ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПОДДЕРЖКИ

Модуль на основе СНК NM6408

Руководство программиста

Листов 44

АННОТАЦИЯ

Данное руководство описывает назначение и состав средств программного обеспечения поддержки модулей на основе СНК NM6408 (далее по тексту - "ПО поддержки" и "модуль"), процедуру установки ПО поддержки и правила использования ПО поддержки для организации работы с модулем. Подробное описание правил работы с модулем приводится в соответствующем руководстве по эксплуатации.

Данное руководство применимо для следующих модулей: MC127.05; NM_Card; NM_Mezzo; NM_Quad.

Модуль предназначен для работы в составе ПЭВМ с системной шиной PCIe для отработки функционального программного обеспечения вычислительных систем на базе микросхемы интегральной 1879ВМ8Я (далее по тексту – "система на кристалле" или "СнК"). Подробное описание архитектуры СнК приводится в соответствующем руководстве по эксплуатации (ЮФКВ.431282.020РЭ).

СОДЕРЖАНИЕ

1. ПОДГОТОВКА К РАБОТЕ

В качестве предварительного условия установки ПО поддержки требуется:

- установить кросс-средства NMC-SDK;
- установить кросс-средства ARM-SDK;
- экземпляр модуля.

Установка модуля производится согласно соответствующим инструкциям по установке.

Установка ПО поддержки состоит из следующих шагов:

1) установка файлов ПО поддержки;
2) инсталляция драйвера модуля.

1.1. Установка файлов ПО поддержки

1.1.1. Windows

ПО поддержки модуля поставляется в виде дистрибутивного архивного файла `<Board_name>.rar`. Также имя файла может быть `<Board_name>_v_x_y.rar`, где x и y – major и minor номера версии соответственно.

ПО предназначено для работы в 64-разр. Windows 7 / 10.

Содержание архивного файла показано в табл. 1.1.

Дистрибутив ПО поддержки включает:

1) драйвер модуля под Windows 7, 10 x64;
2) библиотеку загрузки и обмена;
3) загрузчик программ (на исполнение);
4) набор примеров программ.

Для установки ПО поддержки модуля необходимо разархивировать дистрибутивный архивный файл в рабочий каталог.

В результате, в структуре установленного ПО поддержки должны находиться следующие файлы и каталоги:

Таблица 1.1. Содержимое дистрибутивного архивного файла ПО поддержки для Windows

Библиотека загрузки и обмена	
`bin\<Board_name>load.dll`	Host-часть библиотеки
`include\<Board_name>load.h`	Интерфейс host-части библиотеки
`lib\<Board_name>load.lib`	Библиотека связи к host-части библиотеки
`lib\libnm6408load_nmc.a`	NM-часть библиотеки
`include\nm6408load_nmc.h`	Интерфейс NM-части библиотеки
`lib\libnm6408load_arm.a`	ARM-части библиотеки

include\nm6408load_arm.h	Интерфейс ARM-части библиотеки
Примеры	
examples\	Исходные тексты примеров прикладных программ
Прочее	
bin\<Board_name>run.exe	Загрузчик программ (на исполнение)
drv\	Драйвер модуля

Для работы библиотеки загрузки и обмена необходимо, чтобы переменная окружения PATH содержала путь к файлу <Board_name>load.dll (каталог bin). Если данный путь в переменной окружения PATH отсутствует его нужно добавить:

```
PATH=%PATH%;<Путь к каталогу установки>\bin
```

1.1.2. Linux

ПО поддержки модуля поставляется в виде дистрибутивного архивного файла <Board_name>.tar.gz. Также имя файла может быть <Board_name>_v_x_y.tar.gz, где x и y – major и minor номера версии соответственно. Содержание архивного файла показано в табл. 1.2.

Дистрибутив ПО поддержки включает:

1) драйвер модуля под CentOS 7;
2) библиотеку загрузки и обмена;
3) загрузчик программ (на исполнение);
4) набор примеров программ;
5) код начальной инициализации модуля.

Для установки ПО поддержки модуля необходимо разархивировать дистрибутивный архивный файл в рабочий каталог:

```
> tar xpzf <Board_name>.tar.gz –C <Путь к рабочему каталогу>
```

В результате, в структуре установленного ПО поддержки должны находиться следующие файлы и каталоги:

Таблица 1.2. Содержимое дистрибутивного архивного файла ПО поддержки для Linux

Библиотека загрузки и обмена	
libload/	Общие исходные коды host-части библиотеки
libload/<Board_name>/	Исходные коды host-части библиотеки, специфичные для модуля
libload/arm/	Исходные коды ARM-части библиотеки
libload/nmc/	Исходные коды NM-части библиотеки

Драйвер модуля	
driver/	Общие исходные коды драйвера устройства
driver/<Board_name>/	Исходные коды драйвера, специфичные для модуля
Код начальной инициализации модуля	
firmware/	Исходные коды программы начальной инициализации модуля
Примеры	
example/	Исходные тексты примеров прикладных программ
Прочее	
Makefile-<Board_name>	Сценарий сборки библиотеки загрузки и обмена, драйвера модуля, программы начальной инициализации и загрузчика программ
install.sh	Скрипт для инсталляции драйвера модуля и host-части библиотеки загрузки и обмена
uninstall.sh	Скрипт для деинсталляции драйвера модуля и host-части библиотеки загрузки и обмена
run/	Исходные коды загрузчика программ (на исполнение)
libload/libelf/	Исходные коды библиотеки для разбора elf-файлов (необходимы для сборки host-части библиотеки загрузки и обмена, а также загрузчика программ)

