**Руководство (v7.0)**

# По работе с библиотекой модулей “mPCIe – CAN”, “PCIe – CAN”

Интерфейс ISO-11898
(CAN Bus)

Для библиотек версии 7.0

**ОС LINUX**


**29.07.2020**

**ООО “Новомар”**

# Оглавление

# 1 Введение.

Библиотека поддерживает модули “**mPCIe-CAN**”, “**PCIe-CAN**”(далее **xPCIe-CAN**).
Взаимодействие с библиотекой происходит посредством функций, перечень которых находится в файле “libnmcan.h”.

# 2 Сборка библиотеки.

1. Создайте отдельную папку.
2. Скачайте в эту папку архив с исходными текстами драйвера и распакуйте его.
3. Скачайте в эту папку архив с исходными текстами библиотеки и распакуйте его.
4. Выполните команду “cd mPCIe-CAN/lib” для перехода в каталог с исходными текстами библиотеки.
5. Выполните команду “make”.

# 3 Список доступных функций по версиям библиотек.

| Название вызова | Краткое описание |
|---|---|
| Список функций, доступных **в библиотеке версии 6.0** | |
| NM_CAN_GetDevName | Получить имя устройства по его номеру. |
| NM_CAN_GetDevPath | Получить полный маршрут устройства по его номеру. |
| NM_CAN_WriteReg | Запись регистра модуля. |
| NM_CAN_ReadReg | Чтение регистра модуля. |
| NM_CAN_WriteCanReg | Запись регистра контроллера канала. |
| NM_CAN_ReadCanReg | Чтение регистра контроллера канала. |
| NM_CAN_ModifyCanReg | Модификация регистра контроллера канала. |
| NM_CAN_WriteCanRegs | Запись регистров контроллера канала. |
| NM_CAN_ReadCanRegs | Чтение регистров контроллера канала. |
| NM_CAN_GetVersion | Информация о плате. |
| NM_CAN_GetDriverVersion | Информация о драйвере. |
| NM_CAN_EnableDMA | Разрешение работы DMA. |
| NM_CAN_DisableDMA | Запрет работы DMA. |
| NM_CAN_SetMode | Установка режима работы канала. |
| NM_CAN_GetMode | Чтение режима работы канала. |
| NM_CAN_SetOneshotMode | Установка/снятие режима однократной попытки отправки сообщения для канала. |
| NM_CAN_SetSpeed | Установка скорости работы канала. |
| NM_CAN_SetSpeedByParams | Установка специальной скорости работы канала. |
| NM_CAN_GetErrors | Чтение регистров ошибок канала. |
| NM_CAN_SetMasks | Установка масок и фильтров на прием сообщений канала. |
| NM_CAN_ResetTimer | Остановка таймера канала. |
| NM_CAN_GetTimer | Чтение текущего значения таймера канала. |
| NM_CAN_SetTimeouts | Установка абсолютного и интервального таймера прерываний канала. |
| NM_CAN_Reset | Сброс модуля. |
| NM_CAN_ResetCAN | Сброс канала. |
| NM_CAN_ReadChannelRawDMA | Чтение данных из DMA канала. |
| NM_CAN_WriteDataToSendBuf | Запись данных в буфер отправки канала. |
| NM_CAN_SendData | Запись данных в буфер отправки канала, запуск передачи сообщения и ожидание завершения передачи. |
| NM_CAN_SendDataRightNow | Запуск передачи сообщения из буфера отправки канала. |

