



**Руководство (v7.2)**

**По работе с драйвером модулей  
“mPCIe – CAN”, “PCIe – CAN”**

Интерфейс ISO-11898  
(CAN Bus)

Для драйверов версии 7.1

**RTOS QNX6.5  
(ЗОСРВ «Нейтрино» КПДА.10964-01)**

**10.02.2022**

**ООО “Новомар”**

## Оглавление

1. Введение.....	3
2. Сборка и установка драйвера .....	3
3. Подключение файла с командами к разрабатываемому проекту .....	3
4. Список доступных команд.....	4
5. Описание использования команд драйвера .....	6
6. Стандартные функции.....	7
6.1 IOCTL_WR_MAINREG_CAN .....	7
6.2 IOCTL_RD_MAINREG_CAN .....	8
6.3 IOCTL_WR_CANREG_CAN .....	9
6.4 IOCTL_RD_CANREG_CAN .....	10
6.5 IOCTL_MODIFY_CANREG_CAN .....	11
6.6 IOCTL_WR_CANREGS_CAN .....	12
6.7 IOCTL_RD_CANREGS_CAN .....	13
6.8 IOCTL_VERSION_CAN .....	14
6.9 IOCTL_VERSION_DRIVER_CAN .....	15
7. Функции конфигурации.....	16
7.1 IOCTL_ENABLE_DMA_CAN .....	16
7.2 IOCTL_DISABLE_DMA_CAN .....	17
7.3 IOCTL_SET_MODE_CAN .....	18
7.4 IOCTL_GET_MODE_CAN .....	19
7.5 IOCTL_SET_ONESHOT_MODE_CAN.....	20
7.6 IOCTL_SET_SPEED_CAN .....	21
7.7 IOCTL_SET_SPEED_PARAMS_CAN .....	22
7.8 IOCTL_GET_ERRORS_CAN .....	23
7.9 IOCTL_SET_MASKS_CAN .....	24
7.10 IOCTL_RESET_CANn_TIMER_CAN .....	26
7.11 IOCTL_GET_CANn_TIMER_CAN.....	27
7.12 IOCTL_SET_CANn_TIMEOUTS_CAN .....	28
7.13 IOCTL_RESET_CAN .....	29
7.14 IOCTL_RESET_CANn_CAN.....	30
8. Функции для чтения принятых данных.....	31
8.1 IOCTL_RD_CH_RAW_DMA_CAN .....	31
8.2 IOCTL_READ_DMA_BLOCKS_CAN .....	33
9. Функции для передачи данных .....	34
9.1 IOCTL_WRITE_DATA_TO_FIFO_CAN1_V2.....	35
9.2 IOCTL_WRITE_DATA_TO_FIFO_CAN2_V2.....	36
9.3 IOCTL_WRITE_DATA_TO_FIFO_CAN1 .....	37
9.4 IOCTL_WRITE_DATA_TO_FIFO_CAN2 .....	38
9.5 IOCTL_WRITE_DATA_TO_TR_BUF_CAN .....	39
9.6 IOCTL_SEND_DATA_CAN.....	40
9.7 IOCTL_SEND_DATA_NOW_CAN .....	42
9.8 IOCTL_CHECK_TRANSMIT_CAN .....	43
9.9 IOCTL_WAIT_TRANSMIT_CAN .....	44
9.10 IOCTL_END_TRANSMIT_CAN.....	45
9.11 IOCTL_ABAT_CAN .....	46
10. Обновление драйвера .....	47
11. Обновление руководства .....	48

## 1. Введение

Драйвер является самостоятельно запускаемым приложением, реализующим технологию «мультипоточного менеджера ресурсов» (символьных устройств) и предназначен для функционирования в RTOS QNX6.5 (ЗОСРВ «Нейтрино» КПДА.10964-01). Представляет собой исполняемый файл с именем «dev-nmcan». Перед использованием драйвер необходимо установить.

Драйвер поддерживает модули “mPCIe-CAN”, “PCIe-CAN” (далее xPCIe-CAN).

В начале работы в процессе обнаружения установленных модулей драйвер присваивает устройствам уникальные символьные имена вида “can\_dev\_x”, где x — индекс устройства, начиная с 0.

Каждый модуль представляет собой отдельный файл устройства в файловой системе и отображается в папке “/dev”.

Взаимодействие с драйвером происходит посредством ioctl-команд, перечень которых находится в файле “nmcan.h”.

Обращение к независимым каналам платы осуществляется по номеру канала (1 и 2).

## 2. Сборка и установка драйвера

Создайте отдельную папку, например: */modules*.

Скачайте в эту папку архив с исходными текстами драйвера и распакуйте его.

Выполните команду “*cd /modules/xPCIe-CAN/src/dev-nmcan*” для перехода в каталог с исходными текстами драйвера.

Для сборки драйвера выполните команду: *make*

Для очистки проекта выполните команду: *make clean*

Для установки драйвера выполните команду: *cp x86/o/dev-nmcan /sbin*

Для удаления драйвера выполните команду: *rm /sbin/dev-nmcan*

Для запуска установленного драйвера выполните команду: *dev-nmcan &*

Для останова работающего драйвера: *slay dev-nmcan*

Для проверки работает ли драйвер в данный момент выполните команду: *pidin a | grep dev-nmcan*

**Внимание!** Для работы с драйвером версии 7.0 и выше в режиме FIFO, со счётчиками статистики и таблицей Timemark необходимо Firmware модуля не ниже «07» от 03.07.2020 (См. поле: «Дата предпродажной проверки» на этикетке).

Для обновления Firmware обратитесь к производителю.

## 3. Подключение файла с командами к разрабатываемому проекту

1. Скопировать файл “nmcan.h” из папки “/modules/xPCIe-CAN/lib” в папку с проектом.

2. В начале проекта добавить строку:

```
#include "nmcan.h"
```

3. Использовать команды. Формат команд описан ниже.

### **ВНИМАНИЕ**

Системная функция **open** открывает модуль “xPCIe-CAN” целиком.

Если вызов драйвера не требует номер канала, значит, драйвер работает с модулем как с единым целым устройством.

Если вызов запрашивает номер канала, значит, драйвер работает только с одним конкретным каналом связи.

#### 4. Список доступных команд

Название вызова	Краткое описание
<a href="#">IOCTL_WR_MAINREG_CAN</a>	Запись регистра модуля.
<a href="#">IOCTL_RD_MAINREG_CAN</a>	Чтение регистра модуля.
<a href="#">IOCTL_WR_CANREG_CAN</a>	Запись регистра контроллера канала.
<a href="#">IOCTL_RD_CANREG_CAN</a>	Чтение регистра контроллера канала.
<a href="#">IOCTL_MODIFY_CANREG_CAN</a>	Модификация регистра контроллера канала.
<a href="#">IOCTL_WR_CANREGS_CAN</a>	Запись регистров контроллера канала.
<a href="#">IOCTL_RD_CANREGS_CAN</a>	Чтение регистров контроллера канала.
<a href="#">IOCTL_VERSION_CAN</a>	Информация о плате.
<a href="#">IOCTL_VERSION_DRIVER_CAN</a>	Информация о драйвере.
<a href="#">IOCTL_ENABLE_DMA_CAN</a>	Разрешение работы DMA.
<a href="#">IOCTL_DISABLE_DMA_CAN</a>	Запрет работы DMA.
<a href="#">IOCTL_SET_MODE_CAN</a>	Установка режима работы канала.
<a href="#">IOCTL_GET_MODE_CAN</a>	Чтение режима работы канала.
<a href="#">IOCTL_SET_ONESHOT_MODE_CAN</a>	Установка/снятие режима однократной попытки отправки сообщения для канала.
<a href="#">IOCTL_SET_SPEED_CAN</a>	Установка скорости работы канала.
<a href="#">IOCTL_SET_SPEED_PARAMS_CAN</a>	Установка специальной скорости работы канала.
<a href="#">IOCTL_GET_ERRORS_CAN</a>	Чтение регистров ошибок канала.
<a href="#">IOCTL_SET_MASKS_CAN</a>	Установка масок и фильтров на прием сообщений канала.
<a href="#">IOCTL_RESET_CANn_TIMER_CAN</a>	Сброс таймера канала.
<a href="#">IOCTL_GET_CANn_TIMER_CAN</a>	Чтение текущего значения таймера канала.
<a href="#">IOCTL_SET_CANn_TIMEOUTS_CAN</a>	Установка абсолютного и интервального таймера прерываний канала.
<a href="#">IOCTL_RESET_CAN</a>	Сброс модуля.
<a href="#">IOCTL_RESET_CANn_CAN</a>	Сброс канала.
<a href="#">IOCTL_RD_CH_RAW_DMA_CAN</a>	Чтение данных из DMA канала.
<a href="#">IOCTL_WRITE_DATA_TO_TR_BUF_CAN</a>	Запись данных в буфер отправки канала.
<a href="#">IOCTL_SEND_DATA_CAN</a>	Запись данных в буфер отправки канала, запуск передачи сообщения и ожидание завершения передачи.
<a href="#">IOCTL_SEND_DATA_NOW_CAN</a>	Запуск передачи сообщения из буфера отправки канала.

