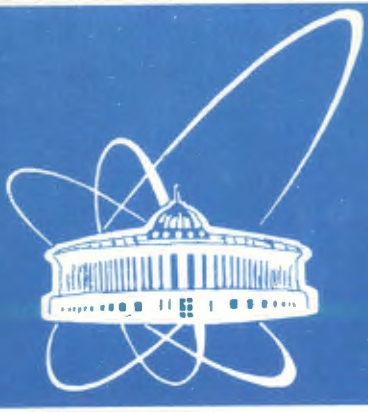

СООБЩЕНИЯ
ОБЪЕДИНЕННОГО
ИНСТИТУТА
ЯДЕРНЫХ
ИССЛЕДОВАНИЙ

Дубна

Р10-98-163

К.И.Гришай, В.Г.Ольшевский

ПРОГРАММНЫЙ ПАКЕТ ДЛЯ РАБОТЫ С КАМАК В ОПЕРАЦИОННОЙ СИСТЕМЕ FreeBSD

1998

1 Введение

Системы сбора данных и управления экспериментом на основе аппаратуры КАМАК и IBM-совместимых персональных компьютеров сегодня широко применяются в тех случаях, когда не требуется большая скорость передачи данных. Распространенность таких систем связана с относительной дешевизной оборудования и широким выбором серийно выпускаемых блоков в стандарте КАМАК.

Используемая на компьютере операционная система во многом определяет логику построения и работы системы сбора данных и может как упростить, так и усложнить написание, сопровождение и использование такой системы.

Для большого круга задач сбора данных и управления экспериментом оптимальной является операционная система UNIX. UNIX — это многозадачная многопользовательская операционная система с развитыми механизмами межзадачного и межмашинного взаимодействия, естественно включающая в себя средства удаленного доступа, поддержку стандартных сетевых протоколов и графических интерфейсов. Накладные расходы на работу UNIX невелики, и средняя конфигурация современного персонального компьютера вполне позволяет использовать эту операционную систему. UNIX реализована практически на всех существующих платформах, для нее имеется богатый выбор программных продуктов, при этом программы, написанные для этой операционной системы, имеют высокую степень переносимости.

Одной из свободно распространяемых UNIX-подобных операционных систем для IBM-совместимых персональных компьютеров является ОС FreeBSD.

В ОИЯИ получили распространение контроллеры КАМАК КК009[1] и КК012[2] для IBM-совместимых персональных компьютеров, разработанные в Лаборатории ядерных проблем[1]. Отличительной особенностью этих контроллеров является реализация работы с КАМАК через операции чтения/записи ячеек памяти, а не портов ввода/вывода. Эта особенность позволяет достаточно просто организовать работу с контроллером из режима задачи UNIX даже без написания полноценного драйвера устройства.

Для организации работы с КАМАК через контроллер КК012 в ОС FreeBSD была написана библиотека для языка программирования Си и несколько сопутствующих утилит. Библиотека разработана для использования на установке МЮСПИН, но мы надеемся, что заложенные в библиотеку возможности позволяют использовать ее и на других установках.

2 Принцип работы библиотеки

Используемый в библиотеке метод работы с КАМАК является логическим развитием подхода, изложенного в работе [3]. Новшеством является возможность работы с КАМАК по прерыванию. Однако это потребовало, при сохранении основных

[1]Далее термины «контроллер КК012» и «интерфейсная плата ПК012» обозначают обе модели контроллера и интерфейсной платы.

принципов работы с КАМАК, изменения реализации метода, в частности, добавления некоторого кода к ядру операционной системы.

Этот код оформлен в виде драйвера устройства, что позволяет естественным образом встроить его в ядро операционной системы, избежав при этом каких-либо изменений в основном коде ядра, и использовать стандартные средства настройки драйвера для задания необходимых параметров на стадии загрузки операционной системы. Следует заметить, что данный код по логике своей работы не является чистым драйвером устройства, что будет понятно из дальнейшего изложения.

