Современные устройства сбора данных

L-502

Низкоуровневое описание

Ревизия 1.00

Февраль 2013

Автор руководства:
Борисов Алексей

ООО "Л КАРД"
117105, г. Москва, Варшавское ш., д. 5, корп. 4, стр. 2

тел.: (495) 785-95-25
факс: (495) 785-95-14

Адреса в Интернет:
www.lcard.ru

E-Mail:
Отдел продаж: sale@lcard.ru
Техническая поддержка: support@lcard.ru
Отдел кадров: job@lcard.ru
Общие вопросы: lcard@lcard.ru
Отдел производства: pro@lcard.ru

Представители в регионах:
Украина: ХОЛИТ Дэйта Системс,www.holit.com.ua, (044) 241-6754
Санкт-Петербург: Autex Spb Ltd., www.autex.spb.ru, (812) 567-7202
Новосибирск: Сектор-Т, www.sector-t.ru, (383-2) 396-592
Казань: ООО 'Шатл', shuttle@kai.ru, (8432) 38-1600
Екатеринбург: Авеон, aveon@aveon.ru, +7(343) 381-75-75
Пенза: НПП Технолинк, http://www.tl.ru/ru/departments/industry (8412) 49-10-59

Оглавление

Глава 1

О чем этот документ

Данный документ представляет собой низкоуровневое описание модуля L502 и предназначен в первую очередь для следующих двух групп пользователей:

- пользователи, пишущие свою прошивку для сигнального процессора BlackFin, чтобы расширить функциональность модуля.
- пользователи, пишущие свое программное обеспечения без использования библиотеки и драйвера, предоставляемых фирмой Л Кард’’ (например, для написания программ для операционных систем, для которых нет официальной поддержки отЛ Кар”).

Для программистов, использующих штатные возможности модуля L502, предоставляемые “L Card” драйвер и библиотека l502api скрывают низкоуровневые особенности модуля L502. Этим пользователям в первую очередь следует прочитать "Руководство программиста", в то время как данный документ может использоваться как дополнительная информация для лучшего понимания внутреннего устройства модуля.

Также в настоящем документе не рассматриваются какие-либо вопросы, касающиеся подключения сигналов и характеристик модуля. Эти вопросы рассматриваются в "Руководстве пользователя", с которым рекомендуется ознакомится перед прочтением данного документа.

Глава 2

Общее описание устройства модуля L502

Данный раздел предоставляет дополнительные сведения о низкоуровневом устройстве платы L502, которые необходимы для создания своих драйверов или своей прошивки сигнального процессора. В этом разделе в первую очередь будет рассматриваться низкоуровневое устройство с точки зрения программиста, а не непосредственно аппаратной реализации.

Ниже представлена общая схема L502 с точки зрения взаимодействия различных составных частей модуля L502.

