ДРАЙВЕР DELCORE30M. РУКОВОДСТВО ПРОГРАММИСТА

Версия v3.1

01.11.2019

ОГЛАВЛЕНИЕ

1. О ДОКУМЕНТЕ

Документ содержит описание драйвера для DSP-кластера DELcore-30M, состоящего из двух DSP-ядер (далее DSP) ELcore-30M.

В документе используются ссылки на системный контроллер DMA (SDMA). Описание SDMA приведено в документе Микросхема интегральная 1892ВМ14Я. Руководство пользователя[1].

[1] http://multicore.ru/mc/data_sheets/Manual_1892VM14YA.pdf

2. ОБЩЕЕ ОПИСАНИЕ

Интерфейс драйвера *delcore30m* предоставляет функциональность:

1. Управление памятью XYRAM (выделение, освобождение, mmap).
2. Выделение, импорт, экспорт непрерывных буферов DMA в системной памяти (DDR).
3. Предоставление доступа к PRAM DSP.
4. Управление ресурсами: DSP, SDMA (выделение, освобождение).
5. Подготовка программы для SDMA.
6. Отправка заданий на DSP. Возможно параллельное выполнение нескольких задач на нескольких DSP, одной задачи на нескольких DSP. Уведомление о готовности заданий.
7. Отмена заданий.
8. Профилирование кода DSP.

Драйвер *delcore30m* использует следующие ресурсы 1892ВМ14Я:

1. цифровой сигнальный процессор DELcore-30M,
2. системный DMA-контроллер SDMA,
3. блок поддержки атомарных операций SPINLOCK.

Ограничения драйвера:

1. Не поддерживается одновременная работа DSP и VPU.
2. Драйвер SDMA *pl330* должен быть выгружен.
3. Адреса памяти XYRAM, используемые в DMA-пересылках, должны быть выровнены по границе четыре байта, см. главу 4.3.1 в Микросхема интегральная 1892ВМ14Я. Руководство пользователя[2].
4. Запрещено использовать глобальные переменные в коде для DSP, написанном на C.
5. Профилирование возможно только для кода, написанного на ассемблере.
6. Не поддерживается одновременная передача нескольких SDMA каналов разных типов.
7. Не поддерживается работа драйвера более, чем из двух процессов.

[2] http://multicore.ru/mc/data_sheets/Manual_1892VM14YA.pdf

3. СТАНДАРТНЫЕ СИСТЕМНЫЕ ВЫЗОВЫ

Интерфейс реализует стандартные для файловых дескрипторов системные вызовы со следующими особенностями:

- `mmap()`
  - При указании файлового дескриптора устройства возвращает ошибку.
  - При указании файлового дескриптора буфера отображает память буфера в карту памяти процесса.
  - При указании файлового дескриптора задания возвращает ошибку.
  - При указании файлового дескриптора ресурса DSP отображает память программ DSP N, где N = offset / 4096, offset — последний аргумент функции `mmap()`.
- `poll()`
  - При указании файлового дескриптора задания ожидает завершения задания (задание считается завершившимся, если его статус — `DELCORE30M_JOB_IDLE`).
  - При указании файлового дескриптора буфера, дескриптор игнорируется.
  - При указании файлового дескриптора устройства, дескриптор игнорируется.

Также интерфейс реализует системный вызов `ioctl()` с функциями, описанными ниже.

4. ОПИСАНИЕ IOCTL

Перед вызовом функций `ioctl()` необходимо открыть устройство `/dev/elcore0` посредством вызова функции `open()`, которая возвращает файловый дескриптор устройства.

4.1 ELCIOC_BUF_ALLOC

4.1.1 Назначение

Выделение непрерывного буфера в XYRAM или DDR.

4.1.2 Объявление

```
enum delcore30m_memory_type {
  DELCORE30M_MEMORY_XYRAM,
  DELCORE30M_MEMORY_SYSTEM
};

struct delcore30m_buffer {
  int fd;
  enum delcore30m_memory_type type;
  size_t size;
};

int ioctl(int fd, ELCIOC_BUF_ALLOC, struct delcore30m_buffer *buf);
```

4.1.3 Аргументы

- `fd` — дескриптор устройства драйвера.
- `buf` — указатель на структуру *`delcore30m_buffer`*.

4.1.4 Типы аргументов

`delcore30m_buffer`

`fd`
: Файловый дескриптор буфера.