| | |
|---|---|
| NM_CAN_CheckTransmit | Проверка завершения и правильности передачи сообщения из буфера отправки канала. |
| NM_CAN_WaitTransmit | Ожидание завершения и проверка правильности передачи сообщения из буфера отправки канала. |
| NM_CAN_EndTransmit | Снятие запроса на передачу сообщения из буфера отправки канала. |
| NM_CAN_ABAT | Установка/снятие режима остановки всех активных передач. |
| NM_CAN_DecodeBuf | Декодирование пакета, полученного из буфера DMA. |
| Список функций, добавленных **в библиотеке версии 7.0** | |
| NM_CAN_ReadDMA | Чтение данных из буфера DMA |
| NM_CAN_SetSendMode | Устанавливает режим передачи канала |
| NM_CAN_GetCountMsgInFifo | Получает текущее кол-во сообщений в FIFO |
| NM_CAN_GetCountMsgInHpFifo | Получает текущее кол-во сообщений в HPFIFO |
| NM_CAN_WriteDataToFifo | Записать сообщение в FIFO на отправку |
| NM_CAN_WriteDataToHpFifo | Записать сообщение в HPFIFO на отправку |
| NM_CAN_SetTxPause | Управление флагом TX_Pause |

# 4 Описание использования функций библиотек.

Функции NM_CAN_GetDevName и NM_CAN_GetDevPath возвращают число байт, записанных в буфер.
Остальные функции, в случае удачного выполнения запроса, возвращают 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.**

# 5 Стандартные функции

## 1.1 NM_CAN_GetDevName

**Назначение:**
Получение имени устройства по его номеру.

**Действие:**
Функция записывает в **psz** имя устройства, не переходя за границу буфера **strsize**.

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| uint32_t **devnum** | Номер устройства |
| char* **psz** | Указатель на буфер, в который будет записано имя устройства |
| uint32_t **strsize** | Размер буфера |

**Пример вызова:**

```
char devname[32];
uint32_t devnamelen;
devnamelen = NM_CAN_GetDevName(0, devname, sizeof(devname));
if (devnamelen > 0)
  printf("devname: ‘%s’\n", devname);
```

## 1.2 NM_CAN_GetDevPath

**Назначение:**
Получение полного маршрута устройства по его номеру.

**Действие:**
Функция записывает в **psz** полный маршрут к устройству, не переходя за границу буфера **strsize**.

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| uint32_t **devnum** | Номер устройства |
| char* **psz** | Указатель на буфер, в который будет записан полный маршрут к устройству |
| uint32_t **strsize** | Размер буфера |

**Пример вызова:**

```
char devpath[32];
uint32_t devpathlen;
devpathlen = NM_CAN_GetDevPath(0, devpath, sizeof(devpath));
if (devpathlen > 0)
  printf("devpath: ‘%s’\n", devpath);
```

## 1.3 NM_CAN_WriteReg

**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!!!

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **addr** | Адрес регистра |
| uint32_t **data** | Данные для записи |

**Пример вызова:**

```
int fd, ret;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_WriteReg(fd, 0x2000, 0);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
```

## 1.4 NM_CAN_ReadReg

**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!

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **addr** | Адрес регистра |
| uint32_t* **pdata** | Указатель на возвращаемый результат |

**Пример вызова:**

```
int fd, ret;
uint32_t data;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_ReadReg(fd, 0x2000, &data);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
```

## 1.5 NM_CAN_WriteCanReg

**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 в драйвере 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t **addr** | Адрес регистра |
| uint8_t **data** | Данные для записи |

**Пример вызова:**

```
int fd, ret;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_WriteCanReg(fd, 1, CANINTE, 0);
if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
```

## 1.6 NM_CAN_ReadCanReg

**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 в драйвере 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t **addr** | Адрес регистра |
| uint8_t* **pdata** | Указатель на возвращаемый результат |

**Пример вызова:**

```
int fd, ret;
uint8_t data;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_ReadCanReg(fd, 1, CAN_CTRL, &data);
if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
```

## 1.7 NM_CAN_ModifyCanReg

**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 errno будет установлено значение ETIME.
Все функции работы с регистрами контроллера CAN защищены в драйвере 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t **addr** | Адрес регистра |
| uint8_t **mask** | Маска для указания модифицируемых битов регистра |
| uint8_t **data** | Новое значение для модифицируемых битов регистра |

**Пример вызова:**

```
int fd, ret;
fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_ModifyReg(fd, 1, TXB0CTRL, 0x08, 0x08);
if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
```

