти номера CAN узла устройства.

Возвращаемые значения: сохраненное значение >= 0; ошибка < 0.
- >= 0 – номер CAN узла (младший байт).
- CAN_ERRET_FLASH_DATA – ошибка данных в энергонезависимой памяти.
- CAN_ERRET_FLASH_VALUE – параметр не сохранялся в энергонезависимой память.

int16 get_flash_bitrate_index(void);

Производит чтение из энергонезависимой памяти индекса битовой скорости CAN сети.

Возвращаемые значения: сохраненное значение >= 0; ошибка < 0.
- >= 0 – индекс битовой скорости CAN сети (младший байт).
- CAN_ERRET_FLASH_DATA – ошибка данных в энергонезависимой памяти.
- CAN_ERRET_FLASH_VALUE – параметр не сохранялся в энергонезависимой память.

void monitor_flash_operations(void);

Диспетчер операций с энергонезависимой памятью. Вызывается из главного цикла программы.

void flush_store_operations(void);

Завершает выполнение всех ожидающих операций с энергонезависимой памятью. Вызывается из функции stop_hardware().

int16 check_init_flash(void);

Устаревшая функция.

Возвращаемые значения: нормальное завершение = 0;
- CAN_RETOK – нормальное завершение.

void can_init_re_store(void);

Инициализирует данные модуля.

5. Прикладной CANopen профиль CiA 401

Модули приложения размещаются в директории CANopen_WinAppl_CiA401\ Windows_CiA401_Project\src для ОС Windows или CiA401_Starter_Kit\MCU_CiA401_Project\ Application для микроконтроллерной платформы. Расположение файлов приводится относительно этих директорий.

5.1 Параметры сборки приложения

5.1.1 Параметры CANopen устройства

Определены в файлах __CiA401_device.h и __CiA401_config.h в директории \include.

- CAN_MASK_NETWORKS
  Битовая маска физических CAN сетей. Определяет конфигурацию CAN контроллеров устройства. Единичное значение бита маски указывает наличие соответствующей CAN сети.
- CAN_NODEID_SLAVE
  Номер CANopen узла, устанавливаемый по умолчанию. Допустимые значения от 1 до 127.
- CAN_BITRATE_INDEX
  Индекс битовой скорости CAN сети, который устанавливается по умолчанию.
- CAN_DATALINK_OUTOFF
  Задает режим работы с CAN сетью при переполнении выходного CANopen кэша.
  FALSE – режим с вызовом функции can_cache_overflow(…) и обработкой ошибки согласно установкам объекта 1029_h.
  TRUE – логическое отключение канального уровня CAN сети по записи. Данный режим используется, когда устройство должно сохранять автономную работоспособность при физическом отключении от CAN сети.
- CAN_LED_INDICATOR
  Тип светодиодной индикации состояния устройства.
  COMBINED – используется совмещенный красно/зеленый светодиод.
  SEPARATE – применяются отдельно красный и зеленый светодиоды.
- CAN_TPDO_LOCAL_LOOPBACK
  Режим дополнительной обработки передаваемых TPDO.
  OFF – без дополнительной обработки TPDO.
  ON – после передачи TPDO локально осуществляется его прием. Данный режим может привести к неожиданным результатам и должен использоваться с осторожностью.
- CANOPEN_RECV_FIFO
  Режим обработки входящих CAN кадров канального уровня. Предполагает различный API обработчика CAN кадров. Используется в микроконтроллерных приложениях.
  OFF – принятые CAN кадры направляются в CANopen без дополнительной буферизации.
  ON – принятые CAN кадры сохраняются в FIFO и лишь затем направляются в CANopen.
- CAN_HARD_ACCEPTANCE_FILTER
  Определяет число уровней масочного фильтра входящих кадров CAN контроллера.
  AFSINGLE – одноуровневый фильтр.
  AFDUAL – двухуровневый фильтр (при поддержке в CHAI).
  AFTRIPLE – трехуровневый фильтр для EN50325-5 (при поддержке в драйвере канального уровня CHAI).