<a href="#"><u>IOCTL_CHECK_TRANSMIT_CAN</u></a>	Проверка завершения и правильности передачи сообщения из буфера отправки канала.
<a href="#"><u>IOCTL_WAIT_TRANSMIT_CAN</u></a>	Ожидание завершения и проверка правильности передачи сообщения из буфера отправки канала.
<a href="#"><u>IOCTL_END_TRANSMIT_CAN</u></a>	Снятие запроса на передачу сообщения из буфера отправки канала.
<a href="#"><u>IOCTL_ABAT_CAN</u></a>	Установка/снятие режима остановки всех активных передач.
<a href="#"><u>IOCTL_READ_DMA_BLOCKS_CAN</u></a>	Чтение данных из буфера DMA
<a href="#"><u>IOCTL_WRITE_DATA_TO_FIFO_CAN1</u></a>	Отправка данных в режиме FIFO для канала 1
<a href="#"><u>IOCTL_WRITE_DATA_TO_FIFO_CAN2</u></a>	Отправка данных в режиме FIFO для канала 2
<a href="#"><u>IOCTL_WRITE_DATA_TO_FIFO_CAN1_V2</u></a>	Отправка данных в режиме FIFO для канала 1 с возможностью установки RTR-бита
<a href="#"><u>IOCTL_WRITE_DATA_TO_FIFO_CAN2_V2</u></a>	Отправка данных в режиме FIFO для канала 2 с возможностью установки RTR-бита

## 5. Описание использования команд драйвера

Структура ioctl-команды:

```
#include <sys/ioctl.h>
```

```
int ioctl(int fd, IOCTL_..., unsigned long param),
```

где:

*int fd* – дескриптор файла, полученный при вызове функции *open* для файла устройства;

*IOCTL\_...* – команда из набора, описанного в файле *ntcan.h*;

*unsigned long param* – параметр, передаваемый с командой. Содержимое параметра зависит от команды (см. описание команд ниже).

Команда, в случае удачного выполнения запроса, возвращает 0.

В случае неудачного выполнения запроса возвращается значение -1. Причину ошибки можно узнать из значения переменной *errno*:

1. EINVAL – ошибки в параметрах запроса;
2. ETIME – таймаут выполнения запроса;
3. EBUSY – запрос ещё не выполнен;
4. EFAULT – ошибка обращения к памяти пользовательского процесса.
5. ENOTTY – неизвестный запрос.
6. EPERM – запрос не поддерживается.

**После загрузки драйвера запрещена работа устройства, работа DMA, указатель DMA сброшен в 0.**

## 6. Стандартные функции

### 6.1 IOCTL\_WR\_MAINREG\_CAN

**Назначение:**

Запись данных в регистровое пространство устройства (BAR) (но не контроллеров!).

**Действие:**

Функция записывает данные по желаемому адресу в регистровое пространство устройства (BAR).

**Примечание:**

Недопустимо обращение к устройству по адресам, не кратным четырём!!!

**Входные параметры:**

param – указатель на структуру типа SADDR\_DATA\_MAIN\_CAN:

Поле структуры	Описание
uint32_t <b>daddr</b>	Адрес регистра
uint32_t <b>data</b>	Данные для записи

**Пример вызова:**

```
int fd, ret;
    SADDR_DATA_MAIN_CAN saddr;
fd = open("/dev/can_dev_0", O_RDWR);
saddr.daddr = 0x2000;
saddr.data = 0;
ret = ioctl(fd, IOCTL_WR_MAINREG_CAN, &saddr);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("Data %08XH was written to BAR register %08XH \n", saddr.data, saddr.daddr);
```

## 6.2 IOCTL\_RD\_MAINREG\_CAN

### Назначение:

Чтение данных из регистрового пространства устройства (BAR) (но не контроллеров!).

### Действие:

Функция читает данные из желаемого адреса регистрового пространства устройства (BAR) в поле **data**.

### Примечание:

Недопустимо обращение к устройству по адресам, не кратным четырём!

### Входные параметры:

param – указатель на структуру типа SADDR\_DATA\_MAIN\_CAN

Поле структуры	Описание
uint32_t <b>daddr</b>	Адрес регистра
uint32_t <b>data</b>	Возвращаемый результат

### Пример вызова:

```
int fd, ret;
    SADDR_DATA_MAIN_CAN saddr;
fd = open("/dev/can_dev_0", O_RDWR);
saddr.daddr = 0x2000;
ret = ioctl(fd, IOCTL_RD_MAINREG_CAN, &saddr);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("Data %08XH was read from BAR register %08XH\n", saddr.data, saddr.daddr);
```



## 6.3 IOCTL\_WR\_CANREG\_CAN

### Назначение:

Запись данных в одиночный регистр контроллера CAN.

### Действие:

Функция записывает данные в **CANn\*\_BUF\*\***. После чего записывает команду “Write” в регистр **CANn\*\_ACS\*\*\*** и дожидается её выполнения путём перевода процесса в состояние ожидания, из которого процесс будет выведен после прихода прерывания, означающего окончание операции или по истечении 10 миллисекунд. В последнем случае функция возвратит значение -1, а в errno будет установлено значение ETIME.

Все функции работы с регистрами контроллера CAN защищены общим семафором (индивидуальным для каждого канала) для возможности их корректного использования из параллельных процессов.

\* n – номер контроллера.

\*\* См. Раздел 6.1.4 или 6.1.6 документа “Руководство по программированию модуля “xPCIE-CAN”.

\*\*\* См. Раздел 6.1.3 или 6.1.5 документа “Руководство по программированию модуля “xPCIE-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа SADDR\_DATA\_CAN:

Поле структуры	Описание
uint8_t <b>daddr</b>	Адрес регистра
uint8_t <b>data</b>	Данные для записи
uint32_t <b>channel</b>	Номер канала (допустимые значения – 1...2)

### Пример вызова:

```
int fd, ret;
SADDR_DATA_CAN saddr;
fd = open("/dev/can_dev_0", O_RDWR);
saddr.daddr = CANINTE;
saddr.data = 0x0;
saddr.channel = 1;
ret = ioctl(fd, IOCTL_WR_CANREG_CAN, &saddr);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("Data %02XH was written to register %02XH of CAN%u\n", saddr.data, saddr.daddr,
        saddr.channel);
```

## 6.4 IOCTL\_RD\_CANREG\_CAN

### Назначение:

Чтение данных из одиночного регистра контроллера CAN.

### Действие:

Функция записывает команду “Read” в регистр **CANn\*\_ACS\*\*** и дожидается её выполнения путём перевода процесса в состояние ожидания, из которого процесс будет выведен после прихода прерывания, означающего окончание операции или по истечении 10 миллисекунд. В последнем случае функция возвратит значение -1, а в `errno` будет установлено значение `ETIME`. В случае получения прерывания о завершении операции вычитанные данные из **CANn\*\_BUF\*\*\*** копируются в поле **data**.

Все функции работы с регистрами контроллера CAN защищены общим семафором (индивидуальным для каждого канала) для возможности их корректного использования из параллельных процессов.

\* n – номер контроллера.

\*\* См. Раздел 6.1.4 или 6.1.6 документа “Руководство по программированию модуля “xPCIe-CAN”.

\*\*\* См. Раздел 6.1.3 или 6.1.5 документа “Руководство по программированию модуля “xPCIe-CAN”.

### Примечание:

-

### Входные параметры:

`param` – указатель на структуру типа `SADDR_DATA_CAN`:

Поле структуры	Описание
<code>uint8_t daddr</code>	Адрес регистра
<code>uint8_t data</code>	Возвращаемый результат
<code>uint32_t channel</code>	Номер канала (допустимые значения – 1...2)

### Пример вызова:

```
int fd, ret;
    SADDR_DATA_CAN saddr;
fd = open("/dev/can_dev_0", O_RDWR);
saddr.daddr = CAN_CTRL;
saddr.channel = 1;
ret = ioctl(fd, IOCTL_RD_CANREG_CAN, &saddr);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("Data %02XH was read from register %02XH of CAN%u \n", saddr.data, saddr.daddr,
        saddr.channel);
```

## 6.5 IOCTL\_MODIFY\_CANREG\_CAN

### Назначение:

Модификация регистра контроллера CAN.