## 1.8 NM_CAN_WriteCanRegs

**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 в драйвере 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t **addr** | Адрес первого регистра |
| uint8_t* **pdata** | Указатель на записываемые данные |
| uint32_t **datasize** | Количество записываемых данных (допустимые значения – 1…16) |

**Пример вызова:**

```
int fd, ret;
uint8_t data[] = { 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF };
fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_WriteCanRegs(fd, 1, TXB0CTRL + 1, data,
sizeof(data));
if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
```

## 1.9 NM_CAN_ReadCanRegs

**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 в драйвере 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t **addr** | Адрес первого регистра |
| uint8_t* **pdata** | Указатель на считываемые данные |
| uint8_t **datasize** | Количество считываемых данных (допустимые значения – 1…16) |

**Пример вызова:**

```
int fd, ret;
uint8_t data[6];
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_ReadCanRegs(fd, 1, TXB0CTRL + 1, data, sizeof(data));
if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
```

## 1.10 NM_CAN_GetVersion

**Назначение:**
Чтение информации о модуле.

**Действие:**
Функция заполняет все поля структуры **VERSION_CAN**.

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| VERSION_CAN* **pversion** | Указатель на возвращаемую структуру VERSION_CAN |

**Пример вызова:**

```
int fd, ret;
    VERSION_CAN ver;
fd = open("/dev/can_dev_0", O_RDWR);
    ret = NM_CAN_GetVersion(fd, &ver);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
```

## 1.11 NM_CAN_GetDriverVersion

**Назначение:**
Чтение версии и даты драйвера.

**Действие:**
Функция заполняет 32-битную переменную.

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t* **pdrv_version** | Указатель на возвращаемое значение, в котором 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 = NM_CAN_GetDriverVersion(fd, &drv_ver_date);
if (ret == -1)
  printf("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);
```

# 2 Функции конфигурации

## 2.1 NM_CAN_EnableDMA

**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.

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |

**Пример вызова:**

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

## 2.2 NM_CAN_DisableDMA

**Назначение:**
Запрет работы DMA.

**Действие:**
Нулевой бит регистра **DMA_DATA_BASE*** (адрес 1000h) и биты BnBFM, BnBFE и BnBFS регистра **BFPCTRL**** (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.

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |

**Пример вызова:**

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

## 2.3 NM_CAN_SetMode

**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 NM_CAN_GetMode.

* 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*".

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| CAN_MODE **mode** | Новый режим работы контроллера |

**Пример вызова:**

```
int fd, ret;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_SetMode (fd, 1, CAN_WORK);
    if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
else
 printf("CAN%u mode set to %u\n", 1, CAN_WORK);
```

## 2.4 NM_CAN_GetMode

**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:**

-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| CAN_MODE* **pmode** | Указатель на текущий режим работы контроллера |

**Пример вызова:**

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

## 2.5 NM_CAN_SetOneshotMode

**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:**

-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| int **osm** | 0 – снятие однократного режима.<br>Не 0 – установка однократного режима |

**Пример вызова:**

```
int fd, ret;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_SetOneshotMode (fd, 1, 1);
    if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
else
 printf("CAN%u oneshot mode set\n", 1);
```

## 2.6 NM_CAN_SetSpeed

**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.*

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| 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;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_SetSpeed (fd, 1, WORK_SPEED_125);
if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u speed was set to %u kHz\n", 1, WORK_SPEED_125);
```

## 2.7 NM_CAN_SetSpeedByParams

**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.*

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| 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 сэмплирования (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;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_SetSpeedByParams(fd, 1, 1, 1, 1, 1, 3, 4, 2, 0);
if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
else
     printf("CAN%u speed was set\n", 1);
```

## 2.8 NM_CAN_GetErrors

**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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t* **pEFLG** | Укаазтель на флаги ошибок шины |
| uint8_t* **pTEC** | Указатель на счётчик ошибок передачи |
| uint8_t* **pREC** | Указатель на счётчик ошибок приёма |

**Пример вызова:**