Добавленный код выполняет три функции:

- Обеспечивает доступ к адресам физической памяти, занимаемым интерфейсной платой ПК012. Для взаимодействия с процессом используется специальный файл символьного типа /dev/kk. К файлу /dev/kk можно применять операции *open*, *close*, *mmap* и *munmap*. Для получения доступа к физической памяти в принципе можно воспользоваться стандартным специальным файлом /dev/mem, однако использование отдельного файла позволяет открывать доступ только к адресам интерфейсной платы, а не ко всему диапазону адресов, что упрощает библиотеку и повышает ее надежность.

- Осуществляет синхронизацию доступа к КАМАК из нескольких процессов и обработчика прерывания. Взаимодействие с процессом организовано через специальный системный вызов, реализующий следующие операции:

  - захват КАМАК в монопольное пользование;
  - освобождение КАМАК;
  - безусловное освобождение КАМАК с аварийным завершением процесса, захватившего КАМАК;
  - чтение статистической информации о работе с КАМАК;
  - очистка статистической информации.

  В предыдущей версии библиотеки[3] функцию синхронизации доступа выполнял механизм на основе блокировки доступа к файлу, но такой подход неприменим для обработчика прерывания.

- Вызывает пользовательский обработчик прерывания КАМАК. Пользовательский обработчик прерывания пишется в виде загружаемого модуля ядра операционной системы. Тип модуля — системный вызов. Использование загружаемого модуля позволяет вносить изменения в код обработчика прерывания без перезагрузки операционной системы и обеспечивает удобный механизм взаимодействия между обработчиком прерывания и процессом.

Работа с КАМАК из режима задачи строится следующим образом. Для получения доступа к адресам физической памяти, занимаемым интерфейсной платой ПК012, процесс выполняет отображение в свое виртуальное адресное пространство специального файла /dev/kk, используя системные вызовы *open* и *mmap*.

Системный вызов *mmap* возвращает базовый адрес интерфейсной платы в адресном пространстве процесса. Эта операция выполняется один раз при старте процесса. Далее работа с КАМАК происходит аналогично тому, как это делается в MS-DOS, с той разницей, что необходимо осуществлять синхронизацию доступа к КАМАК из нескольких процессов. Поэтому перед началом операции с КАМАК процесс выполняет системный вызов с целью получить КАМАК в монопольное пользование. Если КАМАК в этот момент свободен, управление возвращается процессу без задержки. В случае, когда КАМАК занят, выполнение процесса приостанавливается до момента освобождения ресурса. После окончания операции с КАМАК процесс выполняет системный вызов для освобождения КАМАК[2], при этом процессы, находившиеся в состоянии ожидания освобождения этого ресурса, «пробуждаются» и продолжают свое выполнение. Механизм приостановки и возобновления выполнения процесса, реализованный в драйвере КАМАК, основан на стандартных функциях *sleep* и *wakeup*, используемых в ядре UNIX[4] для выполнения такого рода синхронизации. Таким образом, в каждый момент времени только один процесс может работать с КАМАК.

Конечно, такой механизм обеспечивает правильную работу с КАМАК из нескольких процессов только при условии, что все процессы придерживаются принятых «правил игры». Наряду с этим недостатком такой подход имеет определенные преимущества перед использованием полноценного драйвера КАМАК:

- он обладает большей гибкостью, т.к. непосредственно в ядре не содержится кода, работающего с КАМАК, и все необходимые изменения можно внести на уровне библиотеки;

- он позволяет достичь большей скорости обмена при пересылке данных большими порциями, поскольку процесс, захватив КАМАК в монопольное пользование, может выполнить сразу несколько команд КАМАК без накладных расходов на обращение к драйверу устройства.

Пользовательский обработчик прерывания работает в адресном пространстве ядра, поэтому структуры данных ядра и физическая память доступны ему непосредственно. Базовый адрес памяти, соответствующей адресам платы ПК012 в адресном пространстве ядра, содержится в переменной в драйвере КАМАК. Синхронизация доступа к КАМАК для обработчика прерывания выполняется драйвером автоматически — если в момент появления прерывания КАМАК оказывается занят каким-либо процессом, то обработка прерывания (вызов пользовательского обработчика прерывания) откладывается до момента освобождения ресурса, при этом обработчик прерывания имеет приоритет перед процессами, также ожидающими освобождения КАМАК.

Реализовано два варианта вызова пользовательского обработчика прерывания, основным отличием между которыми является реакция на другие прерывания во

[2] В случае окончания захватившего КАМАК процесса КАМАК освобождается автоматически.

