Набор разработчика драйверов (DDK) для адаптеров семейства Cronyx Tau-PCI

Октябрь 2003 *Версия 4.3.3-1*

Содержание

Список функций 6

Файлы 9

Структуры данных 10

Структура cp_board_t 10

Структура cp_chan_t 11

Структура cp_gstat_t 14

Инициализация 16

Константы cp_device_id, cp_vendor_id 16

Функция cp_init() 16

Функция cp_reset() 18

Функция cp_hard_reset() 18

Обработка событий 20

Функция cp_register_transmit() 20

Функция cp_register_receive() 21

Функция cp_register_error() 21

Пуск канала 23

Функция cp_start_chan() 23

Функция cp_stop_chan() 23

Функция cp_start_e1() 24

Функция cp_stop_e1() 24

Передача данных 25

Функция cp_send_packet() 25

Функция cp_transmit_space() 26

Функция cp_handle_interrupt() 26

Функция cp_enable_interrupt() 26

Функция cp_interrupt_poll() 27

Функция cp_interrupt() 27
Функция cp_led() 27

Сбор статистики 29

Функция cp_g703_timer() 29
Функция cp_e1_timer() 29
Функция cp_e3_timer() 29

Установка и опрос параметров канала 30

Функция cp_set_baud() 30
Функция cp_set_dpll() 30
Функция cp_set_nrzi() 31
Функция cp_set_invtxc() 31
Функция cp_set_invrxc() 31
Функция cp_get_txcerr() 31
Функция cp_get_rxcerr() 31
Функция cp_set_lloop() 32
Функция cp_get_rloop() 32
Функция cp_get_cable() 32

Модемные сигналы 33

Функция cp_set_dtr() 33
Функция cp_set_rts() 33
Функция cp_get_dsr() 33
Функция cp_get_cts() 33
Функция cp_get_cd() 34

Параметры каналов E1 и G.703 35

Функция cp_set_mux() 37
Функция cp_set_dir() 37
Функция cp_set_gsyn() 38
Функция cp_set_ts() 38
Функция cp_set_higain() 38

Функция cp_set_monitor() 39
Функция cp_set_use16() 39
Функция cp_set_crc4() 39
Функция cp_set_phony() 40
Функция cp_set_unfram() 40
Функция cp_set_scrambler() 40
Функция cp_get_lq() 41

Параметры каналов E3 42

Функция cp_set_gsyn() 42
Функция cp_set_rloop() 42
Функция cp_set_ber() 42
Функция cp_set_losais() 43

Примеры 44

Тест для адаптера Tau-PCI 44
Тест для адаптера Tau-PCI-E1 47

Набор разработчика для адаптеров семейства Tau-PCI 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 6-ти). Набор тестировался с компилятором GNU C (Linux и FreeBSD) и MS Visual Studio (6.0, NET). Примеры в тексте даны для ОС Linux (ядро 2.2.x).

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:

cp_device_id — идентификатор устройства
cp_vendor_id — идентификатор производителя
cp_init() — инициализация структуры данных адаптера
cp_reset() — программный сброс адаптера
cp_hard_reset() — аппаратный сброс адаптера

Установка обработчиков:

cp_register_transmit() — установка обработчика передачи пакета
cp_register_receive() — установка обработчика приёма пакета
cp_register_error() — установка обработчика ошибок

Пуск канала:

cp_start_chan() — запуск канала
cp_stop_chan() — останов канала
cp_start_e1() — запуск контроллера E1
cp_stop_e1() — останов контроллера E1

Передача данных:

cp_send_packet() — передача пакета
cp_transmit_space() — проверка свободного места в буфере передатчика
cp_enable_interrupt() — включение/выключение прерываний HDLC контроллера
cp_handle_interrupt() — обработка прерывания
cp_interrupt_poll() — проверка и сброс статуса прерывания
cp_interrupt() — обработка прерывания и сброс статуса прерывания
cp_led() — управление светодиодом на адаптерах E1 и G.703

Сбор статистики:

cp_g703_timer() — накопление статистики G.703
cp_e1_timer() — накопление статистики E1
cp_e3_timer() — накопление статистики E3

Установка и опрос параметров канала:

cp_set_baud() — выбор частоты генератора синхросигнала
cp_set_dpll() — выключение/включение DPLL

cp_set_nrzi() — выключение/включение кодирования NRZI
cp_set_invtxc() — выключение/включение инвертирования синхроимпульсов передатчика
cp_set_invrxc() — выключение/включение инвертирования синхроимпульсов приемника
cp_get_rxcerr() — опрос ошибки синхросигнала RXCIN
cp_get_txcerr() — опрос ошибки синхросигнала TXCIN
cp_set_lloop() — выключение/включение локального шлейфа
cp_get_rloop() — опрос состояния включения удаленного шлейфа
cp_get_cable() — опрос типа кабеля

Модемные сигналы:

cp_set_dtr() — установка сигнала DTR
cp_set_rts() — установка сигнала RTS
cp_get_dsr() — опрос сигнала DSR
cp_get_cts() — опрос сигнала CTS
cp_get_cd() — опрос сигнала CD

Параметры каналов E1/G.703:

cp_set_mux() — выбор конфигурации адаптера E1
cp_set_dir() — выбор линии E1 для данного канала
cp_set_gsyn() — выбор источника синхронизации канала G.703/E1
cp_set_scrambler() — включение/выключение скремблера (G.703)
cp_set_ts() — выбор канальных интервалов E1
cp_set_higain() — включение/выключение высокой чувствительности приёмника E1 (нелинейное усиление)
cp_set_monitor() — включение/выключение высокой чувствительности приёмника E1 (линейное усиление)
cp_set_use16() — включение/выключение режима сверхцикловой синхронизации CAS (E1)
cp_set_crc4() — включение/выключение режима контрольной суммы CRC4 (E1)
cp_set_phony() — включение/выключение “телефонного” режима E1
cp_set_unfram() — включение/выключение цикловой структуры (G.703/E1)
cp_get_lq() — опрос уровня сигнала в линии (G.703)

Параметры E3:

cp_set_gsyn() — выбор источника синхронизации канала E3
cp_set_rloop() — включение/выключение удаленного шлейфа

cp_set_ber() — включение/выключение ber-тестера
cp_set_losais() — включение/выключение выдачи AIS на LOS

Устаревшие функции:

cp_register_modem() — установка обработчика изменения модемных сигналов
Более не существует.

cp_set_gmode() — выбор конфигурации адаптера E1
Заменена функцией *cp_set_mux()*.

cp_set_precoder() — включение/выключение скремблера (G.703)
Переименована в *cp_set_scrambler()*.

cp_set_subts() — выбор канальных интервалов подканала E1
Более не существует.

Файлы

Набор разработчика состоит из следующих файлов:

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

ctaupci.h, peb20534.h, ds2153.h, lxt318.h — Описания интерфейса используемых контроллеров.

ctp2efw.h — Микропрограмма для адаптера Tau-PCI-2E1

ctp4efw.h — Микропрограмма для адаптера Tau-PCI-4E1

machdep.h — Машинно-зависимые определения.

ctddk.c — Код процедур пакета.

Для написания своих собственных программ с использованием стандартной поставки DDK необходимы только файлы cpddk.c и cpddk.h. Константы в файле cpddk.h не должны изменяться. Их изменение может привести к сбоям в работе драйвера и как следствия сбоям в системе.

В каталоге examples находятся примеры использования DDK:

test.c — Пример для адаптера Tau-PCI.

teste1.c — Пример для адаптера Tau-PCI-E1.

testg703.c — Пример для адаптера Tau-PCI-G.703.

Структуры данных

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

Структура cp_board_t

```
typedef struct {
  unsigned char type;
  char name[16];
  unsigned char num;
  unsigned char *base;
  cp_chan_t chan[NCHAN];
  unsigned char mux;
  void *sys;
  ...
} cp_board_t;
```

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

unsigned char type; — тип адаптера:

- *B_TAUPCI* — Модель Tau-PCI
- *B_TAUPCI_E3* — Модель Tau-PCI-E3
- *B_TAUPCI_G703* — Модель Tau-PCI-G703
- *B_TAUPCI_E1* — Модель Tau-PCI-E1
- *B_TAUPCI4* — Модель Tau-PCI4
- *B_TAUPCI4_G703* — Модель Tau-PCI4-G703/X
- *B_TAUPCI4_E1* — Модель Tau-PCI4-E1/X
- *B_TAUPCI_2E1* — Модель Tau-PCI-2E1
- *B_TAUPCI4_4E1* — Модель Tau-PCI4-4E1
- *B_TAUPCI_HSSI* — Модель Tau-PCI-HSSI

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=6

— максимальное количество адаптеров. Номер

задаётся при инициализации адаптера.
*unsigned char *base;* — базовый адрес регистров адаптера.
cp_chan_t chan[NCHAN]; — структуры данных каналов (NCHAN=4).
unsigned char mux; — конфигурация адаптера (для моделей E1)
*void *sys;* — данные пользователя.

