

Контроллер КАМАК СМ5307-PPC Программное обеспечение.

В.Р.Мамкин (т. 394284)
V.R.Mamkin@inp.nsk.su

Новосибирск, 2004

ОГЛАВЛЕНИЕ

1. Ядро	3
1.1. Исходный текст и компиляция	3
1.2. Командная строка.	3
2. Корневая файловая система	3
2.1. NFS	3
2.2. RAM диск.....	4
3. Редактирование и сохранение конфигурации ОС Linux	4
4. Средства разработки приложений.....	6
Приложения.....	7
Приложение 1. Драйвер шины КАМАК для контроллера CM5307.....	8
Приложение 2. ППИ-6 совместимый драйвер шины КАМАК.....	10
1. Введение	10
2. Использование драйвера	10
3. Системные вызовы.....	10
4. Макросы, определенные в libcamac.h	12
4.1. Кодирование	12
4.3. Специальные функции	13
4.4. Работа с LAM запросами.....	13
4.5. Линия INHIBIT	14
5. Сообщения об ошибках	14
Приложение 3. Библиотека libcamdirect	14
1. Особенности библиотеки	14
2. Инициализация.....	14
3. NAF операции	15

1. Ядро

1.1. Исходный текст и компиляция

Для контроллера используются стандартные ядра Linux, например, с сайта <http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.22.tar.bz2>, с патчем <http://www.kernel.org/pub/linux/kernel/ports/ppc/2.4/linuxppc-2.4.22.patch.bz2>.

Для кросс компиляции ядра необходимо внести изменения в Makefile.

Во-первых, добавить строку

```
export CROSS_COMPILE = $(PPC860_BASE)/linux/tools/bin/ppc-linux-
```

Во-вторых, изменить переменную

```
ARCH := ppc
```

Возможна также компиляция ядра «родным» компилятором под управлением контроллера.

Перед компиляцией ядра необходимо заменить конфигурационный файл (.config в корне дерева) и выбрать необходимые опции.

1.2. Командная строка.

В командной строке ядру сообщается устройство, на котором содержится корневая файловая система, а также ряд других параметров. Командная строка редактируется в мониторе (см. техническое описание).

Будем рассматривать два основных случая.

1. Корневой файловой системой является nfs

В этом случае командная строка должна иметь вид (как вариант):

```
root=/dev/nfs rw nfsroot=192.168.1.1:/usr/ppc860/linux/initrd  
ip=192.168.1.2:192.168.1.1::255.255.255.0::eth0:off
```

где

192.168.1.1 – ip адрес nfs сервера

192.168.1.2- ip адрес контроллера

255.255.255.0 – сетевая маска

/usr/ppc860/linux/initrd – экспортируемая директория

2. Корневой файловой системой является ram диск

Командная строка должна иметь вид:

```
root=/dev/ram0 rw ramdisk_size=5000
```

где параметр ramdisk_size задает размер образа файловой системы, в килобайтах (см. следующую главу).

2. Корневая файловая система

2.1. NFS

При использовании nfs в качестве корневой файловой системы можно использовать два поддерева, которые являются частью sdk.

\$(PPC860_BASE)/linux/initrd_cm5307 – компактная реализация, основанная на проекте busybox. Данное поддерево также используется для создания образа ram диска.

\$(PPC860_BASE)/linux/initrd_big – более полная реализация, с большим количеством утилит и библиотек.

Драйвер КАМАК, а также утилиты, специфичные для контроллера CM5307PPC, расположены в поддереве initrd_cm5307. Ниже приведено описание некоторых из них.

Утилита pload

Использование: **/usr/sbin/pload filename**

Загружает из файла с именем filename микропрограмму КАМАК, это должно быть сделано до загрузки модуля КАМАК.

Утилита flash

См. главу «Редактирование и сохранение конфигурации ОС Linux»

Скрипт writeflash

См. главу «Редактирование и сохранение конфигурации ОС Linux»

Утилита /usr/sbin/camtest

Тестирование шины КАМАК, используется при наладке контроллера

Файл /etc/cm5307/cam_bus.rbf

Микропрограмма КАМАК

Модуль /lib/modules/2.4.22/camac.o

Реализует драйвер КАМАК, совместимый с контроллером CM5307. Описание приведено в приложении 1.