После разархивации дистрибутивного архивного файла необходимо произвести сборку библиотеки загрузки и обмена, драйвера модуля, а также инсталлировать host-часть библиотеки и драйвер. Для сборки из рабочего каталога выполнить:

```
> make –f Makefile-<Board_name> ARM_CROSS_COMPILE=${ARM_PREFIX}
NMC_CROSS_COMPILE=${NMC_PREFIX}
```

В результате в каталоге driver/<Board_name>/ должен появиться файл драйвера ядра Linux – <Board_name>.ko, а в каталоге libload/<Board_name>/ исполняемый файл host-части библиотеки загрузки и обмена – lib<Board_name>load.so.X.X. Помимо этого создаётся каталог board-<Board_name>/, содержащий:

- arm_template.x – шаблон для получения скрипта компоновщика ARM;
- nmc_template.x – шаблон для получения скрипта компоновщика NMC;
- lib/libnm6408load_arm.a – ARM-часть библиотеки;

- lib/libnm6408load_nmc.a – NM-часть библиотеки;
- lib/lib<Board_name>load.so.x.x – host-часть библиотеки;
- include/nm6408load_arm.h – интерфейс ARM-части библиотеки;
- include/nm6408load_nmc.h – интерфейс NM-части библиотеки;
- include/<Board_name>load.h – интерфейс host-части библиотеки;
- bin/<Board_name>firmware.bin – прошивка SPI-Flash памяти платы;
- bin/<Board_name>init.bin – код инициализации при загрузке по EDCL;
- bin/<Board_name>run – загрузчик программ на исполнение.

Для инсталляции host-части библиотеки и драйвера из рабочего каталога с правами суперпользователя выполнить:

```
> ./install.sh <Board_name>
```

В результате успешной инсталляции драйвера код возврата (echo $?) следующей команды равен нулю:

```
> lsmod | grep -i <Board_name>
```

В результате успешной инсталляции host-части библиотеки код возврата (echo $?) следующей команды равен нулю:

```
> ldconfig -p | grep -i <Board_name>load
```

Каждый каталог дистрибутивного архива содержит файл ReadMe.txt, в котором содержится дополнительная информация по сборке, установке и использованию компонент ПО поддержки.

1.2. Инсталляция драйвера модуля

1.2.1. Windows

После первого подключения устройства в диспетчере устройств появится «неизвестное устройство» (отметим, что возможно вместо «неизвестного устройства» может появиться какое-то другое «проблемное» устройство). Необходимо в качестве места для поиска драйвера указать путь к директории drv (см. ниже).

Установка драйвера:

1. В диспетчере устройств найти «Неизвестное устройство» и дважды щёлкнуть по строке.
2. Появится окно свойств «Неизвестное устройство». Щёлкнуть кнопку «Обновить драйвер».
3. В окне «Обновление драйверов» выбрать «Выполнить поиск драйверов на этом компьютере».
4. В открывшемся окне указываем полный путь к папке - drv. Например: C:\mc12705\drv.
5. Щёлкнуть кнопку «Далее».
6. На все возможные вопросы установщика драйвера - продолжать установку.

В результате успешной установки в списке периферийных устройств ("Control Panel/System/Hardware/Device Manager") появится новое устройство с соответствующим названием.

1.2.2. Linux

После подключения устройства и установки драйвера, описанной в пункте 1.1.2, в системе

должен появиться символьный файл устройства /dev/<Board_name>_xx, где xx может принимать значения от 0 до 31 включительно. Если устройство не появилось, то нужно выполнить повторную энумерацию шины PCIe, в результате запустится проба драйвера и инициализация аппаратуры. Для этого сделать каталог driver/<Board_name>/ текущим и выполнить команду (с правами суперпользователя):

```
> ./upload_<Board_name>.sh reload ${PCIE_DEVICE_NUMBER} ${PCIE_BRIDGE_NUMBER}
```

Номер устройства и моста на шине PCIe (в формате "domain:bus:slot.func") можно узнать, выполнив команды:

```
> lspci –D –d0x17CD:
> lspci -tv
```

2. БИБЛИОТЕКА ЗАГРУЗКИ И ОБМЕНА

Библиотека загрузки и обмена содержит минимально необходимый набор функций для организации работы с модулем.

2.1. Организация библиотеки

Библиотека загрузки и обмена состоит из следующих частей:

1) набор функций для целевого NMC процессорного ядра (*NM-часть* или *NM-сторона*);
2) набор функций для целевого ARM процессорного ядра (*ARM-часть* или *ARM-сторона*);
3) набор функций для host-процессора (*host-часть* или *host-сторона*).

NM-часть состоит из файла `nm6408load_nmc.h`, задающего интерфейс библиотеки со стороны целевого NMC процессорного ядра, и файла реализации `libnm6408load_nmc.a`.

ARM-часть состоит из файла `nm6408load_arm.h`, задающего интерфейс библиотеки со стороны целевого ARM процессорного ядра, и файла реализации `libnm6408load_arm.a`.

Для ОС Windows host-часть состоит из интерфейсного файла `<Board_name>load.h` и файла реализации `<Board_name>load.dll` (к данной DLL поставляется библиотека связи `<Board_name>load.lib` для компилятора Visual C++ версии 7.0 и более).

Ниже приводится состав библиотеки загрузки и обмена для ОС Windows с описанием местоположения файлов, в структуре установленного ПО поддержки:

Таблица 2.1. Файлы библиотеки для ОС Windows

Файл	**Описание**
`bin\<Board_name>load.dll`	Динамическая библиотека с кодом функций библиотеки загрузки и обмена, исполняемых на host-машине.
`include\<Board_name>load.h`	Заголовочный файл для функций библиотеки, исполняемых на host-машине.
`lib\<Board_name>load.lib`	Библиотека связи к `bin\<Board_name>load.dll` (только для Visual C++ v7.0 и выше).
`lib\libnm6408load_nmc.a`	Библиотека времени исполнения с кодом функций библиотеки загрузки и обмена, исполняемых на NMC процессорных ядрах СнК (NeuroMatrix® NMC4).
`include\nm6408load_nmc.h`	Заголовочный файл для функций библиотеки, исполняемых на NMC процессорных ядрах СнК (NeuroMatrix® NMC4).
`lib\libnm6408load_arm.a`	Библиотека времени исполнения с кодом функций библиотеки загрузки и обмена, исполняемых на ARM процессорных ядрах СнК (ARM® Cortex-A5).
`include\nm6408load_arm.h`	Заголовочный файл для функций библиотеки, исполняемых на ARM процессорных ядрах СнК (ARM® Cortex-A5).