```
int fd, ret;
uint8_t nEFLG, nTEC, nREC;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_GetErrors (fd, 1, &nEFLG, &nTEC, &nREC);
if (ret == -1)
 printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u EFLG=%02XH TEC=%u REC=%u \n", 1, nEFLG, nTEC, nREC);
```

## 2.9 NM_CAN_SetMasks

**Назначение:**
Установка масок и фильтров на прием сообщений.

**Действие:**
В зависимости от выбранного канала (**channel**) и номера фильтра (**filter**) функция записывает значение **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 (**eid**) 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 **mode** должна быть равно RXB_MODE_OFF. В этом случае параметры **filter**, **eid**, **sidm**, **eidm_or_2bytes**, **sidf**, **eidf_or_2bytes** игнорируются и модуль будет принимать из линии все сообщения.
Если **mode** равно RXB_MODE_ALL и **eid** равно нулю, то **eidm_or_2bytes** и **eidf_or_2bytes** должны содержать маску и фильтр для двух первых байт данных.

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| RXB_MODE **mode** | Режим работы буфера (допустимые значения – RXB_MODE_ALL , RXB_MODE_SID, RXB_MODE_EID, RXB_MODE_OFF) |
| uint8_t **filter** | Номер фильтра (допустимые значения – 0…5). Первые два фильтра (0…1) – 0 буфер, 0 маска; следующие четыре фильтра (2…5) – 1 буфер, 1 маска |
| int **eid** | 0 – фильтр применяется для сообщений только с SID; Не 0 – фильтр применяется для сообщений только с EID |
| uint32_t **sidm** | идентификатор маски SID (допустимые значения – 0…7FFH) |
| uint32_t **eidm_or_2bytes** | В зависимости от **mode** и **eid** – идентификатор маски EID (допустимые значения – 0…3FFFFH) или маска первых 2х байт данных (биты 15-8 – маска 0го байта данных; биты 7-0 – маска 1го байта данных) |
| uint32_t **sidf** | идентификатор фильтра EID (допустимые значения – 0…3FFFFH) |
| uint32_t **eidf_or_2bytes** | В зависимости от **mode** и **eid** – идентификатор фильтра EID (допустимые значения – 0…3FFFFH) или первые 2 байта данных (биты 15-8 – 0й байт данных; биты 7-0 – 1й байт данных) |

**Пример вызова:**

```
int fd, ret;
    uint32_t sidm, sidf, eidm, eidf;
fd = open("/dev/can_dev_0", O_RDWR);
retval = NM_CAN_SetMasks(fd, 1, RXB_MODE_EID, 1, 1, sidm, eidm, sidf, eidf);
if (ret == -1)
```

```
  printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("Filters and masks for CAN%u was enabled\n", 1);
```

## 2.10 NM_CAN_ResetTimer

**Назначение:**
Сброс таймера.

**Действие:**
В регистре **CANn*_CTRL**** бит RST_TIMn* устанавливается в ноль. 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |

**Пример вызова:**

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

## 2.11 NM_CAN_GetTimer

**Назначение:**
Чтение текущего значения таймера.

**Действие:**
Поле CUR_TIM регистра **CANn*_TIMER**** читается в параметр **ptmr**.

* n – номер канала.
** См. раздел 5.2.3 документа "*Руководство по программированию модуля "xPCIe-CAN*".

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint32_t* **ptmr** | Указатель на значение поля CUR_TIM регистра **CANn*_TIMER**** |

**Пример вызова:**

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

## 2.12 NM_CAN_SetTimeouts

**Назначение:**
Установка значений абсолютного и интервального таймера.

**Действие:**
Поле **abs_timeout** пишется в регистр **CANn*_TIMEOUT_ABSOLUTE****, а поле **itv_timeout** 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint32_t **abs_timeout** | Значение абсолютного таймера, в микросекундах (допустимые значения – 0…3FFFFFH) |
| uint32_t **itv_timeout** | Значение интервального таймера, в микросекундах (допустимые значения – 0…3FFFFFH) |

**Пример вызова:**

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

## 2.13 NM_CAN_Reset

**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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |

**Пример вызова:**

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

## 2.14 NM_CAN_ResetCAN

**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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |

**Пример вызова:**

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

## 2.15 NM_CAN_SetTxPause

**Назначение:**
Управление флагом TX_Pause.

**Действие:**
Управление битом TX_Pause регистра REG_CANx_FIFO_CONSTAT.

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t fifo | 0 – fifo, 1 - hpfifo |
| uint8_t value | Бит TX_Pause |

**Пример вызова:**