`type`
: Тип памяти, в которой выделяется буфер:

  - `DELCORE30M_MEMORY_XYRAM` — память данных XYRAM.
  - `DELCORE30M_MEMORY_SYSTEM` — общая системная память DDR.

size
: Размер выделяемого буфера в байтах. Для каждого DSP драйвер выделяет стек размером 4 КБ. Максимально доступный объем памяти XYRAM для каждого DSP равен 124 КБ.

4.1.5 Описание

Перед вызовом `ioctl()` необходимо заполнить поля *`delcore30m_buffer.type`* и *`delcore30m_buffer.size`*.

В случае успеха вызов `ioctl()` возвращает нулевое значение, при этом значение *`delcore30m_buffer.fd`* соответствует значению файлового дескриптора выделенного буфера.

Для освобождения выделенного буфера необходимо вызвать `close()` с указанием *`delcore30m_buffer.fd`*.

Ошибки, возвращаемые `ioctl()`:

- `ENOMEM` — недостаточно свободной памяти *`delcore30m_buffer.type`* для выделения данного буфера.
- `EINVAL` — значение поля *`delcore30m_buffer.type`* некорректно.

4.2 ELCIOC_RESOURCE_REQUEST

4.2.1 Назначение

Выделение ресурсов.

4.2.2 Объявление

```
enum delcore30m_resource_type {
  DELCORE30M_CORE,
  DELCORE30M_SDMA,
};

struct delcore30m_resource {
  int fd;
  enum delcore30m_resource_type type;
  unsigned int num;
  unsigned long mask;
};

int ioctl(int fd, ELCIOC_RESOURCE_REQUEST, struct delcore30m_resource *res);
```

4.2.3 Аргументы

- `fd` — дескриптор устройства драйвера.
- `res` — указатель на структуру *`delcore30m_resource`*.

4.2.4 Типы аргументов

delcore30m_resource

fd
Файловый дескриптор ресурса.

type
Тип выделяемого ресурса:

- DELCORE30M_CORE — DSP.
- DELCORE30M_SDMA — каналы SDMA.

num
Количество единиц выделяемого ресурса.

mask
Маска выделяемого ресурса.

4.2.5 Описание

Перед вызовом ioctl() необходимо заполнить поля *delcore30m_resource.type* и *delcore30m_resource.num*.

В случае успеха вызов ioctl() возвращает нулевое значение, при этом значение *delcore30m_resource.fd* соответствует значению файлового дескриптора выделенного буфера, а значение *delcore30m_resource.mask* будет содержать маску номеров выделенных ресурсов.

Для освобождения выделенного буфера необходимо вызвать close() с указанием *delcore30m_resource.fd*.

Ошибки, возвращаемые ioctl():

- ENOMEM — в системе недостаточно свободной памяти DDR.
- EINVAL — значения полей *delcore30m_resource.type*, *delcore30m_resource.num* некорректны. Количество ресурсов типа DELCORE30M_CORE не может превышать 2. Количество ресурсов типа DELCORE30M_SDMA не может превышать 8.
- EBUSY — запрашиваемые ресурсы в данный момент заняты. Необходимо вызвать ioctl() в другое время.

4.3 ELCIOC_JOB_CREATE

4.3.1 Назначение

Создание задания для DSP.

4.3.2 Объявление

```
#define MAX_INPUTS 15
#define MAX_OUTPUTS 15

#define DELCORE30M_PROFILE (1 << 0)

enum delcore30m_job_status {
  DELCORE30M_JOB_IDLE,
  DELCORE30M_JOB_ENQUEUED,
  DELCORE30M_JOB_RUNNING,
};

enum delcore30m_job_rc {
  DELCORE30M_JOB_ERROR = -2,
  DELCORE30M_JOB_CANCELLED = -1,
  DELCORE30M_JOB_SUCCESS = 0
};

struct delcore30m_job {
  int fd;

  unsigned int inum;
  unsigned int onum;

  int input[MAX_INPUTS];
  int output[MAX_OUTPUTS];

  int cores_fd;
  int sdmas_fd;

  enum delcore30m_job_status status;
  enum delcore30m_job_rc rc;

  __u32 flags;
};

int ioctl(int fd, ELCIOC_JOB_CREATE, struct delcore30m_job *job);
```

4.3.3 Аргументы

- `fd` — дескриптор устройства драйвера.
- `job` — указатель на структуру *`delcore30m_job`*.

