

**Набор разработчика драйверов
(DDK)
для адаптеров семейства
Кроникс-Сигма**

Содержание

Список функций	4
Файлы.....	6
Структуры данных	6
Структура cx_board_t.....	6
Структура cx_chan_t.....	7
Инициализация	8
Функция cx_find()	8
Функция cx_open_board()	10
Функция cx_close_board().....	10
Функция cx_probe_board().....	10
Функция cx_init()	11
Функция cx_setup_board()	11
Функция cx_probe_irq()	12
Функция ddk_int_alloc()	13
Функция ddk_int_restore	13
Обработка событий	14
Функция cx_register_transmit().....	14
Функция cx_register_receive()	15
Функция cx_register_modem()	15
Функция cx_register_error()	16
Пуск канала	16
Функция cx_start_chan()	16
Функция cx_enable_receive()	17
Функция cx_enable_transmit()	17
Функция cx_receive_enabled()	17
Функция cx_transmit_enabled()	17
Передача данных	18
Функция cx_send_packet()	18
Функция cx_buf_free()	19
Функция cx_int_handler()	19
Функция cx_led()	19
Установка и опрос параметров канала	20
Функция cx_set_baud()	20
Функция cx_get_baud()	20
Функция cx_set_mode()	20

Функция cx_set_nrzi()	20
Функция cx_get_nrzi()	21
Функция cx_set_dpll()	21
Функция cx_get_dpll()	21
Функция cx_set_loop()	21
Функция cx_get_loop()	21
Асинхронный режим	22
Функция cx_set_async_param()	22
Функция cx_flush_transmit()	22
Функция cx_transmitter_ctl()	22
Функция cx_xflow_ctl()	23
Функция cx_send_break()	23
Модемные сигналы	23
Функция cx_set_dtr()	23
Функция cx_set_rts()	23
Функция cx_get_dsr()	24
Функция cx_get_cts()	24
Функция cx_get_cd()	24
Пример	25

Набор разработчика для адаптеров семейства Сигма представляет собой библиотеку функций написанных на языке Си и предназначен для создания программ и драйверов, непосредственно обращающихся к адаптеру без необходимости программирования низкоуровневого интерфейса. Основной задачей при создании библиотеки было создание простого интерфейса (API) к основным возможностям адаптеров, таким как изменение текущей конфигурации, передача и приём HDLC пакетов, контроль состояния каналов. Поддерживается работа одновременно с несколькими адаптерами (до 3-х). Набор тестировался со следующими компиляторами: Borland Turbo C/C++ 1.0 (DOS), GNU C (Linux и FreeBSD). Примеры в тексте даны для операционной системы DOS.

Процесс работы с адаптером состоит из следующих этапов:

- Инициализация аппаратуры и структуры данных адаптера.
- Установка обработчика прерывания.
- Регистрация обработчиков событий.
- Установка конфигурации адаптера.
- Установка параметров каналов и их запуск.
- Процесс приёма и передачи данных.
- После окончания работы — обязательно — сброс платы и восстановление вектора прерывания.

Поскольку набор разработчика реализован независимо от операционной системы, выделение памяти и установку обработчика прерывания должен обеспечить пользователь. Все функции DDK должны вызываться при закрытых прерываниях.

Список функций

Инициализация:

<i>cx_find()</i>	— поиск установленных адаптеров
<i>cx_open_board()</i>	— распознавание и инициализация адаптера
<i>cx_close_board()</i>	— сброс адаптера
<i>cx_probe_board()</i>	— распознавание адаптера
<i>cx_init()</i>	— инициализация структуры данных адаптера
<i>cx_setup_board()</i>	— сброс адаптера и загрузка firmware
<i>cx_probe_irq()</i>	— проверка прерывания
<i>ddk_int_alloc()</i>	— установка обработчика прерывания (для DOS)
<i>ddk_int_restore()</i>	— восстановление вектора прерывания (для DOS)

Установка обработчиков:

<i>cx_register_transmit()</i>	— установка обработчика передачи пакета
<i>cx_register_receive()</i>	— установка обработчика приёма пакета
<i>cx_register_modem()</i>	— установка обработчика изменения модемных сигналов
<i>cx_register_error()</i>	— установка обработчика ошибок

Пуск канала:

<i>cx_start_chan()</i>	— запуск канала
<i>cx_enable_receive()</i>	— выключение/включение приёмника
<i>cx_enable_transmit()</i>	— выключение/включение передатчика
<i>cx_receive_enabled()</i>	— определение состояния включения приёмника
<i>cx_transmit_enabled()</i>	— определение состояния включения передатчика

Передача данных:

<i>cx_send_packet()</i>	— передача пакета
<i>cx_buf_free()</i>	— опрос наличия свободных буферов передатчика
<i>cx_int_handler()</i>	— обработка прерывания
<i>cx_led()</i>	— управление светодиодом

Установка и опрос параметров канала:

<i>cx_set_baud()</i>	— выбор частоты генератора синхросигнала
<i>cx_set_mode()</i>	— выбор режима передачи данных
<i>cx_set_nrzi()</i>	— выбор режима кодирования
<i>cx_set_dpll()</i>	— выбор режима синхронизации
<i>cx_set_loop()</i>	— выключение/включение локального шлейфа
<i>cx_get_baud()</i>	— определение частоты генератора синхросигнала
<i>cx_get_nrzi()</i>	— определение текущего режима кодирования
<i>cx_get_dpll()</i>	— определение текущего режима синхронизации
<i>cx_get_loop()</i>	— опрос состояния включения локального шлейфа

Модемные сигналы:

<i>cx_set_dtr()</i>	— установка сигнала DTR
<i>cx_set_rts()</i>	— установка сигнала RTS
<i>cx_get_dsr()</i>	— опрос сигнала DSR
<i>cx_get_cts()</i>	— опрос сигнала CTS
<i>cx_get_cd()</i>	— опрос сигнала CD

Асинхронный режим передачи данных:

<i>cx_set_async_param()</i>	— установка параметров асинхронного режима
<i>cx_flush_transmit()</i>	— очистка буфера передатчика
<i>cx_transmit_ctl()</i>	— включение/выключение передатчика
<i>cx_xflow_ctl()</i>	— передача символов управления потоком
<i>cx_send_break()</i>	— передача сигнала break

Структуры данных:

<i>cx_board_t</i>	— данные адаптера
<i>cx_chan_t</i>	— данные канала
<i>cx_buf_t</i>	— буфер обмена канала

Файлы

Для создания программ необходимы следующие файлы:

<i>cxddk.h</i>	— Прототипы функций библиотеки, типы данных и константы. Включать в программу нужно только этот файл.
<i>cxreg.h</i>	— Описания интерфейса коммуникационного контроллера, используемого в адаптере.
<i>cronyxfw.h</i>	— Структуры данных <i>firmware</i> .
<i>csigmafw.h</i>	— Загружаемое аппаратное обеспечение адаптера (<i>firmware</i>).
<i>tachdep.h</i>	— Машинно-зависимые определения.
<i>cxddk.c</i>	— Си-код процедур пакета.
<i>csigma.c</i>	— Код низкоуровневых процедур, работающих с аппаратурой.
<i>ddkdos.c</i>	— Функции для работы с прерываниями (только для DOS).