время выполнения кода обработчика прерывания: прерывания могут быть запрещены или разрешены. Первый вариант дает меньшее время отклика на запрос прерывания и позволяет обрабатывать прерывания с большей частотой. Однако если обработчик прерывания выполняется достаточно долго, то использование этого варианта может приводить к потере прерываний от других устройств.

Процесс может обратиться к обработчику прерывания через зарезервированный для этой цели системный вызов, ответные действия обработчика прерывания определяются его конкретной реализацией.

Разработка пользовательского обработчика прерывания требует определенной аккуратности, поскольку он является частью ядра операционной системы, и ошибка в нем будет приводить к сбою в работе всей операционной системы. С другой стороны, есть несколько моментов, облегчающих эту работу:

- код обработчика прерывания можно полностью написать без использования языка ассемблера, оформив его как обычную функцию языка Си;
- все необходимые действия, связанные с реакцией на прерывание, например обслуживание контроллера прерываний, выполняет операционная система;
- ядро операционной системы предоставляет удобные средства для организации взаимодействия между обработчиком прерывания и процессом.

3 Программное обеспечение для работы с КАМАК

В состав программного пакета для работы с контроллером КАМАК КК012 входят: драйвер КАМАК, библиотека низкого уровня libc12, несколько программ для организации работы с КАМАК, документация и примеры.

Драйвер КАМАК предназначен для использования с ОС FreeBSD версии 2.2.x и поддерживает одну ветвь КАМАК. Для установки драйвера необходимо пересобрать ядро операционной системы; подробно процесс установки драйвера описан в прилагаемой документации. Настроечные параметры (базовый адрес интерфейсной платы, номер IRQ и тип пользовательского обработка прерывания) можно изменять на стадии загрузки операционной системы.

Библиотека libc12 содержит функции для взаимодействия с драйвером КАМАК и два независимых набора функций для работы собственно с КАМАК через контроллер КК012. Один набор функций написан в соответствии со стандартом ESONE «Subroutines for CAMAC»[5] и реализует уровень B стандарта. Второй набор функций ориентирован на работу именно с контроллером КК012 и имеет интерфейс, более традиционный для языка программирования Си. Библиотека libc12 не может быть использована при написании пользовательского обработчика прерывания КАМАК.

Помимо библиотеки, написан набор макрокоманд для доступа к регистрам контроллера, который позволяет достичь большей скорости передачи данных и может

быть использован при написании пользовательского обработчика прерывания или другой библиотеки низкого уровня.

В состав программного пакета входят следующие утилиты:

- *c_master* — программа для выполнения управляющих операций в крейте КАМАК (инициализация контроллера, выполнение циклов «Пуск» (Z) и «Сброс» (C), установка и сброс сигнала «Запрет» (I));
- *crate* — эмулятор ручного контроллера КАМАК, написанный в виде интерпретатора команд. Поддерживает все операции, реализованные в контроллере КК012, кроме тестовых, имеет средства для редактирования командной строки, механизм повторного вызова команды и возможность выполнения команды в цикле;
- *naf* — программа для выполнения команд КАМАК в крейте, предназначена для использования в командных файлах;
- *c12_test* — программа тестирования контроллера крейта (адаптированная версия программы Гилева А.И.[1]).

Прилагаемая к пакету документация написана в виде, пригодном для использования командой *man*, и содержит полное описание библиотеки libc12 и сопутствующих утилит. Для облегчения разработки пользовательского обработчика прерывания в состав пакета включена соответствующая заготовка, а также примеры использования библиотеки.

При тестировании пакета на компьютере с процессором Pentium-100 были получены следующие временные характеристики:

время между установкой запроса и первой командой обработчика прерывания	2.7–4.0 мкс
операция КАМАК (макрокоманда)	2.6–2.8 мкс
операции захвата/освобождения КАМАК (суммарно)	7.9 мкс

Программный пакет находится по адресу:

ftp://silent.msu.dubna.ru:/pub/FreeBSD/camac

В заключение авторы хотят выразить глубокую благодарность В.Н. Дугинову и А.Ю. Исупову, чьи замечания и советы оказали неоценимую помощь в работе.

