Набор разработчика драйверов (DDK) для адаптеров семейства Кроникс-Тау

Содержание

Набор разработчика для адаптеров семейства Tau представляет собой библиотеку функций 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 DOS.

Процесс работы с адаптером состоит из следующих этапов:

• Инициализация аппаратуры и структуры данных, содержащей информацию о текущем состоянии 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:

ct_find()	— поиск установленных адаптеров
ct_open_board()	— инициализация платы
ct_close_board()	— сброс платы
ct_probe_board()	— распознавание адаптера
ct_init()	— инициализация структуры данных адаптера
ct_setup_board()	— сброс адаптера и загрузка firmware
ct_probe_irq()	— проверка прерывания
ddk_int_alloc()	— установка обработчика прерывания
ddk_int_restore()	— восстановление вектора прерывания

Установка обработчиков:

ct_register_transmit()	— установка обработчика передачи пакета
ct_register_receive()	— установка обработчика приёма пакета
ct_register_modem()	— установка обработчика изменения модемных сигналов
ct_register_error()	— установка обработчика ошибок

Пуск канала:

ct_start_chan()	— запуск канала

ct_enable_receive() — выключение/включение приёмника
ct_enable_transmit() — выключение/включение передатчика
ct_receive_enabled() — определение состояния включения приёмника
ct_transmit_enabled() — определение состояния включения передатчика

Передача данных:

ct_send_packet() — передача пакета
ct_buf_free() — проверка свободного места в буфере передатчика
ct_int_handler() — обработка прерывания
ct_g703_timer() — накопление статистики G.703/E1
ct_led() — выключение/включение светодиода на адаптерах E1 и G.703

Установка и опрос параметров канала:

ct_set_baud() — выбор частоты генератора синхросигнала
ct_get_baud() — определение частоты генератора синхросигнала
ct_set_dpll() — выключение/включение DPLL
ct_get_dpll() — опрос режима DPLL
ct_set_nrzi() — выключение/включение кодирования NRZI
ct_get_nrzi() — опрос кодирования NRZI/NRZ
ct_set_invclk() — выключение/включение инвертирования синхроимпульсов
ct_get_invclk() — опрос режима инвертирование синхроимпульсов
ct_set_loop() — выключение/включение локального шлейфа
ct_get_loop() — опрос состояния включения локального шлейфа

Модемные сигналы:

ct_set_dtr() — установка сигнала DTR
ct_set_rts() — установка сигнала RTS
ct_get_dsr() — опрос сигнала DSR
ct_get_cts() — опрос сигнала CTS
ct_get_cd() — опрос сигнала CD

Параметры каналов E1/G.703:

ct_set_config() — выбор конфигурации адаптера G.703 или E1
ct_get_config() — опрос конфигурации адаптера G.703 или E1
ct_set_clk() — выбор источника синхронизации.канала G.703 или E1
ct_get_clk() — опрос источника синхронизации.канала G.703 или E1
ct_set_ts() — выбор канальных интервалов E1
ct_get_ts() — опрос канальных интервалов E1
ct_set_subchan() — выбор канальных интервалов подканала E1
ct_get_subchan() — определение набора канальных интервалов подканала E1
ct_set_higain() — включение/выключение высокой чувствительности приёмника
ct_get_higain() — опрос режима чувствительности приёмника канала E1
ct_set_phony() — включение/выключение “телефонного” режима (для Tau/E1d)
ct_get_phony() — опрос “телефонного” режима
ct_get_lq() — опрос уровня сигнала в линии (G.703)

Файлы

Для создания программ необходимы следующие файлы:

ctddk.h — Прототипы функций библиотеки, типы данных и константы. Включать в программу нужно только этот файл.

ctaureg.h, hdc64570.h, ds2153.h, am8530.h, lxt318.h — Описания интерфейса используемых контроллеров.

cronyxfw.h — Структуры данных firmware.

ctaufw.h, ctaue1fw.c, ctaug7fw.c — Загружаемое аппаратное обеспечение адаптеров (firmware).

machdep.h — Машинно-зависимые определения.

ctddk.c — С-код процедур пакета.

ctau.c — Код низкоуровневых процедур работы с аппаратурой.

ddkdos.c — Функции для работы с прерываниями в DOS.

В каталоге examples находятся примеры использования DDK:

test.c — Пример для адаптера Tau.

teste1.c — Пример для адаптера Tau/E1.

testg703.c — Пример для адаптера Tau/G.703.

Структуры данных

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

Структура ct_board_t

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

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

unsigned char type; — тип адаптера:

- *B_TAU* — Модель Tau
- *B_TAU_E1* — Модель Tau-E1 ревизии A, B

B_TAU_E1C — Модель Tau-E1 ревизии C
B_TAU_E1D — Модель Tau-E1D (для телефонии)
B_TAU_G703 — Модель Tau-G703 ревизии A, B
B_TAU_G703C — Модель Tau-G703 ревизии C

char name[16]; — название модели адаптера в текстовом виде (см. также поле *type*).

unsigned char num; — 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.

ct_chan_t chan[NCHAN]; — Структуры данных каналов (NCHAN=2).

Структура ct_chan_t

```
typedef struct {
  unsigned char type;
  unsigned char num;
  ct_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;
  unsigned short status;
  unsigned long totsec;
  unsigned long cursec;
  ct_gstat_t currnt;
  ct_gstat_t total;
  ct_gstat_t interval [48];
  ...
} ct_chan_t;
```

unsigned char type; — тип канала:

T_SERIAL — V.35 / RS-530 / RS-232
T_E1 — E1 (ИКМ-30)
T_G703 — G.703.6 (2048 кбит/сек)
T_E1_SERIAL — переключаемый E1 или V.35 / RS-530 / RS-232
T_G703_SERIAL — переключаемый G.703.6 или

V.35 / RS-530 / RS-232

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

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

unsigned char mode; — режим передачи данных:
- M_HDLC — синхронный режим V.35 / RS-530 / RS-232
- M_E1 — режим E1
- M_G703 — режим G.703

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

unsigned char rts; — состояние сигнала RTS: *1,* если сигнал активен, иначе — *0.*

*void *sys;* — указатель общего назначения, может использоваться разработчиком драйвера для своих целей

int debug; — режим отладочной печати, может использоваться разработчиком драйвера для своих целей

unsigned long rintr; — счетчик прерываний приемника

unsigned long tintr; — счетчик прерываний передатчика

unsigned long mintr; — счетчик модемных прерываний

unsigned long ibytes; — счетчик принятых байтов

unsigned long ipkts; — счетчик принятых пакетов

unsigned long ierrs; — счетчик ошибок приемника

unsigned long obytes; — счетчик переданных байтов

unsigned long opkts; — счетчик переданных пакетов

unsigned long oerrs; — счетчик ошибок передатчика

unsigned short status; — состояние линии G.703
- ESTS_NOALARM — нормальное состояние
- ESTS_LOS — нет сигнала в линии G.703/E1
- ESTS_AIS — принимается сигнал “все единицы” (E1)
- ESTS_AIS16 — в канальном интервале 16 принимается сигнал “все единицы”, при включенном режиме CAS (E1)
- ESTS_LOF — нет цикловой синхронизации E1
- ESTS_LOMF — нет сверхцикловой синхронизации E1, при включенном режиме CAS или CRC4
- ESTS_FARLOF — нет удаленной цикловой синхронизации E1 (remote alarm)
- ESTS_FARLOMF — нет удаленной сверхцикловой синхронизации E1 (remote alarm в канальном интервале 16)

unsigned long totsec; — общее время работы канала G.703 в секундах

unsigned long cursec; — длительность текущего периода в секундах (см. *currnt*)

ct_gstat_t currnt; — статистика G.703/E1 за текущий период (до 15 минут)

ct_gstat_t total; — общая статистика G.703/E1

ct_gstat_t interval [48]; — статистика G.703/E1 за последние 48 периодов (12 часов)