В каталоге *example* находится пример:

<i>test.c</i>	— Программа-пример использования DDK.
---------------	---------------------------------------

Структуры данных

Поля структур данных DDK следует считать доступными только на чтение. Для изменения режимов работы устройства рекомендуется применять вызовы функций.

Структура *cx_board_t*

```
typedef struct {
    unsigned char type;
    char name[16];
    unsigned char num;
    unsigned char irq;
    unsigned char dma;
    unsigned short port;
    cx_chan_t chan[NCHAN];
    ...
} cx_board_t;
```

Каждому адаптеру соответствует структура данных типа *cx_board_t*, содержащая информацию о его текущем состоянии. Перед вызовом функции инициализации *cx_open_board* следует выделить память под структуру *cx_board_t*, обнулить ее и передать в качестве аргумента.

<i>unsigned char type;</i>	— тип адаптера: <i>B_SIGMA_XXX</i> — Модели Сигма-4xx, -8xx, -100, -500 <i>B_SIGMA_2X</i> — Сигма-22 <i>B_SIGMA_800</i> — Сигма-800
<i>char name[16];</i>	— название модели адаптера в текстовом виде (см. также поле <i>type</i>).
<i>unsigned char num;</i>	— идентификационный номер адаптера. Для различения

адаптеров при инициализации им ставятся в соответствие идентификационные номера, числа в диапазоне 0...NBRD-1. NBRD=3 — максимальное количество адаптеров. Номер задаётся при инициализации адаптера.

unsigned char irq; — номер линии запроса прерывания.

unsigned char dma; — номер линии запроса ПДП.

unsigned short port; — номер базового порта ввода/вывода адаптера. Адаптер занимает NPORT=32 последовательных адресов.

cx_chan_t chan[NCHAN]; — Структуры данных каналов (NCHAN=16).

Структура *cx_chan_t*

```
typedef struct {
    unsigned char type;
    unsigned char num;
    cx_board_t *board;
    unsigned char mode;
    unsigned char dtr;
    unsigned char rts;
    void *sys;
    int debug;
    unsigned long rintr;
    unsigned long tintr;
    unsigned long mintr;
    unsigned long ibytes;
    unsigned long ipkts;
    unsigned long ierrs;
    unsigned long obytes;
    unsigned long opkts;
    unsigned long oerrs;
    ...
} cx_chan_t;
```

unsigned char type; — тип канала:

- T_NONE — канал отсутствует в данной модели адаптера
- T_ASYNC — асинхронный RS-232
- T_SYNC_RS232 — синхронный RS-232
- T_SYNC_V35 — синхронный V.35
- T_SYNC_RS449 — асинхронный RS-449
- T_UNIV_RS232 — синхронно-асинхронный RS-232
- T_UNIV_RS449 — синхронно-асинхронный переключаемый RS-232/RS-449
- T_UNIV_V35 — синхронно-асинхронный переключаемый RS-232/V.35
- T_UNIV — синхронно-асинхронный переключаемый, интерфейс неизвестен

unsigned char num; — идентификационный номер канала, 0...NCHAN-1.

*cx_board_t *board;* — указатель на структуру данных адаптера.

unsigned char mode; — режим передачи данных:

- M_ASYNC — асинхронный режим
- M_HDLC — синхронный режим

unsigned char dtr; — состояние сигнала DTR: 1, если сигнал активен, иначе — 0.

<i>unsigned char rts;</i>	— состояние сигнала RTS: 1, если сигнал активен, иначе — 0
<i>void *sys;</i>	— указатель общего назначения, может использоваться разработчиком драйвера для своих целей
<i>int debug;</i>	— режим отладочной печати, может использоваться разработчиком драйвера для своих целей
<i>unsigned long rintr;</i>	— счетчик прерываний приемника
<i>unsigned long tintr;</i>	— счетчик прерываний передатчика
<i>unsigned long mintr;</i>	— счетчик модемных прерываний
<i>unsigned long ibytes;</i>	— счетчик принятых байтов
<i>unsigned long ipkts;</i>	— счетчик принятых пакетов
<i>unsigned long ierr;</i>	— счетчик ошибок приемника
<i>unsigned long obytes;</i>	— счетчик переданных байтов
<i>unsigned long opkts;</i>	— счетчик переданных пакетов
<i>unsigned long oerrs;</i>	— счетчик ошибок передатчика

