ДРАЙВЕР КОНТРОЛЛЕРА SWIC ДЛЯ МИКРОСХЕМЫ 1892ВМ14Я. РУКОВОДСТВО ПРОГРАММИСТА

Версия

11.03.2022

ОГЛАВЛЕНИЕ

1 О документе **3**

2 Общее описание **4**

3 Стандартные системные вызовы **5**
3.1 Функция `open()` . . . 5
3.2 Функция `write()` . . . 5
3.3 Функция `read()` . . . 5

4 Описание IOCTL **7**
4.1 SWICIOC_SET_LINK . . . 7
4.2 SWICIOC_GET_LINK_STATE . . . 7
4.3 SWICIOC_GET_SPEED . . . 8
4.4 SWICIOC_SET_TX_SPEED . . . 8
4.5 SWICIOC_SET_MTU . . . 9
4.6 SWICIOC_GET_MTU . . . 9
4.7 SWICIOC_FLUSH . . . 10
4.8 SWICIOC_GET_STATS . . . 10
4.9 SWICIOC_RESET_STATS . . . 11

1. О ДОКУМЕНТЕ

Документ содержит описание Linux-драйвера для контроллера SWIC микросхемы 1892ВМ14Я.

2. ОБЩЕЕ ОПИСАНИЕ

Драйвер *elvees-swic* предоставляет поддержку работы на канальном уровне SpaceWire:

1. Установка/разрыв соединения.
2. Получение статуса соединения.
3. Приём/передача пакетов SpaceWire.
4. Ограниченная поддержка дуплексного режима.
5. Получение скорости приёма.
6. Установка/получение скорости передачи.
7. Установка/получение размера пакета.
8. Сбор статистики передачи/приема данных.

Драйвер реализует интерфейс символьного устройства Linux.

Ограничения драйвера:

1. Максимальный размер пакета (MTU) равен 1 МиБ (спецификация SpaceWire не регламентирует максимальный размер пакета).
2. Не поддерживается передача данных из/в пространство пользователя без копирования (zero-copy).
3. Не поддерживается передача коммуникационного пакета.
4. Не поддерживается работа с управляющими кодами.
5. Из-за ограничения rf#7224 возможна некорректная работа драйвера если к контроллеру DDRMC1 подключено более 1 ГиБ памяти.
6. При работе в дуплексном режиме прерывание пользователем приема или передачи данных приводит к завершению и приема и передачи данных.

3. СТАНДАРТНЫЕ СИСТЕМНЫЕ ВЫЗОВЫ

3.1 Функция `open()`

Данная функция принимает в качестве аргумента имя устройства `/dev/spacewireX`, которое необходимо открыть и возвращает в пространство пользователя файловый дескриптор устройства.

При открытии устройства статистика не сбрасывается. Для сброса статистики необходимо вызвать *SWICIOC_RESET_STATS*.

3.2 Функция `write()`

При вызове функции `write()` драйвер выполняет:

1. Проверку на наличие соединения. Если соединение неактивно, вызов функции завершается с ошибкой `ENOLINK`.
2. Передачу входных пользовательских данных в канал SpaceWire. Если размер пользовательских данных больше установленного размера пакета (MTU), драйвер автоматически разбивает данные на порции размером не больше MTU. Каждая порция данных отправляется отдельным пакетом.

Если во время передачи данных произошел разрыв соединения, `write()` прекращает работу и возвращает количество байт, которое было передано на момент разрыва соединения.

Функция `write()` потокобезопасна.

Предупреждение: Из-за ограничения контроллера SWIC rf#11325 при завершении функции `write()` драйвер не гарантирует, что все пользовательские данные были отправлены. В результате при разрыве соединения после завершения `write()` возможна потеря данных.

3.3 Функция `read()`

При вызове функции `read()` драйвер выполняет:

1. Проверку на наличие соединения. Если соединение неактивно, вызов функции завершается с ошибкой `ENOLINK`.
2. Проверку размера пользовательского буфера. Если он меньше максимального размера пакета, функция завершается с ошибкой `ENOBUFS`.
3. Приём данных одного пакета из канала SpaceWire в пользовательский буфер.

Если во время приема данных произошел разрыв соединения, `read()` прекращает работу и возвращает количество байт, которое было принято на момент разрыва соединения.

Функция `read()` потокобезопасна.

Предупреждение: При прерывании приема данных пользователем, из-за особенности реализации контроллера, необходимо очистить FIFO буфера перед началом следующего приема вызовом *SWICIOC_FLUSH*.

4. ОПИСАНИЕ IOCTL

Перед вызовом функций `ioctl()` необходимо получить файловый дескриптор устройства, см. *Функция open()*.

4.1 SWICIOC_SET_LINK

Установка/сброс соединения:

```
int ioctl(int fd, SWICIOC_SET_LINK, unsigned int link_status);
```

Данный вызов применим к файловому дескриптору устройства.

Аргументы:

- `link_status` — тип выполняемой операции: 0 — сброс соединения, 1 — установка соединения.

При сбросе соединения драйвер выполняет:

1. Сброс link-интерфейса контроллера.
2. Остановку RX DATA DMA.
3. Очистку всех FIFO.

При установке соединения драйвер выполняет:

1. Сброс соединения.
2. Включение PLL TX SWIC, LVDS SWIC.
3. Установку коэффициента для подсчёта таймаутов установки соединения.
4. Запуск TX DATA DMA в режиме самоинициализации.

Данный вызов возвращает управление, не дожидаясь установки соединения. Для чтения статуса соединения необходимо вызвать *SWICIOC_GET_LINK_STATE*.