Структура cp_chan_t

```
typedef struct {
  unsigned char type;
  unsigned char num;
  cp_board_t *board;
  unsigned char dtr;
  unsigned char rts;
  void *sys;
  int debug;
  unsigned long baud;
  unsigned char dpll;
  unsigned char nrzi;
  unsigned char invtxc;
  unsigned char invrxc;
  unsigned char lloop;
  unsigned char gsyn;
  unsigned char scrambler;
  unsigned long ts;
  unsigned char higain;
  unsigned char monitor;
  unsigned char unfram;
  unsigned char dir;
  unsigned char use16;
  unsigned char crc4;
  unsigned char phony;
  unsigned char losais;
  unsigned char ber;
  unsigned long rintr;
  unsigned long tintr;
  unsigned long long ibytes;
  unsigned long long obytes;
  unsigned long ipkts;
  unsigned long opkts;
  unsigned long underrun;
  unsigned long overrun;
  unsigned long frame;
  unsigned long crc;
  unsigned short status;
  unsigned long totsec;
```

```
        unsigned long cursec;
        cp_gstat_t currnt;
        cp_gstat_t total;
        cp_gstat_t interval [48];
        unsigned long e3status;
        unsigned long e3csec_5;
        unsigned long e3tsec;
        unsigned long e3ccv;
        unsigned long e3tcv;
        unsigned long e3icv[48];
        ...
    } cp_chan_t;
```

unsigned char type; — тип канала:
- *T_NONE* — канал отсутствует
- *T_SERIAL* — V.35 / RS-530 / RS-232
- *T_E1* — E1 (ИКМ-30)
- *T_G703* — G.703.6 (2048 кбит/сек)
- *T_E3* — E3
- *T_T3* — T3
- *T_STS1* — STS1
- *T_HSSI* — HSSI
- *T_DATA* — дополнительные подканалы E1

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

*cp_board_t *board;* — указатель на структуру данных адаптера

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 baud; — скорость передачи данных (бит/сек), или 0 при внешней синхронизации

unsigned char dpll; — режим DPLL

unsigned char nrzi; — режим NRZI

unsigned char invtxc; — инвертирование синхросигнала передатчика

unsigned char invrxc; — инвертирование синхросигнала приемника

unsigned char lloop; — локальный шлейф

unsigned char gsyn; — режим синхронизации E1/G.703:
- *GSYN_INT* — от внутреннего генератора
- *GSYN_RCV* — от приёмника

GSYN_RCV0 — от приёмника канала 0
GSYN_RCV1 — от приёмника канала 1
GSYN_RCV2 — от приёмника канала 2
GSYN_RCV3 — от приёмника канала 3

unsigned char scrambler; — режим скремблирования G.703
unsigned long ts; — маска канальных интервалов канала E1
unsigned char higain; — режим (нелинейного) усиления приемника E1
unsigned char monitor; — режим (линейного) усиления приемника E1
unsigned char unfram; — режим без цикловой структуры E1
unsigned char dir; — линия E1, с которой связан данный канал
unsigned char use16; — режим сверхцикловой синхронизации CAS (E1)
unsigned char crc4; — режим контрольной суммы CRC4 (E1)
unsigned char phony; — “телефонный” режим E1
unsigned char losais; — выдача AIS на LOS (E3)
unsigned char ber; — режим BER-тестера (E3)
unsigned long rintr; — счетчик прерываний приемника
unsigned long tintr; — счетчик прерываний передатчика
ulong64 ibytes; — счетчик принятых байтов
ulong64 obytes; — счетчик переданных байтов
unsigned long ipkts; — счетчик принятых пакетов
unsigned long opkts; — счетчик переданных пакетов
unsigned long ierrs; — счетчик ошибок приемника
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*)
cp_gstat_t currnt; — статистика G.703/E1 за текущий период (до 15 минут)
cp_gstat_t total; — общая статистика G.703/E1
cp_gstat_t interval [48]; — статистика G.703/E1 за последние 48 периодов (12 часов)
unsigned short status; — состояние линии E3
E3STS_LOS— потеря синхронизации E3
E3STS_TXE— ошибка передатчика E3
unsigned long e3csec_5; — длительность текущего периода в 1/5 секунд (см e3_ccv) (E3)
unsigned long e3tsec; — общее время работы канала E3 в секундах
unsigned long e3ccv; — статистика нарушений кодирования E3 за текущий период (до 15 минут)
unsigned long e3ecv; — общее количество нарушения кодирования E3
unsigned long e3icv[48]; — статистика нарушений кодирования E3 за последние 48 секундных периодов (до 12 часов)

Структура cp_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;
} cp_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)

Инициализация

Константы cp_device_id, cp_vendor_id

```
unsigned short cp_vendor_id, cp_device_id;
```

Идентификаторы устройства и производителя для поиска устройства.

Функция cp_init()

```
unsigned short
  cp_init (cp_board_t *b, int num, unsigned char *base)
```

Производит инициализацию структуры адаптера. В случае успешной инициализации возвращается 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=6). Используется для различения нескольких установленных адаптеров.

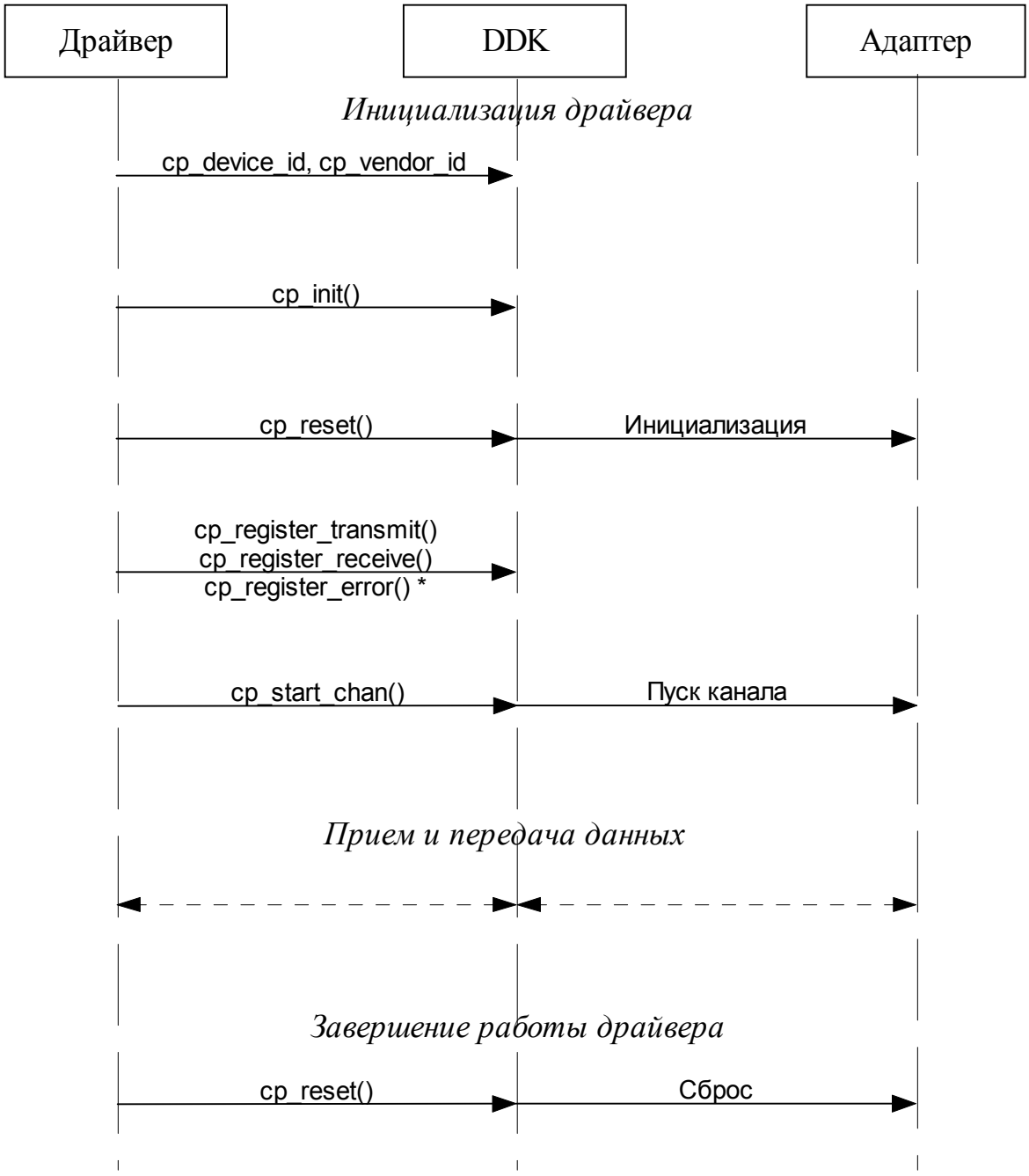
base — Базовый адрес регистров адаптера (виртуальный).

Базовый адрес определяется из регистров PCI адаптера.