Инициализация

Функция *cx_find()*

```
int cx_find (unsigned short *porttab)
```

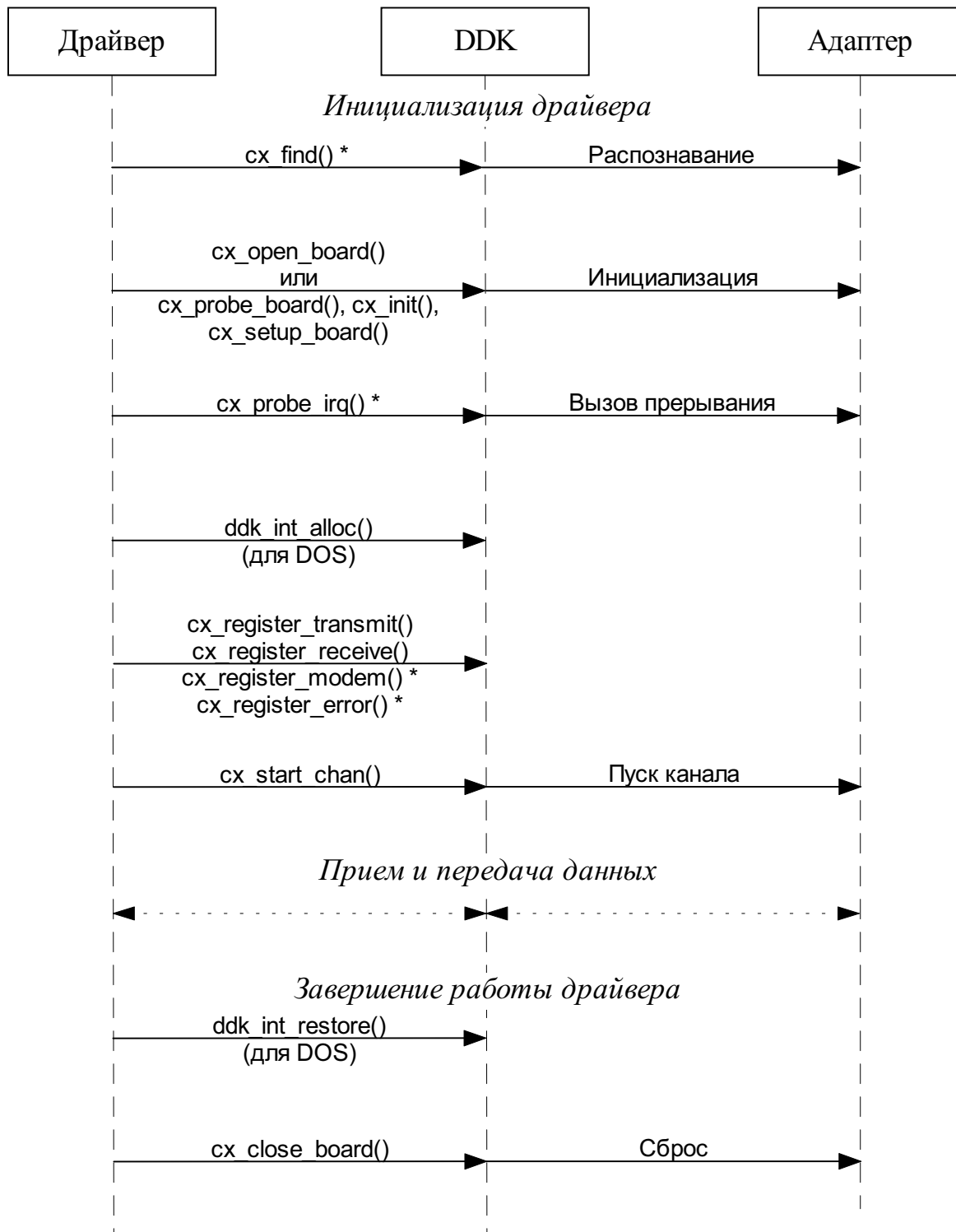
Производит поиск установленных адаптеров. Возвращает количество найденных адаптеров.

porttab[3] — массив, в котором возвращаются адреса ввода/вывода найденных адаптеров.

Пример:

```
main()
{
    cx_board_t b;
    unsigned short porttab[NBRD];

    disable();
    if (! cx_find (porttab)) {
        enable();
        printf ("No adapters found\n");
        exit (-1);
    }
    cx_open_board (&b, 0, porttab[0], IRQ, DRQ);
    enable();
    ...
}
```

* — необязательные вызовы

Функция `cx_open_board()`

```
int cx_open_board (cx_board_t *b, int num,
                  unsigned short port, int irq, int dma)
```

Производит сброс и инициализацию платы, загрузку `firmware`, инициализирует параметры состояния адаптера. Заменяет собой функции `cx_probe_board()`, `cx_init()` и `cx_setup_board()`. Возвращает `1` в случае успешного окончания, иначе — `0`.

- b* — Структура данных адаптера. Перед вызовом ее следует обнулить. После вызова содержит данные адаптера: тип, имя, режимы и пр.
- num* — Идентификатор платы (`0...NBRD-1`, `NBRD=3`). Используется для различения нескольких установленных адаптеров.
- port* — Базовый порт ввода-вывода адаптера.
- irq* — Номер запроса прерывания.
- dma* — Номер канала прямого доступа в память (ПДП).

Адрес базового порта устанавливается переключателями на плате. IRQ и DMA выбираются и устанавливаются программно. Поиск установленных плат может быть произведен с помощью функции `cx_find()`. Проверку работоспособности прерывания можно произвести с помощью функции `cx_probe_irq()`.

Пример:

```
main()
{
    cx_board_t b;

    disable();
    if (! cx_open_board (&b, 0, 0x240, 5, 7)) {
        enable();
        printf ("Initialization error\n");
        exit (-1);
    }
    ...
}
```

Функция `cx_close_board()`

```
void cx_close_board (cx_board_t *b)
```

Производит сброс адаптера и переводит его в неактивное состояние. Должна вызываться при окончании работы с адаптером.

Функция `cx_probe_board()`

```
int cx_probe_board (unsigned short port, int irq, int dma)
```

Проверяет наличие адаптера на указанном адресе ввода/вывода, а также корректность номеров прерывания и ПДП. Возвращает `1` в случае успешного окончания, иначе — `0`.

- port* — Базовый порт ввода-вывода адаптера. Допустимые значения: `0x200`, `0x220`, `0x240`, `0x260`, `0x280`, `0x2a0`, `0x2c0`, `0x2e0`, `0x300`, `0x320`, `0x340`, `0x360`, `0x380`, `0x3a0`, `0x3c0`, `0x3e0`.
- irq* — Номер запроса прерывания, либо `-1`. Допустимые значения: `3`,

5, 7, 10, 11, 12, 15.

dma — Номер канала прямого доступа в память (ПДП), либо -1.
Допустимые значения: 5, 6, 7.

Адрес базового порта устанавливается переключателями на плате. IRQ и DMA выбираются и устанавливаются программно. Поиск установленных плат может быть произведен с помощью функции *cx_find()*. Проверку работоспособности прерывания можно произвести с помощью функции *cx_probe_irq()*.