4.3.4 Типы аргументов

`delcore30m_job`

`fd`

Файловый дескриптор задания.

inum
Количество входных буферов. Максимально допустимое значение — 15.

onum
Количество выходных буферов. Максимально допустимое значение — 15.

input
Массив файловых дескрипторов входных буферов размера *delcore30m_job.inum*, выделенных через *ELCIOC_BUF_ALLOC*. Максимально допустимое значение — 15.

output
Массив файловых дескрипторов выходных буферов размера *delcore30m_job.onum*, выделенных через *ELCIOC_BUF_ALLOC*. Максимально допустимое значение — 15.

cores_fd
Файловый дескриптор ресурса DSP, выделенного через *ELCIOC_RESOURCE_REQUEST*, на которых может быть запущено задание.

sdmas_fd
Дескриптор каналов SDMA, выделенный через *ELCIOC_RESOURCE_REQUEST*, которые могут быть использованы заданием. Если для выполнения задания не нужно использовать SDMA, установить это поле в ноль.

status
Статус задания:

- `DELCORE30M_JOB_IDLE` — задача либо выполнена, либо еще не помещена в очередь на выполнение, см. *ELCIOC_JOB_ENQUEUE*.
- `DELCORE30M_JOB_ENQUEUED` — задача находится в очереди на выполнение на DSP.
- `DELCORE30M_JOB_RUNNING` — задача выполняется на DSP.

rc
Код возврата последнего завершённого задания:

- `DELCORE30M_JOB_ERROR` — задача завершилась с ошибкой.
- `DELCORE30M_JOB_CANCELED` — задача была отменена.
- `DELCORE30M_JOB_SUCCESS` — задача успешно завершилась.

flags
Флаги задания. Допустимые флаги:

- `DELCORE30M_PROFILE` — флаг профилирования DSP-программы, см. *Профилирование программы, исполняемой на DSP*.

4.3.5 Описание

Перед вызовом `ioctl()` необходимо заполнить поля *`delcore30m_job.inum`*, *`delcore30m_job.onum`*, *`delcore30m_job.input`*, *`delcore30m_job.output`* и *`delcore30m_job.flags`*.

В случае успеха вызов `ioctl()` возвращает нулевое значение, при этом значение поля *`delcore30m_job.fd`* соответствует значению файлового дескриптора созданного задания, а поле *`delcore30m_job.status`* равно `DELCORE30M_JOB_IDLE`.

Для уничтожения задания необходимо вызвать `close()` с указанием *`delcore30m_job.fd`*

Ошибки, возвращаемые `ioctl()`:

- `ENOMEM` — в системе недостаточно свободной памяти DDR.
- `EBADFD` — значение поля *`delcore30m_job.cores_fd`* некорректно.
- `EPERM` — в PRAM каждого DSP, для которых создается задание, не загружен исполняемый код, см. описание системного вызова `mmap()` в *Стандартные системные вызовы*.
- `EINVAL` — поля *`delcore30m_job.inum`* и *`delcore30m_job.onum`* выходят за максимальную границу возможных значений.

4.4 ELCIOC_JOB_ENQUEUE

4.4.1 Назначение

Добавление задания в очередь на выполнение.

4.4.2 Объявление

```
int ioctl(int fd, ELCIOC_JOB_ENQUEUE, struct delcore30m_job *job);
```

4.4.3 Аргументы

- `fd` — дескриптор устройства драйвера.
- `job` — указатель на структуру *`delcore30m_job`*, который был получен после вызова *ELCIOC_JOB_CREATE*.

4.4.4 Описание

Если DSP находится в состоянии ожидания, то производится старт выполнения задачи. В противном случае, задача будет запущена после завершения задачи, выполняемой на DSP.

В случае успеха вызов `ioctl()` возвращает нулевое значение.

Ошибки, возвращаемые `ioctl()`:

- `EBADFD` — значение поля *`delcore30m_job.fd`* некорректно.

- `EBUSY` — переданная задача *`delcore30m_job`* в очереди на исполнение или выполняется на DSP.

4.5 ELCIOC_JOB_STATUS

4.5.1 Назначение

Получение статуса задания.

4.5.2 Объявление

```
int ioctl(int fd, ELCIOC_JOB_STATUS, struct delcore30m_job *job);
```

4.5.3 Аргументы

- `fd` — дескриптор устройства драйвера.
- `job` — указатель на структуру *`delcore30m_job`*, который был получен после вызова *ELCIOC_JOB_CREATE*.