Пример:

```
main()
{
  cp_board_t adapter;
  struct pci_dev *dev;
  unsigned long flags;

  dev = pci_find_device (cp_vendor_id, cp_device_id, 0);
  if (! dev) {
      printk ("adapter not found\n");
      return -ENXIO;
  }
  save_flags (flags);
  cli ();
  cp_init (&adapter, 0,
           ioremap (dev->base_address[0], 0x800));
  ...
```



* — необязательные вызовы

Функция cp_reset()

```
void cp_reset (cp_board_t *b, cp_qbuf_t *buf,
               unsigned long phys)
```

Производит сброс адаптера и переводит его в неактивное состояние, инициализирует очередь прерываний. Должна вызываться после инициализации и при окончании работы с адаптером. Перед вызовом следует выделить физически непрерывный участок памяти для структуры *cp_qbuf_t* и передать его виртуальный адрес в качестве параметра *buf*, и физический адрес - в качестве параметра *phys*. При последующих вызовах *cp_reset()* можно в качестве *buf* и *phys* передавать 0.

b — Структура данных адаптера, инициализированная функцией *cp_init()*.
buf — Буфер данных для очереди прерываний адаптера.
phys — Физический адрес структуры *buf*.

Пример:

```
...
cp_qbuf_t *queue;

queue = kmalloc (sizeof (*queue), GFP_KERNEL);
if (! queue) {
    printk ("out of memory\n");
    return -ENOMEM;
}
...
cp_init (&adapter, 0,
         ioremap (dev->base_address[0], 0x800));
cp_reset (&adapter, queue, virt_to_phys (queue));
...
```

Функция cp_hard_reset()

```
int cp_hard_reset (cp_board_t *b)
```

Производит аппаратный сброс адаптера. После сброса требуется пауза в 12 микросекунд. При сбросе адаптер теряет содержимое регистров конфигурации PCI, и для продолжения работы требуется их восстановление.

b — Структура данных адаптера.

Пример:

```
unsigned int stacmd, clsiz, base1, base2, intlin;

pci_read_config_dword (dev, PCI_COMMAND, &stacmd);
pci_read_config_dword (dev, PCI_CACHE_LINE_SIZE,
```

```
                        &clsiz);
pci_read_config_dword  (dev,  PCI_BASE_ADDRESS_0,
                        &base1);
pci_read_config_dword  (dev,  PCI_BASE_ADDRESS_1,
                        &base2);
pci_read_config_dword  (dev,  PCI_INTERRUPT_LINE,
                        &intlin);
cp_hard_reset  (b);
udelay  (12);
pci_write_config_dword  (dev,  PCI_COMMAND,  stacmd);
pci_write_config_dword  (dev,  PCI_CACHE_LINE_SIZE,
                         clsiz);
pci_write_config_dword  (dev,  PCI_BASE_ADDRESS_0,
                         base1);
pci_write_config_dword  (dev,  PCI_BASE_ADDRESS_1,
                         base2);
pci_write_config_dword  (dev,  PCI_INTERRUPT_LINE,
                         intlin);
cp_reset  (b,  0,  0);
...
```

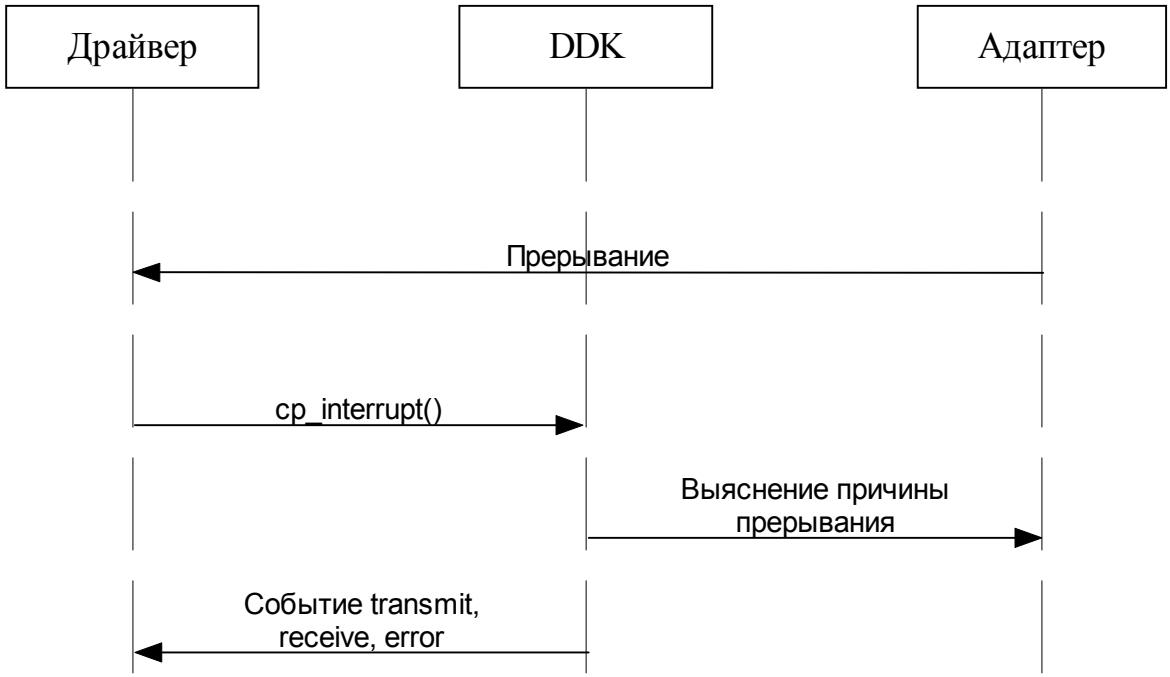
Обработка событий

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

• Прерывание по приему пакета.

• Прерывание по завершению передачи пакета.

• Прерывание по ошибке приема или передачи.



Функция cp_register_transmit()

```
void cp_register_transmit (cp_chan_t *c,
  void (*func)(cp_chan_t *c, void *attachment, int len))
```

Функция-обработчик вызывается при успешном завершении передачи пакета.

Аргументы, передаваемые обработчику:

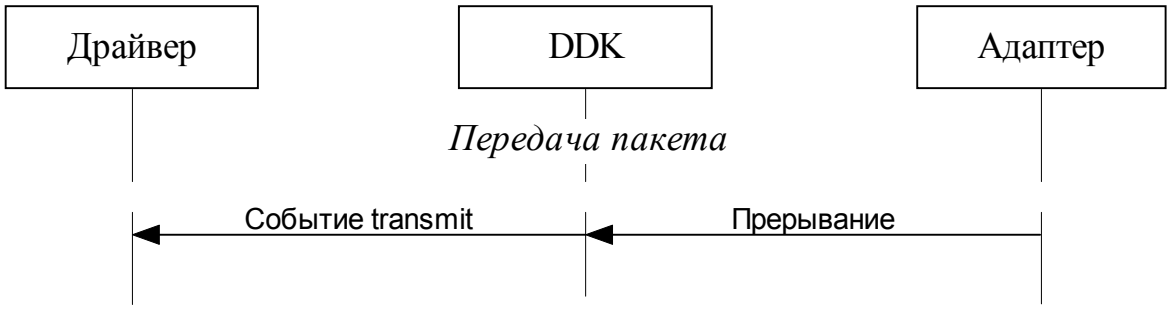
c — Указатель на структуру канала

func — Функция - обработчик события

attachment — Аргумент соответствующего вызова функции *cp_send_packet()*, может использоваться драйвером

для передачи ссылки на системно-зависимую структуру данных, относящуюся к пакету.

len — Длина пакета в байтах



Функция cp_register_receive()

```
void cp_register_receive (cp_chan_t *c,
  void (*func) (cp_chan_t *c, char *data, int len))
```

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

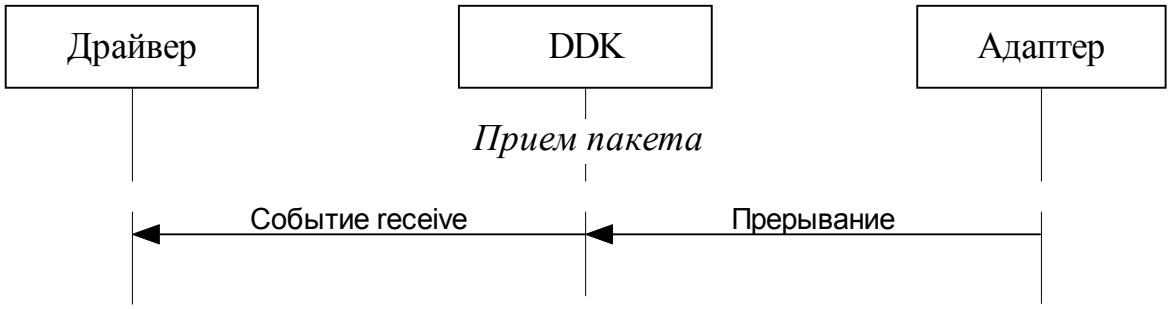
Аргументы, передаваемые обработчику:

c — Указатель на структуру канала

func — Функция - обработчик события

data — Указатель на буфер данных

len — Длина пакета в байтах



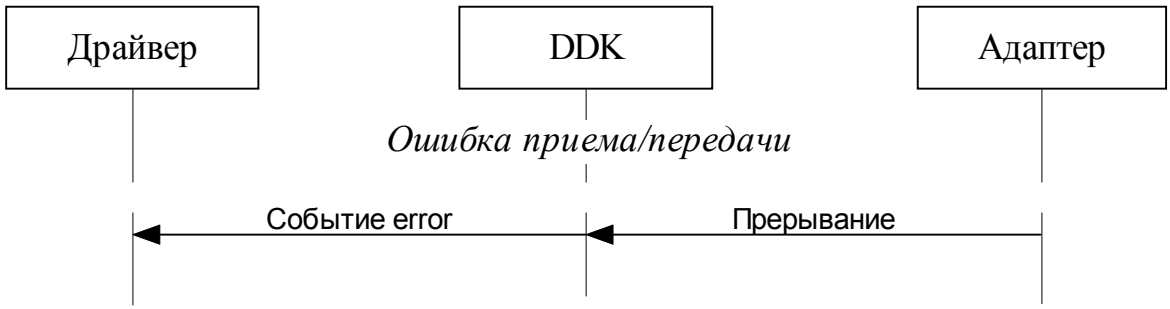
Функция cp_register_error()

```
void cp_register_error (cp_chan_t *c,
  void (*func)(cp_chan_t *c, int data))
```