- CAN_DEVICE_TYPE
  Задают совместно тип устройства и его функциональность (объект 1000_h).
- CAN_MAN_STATUS
  Регистр статуса от производителя устройства (объект 1002_h).
- CAN_VENDOR_ID
  Код CiA, присвоенный производителю устройства (объект $1018_h sub1_h$).
- CAN_PRODUCT_CODE
  Код изделия, задаваемый производителем (объект $1018_h sub2_h$).
- CAN_REV_NUM
  Версия устройства, задаваемая производителем (объект $1018_h sub3_h$).
- CAN_SERIAL_NUMBER
  Серийный номер CANopen устройства (объект $1018_h sub4_h$).
- DIGITAL_INP8_BLOCKS_MAX
  Максимальное число блоков цифрового ввода по 8 разрядов.
- DIGITAL_OUT8_BLOCKS_MAX
  Максимальное число блоков цифрового вывода по 8 разрядов.
- ANALOG_INPUT_CHANS_MAX
  Максимальное число каналов аналогового ввода.
- ANALOG_OUTPUT_CHANS_MAX
  Максимальное число каналов аналогового вывода.
- CAN_TIMERUSEC
  Период CANopen таймера в микросекундах. Значение параметра должно быть не менее 100. Рекомендуемый период таймера 1..5 миллисекунд (значение параметра от 1000 до 5000) при поддержке протокола EN50325-5 и до 50 миллисекунд при работе только CiA 301. Период CANopen таймера можно изменять в зависимости от требований к разрешению различных временны*х* CANopen объектов (SYNC, PDO, времена подавления, коммуникационные SRDO объекты и др.).
- WATCHDOG_MS_DEFAULT
  Значение тайм-аута внутреннего сторожевого таймера в миллисекундах. Используется во время штатной работы приложения.
- WATCHDOG_MS_RESET
  Значение тайм-аута внутреннего сторожевого таймера в миллисекундах. Используется во время (пере)инициализации устройства.
- CAN_TIMEOUT_SERVER
  Таймаут базовой SDO транзакции сервера. Задается в микросекундах. В базовой SDO транзакции сервер ожидает запрос очередного сегмента данных от клиента.
- CAN_BOOTUP_DELAY
  Определяет задержку перехода устройства в пред-операционное состояние и выдачи сообщения загрузки (Boot-up протокол). Задается в микросекундах. Нулевое значение отключает задержку.
- CAN_OPERATIONAL_DELAY
  Определяет задержку автономного перехода устройства в операционное состояние. Задается в микросекундах. Нулевое значение отключает задержку и устройство остается в пред-операционном состоянии.
- CAN_HBT_PRODUCER_MS
  Значение по умолчанию для периода сердцебиения поставщика в миллисекундах. Инициализирует объект 1017_h.
- CAN_EMCY_INHIBIT_100MCS

Значение по умолчанию для времени подавления посылок объекта EMCY.
Инициализирует объект 1015_h.

- CAN_RPDO_TRTYPE
  Значение по умолчанию для типа передачи RPDO. Используется для инициализации субиндекса 2 объектов 1400_h..1403_h – коммуникационные параметры принимаемых PDO.
- CAN_TPDO_TRTYPE
  Значение по умолчанию для типа передачи TPDO. Используется для инициализации субиндекса 2 объектов 1800_h..1803_h – коммуникационные параметры передаваемых PDO.
- CAN_TPDO_INHIBIT_100MCS
  Значение по умолчанию для времени подавления посылок TPDO. Инициализирует субиндекс 3 объектов 1800_h..1803_h – коммуникационные параметры передаваемых PDO.
- CAN_RPDO_ET_MS
  Значение по умолчанию для таймера события RPDO. Инициализирует субиндекс 5 объектов 1400_h..1403_h – коммуникационные параметры принимаемых PDO.
- CAN_TPDO_ET_MS
  Значение по умолчанию для таймера события TPDO. Инициализирует субиндекс 5 объектов 1800_h..1803_h – коммуникационные параметры передаваемых PDO.
- CAN_TPDO_SYNC_START
  Значение по умолчанию для начального значения SYNC счетчика TPDO. Инициализирует субиндекс 6 объектов 1800_h..1803_h – коммуникационные параметры передаваемых PDO.