4.5.4 Описание

В случае успеха вызов `ioctl()` возвращает нулевое значение, при этом значение поля *`delcore30m_job.status`* соответствует текущему статуса задания.

Ошибки, возвращаемые `ioctl()`:

- `EBADFD` — значение поля *`delcore30m_job.fd`* некорректно.

4.6 ELCIOC_JOB_CANCEL

4.6.1 Назначение

Отмена задания.

4.6.2 Объявление

```
int ioctl(int fd, ELCIOC_JOB_CANCEL, struct delcore30m_job *job);
```

4.6.3 Аргументы

- `fd` — дескриптор устройства драйвера.
- `job` — указатель на структуру *`delcore30m_job`*, который был получен после вызова *ELCIOC_JOB_CREATE*.

4.6.4 Описание

Задание удаляется из всех очередей. Если задание находится в стадии выполнения, то осуществляется остановка DSP.

В случае успеха вызов `ioctl()` возвращает нулевое значение.

Ошибки, возвращаемые `ioctl()`:

- `EBADFD` — значение поля *`delcore30m_job.fd`* некорректно.

4.7 ELCIOC_SYS_INFO

4.7.1 Назначение

Получение информации об аппаратуре.

4.7.2 Объявление

```
struct delcore30m_hardware {
  int ncores;
  size_t xyram_size;
  size_t core_pram_size;
};

int ioctl(int fd, ELCIOC_SYS_INFO, delcore30m_hardware *hw);
```

4.7.3 Аргументы

- `fd` — дескриптор устройства драйвера.
- `hw` — указатель на структуру *`delcore30m_hardware`*.

4.7.4 Типы аргументов

`delcore30m_hardware`

`ncores`
: Количество DSP в кластере.

`xyram_size`
: Суммарный объем памяти данных XYRAM обоих DSP без учета зарезервированных драйвером участков памяти, см. *`delcore30m_buffer.size`*.

`core_pram_size`
: Размер памяти программ PRAM для каждого DSP.

4.7.5 Описание

В случае успеха вызов `ioctl()` возвращает нулевое значение.

4.8 ELCIOC_DMACHAIN_SETUP

4.8.1 Назначение

Настройка канала DMA для заданной DMA цепочки.

4.8.2 Объявление

```
enum sdma_descriptor_type {
  SDMA_DESCRIPTOR_E1I1,
  SDMA_DESCRIPTOR_E1I0,
  SDMA_DESCRIPTOR_E0I0,
  SDMA_DESCRIPTOR_E0I1,
};

struct sdma_descriptor {
  __u32 a0e;
  __u32 a0i;
  __u32 astride;
  __u32 bcnt;
  __u32 ccr;
  __u32 asize;
  enum sdma_descriptor_type type;
  __u32 a_init;
};

enum sdma_channel_type {
  SDMA_CHANNEL_INPUT,
  SDMA_CHANNEL_OUTPUT,
};

struct sdma_channel {
  enum sdma_channel_type type;
  unsigned int num;
};

struct delcore30m_dmachain {
  int codebuf;
  int core;
  int external;
  int internal[2];
  int chain;
  int job;
  struct sdma_channel channel;
};

int ioctl(int fd, ELCIOC_DMACHAIN_SETUP, struct delcore30m_dmachain *chain);
```

4.8.3 Аргументы

- `fd` — файловый дескриптор устройства драйвера.
- `chain` — указатель на структуру *`delcore30m_dmachain`*.

4.8.4 Типы аргументов

`delcore30m_dmachain`

`codebuf`
Файловый дескриптор буфера исполняемого кода для канала SDMA. Дескриптор должен быть получен с помощью *ELCIOC_BUF_ALLOC*. Буфер должен быть выделен в DDR. Размер буфера должен быть не менее количества передаваемых буферов умноженного на 60.

`core`
Номер DSP, которому будут поступать прерывания от канала SDMA. Номер DSP записывается, исходя из значения *`delcore30m_resource.mask`* для ресурса `DELCORE30M_CORE`, который был выделен через *ELCIOC_RESOURCE_REQUEST*.

`external`
Файловый дескриптор буфера в DDR, выделенный с помощью *ELCIOC_BUF_ALLOC*.