Для ОС Linux состав ARM-части и NM-части библиотеки полностью совпадает с составом ARM-части и NM-части библиотеки для ОС Windows, за исключением того, что библиотеки распространяются в виде исходных кодов. Сборка библиотек описана в разделе 1.1.2.

Host-часть библиотеки, для ОС Linux, распространяется в виде исходных кодов. Сборка и установка исполняемых файлов описана в разделе 1.1.2.

2.2. Концепция построения интерфейса

Host-интерфейс библиотеки загрузки и обмена строился, исходя из следующих предпосылок и требований:

1) наиболее распространённым прикладным сценарием взаимодействия PC-машины и целевых устройств конструктива PCIe с разделяемой памятью является сценарий, в котором инициатором взаимодействия является host-процесс: он инициализирует устройство, загружает прикладную задачу, принимает и отображает результаты её работы;
2) базовый интерфейс библиотеки должен быть инвариантен относительно разных типов устройств;
3) интерфейс библиотеки должен обеспечивать инвариантность кода прикладных host-программ относительно разных процессоров многопроцессорных устройств (или, как минимум, лёгкую перенацеливаемость);
4) интерфейс библиотеки должен быть минимальным.

Инвариантность относительно типов устройств.

Host-интерфейс библиотеки достаточно универсален, чтобы покрыть большой класс устройств с общей памятью и произвольным количеством процессоров:

- сокрытие особенностей доступа к конкретному устройству и особенностей взаимодействия с процессорными ядрами устройства достигается за счёт использования дескрипторов доступа к экземплярам устройств (PL_Board) и процессорных ядер (PL_Access);
- в функциях выдачи дескрипторов доступа (PL_GetBoardDesc и PL_GetAccess соответственно) экземпляры устройств и процессорных ядер идентифицируются порядковыми номерами в 32-разрядном слове.

Имена типов дескрипторов фиксированы и от реализаций библиотеки для разных устройств не зависят. Благодаря этому host-часть прикладной программы возможно перенацеливать на иное устройство без изменения существенного кода, простой заменой заголовочного файла. (Используя динамическую загрузку host-части библиотеки, возможно параметризовать выбор реализации библиотеки во время исполнения, в таком случае для перенацеливания перекомпиляция прикладной программы не требуется вовсе).

2.3. NM-сторона библиотеки

Объявления NM-функций библиотеки для модуля содержатся в заголовочном файле `nm6408load_nm.h`. В состав этой части библиотеки входят функции, используемые для синхронизации программ, работающих на процессоре модуля и на host-машине.

Все функции библиотеки имеют "C" связывание и могут вызываться как из ассемблерных программ, так и из программ, написанных на Си/Си++.

ncl_hostSync

ncl_hostSync — барьерная синхронизация с host-процессом, скаляром

Синтаксис

`int` **`ncl_hostSync`** (int *value*);

Описание

Функция `ncl_hostSync` осуществляет барьерную синхронизацию с процессом, выполняющимся на host-машине. Является парной процедурой к функции `PL_Sync` из host-стороны библиотеки.

Host- и NM- процессы дожидаются друг друга у "барьера" `PL_Sync` | `ncl_hostSync` и возобновляют исполнение только после того, как обменяются сообщениями. Сообщения состоят из одного 32-разрядного слова (аргументы *`value`* каждой из участвующих в синхронизации функций).

Для обмена используются ячейки области разделяемой памяти, занятой под нужды библиотеки.

Смысл сообщений определяется пользователем.

Время ожидания функцией `ncl_hostSync` ответа от `PL_Sync` - бесконечное, и не может быть изменено.

Аргумент *`value`* посылается host-процессу.

В качестве возвращаемого значения выдаётся значение аргумента *`value`*, с которым была вызвана соответствующая функция `PL_Sync`.

ncl_hostSyncArray

ncl_hostSyncArray — барьерная синхронизация с host-процессом, массивом

Синтаксис

```
int ncl_hostSyncArray (int value,
                       void *outAddress,
                       size_t outLen,
                       void **inAddress,
                       size_t *inLen);
```

Описание

Функция `ncl_hostSyncArray` осуществляет барьерную синхронизацию с процессом, выполняющимся на host-машине. Является парной процедурой к функции `PL_SyncArray` из host-стороны библиотеки.

Host- и NM- процессы дожидаются друг друга у "барьера" `PL_SyncArray` | `ncl_hostSyncArray` и возобновляют исполнение только после того, как обменяются сообщениями. Сообщения состоят из трёх 32-разрядных слов (аргументы *`value`*, *`outAddress`*, *`outLen`* каждой из участвующих в синхронизации функций).

Для обмена используются ячейки области разделяемой памяти, занятой под нужды библиотеки.

Смысл сообщений определяется пользователем, но предполагаемое применение функций SyncArray - взаимный обмен информацией о расположении в разделяемой памяти подготовленных к обмену блоков (массивов) данных. При этом назначение *`value`* остаётся произвольным (какой-либо статус, идентификатор сообщения или момента времени), а остальные параметры рекомендуется использовать следующим образом: в *`outAddress`* и *`outLen`* передавать адрес и размер массива, предназначенный для передачи host-процессу (*передача*), в переменные по указателям *`inAddress`* и *`inLen`* принимать адрес и размер массива, предназначенный для передачи NM-процессу (*приём*).