Функция-обработчик вызывается при обнаружении ошибки.

Аргументы, передаваемые обработчику:

c — Указатель на структуру канала
func — Функция - обработчик события
data — Код ошибки:
CP_FRAME — Ошибка кадра
CP_CRC — Ошибка контрольной суммы
CP_UNDERRUN — Опустошение FIFO передатчика
CP_OVERRUN — Переполнение FIFO приёмника
CP_OVERFLOW — Переполнение буфера приемника



Пуск канала

Функция cp_start_chan()

```
void cp_start_chan (cp_chan_t *c, int tx, int rx,
                    cp_buf_t *buf, unsigned long phys)
```

Запускает канал, сбрасывает сигналы DTR и RTS в 0. Пуском приемника и передатчика можно управлять при помощи флагов *rx* и *tx*. Перед вызовом следует выделить физически непрерывный участок памяти для структуры *cp_buf_t* и передать его виртуальный адрес в качестве параметра *buf*, и физический адрес - в качестве параметра *phys*. При последующих вызовах *cp_start_chan()* можно в качестве *buf* и *phys* передавать 0.

c — Указатель на переменную состояния канала.
rx — Флаг запуска приемника.
tx — Флаг запуска передатчика.
buf — Указатель на область памяти, предназначенную для буферов приёма-передачи.
phys — Физический адрес области памяти *buf*.

Пример:

```
main()
{
  cp_board_t adapter;
  cp_buf_t *buf;

  ...
  buf = kmalloc (sizeof (*buf), GFP_KERNEL);
  if (! buf) {
      printk ("out of memory\n");
      return -ENOMEM;
  }
  ...
  c = &adapter.chan[0];
  cp_start_chan (c, 1, 1, buf, virt_to_phys (buf));
  ...
```

Функция cp_stop_chan()

```
void cp_stop_chan (cp_chan_t *c)
```

Останавливает работу канала.

c — Указатель на переменную состояния канала.

Функция cp_start_e1()

```
void cp_start_e1 (cp_chan_t *c)
```

Запускает контроллер E1 указанного канала. Рекомендуется вызывать до запуска канала.

c — Указатель на переменную состояния канала.

Пример:

```
...
  cp_start_e1 (c);
  cp_start_chan (c, 1, 1, buf, virt_to_phys (buf));
...
```

Функция cp_stop_e1()

```
void cp_stop_e1 (cp_chan_t *c)
```

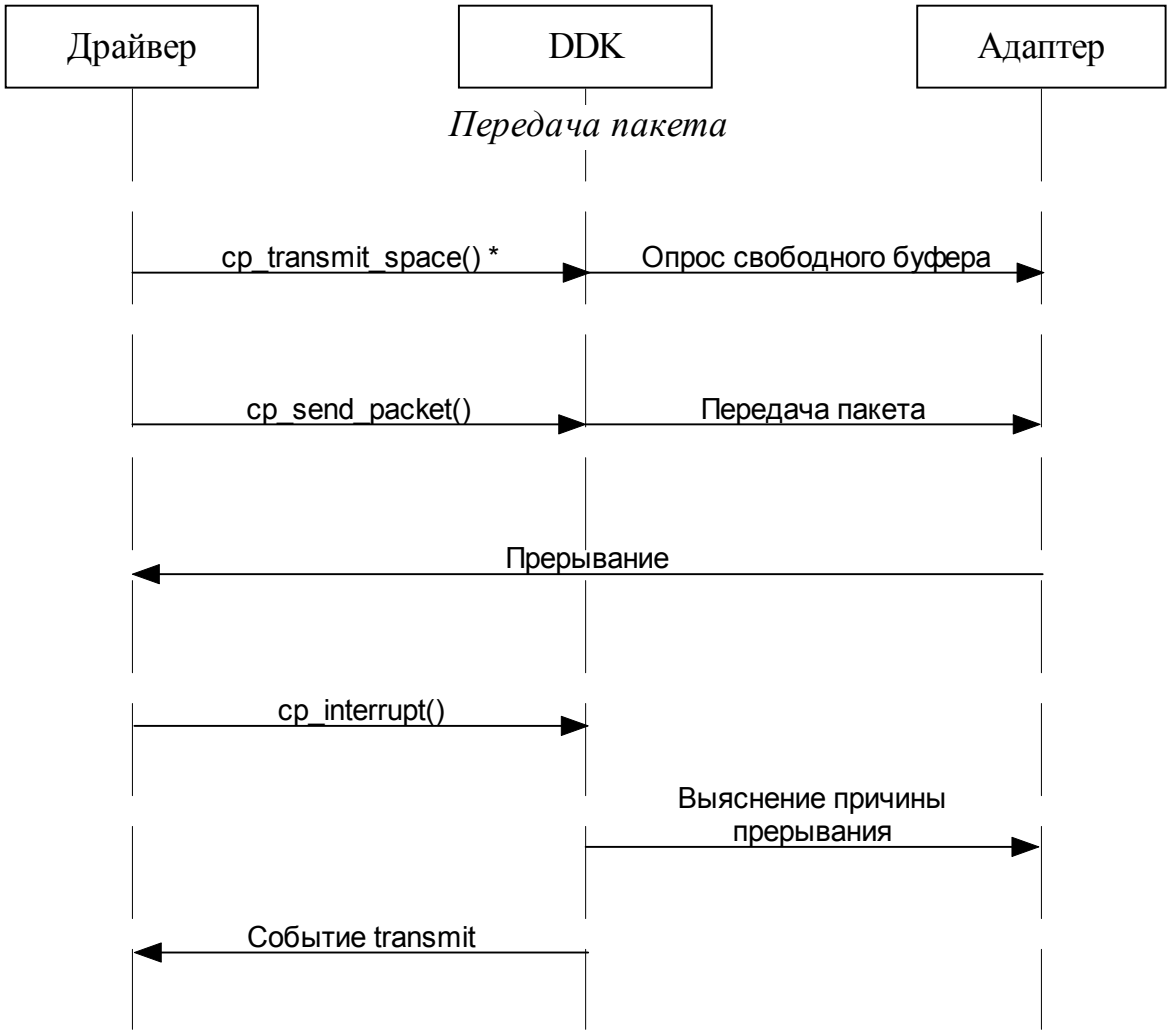
Останавливает контроллер E1 указанного канала. Рекомендуется вызывать после останова канала.

c — Указатель на переменную состояния канала.

Пример:

```
...
  cp_stop_chan (c);
  cp_stop_e1 (c);
...
```

Передача данных



Функция cp_send_packet()

```
int cp_send_packet (cp_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 *cp_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. *cp_transmit_space()*).

-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 cp_transmit_space()

```
int cp_transmit_space (cp_chan_t *c)
```

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

c — Указатель на структуру канала

Функция cp_handle_interrupt()

```
void cp_handle_interrupt (cp_board_t *b)
```

Обработка аппаратного прерывания от адаптера. Пользователь DDK должен обеспечить вызов обработчика прерывания при его возникновении. Функции *cp_handle_interrupt()* должен быть передан указатель на структуру состояния платы, вызвавшей прерывание. Функция не делает проверок о том, что прерывание действительно было возбужденно данным адаптером. См. также описание *cp_interrupt()* и *cp_interrupt_poll()*.

b — Структура данных адаптера

Функция cp_enable_interrupt()

```
void cp_enable_interrupt (cp_board_t *b, int on)
```

Запрещает или разрешает прерывания HDLC контроллеров каналов указанного адаптера. Может применяться, если необходимо запретить выставление прерываний на шине PCI с момента обнаружения прерывания и сброса его на шине PCI до его обработки. Существуют другие типы прерываний, которые будут вызывать индикацию на шине PCI при их возникновении, например, секундное прерывание контроллера E1.

b — Структура данных адаптера
on — Если равно 0 замаскировать прерывания, иначе разрешить их

Функция cp_interrupt_poll()