Приложение. Описание библиотеки libc12

Библиотека libc12 включает в себя:

1. интерфейс драйвера KAMAK;
2. ESONE библиотеку;
3. MUSPIN библиотеку.

ESONE и MUSPIN библиотеки являются самодостаточными, прямой вызов функций из секции 1 требуется только в редких случаях. Функции, выполняющие обращение к контроллеру крейта, используют функции из секции 1 через макрокоманды *c12_MAP*, *c12_DISABLE* и *c12_RESTORE* (см. описание ниже) и работают по следующей схеме:

- отображение адресов интерфейсной платы ПК012 в адресное пространство процесса (*c12_MAP*);
- захват KAMAK в монопольное пользование (*c12_DISABLE*);
- выбор крейта (кроме многокрейтных операций);
- обращение к контроллеру крейта;
- освобождение KAMAK (*c12_RESTORE*).

Таким образом, пользователь библиотеки может не заботиться об обеспечении доступа к адресам интерфейсной платы и имеет возможность явно управлять операциями захвата/освобождения KAMAK с целью оптимизации скорости выполнения программы, например:

```
    ...
c12_DISABLE();
for (i=0; i<1000; i++) c12_write(...);
c12_RESTORE();
    ...
```

В этом примере функция *c12_write* не будет выполнять операции захвата/освобождения KAMAK, поскольку операция захвата KAMAK уже была выполнена до вызова функции.

Интерфейс драйвера KAMAK

Синтаксис

```
#include <kk_device.h>
#include <c12kk.h>
```



```
struct kkaddr_s {
        u_long  kkaddr;
        caddr_t kkbase;
};
struct kklock_s {
        pid_t   kklock;
        u_long  kkwakeup;
};
struct kkintr_s {
        int     kkwait;
        u_long  kkreal;
        u_long  kkserv;
};

int c12_lock(int flag);
int c12_unlock(void);
int c12_release(void);
int c12_get_addr(struct kkaddr_s *addr);
int c12_get_lock(struct kklock_s *lock);
int c12_get_intr(struct kkintr_s *intr);
int c12_clr_lock(void);
int c12_clr_intr(void);
```

Аргументы

kkaddr — физический базовый адрес интерфейсной платы.

kkbase — базовый адрес интерфейсной платы в адресном пространстве ядра.

kklock — текущий статус KAMAK:

0	KAMAK свободен;
1	KAMAK занят драйвером KAMAK;
> 1	идентификатор процесса, захватившего KAMAK.

kkwakeup — статистика операций освобождения KAMAK.

kkwait — флаг отложенного прерывания, ненулевое значение переменной означает наличие необработанного прерывания KAMAK.

kkreal — статистика зарегистрированных прерываний KAMAK.

kkserv — статистика обработанных прерываний KAMAK.

Описание

c12_lock выполняет операцию захвата KAMAK в монопольное пользование. Аргумент *flag* может иметь одно из следующих значений:

0 выполнение функции, прерванное получением перехватываемого сигнала, безусловно возобновляется;

c12_NR выполнение функции, прерванное получением перехватываемого сигнала, зависит от флагов обработчика сигнала (см. описание функции *sigaction*);

c12_NB выполнение процесса в случае, когда KAMAK занят другим процессом, не приостанавливается.

c12_unlock выполняет операцию освобождения KAMAK.

c12_release, если KAMAK занят текущим процессом, — выполняет операцию освобождения KAMAK, в противном случае — аварийно завершает захвативший KAMAK процесс путем посылки ему сигнала SIGKILL и освобождает KAMAK. Функция может вызываться только привилегированным процессом.

c12_get_addr заносит информацию о адресе интерфейсной платы в переменную **addr*.

c12_get_lock заносит информацию о текущем статусе KAMAK и статистике освобождения KAMAK в переменную **lock*.

c12_get_intr заносит информацию о прерываниях KAMAK в переменную **intr*.

c12_clr_lock сбрасывает статистику освобождения KAMAK. Функция может вызываться только привилегированным процессом.

c12_clr_int сбрасывает статистику прерываний KAMAK. Функция может вызываться только привилегированным процессом.

Возвращаемое значение

При успешном завершении возвращается 0. В случае ошибки возвращается −1 и код ошибки заносится во внешнюю переменную *errno*.