4.2 SWICIOC_GET_LINK_STATE

Получение статуса соединения:

```
enum swic_link_state {
    LINK_ERROR_RESET,
    LINK_ERROR_WAIT,
    LINK_READY,
    LINK_STARTED,
    LINK_CONNECTING,
    LINK_RUN
```

(continues on next page)

(продолжение с предыдущей страницы)

```
};

int ioctl(fd, SWICIOC_GET_LINK_STATE, enum swic_link_state *link_status);
```

Данный вызов применим к файловому дескриптору устройства.

Статусы соединения:

- `LINK_ERROR_RESET` — соединение отсутствует.
- `LINK_ERROR_WAIT` — ошибка ожидания соединения.
- `LINK_READY` — узлы готовы к соединению.
- `LINK_STARTED` — установка соединения началась.
- `LINK_CONNECTING` — соединение устанавливается.
- `LINK_RUN` — соединение установлено.

При вызове драйвер возвращает в пространство пользователя один из статусов соединения.

4.3 SWICIOC_GET_SPEED

Получение скорости приёма и передачи:

```
struct elvees_swic_speed {
    unsigned int rx;
    unsigned int tx;
};

int ioctl(fd, SWICIOC_GET_SPEED, struct elvees_swic_speed *speed);
```

Данный вызов применим к файловому дескриптору устройства.

При вызове драйвер возвращает в пространство пользователя значения скоростей (в Кбит/с) приёма и передачи.

4.4 SWICIOC_SET_TX_SPEED

Установка скорости соединения:

```
enum swic_tx_speed {
    TX_SPEED_2P4,
    TX_SPEED_4P8,
    TX_SPEED_72 ,
    TX_SPEED_120,
    TX_SPEED_168,
    TX_SPEED_216,
    TX_SPEED_264,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    TX_SPEED_312,
    TX_SPEED_360,
    TX_SPEED_408
};

int ioctl(fd, SWICIOC_SET_TX_SPEED, enum swic_tx_speed tx_speed);
```

Данный вызов применим к файловому дескриптору устройства.

Возможные скорости передачи:

- `TX_SPEED_2P4` — 2.4 Мбит/с,
- `TX_SPEED_4P8` — 4.8 Мбит/с,
- `TX_SPEED_72` — 72 Мбит/с,
- `TX_SPEED_120` — 120 Мбит/с,
- `TX_SPEED_168` — 168 Мбит/с,
- `TX_SPEED_216` — 216 Мбит/с,
- `TX_SPEED_264` — 264 Мбит/с,
- `TX_SPEED_312` — 312 Мбит/с,
- `TX_SPEED_360` — 360 Мбит/с,
- `TX_SPEED_408` — 408 Мбит/с.

При вызове драйвер устанавливает скорость передачи, равную значению входного аргумента `tx_speed`.

Если значение входного аргумента `ioctl()` не входит в вышеперечисленный набор возможных скоростей, драйвер возвращает ошибку `EINVAL`.

4.5 SWICIOC_SET_MTU

Установка размера пакета:

```
int ioctl(fd, SWICIOC_SET_MTU, unsigned long mtu);
```

Данный вызов применим к файловому дескриптору устройства.

При вызове драйвер устанавливает размер пакета, равный значению входного аргумента `mtu`.

Если значение входного аргумента `ioctl()` превышает определённый в драйвере максимальный размер пакета, драйвер возвращает ошибку `EINVAL`.

4.6 SWICIOC_GET_MTU

Получение размера пакета:

```
int ioctl(fd, SWICIOC_GET_MTU, unsigned long *mtu);
```

Данный вызов применим к файловому дескриптору устройства.

При вызове драйвер возвращает в пространство пользователя значение размера пакета.

Если ранее размер пакета не был установлен вызовом *SWICIOC_SET_MTU*, то вызов вернет размер пакета, определенный в драйвере по умолчанию (16 Кбайт).

4.7 SWICIOC_FLUSH

Сброс всех RX, TX FIFO контроллера:

```
int ioctl(fd, SWICIOC_FLUSH, 0);
```

Данный вызов применим к файловому дескриптору устройства.

При сбросе контроллера драйвер выполняет:

1. Остановку запущенных каналов DMA.
2. Очистку всех FIFO.
3. Сброс счётчика принятых контроллером пакетов.

Предупреждение: При вызове *SWICIOC_FLUSH* прерывается прием/передача данных. Во избежание потери данных `ioctl()` следует использовать после завершения выполнения всех `read()`/`write()` функций.

4.8 SWICIOC_GET_STATS

Получение статистики посчитанную драйвером:

```
struct elvees_swic_stats {
    __u64 tx_data_bytes;
    __u64 rx_data_bytes;
    __u32 tx_packets;
    __u32 rx_eop_packets;
    __u32 rx_eep_packets;
    __u32 dc_err;
    __u32 parity_err;
    __u32 escape_err;
    __u32 credit_err;
};

int ioctl(fd, SWICIOC_GET_STATS, struct elvees_swic_stats *stats);
```

Данный вызов применим к файловому дескриптору устройства.

Значение переменных:

- `tx_data_bytes` — количество переданных по DMA в контроллер байт,
- `rx_data_bytes` — количество полученных по DMA из контроллера байт,
- `tx_packets` — количество переданных дескрипторов пакетов,
- `rx_eop_packets` — количество полученных дескрипторов пакетов с признаком EOP,
- `rx_eep_packets` — количество полученных дескрипторов пакетов с признаком EEP,
- `dc_err` — количество Disconnect error,
- `parity_err` — количество Parity error,
- `escape_err` — количество Escape error,
- `credit_err` — количество Credit error,

4.9 SWICIOC_RESET_STATS

Сброс счётчиков статистики в драйвере:

```
int ioctl(fd, SWICIOC_RESET_STATS, 0);
```

Данный вызов применим к файловому дескриптору устройства.