Пример:

```
#include "csigmafw.h"

main()
{
    cx_board_t b;

    disable();
    if (! cx_probe_board (PORT, IRQ, DRQ)) {
        enable();
        printf ("Adapter not found\n");
        exit (-1);
    }
    cx_init (&b, 0, PORT, IRQ, DRQ);
    if (! cx_setup_board (&b, csigma_fw_data, csigma_fw_len,
        csigma_fw_tvec)) {
        enable();
        printf ("Loading firmware failed\n");
        exit (-1);
    }
    ...
}
```

Функция *cx_init()*

```
void cx_init (cx_board_t *b, int num,
    unsigned short port, int irq, int dma)
```

Производит инициализацию структуры данных адаптера.

b — Структура данных адаптера. Перед вызовом ее следует обнулить. После вызова содержит данные адаптера: тип, имя, режимы и пр.

num — Идентификатор платы (0...NBRD-1, NBRD=3). Используется для различения нескольких установленных адаптеров.

port — Базовый порт ввода-вывода адаптера.

irq — Номер запроса прерывания.

dma — Номер канала прямого доступа в память (ПДП).

Пример: см. *cx_probe_board()*.

Функция *cx_setup_board()*

```
int cx_setup_board (cx_board_t *b, const unsigned char *firmware,
    long bits, const cr_dat_tst_t *tst)
```

Производит аппаратный сброс адаптера в пассивное состояние и загрузку аппаратного

обеспечения (firmware).

- b* — Структура данных адаптера.
- firmware* — Массив данных firmware, или NULL если загрузка не требуется. Данные для загрузки находятся в файле *csigmafw.h*.
- bits* — Объем firmware в битах.
- tst* — Вектор контроля загрузки, или NULL если загрузка не требуется.

Пример: см. *cx_probe_board()*.

Функция *cx_probe_irq()*

```
int cx_probe_irq (cx_board_t *b, int irq)
```

Служит для проверки работоспособности прерывания и отсутствия конфликтов с другими устройствами компьютера. При *irq*>0 — вызывает искусственное прерывание от адаптера, устанавливает IRQ в активное состояние. При *irq*<0 — отменяет прерывание от адаптера, устанавливает IRQ в пассивное состояние. При *irq*==0 — освобождает линию IRQ (третье состояние). Возвращает маску активных прерываний на момент вызова, непосредственно до выполнения действий с адаптером.

- b* — Структура данных адаптера.
- irq* — Номер запроса прерывания.

Пример:

```
main()
{
    cx_board_t b;
    int mask, busy;

    disable();
    if (! cx_open_board (&b, 0, PORT, IRQ, DRQ)) {
        enable();
        printf ("Initialization error\n");
        exit (-1);
    }
    /* Опрос маски активных (занятых) прерываний.
     * Активизация прерывания от адаптера. */
    busy = cx_probe_irq (b, IRQ);

    /* Опрос маски активных прерываний.
     * Деактивизация прерывания адаптера. */
    mask = cx_probe_irq (b, -IRQ);

    /* Проверка маски прерывания. */
    if ((mask & ~busy) != 1 << IRQ) {
        /* Освобождение линии прерывания. */
        cx_probe_irq (b, 0);
        enable();
        printf ("Interrupt failure\n");
        exit (-1);
    }
    ...
}
```

Функция *ddk_int_alloc()*

```
int ddk_int_alloc (int irq, void (*func)(), void *arg)
```

Устанавливает обработчик прерывания (для DOS). При возникновении аппаратного прерывания с номером *irq* будет вызвана функция *func* с аргументом *arg*. Возвращает 1 в случае успешного окончания. 0 возвращается, если даны неправильные аргументы или вектор прерывания занят.

irq — Номер линии запроса прерывания.
func — Адрес обработчика прерывания.
arg — Аргумент, который будет передаваться обработчику.

Пример:

```
main()
{
    cx_board_t b;

    disable();
    if (! cx_open_board (&b, 0, PORT, IRQ, DRQ)) {
        enable();
        printf ("Initialization error! \n");
        exit (-1);
    }
    ddk_int_alloc (IRQ, &cx_int_handler, b);
    ...
}
```

Функция *ddk_int_restore*

```
void ddk_int_restore (int irq)
```

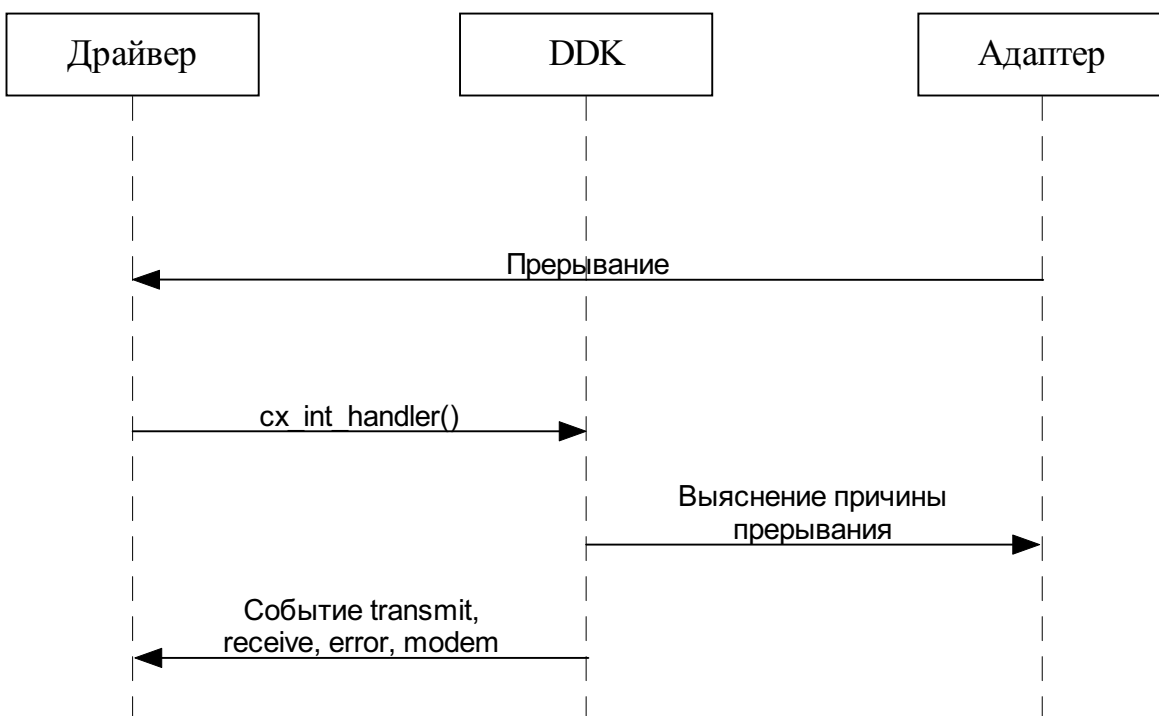
Восстанавливает вектор прерывания.

irq — Номер линии запроса прерывания.