Структура ct_gstat_t

Применяется для хранения статистика канала E1/G.703.

```
typedef struct {
  unsigned long bpv;
  unsigned long fse;
  unsigned long crce;
  unsigned long rcrce;
  unsigned long uas;
  unsigned long les;
  unsigned long es;
  unsigned long bes;
  unsigned long ses;
  unsigned long oofs;
  unsigned long css;
  unsigned long dm;
} ct_gstat_t;
```

unsigned long bpv; — количество нарушений биполярного кода G.703/E1

unsigned long fse; — количество ошибок циклового синхронизма (E1)

unsigned long crce; — количество ошибок CRC4 (E1)

unsigned long rcrce; — количество удаленных ошибок CRC4 (E1)

unsigned long uas; — количество секунд, в течение которых отсутствовал входной сигнал G.703/E1

unsigned long les; — количество секунд, в течение которых происходили нарушения биполярного кода G.703/E1

unsigned long es; — количество секунд, в течение которых происходили ошибки цикловой или сверхцикловой синхронизации E1, либо проскальзывания (slip)

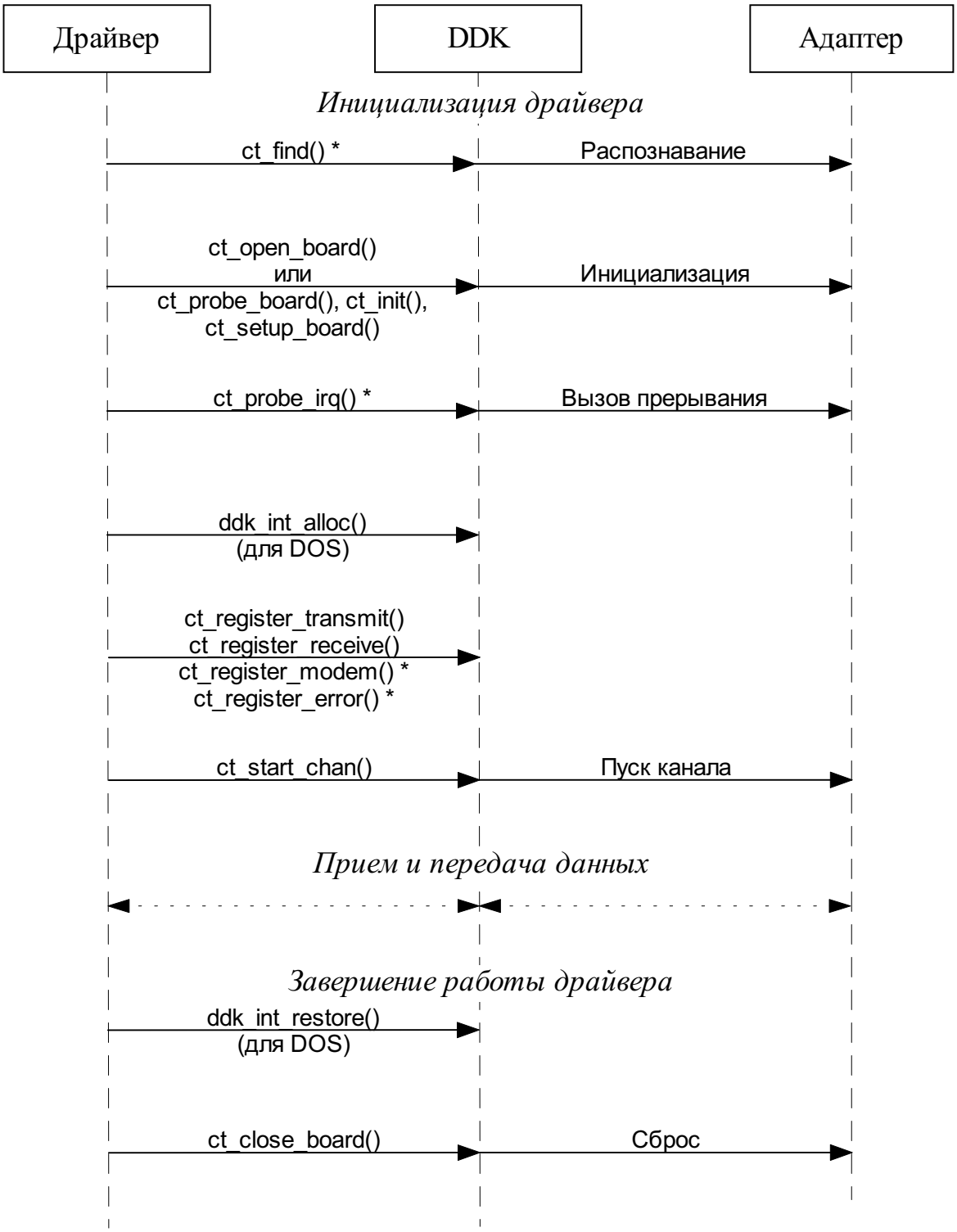
unsigned long bes; — количество секунд, в течение которых наблюдалось более 1, но менее 832 ошибок цикловой синхронизации E1

unsigned long ses; — “существенно ошибочные” секунды, в течение которых наблюдалось 2048 или более нарушений биполярного кода, либо 832 или более ошибок цикловой синхронизации E1

unsigned long oofs; — количество секунд, в течение которых отсутствовала цикловая или сверхцикловая синхронизация E1

unsigned long css; — количество секунд, в течение которых происходили проскальзывания циклов E1 (slip)

unsigned long dm; — количество минут, в течение которых уровень ошибок превышал 10^{-6} (G.703/E1)



* — необязательные вызовы

Инициализация

Функция ct_find()

```
int ct_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()
{
  ct_board_t b;
  unsigned short porttab[NBRD];

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

Функция ct_open_board()

```
int ct_open_board (ct_board_t *b, int num,
  unsigned short port, int irq, int dma)
```

Производит сброс и инициализацию адаптера, загрузку firmware, инициализирует параметры состояния адаптера. Заменяет собой функции *ct_probe_board()*, *ct_init()* и *ct_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 *ct_find()*. Проверку работоспособности прерывания можно произвести с помощью функции *ct_probe_irq()*.

Пример:

```
main()
{
```

```
ct_board_t b;

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

Функция ct_close_board()

```
void ct_close_board (ct_board_t *b)
```

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

Функция ct_probe_board()

```
int ct_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 *ct_find()*. Проверку работоспособности прерывания можно произвести с помощью функции *ct_probe_irq()*.

Пример:

```
#include "ctaufw.h"

main()
{
  ct_board_t b;

  disable();
  if (! ct_probe_board (PORT, IRQ, DRQ)) {
       enable();
       printf ("Adapter not found\n");
       exit (-1);
  }
  ct_init (&b, 0, PORT, IRQ, DRQ,
       ctau_fw_data, ctau_fw_len, ctau_fw_tvec);
  if (! ct_setup_board (&b, ctau_fw_data, ctau_fw_len, ctau_fw_tvec)){
       enable();
       printf ("Loading firmware failed\n");
```

```
        exit (-1);
    }
    ...
```

Функция ct_init()

```
void ct_init (ct_board_t *b, int num, unsigned short port, int irq,
  int dma, const unsigned char *firmware, long bits,
  const cr_dat_tst_t *tst)
```

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).

firmware — Массив данных firmware. Данные находятся в файле *ctaufw.h*.

bits — Объем firmware в битах.

tst — Вектор контроля загрузки.

Пример: см. *ct_probe_board()*.

Функция ct_setup_board()

```
int ct_setup_board (ct_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 файлах *ctaufw.h, ctaue1fw.c, ctaug7fw.c*.

bits — Объем firmware в битах.

tst — Вектор контроля загрузки, или NULL если загрузка не требуется.