> **Замечание**
>
> `ncl_hostSyncArray` и `PL_SyncArray` обмениваются только координатами массивов данных, не самими массивами.

Время ожидания функцией `ncl_hostSyncArray` ответа от `PL_SyncArray` - бесконечное, и не может быть изменено.

Каждый из указателей *`inAddress`* и *`inLen`* (или оба) может быть равен NULL.

Аргумент *`value`* посылается host-процессу.

В качестве возвращаемого значения выдаётся значение аргумента *`value`*, с которым была вызвана соответствующая функция `PL_SyncArray`.

2.4. ARM-сторона библиотеки

Объявления ARM-функций библиотеки для модуля содержатся в заголовочном файле `nm6408load_arm.h`. В состав этой части библиотеки входят функции, используемые для синхронизации программ, работающих на процессоре модуля и на host-машине. Помимо сервисных функций, ARM-часть библиотеки предоставляет функции-заглушки для взаимодействия со стандартной библиотекой языка C (newlib), код инициализации ядра ARM и настройки стека.

Все функции библиотеки имеют "C" связывание и могут вызываться как из ассемблерных программ, так и из программ, написанных на Си/Си++.

acl_hostSync

acl_hostSync — барьерная синхронизация с host-процессом, скаляром

Синтаксис

```
int acl_hostSync (int value);
```

Описание

Функция acl_hostSync осуществляет барьерную синхронизацию с процессом, выполняющимся на host-машине. Является парной процедурой к функции PL_Sync из host-стороны библиотеки. Эта процедура аналогична ncl_hostSync функции из NM-части.

acl_hostSyncArray

acl_hostSyncArray — барьерная синхронизация с host-процессом, массивом

Синтаксис

```
int acl_hostSyncArray (int value,
                       void *outAddress,
                       size_t outLen,
                       void **inAddress,
                       size_t *inLen);
```

Описание

Функция `acl_hostSyncArray` осуществляет барьерную синхронизацию с процессом, выполняющимся на host-машине. Является парной процедурой к функции `PL_SyncArray` из host-стороны библиотеки. Эта процедура аналогична ncl_hostSyncArray функции из NM-части.

2.5. Host-сторона библиотеки

Объявления host-функций библиотеки для модуля содержатся в файле `<Board_name>load.h`.

Код функций этой части библиотеки оформлен в виде DLL-модуля для ОС Windows и в виде *.so.X.X библиотеки для ОС Linux.

2.5.1. Используемые типы данных

Ниже приводятся типы данных, использующиеся этой частью библиотеки.

PL_Word

PL_Word — тип элемента памяти СнК 1879ВМ8Я

Синтаксис

```
typedef unsigned long PL_Word;
```

Описание

32-разрядное слово без знака. Элемент памяти СнК 1879ВМ8Я.

PL_Addr

PL_Addr — тип адреса в памяти СнК 1879ВМ8Я

Синтаксис

```
typedef unsigned long PL_Addr;
```

Описание

32-разрядное слово без знака. Адрес в памяти СнК 1879ВМ8Я.

PL_Board

PL_Board — тип дескриптора экземпляра модуля

Синтаксис

```
struct PL_Board;
```

Описание

Внутренний нераскрываемый тип библиотеки.

Дескриптор модуля выдаётся в результате выбора некоторого конкретного экземпляра устройства и используется для получения дальнейшего доступа к ресурсам этого экземпляра (т. е. процессорным ядрам).

PL_Access

PL_Access — тип дескриптора экземпляра процессорного ядра

Синтаксис

```
struct PL_Access;
```

Описание

Внутренний нераскрываемый тип библиотеки.

Дескриптор процессорного ядра выдаётся в результате выбора некоторого конкретного экземпляра ядра и используется для получения дальнейшего доступа к ресурсам этого ядра.

RetValue

RetValue — коды статуса библиотечных операций

Синтаксис

```
enum RetValue
{
        PL_OK              = 0,
        PL_ERROR           = 1,
        PL_TIMEOUT         = 2,
        PL_FILE            = 3,
        PL_BADADDRESS      = 4,
        PL_AGAIN           = 5,
        PL_NOT_IMPLEMENTED = -1
};
```

Описание

Стандартные коды статуса, возвращаемые функциями библиотеки:

PL_OK	Функция выполнена нормально
PL_ERROR	Ошибка при выполнении функции
PL_TIMEOUT	Время ожидания вышло
PL_FILE	Не найден загружаемый файл
PL_BADADDRESS	Некорректный диапазон адресов
PL_AGAIN	Объект *board* или *access* занят другой функцией БЗО
PL_NOT_IMPLEMENTED	Функция не реализована

2.5.2. Функции общего назначения

PL_GetVersion

PL_GetVersion — определение номера версии библиотеки

Синтаксис

```
DECLSPEC int PL_GetVersion (unsigned int *version_major,
                            unsigned int *version_minor);
```

Описание

В переменные по указателям *version_major* и *version_minor* возвращаются номера версии библиотеки.

version_major и *version_minor* могут иметь значения равные NULL. В таком случае значения версии не возвращаются.

Функция всегда успешна, возвращаемое значение - PL_OK.

PL_SetChannelEDCL

PL_SetChannelEDCL — установка EDCL в качестве канала связи

Синтаксис

```
DECLSPEC int PL_SetChannelEDCL (const unsigned char host_mac_addr[6],
                                const unsigned char board_mac_addr[6]);
```

Описание

Функция устанавливает EDCL в качестве канала связи между хостом и платой. По умолчанию в качестве канала связи используется PCIe.