Обработка событий

Для обработки аппаратных событий применяется механизм обработчиков (callbacks). При возникновении аппаратного прерывания вызывается функция пользователя, предварительно зарегистрированная функциями `cx_register_xxx()`. Для каждого канала может быть зарегистрирован свой обработчик. Различаются четыре вида событий:

- Прерывание по приему пакета.
- Прерывание по завершению передачи пакета.
- Прерывание по ошибке приема или передачи.
- Прерывание по изменению сигнала несущей (DCD).



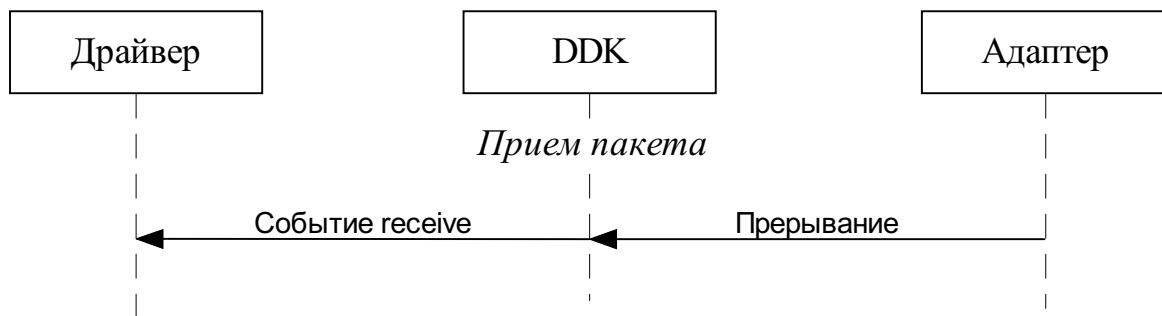
Функция `cx_register_transmit()`

```
void cx_register_transmit (cx_chan_t *c,
    void (*func) (cx_chan_t *c, void *attachment, int len))
```

Функция-обработчик вызывается при успешном завершении передачи пакета.

Аргументы, передаваемые обработчику:

- | | |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>c</code> | — Указатель на структуру канала |
| <code>attachment</code> | — Аргумент соответствующего вызова функции <code>cx_send_packet()</code> , может использоваться драйвером для передачи ссылки на системно-зависимую структуру данных, относящуюся к пакету. |
| <code>len</code> | — Длина пакета в байтах |



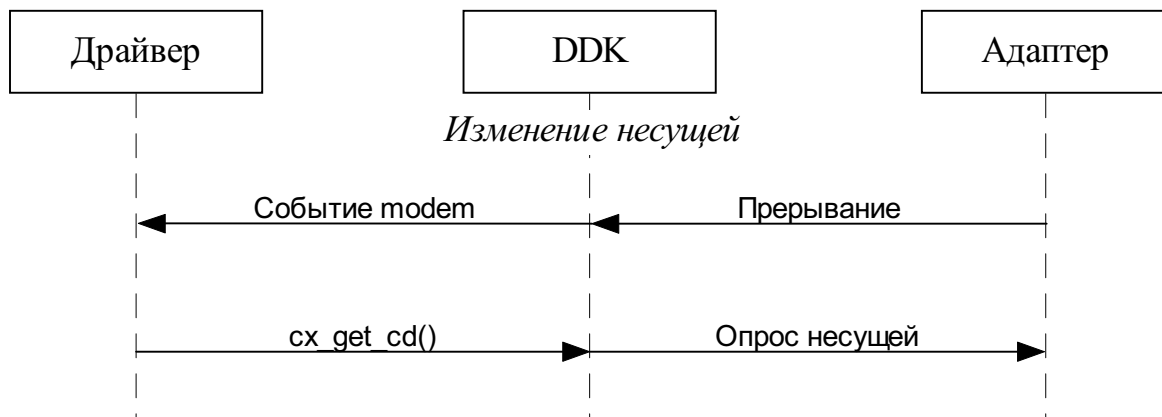
Функция `cx_register_receive()`

```
void cx_register_receive (cx_chan_t *c,
    void (*func) (cx_chan_t *c, char *data, int len))
```

Функция-обработчик вызывается при успешном приеме пакета. В случае возникновения ошибки приема вызывается обработчик события ошибки.

Аргументы, передаваемые обработчику:

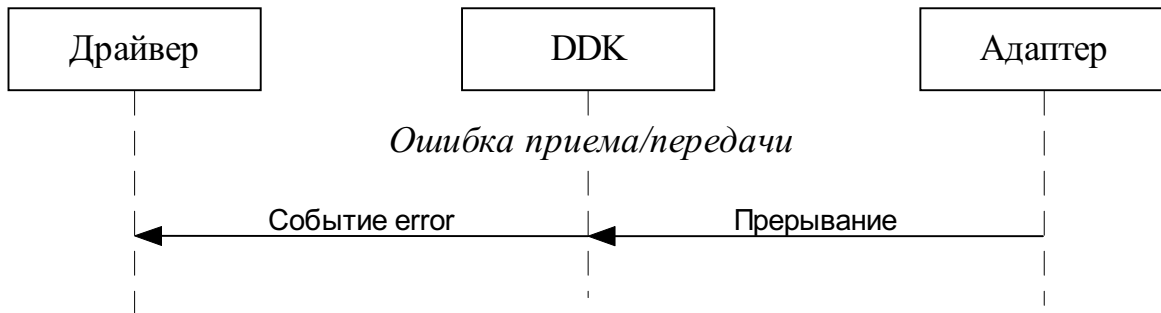
c — Указатель на структуру канала
data — Указатель на буфер данных
len — Длина пакета в байтах



Функция `cx_register_modem()`

```
void cx_register_modem (cx_chan_t *c, void (*func) (cx_chan_t *c))
```

Функция-обработчик вызывается при изменении сигнала несущей (DCD). Опросить состояние несущей можно функцией `cx_get_cd()`.



Функция `cx_register_error()`

```
void cx_register_error (cx_chan_t *c,
                      void (*func) (cx_chan_t *c, int data))
```

Функция-обработчик вызывается при обнаружении ошибки.