Пример: см. *ct_probe_board()*.

Функция ct_probe_irq()

```
int ct_probe_irq (ct_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, **до** выполнения действий с адаптером.

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

Пример:

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

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

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

   /* Проверка маски прерывания. */
   if ((mask & ~busy) != 1 << IRQ) {
        /* Освобождение линии прерывания. */
        ct_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()
{
   ct_board_t b;

   disable();
   if (! ct_open_board (&b, 0, PORT, IRQ, DRQ)) {
        enable();
```

```
        printf ("Initialization error! \n");
        exit (-1);
    }
    ddk_int_alloc (IRQ, &ct_int_handler, b);
    ...
```

Функция ddk_int_restore

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

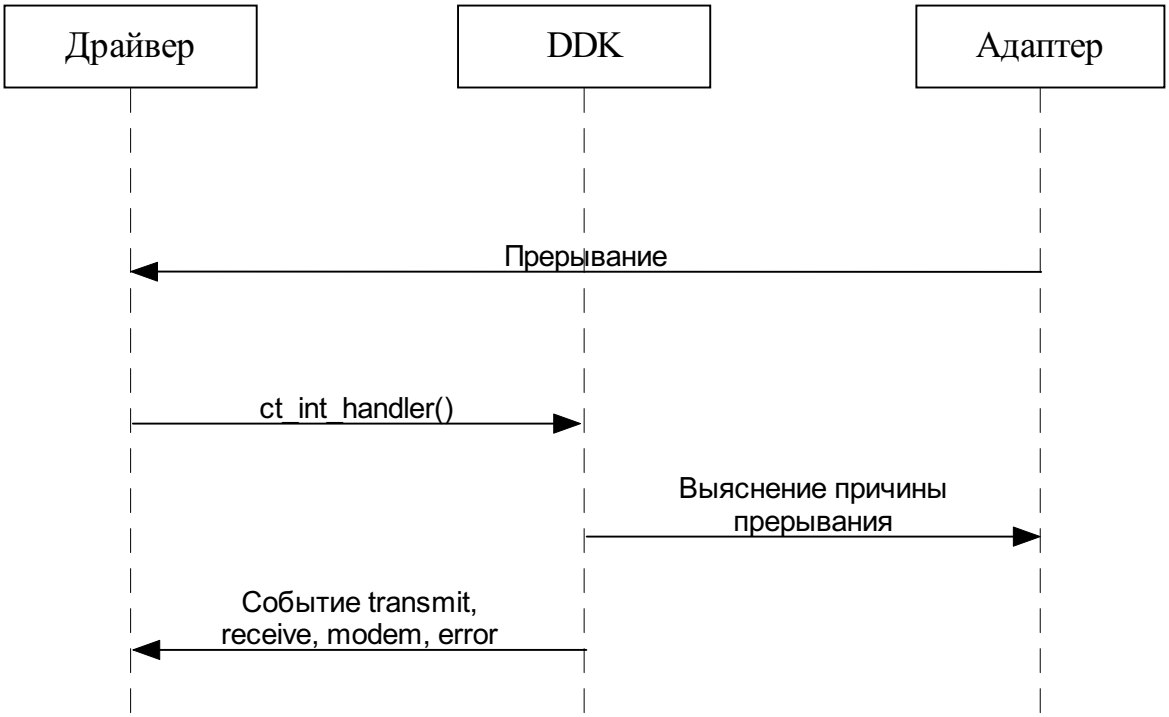
Восстанавливает вектор прерывания.

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 *ct_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 ct_register_transmit()

```
void ct_register_transmit (ct_chan_t *c,
  void (*func) (ct_chan_t *c, void *attachment, int len))
```

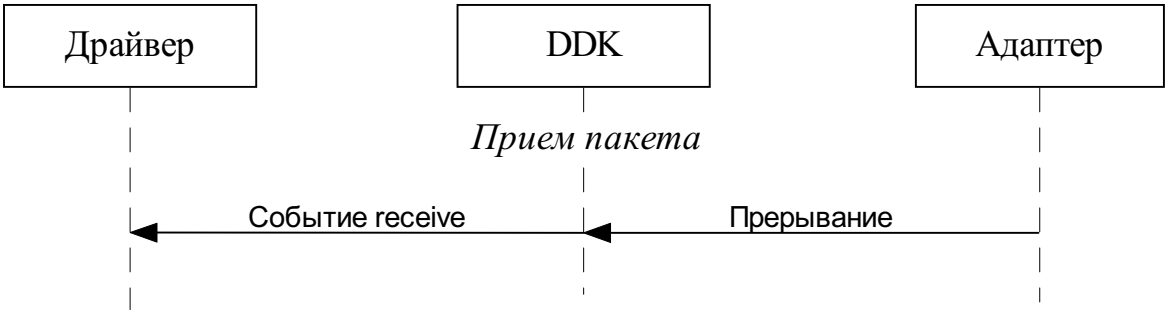
Функция-обработчик вызывается при успешном завершении передачи пакета.

Аргументы, передаваемые обработчику:

c — Указатель на структуру канала

attachment — Аргумент соответствующего вызова функции *ct_send_packet()*, может использоваться драйвером для передачи ссылки на системно-зависимую структуру данных, относящуюся к пакету.