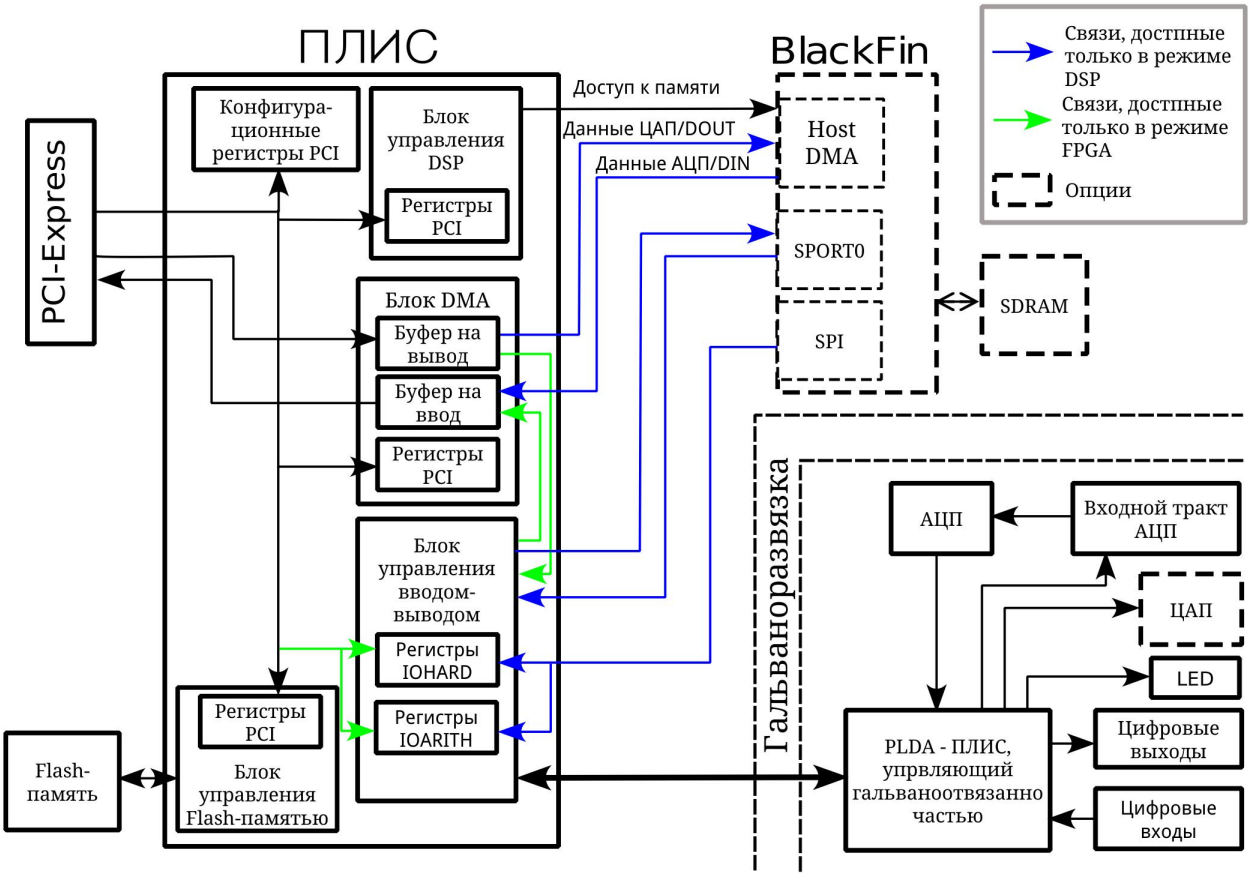

Рис. 2.1: Взаимодействие функциональных блоков модуля L502

Как видно из рисунка, модуль состоит из следующих блоков:

1. ПЛИС. В нем реализовано:
   - Вся логика интерфейса PCI-Express
   - Логика доступа к памяти сигнального процессора с ПК через регистры PCI
   - Блок DMA для прямой передачи данных АЦП/DIN в память ПК и чтения данных ЦАП/DOUT из памяти ПК.
   - Блок управления вводом-выводом. В нем реализовано две группы регистров:
     - IO_HARD - для настройки параметров сбора данных (логическая таблица, делитель частоты, режим синхронизации и т.д.)
     - IO_ARITH - для дополнительной обработки (аппаратное применение калибровочных коэффициентов)
   - Реализованы интерфейсы к Flash-памяти, Сигнальному процессору и гальваноотвязанной части.
2. Flash-память SST25VF16B — память на 2 Мбайта. Старшая половина пользователю не доступна — в ней хранится прошивка ПЛИС, ее резервная копия, калибровочные коэффициенты и серийный номер изделия. Младшая половина доступна пользователю для чтения и записи, и может быть использована для каких-либо своих целей. Следует однако учитывать, что Flash-память доступна только для ПК, и к ней нет прямого доступа у сигнального процессора.
3. Сигнальный процессора ADSP-BF523. Может быть использован для написания собственных программ обработки данных внутри модуля для расширения возможностей L502.
4. PLDA - ПЛИС, управляющий гальваноотвязанной частью. Именно на гальваноотвязанной части находятся все сигнальные входы и выходы модуля, АЦП и ЦАП. Ими непосредственно управляет PLDA. Сам PLDA управляется через гальванноразвязанный интерфейс основным ПЛИС и логика работы PLDA скрыта от пользователя, т.к. пользователю доступно управление сбором данных только через регистры блока ввода-вывода основного ПЛИС.

Все управление модулем со стороны ПК идет через регистры PCI-Express (с программной стороны аналогичны регистрам PCI) и описано подробно в следующем разделе.

