МАРАФОН
CAN технология

CANopen библиотека

Адаптированный **master** для ОС Windows

Руководство программиста

Код проекта: **0003**$_h$

Москва, 2015

Оглавление

1. Общие положения.

Данный документ является дополнением к основному руководству по CANopen библиотеке. В нем приводится описание адаптированного master приложения (мастера) для ОС Windows. Адаптированный мастер может использоваться в качестве прототипа для разработки приложений, которые обеспечивают работу сетей CANopen устройств в составе систем контроля и управления.

2. Изменения в версиях программ.

Версия 2.2

В состав библиотеки включены адаптированные slave и master версии для ОС Windows.

Версия 2.3

Адаптированный CANopen master реализован на основе версии 2.3 CANopen библиотеки.

2.1 Управление версиями адаптированных модулей.

Каждый библиотечный модуль адаптированной версии заключается в условный макрос вида:

```
#if CHECK_VERSION_CANLIB(2, 2, 0)
    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(1, 3, 0)
    код модуля приложения
#endif
```

3. Структура и параметры адаптированного мастера.

Адаптированный мастер используется в качестве прототипа для разработки приложений, которые обеспечивают работу сетей CANopen устройств в составе систем контроля и управления.

Прикладная программа мастера поддерживает следующую функциональность:

1. Создает асинхронный регистратор, который используется для ведения журналов событий как самого приложения, так и узлов CANopen сети.
2. Формирует базу CANopen устройств на основе информации, содержащейся в конфигурационном файле (активные узлы).
3. Инициализирует все узлы CANopen сети, в том числе не входящие в конфигурационную базу.
4. Проверяет фактическое наличие в сети всех активных узлов.
5. Запускает в каждом активном узле протокол сердцебиения.
6. Переводит активные узлы в операционное состояние.
7. Отслеживает состояние всех активных узлов сети, обеспечивая механизм plug-and-play при отключении и последующем подключении CANopen устройства.

3.1 Структура модулей мастера.

Программные модули мастера размещаются в директории CANopen_WinMaster\src. Файлы проекта размещены в поддиректориях:

- \application – модули мастер-приложения для работы с сетью CANopen устройств.
- \CANopen – адаптированные мастер модули CANopen.
- \confile – модули для работы с конфигурационным файлом.
- \include – заголовочные модули определений и прототипов.
- \logger – модули асинхронного регистратора.

3.1.1 Модули определений и прототипов.

- __application_defines.h – определение параметров (констант) приложения.
- __logger_defines.h – определение параметров (констант) регистратора.
- config_defines.h – определение параметров обработчика конфигурационного файла.
- master_defines.h – определение параметров (констант) CANopen.
- master_defunc.h – определение прототипов функций.
- master_genhead.h – модуль заголовков и подключений.
- master_globals.h – список внешних (глобальных) переменных.
- master_header.h – базовый заголовочный модуль.
- master_macros.h – определение макросов.
- master_structures.h – определение структур данных.
- master_typedefs.h – определение типов данных.

3.2 Структуры данных мастера.

3.2.1 Типы данных CANopen мастера.

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.

3.2.2 Структуры данных.

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 суб-индекс прикладного объекта.
};
Структура **sdoixs** определяет индекс и суб-индекс прикладного CANopen объекта для SDO протокола (мультиплексор SDO протокола).

struct cansdo {
unsigned8 cs команда SDO протокола.
struct sdoixs si индекс и суб-индекс словаря 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 sdostatus {
int16 state статус во время и после завершения SDO транзакции клиента.
unsigned32 abortcode SDO аборт код, если по завершении транзакции **state** принимает значение CAN_TRANSTATE_SDO_SRVABORT.
};
Структура **sdostatus** размещает информацию о статусе SDO транзакции клиента.

struct sdoclttrans {

unsigned8 adjcs	команда SDO протокола, которой сервер должен отвечать на запрос клиента.
struct sdostatus ss	статус SDO транзакции.
struct cansdo sd	информация SDO кадра в разобранном виде.
unsigned32 rembytes	число оставшихся для передачи байт прикладного объекта.

};

Структура **sdoclttrans** обеспечивает поддержку базовой SDO транзакции клиента (запрос от клиента, прием и обработка ответа сервера). Здесь же ведется подсчет числа оставшихся для передачи байт, что обеспечивает управления полным SDO обменом.

struct sdocltbasic {

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).
struct sdoclttrans ct	структура поддержки базовой транзакции клиента.

};

Структура **sdocltbasic** размещает данные, необходимые для реализации базовой SDO транзакции на стороне клиента.

struct sdocltappl {

unsigned8 operation	базовый режим передачи SDO (upload / download).
unsigned32 datasize	размер данных в байтах.
canbyte *datapnt	байтовый указатель на локальный буфер.
struct sdoixs si	индекс и суб-индекс прикладного CANopen объекта.
struct sdostatus ss	статус SDO транзакции.

};

Структура **sdocltappl** служит для взаимодействия с приложением клиента и используется при обмене данными с помощью SDO протокола.

struct canframe {

unsigned32 id	CAN–ID.
unsigned8 data[8]	поле данные CAN кадра.
unsigned8 len	реальная 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 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 отсылаемых CAN кадров.

struct canopennode {

unsigned8 node_status	статус узла, определяемый в конфигурационном файле. ON — узел описан в конфигурационном файле (активный узел), OFF — конфигурационный файл не содержит описания узла (пассивный узел).

unsigned8 nmt_state	NMT состояние узла.
unsigned32 DeviceType	тип устройства из конфигурационного файла (объект 1000_h).
unsigned32 VendorID	код производителя устройства из конфигурационного файла (объект $1018_h sub1_h$).
unsigned32 ProductCode	код изделия из конфигурационного файла (объект $1018_h sub2_h$).
unsigned32 Revision	версия устройства из конфигурационного файла (объект $1018_h sub3_h$).
unsigned32 Serial	серийный номер устройства из конфигурационного файла (объект $1018_h sub4_h$).
unsigned16 maskdev	маска описания устройства по итогам обработки конфигурационного файла.
unsigned32 ecpcnt	счетчик протокола сердцебиения.