Аргументы, передаваемые обработчику:

c — Указатель на структуру канала
data — Код ошибки:

- `CX_FRAME` — Ошибка кадра
- `CX_CRC` — Ошибка контрольной суммы
- `CX_UNDERRUN` — Опустошение FIFO передатчика
- `CX_OVERRUN` — Переполнение FIFO приёмника
- `CX_OVERFLOW` — Переполнение буфера приемника
- `CX_BREAK` — Сигнал break (асинхронный режим)

Пуск канала

Функция `cx_start_chan()`

```
void cx_start_chan (cx_chan_t *c, cx_buf_t *buf, unsigned long phys)
```

Запускает приемник и передатчик канала, сбрасывает сигналы DTR и RTS в 0.

c — Указатель на переменную состояния канала.
buf — Указатель на область памяти, предназначенную для буферов приёма-передачи.
phys — Физический адрес области памяти *buf*.

Пример:

```
main()
{
    cx_board_t b;
    cx_buf_t buf;

    disable();
```



```
cx_open_board (&b, 0, PORT, IRQ, DRQ);
cx_start_chan (&b.chan[0], &buf,
              (FP_SEG(&buf) << 4) + FP_OFF(&buf));
enable();
...
```

Функция *cx_enable_receive()*

```
void cx_enable_receive (cx_chan_t *c, int on)
```

Включает / выключает приёмник. Может применяться для выключения приёмника, когда отсутствует необходимость приёма пакетов.

c — Указатель на переменную состояния канала.
on — Не равно 0 — включить приёмник, иначе выключить. После вызова функции *cx_start_chan()* приемник включен.

Функция *cx_enable_transmit()*

```
void cx_enable_transmit (cx_chan_t *c, int on)
```

Включает / выключает передатчик.

c — Указатель на переменную состояния канала.
on — Не равно 0 — включить передатчик, иначе выключить. После вызова функции *cx_start_chan()* передатчик включен.

Функция *cx_receive_enabled()*

```
int cx_receive_enabled (cx_chan_t *c)
```

Проверяет, включен ли приёмник. Если приёмник включен, возвращает значение, отличное от 0.

c — Указатель на переменную состояния канала.

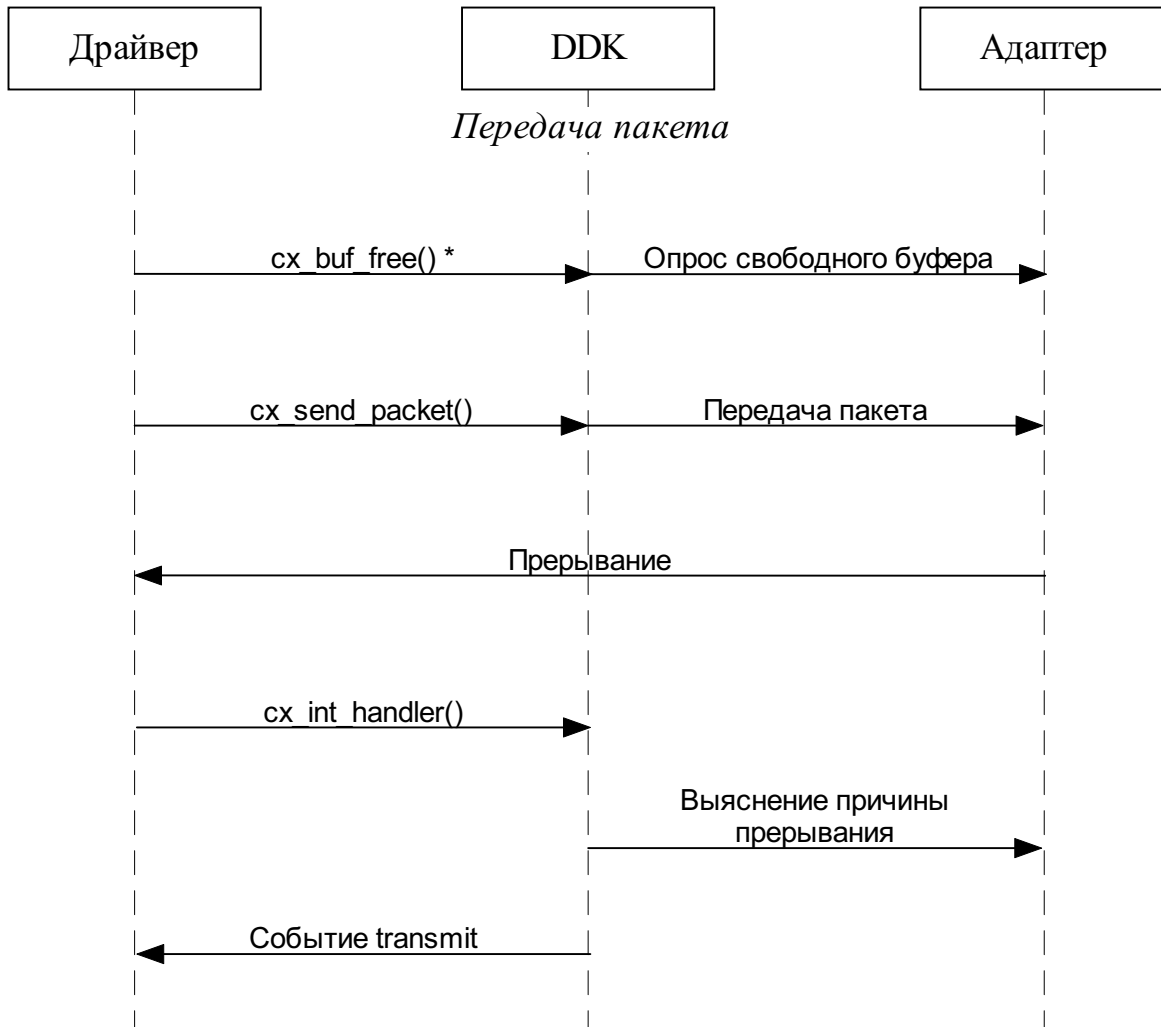
Функция *cx_transmit_enabled()*

```
int cx_transmit_enabled (cx_chan_t *c)
```

Проверяет, включен ли передатчик. Если передатчик включен, возвращает значение, отличное от 0.

c — Указатель на переменную состояния канала.

Передача данных



Функция `cx_send_packet()`

```
int cx_send_packet (cx_chan_t *c, char *data, int len,
void *attachment)
```

Ставит пакет в очередь на передачу. Возвращаемое значение:

- 0 — Пакет успешно помещён в очередь.
- 1 — В очереди нет свободных буферов (см. `cx_buf_free()`).
- 2 — Длина пакета превышает максимально допустимую (1600 байт).