```
int cp_interrupt_poll (cp_board_t *b, int ack)
```

Проверка и/или сброс статуса прерывания. Возвращает значение отличное от 0 в случае, если прерывание было вызвано указанным адаптером. Если указано, происходит также очистка прерывания на шине PCI. См. также описание *cp_interrupt()*.

b — Структура данных адаптера

ack — Если отлично от 0, произвести также сброс статуса прерывания на шине PCI.

Функция cp_interrupt()

```
int cp_interrupt (cp_board_t *b)
```

Обработка прерывания от адаптера. Примерный текст этой функции приведен ниже:

```
int cp_interrupt (cp_board_t *b)
{
  int loopcount = 0;
  while (cp_interrupt_poll (b, 1) != 0) {
      /*
       * Stop if the interrupt rate is too high,
       * or we could never leave the interrupt handler.
       */
      if (++loopcount > 1000)
          return -1;

      cp_handle_interrupt (b);
  }
  return loopcount;
}
```

Функция производит проверку и автоматически сбрасывает статус прерывания на шине PCI. Возвращаемое значение отлично от нуля, в случае если было хотя бы одно прерывание от данного адаптера. См. также описание *cp_handle_interrupt()*.

b — Структура данных адаптера

Функция cp_led()

```
void cp_led (cp_board_t *b, int on)
```

Адаптеры имеют светодиод, который может быть использован программным обеспечением для индикации работы адаптера. Светодиод включается / вык-

лючается функцией *cp_led()* .

b — Структура данных адаптера

on — Если отлично от 0, включить светодиод, иначе выключить светодиод.

Сбор статистики

Функция cp_g703_timer()

```
void cp_g703_timer (cp_chan_t *c)
```

Накопление статистики канала G.703. Только для адаптера Tau-PCI-G703. Пользователь DDK должен обеспечить ежесекундный вызов этой функции для всех каналов типа T_G703.

c — Указатель на структуру канала

Функция cp_e1_timer()

```
void cp_g703_timer (cp_chan_t *c)
```

Накопление статистики канала E1. Только для адаптеров Tau-PCI-E1, Tau-PCI-2E1 и Tau-PCI-4E1. Пользователь DDK должен обеспечить ежесекундный вызов этой функции для всех каналов типа T_E1.

c — Указатель на структуру канала

Функция cp_e3_timer()

```
void cp_e3_timer (cp_chan_t *c)
```

Накопление статистики канала E3. Только для адаптера Tau-PCI-E3. Пользователь DDK должен обеспечить вызов этой функции **каждые 200 миллисекунд** для всех каналов типа T_E3.

c — Указатель на структуру канала

Установка и опрос параметров канала

Функция cp_set_baud()

```
void cp_set_baud (cp_chan_t *c, unsigned long baud)
```

Для каналов V.35/RS-530/RS-232/X.21 — 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-PCI-G703) — управление скоростью данных. Значение *baud* может равняться 2048000, 1024000, 512000, 256000, 128000 или 64000 (бит/сек).

Для каналов E1 (Tau-PCI-E1) функция *cp_set_baud()* неприменима, изменение скорости передачи производится функцией *cp_set_ts()*.

Для каналов E3 скорость фиксирована и изменяться не может.

Текущее значение скорости доступно как поле *baud* структуры канала *cp_chan_t*.

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 cp_set_dpll()

```
void cp_set_dpll (cp_chan_t *c, int on)
```

Включает / выключает режим синхронизации DPLL. Для режима DPLL требуется переключение на внутреннюю синхронизацию. Используется в синхронном режиме передачи данных при отсутствии входных синхросигналов. Во избежание потери синхронизации рекомендуется применять режим DPLL совместно с кодированием NRZI. По умолчанию режим DPLL выключен. Текущее значение режима доступно как поле *dpll* структуры канала *cp_chan_t*.

c — Указатель на переменную состояния канала

on — Не равно 0 — включить DPLL, иначе выключить

Функция cp_set_nrzi()

```
void cp_set_nrzi (cp_chan_t *c, int nrzi)
```

Переключает режим кодирования, NRZ (по умолчанию) или NRZI. Текущее значение режима доступно как поле *nrzi* структуры канала *cp_chan_t*.

c — Указатель на переменную состояния канала

on — Если не равно 0, устанавливается кодирование NRZI, иначе NRZ

Функция cp_set_invtxc()

```
void cp_set_invtxc (cp_chan_t *c, int on)
```

Включает / выключает инвертирование синхроимпульсов передатчика. По умолчанию инвертирование синхроимпульсов выключено. Текущее значение режима доступно как поле *invtxc* структуры канала *cp_chan_t*.

c — Указатель на переменную состояния канала.

on — Если не равно 0, включить , иначе выключить.

Функция cp_set_invrxc()

```
void cp_set_invrxc (cp_chan_t *c, int on)
```

Включает / выключает инвертирование синхроимпульсов приемника. По умолчанию инвертирование синхроимпульсов выключено. Текущее значение режима доступно как поле *invrxc* структуры канала *cp_chan_t*.

c — Указатель на переменную состояния канала.

on — Если не равно 0, включить , иначе выключить.

Функция cp_get_txcerr()

```
int cp_get_txcerr (cp_chan_t *c)
```

Проверяет наличие импульсов синхросигнала TXCIN указанного канала. В случае ошибки возвращает значение отличное от 0.

c — Указатель на переменную состояния канала.

Функция cp_get_rxcerr()

```
int cp_get_rxcerr (cp_chan_t *c)
```

Проверяет наличие импульсов синхросигнала RXCIN указанного канала. В случае ошибки возвращает значение отличное от 0.

c — Указатель на переменную состояния канала.

Функция cp_set_lloop()

```
int cp_set_lloop (cp_chan_t *c, int on)
```

Включает/выключает локальный шлейф. Передаваемые данные заворачиваются на вход приемника. Текущее значение режима доступно как поле *lloop* структуры канала *cp_chan_t*.

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

Функция cp_get_rloop()

```
int cp_get_rloop (cp_chan_t *c)
```

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

c — Указатель на переменную состояния канала

Функция cp_get_cable()

```
int cp_get_cable (cp_chan_t *c)
```

Опрос типа подключенного кабеля. Возвращает:

CABLE_RS232 — RS-232 или кабель не подключен
CABLE_V35 — V.35
CABLE_RS530 — RS-530
CABLE_X21 — X.21
CABLE_COAX — коаксиальный кабель
CABLE_TP — витая пара

c — Указатель на переменную состояния канала

Модемные сигналы

При пуске канала сигналы DTR и RTS сбрасываются. В дальнейшем их можно изменить с помощью функций *cp_set_dtr()* и *cp_set_rts()*. Опрос сигналов DSR, CTS и DCD производится функциями *cp_get_dsr()*, *cp_get_cts()* и *cp_get_cd()*. Все функции кроме *cp_get_cd()*, только для портов адаптеров Tau-PCI, Tau-PCI/R и платы расширения Delta2.

Функция cp_set_dtr()

```
void cp_set_dtr (cp_chan_t *c, int on)
```

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

c — Указатель на структуру канала

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

Функция cp_set_rts()

```
void cp_set_rts (cp_chan_t *c, int on)
```

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

c — Указатель на структуру канала

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

Функция cp_get_dsr()

```
int cp_get_dsr (cp_chan_t *c)
```

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

c — Указатель на структуру канала

Функция cp_get_cts()