};

В структуре **canopennode** содержится описание и статус каждого узла CANopen сети.

struct eventlog {

time_t ts	временн*а*я метка события.
unsigned8 node	номер CAN узла, в котором было порождено событие. для событий мастера равен нулю.
unsigned8 class	класс события.
unsigned8 type	тип события (info, warning, error и т.д.)
unsigned8 misc	зарезервировано (выравнивание).
int16 code	код события.
int32 info	дополнительная информация о событии.

};

Структура **eventlog** содержит информацию о зарегистрированном событии.

struct eventcache {

int16 busy	семафор занятия кэша (инкрементный).
unsigned8 capture	флаг захвата кэша и занесения в него данных.
struct eventlog ev	событие регистратора.

};

Структура **eventcache** формирует кэш для размещения событий регистратора.

union numbers {

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.

3.3 Параметры мастера.

Параметры адаптированного мастера определены в файлах: __application_defines.h и __logger_defines.h, размещаемых в директории \src\include.

3.3.1 Параметры мастер-приложения.

- CAN_NETWORK_CONTROLLER
  Номер канала контроллера CAN сети. Значение по умолчанию.
- CAN_BITRATE_INDEX
  Индекс битовой скорости CAN сети. Значение по умолчанию.
- CAN_TIMERUSEC
  Период CANopen таймера в микросекундах. Значение параметра должно быть не менее 100. Рекомендуемый период таймера 1..10 миллисекунд (значение параметра от 1000 до 10000).
- CAN_TIMEOUT_RETRIEVE
  Таймаут получения данных из CAN сети для базовой SDO транзакции клиента. Значение по умолчанию. Задается в микросекундах. В базовой SDO транзакции клиент ожидает ответ от сервера.
- CAN_TIMEOUT_READ
  Таймаут чтения приложением принятых из CAN сети данных для базовой SDO транзакции клиента. Задается в микросекундах.
- CAN_HBT_PRODUCER_MS
  Периода протокола сердцебиения в миллисекундах, который используется для конфигурирования всех активных CANopen узлов.
- CAN_HBT_CONSUMER_MS
  Период сердцебиения мастера в миллисекундах. Задает таймаут сердцебиения активных CANopen узлов. При наступлении этого таймаута мастер устанавливает NMT состояние узла как «не определенное» (CAN_NODE_STATE_UNCERTAIN). Если до истечения еще одного периода сердцебиения CANopen узел не возобновляет посылку кадров протокола сердцебиения, осуществляется его пере-инициализация.
- CAN_CONFIG_NODE_MS
  Таймаут конфигурирования CANopen узла. Отсчитывается с момента получения boot-up сообщения от узла. Включает полное время конфигурирования до приема мастером кадра сердцебиения узла, переведенного в операционное NMT состояние. По истечении таймаута осуществляется пере-инициализация узла.
- CAN_RESET_NODE_MS
  Таймаут пере-инициализации CANopen узла. Отсчитывается с момента отправки мастером NMT команды Reset Node и до получения boot-up сообщения от узла. По истечении таймаута пере-инициализация активного узла осуществляется заново.
- MASTER_CONFIG_FILE_NAME
  Имя конфигурационного файла.
- MASTER_CONFIG_FILE_VERSION
  Версия конфигурационного файла.
- MASTER_LOG_FILE_NAME_DEF
  Имя файла журнала. Значение по умолчанию.

- CAN_NETWORK_MIN
  CAN_NETWORK_MAX
  Минимальный и максимальный номера каналов CAN контроллера (CAN драйвера).
- STR_FILE_NAME_SIZE
  Размер полного имени файла (ASCIIZ).
- STR_LINE_SIZE
  Максимальная длина ASCIIZ строк (конфигурационного файла).
- STR_TS_SIZE
  Длина ASCIIZ строки временной метки.

3.3.2 Параметры регистратора.

- EVENT_CACHE_SIZE
  Размер кэша регистратора. Один элемент (нулевой) используется для регистрации переполнения кэша.
- EVENT_FIFO_SIZE
  Размер FIFO регистратора. Размещает до (EVENT_FIFO_SIZE-1) событий.
- EVENT_NODE_MASTER
  Для регистрации событий, порожденных в самом CANopen мастере, используется нулевой номер CAN узла.
- EVENT_CLASS_*
  Классы событий регистратора.
- EVENT_TYPE_*
  Типы событий регистратора (info, warning, error и т.д.).
- EVENT_CODE_*
  Коды отдельных событий. Для событий различных классов могут иметь одинаковые значения.

4. CANopen модули адаптированной мастер версии.

Адаптированные мастер модули CANopen размещаются в корневой директории CANopen_WinMaster\src и поддиректории src\CANopen.

4.1 Модуль can_master_system_winlib.c.

Размещается в корневой директории CANopen_WinMaster\src. Содержит системно-зависимые функции.

void can_sleep(int32 microseconds);

Функция временн*о*й задержки.

Параметры:

- **microseconds** – временн*а*я задержка в микросекундах. Точное время задержки определяется разрешением соответствующего таймера системы. Любое положительное значение аргумента функции должно обеспечивать не нулевую задержку.

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

Инициализация CANopen таймера. Период таймера в микросекундах задается константой CAN_TIMERUSEC. Поток CANopen таймера должен обладать высоким приоритетом. Поскольку метод работы диспетчера ОС может не гарантировать непрерывного выполнения этого потока, код обработчика таймера формируется как единая критическая секция.