Как видно из рисунка, плата может работать в двух режимах: штатном (всю обработку выполняет ПЛИС) и DSP-режиме. В зависимости от режима меняются путь, который проходят потоки данных, а также доступ к управляющим регистрам ввода-вывода.

Если в штатном режиме данные, приходящие по гальваноотвязанному интерфейсу (АЦП и DIN), передаются сразу в буфер блока DMA для последующей записи по интерфейсу PCI-Express непосредственно в память ПК, то в DSP-режиме данные передаются через SPORT0 в BlackFin, который может обработать эти данные и, при желании, поставить их на передачу в ПК. В этом случае ПЛИС по HostDMA прочитает данные из памяти BlackFin и передаст их через блок DMA в память ПК.

Аналогично для потока на вывод в режиме DSP данные, прочитанные блоком DMA из памяти ПК, будут записаны через интерфейс HostDMA в память BlackFin, где могут

быть обработаны сигнальным процессором и переданы через SPORT0 обратно в ПЛИС для дальнейшего их вывода на ЦАП или цифровые выходы.

Также при переходе в режим DSP изменяется право доступа к регистрам IO_HARD и IO_ARITH . Если в штатном режиме данные регистры доступны для программирования со стороны ПК как обычные регистры PCI, то в режиме DSP управление этими регистрами передается сигнальному процессору, который может через интерфейс SPI записывать и считывать их содержимое. При этом состав регистров и относительные адреса остаются теми же.

Внимание! : Здесь и далее названия направлений передачи потоков (ввод или вывод) даются относительно ПК. То есть под потоком ввода (IN) понимается поток данных АЦП/DIN, а под потоком вывода (OUT) — поток данных ЦАП/DOUT, даже когда речь идет о прошивке сигнального процессора.

Глава 3

Управление модулем со стороны ПК по интервейсу PCI-Express

3.1 Введение

В данной главе описывается интерфейс между модулем L502 и персональным компьютером. Этот раздел в первую очередь предназначен для пользователей, пишущих свой драйвер и библиотеку для работы с модулем. Для пользователей, разрабатывающих свою прошивку для сигнального процессора BlackFin, из данной главы будет полезен в первую очередь раздел 3.4.3, в котором приводится описание регистров управления вводом-выводом, так как в DSP-режиме этими регистрами управляет сигнальный процессор.

3.2 Конфигурационные регистры

Управление модулем L502 со стороны ПК осуществляется через интерфейс PCI Express. При этом программная модель шины PCI Express полностью совместима с программной моделью PCI.

Как и любое PCI/PCI-Express устройство, модуль реализует набор конфигурационных регистров, который состоит из фиксированного заголовка (Header 0) и связного списка дополнительных свойств (Capabilities List).

Конфигурационные регистры используются для определения, что это за устройство (по регистрам VendorID и DevceID), для назначения ресурсов устройству (областей памяти, номеров прерываний) и для общих функций управления устройством. Как правило, запись в конфигурационные регистры выполняется BIOSом и операционной системой, и драйверу прямая запись в эти регистры не требуется.

Эти регистры стандартны и полностью описаны в *"PCI Local Bus Specification Revision 3.0"* в главе 6 — *"Configuration Space"* . Дополнительные свойства специфичные для PCI Express описаны в *"PCI Express в PCI Express Specification 3.0"* в главе 7 *"Software Initialization and Configuration"* .

Значения регистров, специфичные именно для модуля L502:

- Идентификаторы VendorID = 0x1172, DeviceID=0x0502
- Используется один банк памяти, который может быть расположен в любом месте 64-битного адресного пространства (занимает 2 первых BAR-регистра)

- Реализованные свойства:
  - PCI Express Capability
  - MSI Capability
  - Power Managment Capability

3.3 Прерывания

В модуле L502 реализовано два механизма генерации прерываний:

- Эмуляция стандартного прерывания PCI INTA. Этот режим включен по-умолчанию и предназначен для систем без поддержки MSI. Прерывания срабатывают по уровню и разделяются несколькими устройствами, что приводит к необходимости при прерывании определять, какое именно устройство сгенерировало это прерывание.