```
int cp_get_cts (cp_chan_t *c)
```

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

c — Указатель на структуру канала

Функция cp_get_cd()

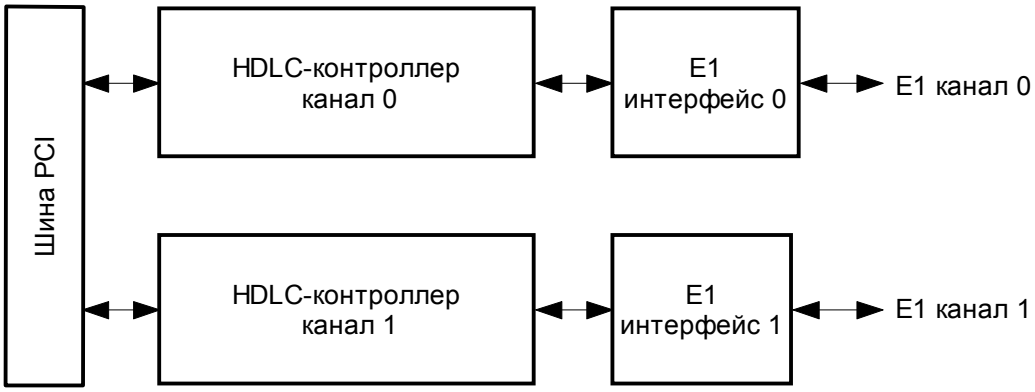
```
int cp_get_cd (cp_chan_t *c)
```

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

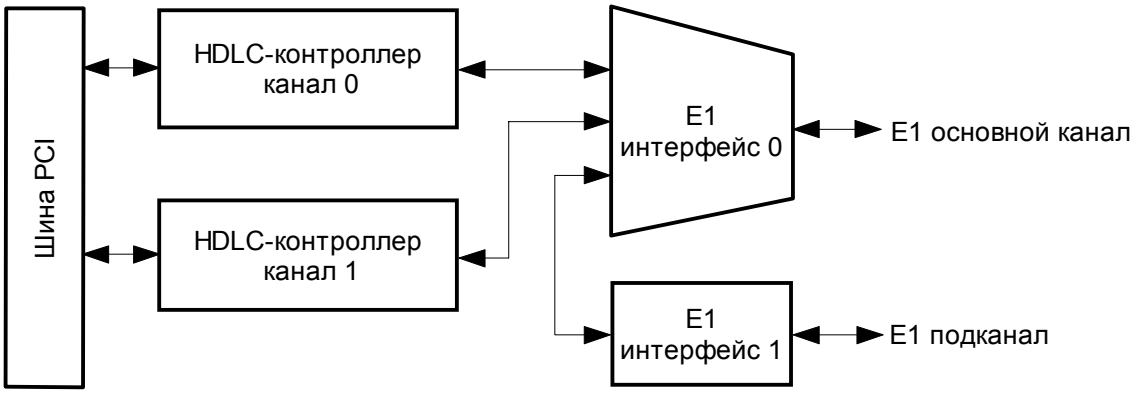
c — Указатель на структуру канала

Параметры каналов E1 и G.703

Мультиплексор Tau-PCI-E1 имеет программный переключатель, с помощью которого можно выбрать две конфигурации платы:



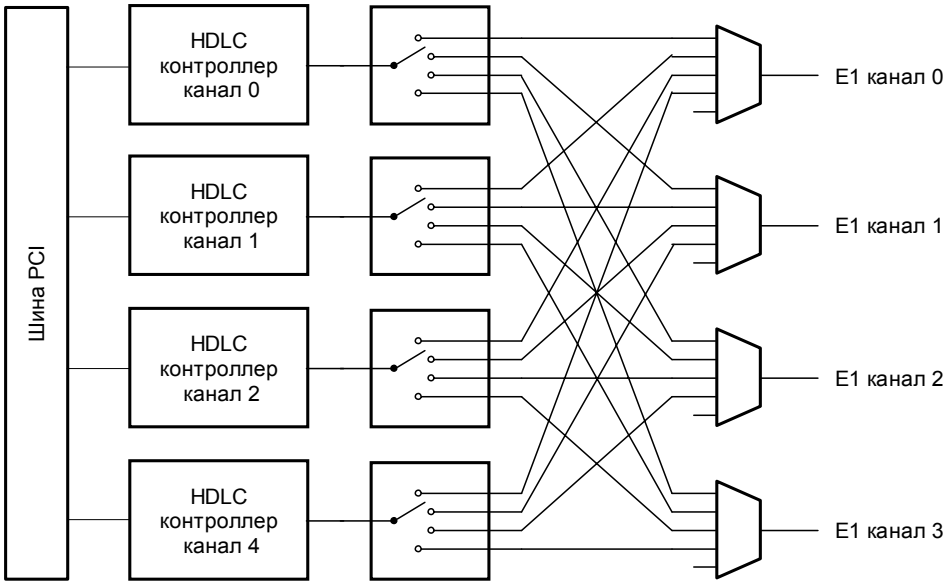
- Конфигурация A: сдвоенный модем E1**.** Каналы E1-0 и E1-1 функционируют независимо и образуют два идентичных потока данных в память компьютера. Скорость передачи данных определяется количеством используемых канальных интервалов. Устанавливается по умолчанию.



- Конфигурация C: мультиплексор с двумя каналами данных и подканалом E1. Канальные интервалы основной линии E1-0 разделяются между двумя каналами 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-PCI-E1 (до 30) в цепочку с расстоянием между узлами до 1.5 км.

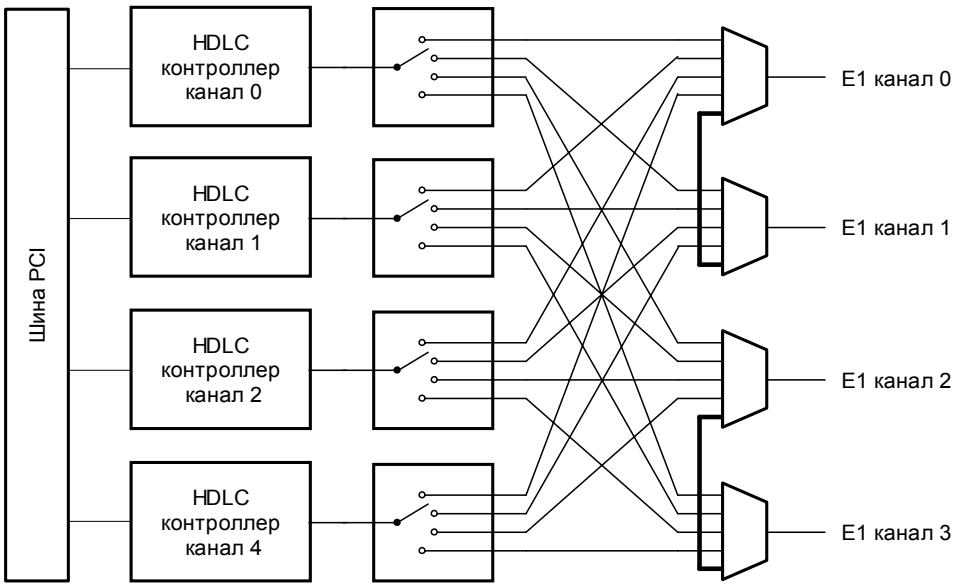
Для плат Tau-PCI-2E1 и Tau-PCI-4E1, также имеют место быть две конфигурации. Они обратно совместимы с работой платы Tau-PCI-E1. Оба адаптера Tau-PCI-2E1 и Tau-PCI-4E1 имеют четыре HDLC контроллера (четыре логи-

ческих канала). Если к адаптеру Tau-PCI-2E1 подключена плата расширения Delta2, то логические каналы 2 и 3 жестко привязаны к портам платы расширения, иначе могут использоваться для разбора потоков E1. В любом конфигурации, любой из логических интерфейсов может быть подключен к любому (и только одному) физическому интерфейсу. К одному физическому интерфейсу может быть подключено до четырех логических интерфейсов.



- Конфигурация A: Каналы E1-0 и E1-1 (E1-2 и E1-3 для Tau-PCI-4E1) функционируют независимо.

Устанавливается по умолчанию.



- Конфигурация C: Все каналы E1, связаны по синхронизации. Между каналами E1-0, E1-1 и E1-2, E1-3 транслируются канальные интервалы, не занятые логическими каналами.

Выбор конфигурации адаптера выполняется функцией *cp_set_mux()*.

Каналы адаптеров Tau-PCI-2E1 и Tau-PCI-4E1 могут работать как в режиме с цикловой структурой (E1) так и без нее (G.703). Переключение производится функцией *cp_set_unfram()*. Канал без цикловой структуры может быть связан только с одним HDLC контроллером, соответствующим его номеру. Если не оговорено обратное, функции, работающие с каналами адаптера Tau-PCI-G703, работают точно также и с каналами в режиме unfram.

Установка скорости канала G.703 производится функцией *cp_set_baud()*, скорость логического интерфейса связанного с каналом E1 определяется набором выбранных канальных интервалов (*cp_set_ts()*). Скорость каналов E1 и G.703 доступна как поле baud структуры канала cp_chan_t.

Функция cp_set_mux()

```
void cp_set_mux (cp_board_t *b, int on)
```

Устанавливает конфигурацию адаптера (только для Tau-PCI-E1, Tau-PCI-2E1 и Tau-PCI-4E1). Текущее значение режима доступно как поле *mux* структуры адаптера *cp_board_t*.

b — Указатель на переменную состояния платы.

on — Если не равно 0 - конфигурация C, иначе конфигурация A.

Функция cp_set_dir()

```
void cp_set_dir (cp_chan_t *c, int dir)
```

Устанавливает номер канала E1 с которым связан данный логический интерфейс (только для Tau-PCI-2E1 и Tau-PCI-4E1). Для каналов работающих в режиме unfram и/или phony значение dir равно номеру E1 канала (c->num) и не может быть изменено. По умолчанию dir = num. Для интерфейсов 2 и 3 адаптера Tau-PCI-2E1 без платы расширения dir = 0, 1. Для адаптера Tau-PCI-E1 dir определяется конфигурацией адаптера. Текущее значение номера канала E1 доступно как поле *dir* структуры канала *cp_chan_t*.

c — Указатель на переменную состояния канала.

dir — Номер канала E1 с которым связан данный логический интерфейс.

Функция cp_set_gsyn()

```
void cp_set_gsyn (cp_chan_t *c, int clk)
```

Устанавливает источник синхронизации передатчика канала E1/G.703/E3. По умолчанию используется синхронизация от внутреннего генератора. Текущее значение режима доступно как поле *gsyn* структуры канала *cp_chan_t*.

c — Указатель на переменную состояния канала.

clk — Источник синхронизации передатчика канала:

GSYN_INT	— от внутреннего генератора
GSYN_RCV	— от приёмника
GSYN_RCV0	— от приёмника канала 0
GSYN_RCV1	— от приёмника канала 1
GSYN_RCV2	— от приёмника канала 2
GSYN_RCV3	— от приёмника канала 3

Функция cp_set_ts()

```
void cp_set_ts (cp_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, кроме 16 (см. *cp_set_use16()*). Текущее значение режима доступно как поле *ts* структуры канала *cp_chan_t*.