Параметры:

- **handler** – функция обработчика таймера: имеет прототип: void background(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 CANopen мастера с помощью макроса CAN_CRITICAL_INIT, определенного в модуле master_defines.h.

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

Функции входа и выхода из критической секции. Служат для обеспечения атомарности семафорных операций при использовании мастера в многопоточной среде, когда CANopen таймер и обработчик CAN кадров запускаются как отдельные потоки (нити). Функции внедряются в код с помощью макросов CAN_CRITICAL_BEGIN и CAN_CRITICAL_END, определенных в модуле master_defines.h.

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.

4.2 Модуль __master_main.c

Размещается в корневой директории CANopen_WinMaster\src. Содержит запускаемую на выполнение функцию программы main(..) и монитор (главный цикл) программы.

4.3 Модуль master_backinit.c

Реализует функции инициализации CANopen мастера. Формирует и поддерживает диспетчер таймера и CANopen монитор.

int16 start_can_master(void);

Осуществляет инициализацию и запуск мастера.

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

- CAN_RETOK – запуск CAN мастера выполнен успешно.
- CAN_ERRET_CI_INIT – ошибка инициализации CAN драйвера.
- CAN_ERRET_CI_OPEN – не удалось открыть канал CAN контроллера.
- CAN_ERRET_CI_CLOSE – не удалось закрыть канал CAN контроллера.
- CAN_ERRET_CI_START – не удалось перевести CAN контроллер в рабочее состояние.
- CAN_ERRET_CI_STOP – не удалось перевести CAN контроллер в состояние останова.
- CAN_ERRET_CI_HANDLER – не удалось назначить обработчик сигналов CAN драйвера.
- CAN_ERRET_CI_BITRATE – ошибка установа битовой скорости CAN контроллера.

int16 stop_can_master(void);

Осуществляет останов мастера.

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

- CAN_RETOK – останов CAN мастера выполнен успешно.
- CAN_ERRET_CI_STOP – не удалось перевести CAN контроллер в состояние останова.
- CAN_ERRET_CI_CLOSE – не удалось закрыть канал CAN контроллера.

void canopen_monitor(void);

CANopen монитор мастера. Вызывается из главного цикла программы.

4.4 Модуль master_canid.c

Осуществляет проверку CAN идентификаторов ограниченного использования.

unsigned8 check_sdo_canid(cansubind subind, canlink canid);

Контролирует допустимость значений для идентификатора канального уровня SDO протокола.

Параметры:

- **canid** – проверяемый идентификатор SDO протокола (CAN-ID).

Возвращаемые значения:

- RESTRICTED – не допустимое значение идентификатора.
- UN_RESTRICTED – значение идентификатора допустимо.

unsigned8 check_canid_restricted(canlink canid);

Определяет принадлежность CAN идентификатора канального уровня к идентификаторам ограниченного использования.

Параметры:

- **canid** – проверяемый CAN идентификатор канального уровня (CAN-ID).

Возвращаемые значения:

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

4.5 Модуль master_client.c

Реализует функции SDO протокола клиента.

void can_sdo_client_transfer(struct sdocltappl *ca);

Выполняет полную транзакцию передачи данных между клиентом и сервером с использованием SDO протокола. Поддерживает возможность передачи данных переменного размера. Режимы проведения транзакции, ее условия и результаты содержатся в структуре ***ca**.

Параметры:

- **ca.operation** – определяет базовый режим передачи SDO. Задается пользователем и модифицируется функцией.
  CAN_SDOPER_DOWNLOAD – передача данных от клиента серверу (download),
  CAN_SDOPER_UPLOAD – передача данных от сервера клиенту (upload).
  Если размер данных не превышает 4 байта, используется ускоренный (expedited) режим передачи. При размере данных более 4 байт, применяется сегментированный (segmented) SDO протокол. После выполнения функции параметр **ca.operation** содержит код режима, фактически использованного при SDO обмене:
  CAN_SDOPER_(UP/DOWN)_EXPEDITED – ускоренный режим или
  CAN_SDOPER_(UP/DOWN)_SEGMENTED – сегментированный режим.
- **ca.datasize** – размер в байтах прикладных данных, передаваемых посредством SDO. Задается пользователем и модифицируется функцией для upload протокола. При передаче данных серверу (download) определяет фактический размер прикладных данных. В случае передачи данных от сервера клиенту (upload) задает максимально возможный размер данных. После выполнения upload операции содержит фактическое число принятых клиентом байт данных. Нулевое значение данного параметра не допустимо.
- **ca.datapnt** – указатель на локальный буфер прикладных данных. Задается пользователем. Значение NULL для данного параметра не допустимо.
- **ca.si (sdoixs)** – определяет индекс и суб-индекс прикладного CANopen объекта для SDO протокола (мультиплексор SDO протокола). Задается пользователем.
- **ca.ss (sdostatus)** – структура статуса транзакции. Устанавливается функцией и содержит код завершения (статус) SDO транзакции клиента.
  CAN_TRANSTATE_OK – успешное завершение SDO транзакции.
  CAN_TRANSTATE_SDO_TOGGLE – ошибка мерцающего бита (toggle) в протоколе сегментированной передачи;
  CAN_TRANSTATE_SDO_DATASIZE – неверное значение размера данных;
  CAN_TRANSTATE_SDO_MPX – несоответствие мультиплексоров клиента и сервера;
  CAN_TRANSTATE_SDO_SRVABORT – от сервера получен аборт SDO протокола. При этим поле **ca.ss.abortcode** содержит значение аборт кода.
  CAN_TRANSTATE_SDO_WRITERR – ошибка отправки SDO кадра;
  CAN_TRANSTATE_SDO_SCSERR – SDO клиент получил от сервера неверную или неизвестную команду;
  CAN_TRANSTATE_SDO_TRANS_TIMEOUT – внутренний таймаут базовой транзакции SDO клиента;
  CAN_TRANSTATE_SDO_NET_TIMEOUT – сетевой таймаут базовой транзакции SDO клиента;
  CAN_TRANSTATE_SDO_READ_TIMEOUT – таймаут чтения данных приложением;
  CAN_TRANSTATE_SDO_NOWORKB – переполнение рабочего буфера базовых транзакций SDO клиента;
  CAN_TRANSTATE_ERROR – другая общая ошибка.