5.1.2 Параметры энергонезависимой памяти

Параметры для работы с энергонезависимой памятью определены в файле, имя которого задается параметром STORE_PARAMS_CONFIG в __CiA401_device.h, директория \include.

- FLASH_MODE_EMULATED
  Если данный параметр определен, энергонезависимая память эмулируется в статическом массиве данных.
- FLASH_MODE_1K_PAGE
  Только для модуля re_store_twoK_segm.c, в котором используются 2 Кб сегменты памяти. Если данный параметр определен, физические страницы энергонезависимой памяти микроконтроллера размером по 1 Кб группируются попарно.
- CIA301_RE_STORE_COMMUNICATION
  Если данный параметр определен, осуществляется сохранение и восстановление коммуникационных параметров (объекты 1010_h и 1011_h, субиндексы 02_h и 04_h).
- FLASH_PAGE_SIZE_BYTES
  Размер страницы памяти программ в байтах (1024 или 2048 байт).
- SIZE_BYTES_*
  Полный размер в байтах соответствующей группы параметров. Является суммарным размером всех блоков группы плюс один байт. Выравнивается до четного числа байт.
- STORE_ITEMS_*
  Число блоков данных (индексов объектного словаря) в соответствующей группе параметров.

5.1.3 Дополнительные параметры

Определены в файле CiA401_defines.h в директории \include.

- CAN_EMCY_INT_DISABLED
  Код срочного сообщения: при переходе устройства в операционное NMT состояние общее

прерывание для аналоговых входов запрещено (предупреждение).

- CAN_EMCY_WDT_REBOOT
  Дополнительный EMCY код: произойдет перезагрузка устройства по сторожевому таймеру. Сообщение передается в CAN сеть при наличии возможности.
- CAN_EMCY_WDT_FAILED
  Дополнительный EMCY код: ошибка (ре)конфигурирования сторожевого таймера.
- CAN_EMCY_ERROR_STATE
  Дополнительный EMCY код: устройство находится в режиме ошибки.
- CAN_NOF_ERRBEH_SUBIND
  Максимальный субиндекс объекта 1029_h – поведение CAN устройства при возникновении серьезных ошибок.
- CAN_SUBIND_ERRBEH_COMM
  Значение субиндекса коммуникационной ошибки объекта 1029_h.
- MASK_*
  Задают битовые маски, которые используются в прикладном профиле.
- INDEX_*
  SUBIND_*
  Задают индексы и субиндексы коммуникационных и прикладных объектов, которые используются в прикладном профиле устройства.

5.2 Модули приложения

5.2.1 Модули определений и прототипов

Размещаются в директории \include.

- __CiA401_config.h – конфигурационные параметры устройства (число каналов, таймеры, параметры инициализации, приоритеты прерываний и др.).
- __CiA401_device.h – основные прикладные параметры устройства.
- __CiA401_re_store.h – параметры для работы с энергонезависимой памятью.
- CiA401_defines.h – параметры для формирования объектного словаря и прикладного профиля устройства.
- CiA401_defunc.h – прототипы видимых функций.
- CiA401_genhead.h – модуль заголовков и подключений. Задает номер версии приложения.
- CiA401_globals.h – список внешних (глобальных) переменных.
- CiA401_header.h – базовый заголовочный модуль для файлов приложения.
- CiA401_limits.h – определяет допустимые пределы значений параметров.
- CiA401_project_codes.h – задает уникальные коды проектов устройства.
- CiA401_structures.h – содержит определения структур данных приложения.
- lib_application.h, ARM_application.h – заголовочный файл, посредством которого CANopen библиотека получает доступ к параметрам и данным приложения.

5.2.2 Модуль __CiA401_WD_main.c

Размещается в корневой директории приложения. Заменяет библиотечный модуль __lib_main.c. Обеспечивает поддержку сторожевого таймера для микроконтроллерных приложений.

void push_watchdog(void);

Перезагружает сторожевой таймер в микроконтроллерных приложениях.

void set_watchdog_params_ms(unsigned16 wdt)

Устанавливает тайм-аут сторожевого таймера и разрешает его работу. Используется в микроконтроллерных приложениях.