В переменной *host_mac_addr* задаётся MAC-адрес сетевого интерфейса хоста. В переменной *board_mac_addr* задаётся MAC-адрес платы. Для поддерживаемой библиотекой платы MAC-адрес – EC-17-66-64-08-0X, где X – определяется положением переключателей на плате (см. в руководстве по эксплуатации используемого модуля описание управления режимами загрузки). После установки канала связи по EDCL обратного переключения на PCIe в текущей версии библиотеки не предусмотрено.

Функция всегда успешна, возвращаемое значение - PL_OK.

PL_GetBoardCount

PL_GetBoardCount — определение количества доступных экземпляров модуля

Синтаксис

```
DECLSPEC int PL_GetBoardCount (unsigned int *count);
```

Описание

В переменную по указателю *count* возвращается количество готовых к работе модулей, установленных на host-машине.

Функция реализована только для PCIe.

При успешном завершении возвращает PL_OK или PL_ERROR в случае ошибки.

PL_GetBoardDesc

PL_GetBoardDesc — получение дескриптора доступа к экземпляру модуля

Синтаксис

```
DECLSPEC int PL_GetBoardDesc (unsigned int index,
                              PL_Board **board);
```

Описание

Функция выдаёт (открывает) дескриптор экземпляра модуля в переменной по указателю *board*. Для PCIe порядковый номер определятся параметром *index*; для EDCL этот параметр игнорируется.

board может иметь значение, равное NULL. В таком случае дескриптор не возвращается.

Нумерация модулей начинается с 0 и производится согласно порядку нумерации занимаемых модулями PCIe-слотов. Т. е. номер 0 присваивается модулю, находящемуся в слоте с наименьшим номером из всех слотов, занимаемых такими модулями; номер 1 присваивается модулю, находящемуся в слоте с наименьшим номером из всех слотов, занимаемых такими модулями, за исключением слота, модулю в котором присвоен номер 0 и т. д. Таким образом, номер каждого модуля остаётся неизменным, пока неизменно распределение модулей *этого типа* по слотам; при добавлении, удалении или перестановке в иной слот, нумерация может меняться.

Возвращает PL_OK, или PL_ERROR, если драйвер этого типа модуля недоступен, в случае некорректности аргумента *index* или в случае другой ошибки.

PL_CloseBoardDesc

PL_CloseBoardDesc — завершение работы с модулем

Синтаксис

```
DECLSPEC int PL_CloseBoardDesc (PL_Board *board);
```

Описание

Закрывает дескриптор экземпляра модуля.

Закрытие дескриптора модуля не является критически важным действием, однако весьма желательно. Отсутствие вызова данной функции приводит к потере памяти, что может (теоретически) вызывать проблемы в работе программ, выполняющих многократные открытия дескрипторов в течение времени своей работы. Ресурсы, ассоциированные с дескрипторами модулей, являются ресурсами процесса, так что они освобождаются автоматически по его завершении.

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_AGAIN	при занятости объекта *board* (создан объект *access*),
PL_ERROR	в случае некорректности аргумента *board* или другой ошибки.

2.5.3. Функции для работы с выбранным модулем

PL_GetSerialNumber

PL_GetSerialNumber — определение серийного номера платы

Синтаксис

```
DECLSPEC int PL_GetSerialNumber (PL_Board *board,
                                 unsigned long *serial_number);
```

Описание

Возвращает серийный номер платы, доступной по дескриптору *board*, в переменную по указателю *serial_number*.

Функция реализована только для PCIe.

При успешном завершении возвращает PL_OK, или PL_ERROR, в случае некорректности аргумента *board* или другой ошибки.

PL_GetFirmwareVersion

PL_GetFirmwareVersion — получение версии прошивки платы

Синтаксис

```
DECLSPEC int PL_GetFirmwareVersion (PL_Board *board,
                                    unsigned int *version_major,
                                    unsigned int *version_minor);
```

Описание

Возвращает версию прошивки платы, доступной по дескриптору *board*, в переменные по указателям *version_major* и *version_minor*.

Функция реализована только для PCIe.

При успешном завершении возвращает PL_OK, или PL_ERROR, в случае некорректности аргумента *board* или другой ошибки.

PL_ResetBoard

PL_ResetBoard — перезагрузка экземпляра модуля

Синтаксис

```
DECLSPEC int PL_ResetBoard (PL_Board *board);
```

Описание

Выполняет перезагрузку всех процессорных ядер СнК модуля.

При успешном завершении возвращает PL_OK, или PL_ERROR, в случае некорректности аргумента *board*.

PL_LoadInitCode

PL_LoadInitCode — загрузка кода начальной инициализации

Синтаксис

```
DECLSPEC int PL_LoadInitCode (PL_Board *board);
```

Описание

Загружает в память платы, доступной по дескриптору `board`, и запускает на исполнение код начальной инициализации.

Загрузка и выполнение пользовательских программ для EDCL возможны только после успешной отработки данной функции. Для PCIe вызов данной функции не обязателен.

Возвращает PL_OK, или PL_ERROR, в случае ошибки.

PL_GetAccess

PL_GetAccess — получение дескриптора доступа к процессорному ядру СнК модуля

Синтаксис

```
DECLSPEC int PL_GetAccess (PL_Board *board,
                           PL_CoreNo *coreNo,
                           PL_Access **access);
```

Описание

Выдаёт (открывает) дескриптор процессорного ядра системы на кристалле, определяемого структурой *coreNo* модуля *board,* в переменной по указателю *access*.

Структура *coreNo* содержит два поля: *cluster_id* и *nm_id*. Поле *cluster_id* определяет номер кластера, содержащего определяемое ядро; для центрального ARM ядра это поле должно содержать значение (–1). Поле *nm_id* определяет номер NM в кластере; для ARM ядра это поле должно содержать значение (–1).