Ошибки

[EINTR] Функция *c12_lock*, вызванная с параметром c12_NR, была прервана получением сигнала, обработчик которого не имел флага SA_RESTART.

[EBUSY] Попытка выполнить функцию *c12_lock* с параметром c12_NB, когда KAMAK занят другим процессом.

[EACCESS] Попытка выполнить функцию *c12_unlock*, когда KAMAK не занят вызывающим процессом.

[EPERM] Попытка выполнить функцию *c12_release*, *c12_clr_lock* или *c12_clr_intr* процессом, не имеющим соответствующих привилегий.

[EFAULT] Аргумент функции *c12_get_lock* или *c12_get_intr* содержал недопустимый адрес.

[EDEADLOCK] Попытка выполнить функцию *c12_release*, когда KAMAK занят драйвером KAMAK.

Синтаксис

```
#include <c12map.h>

extern u_long          c12_lock_count;
extern caddr_t         c12_base;

void c12_map(void);
void c12_unmap(void);
void c12_child(void);

c12_DISABLE();
c12_RESTORE();
c12_MAP();
```

Описание

c12_lock_count является счетчиком, используемым функцией *c12_child* и макрокомандами *c12_DISABLE* и *c12_RESTORE*.

c12_base содержит базовый адрес интерфейсной платы ПК012 в адресном пространстве процесса.

c12_map открывает специальный файл /dev/kk и выполняет отображение адресов интерфейсной платы в адресное пространство процесса. В переменную *c12_base* заносит базовый адрес отображенной памяти.

c12_unmap удаляет отображение адресов интерфейсной платы и закрывает файл /dev/kk.

c12_child выполняет реинициализацию отображения адресов интерфейсной платы и обнуляет переменную *c12_lock_count*. Функция должна вызываться в дочернем процессе после вызова *fork*.

c12_DISABLE (макрокоманда) инкрементирует переменную *c12_lock_count* и, если значение переменной *c12_lock_count* равно 1, вызывает функцию *c12_lock* с аргументом 0.

c12_RESTORE (макрокоманда) декрементирует переменную *c12_lock_count* и, если значение переменной *c12_lock_count* равно 0, вызывает функцию *c12_unlock*.

c12_MAP (макрокоманда) вызывает функцию *c12_map*, если отображение адресов интерфейсной платы еще не было выполнено.

Обработка ошибок

В случае обнаружения ошибки сообщение об ошибке выводится в стандартный поток диагностики и выполнение процесса прекращается с кодом завершения 1.

ESONE библиотека

Следующий набор функций выполнен в соответствии со стандартом ESONE «Subroutines for CAMAC» и реализует уровень B стандарта со следующими ограничениями:

- поддерживается только одна ветвь KAMAK;
- доступ к информации о запросах реализован только через команды управления: «Сброс запроса» (F(10)), «Разрешение» (F(26)), «Запрещение» (F(24)) и «Проверка запроса» (F(8)), и, соответственно, указатель доступа *m* всегда интерпретируется как субадрес;
- нулевое значение идентификатора запроса *lam* не указывает на неспецифицированный запрос;
- функция *cclnk* не реализована,

и расширениями:

- некоторые функции возвращают значение, отличное от void;
- добавлены системно-зависимые функции: *ctcr12*, *cci12*, *ccfs12*, *ctfs12*, *ctfl12* и *ctlm12*;
- в функции *ctlm12* идентификатор запроса *lam* используется как идентификатор сигнала «Запрос на внимание» (L);
- функции выполняют проверку допустимости значений своих аргументов.