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 cp_set_higain()

```
void cp_set_higain (cp_chan_t *c, int on)
```

Включает/выключает режим высокой (нелинейной) чувствительности приёмника контроллера E1. По умолчанию режим высокой чувствительности выключен. При включенном режиме происходит нелинейное усиление входного сигнала исходя из затухания и искажения сигнала при прохождении по кабелю. Текущее значение режима доступно как поле *higain* структуры канала *cp_chan_t*.

b — Указатель на переменную состояния платы.

on — Если не равно 0, включить режим высокой чувствительности (-36 dB для Tau-PCI-E1, -43dB для Tau-PCI-2E1 и Tau-PCI-4E1), иначе выключить (-12 dB).

Функция cp_set_monitor()

```
void cp_set_monitor (cp_chan_t *c, int on)
```

Включает/выключает режим высокой (линейной) чувствительности приёмника контроллера E1. По умолчанию режим высокой чувствительности выключен. При включенном режиме происходит линейное усиление входного сигнала. Может использоваться для прослушивания линии, с подключением к ней через резисторы, используемые для снижения нагрузки на передающую сторону. Текущее значение режима доступно как поле *monitor* структуры канала *cp_chan_t*.

b — Указатель на переменную состояния платы.

on — Если не равно 0, включить режим высокой чувствительности (-30 dB), иначе выключить (-12 dB).

Функция cp_set_use16()

```
void cp_set_use16 (cp_chan_t *c, int u)
```

Включает/выключает режим сверхцикловой синхронизации CAS (E1). Режим CAS влияет на использование 16-го канального интервала. При включенном CAS (по умолчанию) 16-й интервал недоступен для передачи данных (use16=0). Текущее значение режима доступно как поле *use16* структуры канала *cp_chan_t*.

b — Указатель на переменную состояния платы.

u — Если не равно 0: отключить режим CAS, 16-й канальный интервал доступен для передачи данных. Равно 0: включить режим CAS, 16-й канальный интервал недоступен.

Функция cp_set_crc4()

```
void cp_set_crc4 (cp_chan_t *c, int on)
```

Включает/выключает режим контрольной суммы CRC4 (E1). По умолчанию режим CRC4 выключен. Текущее значение режима доступно как поле *crc4* структуры канала *cp_chan_t*.

b — Указатель на переменную состояния платы.

on — Если не равно 0, включить режим CRC4, иначе выключить.

Функция cp_set_phony()

```
void cp_set_phony (cp_chan_t *c, int on)
```

Включает/выключает “телефонный” режим. Текущее значение режима доступно как поле *phony* структуры канала *cp_chan_t*.

Для адаптера Tau-PCI-E1, 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. Телефонный режим можно включить только в конфигурации A.

Для адаптера Tau-PCI-2E1 и Tau-PCI-4E1, в телефонном режиме выбранные канальные интервалы циклов E1 объединяются по 32 цикла и поступают в (из) компьютер. Программное обеспечение получает возможность принимать и передавать телефонные и сигнальные данные. Длина пакетов данных в телефонном режиме равняется 1024 байта. Порядок бит в байте обратен тому, что у платы Tau-PCI-E1 (принят в телефонии). Только один из логических интерфейсов, связанных с данным физическим интерфейсом, может принимать данные в телефонном режиме.

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

Функция cp_set_unfram()

```
void cp_set_unfram (cp_chan_t *c, int on)
```

Включает/выключает режим без цикловой структуры - G.703 (только для моделей Tau-PCI-2E1 и Tau-PCI-4E1). По умолчанию режим без цикловой структуры выключен. Текущее значение режима доступно как поле *unfram* структуры канала *cp_chan_t*.

b — Указатель на переменную состояния платы.
on — Если не равно 0, включить режим без цикловой структуры, иначе выключить.

Функция cp_set_scrambler()

```
void cp_set_scrambler (cp_chan_t *c, int on)
```

Включает/выключает режим скремблирования потока G.703 (только каналов G.703). При включенном режиме происходит перемешивание данных для уст-

ранения длительных последовательностей нулей или единиц, что повышает стабильность работы некоторых видов коммуникационного оборудования. Данный режим не является стандартным и гарантируется совместимость только с оборудованием компании Cronyx. По умолчанию режим скремблирования выключен. Текущее значение режима доступно как поле *scrambler* структуры канала *cp_chan_t*.

b — Указатель на переменную состояния платы.

on — Если не равно 0, включить режим без цикловой структуры, иначе выключить.

Функция cp_get_lq()

```
int cp_get_lq (cp_chan_t *c)
```

Измеряет уровень сигнала в линии (только для Tau-PCI-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

Параметры каналов E3

Функция cp_set_gsyn()

```
void cp_set_gsyn (cp_chan_t *c, int clk)
```

Устанавливает источник синхронизации передатчика канала E1/G.703/E3. По умолчанию используется синхронизация от внутреннего генератора. Текущее значение режима доступно как поле *gsyn* структуры канала *cp_chan_t*.

c — Указатель на переменную состояния канала.

clk — Источник синхронизации передатчика канала:

GSYN_INT	— от внутреннего генератора
GSYN_RCV	— от приёмника
GSYN_RCV0	— от приёмника канала 0
GSYN_RCV1	— от приёмника канала 1
GSYN_RCV2	— от приёмника канала 2
GSYN_RCV3	— от приёмника канала 3

Функция cp_set_rloop()

```
void cp_set_rloop (cp_chan_t *c, int on)
```

Включает/выключает удаленный шлейф (принимаемые данные заворачиваются обратно в линию). По умолчанию удаленный шлейф выключен. Текущее значение можно узнать вызовом функции *cp_get_rloop ()*.

b — Указатель на переменную состояния платы.

on — Если не равно 0, включить удаленный шлейф, иначе выключить.

Функция cp_set_ber()

```
void cp_set_ber (cp_chan_t *c, int on)
```

Включает/выключает режим BER-тестера. В этом режиме в линию выдается специальная тестовая последовательность. В этом режиме ошибок нарушения кодирования считает битовые ошибки нарушения тестовой последовательности. По умолчанию режим BER-тестера выключен. Текущее значение режима доступно как поле *ber* структуры канала *cp_chan_t*.

b — Указатель на переменную состояния платы.

on — Если не равно 0, включить режим BER-тестера, иначе выключить.

Функция cp_set_losais()

```
void cp_set_losais (cp_chan_t *c, int on)
```

Включает/выключает автоматическую выдачу сигнала AIS в случае потери синхронизации по E3 (LOS). По умолчанию автоматическая выдача AIS выключена. Текущее значение режима доступно как поле *losais* структуры канала *cp_chan_t*.

b — Указатель на переменную состояния платы.

on — Если не равно 0, включить автоматическую выдачу AIS, иначе выключить.

Примеры

Приведены тексты примеров использования DDK под ОС Linux: тесты адаптеров Tau-PCI и Tau-PCI-E1. Тексты этих и других примеров находятся в дистрибутиве в директории examples. Тесты представляют собой модули, загружаемые в ядро командой *insmod*. При загрузке вызывается функция *init_module()*, производящая поиск и инициализацию адаптера, и запуск канала. Каждую секунду по таймеру вызывается функция *info()*, печатающая на консоли значение счетчиков переданных и принятых байтов. Остановить тест и выгрузить модуль можно командой *rmmod*.