Допустимый диапазон значений *cluster_id*: (–1) – 3. Допустимый диапазон значений *nm_id*: (–1) – 3.

При успешном завершении возвращает PL_OK, или PL_ERROR, в случае некорректности аргументов или другой ошибки.

PL_CloseAccess

PL_CloseAccess — завершение работы с процессорным ядром СнК модуля

Синтаксис

```
DECLSPEC int PL_CloseAccess (PL_Access *access);
```

Описание

Закрывает дескриптор процессорного ядра.

Закрытие дескриптора процессорного ядра не является критически важным действием, однако весьма желательно. Отсутствие вызова данной функции приводит к потере памяти, что может (теоретически) вызывать проблемы в работе программ, выполняющих многократные открытия дескрипторов в течение времени своей работы. Ресурсы, ассоциированные с дескрипторами процессоров, являются ресурсами процесса, так что они освобождаются автоматически по его завершении.

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_AGAIN	при занятости объекта *access* другой функцией БЗО (например *PL_Sync*),
PL_ERROR	в случае некорректности аргумента *access* или другой ошибки.

2.5.4. Функции для работы с выбранным процессором

PL_LoadProgramFile

PL_LoadProgramFile — загрузка и исполнение пользовательской программы

Синтаксис

```
DECLSPEC int PL_LoadProgramFile (PL_Access *access,
                                 const char *filename);
```

Описание

Загружает программу пользователя из исполняемого файла *filename* на процессорное ядро, доступное по дескриптору *access*, и запускает её на исполнение.

filename должно быть полным именем (включая абсолютный или относительный путь) исполняемого ELF-файла программы в файловой системе host-машины.

Из исполняемого файла в память системы на кристалле загружаются программные сегменты, помеченные как загружаемые (то есть, код и данные загружаются, а, к примеру, отладочная информация - нет). Сегменты размещаются по адресам, заданным файлом конфигурации на этапе компоновки программы.

В памяти модуля и системы на кристалле существуют особые области, перезапись которых может привести к нарушению функциональности средств библиотеки загрузки и обмена или зависанию host-машины (служебные области библиотеки в разделяемой памяти и т. п.). При загрузке такие служебные области охраняются библиотекой. Непосредственно перед загрузкой проверяются адресные диапазоны сегментов программы и, в случае нарушения дозволенной конфигурации, пользовательская программа не грузится.

Если загрузка прошла удачно - программа стартует.

После завершения работы программы возвращаемое значение можно получить с помощью функции PL_GetResult. О завершении работы свидетельствует установка первого бита PROGRAM_FINISHED в слове состояния системного загрузчика (можно получить с помощью функции PL_GetStatus).

Загрузка программы возможна, если состояние системного загрузчика равно PROGRAM_NO (см. функцию PL_GetStatus).

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_FILE	если указанный файл не найден или не является исполняемым ELF-файлом,
PL_BADADDRESS	если конфигурация программы (размещение сегментов в памяти) для данного модуля является некорректной,
PL_ERROR	если аргументы функции некорректны или загрузчик на процессоре не находится в состоянии готовности, а также в случае любой другой ошибки.

PL_LoadProgram

PL_LoadProgram — загрузка и исполнение пользовательской программы из буфера

Синтаксис

```
DECLSPEC int PL_LoadProgram ( PL_Access *access,
                              const void *addrProgram,
                              unsigned int sizeProgram);
```

Описание

Загружает программу пользователя из буфера в памяти host-машины на процессорное ядро, доступное по дескриптору *access*, и запускает её на исполнение.

addrProgram адрес буфера, который содержит образ исполняемого файла в формате ELF, *sizeProgram* – размер буфера в байтах.

В память системы на кристалле загружаются программные сегменты, помеченные как загружаемые (то есть, код и данные загружаются, а, к примеру, отладочная информация - нет). Сегменты размещаются по адресам, заданным файлом конфигурации на этапе компоновки программы. Если загрузка прошла удачно - программа стартует (для справки смотри функцию PL_LoadProgramFile).

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_FILE	если указанный буфер не содержит отображение исполняемого файла в формате ELF,
PL_BADADDRESS	если конфигурация программы (размещение сегментов в памяти) для данного модуля является некорректной,
PL_ERROR	если аргументы функции некорректны или загрузчик на процессоре не находится в состоянии готовности, а также в случае любой другой ошибки.

PL_ReadMemBlock

PL_ReadMemBlock — чтение блока из разделяемой памяти модуля

Синтаксис

```
DECLSPEC int PL_ReadMemBlock (PL_Access *access,
                              PL_Word *block,
                              PL_Addr address,
                              PL_Word len);
```

Описание

Копирует блок данных размером *len* слов процессорного ядра из памяти модуля (по адресу *address*) в память host-процесса по адресу *block*.

Значение *address* должно быть адресом (не смещением), принадлежащим памяти модуля, с точки зрения процессорного ядра, доступного по дескриптору *access*.

Адресация для NM должна быть в 32-разр. словах. Адресация для ARM должна быть в байтах. Размер передаваемых данных должен быть в 32-разр. словах.

Допустимые диапазоны адресов для NM (в 32-разр. словах):

- 0x00000000 – 0x0001FFFF
- 0x00040000 – 0x000BFFFF
- 0x000C0000 – 0x000CFFFF
- 0x000D0000 – 0x07FFFFFF
- Соответствующие адреса глобального адресного пространства (≥ 0x18000000).

Допустимые диапазоны адресов для ARM кластера (в байтах):

- 0x00000000 – 0x0003FFFF
- 0x00100000 – 0x002FFFFF
- 0x00300000 – 0x1FFFFFFF
- Соответствующие адреса глобального адресного пространства (≥ 0x60000000).