len — Длина пакета в байтах



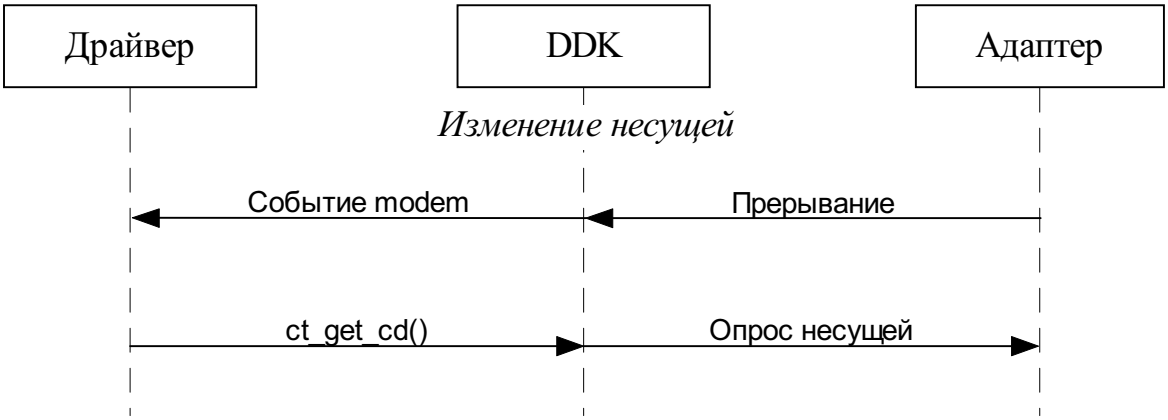
Функция ct_register_receive()

```
void ct_register_receive (ct_chan_t *c,
  void (*func) (ct_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:

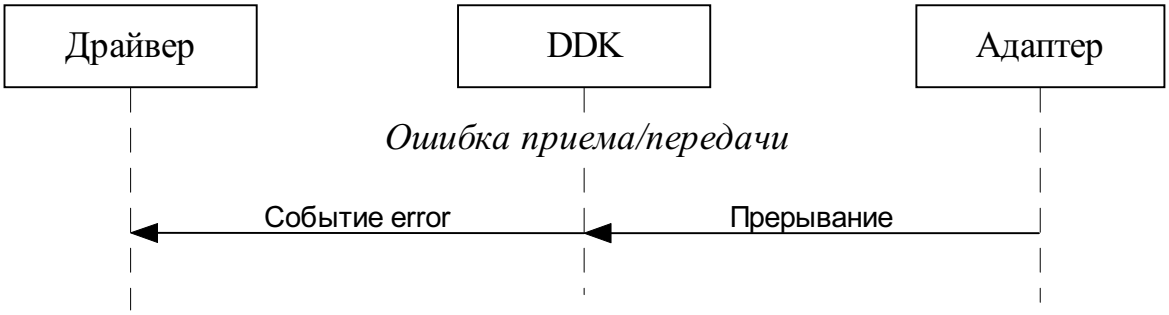
c — Указатель на структуру канала
data — Указатель на буфер данных
len — Длина пакета в байтах



Функция ct_register_modem()

```
void ct_register_modem (ct_chan_t *c, void (*func) (ct_chan_t *c))
```

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



Функция ct_register_error()

```
void ct_register_error (ct_chan_t *c,
  void (*func)(ct_chan_t *c, int data))
```

Функция-обработчик вызывается при обнаружении ошибки.

Аргументы, передаваемые обработчику:

c — Указатель на структуру канала

data — Код ошибки:

- *CT_FRAME* — Ошибка кадра
- *CT_CRC* — Ошибка контрольной суммы
- *CT_UNDERRUN* — Опустошение FIFO передатчика
- *CT_OVERRUN* — Переполнение FIFO приёмника
- *CT_OVERFLOW* — Переполнение буфера приемника
- *CT_BREAK* — Сигнал break (асинхронный режим)

Пуск канала

Функция ct_start_chan()

```
void ct_start_chan (ct_chan_t *c, ct_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()
{
  ct_board_t b;
  ct_buf_t buf;

  disable();
```

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

Функция ct_enable_receive()

```
void ct_enable_receive (ct_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 *ct_start_chan()* приемник включен.

Функция ct_enable_transmit()

```
void ct_enable_transmit (ct_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 *ct_start_chan()* передатчик включен.

Функция ct_receive_enabled()

```
int ct_receive_enabled (ct_chan_t *c)
```

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

c — Указатель на переменную состояния канала.

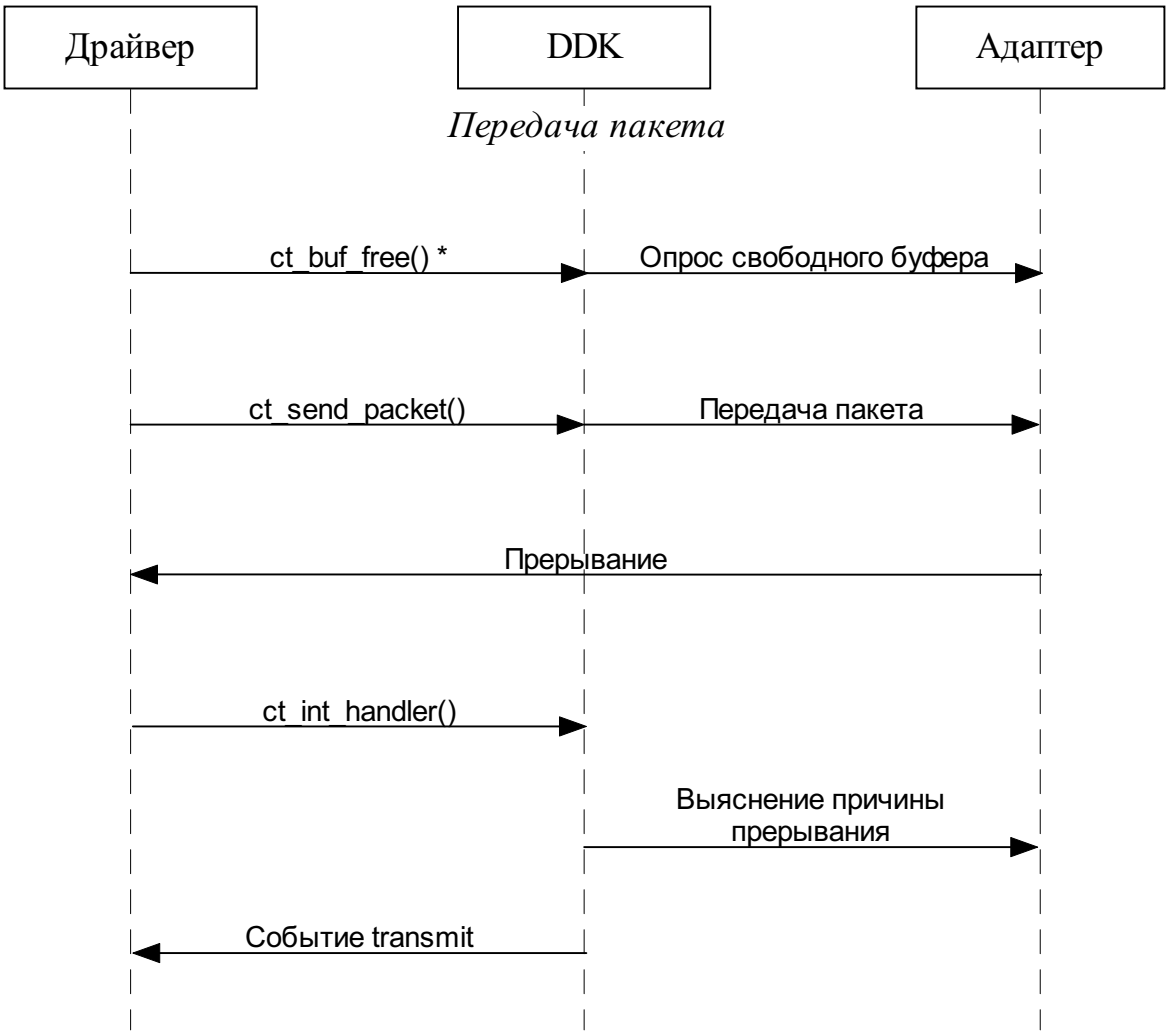
Функция ct_transmit_enabled()

```
int ct_transmit_enabled (ct_chan_t *c)
```

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

c — Указатель на переменную состояния канала.

Передача данных



Функция ct_send_packet()

```
int ct_send_packet (ct_chan_t *c, char *data, int len,
  void *attachment)
```

Ставит пакет в очередь на передачу. Данные помещаются в буфер передатчика. После успешной передачи пакета вызывается обработчик события. Для идентификации пакета обработчиком функции *ct_send_packet()* может передаваться указатель на связанные с пакетом данные. Возвращаемое значение:

0 — Пакет успешно помещён в очередь.
-1 — В очереди нет свободных буферов (см. *ct_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 прикреплённые данные, передается обработчику события при завершении передачи данного пакета

Функция ct_buf_free()

```
int ct_buf_free (ct_chan_t *c)
```

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

Функция ct_int_handler()

```
void ct_int_handler (ct_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 ct_g703_timer()

```
void ct_g703_timer (ct_chan_t *c)
```

Накопление статистики канала G.703. Только для адаптера Tau/G703. Пользователь DDK должен обеспечить ежесекундный вызов этой функции для всех каналов, работающих в режиме G.703.