### Действие:

Функция записывает данные в `CANn*_BUF**`. После чего записывает команду “Bit Modify” в регистр `CANn*_ACS***` и дожидается её выполнения путём перевода процесса в состояние ожидания, из которого процесс будет выведен после прихода прерывания, означающего окончание операции или по истечении 10 миллисекунд. В последнем случае функция возвратит значение -1, а в егпо будет установлено значение ETIME.

Все функции работы с регистрами контроллера CAN защищены общим семафором (индивидуальным для каждого канала) для возможности их корректного использования из параллельных процессов.

\* n – номер контроллера.

\*\* См. Раздел 6.1.4 или 6.1.6 документа “Руководство по программированию модуля “xPCIe-CAN””.

\*\*\* См. Раздел 6.1.3 или 6.1.5 документа “Руководство по программированию модуля “xPCIe-CAN””.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа `SADDR_DATA_MODIFY_CAN`:

Поле структуры	Описание
<code>uint8_t daddr</code>	Адрес регистра
<code>uint16_t data</code>	Данные для записи (младшие 8 бит – маска для указания модифицируемых битов регистра, старшие 8 бит – новое значение для модифицируемых битов регистра)
<code>uint32_t channel</code>	Номер канала (допустимые значения – 1...2)

### Пример вызова:

```
int fd, ret;
    SADDR_DATA_MODIFY_CAN saddr;
fd = open("/dev/can_dev_0", O_RDWR);
saddr.daddr = TXB0CTRL;
saddr.data = 0x0808;
saddr.channel = 1;
ret = ioctl(fd, IOCTL_MODIFY_CANREG_CAN, &saddr);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("Data %02H was written to bits %02XH of register %02XH of CAN%u\n", saddr.data >> 8,
        saddr.data & 0xFF, saddr.daddr, saddr.channel);
```

## 6.6 IOCTL\_WR\_CANREGS\_CAN

### Назначение:

Запись блока данных в регистры контроллера CAN.

### Действие:

Функция записывает данные в `CANn*_BUF**`. После чего записывает команду “Write” в регистр `CANn*_ACS***` и дожидается её выполнения путём перевода процесса в состояние ожидания, из которого процесс будет выведен после прихода прерывания, означающего окончание операции или по истечении 10 миллисекунд. В последнем случае функция возвратит значение -1, а в `errno` будет установлено значение `ETIME`.

Все функции работы с регистрами контроллера CAN защищены общим семафором (индивидуальным для каждого канала) для возможности их корректного использования из параллельных процессов.

\* n – номер контроллера.

\*\* См. Раздел 6.1.4 или 6.1.6 документа “Руководство по программированию модуля “xPCIE-CAN””.

\*\*\* См. Раздел 6.1.3 или 6.1.5 документа “Руководство по программированию модуля “xPCIE-CAN””.

### Примечание:

-

### Входные параметры:

`param` – указатель на структуру типа `SADDR_DATAS_CAN`:

Поле структуры	Описание
<code>uint8_t channel</code>	Номер канала (допустимые значения – 1...2)
<code>uint8_t addr</code>	Адрес первого регистра
<code>uint8_t nsize</code>	Количество записываемых данных (допустимые значения – 1...16)
<code>uint8_t data[16]</code>	Записываемые данные

### Пример вызова:

```
int fd, ret;
```

```
    SADDR_DATAS_CAN saddr;
```

```
fd = open("/dev/can_dev_0", O_RDWR);
```

```
saddr.channel = 1;
```

```
saddr.addr = TXB0CTRL + 1;
```

```
saddr.nsize = 6;
```

```
saddr.data[0] = 0x00;
```

```
saddr.data[1] = 0x00; // SID=000, EID absent
```

```
saddr.data[2] = 0x00;
```

```
saddr.data[3] = 0x00;
```

```
saddr.data[4] = 0x01; // DLC=1
```

```
saddr.data[5] = 0xFF; //TXB0D0=FFH
```

```
ret = ioctl(fd, IOCTL_WR_CANREGS_CAN, &saddr);
```

```
if (ret == -1)
```

```
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
```

```
else
```

```
    printf("CAN%u registers was filled with data starting from address %02XH\n", saddr.channel);
```

## 6.7 IOCTL\_RD\_CANREGS\_CAN

### Назначение:

Чтение блока данных из регистров контроллера CAN.

### Действие:

Функция записывает команду “Read” в регистр **CANn\*\_ACS\*\*** и дожидается её выполнения путём перевода процесса в состояние ожидания, из которого процесс будет выведен после прихода прерывания, означающего окончание операции или по истечении 10 миллисекунд. В последнем случае функция возвратит значение -1, а в errno будет установлено значение ETIME. В случае получения прерывания о завершении операции вычитанные данные из **CANn\*\_BUF\*\*\*** копируются в поле **data**.

Все функции работы с регистрами контроллера CAN защищены общим семафором (индивидуальным для каждого канала) для возможности их корректного использования из параллельных процессов.

\* n – номер контроллера.

\*\* См. Раздел 6.1.3 или 6.1.5 документа “Руководство по программированию модуля “xPCIe-CAN””.

\*\*\* См. Раздел 6.1.4 или 6.1.6 документа “Руководство по программированию модуля “xPCIe-CAN””.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа SADDR\_DATAS\_CAN:

Поле структуры	Описание
uint8_t <b>channel</b>	Номер канала (допустимые значения – 1...2)
uint8_t <b>addr</b>	Адрес первого регистра
uint8_t <b>nsize</b>	Количество считываемых данных (допустимые значения – 1...16)
uint8_t <b>data</b> [16]	Возвращаемый результат

### Пример вызова:

```
int fd, ret;
    SADDR_DATAS_CAN saddr;
fd = open("/dev/can_dev_0", O_RDWR);
saddr.channel = 1;
saddr.addr = TXB0CTRL + 1;
saddr.nsize = 6;
ret = ioctl(fd, IOCTL_RD_CANREGS_CAN, &saddr);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("Data %02XH %02XH %02XH %02XH %02XH %02XH was read from registers starting
    from %02XH of CAN%u\n", saddr.data[0], saddr.data[1], saddr.data[2], saddr.data[3],
    saddr.data[4], saddr.data[5], saddr.addr, saddr.channel);
```

## 6.8 IOCTL\_VERSION\_CAN

### Назначение:

Чтение информации о модуле.

### Действие:

Функция заполняет все поля структуры `VERSION_CAN`.

### Примечание:

-

### Входные параметры:

`param` – указатель на структуру типа `VERSION_CAN`:

Поле структуры	Описание
<code>uint32_t device_id</code>	Идентификатор модуля
<code>uint32_t vendor_id</code>	Идентификатор производителя
<code>uint8_t revision</code>	Номер ревизии модуля
<code>char dev_name[30]</code>	Имя модуля, назначенное драйвером
<code>uint32_t minor</code>	Порядковый номер модуля, присвоенный драйвером
<code>uint32_t irq</code>	Номер линии прерываний
<code>uint64_t size_dma</code>	Размер буфера DMA
<code>uint64_t addr_dma_virt</code>	Виртуальный адрес в ядре ОС буфера DMA
<code>uint64_t pciBars</code>	Виртуальный адрес в ядре ОС регистрового пространства (BAR)

### Пример вызова:

```
int fd, ret;
VERSION_CAN ver;
fd = open("/dev/can_dev_0", O_RDWR);
ret = ioctl(fd, IOCTL_VERSION_CAN, &ver);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("Vendor ID = %04XH\nDevice ID = %04XH\nRevision ID = %02XH\nDevice name =
    %s\nMinor = %u\nInterrupt=%u\nDMA bufsize=%u\n", ver.vendor_id, ver.device_id,
    ver.revision, ver.dev_name, ver.minor, ver.irq, ver.size_dma);
```

## 6.9 IOCTL\_VERSION\_DRIVER\_CAN

**Назначение:**

Чтение версии и даты драйвера.

**Действие:**

Функция заполняет 32-битную переменную.

**Примечание:**

-

**Входные параметры:**

param – указатель на 32-битную переменную, в которую будет записано в BCD формате значение версии и даты создания драйвера. В старших 8 битах – старшая и младшая цифры версии драйвера. В младших 24 битах шесть цифр - дата создания драйвера (две цифры день, две цифры месяц и младшие две цифры года).