Допустимые диапазоны адресов для центрального ARM (в байтах):

- 0x00000000 – 0x0007FFFF
- 0x00100000 – 0x1FFFFFFF
- Соответствующие адреса глобального адресного пространства (≥ 0x60000000).

Наименьший допустимый размер блока - 1 слово (32-разрядное слово), наибольший определяется размером памяти.

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_BADADDRESS	если переданный диапазон адресов не является допустимым,
PL_ERROR	если аргументы функции некорректны, а также в случае любой другой ошибки.

PL_WriteMemBlock

PL_WriteMemBlock — запись блока данных в разделяемую память модуля

Синтаксис

```
DECLSPEC int PL_WriteMemBlock (PL_Access *access,
                               PL_Word *block,
                               PL_Addr address,
                               PL_Word len);
```

Описание

Копирует блок данных размером *len* слов процессорного ядра из памяти host-процесса (по адресу *block*) в разделяемую память модуля по адресу *address*.

Значение *address* должно быть адресом (не смещением), принадлежащим памяти модуля, с точки зрения процессорного ядра, доступного по дескриптору *access*.

Адресация для NM должна быть в 32-разр. словах. Адресация для ARM должна быть в байтах. Размер передаваемых данных должен быть в 32-разр. словах.

Допустимые диапазоны адресов для NM (в 32-разр. словах):

- 0x00000000 – 0x0001FFFF
- 0x00040000 – 0x000BFFFF
- 0x000C0000 – 0x000CFFFF
- 0x000D0000 – 0x07FFFFFF
- Соответствующие адреса глобального адресного пространства (≥ 0x18000000).

Допустимые диапазоны адресов для ARM кластера (в байтах):

- 0x00000000 – 0x0003FFFF
- 0x00100000 – 0x002FFFFF
- 0x00300000 – 0x1FFFFFFF
- Соответствующие адреса глобального адресного пространства (≥ 0x60000000).

Допустимые диапазоны адресов для центрального ARM (в байтах):

- 0x00000000 – 0x0005FFFF
- 0x00100000 – 0x1FFFFFFF
- Соответствующие адреса глобального адресного пространства (≥ 0x60000000).

Наименьший допустимый размер блока - 1 слово (32-разрядное слово), наибольший определяется размером памяти.

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_BADADDRESS	если переданный диапазон адресов не является допустимым,
PL_ERROR	если аргументы функции некорректны, а также в случае любой другой ошибки.

PL_WriteRegister

PL_WriteRegister — запись значение в регистр процессорного ядра ARM

Синтаксис

```
DECLSPEC int PL_WriteRegister (PL_Access *access,
                               PL_Word value,
                               PL_Addr address);
```

Описание

Записывает значение *value* в регистр по адресу *address*. Значение *address* должно быть адресом (не смещением), принадлежащим области регистров процессорного ядра, доступного по дескриптору *access*.

Допустимый диапазон адресов для NM (в 32-разр. словах):

- нет доступных адресов

Допустимый диапазон адресов для ARM кластера (в байтах):

- 0x00080000 – 0x000FFFFC

Допустимый диапазон адресов для центрального ARM (в байтах):

- 0x00080000 – 0x000FFFFC

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_BADADDRESS если передаваемый адрес вне допустимого диапазона или является резервным,

PL_ERROR если аргументы функции некорректны, а также в случае любой другой ошибки.

PL_ReadRegister

PL_ReadRegister — прочитать значение из регистра процессорного ядра ARM

Синтаксис

```
DECLSPEC int PL_ReadRegister (PL_Access *access,
                              PL_Word *returnValue,
                              PL_Addr address);
```

Описание

Прочитывает значение регистра по адресу *address* в переменную по указателю *returnValue*. Значение *address* должно быть адресом (не смещением), принадлежащим области регистров процессорного ядра, доступного по дескриптору *access*.

Допустимый диапазон адресов для NM (в 32-разр. словах):

- нет доступных адресов

Допустимый диапазон адресов для ARM кластера (в байтах):

- 0x00080000 – 0x000FFFFC

Допустимый диапазон адресов для центрального ARM (в байтах):

- 0x00080000 – 0x000FFFFC

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_BADADDRESS — если передаваемый адрес вне допустимого диапазона или является резервным,

PL_ERROR — если аргументы функции некорректны, а также в случае любой другой ошибки.

PL_Sync

PL_Sync — барьерная синхронизация с ARM/NM-процессом, скаляром

Синтаксис

```
DECLSPEC int PL_Sync (PL_Access *access,
                      int value,
                      int *returnValue);
```

Описание

Функция PL_Sync осуществляет барьерную синхронизацию с процессом, выполняющимся на процессорном ядре модуля, доступном по дескриптору *access*. Является парной процедурой к функции ncl_hostSync из NM-стороны библиотеки и к функции acl_hostSync из ARM-стороны библиотеки.

Host- и NM- процессы дожидаются друг друга у "барьера" PL_Sync | ncl_hostSync и возобновляют исполнение только после того, как обменяются сообщениями. Сообщения состоят из одного 32-разрядного слова (аргументы *value* каждой из участвующих в синхронизации функций).

Для обмена используются ячейки области разделяемой памяти, занятой под нужды библиотеки.

Смысл сообщений определяется пользователем.

Время ожидания функцией PL_Sync ответа от ncl_hostSync - по-умолчанию бесконечное, однако может быть изменено функцией PL_SetTimeout.

Аргумент *value* посылается NM-процессу.

Аргумент, с которым была вызвана соответствующая ncl_hostSync возвращается в переменной по указателю *returnValue*.

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_TIMEOUT если истёк предписанный период ожидания,

PL_ERROR в случае другой ошибки.

Поведение функции ARM-стороны библиотеки аналогично поведению, описанному для NM-стороны библиотеки.