Модуль /lib/modules/2.4.22/camac_p.o

Реализует драйвер КАМАК, совместимый с контроллером PPI. Описание приведено в приложении 2.

Одновременное использование двух вариантов драйвера не предусмотрено.

Для редактирования файлов конфигурации Linux средствами контроллера можно использовать экранный редактор **joe**.

2.2. RAM диск

Если как корневая файловая система используется ram диск, он должен быть проинициализирован до запуска init процесса. Для этого вместе с ядром в память контроллера загружается сжатый образ ram диска. Содержимое ram диска находится в sdk - \$(PPC860_BASE)/linux/initrd_cm5307. Для подготовки образа в sdk есть следующий инструментарий.

mkramdisk

Создает образ ram диска из указанной директории и сжимает его. Результат в ramdisk.gz.

mkbootimage

Создает загружаемый образ (сжатое ядро + образ диска) в виде пакета linux, совместимого с монитором-загрузчиком. Результат в image.bin.

mount_ramdisk

Распаковывает файл диска (ramdisk.gz) и монтирует его к директории ramdisk_mnt.

umount_ramdisk

Размонтирует ramdisk_mnt и упаковывает ramdisk.gz.

3. Редактирование и сохранение конфигурации ОС Linux

Конфигурация ОС Linux может быть сохранена в энергонезависимую память (flash) и затем автоматически восстановлена при загрузке системы. Большинство файлов конфигурации находятся в директории etc и являются текстовыми, поэтому для их редактирования можно использовать экранный текстовый редактор **joe**.

Последовательность действий при сохранении конфигурации следующая:

1. Файлы etc директории запаковываются в tar архив
2. Архив сжимается утилитой gzip
3. Полученный сжатый файл записывается в flash память утилитой /usr/sbin/flash

Для упрощения процесса сохранения конфигурации существует скрипт **writeflash**, который автоматически выполняет действия 1-3. Загрузка конфигурации из flash памяти осуществляется в обратной последовательности.

Объем памяти, доступный для конфигурационных данных, ограничивается общим размером flash памяти (4 Мбайта) и количеством flash памяти, занятой для пакета ОС Linux (сжатый образ ядра и ram диска).

Утилита flash имеет следующие параметры:

flash [r|w] filename

где

r – опция чтения, конфигурационные данные из flash памяти читаются в файл с именем **filename**;

w – опция записи, конфигурационные данные из файла с именем **filename** записываются во flash память.

Для восстановления конфигурации при запуске системы, утилита flash имеет специальный режим работы, переход в который осуществляется, если `pid = 1`. Иными словами, утилита flash может запускаться вместо процесса `init`. Чтобы сохраненная конфигурация восстанавливалась каждый раз при запуске ОС, необходимо в командной строке указать в качестве `init` процесса утилиту flash:

init=/usr/sbin/flash

Если в командной строке указанная подстрока отсутствует, загружается конфигурация по умолчанию (сохраненная в образе `ram` диска).

Внимание! При обновлении версии программного обеспечения (ядра и образа `ram` диска) текущая конфигурация теряется. Для ее сохранения необходимо прочитать конфигурацию утилитой flash во временный файл, который далее можно забрать из контроллера через ftp.

4. Средства разработки приложений

По умолчанию корневой директорией для программного обеспечения является /usr/ppc860. Путь к корневой директории используется в make файлах и должен экспортироваться как переменная окружения:

```
export PPC860_BASE=/usr/ppc860
```

Приложения могут разрабатываться как с кросс компилятором, так с «родным».

В корневой директории находятся файлы:

linux/apps – примеры приложений, модули КАМАК

linux/apps/camac_module – модуль КАМАК, совместимый с контроллером CM5307

linux/apps/camac_module_ppi – модуль КАМАК, совместимый с ППИ-6

linux/apps/makeconfig - определения путей к библиотекам и h-файлам, флагов компилятора и т.д. Этот файл должен подключаться директивой include в пользовательских make-файлах.

linux/initrd_big – образ корневой файловой системы для подключения по nfs, полный вариант. В этой же директории находятся h-файлы и библиотеки.

linux/initrd_cm5307 – образ корневой файловой системы для подключения по nfs или создания образа ram диска

linux/kernel – исходные тексты ядра, сконфигурированного для контроллера

linux/tools – кросс компилятор, ассемблер и другие средства

Приложения

В приложении 1 приведено описание драйвера шины КАМАК, совместимого с предыдущей реализацией в контроллере CM5307 на процессоре Motorola CM5307. Модуль драйвера расположен в файловой системе контроллера, в директории /lib/modules/(KERNEL_VERSION)/camac.o.