Функция ct_led()

```
void ct_led (ct_board_t *b, int on)
```

Адаптеры Tau/E1 и Tau/G.703 (но не Tau) имеют имеет светодиод, который может быть использован программным обеспечением для индикации работы адаптера. Светодиод включается / выключается функцией *ct_led()* .

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

Установка и опрос параметров канала

Функция ct_set_baud()

```
void ct_set_baud (ct_chan_t *c, unsigned long baud)
```

Для каналов V.35/RS-530/RS-232 — 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.

Для каналов G.703 (Tau/G703) — управление скоростью данных. Значение *baud* может равняться 2048000, 1024000, 512000, 256000, 128000 или 64000 (бит/сек).

Для каналов E1 (Tau/E1) функция *ct_set_baud()* неприменима, изменение скорости передачи производится функцией *ct_set_ts()*.

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 ct_get_baud()

```
unsigned long ct_get_baud (ct_chan_t *c)
```

Опрос режима синхронизации и скорости передачи данных. Возвращает значение частоты внутреннего генератора при внутренней синхронизации, или 0 при внешней синхронизации. Для каналов G.703 и E1 — возвращает значение скорости данных.

c — Указатель на переменную состояния канала

Функция ct_set_dpll()

```
void ct_set_dpll (ct_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 ct_get_dpll()

```
int ct_get_dpll (ct_chan_t *c)
```

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

c — Указатель на переменную состояния канала

Функция ct_set_nrzi()

```
void ct_set_nrzi (ct_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
on — Если не равно 0, устанавливается кодирование NRZI, иначе NRZ

Функция ct_get_nrzi()

```
int ct_get_nrzi (ct_chan_t *c)
```

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

c — Указатель на переменную состояния канала

Функция ct_set_invclk()

```
void ct_set_invclk(ct_chan_t *c, int on)
```

Включает / выключает инвертирование синхроимпульсов передатчика. По умолчанию инвертирование синхроимпульсов выключено.

c — Указатель на переменную состояния канала.
on — Если не равно 0 , включить , иначе выключить.

Функция ct_get_invclk()

```
int ct_get_invclk (ct_chan_t *c)
```

Проверяет, включен ли режим инвертирования синхроимпульсов передатчика. Возвращает *1* если инвертирование включено, иначе — *0*.

c — Указатель на переменную состояния канала.

Функция ct_set_loop()

```
int ct_set_loop (ct_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 ct_get_loop()

```
int ct_get_loop (ct_chan_t *c)
```

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

c — Указатель на переменную состояния канала

Модемные сигналы

При пуске канала сигналы DTR и RTS сбрасываются. В дальнейшем их можно изменить с помощью функций *ct_set_dtr()* и *ct_set_rts()*. Опрос сигналов DSR, CTS и DCD производится функциями *ct_get_dsr()*, *ct_get_cts()* и *ct_get_cd()*. При изменении сигнала DCD вызывается обработчик модемного события.

Функция ct_set_dtr()

```
void ct_set_dtr (ct_chan_t *c, int on)
```

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

c — Указатель на структуру канала

on — Не равно 0 — включить сигнал DTR, иначе выключить

Функция ct_set_rts()

```
void ct_set_rts (ct_chan_t *c, int on)
```

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

c — Указатель на структуру канала

on — Не равно 0 — включить сигнал RTS, иначе выключить

Функция ct_get_dsr()

```
int ct_get_dsr (ct_chan_t *c)
```

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

c — Указатель на структуру канала

Функция ct_get_cts()

```
int ct_get_cts (ct_chan_t *c)
```

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

c — Указатель на структуру канала

Функция ct_get_cd()