`internal`
Массив двух файловых дескрипторов внутренних буферов в XYRAM, выделенных с помощью *ELCIOC_BUF_ALLOC*. Внутренние буфера участвуют в двухбуферной системе SDMA-пересылок, описание которой приведено ниже.

`chain`
Файловый дескриптор буфера дескрипторов *`sdma_descriptor`* SDMA. Дескриптор выделяется с помощью *ELCIOC_BUF_ALLOC*. Буфера дескрипторов SDMA должны иметь заполненные поля:

- *`sdma_descriptor.a0e`*
- *`sdma_descriptor.astride`*
- *`sdma_descriptor.bcnt`*
- *`sdma_descriptor.ccr`*
- *`sdma_descriptor.asize`*
- *`sdma_descriptor.a_init`*

Последний дескриптор SDMA должен иметь поле *`sdma_descriptor.a_init`* установленное в 0.

`job`
Файловый дескриптор задания *`delcore30m_job`*, выделенный с помощью *ELCIOC_JOB_CREATE*.

channel
Структура см. *sdma_channel* описания канала SDMA.

sdma_descriptor

a0e
Смещение дескриптора буфера данных для обработки в байтах относительно начала буфера в DDR.

a0i
Зарезервировано.

astride
Расстояние в байтах между соседними строками данных, находящихся в DDR.

bcnt
Количество строк, передаваемых в одном буфере.

ccr
Конфигурационный регистр канала SDMA. Формат регистра соответствует регистру управления для канала `CCRn` SDMA. Необходимо заполнить следующие поля, описанные в Микросхема интегральная 1892ВМ14Я. Руководство пользователя[3]:

- `src_inc` — автоинкремент адреса источника.
- `src_burst_size` — разрядность одной пересылки из источника внутри пакета может быть 4 или 8 байт, при этом значение поля *sdma_descriptor.asize* должно быть кратно `src_burst_size`.
- `dst_inc` — автоинкремент адреса приемника.
- `dst_burst_size` — разрядность одной пересылки приемнику внутри. пакета
- `endian_swap_size` — перестановка порядка байт (опционально).

Остальные поля должны быть заполнены нулями.

asize
Размер в байтах строки передаваемого буфера.

type
Тип дескриптора SDMA:

- `SDMA_DESCRIPTOR_E1I1` — для начала передачи буфера необходимо отправить событие SDMA. По завершении передачи SDMA генерирует прерывание.
- `SDMA_DESCRIPTOR_E1I0` — для начала передачи буфера необходимо отправить событие SDMA. По завершении передачи SDMA прерывние не генерируется.

[3] http://multicore.ru/mc/data_sheets/Manual_1892VM14YA.pdf

- `SDMA_DESCRIPTOR_E0I0` — передача буфера начнется сразу после завершения передачи предыдущего. По завершении передачи прерывание не генерируется.
- `SDMA_DESCRIPTOR_E0I1` — передача буфера начнется сразу после завершения передачи предыдущего. По завершении передачи SDMA генерирует прерывание.

`a_init`
Смещение до следующего дескриптора SDMA относительно текущего дескриптора. Последний дескриптор должен иметь поле `a_init`, равное нулю.

`sdma_channel`

`type`
Тип канала SDMA:

- `SDMA_CHANNEL_INPUT` — входной (передача из DDR в XYRAM).
- `SDMA_CHANNEL_OUTPUT` — выходной (передача из XYRAM в DDR).

`num`
Номер канала SDMA, который записывается исходя из значения *`delcore30m_resource.mask`* для ресурса `DELCORE30M_SDMA`, который был выделен через *ELCIOC_RESOURCE_REQUEST*.

4.8.5 Описание

Ввиду ограниченности памяти XYRAM, передаваемые данные необходимо разбить на фрагменты одинакового размера (размер таких фрагментов не должен превышать 58 КБ), для каждого из которых составляется дескриптор. Драйвер, основываясь на полученных дескрипторах, составляет программу для SDMA и загружает ее в DDR. Программа SDMA состоит из ожидания события для передачи очередного буфера, самой передачи буфера и посылки прерывания DSP, сигнализирующим о завершении передачи очередного буфера. Таким образом, программа DSP должна посылать событие каналу SDMA (см. документацию на SDMA), ожидать прерывание от SDMA и после этого обрабатывать полученный буфер. Драйвер для i-ого канала SDMA настраивает событие с номером 8+i и прерывание с номером i. На рисунке 4.1 приведена диаграмма взаимодействия DSP и SDMA при двухбуферной обработке. Числами обозначены номера буферов, загружаемых SDMA. Синим цветом обозначена часть кода, выполняемая на DSP и отвечающая за обработку прерываний, ожидание и запуск SDMA-каналов.