4.6 Модуль master_cltrans.c

Обеспечивает базовый обмен данными SDO протокола: запрос клиента, прием и обработка ответа сервера.

void can_client_sdo(canframe *cf);

Производит обработку принятого из CAN сети SDO ответа сервера.

Параметры:

- ***cf** – принятый кадр SDO протокола (ответ сервера на запрос клиента).

void can_client_basic(struct sdoclttrans *ct);

Формирует и проводит базовую SDO транзакцию клиента (запрос клиента, прием и обработка ответа сервера).

Параметры:

- ***ct** – структура поддержки базовой SDO транзакции клиента.

void can_client_control(void);

Контролирует таймаут базовой SDO транзакции клиента (запрос клиента и ответ сервера). Вызывается из CANopen таймера.

void set_sdo_timeout(unsigned32 microseconds);

Устанавливает таймаут получения данных из CAN сети для базовой SDO транзакции клиента. Значение по умолчанию задается параметром CAN_TIMEOUT_RETRIEVE.

Параметры:

- **microseconds** – таймаут приема SDO данных в микросекундах.

unsigned32 get_sdo_timeout(void);

Возвращает значение таймаута получения SDO данных из CAN сети.

Возвращаемое значение:

- таймаут приема SDO данных в микросекундах.

void can_init_client(void);

Инициализирует данные модуля.

4.7 Модуль master_inout.c

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 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.

int16 send_can_data(canframe *cf, unsigned16 priority);

Размещает в CANopen кэше кадр канального уровня. Осуществляет пересылку CAN драйверу всех накопленных в кэше кадров. Функция является сигналобезопасной.

Параметры:

- ***cf** – кадр, предназначенный для пересылки CAN драйверу.
- **priority** – программный приоритет CAN кадра.

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

- CAN_RETOK – данные размещены в CANopen кэше. Для вызывающих функций означает успешное завершение отправки кадра в CAN сеть.
- CAN_ERRET_COMM_SEND – коммуникационная ошибка: не удалось отправить CAN кадр из-за переполнения CANopen кэша.

void can_read_handler(canev ev);

Обработчик сигналов приема кадров канального уровня из CAN сети. Функция является сигналобезопасной и обеспечивает чтение кадров, поступивших в буфер драйвера как до выдачи сигнала, так и принятых в процессе его обработки.

void can_init_io(void);

Инициализирует семафоры и другие данные модуля.

4.8 Модуль master_lib.c

Функции общего применения.

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 (1..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.

void clear_can_data(canbyte *data);

Очистка поля данных CAN кадра (8 байт).

Параметры:

- **data** – поле данных CAN кадра.

void u16_to_canframe(unsigned16 ud, canbyte *data);

Преобразование данных типа unsigned16 в байтовый (сетевой) формат.

Параметры:

- **ud** – преобразуемое данное.
- ***data** – байтовый указатель на преобразованные данные.

unsigned16 canframe_to_u16(canbyte *data);

Преобразование данных из байтового (сетевого) формата в тип unsigned16.

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.

Параметры:

- **ud** – преобразуемое данное.

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

unsigned32 canframe_to_u32(canbyte *data);

Преобразование данных из байтового (сетевого) формата в тип unsigned32.

Параметры:

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

Возвращаемое значение:

- данные типа unsigned32.

4.9 Модуль master_obdsdo_client.c

Формирует и поддерживает адаптированный объектный словарь SDO параметров клиента. Этот словарь используется приложением для динамического формирования CAN идентификаторов SDO протокола различных узлов сети.

int16 find_sdo_client_recv_canid(canlink *canid);

Выдает идентификатор принимаемого (от сервера клиенту) CAN кадра SDO протокола.

Параметры:

- ***canid** – CAN идентификатор канального уровня SDO протокола. Его значение определено только если SDO действителен.

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

- CAN_RETOK – SDO действителен.
- CAN_ERRET_SDO_INVALID – SDO не действителен.

int16 find_sdo_client_send_canid(canlink *canid);

Выдает идентификатор передаваемого (от клиента серверу) CAN кадра SDO протокола.

Параметры:

- ***canid** – CAN идентификатор канального уровня SDO протокола. Его значение определено только если SDO действителен.

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

- CAN_RETOK – SDO действителен.
- CAN_ERRET_SDO_INVALID – SDO не действителен.

int16 read_sdo_client_data(cansubind subind, unsigned32 *data);

Осуществляет чтение из объектного словаря идентификаторов коммуникационных SDO объектов (SDO COB-IDs).

Параметры:

- **subind** – субиндекс SDO объекта.
- ***data** – идентификатор коммуникационного SDO объекта.

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

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

int16 write_sdo_client_data(cansubind subind, unsigned32 data);

Осуществляет запись в объектный словарь идентификаторов коммуникационных SDO объектов (SDO COB-IDs).

Параметры:

- **subind** – субиндекс SDO объекта.
- **data** – идентификатор коммуникационного SDO объекта.

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

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOSUBIND – несуществующий субиндекс объекта.