Параметры:

- **wdt** – тайм-аут сторожевого таймера в миллисекундах.

5.2.3 Модуль __CiA401_devices.c

Размещается в директории \Device.
Поддерживает объекты идентификации и описания устройства.

unsigned32 read_dev_type_object(canindex index, cansubind subind);

Определяет тип устройства, регистр статуса производителя и объект идентификации. Поддерживает коммуникационные объекты, заданные индексами:
1000_h – тип устройства.
1002_h – регистр статуса от производителя.
1018_h – объект идентификации устройства.

Параметры:

- **index** – индекс объекта.
- **subind** – субиндекс объекта.

Возвращаемое значение:
- Значение объекта.

void read_dev_string_object(canindex index, cansubind subind, canbyte *data);

Формирует строковые описания названия устройства, версии железа и программного обеспечения. Поддерживает коммуникационные объекты, заданные индексами:
1008_h – название устройства от производителя.
1009_h – версия «железа» устройства от производителя.
$100A_h$ – версия программного обеспечения устройства от производителя.
Параметры:
- **index** – индекс объекта.
- **subind** – субиндекс объекта.
- ***data** – байтовый указатель на размещаемые данные типа "vis-string".

5.2.4 Модуль __CiA401_init.c

Размещается в директории \Device.
Определяет параметры для инициализации устройства.

cannode get_node_id(void);

Возвращает номер узла CANopen устройства. Параметр считывается из энергонезависимой памяти либо используется значение по умолчанию.
Возвращаемое значение:
- Номер узла CANopen устройства от 1 до 127.

int16 check_supported_bitrates(unsigned8 br);

Проверяет допустимость (поддержку устройством) заданной битовой скорости.
Параметры:
- **br** – индекс битовой скорости.

Возвращаемые значения: нормальное завершение = 0; ошибка < 0:
- CAN_RETOK – битовая скорость поддерживается.
- CAN_ERRET_BITRATE – битовая скорость не поддерживается.

unsigned8 get_bit_rate_index(void);

Возвращает значение индекса битовой скорости устройства. Параметр считывается из энергонезависимой памяти либо используется значение по умолчанию.
Возвращаемое значение:
- Индекс битовой скорости CAN сети из числа допустимых.

unsigned32 get_serial_number(void);

Возвращает серийный номер устройства, заданный по умолчанию.
Возвращаемое значение:
- Серийный номер устройства (объект $1018_h sub4_h$).

void get_device_config(void);

Считывает аппаратную конфигурацию устройства. Вызывается однократно до программной инициализации устройства. При пере-инициализации устройства с использованием NMT команд вызов этой функции не осуществляется.

5.2.5 Модуль CiA401_globals.c

Размещается в директории \Device.
Определяет внешние (глобальные) переменные и структуры данных прикладного профиля.

5.2.6 Модуль CiA401_standev.c

Размещается в директории \Profile.
Реализует прикладные профили стандартизированных устройств.

void process_digital_input(unsigned8 blk, unsigned8 data);

Обработка первичных данных 8-разрядной группы цифровых входов согласно профилю CiA 401. Возможна инициализация передачи данных посредством TPDO.
Параметры:
- **blk** – номер группы цифровых входов от 0 до (DIGITAL_INP8_BLOCKS_MAX-1).
- **data** – первичные данные группы цифровых входов.

void process_analog_input(unsigned8 chan, int32 data);

Обработка первичных данных аналогового ввода согласно профилю CiA 401. Возможна инициализация передачи данных посредством TPDO.
Параметры:
- **chan** – номер канала аналогового ввода от 0 до (ANALOG_INPUT_CHANS_MAX-1).
- **data** – первичные аналоговые данные.

void write_digital_output(unsigned8 blk);