```
int ct_get_cd (ct_chan_t *c)
```

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

c — Указатель на структуру канала

Параметры каналов E1 и G.703

Мультиплексор Tau/E1 имеет программный переключатель, с помощью которого можно выбрать три конфигурации платы:

- Конфигурация A (рис.1): сдвоенный модем E1**.** Каналы E1-0 и E1-1 функционируют независимо и образуют два идентичных потока данных в память компьютера. Скорость передачи данных определяется количеством используемых канальных интервалов. Устанавливается по умолчанию.
- Конфигурация B (рис.2): мультиплексор с одним каналом данных, подканалом E1 и цифровым интерфейсом. Канальные интервалы основной линии E1-0 разделяются между каналом данных и подканалом E1-1. Поток данных в/из памяти компьютера формируется HDLC-контроллером 0. Остальные канальные интервалы транслируются в подканал E1-1. Эта конфигурация позволяет соединять несколько адаптеров Tau/E1 (до 30) в цепочку, с расстоянием между узлами до 1.5 км. Цифровой интерфейс и HDLC-контроллер 1 образуют независимый синхронно/асинхронный канал со скоростью до 4 Мбит/сек.
- Конфигурация C (рис.3): мультиплексор с двумя каналами данных и подканалом E1. Канальные интервалы основной линии E1-0 разделяются между двумя каналами

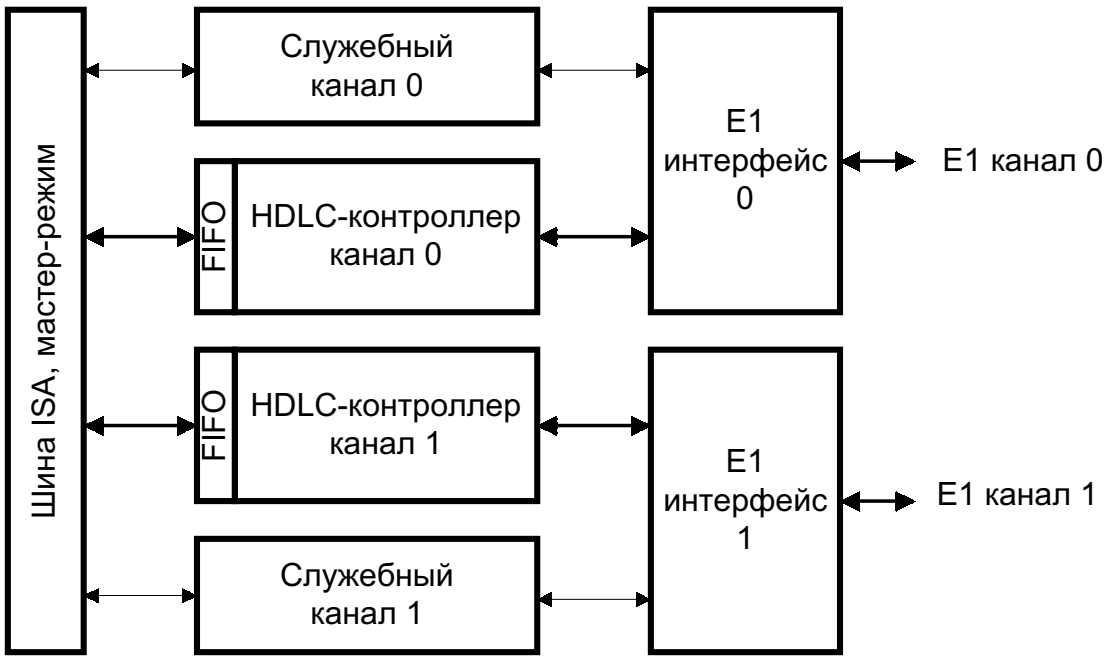


Рисунок 1: Конфигурация A — сдвоенный модем E1.

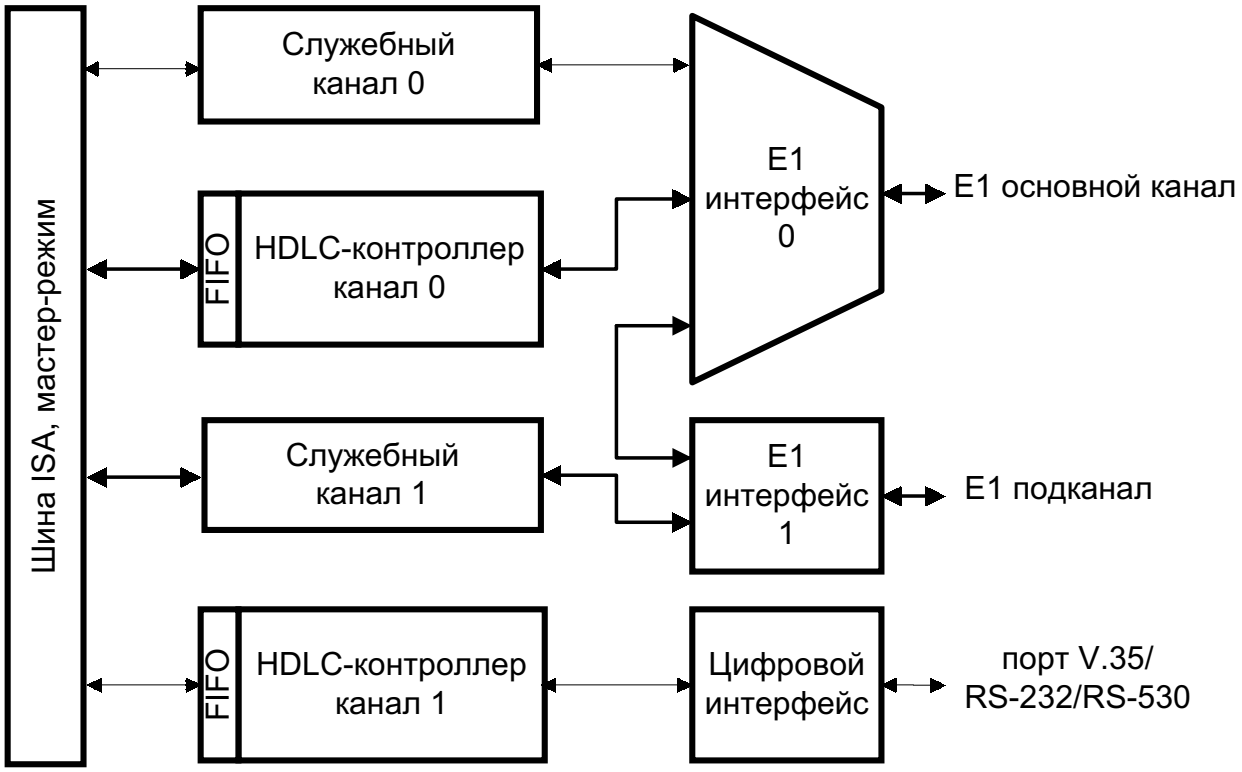


Рисунок 2: Конфигурация B — мультиплексор с одним каналом данных, подканалом E1 и цифровым интерфейсом.

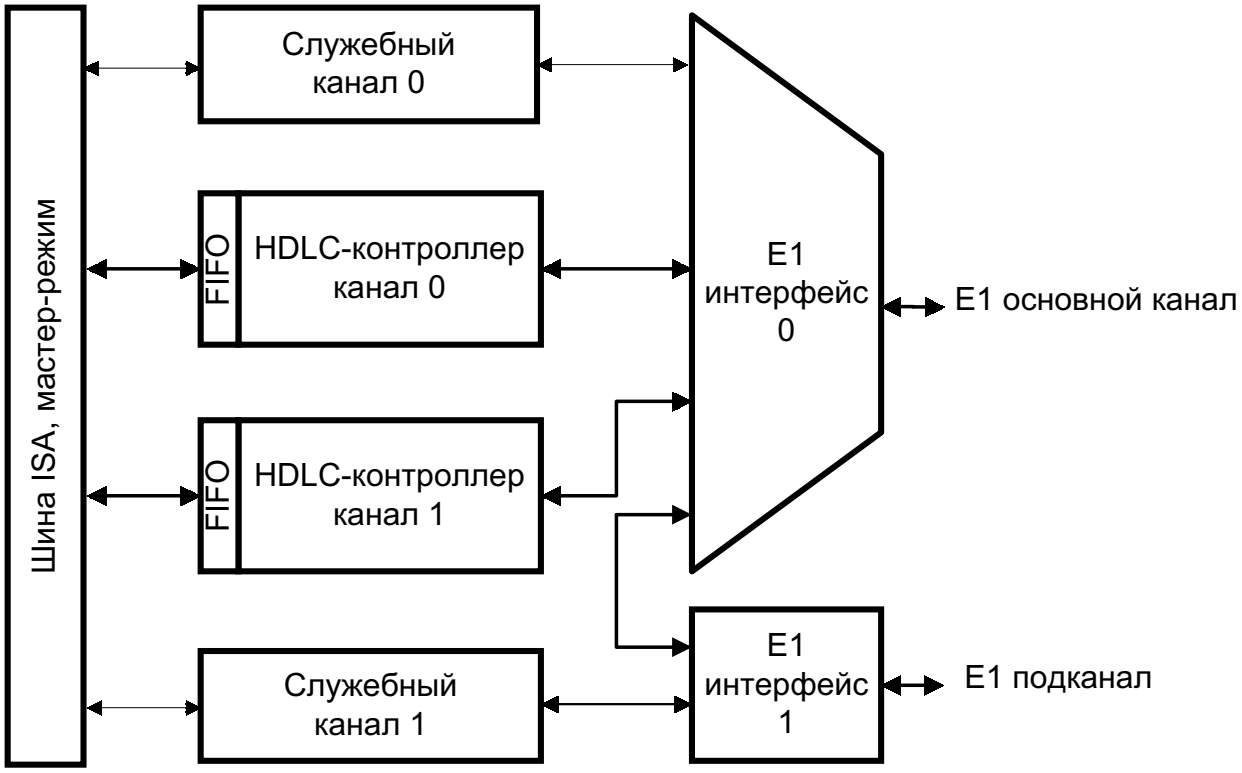


Рисунок 3: Конфигурация C — мультиплексор с двумя каналами данных и подканалом E1.

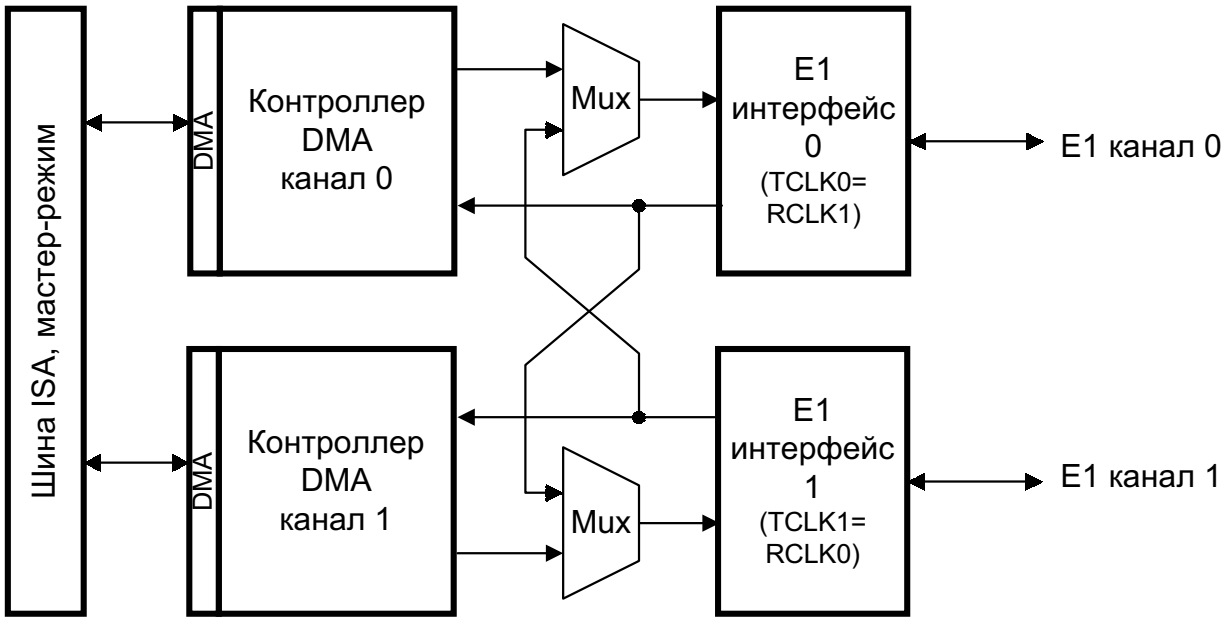


Рисунок 3: Конфигурация DI — drop-insert мультиплексор.

данных и подканалом E1-1. Два потока данных в/из памяти компьютера формируются HDLC-контроллерами 0 и 1. Остальные канальные интервалы транслируются в подканал E1-1. Эта конфигурация позволяет соединять несколько адаптеров Tau/E1 (до 30) в цепочку с расстоянием между узлами до 1.5 км. Цифровой интерфейс не используется.

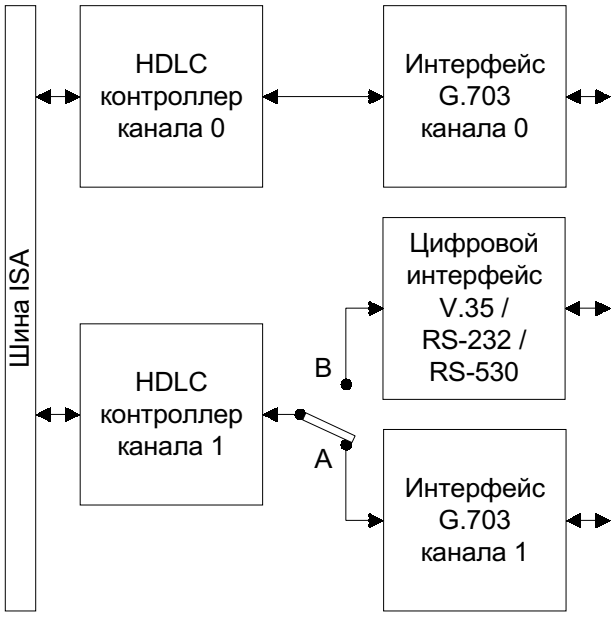
- Конфигурация D (рис.4): drop-insert мультиплексор. Выбранные канальные интервалы линий E1 поступают в память компьютера. Передаваемые данные замещают часть канальных интервалов. Остальные канальные интервалы транслируются из канала E1-0 в канал E1-1 и обратно.

При работе канала E1 накапливается статистика работы канала.

Мультиплексор Tau/G703 имеет программный переключатель, с помощью которого можно выбрать две конфигурации платы (см. рисунок).

Выбор конфигурации адаптеров Tau/E1 и Tau/G703 выполняется функцией *ct_set_config()*.

Установка скорости канала G.703 производится функцией *ct_set_baud()*. Опрос скорости каналов E1 и G.703 может производиться с помощью функции *ct_get_baud()*.



Функция ct_set_config()