**Пример вызова:**

```
int fd, ret;
uint32_t drv_ver_date;
fd = open("/dev/can_dev_0", O_RDWR);
ret = ioctl(fd, IOCTL_VERSION_DRIVER_CAN, &drv_ver_date);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("Driver Version: %X.%X, Date: %02X.%02X.%02X\n", drv_ver_date >> 28, (drv_ver_date >> 24) & 0x0F, (drv_ver_date >> 16) & 0xFF, (drv_ver_date >> 8) & 0xFF, drv_ver_date & 0xFF);
```

## 7. Функции конфигурации

### 7.1 IOCTL\_ENABLE\_DMA\_CAN

**Назначение:**

Разрешение работы DMA.

**Входные параметры:**

Нулевой бит регистра **DMA\_DATA\_BASE\*** (адрес 1000h) и биты BnBFM, BnBFE и BnBFS регистра **BFPCTRL\*\*** (адрес 0Ch) устанавливается в единицу.

\* См. Раздел 5.1.1 документа “Руководство по программированию модуля “xPCIE-CAN”.

\*\* См. Раздел 6.4.3 документа “Руководство по программированию модуля “xPCIE-CAN”.

**Примечание:**

После загрузки операционной системы работа DMA не разрешена.

Входные данные

—

**Пример вызова:**

```
int fd, ret;
fd = open("/dev/can_dev_0", O_RDWR);
ret = ioctl(fd, IOCTL_ENABLE_DMA_CAN);
if (ret == -1)
printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
printf("DMA work was enabled\n");
```



## 7.2 IOCTL\_DISABLE\_DMA\_CAN

**Назначение:**

Запрет работы DMA.

**Действие:**

Нулевой бит регистра **DMA\_DATA\_BASE\*** (адрес 1000h) и биты VnBFM, VnBFE и VnBFS регистра **BFCTRL\*\*** (адрес 0Ch) сбрасываются в ноль.

\* См. Раздел 5.1.1 документа “Руководство по программированию модуля “xPCIe-CAN”.

\*\* См. Раздел 6.4.3 документа “Руководство по программированию модуля “xPCIe-CAN”.

**Примечание:**

После загрузки операционной системы работа не разрешена.

Входные данные

–

**Пример вызова:**

```
int fd, ret;
fd = open("/dev/can_dev_0", O_RDWR);
ret = ioctl(fd, IOCTL_DISABLE_DMA_CAN);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("DMA work was disabled\n");
```

### 7.3 IOCTL\_SET\_MODE\_CAN

**Назначение:**

Установка нового режима работы контроллера.

См. Раздел 6.3 документа “Руководство по программированию модуля “xPCIe-CAN”

**Действие:**

Значение из поля **mode** записывается с 5 по 7 биты регистра **CAN\_CTRL\*** (адрес Fh). После этого следует проверить, установился ли данный режим работы. Для этого следует воспользоваться вызовом [IOCTL\\_GET\\_MODE\\_CAN](#).

\* См. Раздел 6.3.1 документа “Руководство по программированию модуля “xPCIe-CAN”.

**Примечание:**

Будьте внимательны при выставлении режима сна.

Внимательно прочитайте какие действия нужно совершить для перевода устройства в режим сна и для вывода устройства из этого режима в разделе 6.3 документа “Руководство по программированию модуля “xPCIe-CAN”.

**Входные параметры:**

param – указатель на структуру типа CONF\_CAN:

Поле структуры	Описание
uint32_t <b>channel</b>	Номер канала (допустимые значения – 1...2)
uint8_t <b>mode</b>	Новый режим работы контроллера

**Пример вызова:**

```
int fd, ret;
    CONF_CAN conf;
fd = open("/dev/can_dev_0", O_RDWR);
conf.channel = 1;
conf.mode = CAN_WORK; // CAN_CONF, CAN_WORK, CAN_MON, CAN_SLEEP, CAN_LOOP
ret = ioctl(fd, IOCTL_SET_MODE_CAN, &conf);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u mode set to %u\n", conf.channel, conf.mode);
```

## 7.4 IOCTL\_GET\_MODE\_CAN

### Назначение:

Чтение текущего режима работы контроллера.

### Действие:

Читает значение битов с 5 по 7 регистра CAN\_STAT\* (адрес Eh) в поле **mode**.

\* См. Раздел 6.3.2 документа “Руководство по программированию модуля “xPCIe-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа CONF\_CAN:

Поле структуры	Описание
uint32_t <b>channel</b>	Номер канала (допустимые значения – 1...2)
uint8_t <b>mode</b>	Текущий режим работы контроллера

### Пример вызова:

```
int fd, ret;
CONF_CAN conf;
fd = open("/dev/can_dev_0", O_RDWR);
conf.channel = 1;
ret = ioctl(fd, IOCTL_GET_MODE_CAN, &conf);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u mode is %u\n", conf.channel, conf.mode);
```

## 7.5 IOCTL\_SET\_ONESHOT\_MODE\_CAN

### Назначение:

Установка или снятие режима однократной попытки передачи сообщения контроллером.

### Действие:

Бит 0 из поля **mode** записывается в бит 3 регистра **CAN\_CTRL\*** (адрес Fh).

\* См. раздел 6.3.1 документа “Руководство по программированию модуля “xPCIe-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа CONF\_CAN:

Поле структуры	Описание
uint32_t <b>channel</b>	Номер канала (допустимые значения – 1...2)
uint8_t <b>mode</b>	0 – снятие однократного режима. Не 0 – установка однократного режима

### Пример вызова:

```
int fd, ret;
CONF_CAN conf;
fd = open("/dev/can_dev_0", O_RDWR);
conf.channel = 1;
conf.mode = 1;
ret = ioctl(fd, IOCTL_SET_ONESHOT_MODE_CAN, &conf);
if (ret == -1)
printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u oneshot mode %s\n", conf.channel, conf.mode ? "set" : "reset");
```

## 7.6 IOCTL\_SET\_SPEED\_CAN

### Назначение:

Функция конфигурации скорости шины CAN.

### Действие:

В зависимости от значения скорости, определённые значения записываются в регистры CNF1\*(адрес 2AH), CNF2\*\*(адрес 29H) и CNF3\*\*\* (адрес 28H) выбранного канала.

\* См. раздел 6.5.1 документа “Руководство по программированию модуля “xPCIe-CAN”.

\*\* См. раздел 6.5.2 документа “Руководство по программированию модуля “xPCIe-CAN”.

\*\*\* См. раздел 6.5.3 документа “Руководство по программированию модуля “xPCIe-CAN”.

### Примечание:

До вызова этой функции следует обязательно выставить режим работы контроллера в режим конфигурации.

### Входные параметры:

param – указатель на структуру типа SPEED\_CAN:

Поле структуры	Описание
uint32_t channel	Номер канала (допустимые значения – 1...2)
uint32_t speed	Значение скорости (допустимые значения – 125, 250, 500 или 1000)

### Пример вызова:

```
int fd, ret;
    SPEED_CAN spd;
fd = open("/dev/can_dev_0", O_RDWR);
spd.channel = 1;
spd.speed = WORK_SPEED_125;
ret = ioctl(fd, IOCTL_SET_SPEED_CAN, &spd);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u speed was set to %u kHz\n", spd.channel, spd.speed);
```

## 7.7 IOCTL\_SET\_SPEED\_PARAMS\_CAN

### Назначение:

Функция конфигурации скорости шины CAN для нестандартных скоростей.

### Действие:

В соответствии со входными параметрами определённые значения записываются в регистры CNF1\*(адрес 2AH), CNF2\*\*(адрес 29H) и CNF3\*\*\* (адрес 28H) выбранного канала.

\* См. раздел 6.5.1 документа “Руководство по программированию модуля “xPCIe-CAN”.

\*\* См. раздел 6.5.2 документа “Руководство по программированию модуля “xPCIe-CAN”.

\*\*\* См. раздел 6.5.3 документа “Руководство по программированию модуля “xPCIe-CAN”.

### Примечание:

До вызова этой функции следует обязательно выставить режим работы контроллера в режим конфигурации.