- CAN_ERRET_OBD_READONLY – попытка записи объекта с доступом только по чтению (субиндекс 0).
- CAN_ERRET_OBD_VALRANGE – ошибка диапазона записываемого значения.
- CAN_ERRET_OBD_OBJACCESS – в текущем состоянии объект не может быть изменен.

void can_init_sdo_client(void);

Инициализирует данные модуля.

4.10 Модуль master_obj_sync.c

Формирует и поддерживает объект синхронизации SYNC.

int16 find_sync_recv_canid(canlink *canid);

Выдает идентификатор CAN кадра SYNC протокола.

Параметры:

- ***canid** – CAN идентификатор канального уровня SYNC протокола (CAN-ID).

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

- CAN_RETOK – нормальное завершение.

int16 read_sync_object(canindex index, unsigned32 *data);

Чтение объекта протокола синхронизации мастера.

Параметры:

- **index** – индекс объекта синхронизации.
- ***data** – значение объекта.

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

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – не существует SYNC объекта с индексом **index**.

int16 write_sync_object(canindex index, unsigned32 data);

Запись объекта протокола синхронизации мастера.

Поддерживает коммуникационные объекты, заданные индексами:

1005_h – COB-ID объекта синхронизации SYNC.

1006_h – период объекта синхронизации.

1007_h – длительность окна синхронизации.

1019_h – значение переполнения для SYNC счетчика.

Параметры:

- **index** – индекс объекта синхронизации.
- **data** – значение записываемого объекта.

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

- CAN_RETOK – нормальное завершение.
- CAN_ERRET_OBD_NOOBJECT – не существует SYNC объекта с индексом **index**.
- CAN_ERRET_OBD_VALRANGE – ошибка диапазона записываемого значения.
- CAN_ERRET_OBD_OBJACCESS – в текущем состоянии объект не может быть изменен.
- CAN_ERRET_OBD_DEVSTATE – состояние других объектов не позволяет изменить значение данного объекта.

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 CAN сети объекта синхронизации SYNC. Если

устройство является генератором SYNC, функция автоматически вызывается при каждой передаче SYNC кадра.
Параметры:
- ***cf** – принятый или переданный CAN кадр, содержащий объект синхронизации SYNC.

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

void can_init_sync(void);
Инициализирует данные модуля.

4.11 Модуль master_sdo_proc.c

Осуществляет прием и 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 CAN кадра SDO протокола.

int16 send_can_sdo(struct cansdo *sd);
Осуществляет сборку и отсылку 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 успешно отправлен CAN драйверу.
- CAN_ERRET_SDO_INVALID – SDO не действителен.
- CAN_ERRET_COMM_SEND – коммуникационная ошибка: не удалось отправить CAN кадр из-за переполнения CANopen кэша.

void abort_can_sdo(struct sdoixs *si, unsigned32 abortcode);
Производит отправку кадра Abort SDO Transfer протокола.
Параметры:
- ***si** – индекс словаря прикладного объекта (мультиплексор SDO протокола).
- **abortcode** – значение Abort кода.

5. Модули мастер-приложения.

Модули мастер-приложения размещаются в директории src\application.

5.1 Модуль master_application.c

Модуль поддерживает инфраструктурные компоненты CANopen мастера.

void reset_can_node(cannode node);

Осуществляет пере-инициализацию CANopen узла NMT командой «Reset Node».

Параметры:

- **node** – номер CAN узла (1..127). CAN узел должен быть активным (определен в конфигурационном файле).

void application_timer_routine(void);

Таймер CANopen мастера.

void application_monitor_routine(void);

Монитор (главный цикл) CANopen мастера.

void start_can_network(void);

Осуществляет пере-инициализацию всех 127 возможных узлов CANopen сети NMT командой «Reset Node». Таким образом выявляются все присутствующие в сети узлы, в том числе не активные. Выполняется однократно при запуске мастер приложения.

void init_defaults(void);

Инициализирует значения по умолчанию для параметров CANopen мастера.

5.2 Модуль master_can_nodes.c

Модуль выполняет конфигурирование узлов CANopen сети.

void configure_can_nodes(void);

Инициирует конфигурирование всех активных CANopen узлов, от которых получено сообщение загрузки (Boot-up).

5.3 Модуль master_events.c

Содержит обработчики CAN и CANopen событий мастера.

void consume_sync(unsigned8 sc);

Вызывается при получении объекта синхронизации после проверки состоятельности SYNC кадра.

Параметры:

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

void no_sync_event(void);

Вызывается в случае, если потребитель SYNC не получил объекта синхронизации в течение промежутка времени, определяемого объектом 1006_h (период объекта синхронизации).

void consume_controller_error(canev ev);

Обрабатывает сигнал ошибок от CAN контроллера. Коды ошибок определены в заголовочном файле CAN драйвера канального уровня.

Параметры:

- **ev** (тип int16) – код ошибки:
  CIEV_WTOUT – write timeout occurred,
  CIEV_EWL – error warning limit,
  CIEV_BOFF – bus off,
  CIEV_HOVR – hardware overrun,
  CIEV_SOVR – software overrun.

void master_emcy(unsigned16 errorcode);

Вызывается при возникновении в мастере срочного сообщения Emergency. EMCY объект только регистрируется, но не передается в CAN сеть.

Параметры:

- **errorcode** – код ошибки.

void consume_emcy(canframe *cf);

Обрабатывает принятые из CAN сети объекты срочного сообщения Emergency.

Параметры:

- ***cf** – CAN кадр объекта срочного сообщения EMCY.

void can_cache_overflow(void);

Вызывается при переполнении выходного CANopen кэша.

5.4 Модуль master_globals.c

В модуле определены внешние (глобальные) переменные и структуры данных CANopen мастера.