В приложении 2 приведено описание драйвера шины КАМАК и библиотеки, имеющих программный интерфейс аналогичный драйверу и библиотеке адаптера ППИ-6, в реализации Никифорова. Модуль драйвера расположен в файловой системе контроллера, в директории /lib/modules/(KERNEL_VERSION)/camac_p.o. Кроме того, драйвер имеет расширение – возможность работать с LAM запросами через вызов select.

Оба драйвера не могут использоваться одновременно.

Недостатком драйверов является достаточно большое время исполнения одиночных NAF инструкций – около 23 мкс. Блочные операции с использованием драйвера занимают время около 1.5 мкс на один NAF. Для быстрого исполнения одиночных NAF можно использовать библиотеку libcamdirect, описание которой приведено в приложении 3. Библиотека может использоваться совместно с одним из драйверов. При этом через драйвер осуществляется работа с LAM запросами и операции с линиями C,Z,I, а через библиотеку – NAF операции. Временные параметры библиотеки следующие – 2.5 мкс на одиночный NAF и 1.5 мкс при блочных NAF операциях.

Приложение 1. Драйвер шины КАМАК для контроллера CM5307

Драйвер шины КАМАК разработан в виде модуля и обеспечивает программный интерфейс для доступа к шине КАМАК прикладных процессов.

Прикладные процессы могут обращаться к шине КАМАК через файл устройства /dev/camac. Число процессов, одновременно совершающих операции ввода-вывода не ограничено. Появление активного LAM запроса приводит к посылке сигнала SIGUSR1, если процесс предварительно разрешил обработку данного LAM.

Перед началом работы с шиной процесс открывает специальный файл:
fd = open("/dev/camac", O_RDWR)

Работа с шиной должна завершаться закрытием специального файла:
close(fd)

Операции чтения/записи

```
int read(int fd, char *buf, int len)
int write(int fd, char *buf, int len)
```

Где

Первый параметр – файловый дескриптор.

Второй параметр – указатель на заголовок cam_header_t (см. предыдущий пункт)

Третий параметр – число операций чтения/записи, <= 256

Возвращаемое значение – число операций чтения/записи

Выполнение циклов C и Z

ioctl вызовы:

```
ioctl(fd, CAM_IOCTL_EXC, 0)
ioctl(fd, CAM_IOCTL_EXZ, 0)
```

приводят к появлению на шине C и Z циклов соответственно.

Первый параметр – файловый дескриптор.

Второй параметр – код операции (константа, определенная в файле camac.h)

Третий параметр игнорируется.

Возвращаемое значение – 0.

Управление сигналом Inhibit

Для чтения сигнала I на шине КАМАК выполняется ioctl вызов:

```
unsigned x;
...
ioctl(fd, CAM_IOCTL_GETI, &x);
```

Первый параметр – файловый дескриптор.

Второй параметр – код операции

Третий параметр – указатель на unsigned

Возвращаемое значение – ioctl вызов возвращает 0,

состояние сигнала Inhibit находится в переменной x.

Для установки сигнала I выполняется ioctl вызов:

unsigned x;

...

ioctl(fd, CAM_IOCTL_SETI, &x);

Первый параметр – файловый дескриптор.

Второй параметр – код операции

Третий параметр – указатель на unsigned

Возвращаемое значение – ioctl вызов возвращает 0

Сигнал Inhibit будет установлен в соответствии с содержимым переменной x.

Управление LAM запросами

unsigned x;

...

ioctl(fd, CAM_IOCTL_LAMENABLE, &x);

Данный вызов “приписывает” указанный LAM процессу и размаскирует его. В случае активизации данного LAM запроса, процесс получит сигнал SIGUSR1.

Первый параметр – файловый дескриптор.

Второй параметр – код операции

Третий параметр – указатель на unsigned

Возвращаемое значение – ioctl вызов возвращает 0 в случае успешного выполнения и -1 если LAM занят другим процессом или модулем.

Переменная x содержит номер LAM запроса 0..23

unsigned x;

...

ioctl(fd, CAM_IOCTL_LAMDISABLE, &x);

Данный вызов маскирует LAM запрос и освобождает его для захвата другими процессами.