### Входные параметры:

param – указатель на структуру типа SPEED\_PARAMS\_CAN:

Поле структуры	Описание
uint32_t channel	Номер канала (допустимые значения – 1...2)
uint8_t brp	Коэффициент деления частоты опорного генератора (допустимые значения – 1...32)
uint8_t sjw	Шаг перестройки синхронизации (допустимые значения – 1...4)
uint8_t sam	Конфигурация точки сэмпирования (допустимые значения – 0...1)
uint8_t btlmode	Выбор величины PS2 (допустимые значения – 0...1)
uint8_t phseg1	Длительность сегмента фазы 1 (допустимые значения – 1...8)
uint8_t phseg2	Длительность сегмента фазы 2 (допустимые значения – 1...8)
uint8_t prseg	Длительность сегмента фазы распространения (допустимые значения – 1...8)
uint8_t wakfil	ФНЧ шины для детектора активности шины в режиме сна (допустимые значения – 0...1)

### Пример вызова:

```
int fd, ret;
SPEED_PARAMS_CAN sp;
fd = open("/dev/can_dev_0", O_RDWR);
sp.channel = 1;
sp.brp = 1; sp.sjw = 1;
sp.btlmode = 1; sp.sam = 1; sp.phseg1 = 3; sp.prseg = 2;
sp.phseg2 = 4; sp.wakfil = 0;
ret = ioctl(fd, IOCTL_SET_SPEED_PARAMS_CAN, &sp);
if (ret == -1)
printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u speed was set\n", spd.channel);
```

## 7.8 IOCTL\_GET\_ERRORS\_CAN

### Назначение:

Функция чтения регистров ошибок контроллера CAN.

### Действие:

Читаются регистры **TEC**\*(адрес 1CH), **REC**\*\* (адрес 1DH) и **EFLG**\*\*\* (адрес 2DH) выбранного канала.

\* См. раздел 6.6.1 документа “Руководство по программированию модуля “xPCie-CAN”.

\*\* См. раздел 6.6.2 документа “Руководство по программированию модуля “xPCie-CAN”.

\*\*\* См. раздел 6.6.3 документа “Руководство по программированию модуля “xPCie-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа ERRORS\_CAN:

Поле структуры	Описание
uint8_t channel	Номер канала (допустимые значения – 1...2)
uint8_t nEFLG	Флаги ошибок шины
uint8_t nTEC	Счётчик ошибок передачи
uint8_t nREC	Счётчик ошибок приёма

### Пример вызова:

```
int fd, ret;
    ERRORS_CAN er;
fd = open("/dev/can_dev_0", O_RDWR);
er.channel = 1;
ret = ioctl(fd, IOCTL_GET_ERRORS_CAN, &er);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u EFLG=%02XH TEC=%u REC=%u \n", er.channel, er.nEFLG, er.nTEC,
        er.nREC);
```

## 7.9 IOCTL\_SET\_MASKS\_CAN

### Назначение:

Установка масок и фильтров на прием сообщений.

### Действие:

В зависимости от выбранного канала (**channel**) и номера фильтра (**filter**) функция записывает значение **rxb\_mode** в биты RXM\*\* регистра **RXBn\*CTRL\*\***.

\* n – номер буфера.

\*\* См. раздел 6.8.1 и 6.8.2 документа “Руководство по программированию модуля “xPCIe-CAN””.

Далее в зависимости от выбранного идентификатора (**ident**) и номера фильтра (**filter**) функция записывает значения масок и фильтров в соответствующие регистры\*.

\* См. раздел 6.9 документа “Руководство по программированию модуля “xPCIe-CAN””.

### Примечание:

Для отключения фильтров и масок поле **rxb\_mode** должна быть равна 3. В этом случае поля **filter**, **ident**, **identific\_m**, **identific\_f** игнорируются и модуль будет принимать из линии все сообщения.

Если **rxb\_mode** и **ident** равны 0, то 26-19 биты **identific\_m** и **identific\_f** должны содержать 0й байт данных, а 18-11 биты **identific\_m** и **identific\_f** должны содержать 1й байт данных.

### Входные параметры:

param – указатель на структуру типа MASKS\_CAN:

Поле структуры	Описание
uint8_t <b>channel</b>	Номер канала (допустимые значения – 1...2)
uint8_t <b>rxb_mode</b>	Режим работы буфера (допустимые значения – 0...3)
uint8_t <b>filter</b>	Номер фильтра (допустимые значения – 0...5). Первые два фильтра (0...1) – 0 буфер, 0 маска; следующие четыре фильтра (2...5) – 1 буфер, 1 маска
uint8_t <b>ident</b>	0 – фильтр применяется для сообщений только с SID; 1 – фильтр применяется для сообщений только с EID
uint32_t <b>identific_m</b>	идентификатор маски; 10-0 биты – SID; 28-11 биты – EID или первые 2 байта данных
uint32_t <b>identific_f</b>	идентификатор фильтра; 10-0 биты – SID; 28-11 биты – EID или первые 2 байта данных

### Пример вызова:

```
int fd, ret;
MASKS_CAN mask;
uint16_t sidmask, sidfilter;
uint32_t eidmask, eidfilter;
fd = open("/dev/can_dev_0", O_RDWR);
mask.channel = 1;
mask.rxb_mode = 2;
mask.filter = 1;
mask.ident = 1;
mask.identific_m = (eidmask << 11) | sidmask;
mask.identific_f = (eidfilter << 11) | sidfilter;
retval = ioctl(fd, IOCTL_SET_MASKS_CAN, &mask);
if (ret == -1)
```



```
printf("ioctl error: %d (%s)\n", errno, strerror(errno));  
else  
printf("Filters and masks for CAN%u was enabled\n", mask.channel);
```

## 7.10 IOCTL\_RESET\_CANn\_TIMER\_CAN

**Назначение:**

Сброс таймера.

**Действие:**

В регистре CANn\*\_CTRL\*\* бит RST\_TIMn\* устанавливается в единицу. Значения остальных битов данного регистра не изменяются.

\* n – номер канала.

\*\* См. раздел 6.1.1 и 6.1.2 документа “Руководство по программированию модуля “xPCIe-CAN”.

**Примечание:**

-

**Входные параметры:**

param – указатель на 32-битную переменную, в которой указан номер канала. Допустимые значения – 1...2.

**Пример вызова:**

```
int fd, ret;
uint32_t channel;
fd = open("/dev/can_dev_0", O_RDWR);
channel = 1;
ret = ioctl(fd, IOCTL_RESET_CANn_TIMER_CAN, &channel);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u TIMER reset\n", channel);
```

## 7.11 IOCTL\_GET\_CANn\_TIMER\_CAN

### Назначение:

Чтение текущего значения таймера.

### Действие:

Поле CUR\_TIM регистра CANn\*\_TIMER\*\* читается в поле nValue.

\* n – номер канала.

\*\* См. раздел 5.2.3 документа “Руководство по программированию модуля “xPCIE-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа TIMER\_TRSH\_CAN:

Поле структуры	Описание
uint8_t nCh	Номер канала (допустимые значения – 1...2)
uint32_t nValue	Значение поля CUR_TIM регистра CANn*_TIMER**

### Пример вызова:

```
int fd, ret;
    TIMER_TRSH_CAN tt;
fd = open("/dev/can_dev_0", O_RDWR);
tt.nCh = 1;
    ret = ioctl(fd, IOCTL_GET_CANn_TIMER_CAN, &tt);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u TIMER.CUR_TIM=%u \n", tt.nValue);
```

## 7.12 IOCTL\_SET\_CANn\_TIMEOUTS\_CAN

### Назначение:

Установка значений абсолютного и интервального таймера.

### Действие:

Поле **absolute** пишется в регистр **CANn\*\_TIMEOUT\_ABSOLUTE\*\***, а поле **interval** пишется в регистр **CANn\*\_TIMEOUT\_INTERVAL\*\*\***.

\* n – номер канала.

\*\* См. раздел 5.1.5 документа “Руководство по программированию модуля “xPCIE-CAN”.

\*\*\* См. раздел 5.1.6 документа “Руководство по программированию модуля “xPCIE-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа **TIMEOUTS\_CAN**:

Поле структуры	Описание
uint8_t <b>channel</b>	Номер канала (допустимые значения – 1...2)
uint32_t <b>absolute</b>	Значение абсолютного таймера, в микросекундах (допустимые значения – 0...3FFFFFFH)
uint32_t <b>interval</b>	Значение интервального таймера, в микросекундах (допустимые значения – 0...3FFFFFFH)

### Пример вызова:

```
int fd, ret;
    TIMEOUTS_CAN t;
fd = open("/dev/can_dev_0", O_RDWR);
t.channel = 1;
    t.absolute = 4 * 16 * 8;
t.interval = 1 * 16 * 8;
ret = ioctl(fd, IOCTL_SET_CANn_TIMEOUTS_CAN, &t);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u timeouts set: absolute=%u us; interval = %u us\n", t.channel, t.absolute,
        t.interval);
```

## 7.13 IOCTL\_RESET\_CAN

**Назначение:**

Программный сброс модуля и аппаратный сброс всех контроллеров CAN.

**Действие:**

Запрещается DMA, очищается буфер DMA, сбрасываются в исходные значения все регистры, таймеры и триггеры модуля. Происходит аппаратный сброс всех контроллеров CAN. Все регистры всех контроллеров CAN возвращаются к значениям по умолчанию.