5.5 Модуль master_nmt_master.c

Модуль поддерживает сетевой менеджер (NMT протоколы).

void nmt_master_command(unsigned8 cs, cannode node);

Формирует и отсылает в сеть кадр NMT протокола. Функция не осуществляет проверку значения NMT команды и номера CAN узла.

Параметры:

- **cs** – NMT команда.
- **node** – номер CAN узла.

void consume_nmt(canframe *cf);

Производит обработку принятого из CAN сети кадра NMT протокола.

Параметры:

- ***cf** – принятый NMT кадр (протоколы загрузки boot-up или сердцебиения heartbeat).

void manage_master_ecp(void);

Осуществляет контроль за прохождением сердцебиения для всех активных узлов сети. Вызывается из CANopen таймера.

5.6 Модуль master_pdo_process.c

Модуль содержит прикладные функции для обработки PDO.

int16 transmit_can_pdo(cannode node, unsigned8 pn);

Формирует и отсылает PDO (RPDO для CANopen узла).

Параметры:

- **node** – номер CAN узла.
- **pn** – номер RPDO (1..4).

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

- CAN_RETOK – PDO успешно отправлен CAN драйверу.
- CAN_ERRET_OBD_NOOBJECT – неверный номер PDO.
- CAN_ERRET_COMM_SEND – коммуникационная ошибка: не удалось отправить CAN кадр из-за переполнения CANopen кэша.

void receive_can_pdo(canframe *cf);

Принимает и обрабатывает PDO (TPDO для CANopen узла).

Параметры:

- ***cf** – принятый CAN кадр, содержащий PDO.

void process_sync_pdo(unsigned8 sc);

Обрабатывает синхронные PDO (заготовка). Вызывается из обработчика объекта синхронизации sync_received(...).

Параметры:

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

void can_init_pdo(void);

Инициализация данных модуля (заготовка).

5.7 Модуль master_sdo_transfer.c

Модуль содержит прикладные функции SDO клиента.

int16 read_device_object(cannode node, canindex index, cansubind subind, canbyte *data, unsigned32 datasize);

Чтение данных прикладного объекта из CAN узла с использованием SDO upload протокола.

Параметры:

- **node** – номер CAN узла.
- **index** – индекс прикладного объекта.
- **subind** – субиндекс прикладного объекта.
- ***data** – байтовый указатель на размещаемые данные.
- **datasize** – размер прикладных данных в байтах.

Возвращаемые значения: статус SDO транзакции.

- см. функцию can_sdo_client_transfer(...) модуля master_client.c

int16 write_device_object(cannode node, canindex index, cansubind subind, canbyte *data, unsigned32 datasize);

Запись данных прикладного объекта в CAN узел с использованием SDO download протокола.

Параметры:

- **node** – номер CAN узла.
- **index** – индекс прикладного объекта.

- **subind** – субиндекс прикладного объекта.
- ***data** – байтовый указатель на размещаемые данные.
- **datasize** – размер прикладных данных в байтах.

Возвращаемые значения: статус SDO транзакции.

- см. функцию can_sdo_client_transfer(...) модуля master_client.c

6. Модули для работы с конфигурационным файлом.

6.1 Конфигурационный файл CANopen мастера.

Конфигурационные параметры CANopen мастера загружаются из файла CANopenMaster.cfg, который должен размещаться в одной директории с исполняемой программой мастера CANopen_winMaster.exe.

Все символы в названиях разделов и записей преобразуются к заглавным.

Файл CANopenMaster.cfg имеет следующую структуру:

[PCFG 00030001]

Версия конфигурационного файла. Записывается в первой не пустой строке файла.

Раздел **[Comments]**

Содержит произвольный комментарий, который при необходимости может обрабатываться приложением.

Комментарий, игнорируемый синтаксическим разборщиком CANopen мастера, может быть оформлен тремя способами:

- Начинаться с символа #. При этом игнорируется любой текст от символа # до конца текущей строки.
- Начинаться с символов //. При этом игнорируется любой текст от символов // до конца текущей строки.
- Начинаться с символов /* и заканчиваться */. Любой текст, включающий произвольное число строк и расположенный между этими символами игнорируется.

Раздел **[FileNames]**

Список имен файлов CANopen мастера.

Если первым символом имени файла является ‘\’, то имя интерпретируется как абсолютное, т.е. включающее в себя полный путь доступа к файлу. В противном случае имя файла определяется относительно директории размещения исполняемой программы CANopen мастера.

Содержит записи:

- **Logfile** – имя файла журнала для записи сообщений регистратора CANopen мастера.

Раздел **[CANnetwork]**

Параметры CAN/CANopen сети.

Файл конфигурации может содержать единственный раздел [CANnetwork]. Осуществляется контроль дублирования параметров CAN сети.

Содержит записи:

- **Network** – номер CAN сети от 0 до 3 (канал CAN контроллера). Значение по умолчанию задается параметром CAN_NETWORK_CONTROLLER модуля __application_defines.h.
- **BitrateIndex** – индекс битовой скорости CAN сети. Значение по умолчанию задается параметром CAN_BITRATE_INDEX модуля __application_defines.h.

Разделы **[CANopenNode]**

Параметры CANopen устройств (узлов CAN сети). Обязательным параметром является только номер CAN узла. При отсутствии остальных параметров подходящим считается устройство с любыми их значениями.

Файл конфигурации может содержать необходимое число разделов [CANopenNode]. Осуществляется контроль дублирования параметров CANopen узлов.

Содержит записи:

- **NodeId** – номер узла CANopen устройства в диапазоне 1..127. Должен быть определен до любых других параметров CANopen устройства.