```
int fd, ret;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_SetTxPause(fd, 1, 0, 1);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u SetTxPause OK \n", 1);
```

# 3 Функции для чтения принятых данных

## 3.1 NM_CAN_ReadChannelRawDMA

**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 параметре **pnblocks** запрашивается меньше блоков, чем накопилось в буфере DMA, то функция копирует эти блоки данных в параметр **pdata** 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 параметр **pdata** 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 параметр **pdata**. В любом случае после возврата из функции в параметре **pnblocks** будет указано количество скопированных блоков данных в параметр **pdata**.

**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 “nmcan.h” и в разделе 6.8 документа “*Руководство по программированию модуля “xPCIe-CAN*”.

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| DMA_SLOT_CAN* **pdata** | Указатель на область, в которую будут записаны накопившиеся блоки данных |
| uint32_t* **pnblocks** | Кол-во запрашиваемых/полученных блоков (допустимые значения при вызове – 1…400). После вызова значение может быть 0…<исходное значение при вызове> |
| uint32_t **timeout** | Максимальное время ожидания требуемого количества блоков данных, в миллисекундах |

**Пример вызова:**

```
int fd, ret;
DMA_SLOT_CAN data[64];
uint32_t nblocks = sizeof(data) / sizeof(*data);
fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_ReadChannelRawDMA(fd, 1, &data, &nblocks, 1);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
else{
  if (nblocks > 0)
    printf("Received %u blocks from DMA\n", nblocks);
  else
    printf("No new data in DMA\n");
     }
```

## 3.2 NM_CAN_ReadDMA

**Назначение:**
Чтение данных из буфера DMA. Данная функция пытается считать возможное количество новых блоков данных по запрашиваемому каналу.

**Действие:**
Функция проверяет в буфере DMA количество новых блоков данных, копирует эти блоки данных в параметр **pdata** 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 "nmcan.h" и в разделе 6.8 документа "*Руководство по программированию модуля "xPCIe-CAN*".

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| `DMA_READ_BLOCK` * **pdata** | Указатель на область, в которую будут записаны накопившиеся блоки данных |

**Пример вызова:**

```
int fd, ret;
DMA_READ_BLOCK data;
fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_ReadDMA (fd, &data);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
else{
  if (nblocks > 0)
    printf("Received %u blocks from DMA\n", nblocks);
  else
    printf("No new data in DMA\n");
     }
  printf("CAN%u ABAT was set\n", 1);
```

## 3.3 NM_CAN_DecodeBuf

**Назначение:**
Распаковка данных, полученных из DMA.

**Действие:**
Функция декодирует данные из параметра **psrcdata** и устанавливает все остальные параметры соответствующими значениями.

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| DMA_SLOT_CAN* **psrcdata** | Указатель на данные, полученные из DMA |
| uint32_t* **pbuf** | Указатель на номер буфера RXB, в который был записан пакет |
| uint32_t* **pch** | Указатель на номер канала, в который пришёл пакет |
| uint32_t* **ptmr** | Указатель на значение таймера на момент старта получения пакета |
| uint32_t* **psid** | Указатель на SID пакета |
| uint32_t* **peid** | Указатель на EID пакета (в случае если там будет FFFFFFFFH, то пакет был передан без EID) |
| uint8_t **ponlydata[15]** | Указатель на данные пакета |
| uint32_t* **pdatasize** | Указатель на размер данных полученного пакета |

**Пример вызова:**