Первый параметр – файловый дескриптор.

Второй параметр – код операции

Третий параметр – указатель на unsigned

Возвращаемое значение – ioctl вызов возвращает 0 в случае успешного выполнения и -1 если LAM занят другим процессом или модулем.

Переменная x содержит номер LAM запроса 0..23

unsigned x;

...

ioctl(fd, CAM_IOCTL_LAMPENDING, &x);

Вызов возвращает маску LAM запросов, активных в настоящий момент. Возвращаемое значение не зависит от установленной маски разрешения/запрета и отображает текущее состояние на шине.

Первый параметр – файловый дескриптор.

Второй параметр – код операции

Третий параметр – указатель на unsigned

Возвращаемое значение – ioctl вызов возвращает 0, маска находится в переменной x.

Приложение 2. ППИ-6 совместимый драйвер шины КАМАК

1. Введение

Обращение к устройству осуществляется через специальный файл `/dev/camac`.
Открыв эти файлы, можно работать с КАМАКом, используя функции работы с файлами:
`open`, `close`, `ioctl`, `lseek`, `read`, `write` и макросы из файла ``libcamac.h'`

Драйвер имеет следующие черты:

- Устанавливается как загружаемый модуль;
- Может выполнять 16-и или 24-х разрядные одиночные КАМАК циклы или циклы без данных;
- Может выполнять блочные КАМАК 16-и или 24-х разрядные циклы;
- Может проверять наличие LAM запроса и ожидать LAM запрос в течении указанного времени;
- Управляет линией *Inhibit*;
- Может проводить C и Z КАМАК циклы.

Загрузка модуля, в простейшем случае, может осуществляться командой:

`insmod camac_p.o`

без параметров.

2. Использование драйвера

Драйвер доступен через специальный файлы `/dev/camac`. Если в файловой системе контроллера данный драйвер отсутствует, его необходимо создать командой:

`mknod /dev/camac c 21 0`

В программе на "C" и "C++" для использования драйвера необходимо подключить файл ``libcamac.h'`:

`#include<libcamac.h>`

В этом файле находятся константы и макросы для кодирования параметров при вызове системных функций и описаны функции, облегчающие работу с КАМАКом.

3. Системные вызовы

Драйвер обрабатывает следующие системные вызовы:

`open()`

`int open(const char *pathname, int flafs);`

Открыть КАМАК крейт.

`pathname` это строка ``/dev/camac '`

`flags` должен быть `O_RDWR`.

Функция возвращает дескриптор файла (`fd`), с которым будут работать остальные функции, или `-1` в случае ошибки.

`close()`

`int close(int fd);`

Функция 'закрывает' КАМАК крейт.

Возвращает 0 при успешном выполнении.

lseek()

`off_t lseek(int fd, off_t offset, int whence)`

Функция передает в драйвер адрес *n*, субадрес *a*, функцию *f* и, возможно, признак 24-х битовой работы через *offset* параметр. Следующие за этим функции *read* и *write* будут проводить САМАС обмены, используя переданные параметры.

offset можно построить, используя макро *NAF(N,A,F)* или *NAF24(N,A,F)* из ``ppibcamac.h'`.

whence параметр может быть только *SEEK_SET*. Функция возвращает -1 при ошибке. Т.о. типичное использование этой функции:

```
lseek(fd, NAF(n, a, f), SEEK_SET);
```

lseek не выполняет никаких операций обмена с крейтом, она только устанавливает в драйвере 'указатель позиции в файле', согласно *N*, *A* и *F* для последующего использования. При отсутствии ошибки функция возвращает параметр *offset*.

read() и write()

```
int read(int fd, char *buf, size_t count);
```

```
int write(int fd, char *buf, size_t count);
```

Эти функции обеспечивают проведение КАМАК цикла с предварительно установленными функцией *lseek* адресом, субадресом и функцией.

Тип проводимого КАМАК цикла зависит от значения счетчика *count*.

```
count = 2
```

- проводится 16-и разрядный КАМАК цикл

```
count = 3
```

- проводится 24-х разрядный КАМАК цикл

```
count = 4*n
```

- проводится *n* 16-и или 24-х разрядных КАМАК циклов (аналог LONG NAF)

Данные читаются/пишутся из/в *buf*.

При успешном выполнении возвращается количество переданных байтов. При ошибке возвращается -1.