- **DeviceType** – тип CANopen устройства (объект 1000_h).
- **VendorID** – код, присвоенный производителю устройства (объект $1018_h sub1_h$).
- **ProductCode** – код изделия, задаваемый производителем (объект $1018_h sub2_h$).
- **Revision** – версия устройства, задаваемая производителем (объект $1018_h sub3_h$).
- **Serial** – серийный номер устройства, задаваемый производителем (объект $1018_h sub4_h$).

6.2 Модуль master_config_file.c

Синтаксический разборщик конфигурационного файла.

void read_config(void);

Загружает конфигурационный файл. Осуществляет первичную обработку прочитанных строк (исключение комментариев, удаление «непечатных» символов и лидирующих пробелов). Выделяет разделы конфигурационного файла и выполняет функции диспетчера при их обработке.

6.3 Модуль master_filename.c

Выполняет преобразование имен файлов.

void transform_file_name(char *fname, char *initfn);

Преобразует имя файла таким образом, чтобы учесть размещение исполняемой программы. Для этого у операционной системы запрашивается информация о командной строке, с использованием которой была запущена программа. В итоге обеспечивается возможность формирования имен файлов с учетом их размещения относительно исполняемой программы.

Параметры:

- ***fname** – строка имени преобразованного файла.
- ***initfn** – строка имени исходного файла.

void time_stamp_logfile(char *fname, char *initfn);

Преобразует имя файла, добавляя к нему временную метку вида _yyyymmdd_hhmm. Метка вставляется непосредственно перед последним расширением имени файла.

Параметры:

- ***fname** – строка имени преобразованного файла.
- ***initfn** – строка имени исходного файла.

6.4 Модуль master_pac.c

Функции модуля осуществляют упаковку числовых данных различных форматов, а также обработку символьных строк.

int16 find_token(char *dtok, char *data, unsigned16 *npos);

Осуществляет поиск последовательности заглавных символов (токена) в строке.

Параметры:

- ***dtok** – строка заданного токена (только заглавные символы).
- ***data** – строка для поиска (преобразуется к заглавным символам).
- ***npos** – счетчик обработанных символов (позиция очередного символа строки).

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

- GEN_RETOK – заданный токен обнаружен.
- GEN_ERRET_TOKEN – токен не найден.

int16 parse_float(union numbers *num, char *data, unsigned16 dtype, unsigned16 *npos);

Осуществляет упаковку числа с плавающей точкой.

Параметры:

- ***num** – упакованное число с плавающей точкой (num.re32 или num.re64).
- ***data** – строка исходных символов для упаковки.
- **dtype** – тип результата. CAN_DEFTYPE_REAL32 или CAN_DEFTYPE_REAL64.
- ***npos** – счетчик обработанных символов (позиция очередного символа строки).

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

- GEN_RETOK – данные успешно упакованы.
- GEN_ERRET_EMPTY – строка не содержит записи числа (пустая строка).
- GEN_ERRET_DATATYPE – неверный тип результата.

int16 parse_integer(union numbers *num, char *data, unsigned16 dtype, int16 base, unsigned16 *npos);

Осуществляет упаковку целого числа.

Параметры:

- ***num** – упакованное целое число.
- ***data** – строка исходных символов для упаковки.
- **dtype** – тип целочисленного результата.
- **base** – основание системы счисления 0, 2..36:
  0 – основание определяется функцией по форме записи числа.
- ***npos** – счетчик обработанных символов (позиция очередного символа строки).

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

- GEN_RETOK – данные успешно упакованы.
- GEN_ERRET – неверное основание системы счисления.
- GEN_ERRET_EMPTY – строка не содержит записи числа (пустая строка).
- GEN_ERRET_VALUE – ошибка значения числа.
- GEN_ERRET_DATATYPE – неверный тип результата.

int16 parse_number(union numbers *num, char *data, unsigned16 dtype, unsigned16 *npos);

Осуществляет упаковку чисел различных типов, включая булевые.

Параметры:

- ***num** – упакованное число.
- ***data** – строка исходных символов для упаковки.
- **dtype** – тип результата.
- ***npos** – счетчик обработанных символов (позиция очередного символа строки).

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

- GEN_RETOK – данные успешно упакованы.
- GEN_ERRET – неверное основание системы счисления.
- GEN_ERRET_EMPTY – строка не содержит записи числа (пустая строка).
- GEN_ERRET_VALUE – ошибка значения числа.
- GEN_ERRET_DATATYPE – неверный тип результата.

int16 parse_string(char *dest, char *src, int16 strlen, unsigned16 *npos);

Производит обработку строки. Заменяет «непечатные» символы на пробелы, удаляет лидирующие пробелы. Дополняет строку нулями и всегда устанавливает завершающий ноль. Исходная строка остается неизменной.

Параметры:

- ***dest** – обработанная ASCIIZ строка.
- ***src** – исходная строка.
- **strlen** – длина строк.
- ***npos** – счетчик обработанных символов (позиция очередного символа исходной строки).

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

- GEN_RETOK – строка успешно обработана.
- GEN_ERRET_EMPTY – исходная строка не содержит «видимых» символов (пустая строка).

void clip_string(char *src, int16 strlen);

Заменяет «непечатные» символы и пробелы в конце строки на нули.