```
DMA_SLOT_CAN slot;
uint32_t nbuf, channel, tmr, sid, eid, datasize;
uint8_t data[15];
NM_CAN_DecodeBuf(&slot, &nbuf, &channel, &tmr, &sid, &eid, data, &datasize);
```

# 4 Функции для передачи данных

В данной библиотеке предусмотрено три метода передачи данных:

два для режима **Native**:
- синхронный методпередачи (вызов NM_CAN_SendData);
- асинхронный методпередачи (вызов NM_CAN_SendDataRightNow);

- и передача в режиме **FIFO** с помощью вызовов: NM_TTCAN_WriteDataToFifo, NM_TTCAN_WriteDataToHpFifo.

До вызова функции асинхронной отправки данных необходимо записать данные в желаемый буфер с помощью функции NM_CAN_WriteDataToSendBuf. Номера буферов в обеих функциях должны совпадать.

После вызова асинхронной отправки данных NM_CAN_SendDataRightNow необходимо дождаться отправки сообщения либо путём периодического опроса вызовом NM_CAN_CheckTransmit, либо путём синхронного вызова NM_CAN_WaitTransmit.

**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 *nmcantesttrx*.

Для начала работы в режиме **FIFO** необходимо установить режим передачи вызовом NM_CAN_SetSendMode. Предварительно конфигурация CAN контроллера должна быть уже установлена. Доступ к регистрам CAN контроллера будет закрыт при включения режима **FIFO**.

Проверка количества сообщений в FIFO производится через вызовы NM_CAN_GetCountMsgInFifo и NM_CAN_GetCountMsgInHpFifo. Записать в FIFO можно (63-count) сообщений.

Передача сообщений высокоприоритетного FIFO (вызов NM_CAN_WriteDataToHpFifo) происходит в обход обычного FIFO. Пока все высокоприоритетные сообщения не будут отправлены, передача из обычного FIFO приостанавливается. Пример: *nmcantesttxfhpf_m*.

Для минимизации задержек при работе с шиной CAN может применяться следующая техника приостановки передачи: включается флаг TX_PAUSE (вызов NM_CAN_SetTxPause с value = 1),сообщения записываются в FIFO, в нужный момент передача разрешается (вызов NM_CAN_SetTxPause с value = 0). Пример: *nmcantesttxfifops_m*.

**Внимание!** Пустое FIFO (count=0) не означает, что передача всех сообщений завершена. Для подтверждения окончания передачи проверьте регистр CANx_FIFO_CONSTAT.RECENT_ID или CANx_FIFO_HP_CONSTAT.RECENT_ID – он должен быть равен msgID последнего вызова NM_CAN_WriteDataToFifo или NM_CAN_WriteDataToHpFifo. Подробнее см. 5.3.1 «Руководство по программированию xPCIe-CAN».

## 4.1 NM_CAN_SetSendMode

**Назначение:**
Устанавливает режим передачи канала.

**Действие:**
Включает режим FIFO. После этого должны использоваться ф-ции п.8.2 — 8.5.
Подробнее см. примеры реализации тестов.

**Примечание:**
- Доступ к регистрам CAN контроллера возможен только в режиме Native.

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t mode | режим (Native - 0; FIFO - 1) |

**Пример вызова:**

```
int fd, ret;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_SetSendMode (fd, 1, 0);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u data buffer %u was filled with data\n", 1, 0);
```

## 4.2 NM_CAN_GetCountMsgInFifo

**Назначение:**

Получает текущее кол-во сообщений в FIFO.

**Действие:**

-

**Примечание:**

-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| int* count | текущее кол-во сообщений в FIFO |

**Пример вызова:**

```
int fd, ret;
int count;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_GetCountMsgInFifo (fd, 1, & count);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u data buffer %u was filled with data\n", 1, 0);
```

## 4.3 NM_CAN_GetCountMsgInHpFifo

**Назначение:**

Получает текущее кол-во сообщений в HPFIFO.

**Действие:**

-

**Примечание:**

-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| int* count | текущее кол-во сообщений в HPFIFO |

**Пример вызова:**