PL_SyncArray

PL_SyncArray — барьерная синхронизация с ARM/NM-процессом, массивом

Синтаксис

```
DECLSPEC int PL_SyncArray (PL_Access *access,
                           int value,
                           PL_Addr outAddress,
                           PL_Word outLen,
                           int *returnValue,
                           PL_Addr *inAddress,
                           PL_Word *inLen);
```

Описание

Функция `PL_SyncArray` осуществляет барьерную синхронизацию с процессом, выполняющимся на процессорном ядре модуля, доступном по дескриптору *`access`*. Является парной процедурой к функции `ncl_hostSyncArray` из NM-стороны библиотеки и к функции `acl_hostSyncArray` из ARM-стороны библиотеки.

Host- и NM- процессы дожидаются друг друга у "барьера" `PL_SyncArray` | `ncl_hostSyncArray` и возобновляют исполнение только после того, как обменяются сообщениями. Сообщения состоят из трёх 32-разрядных слов (аргументы *`value`*, *`outAddress`*, *`outLen`* каждой из участвующих в синхронизации функций). Для обмена используются ячейки области разделяемой памяти, занятой под нужды библиотеки.

Смысл сообщений определяется пользователем, но предполагаемое применение функций SyncArray - взаимный обмен информацией о расположении в разделяемой памяти подготовленных к обмену блоков (массивов) данных. При этом назначение *`value`* остаётся произвольным (какой-либо статус, идентификатор сообщения или момента времени), а остальные параметры рекомендуется использовать следующим образом: в *`outAddress`* и *`outLen`* передавать адрес и размер массива, предназначенный для передачи NM-процессу (*передача*), в переменные по указателям *`inAddress`* и *`inLen`* принимать адрес и размер массива, предназначенный для передачи host-процессу (*приём*).

> **Замечание**
>
> `ncl_hostSyncArray` и `PL_SyncArray` обмениваются только координатами массивов данных, не самими массивами.

Время ожидания функцией `PL_SyncArray` ответа от `ncl_hostSyncArray` - по-умолчанию бесконечное, однако может быть изменено функцией `PL_SetTimeout`.

Каждый из указателей *`inAddress`* и *`inLen`* (или оба) может быть равен NULL.

Аргумент *`value`* посылается NM-процессу.

Аргумент *`value`*, с которым была вызвана соответствующая `ncl_hostSyncArray` возвращается в переменной по указателю *`returnValue`*.

В случае успеха возвращает PL_OK.

Иначе, возвращает:

PL_TIMEOUT если истёк предписанный период ожидания,

PL_ERROR в случае другой ошибки.

Поведение функции ARM-стороны библиотеки аналогично поведению описанному выше.

PL_SetTimeout

PL_SetTimeout — установка времени ожидания функциям синхронизации

Синтаксис

```
DECLSPEC int PL_SetTimeout (int timeout);
```

Описание

Устанавливает максимальное время ожидания при барьерной синхронизации. Если при вызове функции синхронизации семейства `PL_Sync` синхронизация не произошла в течении времени ожидания, функция синхронизации возвращается с ошибкой PL_TIMEOUT.

По-умолчанию, функции синхронизации ожидают бесконечное время. Установленное `PL_SetTimeout` время ожидания останется таковым до тех пор, пока не будет изменено новым вызовом данной функции.

Время задается в миллисекундах.

Бесконечное время задаётся нулём.

Возвращает PL_OK.

PL_GetStatus

PL_GetStatus — определение текущего состояния системного загрузчика

Синтаксис

```
DECLSPEC int PL_GetStatus (PL_Access *access,
                           PL_Word *status);
```

Описание

Возвращает слово состояния системного загрузчика для процессорного ядра, доступного по дескриптору `access`, в переменную по указателю `status`.

Значение *status* отражает состояние программы, которая была загружена функцией `PL_LoadProgramFile/PL_LoadProgram` на процессор, соответствующий аргументу *access*:

- `PROGRAM_NO` – программа не загружена на исполнение;
- `PROGRAM_PROGRESS` – программа исполняется;
- `PROGRAM_FINISHED` – программа закончила исполнение.

Замечание

Функция `PL_ResetBoard` устанавливает статус загрузчика в значение `PROGRAM_NO`.

При успешном завершении возвращает PL_OK, или PL_ERROR, в случае другой ошибки.

PL_GetResult

PL_GetResult — получение возвращаемого значения пользовательской программы

Синтаксис

```
DECLSPEC int PL_GetResult (PL_Access *access,
                           PL_Word *result);
```

Описание

Копирует возвращаемое значение пользовательской программы, запущенной на процессорном ядре, доступном по дескриптору *access*, в переменную по указателю *result*.

Если пользовательская программа на момент вызова функции ещё не завершилась (или даже не запускалась), то возвращается ошибка PL_ERROR.

При успешном завершении возвращает PL_OK, или PL_ERROR, в случае другой ошибки.

3. ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

В составе ПО поддержки модуля распространяются следующие примеры пользовательских программ:

- `simple` – простейшая программа для исполнения на модуле, вычисляющая факториал 10;
- `use_sync` – более сложная программа, состоящая уже из двух частей: исполнительной NM / ARM - программы и загружающей её на целевой процессор host - программы. Взаимодействие между частями осуществляется средствами библиотеки загрузки и обмена;
- `printf` – демонстрация использования функции `printf` на целевом процессоре (NM или ARM);
- `print_main_arguments` – демонстрация использования (вывода через функцию `printf`) аргументов функции `main`.

Исходные файлы этих примеров после установки ПО поддержки модуля располагаются в соответственно поименованных подкаталогах каталога `examples\nm` (для NM) или `examples\arm` (для ARM).

Для сборки примера следует выполнить `make.bat`. Для запуска примера следует выполнить `run.bat`.