Тест для адаптера Tau-PCI

Программа тестирует один канал на плате Cronyx Tau-PCI. В канал непрерывно посылаются HDLC-пакеты. Принимаемые пакеты сравниваются с образцом. Для тестирования используется канал #0.

```
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/pci.h>
#include <asm/io.h>
#include "machdep.h"
#include "cpddk.h"

MODULE_AUTHOR("(C) 1999-2003 Cronyx Engineering.");
MODULE_DESCRIPTION("Cronyx Tau-PCI example.");
#ifdef MODULE_LICENSE
MODULE_LICENSE("GPL");
#endif

#define BAUD    2048000

cp_board_t adapter;
cp_qbuf_t *queue;
cp_buf_t *buffer;
char data [1024];
unsigned char irq;
struct timer_list timer;

static spinlock_t lock = SPIN_LOCK_UNLOCKED;
#define LCK(f)   spin_lock_irqsave (&lock, f)
#define UNLCK(f) spin_unlock_irqrestore (&lock, f)

/* Прием пакета */
void receive (cp_chan_t *c, unsigned char *buf, int len)
{
   /* Проверка данных */
   if (len != sizeof(data) || memcmp (data, buf, len) != 0)
        printk ("--Data Error--\n");
}
```

```
/* Завершение передачи пакета */
void transmit (cp_chan_t *c, void *attachment, int len)
{
  memset (data, 'Z', sizeof(data));
  while (cp_send_packet (c, data, sizeof(data), NULL) >= 0)
       continue;
}

/* Обработка ошибок */
void error (cp_chan_t *c, int reason)
{
  switch (reason) {
  case CP_FRAME:  printk ("--Framing Error--\n"); break;
  case CP_CRC:    printk ("--CRC Error--\n");     break;
  case CP_OVERRUN:  printk ("--Overrun--\n");       break;
  case CP_OVERFLOW: printk ("--Overflow--\n");      break;
  case CP_UNDERRUN: printk ("--Underrun--\n");      break;
  }
}

/* Каждую секунду: печать статистики */
void info (unsigned long arg)
{
  printk ("bytes sent %ld, received %ld\n",
       (long) adapter.chan[0].obytes, (long) adapter.chan[0].ibytes);
  timer.function = &info;
  timer.expires = jiffies + HZ;
  add_timer (&timer);
}

/* Прерывание: вызов обработчика */
void intr (int irq, void *dev, struct pt_regs *regs)
{
  cp_interrupt ((cp_board_t*) dev);
}

int init_module (void)
{
  struct pci_dev *dev;
  unsigned long flags;
  cp_chan_t *c;

  /* Выделение памяти под буфера */
  queue = kmalloc (sizeof (*queue), GFP_KERNEL);
  buffer = kmalloc (sizeof (*buffer), GFP_KERNEL);
  if (! queue || ! buffer) {
       printk ("out of memory\n");
       return -ENOMEM;
  }

  /* Поиск адаптера */
  dev = pci_find_device (cp_vendor_id, cp_device_id, 0);
  if (! dev) {
       printk ("adapter not found\n");
       return -ENXIO;
  }
```

```
    /* Инициализация адаптера */
    LCK (flags);
    cp_init (&adapter, 0, ioremap (pci_resource_start (dev, 0), 0x800));
    cp_reset (&adapter, queue, virt_to_phys (queue));
    printk ("Found %s at irq %d mem %lx\n",
         adapter.name, dev->irq, pci_resource_start (dev, 0));

    /* Выбор канала для тестирования */
    c = &adapter.chan[0];
    if (c->type != T_SERIAL) {
         printk ("No HDLC channels detected\n");
         UNLCK (flags);
         return -ENXIO;
    }

    /* Установка обработчика прерывания */
    if (request_irq (dev->irq, intr, SA_SHIRQ, "taup", &adapter) != 0) {
         printk ("cannot get irq %d\n", dev->irq);
         UNLCK (flags);
         return -EBUSY;
    }
    irq = dev->irq;

    /* Регистрация обработчиков событий */
    cp_register_receive (c, &receive);
    cp_register_transmit (c, &transmit);
    cp_register_error (c, &error);

    /* Запуск канала */
    printk ("Starting HDLC channel %d at %d baud\n", c->num, BAUD);
    printk ("Enabling internal loopback\n");
    cp_start_chan (c, 1, 1, buffer, virt_to_phys (buffer));
    cp_led (&adapter, 1);
    cp_set_baud (c, BAUD);
    cp_set_lloop (c, 1);

    /* Устанавливаем сигналы RTS и DTR */
    cp_set_rts (c, 1);
    cp_set_dtr (c, 1);

    /* Передача первого пакета */
    transmit (c, 0, 0);
    UNLCK (flags);

    /* Печать информации каждую секунду */
    init_timer (&timer);
    info (0);
    return 0;
}

void cleanup_module (void)
{
    unsigned long flags;

    /* Сброс платы */
    printk ("Closing\n");
```

```
  LCK (flags);
  cp_stop_chan (&adapter.chan[0]);
  cp_reset (&adapter, 0, 0);
  UNLCK (flags);

  /* Освобождаем ресурсы */
  del_timer (&timer);
  free_irq (irq, &adapter);
  iounmap (adapter.base);
  kfree (queue);
  kfree (buffer);
}
```

Тест для адаптера Tau-PCI-E1

```
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/pci.h>
#include <asm/io.h>
#include "machdep.h"
#include "cpddk.h"

MODULE_AUTHOR("(C) 1999-2003 Cronyx Engineering.");
MODULE_DESCRIPTION("Cronyx Tau-PCI example.");
#ifdef MODULE_LICENSE
MODULE_LICENSE("GPL");
#endif

cp_board_t adapter;
cp_qbuf_t *queue;
cp_buf_t *buffer;
char data [1024];
unsigned char irq;
struct timer_list timer;

static spinlock_t lock = SPIN_LOCK_UNLOCKED;
#define LCK(f)   spin_lock_irqsave (&lock, f)
#define UNLCK(f) spin_unlock_irqrestore (&lock, f)

/* Прием пакета */
void receive (cp_chan_t *c, unsigned char *buf, int len)
{
  /* Проверка данных */
  if (len != sizeof(data) || memcmp (data, buf, len) != 0)
       printk ("--Data Error--\n");
}

/* Завершение передачи пакета */
void transmit (cp_chan_t *c, void *attachment, int len)
{
  memset (data, 'Z', sizeof(data));
  while (cp_send_packet (c, data, sizeof(data), NULL) >= 0)
       continue;
}
```

```
/* Обработка ошибок */
void error (cp_chan_t *c, int reason)
{
   switch (reason) {
   case CP_FRAME:  printk ("--Framing Error--\n"); break;
   case CP_CRC:    printk ("--CRC Error--\n");     break;
   case CP_OVERRUN:  printk ("--Overrun--\n");       break;
   case CP_OVERFLOW: printk ("--Overflow--\n");      break;
   case CP_UNDERRUN: printk ("--Underrun--\n");      break;
   }
}

/* Каждую секунду: печать статистики */
void info (unsigned long arg)
{
   printk ("bytes sent %ld, received %ld\n",
        (long) adapter.chan[0].obytes, (long) adapter.chan[0].ibytes);
   timer.function = &info;
   timer.expires = jiffies + HZ;
   add_timer (&timer);
}

/* Прерывание: вызов обработчика */
void intr (int irq, void *dev, struct pt_regs *regs)
{
   cp_interrupt ((cp_board_t*) dev);
}

int init_module (void)
{
   struct pci_dev *dev;
   unsigned long flags;
   cp_chan_t *c;

   /* Выделение памяти под буфера */
   queue = kmalloc (sizeof (*queue), GFP_KERNEL);
   buffer = kmalloc (sizeof (*buffer), GFP_KERNEL);
   if (! queue || ! buffer) {
        printk ("out of memory\n");
        return -ENOMEM;
   }

   /* Поиск адаптера */
   dev = pci_find_device (cp_vendor_id, cp_device_id, 0);
   if (! dev) {
        printk ("adapter not found\n");
        return -ENXIO;
   }

   /* Инициализация адаптера */
   LCK (flags);
   cp_init (&adapter, 0, ioremap (pci_resource_start (dev, 0), 0x800));
   cp_reset (&adapter, queue, virt_to_phys (queue));
   printk ("Found %s at irq %d mem %lx\n",
        adapter.name, dev->irq, pci_resource_start (dev, 0));

   /* Выбор канала для тестирования */
```

```
    c = &adapter.chan[0];
    if (c->type != T_E1) {
        printk ("No E1 channels detected\n");
        UNLCK (flags);
        return -ENXIO;
    }

    /* Установка обработчика прерывания */
    if (request_irq (dev->irq, intr, SA_SHIRQ, "taup", &adapter) != 0) {
        printk ("cannot get irq %d\n", dev->irq);
        UNLCK (flags);
        return -EBUSY;
    }
    irq = dev->irq;

    /* Регистрация обработчиков событий */
    cp_register_receive (c, &receive);
    cp_register_transmit (c, &transmit);
    cp_register_error (c, &error);

    /* Запуск канала */
    printk ("Starting E1 channel %d, timeslots 1-31\n", c->num);
    cp_start_e1 (c);
    cp_start_chan (c, 1, 1, buffer, virt_to_phys (buffer));
    cp_led (&adapter, 1);

    /* Частота передачи E1 - внутренний генератор.  */
    cp_set_gsyn (c, GSYN_INT);

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

    /* Передача первого пакета */
    transmit (c, 0, 0);
    UNLCK (flags);

    /* Печать информации каждую секунду */
    init_timer (&timer);
    info (0);
    return 0;
}

void cleanup_module (void)
{
    unsigned long flags;

    /* Сброс платы */
    printk ("Closing\n");
    LCK (flags);
    cp_stop_chan (&adapter.chan[0]);
    cp_stop_e1 (&adapter.chan[0]);
    cp_reset (&adapter, 0, 0);
    UNLCK (flags);

    /* Освобождаем ресурсы */
    del_timer (&timer);
    free_irq (irq, &adapter);
```

```
    iounmap (adapter.base);
    kfree (queue);
    kfree (buffer);
}
```