Ответ *X* и *Q* можно узнать через `ioctl(fd, CAMAC_STATUS, &status)`.

При использовании этих функций обеспечивается максимальная скорость передачи блоков данных. Для выполнения одиночных КАМАК функций лучше использовать функцию `ioctl()`.

ioctl()

```
int ioctl(int fd, int cmd, unsigned long *argp);
```

Позволяет проводить КАМАК циклы с данными и без данных, получать ответ блока и таймаут на последний, проведенный из используемого *fd*, КАМАК-цикл.

Допустимы следующие команды *cmd*, определенные в файле `libcamac.h`:

NAF(n, a, f) - провести 16-и разрядный КАМАК цикл. для команд чтения (F0 - F7), *argp* - куда писать, для команд записи (F16 - F23), *argp* - откуда читать. *argp* может быть NULL. В этом случае проводится КАМАК цикл, данные не передаются. Возвращает ответ *x*, *q* в младших двух битах. При ошибке или таймауте возвращает -1. Код ошибки можно посмотреть в `errno`.

```
CAMAC_24 | NAF(n, a, f)
```

- то же что и *NAF*, но проводится 24-х разрядный КАМАК цикл.

NAF24(n, a, f) - см. выше.

CAMAC_STATUS - кладет в **argp* ответы *timeout*, *X*, *Q* в младшие три бита. Ответ дается от последнего, проведенного через используемый дескриптор *fd*, КАМАК цикла. Так можно получить эти ответы от *long naf*, проведенного через *read* или *write*.

CAMAC_NON_DATA - провести КАМАК цикл без данных, с N, A, F предварительно установленных через `lseek`. Ответ `timeout`, X, Q можно получить через `ioctl` командой **CAMAC_STATUS**.

CAMAC_LWAIT(I) - ожидание появления *LAM* с номером `l`. `timeout=argp` если `timeout > 0` то ждать *LAM* в течении этого таймаута; если `timeout < 0` - ждать вечно; если `timeout == 0` то провести проверку наличия *LAM* запроса. `timeout` указывается в "тиках". (для x86 тик равен 10 миллисекунд). Возвращает остаток таймаута, если *LAM* появился в указанный срок, иначе `-1` и `errno=ETIME`. Для "вечного" ожидания при появлении *LAM* возвращается `0`. При ошибках возвращается `-1`.

CAMAC_ION - включает линию INHIBIT

CAMAC_IOFF - выключает линию INHIBIT

CAMAC_GETI - возвращает состояние линии INHIBIT

CAMAC_C - выполнить цикл C

CAMAC_Z - выполнить цикл Z

При успешном выполнении `ioctl` возвращает `0` или положительное число, в котором закодирован осмысленный ответ. При ошибке возвращается `-1` и номер ошибки записывается в `errno`.

4. Макросы, определенные в `libcamac.h`

Макросы и функции в файле `libcamac.h` служат двум целям.

Во-первых они кодируют параметры, такие как N, A, F и другие, для передачи через системные вызовы драйверу. (см `ioctl()`, `lseek()`). Во-вторых они используются для сокрытия деталей реализации и облегчения работы с КАМАКом. Описанные макросы обеспечивают достаточно полный набор инструментов для работы с КАМАКом.

4.1. Кодирование

int NAF(n,a,f) - кодирует n, a, f для функций `lseek`, `ioctl` и CAM для проведения 16-и битового КАМАК цикла.

int NAF24(n,a,f) - кодирует n, a, f для функций `lseek`, `ioctl` и CAM для проведения 24-х битового КАМАК цикла.

Проведение КАМАК цикла

int CAM(int fd, int naf, unsigned *pdata) - проводит 16-и разрядный КАМАК цикл, с данными `*pdata`. `pdata` может быть `NULL`.

int CAMW(int fd, int naf, unsigned data) - проводит 16-и разрядный КАМАК цикл записи, с данными `data`.

int CAM24(int fd, int naf, unsigned *pdata) - проводит 24-х разрядный КАМАК цикл, с данными `*pdata`. `pdata` может быть `NULL`.

int CAM24W(int fd, int naf, unsigned data) - проводит 24-х разрядный КАМАК цикл записи, с данными `data`.

Параметры функции:

`fd` - дескриптор открытого файла устройства, соответствующего КАМАК крейту;

`naf` - NAF инструкция для проведения обмена.