Синтаксис

```
#include <c12e.h>

typedef int     c12reg_t,c12data_t,c12lam_t;

c12reg_t cdreg(c12reg_t *reg,int b,int c,int n,int a);
int cfsa(int f,c12reg_t reg,c12data_t *data,int *q);
void cccz(c12reg_t reg);
void cccc(c12reg_t reg);
void ccci(c12reg_t reg,int l);
int ctci(c12reg_t reg,int *l);
void cccd(c12reg_t reg,int l);
int ctcd(c12reg_t reg,int *l);
int ctgl(c12reg_t reg,int *l);
c12lam_t cdlam(c12lam_t *lam,int b,int c,int n,int m,int *inta);
void cclm(c12lam_t lam,int l);
```



```
void cclc(c12lam_t lam);
int ctlm(c12lam_t lam,int *l);
void cglam(c12lam_t lam,int *b,int *c,int *n,int *m,int *inta);
void cgreg(c12reg_t reg,int *b,int *c,int *n,int *a);
int ctstat(int *k);
int cssa(int f,c12reg_t reg,c12data_t *data,int *q);
int ctcr12(c12reg_t reg,int *l);
void cci12(c12reg_t reg);
void ccfs12(c12reg_t reg,int fs);
int ctfs12(c12reg_t reg,int *fs);
int ctfl12(c12reg_t reg,int *l);
int ctlm12(c12reg_t reg,int *l);
```

Аргументы

reg — идентификатор адреса KAMAK. Может представлять регистр, крейт или ветвь KAMAK.

b — номер ветви KAMAK, в данной реализации не используется.

c — номер крейта. Допустимые значения:
- 0–6 для указания крейта;
- 7 для указания ветви KAMAK.

n — номер станции. Допустимые значения:
- 0 для указания крейта (контроллера крейта);
- 1–23 для указания станции.

a — субадрес. Допустимые значения 0–15.

f — номер функции. Допустимые значения 0–31.

d — слово данных. Используются младшие 24 (*cfsa*) или 16 (*cssa*) бит, остальные биты игнорируются при командах записи и устанавливаются в 0 при командах чтения.

q — состояние сигнала «Ответ» (Q).

lam — идентификатор источника запроса или сигнала «Запрос на внимание» (L). Может представлять регистр, модуль или контроллер крейта.

m — указатель доступа к источникам запросов, в данной реализации интерпретируется как субадрес. Допустимые значения 0–15.

inta — системно-зависимый аргумент, в данной реализации не используется.

k — состояние сигналов «Ответ» (Q) и «Команда принята» (X).

fs — значение регистра FS контроллера крейта. Допустимые значения 0–31.

Описание

cdreg заносит номер ветви *b*, номер крейта *c*, номер станции *n* и субадрес *a* в переменную **reg*. Дополнительно функция возвращает величину **reg*.

cfsa выполняет команду КАМАК *f* в регистре *reg*. В случае команды чтения или записи производит передачу 24-битного слова данных между регистром и переменной **data*, в противном случае аргумент *data* не используется. Переменной **q* присваивает ненулевое значение, если сигнал «Ответ» (Q) после выполнения команды установлен, и нуль — в противном случае. Дополнительно функция возвращает величину **q*.

cccz генерирует сигнал «Пуск» (Z) в крейте *reg*.

cccc генерирует сигнал «Сброс» (C) в крейте *reg*.

ccci, если аргумент *l* имеет ненулевое значение — устанавливает, в противном случае — сбрасывает сигнал «Запрет» (I) в крейте *reg*.

ctci присваивает переменной **l* ненулевое значение, если сигнал «Запрет» (I) в крейте *reg* установлен, и нуль — в противном случае. Дополнительно функция возвращает величину **l*.

cccd, если аргумент *l* имеет ненулевое значение — разрешает, в противном случае — запрещает генерирование сигнала прерывания контроллером крейта *reg*.

ctcd присваивает переменной **l* ненулевое значение, если генерирование сигнала прерывания контроллером крейта *reg* разрешено, и нуль — в противном случае. Дополнительно функция возвращает величину **l*.

ctgl присваивает переменной **l* ненулевое значение, если флаг запроса прерывания в контроллере крейта *reg* установлен, и нуль — в противном случае. Дополнительно функция возвращает величину **l*.

cdlam заносит номер ветви *b*, номер крейта *c*, номер станции *n* и указатель доступа *m* в переменную **lam*. Дополнительно функция возвращает величину **lam*.

cclm, если аргумент *l* имеет ненулевое значение — разрешает, в противном случае — запрещает установку запроса источником *lam*.

cclc сбрасывает состояние источника запроса *lam*.

ctlm присваивает переменной **l* ненулевое значение, если запрос источником *lam* установлен, и нуль — в противном случае. Дополнительно функция возвращает величину **l*.