Примечание: Для каждой задачи может быть задействовано несколько SDMA-каналов. Для каждого канала необходимо по отдельности вызывать *ELCIOC_DMACHAIN_SETUP*.

Перед вызовом `ioctl()` необходимо заполнить все поля структуры *`delcore30m_dmachain`*, кроме *`sdma_descriptor.a0i`*. Значение поля *`sdma_descriptor.a0i`* драйвер получает через *`delcore30m_dmachain.internal`*.

В случае успеха вызов `ioctl()` возвращает нулевое значение.

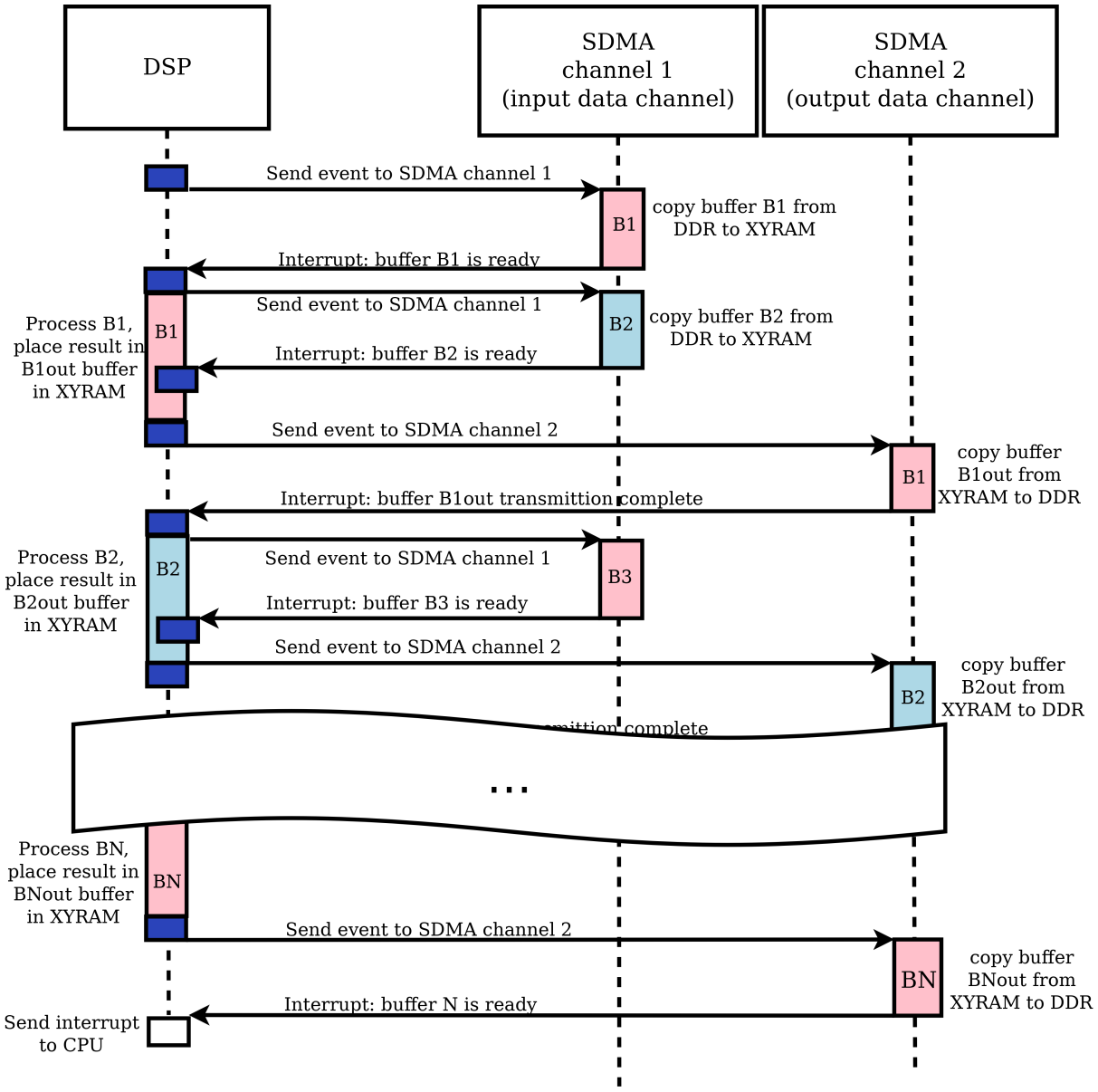

Рисунок 4.1. Диаграмма последовательности взаимодействия DSP и SDMA

Ошибки, возвращаемые `ioctl()`:

- `EBUSY` — канал с номером *`sdma_channel.num`* уже запущен.
- `EBADF` — значение поля *`delcore30m_dmachain.job`* некорректно.
- `ENOMEM` — в системе недостаточно свободной памяти DDR.
- `EFAULT` — неправильно составлена цепочка SDMA-дескрипторов.

4.9 ELCIOC_GET_CAPS

4.9.1 Назначение

Получение информации о драйвере.

4.9.2 Объявление

```
struct elcore_caps {
  char drvname[32];
  __u32 hw_id;
};

int ioctl(int fd, ELCIOC_GET_CAPS, elcore_caps *caps);
```

4.9.3 Аргументы

- `fd` — дескриптор устройства драйвера.
- `caps` — указатель на структуру *`elcore_caps`*.

4.9.4 Типы аргументов

`elcore_caps`

`drvname`
Название драйвера.

`hw_id`
Идентификатор DSP из регистра IDR.

4.9.5 Описание

В случае успеха вызов `ioctl()` возвращает нулевое значение, при этом значение поля *`elcore_caps.drvname`* равно значению `delcore30m`, а значение поля *`elcore_caps.hw_id`* — идентификатору DSP.

5. ПЕРЕДАЧА ПАРАМЕТРОВ DSP

Аргументы, представляющие адреса буферов, которые передаются DSP, размещаются в регистрах R2 и R4. Если количество аргументов больше двух, то все последующие аргументы помещаются в стеке. Порядок передачи аргументов через стек — прямой (у аргумента с меньшим порядковым номером меньший адрес в стеке). Аргументы в стеке выровнены по границе 8 байт. Адрес третьего аргумента содержится в регистре A7. Драйвер сохраняет адреса буферов в байтах. Программа для DSP, написанная на ассемблере, должна преобразовать адреса в словную адресацию. Для программ, написанных на языке C, все необходимые преобразования выполняются компилятором.

Для одновременного выполнения задачи на двух DSP программе необходимо в буферах входных аргументов предусмотреть хранение данных для обоих DSP. Через регистр R0 DSP передается порядковый номер потока, по которому программа для DSP должна определить смещение до своей части аргументов, хранящихся в буфере, относительно адреса буфера аргумента.

Пример:

Пусть задача запускается на двух DSP, и в качестве входного аргумента задачи является структура `struct foo`.

Пользовательская программа должна:

- Выделить буфер с помощью *ELCIOC_BUF_ALLOC* размером `2 * sizeof(struct foo)`.
- Скопировать в буфер данные структуры `foo`, предназначенные для первого DSP.
- Скопировать в буфер данные структуры `foo`, предназначенные для второго DSP, со смещением, равным значению `sizeof(struct foo)`.

Программа DSP для получения доступа к структуре `foo` должна:

- Содержимое регистра RO, через который передается номер потока (для первого DSP R0 = 0, для второго DSP R0 = 1), умножить на значение `sizeof(struct foo)`.
- Полученное число прибавить к адресу буфера, адрес которого передается согласно *Передача параметров DSP*.

6. ПРОФИЛИРОВАНИЕ ПРОГРАММЫ, ИСПОЛНЯЕМОЙ НА DSP

При установке флага `DELCORE30M_PROFILE` в *`delcore30m_job.flags`* драйвер выделяет в DDR буфер, размер которого равен 32 КБ. Адрес буфера передается в DSP через регистр A5. Программа на DSP должна записать в буфер два 32-разрядных числа в начале и конце каждого интересующего участка кода:

- метка, однозначно идентифицирующая данный участок кода
- значение регистра счетчика тактов `TOTAL_CLK_CNTR`

После завершения программы на DSP драйвер для каждой метки считает количество вызовов данного участка кода, минимальное, максимальное и среднее времена выполнения (в тактах).

АЛФАВИТНЫЙ УКАЗАТЕЛЬ

D

E

S