```
void ct_set_config (ct_board_t *b, int mode)
```

Устанавливает конфигурацию платы.

b — Указатель на переменную состояния платы.

mode — Режим работы адаптера:

- *CFG_A* — конфигурация A. Для адаптеров E1 и G.703.
- *CFG_B* — конфигурация B. Для адаптеров E1 и G.703.
- *CFG_C* — конфигурация C. Для адаптеров E1.
- *CFG_D* — конфигурация D. Для адаптеров E1D.

Функция ct_get_config()

```
int ct_get_config (ct_chan_t *c)
```

Возвращает режим работы адаптера *CFG_A*-*CFG_D* (см. *ct_set_config()*).

c — Указатель на переменную состояния канала.

Функция ct_set_clk()

```
void ct_set_clk (ct_chan_t *c, int clk)
```

Устанавливает источник синхронизации передатчика канала E1/G.703. По умолчанию используется синхронизация от внутреннего генератора.

c — Указатель на переменную состояния канала.

clk — Источник синхронизации передатчика канала:

- *GCLK_INT* — от внутреннего генератора
- *GCLK_RCV* — от приёмника
- *GCLK_RCLKO* — от приёмника другого канала

Функция ct_get_clk()

```
int ct_get_clk (ct_chan_t *c)
```

Возвращает режим синхронизации передатчика канала E1/G.703.

c — Указатель на переменную состояния канала.

Функция ct_set_ts()

```
void ct_set_ts (ct_chan_t *c, unsigned long ts)
```

Задаёт набор канальных интервалов, выделенных потоку данных. По умолчанию используются все канальные интервалы с 1-го по 31-й.

c — Указатель на переменную состояния канала.

ts — Битовая маска - набор канальных интервалов. 1-й бит относится к 1-му канальному интервалу, 31-й бит - к 31-му, значение 0-го бита игнорируется.

Функция ct_get_ts()

```
unsigned long ct_get_ts(ct_chan_t *c)
```

Запрашивает набор канальных интервалов, выделенных соответствующему потоку данных. Возвращает битовую маску - набор канальных интервалов, как в функции *ct_set_ts()*.

c — Указатель на переменную состояния канала.

Функция ct_set_subchan()

```
void ct_set_subchan (ct_board_t *b, unsigned long ts)
```

Задаёт набор канальных интервалов, транслируемых в подканал E1 в конфигурациях B и C. По умолчанию маска транслируемых канальных интервалов пуста.

b — Указатель на переменную состояния платы.
ts — Набор канальных интервалов, как в функции *ct_set_ts()*.

Функция ct_get_subchan()