cglam декодирует идентификатор источника запроса *lam* и заносит в переменные **b*, **c*, **n* и **m* номер ветви, номер крейта, номер станции и указатель доступа соответственно.

cgreg декодирует идентификатор адреса КАМАК *reg* и заносит в переменные **b*, **c*, **n* и **a* номер ветви, номер крейта, номер станции и субадрес соответственно.

ctstat присваивает переменной **k* значение, определяемое состоянием сигналов «Ответ» (Q) и «Команда принята» (X) после выполнения последней команды КАМАК в соответствии с таблицей:

0 Q=1, X=1;
1 Q=0, X=1;
2 Q=1, X=0;
3 Q=0, X=0.

Дополнительно функция возвращает величину **k*.

cssa выполняет команду КАМАК *f* в регистре *reg*. В случае команды чтения или записи производит передачу 16-битного слова данных между регистром и переменной **data*, в противном случае аргумент *data* не используется. Переменной **q* присваивает ненулевое значение, если сигнал «Ответ» (Q) после выполнения команды установлен, и нуль — в противном случае. Дополнительно функция возвращает величину **q*.

ctcr12 присваивает переменной **l* ненулевое значение, если крейт *reg* присутствует в системе, и нуль — в противном случае. Дополнительно функция возвращает величину **l*.

cci12 выполняет инициализацию контроллера крейта *reg*.

ccfs12 устанавливает регистр FS контроллера крейта *reg* в состояние *fs*.

ctfs12 присваивает переменной **fs* значение регистра FS контроллера крейта *reg*. Дополнительно функция возвращает величину **fs*.

ctfl12 присваивает переменной **l* ненулевое значение, если флаг FL в контроллере крейта *reg* установлен, и нуль — в противном случае. Дополнительно функция возвращает величину **l*.

ctlm12 присваивает переменной **l* ненулевое значение, если сигнал «Запрос на внимание» (L) источником *lam* установлен, и нуль — в противном случае. Дополнительно функция возвращает величину **l*.

Обработка ошибок

В случае обнаружения ошибки в аргументах функции вызывается функция *abort*.

MUSPIN библиотека

Синтаксис

```
#include <c12c.h>

extern int      c12_check;
extern unsigned c12_crate;
extern int      c12_camac_x;
extern int      c12_camac_q;
```

```
int c12_crate_exists(unsigned c);
void c12_set_crate(unsigned c);
unsigned c12_get_crate(void);
void c12_reset_crate(void);
void c12_zero_crate(void);
void c12_zero_all(void);
void c12_clear_crate(void);
void c12_clear_all(void);
void c12_inhibit_crate(int on);
void c12_inhibit_all(int on);
int c12_inhibit_state(void);
void c12_set_int(int on);
int c12_int_state(void);
int c12_int_exists(void);
void c12_set_fs(unsigned fs);
unsigned c12_get_fs(void);
int c12_fl_state(void);
int c12_lam_state(unsigned lam);
unsigned c12_all_lams(void);
void c12_wait_lam(unsigned lam);
unsigned int c12_read(unsigned n, unsigned a, unsigned f);
unsigned int c12_read24(unsigned n, unsigned a, unsigned f);
void c12_write(unsigned n, unsigned a, unsigned f, unsigned d);
void c12_write24(unsigned n, unsigned a, unsigned f, unsigned d);
void c12_oper(unsigned n, unsigned a, unsigned f);
```

Аргументы

c — номер крейта. Допустимые значения 0–6.

fs — значение регистра FS контроллера крейта. Допустимые значения 0–31.

lam — номер сигнала «Запрос на внимание» (L). Допустимые значения 0–23.

n — номер станции. Допустимые значения 1–23.

a — субадрес. Допустимые значения 0–15.

f — номер функции. Допустимые значения:

0–7	для функций *c12_read* и *c12_read24*;
8–15, 24–31	для функции *c12_oper*;
16–23	для функций *c12_write* и *c12_write24*.

d — слово данных. Допустимые значения:

0–0xffff	для функции *c12_write*;
0–0xffffff	для функции *c12_write24*.