Функции возвращают ответ X, Q в младших двух битах. При ошибках возвращают `-1`.

4.2. Блочные передачи

int CAM_READ(int fd, int naf, unsigned * buf, int num)

int CAM_WRITE(int fd, int naf, unsigned * buf, int num)

CAM_READ служит для чтения блока данных из КАМАК блока, **CAM_WRITE** для записи блока данных в КАМАК блок.

Параметры функции:

fd - дескриптор открытого файла устройства, соответствующего КАМАК крейту;
naf - *NAF* инструкция для проведения обмена. Может означать как 16-и так и 24-х битовый обмен (см. **Кодирование**);
buf - адрес массива данных. Массив должен состоять из беззнаковых целых и иметь достаточный размер;
num - требуемое количество обменов.
При успешном проведении обмена *CAM_READ* и *CAM_WRITE* возвращают ответ *X* и *Q* в двух младших битах. При ошибке возвращается -1.

4.3. Специальные функции

int CAM_C(int fd) - проводит *Clear* цикл.

Возвращает 0 при успешном выполнении, -1 при ошибке.

int CAM_Z(int fd) - проводит *Zero* цикл.

Возвращает 0 при успешном выполнении, -1 при ошибке.

4.4. Работа с LAM запросами

Ожидание LAM может производиться двумя способами – как в старом PPI драйвере через *ioctl* или посредством вызова *select*.

4.4.1. Ожидание LAM через ioctl

Макрос *CAM_LWAIT* может использоваться для вызова *ioctl* при работе с LAM запросами.

int CAM_LWAIT(int fd, int L, int timeout) - ожидает появление LAM от блока в позиции *L*.
timeout > 0 - ожидает появление LAM в групповом запросе *request* в течении таймаута. При возникновении LAM возвращает остаток от таймаута. (0 так же возможен. например LAM прилетел в последний момент);

timeout = 0 - проверяет наличие указанного LAM запроса. При его наличие возвращает 0, иначе -1;

timeout < 0 - навеки зависает в ожидании LAM

При отсутствии LAM запроса в течении указанного таймаута возвращается ошибка *ETIME*.

Ожидание таймаута может быть также прервано немаскированным сигналом. В этом случае возвращается соответствующая ошибка.

4.4.2. Ожидание LAM через select

Для использования *select* необходимо сначала установить маску требуемых запросов вызовом

ioctl (int fd, CAMAC_SELECT_LMASK, int mask)

где младшие 23 бита маски соответствуют позициям модулей в крейте (бит 0 – позиция 1).

Маска 0x08000000 используется для ожидания ошибочной NAF операции (без ответа *X* или *Q*).

Каждому файловому дескриптору *fd* соответствует своя отдельная маска. Таким образом, несколько процессов могут одновременно ожидать LAM, каждый от своей группы модулей. Каждый процесс при этом должен иметь свой файловый дескриптор. Содержимое поля *mask* вызова *CAMAC_SELECT_MASK* переписывается в аппаратный регистр прерываний только после вызова *select*. Сброс битов в регистре прерывания и разблокирование *select* происходит автоматически после получения хотя бы одного из LAM, указанных в маске. После отработки LAM запроса повторного вызова *ioctl* не требуется – поле *mask* сохраняется во внутренних структурах драйвера. Вызов *select* происходит обычным образом, например:

```
fd_set set;  
int r, mask = 0x1;  
ioctl(crate, CAMAC_SELECT_LMASK, mask);  
FD_ZERO(&set);
```

```

FD_SET(crate, &set);
r = select(crate + 1, NULL, NULL, &set, NULL);
if ((r > 0) && FD_ISSET(crate, &set)) {
    printf("lam!\n");
}

```

4.5. Линия INHIBIT

int CAM_ION(int fd) - включает линию INHIBIT;

int CAM_IOFF(int fd) - выключает линию INHIBIT;

int CAM_I(int fd) - проверка состояния линии INHIBIT. Возвращает 0 если выключена, 1 если включена.

Функции CAM_ION, CAM_IOFF и CAM_I возвращают -1 при ошибке.

5. Сообщения об ошибках

Драйвер сообщает о следующих ошибках:

EIO - ошибка ввода/вывода.