```
unsigned long ct_get_subchan (ct_board_t *b)
```

Запрашивает набор канальных интервалов, транслируемых в подканал E1 в конфигурациях B и C. Возвращает битовую маску - набор канальных интервалов, как в функции *ct_set_ts()*.

b — Указатель на переменную состояния платы.

Функция ct_set_higain()

```
void ct_set_higain (ct_chan_t *c, int on)
```

Включает/выключает режим высокой чувствительности приёмника контроллера E1. По умолчанию режим высокой чувствительности выключен.

b — Указатель на переменную состояния платы.
on — Если не равно 0 , включить режим высокой чувствительности (-36 dB), иначе выключить (-12 dB).

Функция ct_get_higain()

```
int ct_get_higain (ct_chan_t *c)
```

Запрашивает состояние режима высокой чувствительности приёмника контроллера E1. Возвращает *1* если режим высокой чувствительности включен, иначе — *0*.

b — Указатель на переменную состояния платы.

Функция ct_set_phony()

```
void ct_set_phony (ct_chan_t *c, int on)
```

Включает/выключает “телефонный” режим (для Tau/E1D). В телефонном режиме выбранные канальные интервалы циклов E1 объединяются по 16 циклов и поступают в (из) компьютер. Программное обеспечение получает возможность принимать и передавать телефонные и сигнальные данные. Длина пакетов данных в телефонном режиме равняется количеству выбранных канальных интервалов, умноженному на 16, в диапазоне от 16 до 496 байт.

b — Указатель на переменную состояния платы.
on — Если не равно 0 , включить “телефонный” режим.

Функция ct_get_phony()

```
int ct_get_phony (ct_chan_t *c)
```

Запрашивает состояние “телефонного” режима (для Tau/E1D). Возвращает *1* если режим включен, иначе — *0*.

b — Указатель на переменную состояния платы.

Функция ct_get_lq()

```
int ct_get_lq (ct_chan_t *c)
```

Измеряет уровень сигнала в линии (только для Tau/G703). Возвращает значение затухания в сантибелах. Возможные значения:

285 — затухание 28.5 dB
195 — затухание 19.5 dB
95 — затухание 9.5 dB
0 — затухание 0 dB

Примеры

Приведены тексты примеров использования DDK: тесты адаптеров Tau и Tau/E1. Тексты этих и других примеров находятся в дистрибутиве в директории examples.

Тест для адаптера Tau

Программа тестирует один канал на плате Кроникс-Tau. В канал непрерывно посылаются HDLC-пакеты. Принимаемые пакеты сравниваются с образцом. При запуске установленная плата обнаруживается с помощью функции *ct_find()* , для тестирования используется канал #0.

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

```
#include <dos.h>
#include "ctddk.h"

#define IRQ 5
#define DMA 7
#define BAUD    2048000L

ct_board_t adapter;
ct_buf_t buffer;
char data [1024];
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 (ct_chan_t *c, char *buf, int len)
{
  received += len;

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

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

/* Обработка изменения несущей */
void modem (ct_chan_t *c)
{
  if (ct_get_cd (c))
       cprintf ("\r\n--Carrier Up--\r\n");
  else
       cprintf ("\r\n--Carrier Down--\r\n");
}

/* Обработка ошибок */
void error (ct_chan_t *c, int reason)
{
  switch (reason) {
  case CT_FRAME:  cprintf ("\r\n--Framing Error--\r\n"); break;
  case CT_CRC:    cprintf ("\r\n--CRC Error--\r\n");     break;
  case CT_OVERRUN:  cprintf ("\r\n--Overrun--\r\n");       break;
  case CT_OVERFLOW: cprintf ("\r\n--Overflow--\r\n");      break;
  case CT_UNDERRUN: cprintf ("\r\n--Underrun--\r\n");      break;
  }
}
```

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

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

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

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

  /* Регистрация обработчиков событий */
  ct_register_receive (c, &receive);
  ct_register_transmit (c, &transmit);
  ct_register_error (c, &error);
  ct_register_modem (c, &modem);

  /* Установка обработчика прерывания */
  disable();
  if (! ddk_int_alloc (IRQ, &ct_int_handler, &adapter)) {
       enable();
       printf ("Irq busy\n");
       exit (-1);
  }

  /* Запуск канала */
  ct_start_chan (c, &buffer, virt_to_phys (&buffer));
  ct_led (&adapter, 1);
  ct_set_baud (c, BAUD);
  ct_set_dpll (c, 1);
  ct_set_nrzi (c, 1);
  ct_set_loop (c, 1);

  /* Устанавливаем сигналы RTS и DTR */
  ct_set_rts (c, 1);
  ct_set_dtr (c, 1);
```

```
  /* Передача первого пакета */
  transmit (c, 0, 0);
  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();
  ct_close_board (&adapter);

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

Тест для адаптера Tau/E1

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

#define IRQ 5
#define DMA 7

ct_board_t adapter;
ct_buf_t buffer;
char data [1024];
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 (ct_chan_t *c, char *buf, int len)
{
  received += len;

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

```
/* Вызывается после окончания передачи пакета */
void transmit (ct_chan_t *c, void *attachment, int len)
{
  memset (data, 'Z', sizeof(data));
  while (ct_send_packet (c, data, sizeof(data), NULL) >= 0)
       sent += sizeof(data);
}

/* Обработка изменения несущей */
void modem (ct_chan_t *c)
{
  if (ct_get_cd (c))
       cprintf ("\r\n--Carrier Up--\r\n");
  else
       cprintf ("\r\n--Carrier Down--\r\n");
}

/* Обработка ошибок */
void error (ct_chan_t *c, int reason)
{
  switch (reason) {
  case CT_FRAME:  cprintf ("\r\n--Framing Error--\r\n"); break;
  case CT_CRC:    cprintf ("\r\n--CRC Error--\r\n");     break;
  case CT_OVERRUN:  cprintf ("\r\n--Overrun--\r\n");       break;
  case CT_OVERFLOW: cprintf ("\r\n--Overflow--\r\n");      break;
  case CT_UNDERRUN: cprintf ("\r\n--Underrun--\r\n");      break;
  }
}

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

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

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

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

```
    if (c->mode != M_E1) {
         printf ("No E1 channels detected\n");
         exit (-1);
    }
    printf ("Starting E1 channel %d, timeslots 1-31\n", c->num);

    /* Регистрация обработчиков событий */
    ct_register_receive (c, &receive);
    ct_register_transmit (c, &transmit);
    ct_register_error (c, &error);
    ct_register_modem (c, &modem);

    /* Установка обработчика прерывания */
    disable();
    if (! ddk_int_alloc (IRQ, &ct_int_handler, &adapter)) {
         enable();
         printf ("Irq busy\n");
         exit (-1);
    }

    /* Запуск канала */
    ct_start_chan (c, &buffer, virt_to_phys (&buffer));
    ct_led (&adapter, 1);

    /* Частота передачи E1 - внутренний генератор.  */
    ct_set_clk (c, GCLK_INT);

    /* Выбираем канальные интервалы 1..31 */
    ct_set_ts (c, ~0UL ^ 1UL);

    /* Передача первого пакета */
    transmit (c, 0, 0);
    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();
    ct_close_board (&adapter);

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