```
int fd, ret;
int count;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_GetCountMsgInHpFifo (fd, 1, & count);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u data buffer %u was filled with data\n", 1, 0);
```

## 4.4 NM_CAN_WriteDataToFifo

**Назначение:**
Записать сообщение в FIFO на отправку.

**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 в драйвере общим семафором (индивидуальным для каждого канала и буфера) для возможности их корректного использования из параллельных процессов.

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t isExtId | признак расширенного сообщения |
| uint32_t sid | SID |
| uint32_t eid | EID |
| uint8_t* pdata | Массив данных |
| uint32_t datasize | Размер данных массива |
| uint8_t msgId | уникальный идентификатор сообщения |

**Пример вызова:**

```
int fd, ret;
int count;
uint8_t pdata[8];
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_WriteDataToFifo (fd, 1, 1, 0x1, 0x1,  pdata, 8, 0);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u data buffer %u was filled with data\n", 1, 0);
```

## 4.5 NM_CAN_WriteDataToHpFifo

**Назначение:**
Записать сообщение в HPFIFO на отправку.

**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 в драйвере общим семафором (индивидуальным для каждого канала и буфера) для возможности их корректного использования из параллельных процессов.

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| uint8_t isExtId | признак расширенного сообщения |
| uint32_t sid | SID |
| uint32_t eid | EID |
| uint8_t* pdata | Массив данных |
| uint32_t datasize | Размер данных массива |
| uint8_t msgId | уникальный идентификатор сообщения |

**Пример вызова:**

```
int fd, ret;
int count;
uint8_t pdata[8];
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_WriteDataToHpFifo (fd, 1, 1, 0x1, 0x1,  pdata, 8, 0);
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u data buffer %u was filled with data\n", 1, 0);
```

## 4.6 NM_CAN_WriteDataToSendBuf

**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 в драйвере 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| CAN_BUF **nbuf** | Номер буфера, в который будут записываться данные (допустимые значения – 0…2) |
| uint32_t **prio** | Приоритет буфера передачи (допустимые значения – 0…3; 0 – низший, 3 – наивысший) |
| uint32_t **sid** | Стандартный идентификатор (допустимые значения – 0…7FFH) |
| uint32_t **eid** | Расширенный идентификатор (допустимые значения – 0…3FFFFH и специальное значение FFFFFFFFH, которое означает, что расширенный идентификатор не используется) |
| uint8_t* **pdata** | Указатель на данные |
| uint32_t **datasize** | Размер передаваемых данных (допустимые значения – 0…8) |

**Пример вызова:**

```
int fd, ret;
uint8_t data[] = { 0xFF };
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_WriteDataToSendBuf(fd, 1, 0, 0, 0x7FF, 0x3FFFF, data, sizeof(data));
if (ret == -1)
  printf("error: %d (%s)\n", errno, strerror(errno));
else
printf("CAN%u data buffer %u was filled with data\n", 1, 0);
```

## 4.7 NM_CAN_SendData

**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 перейдёт 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 в драйвере 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| CAN_BUF **nbuf** | Номер буфера, в который будут записываться данные (допустимые значения – 0…2) |
| uint32_t **prio** | Приоритет буфера передачи (допустимые значения – 0…3; 0 – низший, 3 – наивысший) |
| uint32_t **sid** | Стандартный идентификатор (допустимые значения – 0…7FFH) |
| uint32_t **eid** | Расширенный идентификатор (допустимые значения – 0…3FFFFH и специальное значение FFFFFFFFH, которое означает, что расширенный идентификатор не используется) |
| uint8_t* **pdata** | Указатель на данные |
| uint32_t **datasize** | Размер передаваемых данных (допустимые значения – 0…8) |
| uint32_t **timeout** | Максимальное время ожидания отправки сообщения, в миллисекундах |
| TXBCTRL* **ptxb_ctrl** | Указатель на значение регистра **TXBn*CTRL** в случае истечения времени ожидания отправки сообщения |

**Пример вызова:**

```
int fd, ret;
uint8_t data[] = { 0xFF };
    uint8_t txb_ctrl;
    fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_SendData (fd, 1, 0, 0, 0x7FF, 0x3FFFF, data, sizeof(data), 100, &txb_ctrl);
if (ret == -1)
```