**Примечание:**

-

**Входные параметры:**

-

**Пример вызова:**

```
int fd, ret;
fd = open("/dev/can_dev_0", O_RDWR);
ret = ioctl(fd, IOCTL_RESET_CAN);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN module reset OK\n");
```

## 7.14 IOCTL\_RESET\_CANn\_CAN

**Назначение:**

Аппаратный сброс контроллера CAN. Все регистры контроллера CAN возвращаются к значениям по умолчанию.

**Действие:**

В бит RST\_CANn\* регистра CANn\*\_CTRL\*\* записывается единица.

\* n – номер канала.

\*\* См. раздел 6.1.1 или 6.1.2 документа “Руководство по программированию модуля “xPCIe-CAN”.

**Примечание:**

-

**Входные параметры:**

param – указатель на 32-битную переменную, в которой указан номер канала. Допустимые значения – 1...2.

**Пример вызова:**

```
int fd, ret;
uint32_t channel;
fd = open("/dev/can_dev_0", O_RDWR);
channel = 1;
ret = ioctl(fd, IOCTL_RESET_CANn_CAN, &channel);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u reset OK\n", channel);
```

## 8. Функции для чтения принятых данных

### 8.1 IOCTL\_RD\_CH\_RAW\_DMA\_CAN

#### Назначение:

Чтение данных из буфера DMA. Данная функция пытается считать требуемое количество новых блоков данных по запрашиваемому каналу.

#### Действие:

Функция проверяет в буфере DMA количество новых блоков данных. Если в поле **number\_block** запрашивается меньше блоков, чем накопилось в буфере DMA, то функция копирует эти блоки данных в поле **buf** и немедленно возвращает результат. Если же запрашивается больше блоков, чем накопилось на данный момент в буфере DMA, то при нулевом времени ожидания функция скопирует имеющиеся блоки данных в поле **buf** и немедленно возвратит результат. А при отличном от нуля времени ожидания процесс перейдёт в состояние ожидания и будет возобновлён при накоплении в DMA требуемого количества блоков данных, истечении указанного времени ожидания, или срабатывании абсолютного или интервального таймера, после чего скопирует накопившееся количество блоков данных (но не больше, чем запрошенное) в поле **buf**. В любом случае после возврата из функции в поле **number\_block** будет указано количество скопированных блоков данных в поле **buf**.

При включённом режиме аппаратного контроля переполнения буфера DMA в случае переполнения буфера произойдёт временная остановка записи в буфер DMA до освобождения свободного места. В этом случае в пакете, который будет записан после этого события в буфер DMA, бит OVF поля RXBnCTRL структуры DMA\_SLOT\_CAN будет установлен в единицу.

#### Примечание:

О реакции на переполнение буфера DMA см. разделы 5.1.1, 5.1.2 и 6.8 документа “Руководство по программированию модуля “xPCIe-CAN””.

Формат структуры DMA\_SLOT\_CAN см. в файле “nmcapi.h” и в разделе 6.8 документа “Руководство по программированию модуля “xPCIe-CAN””.

#### Входные параметры:

param – указатель на структуру типа DMA\_STR\_CAN:

Поле структуры	Описание
uint32_t <b>number_block</b>	Кол-во запрашиваемых/полученных блоков (допустимые значения при вызове – 1...400). После вызова значение может быть 0...<исходное значение при вызове>
uint32_t <b>number_channel</b>	Номер канала (допустимые значения – 1...2)
uint32_t <b>timeout</b>	Максимальное время ожидания требуемого количества блоков данных, в миллисекундах
DMA_SLOT_CAN <b>buf[400]</b>	Область, в которую будут записаны накопившиеся блоки данных

#### Пример вызова:

```
int fd, ret;
DMA_STR_CAN dma;
fd = open("/dev/can_dev_0", O_RDWR);
dma.number_block = 64;
dma.number_channel = 1;
dma.timeout = 1;
ret = ioctl(fd, IOCTL_RD_CH_RAW_DMA_CAN, &dma);
if (ret == -1)
```

```
printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else{
if (dma.number_block > 0)
printf("Received %u blocks from DMA\n", dma.number_block);
else
printf("No new data in DMA\n");
}
```



## 8.2 IOCTL\_READ\_DMA\_BLOCKS\_CAN

### Назначение:

Чтение данных из буфера DMA. Данная функция пытается считать доступное количество новых блоков данных по запрашиваемому каналу, но не более 64 за раз.

### Действие:

Функция проверяет в буфере DMA количество новых блоков данных, копирует эти блоки данных в поле **buf** и немедленно возвращает результат.

### Примечание:

О реакции на переполнение буфера DMA см. разделы 5.1.1, 5.1.2 и 6.8 документа “Руководство по программированию модуля “xPCIe-CAN”.

Формат структуры DMA\_READ\_BLOCK см. в файле “nmcан.h.

### Входные параметры:

param – указатель на структуру типа DMA\_READ\_BLOCK:

Поле структуры	Описание
int numChannel	Номер канала (допустимые значения – 1...2)
unsigned int countBlocks	Фактическое считанное кол-во блоков DMA
DMA_RAW_BLOCK_64 blocks	Массив блоков DMA
STAT_INFO info	Информация о состоянии циклического буфера DMA

### Пример вызова:

```
int fd, ret;
DMA_READ_BLOCK dma;
fd = open("/dev/can_dev_0", O_RDWR);
dma.numChannel = 1;
ret = ioctl(fd, IOCTL_READ_DMA_BLOCKS_CAN, &dma);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else{
    if (dma.number_block > 0)
        printf("Received %u blocks from DMA\n", dma.number_block);
    else
        printf("No new data in DMA\n");
}
```

## 9. Функции для передачи данных

В данном драйвере предусмотрено четыре режима передачи данных:

- синхронный режим передачи (вызов [IOCTL\\_SEND\\_DATA\\_CAN](#));
- асинхронный режим передачи (вызов [IOCTL\\_SEND\\_DATA\\_NOW\\_CAN](#));

До вызова функции асинхронной отправки данных необходимо записать данные в желаемый буфер с помощью функции [IOCTL\\_WRITE\\_DATA\\_TO\\_TR\\_BUF\\_CAN](#). Номера буферов в обеих функциях должны совпадать.

После вызова асинхронной отправки данных [IOCTL\\_SEND\\_DATA\\_NOW\\_CAN](#) необходимо дождаться отправки сообщения либо путём периодического опроса вызовом [IOCTL\\_CHECK\\_TRANSMIT\\_CAN](#), либо путём синхронного вызова [IOCTL\\_WAIT\\_TRANSMIT\\_CAN](#).

**Исключение:** в случае асинхронной передачи с одинаковыми приоритетами одновременно из трёх буферов необходимо использовать следующий алгоритм:

1. Загрузить данные в буфер 2 и отправить его на передачу.
2. Загрузить данные в буфер 1 и отправить его на передачу.
3. Загрузить данные в буфер 0 и отправить его на передачу.
4. Дождаться окончания передачи из буфера 2.
5. Дождаться окончания передачи из буфера 1.
6. Дождаться окончания передачи из буфера 0.
7. Перейти к шагу 1.

Пример реализации алгоритма – тест *nmcantesttxq*.

## 9.1 IOCTL\_WRITE\_DATA\_TO\_FIFO\_CAN1\_V2

### Назначение:

Отправка данных в режиме FIFO для канала 1.

### Действие:

Данные попадают в очередь на отправку.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа CAN\_WRT\_MSG\_V2:

Поле структуры	Описание
unsigned char mRtrBit	наличие RTR бита
unsigned char mTypeFifo	тип FIFO (0 –FIFO, 1-HPFIFO)
unsigned char mIsExtMsg	признак расширенного сообщения
unsigned int mSID	Стандартный идентификатор (допустимые значения – 0...7FFH)
unsigned int mEID	Расширенный идентификатор (допустимые значения – 0...3FFFFFFH и специальное значение FFFFFFFFH, которое означает, что расширенный идентификатор не используется)
unsigned char mDataLength	фактический размер данных (0-8)
unsigned char mData	Данные пакета
unsigned char mMsgId	уникальный идентификатор сообщения

### Пример вызова:

```
int fd, ret;
CAN_WRT_MSG_V2 sd;
fd = open("/dev/can_dev_0", O_RDWR);
sd.mRtrBit = 0;
sd.mTypeFifo = 0;
sd.mIsExtMsg = 1;
sd.mSID = 0x7FF;
sd.mEID = 0x3FFFF;
sd.mData[0] = 0xFF;
sd.mDataLength = 1;
ret = ioctl(fd, IOCTL_WRITE_DATA_TO_FIFO_CAN1_V2, &sd);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u data buffer %u was filled with data\n", sd.nChannel, sd.nBufNumber);
```

## 9.2 IOCTL\_WRITE\_DATA\_TO\_FIFO\_CAN2\_V2

### Назначение:

Отправка данных в режиме FIFO для канала 2.