Описание

c12_check управляет проверкой аргументов функций: при ненулевом значении переменной выполняется проверка допустимости аргументов функций, в противном случае такая проверка не выполняется.

c12_crate содержит номер текущего активного крейта. Все функции, кроме *c12_crate_exists*, *c12_set_crate*, *c12_zero_all*, *c12_clear_all* и *c12_inhibit_all*, работают с текущим активным крейтом.

c12_camac_x содержит ненулевое значение, если после выполнения последней команды КАМАК в крейте был установлен сигнал «Команда принята» (X), и нуль — в противном случае.

c12_camac_q содержит ненулевое значение, если после выполнения последней команды КАМАК в крейте был установлен сигнал «Ответ» (Q), и нуль — в противном случае.

c12_crate_exists возвращает ненулевое значение, если крейт *c* присутствует в системе, и нуль — в противном случае.

c12_set_crate назначает крейт *c* текущим активным крейтом. Назначение крейта не связано с выбором крейта.

c12_get_crate возвращает номер текущего активного крейта.

c12_reset_crate выполняет инициализацию контроллера крейта.

c12_zero_crate генерирует сигнал «Пуск» (Z) в крейте.

c12_zero_all генерирует сигнал «Пуск» (Z) во всех крейтах.

c12_clear_crate генерирует сигнал «Сброс» (C) в крейте.

c12_clear_all генерирует сигнал «Сброс» (C) во всех крейтах.

c12_inhibit_crate, если аргумент *on* имеет ненулевое значение — устанавливает, в противном случае — сбрасывает сигнал «Запрет» (I) в крейте.

c12_inhibit_all, если аргумент *on* имеет ненулевое значение — устанавливает, в противном случае — сбрасывает сигнал «Запрет» (I) во всех крейтах.

c12_inhibit_state возвращает ненулевое значение, если сигнал «Запрет» (I) в крейте установлен, и нуль — в противном случае.

c12_set_int, если аргумент *on* имеет ненулевое значение — разрешает, в противном случае — запрещает генерирование сигнала прерывания контроллером крейта.

c12_int_state возвращает ненулевое значение, если генерирование сигнала прерывания контроллером крейта разрешено, и нуль — в противном случае.

c12_int_exists возвращает ненулевое значение, если флаг запроса прерывания в контроллере крейта установлен, и нуль — в противном случае.

c12_set_fs устанавливает регистр FS контроллера крейта в состояние *fs*.

c12_get_fs возвращает состояние регистра FS контроллера крейта.

c12_fl_state возвращает ненулевое значение, если флаг FL в контроллере крейта установлен, и нуль — в противном случае.

c12_lam_state возвращает ненулевое значение, если сигнал «Запрос на внимание» (L) источником *l* в крейте установлен, и нуль — в противном случае.

c12_all_lams возвращает слово, в котором i-ый бит имеет значение 1, если сигнал «Запрос на внимание» (L) i-ым источником в крейте установлен, и значение 0 — в противном случае.

c12_wait_lam ожидает появления в крейте сигнала «Запрос на внимание» (L) от источника *l*.

c12_read возвращает 16-битное слово данных, прочитанное из модуля *n*, субадрес *a* по команде *f*.

c12_read24 возвращает 24-битное слово данных, прочитанное из модуля *n*, субадрес *a* по команде *f*.

c12_write записывает 16-битное слово данных *d* в модуль *n*, субадрес *a* по команде *f*.

c12_write24 записывает 24-битное слово данных *d* в модуль *n*, субадрес *a* по команде *f*.

c12_oper выполняет управляющую команду *f* в модуле *n*, субадрес *a*.

Обработка ошибок

В случае обнаружения ошибки сообщение об ошибке выводится в стандартный поток диагностики и выполнение процесса прекращается с кодом завершения 1.

Литература

[1] Георгиев А., Чурин И.Н. ОИЯИ, Р10-88-381, Дубна, 1988.

[2] Антюхов В.А. и др. ОИЯИ, Р10-90-589, Дубна, 1990.

[3] Ольшевский В.Г., Помякушин В.Ю. ОИЯИ, Р10-94-416, Дубна, 1994.

[4] Bach M.J. The design of the UNIX operating system, Prentice-Hall Corp., New Jersey, 1986.

[5] ESONE Standard «Subroutine for CAMAC», ESONE/SR/01, 1978.

Рукопись поступила в издательский отдел
9 июня 1998 года.