После успешной передачи пакета будет вызван обработчик события.

- `c` — Указатель на переменную состояния канала
- `data` — Указатель на область памяти, содержащую данные
- `len` — Длина пакета в байтах

attachment — Указатель на прикрепленные данные, передается обработчику события при завершении передачи данного пакета

Функция *cx_buf_free()*

```
int cx_buf_free (cx_chan_t *c)
```

Проверяет наличие свободных буферов в очереди передатчика. Возвращает количество свободных буферов, или 0 при их отсутствии.

Функция *cx_int_handler()*

```
void cx_int_handler (cx_board_t *b)
```

Обработка аппаратного прерывания от адаптера. Пользователь DDK должен обеспечить вызов обработчика прерывания при его возникновении. Ей должен быть передан указатель на структуру состояния платы, вызвавшей прерывание. В различных операционных системах этого можно добиться по-разному. В операционных системах, которые передают обработчику прерывания номер происшедшего прерывания, по нему может быть определена плата. Для системы MS-DOS в составе набора разработчика имеются функции, обеспечивающие передачу обработчикам прерываний произвольных аргументов.

Функция *cx_led()*

```
void cx_led (cx_board_t *b, int on)
```

Адаптер Сигма22 имеет светодиод, который может быть использован программным обеспечением для индикации работы адаптера. Светодиод включается / выключается функцией *cx_led()*.

b — Структура данных адаптера
on — Включить / выключить светодиод.

Установка и опрос параметров канала

Функция `cx_set_baud()`

```
void cx_set_baud (cx_chan_t *c, unsigned long baud)
```

Управление режимом синхронизации и скоростью передачи данных. При `baud==0` устанавливает режим внешней синхронизации (от модема). При `baud!=0` устанавливает режим синхронизации от внутреннего генератора синхросигнала. По умолчанию устанавливается синхронизация от внешнего источника. В момент изменения частоты синхронизации могут возникать ошибки неправильного приёма пакетов. Перед изменением скорости рекомендуется убедиться, что буфер передатчика пуст.

`c` — Указатель на переменную состояния канала
`baud` — Частота генератора для внутренней синхронизации, или 0 для внешней синхронизации

Функция `cx_get_baud()`

```
unsigned long cx_get_baud (cx_chan_t *c)
```

Опрос режима синхронизации и скорости передачи данных. Возвращает значение частоты внутреннего генератора при внутренней синхронизации, или 0 при внешней синхронизации.

`c` — Указатель на переменную состояния канала

Функция `cx_set_mode()`

```
int cx_set_mode (cx_chan_t *c, int mode)
```

Устанавливает режим передачи данных. Текущий режим работы канала доступен как поле `mode` структуры `cx_chan_t`. Для асинхронного режима требуется переключение на внутреннюю синхронизацию. В случае успешного завершения возвращает 0. При несоответствии режима и типа канала возвращает -1.

`c` — Указатель на переменную состояния канала
`mode` — Режим передачи:
M_ASYNC — асинхронный режим передачи данных
M_HDLC — синхронный режим передачи данных (HDLC)

Функция `cx_set_nrzi()`

```
void cx_set_nrzi (cx_chan_t *c, int nrzi)
```

Переключает режим кодирования, NRZ (по умолчанию) или NRZI. Используется в синхронном режиме передачи данных.

`c` — Указатель на переменную состояния канала
`nrzi` — Если не равно 0, устанавливается кодирование NRZI, иначе NRZ

Функция `cx_get_nrzi()`

```
int cx_get_nrzi (cx_chan_t *c)
```

Опрос режима кодирования сигнала. Возвращает 1 для NRZI, или 0 для NRZ.

c — Указатель на переменную состояния канала

Функция `cx_set_dp11()`

```
void cx_set_dp11 (cx_chan_t *c, int on)
```

Включает / выключает режим синхронизации DPLL. Для режима DPLL требуется переключение на внутреннюю синхронизацию. Во избежание потери синхронизации рекомендуется применять режим DPLL совместно с кодированием NRZI. Используется в синхронном режиме передачи данных.

c — Указатель на переменную состояния канала

on — Не равно 0 — включить DPLL, иначе выключить

Функция `cx_get_dp11()`

```
int cx_get_dp11 (cx_chan_t *c)
```

Проверяет, включен ли режим DPLL. Возвращает 1 для DPLL, иначе 0.

c — Указатель на переменную состояния канала

Функция `cx_set_loop()`

```
int cx_set_loop (cx_chan_t *c, int on)
```

Включает/выключает локальный шлейф. Передаваемые данные заворачиваются на вход приемника.

c — Указатель на переменную состояния канала

on — Не 0 — включить, иначе выключить

Функция `cx_get_loop()`

```
int cx_get_loop (cx_chan_t *c)
```

Опрос локального шлейфа. Возвращает 1 при включенном шлейфе, иначе 0.

c — Указатель на переменную состояния канала

Асинхронный режим

Функция `cx_set_async_param()`

```
void cx_set_async_param (cx_chan *c, int baud, int bits, int parity,  
int stop2, int ignpar, int rtscts, int ixon, int ixany,  
int symstart, int symstop)
```

Устанавливает параметры асинхронного режима.

<i>c</i>	— Указатель на переменную состояния канала
<i>baud</i>	— Скорость передачи асинхронного канала (бит/сек)
<i>bits</i>	— Число бит на символ (5-8)
<i>parity</i>	— Четность (0 — нет четности, 1 — нечет, 2 — чет)
<i>stop2</i>	— Количество стоповых битов (0 — один бит, иначе два бита)
<i>ignpar</i>	— Игнорировать бит четности при приеме
<i>rtscts</i>	— Аппаратное управление потоком RTS/CTS (0 — выключить, иначе включить)
<i>ixon</i>	— Аппаратное управление потоком XON/XOFF (0 — выключить, иначе включить)
<i>ixany</i>	— Разрешить возобновление передачи по приему любого символа (не только XON; 0 — запретить, иначе разрешить)
<i>symstart</i>	— Значение символа XON
<i>symstop</i>	— Значение символа XOFF

Функция `cx_flush_transmit()`

```
void cx_flush_transmit (cx_chan *c)
```

Очистка FIFO передатчика. Данные, находящиеся в FIFO, теряются.

<i>c</i>	— Указатель на переменную состояния канала
----------	--------------------------------------------

Функция `cx_transmitter_ctl()`

```
void cx_transmitter_ctl (cx_chan *c, int start)
```

Включает / выключает передатчик. Применяется для программного управления потоком (XON/XOFF).