### Действие:

Данные попадают в очередь на отправку.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа CAN\_WRT\_MSG\_V2:

Поле структуры	Описание
unsigned char mRtrBit	наличие RTR бита
unsigned char mTypeFifo	тип FIFO (0 –FIFO, 1-HPFIFO)
unsigned char mIsExtMsg	признак расширенного сообщения
unsigned int mSID	Стандартный идентификатор (допустимые значения – 0...7FFH)
unsigned int mEID	Расширенный идентификатор (допустимые значения – 0...3FFFFH и специальное значение FFFFFFFFH, которое означает, что расширенный идентификатор не используется)
unsigned char mDataLength	фактический размер данных (0-8)
unsigned char mData	Данные пакета
unsigned char mMsgId	уникальный идентификатор сообщения

### Пример вызова:

```
int fd, ret;
CAN_WRT_MSG_V2 sd;
fd = open("/dev/can_dev_0", O_RDWR);
sd.mRtrBit = 0;
sd.mTypeFifo = 0;
sd.mIsExtMsg = 1;
sd.mSID = 0x7FF;
sd.mEID = 0x3FFFF;
sd.mData[0] = 0xFF;
sd.mDataLength = 1;
ret = ioctl(fd, IOCTL_WRITE_DATA_TO_FIFO_CAN1_V2, &sd);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u data buffer %u was filled with data\n", sd.nChannel, sd.nBufNumber);
```

## 9.3 IOCTL\_WRITE\_DATA\_TO\_FIFO\_CAN1

### Назначение:

Отправка данных в режиме FIFO для канала 1.

### Действие:

Данные попадают в очередь на отправку.

### Примечание:

- Функция сохранена для совместимости с предыдущей версией. Для нового ПО используйте [IOCTL\\_WRITE\\_DATA\\_TO\\_FIFO\\_CAN1\\_V2](#), где добавлен бит RTR.

### Входные параметры:

param – указатель на структуру типа CAN\_WRT\_MSG:

Поле структуры	Описание
unsigned char mTypeFifo	тип FIFO (0 –FIFO, 1-HPFIFO)
unsigned char mIsExtMsg	признак расширенного сообщения
unsigned int mSID	Стандартный идентификатор (допустимые значения – 0...7FFH)
unsigned int mEID	Расширенный идентификатор (допустимые значения – 0...3FFFFH и специальное значение FFFFFFFFH, которое означает, что расширенный идентификатор не используется)
unsigned char mDataLength	фактический размер данных (0-8)
unsigned char mData	Данные пакета
unsigned char mMsgId	уникальный идентификатор сообщения

### Пример вызова:

```
int fd, ret;
CAN_WRT_MSG sd;
fd = open("/dev/can_dev_0", O_RDWR);
sd.mTypeFifo = 0;
sd.mIsExtMsg = 1;
sd.mSID = 0x7FF;
sd.mEID = 0x3FFFF;
sd.mData[0] = 0xFF;
sd.mDataLength = 1;
ret = ioctl(fd, IOCTL_WRITE_DATA_TO_FIFO_CAN1, &sd);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u data buffer %u was filled with data\n", sd.nChannel, sd.nBufNumber);
```

## 9.4 IOCTL\_WRITE\_DATA\_TO\_FIFO\_CAN2

### Назначение:

Отправка данных в режиме FIFO для канала 2.

### Действие:

Данные попадают в очередь на отправку.

### Примечание:

- Функция сохранена для совместимости с предыдущей версией. Для нового ПО используйте [IOCTL\\_WRITE\\_DATA\\_TO\\_FIFO\\_CAN2\\_V2](#), где добавлен бит RTR.

### Входные параметры:

param – указатель на структуру типа CAN\_WRT\_MSG:

Поле структуры	Описание
unsigned char mTypeFifo	тип FIFO (0 –FIFO, 1-HPFIFO)
unsigned char mIsExtMsg	признак расширенного сообщения
unsigned int mSID	Стандартный идентификатор (допустимые значения – 0...7FFH)
unsigned int mEID	Расширенный идентификатор (допустимые значения – 0...3FFFFH и специальное значение FFFFFFFFH, которое означает, что расширенный идентификатор не используется)
unsigned char mDataLength	фактический размер данных (0-8)
unsigned char mData	Данные пакета
unsigned char mMsgId	уникальный идентификатор сообщения

### Пример вызова:

```
int fd, ret;
CAN_WRT_MSG sd;
fd = open("/dev/can_dev_0", O_RDWR);
sd.mTypeFifo = 0;
sd.mIsExtMsg = 1;
sd.mSID = 0x7FF;
sd.mEID = 0x3FFFF;
sd.mData[0] = 0xFF;
sd.mDataLength = 1;
ret = ioctl(fd, IOCTL_WRITE_DATA_TO_FIFO_CAN1, &sd);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u data buffer %u was filled with data\n", sd.nChannel, sd.nBufNumber);
```

## 9.5 IOCTL\_WRITE\_DATA\_TO\_TR\_BUF\_CAN

### Назначение:

Запись данных в буфер отправки.

### Действие:

Данные из входных полей последовательно записываются в регистры **TXBn\*CTRL**, **TXBn\*SIDH**, **TXBn\*SIDL**, **TXBn\*EID8**, **TXBn\*EID0**, **TXBn\*DLC**, **TXBn\*Dm \*\***.

Все функции работы с буферами контроллера CAN защищены общим семафором (индивидуальным для каждого канала и буфера) для возможности их корректного использования из параллельных процессов.

\* n – номер буфера.

\*\* См. раздел 6.7.1-6.7.7 документа “Руководство по программированию модуля “xPCIE-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа SEND\_DATA:

Поле структуры	Описание
uint8_t nChannel	Номер канала (допустимые значения – 1...2)
uint8_t nBufNumber	Номер буфера, в который будут записываться данные (допустимые значения – 0...2)
uint8_t nPriority	Приоритет буфера передачи (допустимые значения – 0...3; 0 – низший, 3 – наивысший)
uint32_t SID	Стандартный идентификатор (допустимые значения – 0...7FFF)
uint32_t EID	Расширенный идентификатор (допустимые значения – 0...3FFFFFFH и специальное значение FFFFFFFFH, которое означает, что расширенный идентификатор не используется)
uint8_t nData[8]	Буфер с данными
uint32_t nSize	Размер передаваемых данных

### Пример вызова:

```
int fd, ret;
SEND_DATA sd;
fd = open("/dev/can_dev_0", O_RDWR);
sd.nChannel = 1;
sd.nBufNumber = 0;
sd.nPriority = 0;
sd.SID = 0x7FFF;
sd.EID = 0x3FFFF;
sd.nData[0] = 0xFF;
sd.nSize = Size;
ret = ioctl(fd, IOCTL_WRITE_DATA_TO_TR_BUF_CAN, &sd);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u data buffer %u was filled with data\n", sd.nChannel, sd.nBufNumber);
```

## 9.6 IOCTL\_SEND\_DATA\_CAN

### Назначение:

Синхронная отправка сообщения.

### Действие:

Данные из входных полей последовательно записываются в регистры **TXBn\*CTRL**, **TXBn\*SIDH**, **TXBn\*SIDL**, **TXBn\*EID8**, **TXBn\*EID0**, **TXBn\*DLC**, **TXBn\*Dm** \*\*. После этого в бит TXREQ регистра **TXBn\*CTRL** \*\* будет записана единица и процесс перейдет в состояние ожидания, из которого он будет выведен либо после прерывания, сигнализирующего об окончании отправки данных, либо по истечении максимального времени ожидания отправки. В последнем случае функция возвратит -1, в errno будет установлено значение ETIME, а в поле **txb\_ctrl** будет записано значение регистра **TXBn\*CTRL**.

Все функции работы с буферами контроллера CAN защищены общим семафором (индивидуальным для каждого канала и буфера) для возможности их корректного использования из параллельных процессов.

\* n – номер буфера.