EFAULT - в функцию передан неправильный адрес буфера buf (для функций read и write) или параметр argp (в функции ioctl);

EINVAL - неправильный аргумент при вызове функции;

ENOMEM - не хватает памяти для выполнения операции;

ETIME - не дождалось указанного LAM запроса в указанный отрезок времени.

Приложение 3. Библиотека libcamdirect

1. Особенности библиотеки

В случаях, когда требуется обеспечить быстрое выполнение NAF инструкций, необходимо использовать библиотеку libcamdirect. Библиотека осуществляет прямой доступ к регистрам КАМАК интерфейса путем отображения окна регистров в память процесса. Скорость одиночного чтения/записи NAF при этом составляет около 2.5 мкс, блочных операций – 1.5 мкс на один NAF. Библиотека и заголовочный файл расположены в директории \$(PPC860_BASE)/linux/apps/camac_direct_lib/

Большинство вызовов библиотеки реализованы в виде inline функций. Для оптимизации кода необходимо использовать при компиляции приложения ключ -O3.

Особенностью библиотеки является применение вызова sched_yield(). По этой причине не рекомендуется одновременное использование библиотеки процессами с разными приоритетами (см. man sched_yield), в противном случае возможна блокировка процессов.

Контроль доступа к шине КАМАК осуществляется через флаги, расположенные в разделяемой памяти. Внештатное прекращение процесса, использующего библиотеку, может привести к блокировке флага, контролирующего доступ к шине КАМАК. При этом дальнейшие обращения к библиотеке со стороны других процессов будут блокироваться. В этом случае необходимо удалить страницу разделяемой памяти, которую использует библиотека, с помощью утилит ipcs и ipcrm.

2. Инициализация

Функция

int camdirect_init(void)

используется для инициализации библиотеки, в ходе которой выполняются действия:

- запрашивается доступ к физической памяти регистров КАМАК шины;
- память регистров отображается на память процесса;
- создается страница разделяемой памяти, если страница уже есть в системе, она подключается к текущему процессу;
- в разделяемой памяти инициализируется флаг доступа к шине КАМАК.

Возвращаемые значения:

0 – успешное завершение

-1 – нет доступа к файлу /dev/mem

-2 – нельзя отобразить регистры процессора на память процесса

-3 – нельзя отобразить регистры КАМАК на память процесса

-4 – нельзя создать уникальный ключ для разделяемой памяти (ftok)

-5 – нельзя создать страницу разделяемой памяти

-6 – нельзя подключить страницу разделяемой памяти

-7 – библиотека уже проинициализирована

Работа с библиотекой должна завершаться функцией:

void camdirect_close(void)

3. NAF операции

Исполнение одиночной NAF операции:

camac_status_t camac_read_cycle(int n, int af, unsigned *buf)

camac_status_t camac_write_cycle(int n, int af, unsigned *buf)

где,

n – номер блока (1..23)

af – адрес и функция (см. макрос CAMAC_MAKE_AF)

buf – указатель на данные. 24 бита КАМАК данных будут расположены в младших битах 32-х разрядного слова, на которое указывает buf.

Блочные операции:

camac_status_t

camac_write_block (int n, int af, unsigned *buf, int n_times, int *n_times_done)

camac_status_t

camac_read_block (int n, int af, unsigned *buf, int n_times, int *n_times_done)

где,

n – номер блока (1..23)

af – адрес и функция (см. макрос CAMAC_MAKE_AF)

buf – указатель на массив данных. 24 бита КАМАК данных будут расположены в младших битах 32-х разрядного массива buf[].

n_times – размер массива buf[]

n_times_done – количество успешно выполненных NAF операций. n_times_done всегда меньше либо равно n_times. Блочная операция прекращается, если в цикле (n_times_done + 1) периферийный модуль не выставил X или Q.

Блочные операции с 16 битными данными:

camac_status_t

camac_write_block16 (int n, int af, unsigned short *buf, int n_times, int *n_times_done)

camac_status_t

camac_read_block16 (int n, int af, unsigned short *buf, int n_times, int *n_times_done)

Отличие от предыдущей пары функций – размерность массива buf[] 16 бит.

Все функции возвращают тип `samac_status_t`, который представляет собой битовое поле с флагами:

`SAMAC_MASK_X` – значение линии X

`SAMAC_MASK_Q` – значение линии Q

`SAMAC_MASK_I` – значение линии I

Для блочных операций возвращается статус последнего NAF.