```
{
printf("error: %d (%s); TXB%uCTRL=%02XH\n", errno, strerror(errno), 0, txb_ctrl);
}
else
printf("CAN%u data buffer %u was filled with data\n", 1, 0);
```

## 4.8 NM_CAN_SendDataRightNow

**Назначение:**
Запуск асинхронной передачи сообщения.

**Действие:**
Бит TXREQ регистра **TXBn*CTRL**** устанавливается в единицу.
Все функции работы с буферами контроллера CAN защищены в драйвере 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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| CAN_BUF **nbuf** | Номер буфера, в который будут записываться данные (допустимые значения – 0…2) |

**Пример вызова:**

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

## 4.9 NM_CAN_CheckTransmit

**Назначение:**
Проверка правильности отправки сообщения.

**Действие:**
Данная функция читает регистр **TXBn*CTRL**** в параметр **ptxb_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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| CAN_BUF **nbuf** | Номер буфера, который будет проверен (допустимые значения – 0…2) |
| TXBCTRL* **ptxb_ctrl** | Указатель на значение регистра **TXBn*CTRL** |

**Пример вызова:**

```
int fd, ret;
uint8_t txb_ctrl
fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_CheckTransmit(fd, 1, 0, &txb_ctrl);
if (ret == -1)
{
printf("error: %d (%s); TXB%uCTRL=%02XH\n", errno, strerror(errno), 0, txb_ctrl);
}
else
printf("CAN%u data buffer %u was sended\n", 1, 0);
```

## 4.10 NM_CAN_WaitTransmit

**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 параметр **ptxb_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:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| CAN_BUF **nbuf** | Номер буфера, чья отправка будет ожидаться (допустимые значения – 0…2) |
| uint32_t **timeout** | Максимальное время ожидания отправки сообщения, в миллисекундах |
| TXBCTRL* **ptxb_ctrl** | Указатель на значение регистра **TXBn*CTRL** |

**Пример вызова:**

```
int fd, ret;
uint8_t txb_ctrl;
fd = open("/dev/can_dev_0", O_RDWR);
ret = NM_CAN_WaitTansmit(fd, 1, 0, 100);
if (ret == -1)
{
printf("error: %d (%s); TXB%uCTRL=%02XH\n", errno, strerror(errno), 0, txb_ctrl);
}
else
printf("CAN%u data buffer %u was sended\n", 1, 0);
```

## 4.11 NM_CAN_EndTransmit

**Назначение:**
Снятие запроса отправки сообщения.

**Действие:**
Данная функция записывать ноль в бит TXREQ регистра **TXBn*CTRL****.

* n – номер буфера.
** См. Раздел 6.7.1 документа "*Руководство по программированию модуля "xPCIe-CAN*".

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| CAN_BUF **nbuf** | Номер буфера, с которого будет снят запрос на отправку (допустимые значения – 0…2) |

**Пример вызова:**

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

## 4.12 NM_CAN_ABAT

**Назначение:**
Остановка всех активных передач.

**Действие:**
Функция устанавливает бит ABAT регистра **CAN_CTRL*** (адрес Fh) в соответствии со значением поля **mode**.

* См. Раздел 6.3.1 документа "*Руководство по программированию модуля "xPCIe-CAN*".

**Примечание:**
-

**Аргументы функции:**

| Аргумент | Описание |
|---|---|
| int **fd** | Дескриптор устройства, полученный от функции open() |
| uint32_t **channel** | Номер канала (допустимые значения – 1…2) |
| int **abat** | 0 – снятие запроса на остановку всех активных передач.<br>Не 0 - запрос на остановку всех активных передач |

**Пример вызова:**

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

# 5 Обновление библиотеки

| Версия библиотеки | Дата | Изменение |
|---|---|---|
| 5.0 | 10.09.2018 | - Библиотека создана |
| 7.0 | 29.07.2020 | - Добавлен режим FIFO |

# 6 Обновление руководства.

| Версия документа | Дата | Изменение |
|---|---|---|
| 5.0 | 10.09.2018 | - Документ создан |
| 7.0 | 29.07.2020 | - Добавлен режим FIFO |