Параметры:

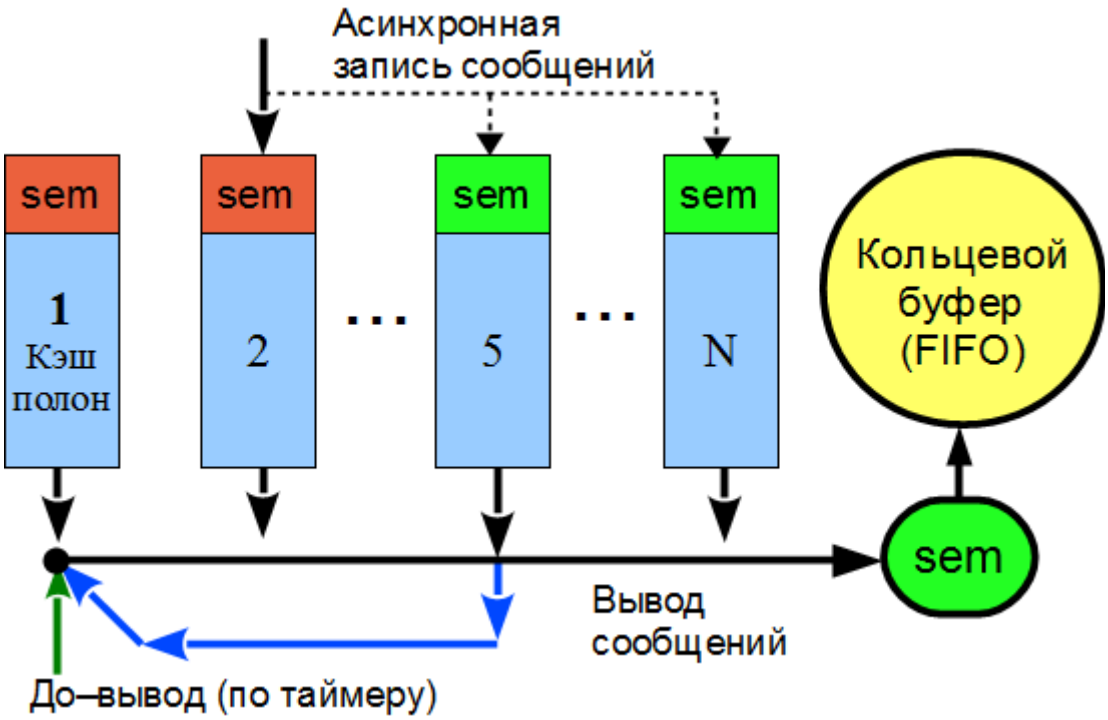
- ***src** – обрабатываемая строка.
- **strlen** – длина строки.

7. Модули асинхронного регистратора.

Модули регистратора размещаются в директории src\logger.

7.1 Структурная схема регистратора.

Асинхронный регистратор осуществляет двухуровневую буферизацию сообщений. Первый уровень обеспечивает асинхронную запись сообщений в кэш. Второй переносит данные из кэша в кольцевой буфер (FIFO). Затем зарегистрированные события в описательном виде заносятся в файлы журнала.



На рисунке приведена схема подсистемы регистратора, которая обеспечивает асинхронную запись сообщений в кэш. Доступ к каждому буферу кэша защищен отдельным семафором. Первый буфер является выделенным и не используется для записи внешних событий. В него заносится на постоянное хранение сообщение о переполнении кэша. При возникновении такого события семафор первого буфера открывается, регистрируя факт переполнения. Использование кэша в редких случаях может привести к тому, что сообщения будут записаны в FIFO, а затем и в файл журнала в последовательности, отличной от очередности их занесения в кэш.

Непосредственно после записи сообщения в кэш предпринимается попытка пересылки всех накопленных данных в кольцевой буфер. Поскольку операция заполнения FIFO инициируется асинхронно, она защищена семафором, а значит запись данных в кольцевой буфер может оказаться безуспешной. В связи с этим осуществляется периодическое (по таймеру) сканирование кэша и до-вывод оставшихся в нем сообщений. В противном случае данные могли бы оставаться в кэше неопределенное время - как минимум до записи очередного сообщения.

Кольцевой буфер снабжен механизмом подчистки при переполнении. Когда голова буфера упирается в его хвост, последний принудительно смещается, освобождая тем самым некоторое число элементов FIFO. При этом самые старые сообщения будут утеряны, но

благодаря наличию кэша обеспечивается регистрация факта переполнения кольцевого буфера.

Подсистема, которая осуществляет извлечение и переправку накопленной в FIFO информации в файлы журнала, включена в главный цикл программы. Это гарантирует корректное проведение всех необходимых файловых операций.

7.2 Модуль master_logfile.c

void write_event_to_file(struct eventlog *ev, FILE *log);

Формирует в описательном виде сообщение о событии и заносит его в файл журнала.

Параметры:

- ***ev** – зарегистрированное событие.
- ***log** – активный файл журнала.

7.3 Модуль master_logger.c

void flush_events_cache(void);

Пересылает в FIFO регистратора накопленные в кэше события. Для гарантированного вывода всех данных из кэша может также вызываться из таймера приложения. Функция является сигналобезопасной.

void log_event(struct eventlog *ev);

Размещает событие в кэше регистратора и осуществляет попытку его вывода в FIFO. Функция является сигналобезопасной.

Параметры:

- ***ev** – зарегистрированное событие.

void push_events_logger(void);

Осуществляет извлечение и переправку накопленной в FIFO информации в файлы журнала. Вызывается из главного цикла программы.

void master_event(unsigned8 class, unsigned8 type, int16 code, int32 info);

Функция прикладного интерфейса для записи событий мастера.

Параметры:

- **class** – класс зарегистрированного события.
- **type** – тип события (info, warning, error и т.д.).
- **code** – код события.
- **info** – дополнительная информация о событии.

void node_event(cannode node, unsigned8 class, unsigned8 type, int16 code, int32 info);

Функция прикладного интерфейса для записи событий CANopen узла.

Параметры:

- **node** – номер CAN узла, в котором было порождено событие.
- **class** – класс зарегистрированного события.
- **type** – тип события (info, warning, error и т.д.).
- **code** – код события.
- **info** – дополнительная информация о событии.

void close_logger(void);

Закрывает регистратор при выходе из программы.

void init_logger(void);

Инициализирует данные регистратора.