Обработка по профилю CiA 401 и вывод в устройство данных 8-разрядной группы цифровых выходов.
Параметры:
- **blk** – номер группы цифровых выходов от 0 до (DIGITAL_OUT8_BLOCKS_MAX-1).

void write_analog_output(unsigned8 chan);

Обработка по профилю CiA 401 и вывод в устройство аналоговых данных.
Параметры:
- **chan** – номер канала аналогового вывода от 0 до (ANALOG_OUTPUT_CHANS_MAX-1).

int32 server_get_standev_objsize(canindex index, cansubind subind);
int16 server_see_standev_access(canindex index, cansubind subind);
int16 server_get_standev_objtype(canindex index, cansubind subind);
int16 server_read_standev_objdict(canindex index, cansubind subind, canbyte *data);
int16 server_write_standev_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю объектов стандартизированных CANopen профилей. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают объекты, заданные индексами:
$6000_h..9FFF_h$ – профили стандартизированных устройств.

void enter_error_state(void);
void exit_error_state(void);

Функции входа и выхода устройства из режима ошибки согласно профилю CiA 401.

void application_standev_timer(void);

Обработчик таймера для стандартизированного прикладного профиля. Вызывается из CANopen таймера.

void slave_init_standev_application(void);

Инициализация данных и объектов приложения для стандартизированного профиля CiA 401.

void slave_init_standev_communication(void);

Инициализация коммуникационных данных и объектов для стандартизированного профиля CiA 401.

5.2.7 Модуль CiA401_manspec.c

Размещается в директории \Profile.
Формирует расширение профилей стандартизированных устройств либо реализует специализированные прикладные профили.

int32 server_get_manspec_objsize(canindex index, cansubind subind);
int32 server_put_manspec_objsize(canindex index, cansubind subind, int32 size);
int16 server_see_manspec_access(canindex index, cansubind subind);
int16 server_get_manspec_objtype(canindex index, cansubind subind);
int16 server_read_manspec_objdict(canindex index, cansubind subind, canbyte *data);
int16 server_write_manspec_objdict(canindex index, cansubind subind, canbyte *data);

Функции доступа к словарю объектов расширения профилей стандартизированных устройств либо специализированных прикладных профилей. Назначение и параметры функций приведены в разделе 4.1.
Поддерживают объекты, заданные индексами:
$2000_h..5FFF_h$ – профили не стандартизированных устройств.

void application_manspec_timer(void);

Обработчик таймера для не стандартизированного прикладного профиля. Вызывается из CANopen таймера.

void slave_init_manspec_application(void);

Инициализация данных и объектов приложения специализированного CANopen профиля или расширения.

void slave_init_manspec_communication(void);

Инициализация коммуникационных данных и объектов специализированного CANopen профиля или расширения.

5.2.8 Модуль CiA401_control.c

Размещается в директории \Profile.
Реализует управляющие функции прикладного профиля.

void set_transmit_pdo(canindex index);

Установ TPDO на передачу. Обеспечивает однократную передачу TPDO при изменении значений нескольких отображенных в него параметров.

void set_traffic_total_value(unsigned16 tout);

Устанавливает параметры контроля трафика временной метки TIME.
Параметры:
- **tout** – таймаут трафика TIME в миллисекундах.

void consume_time(canframe *cf);

Вызывается при получении объекта временной метки TIME. Участвует в контроле выполнения программы с использованием сторожевого таймера.
Параметры:
- ***cf** – принятый CAN кадр, содержащий объект TIME.

void start_hardware(void);

Запуск устройства после (пере)инициализации. В этой функции может осуществляться загрузка групп параметров из энергонезависимой памяти.

void stop_hardware(void);

Останов устройства до (пере)инициализации.

void application_timer_routine(void);

Таймер прикладного профиля. Вызывается из CANopen таймера.

void application_monitor_routine(void);

Монитор (главный цикл) приложения. Вызывается из CANopen монитора.

void application_init_device_routine(void);

Функция начальной инициализации прикладных данных и параметров устройства при запуске (включении питания). Вызывается однократно до программной инициализации устройства. При пере-инициализации устройства с использованием NMT команд вызов этой функции не осуществляется.