<i>c</i>	— Указатель на переменную состояния канала
<i>start</i>	— Не 0 — включить передатчик, иначе выключить

Функция `cx_xflow_ctl()`

```
void cx_xflow_ctl (cx_chan *c, int no)
```

Передача символов XON/XOFF. Указанный символ передается в линию вне очереди. Применяется для программного управления потоком.

c — Указатель на переменную состояния канала
no — 1 — XON, 0 — XOFF

Функция `cx_send_break()`

```
void cx_send_break (cx_chan *c, int msec)
```

Передача сигнала break.

c — Указатель на переменную состояния канала
msec — Длительность сигнала break в миллисекундах (10...10000)

Модемные сигналы

При пуске канала сигналы DTR и RTS сбрасываются. В дальнейшем их можно изменить с помощью функций `cx_set_dtr()` и `cx_set_rts()`. Опрос сигналов DSR, CTS и DCD производится функциями `cx_get_dsr()`, `cx_get_cts()` и `cx_get_cd()`. При изменении сигнала DCD вызывается обработчик модемного события.

Функция `cx_set_dtr()`

```
void cx_set_dtr (cx_chan_t *c, int on)
```

Включает / выключает сигнал DTR. Текущее состояние сигнала доступно как поле *dtr* структуры *cx_chan_t*.

c — Указатель на структуру канала
on — Не равно 0 — включить сигнал DTR, иначе выключить

Функция `cx_set_rts()`

```
void cx_set_rts (cx_chan_t *c, int on)
```

Включает / выключает сигнал RTS. Текущее состояние сигнала доступно как поле *rts* структуры *cx_chan_t*.

c — Указатель на структуру канала
on — Не равно 0 — включить сигнал RTS, иначе выключить

Функция `cx_get_dsr()`

```
int cx_get_dsr (cx_chan_t *c)
```

Опрашивает состояние сигнала DSR. Возвращает 1 если сигнал DSR активен, иначе 0.

`c` — Указатель на структуру канала

Функция `cx_get_cts()`

```
int cx_get_cts (cx_chan_t *c)
```

Опрашивает состояние сигнала CTS. Возвращает 1 если сигнал CTS активен, иначе 0.

`c` — Указатель на структуру канала

Функция `cx_get_cd()`

```
int cx_get_cd (cx_chan_t *c)
```

Возвращает состояние сигнала DCD. Возвращает 1 если сигнал DCD активен, иначе 0.

`c` — Указатель на структуру канала

Пример

Пример дан для операционной системы DOS и компилятора Borland Turbo C/C++ 1.0.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <conio.h>
#include <dos.h>
#include "cxddk.h"

#define IRQ      5
#define DMA      7
#define BAUD     128000L

cx_board_t adapter;
cx_buf_t buffer;
char data [256];
long sent, received;

/* Преобразование виртуального адреса в физический
 * для реального режима i86 */
unsigned long virt_to_phys (void *virt)
{
    return ((unsigned long) FP_SEG(virt) << 4) + FP_OFF(virt);
}

/* Вызывается при приеме пакета */
void receive (cx_chan_t *c, char *buf, int len)
{
    received += len;

    /* Проверка данных */
    if (len != sizeof(data) || memcmp (data, buf, len) != 0)
        fprintf ("\n--Data Error--\n");
}

/* Вызывается после окончания передачи пакета */
void transmit (cx_chan_t *c, void *attachment, int len)
{
    memset (data, 'Z', sizeof (data));
    while (! cx_send_packet (c, data, sizeof (data), 0))
        sent += sizeof (data);
}

/* Обработка ошибок */
void error (cx_chan_t *c, int reason)
{
    switch (reason) {
        case CX_FRAME:      fprintf ("\n--Framing Error--\n"); break;
        case CX_CRC:        fprintf ("\n--CRC Error--\n");   break;
        case CX_OVERRUN:    fprintf ("\n--Overrun--\n");     break;
        case CX_OVERFLOW:   fprintf ("\n--Overflow--\n");    break;
        case CX_UNDERRUN:   fprintf ("\n--Underrun--\n");    break;
    }
}
```

```
int main ()
{
    port_t porttab [NBRD];
    cx_chan_t *c;
    int cnt;

    /* Игнорируем ^C */
    signal (SIGINT, SIG_IGN);

    /* Поиск адаптера */
    disable();
    if (! cx_find (porttab)) {
        enable();
        printf ("Adapter not found\n");
        exit (-1);
    }

    if (! cx_open_board (&adapter, 0, porttab[0], IRQ, DMA)) {
        enable();
        printf ("Initialization failure\n");
        exit (-1);
    }

    /* Установка обработчика прерывания */
    if (! ddk_int_alloc (IRQ, &cx_int_handler, &adapter)) {
        enable();
        printf ("Irq busy\n");
        exit (-1);
    }
    enable();
    printf ("Found adapter Sigma-%s at port %xh irq %d dma %d\n",
        adapter.name, adapter.port, adapter.irq, adapter.dma);

    /* Выбор канала для тестирования */
    c = &adapter.chan[0];

    /* Регистрация обработчиков событий */
    cx_register_receive (c, &receive);
    cx_register_transmit (c, &transmit);
    cx_register_error (c, &error);

    /* Запуск канала */
    printf ("Starting HDLC channel %d at %ld baud\n", c->num, BAUD);
    printf ("Setting up DPLL mode with NRZI encoding\n");
    printf ("Enabling internal loopback\n");
    disable();
    cx_set_mode (c, M_HDLC);
    cx_start_chan (c, &buffer, virt_to_phys (&buffer));
    cx_set_baud (c, BAUD);
    cx_set_dp11 (c, 1);
    cx_set_nrzi (c, 1);
    cx_set_loop (c, 1);

    /* Устанавливаем сигналы RTS и DTR */
    cx_set_rts (c, 1);
    cx_set_dtr (c, 1);
    cx_led (&adapter, 1);
}
```

```
enable();

/* Основной цикл */
while (! kbhit ())
    cprintf ("\r%c Bytes sent %ld, received %ld\r",
            "/-\| |" [cnt++ >> 8 & 3], sent, received);
getch () || getch ();

/* Сброс платы */
printf ("\nClosing\n");

disable();
cx_close_board (&adapter);

/* Освобождаем прерывание */
ddk_int_restore (IRQ);
enable();
return 0;
}
```