\*\* См. раздел 6.7.1-6.7.7 документа “Руководство по программированию модуля “xPCIe-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа SEND\_DATA:

Поле структуры	Описание
uint8_t nChannel	Номер канала (допустимые значения – 1...2)
uint8_t nBufNumber	Номер буфера, в который будут записываться данные (допустимые значения – 0...2)
uint8_t nPriority	Приоритет буфера передачи (допустимые значения – 0...3; 0 – низший, 3 – наивысший)
TXBCTRL txb_ctrl	Значение регистра <b>TXBn*CTRL</b> в случае истечения времени ожидания отправки сообщения
uint32_t timeout	Максимальное время ожидания отправки сообщения, в миллисекундах
uint32_t SID	Стандартный идентификатор (допустимые значения – 0...7FFH)
uint32_t EID	Расширенный идентификатор (допустимые значения – 0...3FFFFFFH и специальное значение FFFFFFFFH, которое означает, что расширенный идентификатор не используется)
uint8_t nData[8]	Буфер с данными
uint32_t nSize	Размер передаваемых данных

### Пример вызова:

```
int fd, ret;
SEND_DATA sd;
fd = open("/dev/can_dev_0", O_RDWR);
sd.nChannel = 1;
sd.nBufNumber = 0;
sd.nPriority = 0;
sd.timeout = 100;
```



```
sd.SID = 0x7FF;
sd.EID = 0x3FFFF;
sd.nData[0] = 0xFF;
sd.nSize=Size;
ret = ioctl(fd, IOCTL_SEND_DATA_CAN, &sd);
if (ret == -1)
{
printf("ioctl error: %d (%s); TXB%uCTRL=%02XH\n", errno, strerror(errno), sd.nBufNumber,
sd.txb_ctrl);
}
else
printf("CAN%u data buffer %u was filled with data\n", sd.nChannel, sd.nBufNumber);
```

## 9.7 IOCTL\_SEND\_DATA\_NOW\_CAN

### Назначение:

Запуск асинхронной передачи сообщения.

### Действие:

Бит TXREQ регистра **TXBn\*CTRL\*\*** устанавливается в единицу.

Все функции работы с буферами контроллера CAN защищены общим семафором (индивидуальным для каждого канала и буфера) для возможности их корректного использования из параллельных процессов.

\* n – номер буфера.

\*\* См. раздел 6.7.1 документа “Руководство по программированию модуля “xPCIe-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа SEND\_DATA\_NOW:

Поле структуры	Описание
uint8_t <b>nChannel</b>	Номер канала (допустимые значения – 1...2)
uint8_t <b>nBuf</b>	Номер буфера, который будет установлен на отправку (допустимые значения – 0...2)

### Пример вызова:

```
int fd, ret;
    SEND_DATA_NOW sdn;
fd = open("/dev/can_dev_0", O_RDWR);
sdn.nChannel = 1;
sdn.nBuf = 0;
ret = ioctl(fd, IOCTL_SEND_DATA_NOW_CAN, &sdn);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u data buffer %u was requested to send\n", sdn.nChannel, sdn.nBuf);
```

## 9.8 IOCTL\_CHECK\_TRANSMIT\_CAN

### Назначение:

Проверка правильности отправки сообщения.

### Действие:

Данная функция читает регистр **TXBn\*CTRL\*\*** в поле **txb\_ctrl**. Если бит TXREQ окажется равен нулю, то функция возвращает 0. В противном случае функция возвратит значение -1, а в errno будет установлено значение EBUSY.

\* n – номер буфера.

\*\* См. Раздел 6.7.1 документа “Руководство по программированию модуля “xPCIE-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа SEND\_DATA\_NOW:

Поле структуры	Описание
uint8_t nChannel	Номер канала (допустимые значения – 1...2)
uint8_t nBuf	Номер буфера, который будет проверен (допустимые значения – 0...2)
TXBCTRL txb_ctrl	Значение регистра <b>TXBn*CTRL</b>

### Пример вызова:

```
int fd, ret;
SEND_DATA_NOW sdn;
fd = open("/dev/can_dev_0", O_RDWR);
sdn.nChannel = 1;
sdn.nBuf = 0;
ret = ioctl(fd, IOCTL_CHECK_TRANSMIT_CAN, &sdn);
if (ret == -1)
{
printf("ioctl error: %d (%s); TXB%uCTRL=%02XH\n", errno, strerror(errno), sdn.nBuf,
sdn.txb_ctrl);
}
else
printf("CAN%u data buffer %u was sended\n", sdn.nChannel, sdn.nBuf);
```

## 9.9 IOCTL\_WAIT\_TRANSMIT\_CAN

### Назначение:

Ожидание отправки сообщения.

### Действие:

Данная функция переводит процесс в состояние ожидания, из которого он будет выведен либо после прерывания, сигнализирующего об окончании отправки данных, либо по истечении максимального времени ожидания отправки. В последнем случае функция возвратит -1, в errno будет установлено значение ETIME, а в поле **txb\_ctrl** будет записано значение регистра **TXBn\*CTRL**.

Функция защищена общим с функциями работы с буферами контроллера CAN семафором (индивидуальным для каждого канала и буфера) для возможности их корректного использования из параллельных процессов.

\* n – номер буфера.

\*\* См. Раздел 6.7.1 документа “Руководство по программированию модуля “xPCIE-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа SEND\_DATA\_NOW:

Поле структуры	Описание
uint8_t nChannel	Номер канала (допустимые значения – 1...2)
uint8_t nBuf	Номер буфера, чья отправка будет ожидаться (допустимые значения – 0...2)
TXBCTRL txb_ctrl	Значение регистра <b>TXBn*CTRL</b>
uint32_t timeout	Максимальное время ожидания отправки сообщения, в миллисекундах

### Пример вызова:

```
int fd, ret;
SEND_DATA_NOW sdn;
fd = open("/dev/can_dev_0", O_RDWR);
sdn.nChannel = 1;
sdn.nBuf = 0;
sdn.timeout = 100;
ret = ioctl(fd, IOCTL_WAIT_TRANSMIT_CAN, &sdn);
if (ret == -1)
{
printf("ioctl error: %d (%s); TXB%uCTRL=%02XH\n", errno, strerror(errno), sdn.nBuf,
sdn.txb_ctrl);
}
else
printf("CAN%u data buffer %u was sended\n", sdn.nChannel, sdn.nBuf);
```

## 9.10 IOCTL\_END\_TRANSMIT\_CAN

### Назначение:

Снятие запроса отправки сообщения.

### Действие:

Данная функция записывает ноль в бит TXREQ регистра **TXBn\*CTRL\*\***.

\* n – номер буфера.

\*\* См. Раздел 6.7.1 документа “Руководство по программированию модуля “xPCIE-CAN”.

### Примечание:

-

### Входные параметры:

param – указатель на структуру типа SEND\_DATA\_NOW:

Поле структуры	Описание
uint8_t nChannel	Номер канала (допустимые значения – 1...2)
uint8_t nBuf	Номер буфера, с которого будет снят запрос на отправку (допустимые значения – 0...2)

### Пример вызова:

```
int fd, ret;
SEND_DATA_NOW sdn;
fd = open("/dev/can_dev_0", O_RDWR);
sdn.nChannel = 1;
sdn.nBuf = 0;
ret = ioctl(fd, IOCTL_END_TRANSMIT_CAN, &sdn);
if (ret == -1)
printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u data buffer %u request to send was cleared\n", sdn.nChannel, sdn.nBuf);
```

## 9.11 IOCTL\_ABAT\_CAN

### Назначение:

Остановка всех активных передач.

### Действие:

Функция устанавливает бит АВАТ регистра `CAN_CTRL*` (адрес Fh) в соответствии со значением поля `mode`.

\* См. Раздел 6.3.1 документа “Руководство по программированию модуля “xPCIE-CAN”.

### Примечание:

-

### Входные параметры:

`param` – указатель на структуру типа `CONF_CAN`:

Поле структуры	Описание
<code>uint32_t channel</code>	Номер канала (допустимые значения – 1...2)
<code>uint8_t mode</code>	0 – снятие запроса на остановку всех активных передач. Не 0 - запрос на остановку всех активных передач

### Пример вызова:

```
int fd, ret;
CONF_CAN conf;
fd = open("/dev/can_dev_0", O_RDWR);
conf.channel = 1;
conf.mode = 1;
ret = ioctl(fd, IOCTL_ABAT_CAN, &conf);
if (ret == -1)
    printf("ioctl error: %d (%s)\n", errno, strerror(errno));
else
    printf("CAN%u ABAT was %s\n", conf.channel, (conf.mode != 0) ? "set" : "reset");
```

## 10. Обновление драйвера

<b>Версия библиотеки</b>	<b>Дата</b>	<b>Изменение</b>
7.1	21.12.2021	- Драйвер создан на основе версии для Linux

## 11. Обновление руководства

Версия документа	Дата	Изменение
7.1	21.12.2021	- Документ создан на основе версии для Linux
7.2	10.02.2022	- Незначительные правки оформления.