- Message Signaled Interrupt (MSI) — поддерживаются в версиях Windows, начиная с Windows Vista, и в Linux, начиная с версии ядра 2.6.8. Для их настройки используются набор конфигурационных регистров `MSI Capability''. При разрешении MSI стандартное INTA прерывание не используется. В MSI прерывание эмулируется записью заданного значения по заданному адресу. Адрес и значение задаются в регистрах из`MSI Capability" и определяются операционной системой или биосом. Такой вариант прерываний дает возможность каждому устройству выделить свой вектор прерывания, что позволяет ускорить обработку, так как не надо определять, кто является источником прерывания, так как при срабатывании прерывания выполняется только один обработчик. В отличие от INTA MSI соответствуют прерыванию по фронту, а не по уроню. Рекомендуется использовать MSI прерывание всегда, когда это возможно.

В L502 используется только одно прерывание. Прерывание используется блоком DMA, чтобы сообщить, что блок данных был записан в память ПК или считан из нее. Для определения источника прерывания достаточно прочитать значение регистра `DMA_IRQ`.

3.4 Доступ к регистрам ПЛИС через память

Модуль L502 использует один регион памяти размером 16КБайт, который может быть спроецирован в любую область 64-битного адресного пространства (для задания адреса используются два первых BAR-регистра в конфигурационном пространстве). Назначение области памяти обычно выполняется биосом или операционной системой, а драйвер от уже как правило получает указатель на требуемую область в своем адресном пространстве с помощью функций, предоставляемых ОС.

Работа с модулем главным образом выполняется путем записи/чтения значений по определенным адресам этой области. При обращении к этой памяти, реально выполняется обращение к регистрам ПЛИС, с помощью которых и осуществляется управление модулем. Все регистры 32-битные. В дальнейшем в качестве адреса регистра будет приведен номер 32-битного регистра. Так как в PCI и PCI-Express адресуемой единицей является байт, то для получения адреса регистра в памяти нужно к начальному адресу области памяти, выделенной модулю, прибавить адрес регистра умноженный на 4:

```
mem_addr = mem_offs + 4*reg_addr
```

Регистры в памяти поделены по своему назначению на блоки, каждый из которых имеет свой диапазон адресов:

Адреса регистров	**Блок регистров**	**Описание**
0x0000 - 0x00FF	BF_CONTROL	Управление доступом к DSP по HostDMA
0x0100 - 0x013F	EEPROM	Доступ к Flash-памяти
0x0140 - 0x017F	DBG	Регистры с отладочной информацией
0x0180 - 0x01FF	-	Резерв
0x0200 - 0x03FF	IO_HARD	Управление вводом-выводом
0x0400 - 0x04FF	IO_ARITH	Дополнительная обработка
0x0500 - 0x05FF	-	Резерв
0x0600 - 0x0FFF	DMA	Управление прямым доступом к памяти

В последующих разделах будут рассмотрены регистры всех этих блоков. При этом в дальнейшем номера регистров будут приводится относительно адреса начала блока.

3.4.1 Блок регистров для управления доступом к сигнальному процессору BlackFin

Данный блок регистров используются для следующих целей:

- Управление сигналом сброса сигнального процессора BlackFin
- Перевод модуля L502 в режим работы с DSP или возврат в режим работы без сигнального процессора
- Прямой доступ к памяти BlackFin по HostDMA для чтения или записи блока данных

В Таблице 3.2 приведен полный список регистров данного блока.

Таблица 3.1: Регистры блока управления сигнальным процессором

Адрес	Регистр	Доступ	Описание
0	`BF_CTL`	RW	Управление сбросом сигнального процессора и режимом работы модуля. Описание битов приведено в Таблице 3.2.
1	`BF_CMD`	W	Запись команды на выполнение по интерфейсу HDMA. Доступные коды команд приведены в Таблице 3.4
2	`BF_STATUS`	R	Статус выполнения команды по HDMA
5	`BF_REQ_ADDR`	W	Адрес в памяти BlackFin для прямого доступа
6	`BF_REQ_SIZE`	W	Размер данных для обмена по HostDMA
128-255	`BF_REQ_DATA`	RW	Данные на запись или прочитанные данные по HostDMA

Таблица 3.2: Биты регистра BF_CTL

Биты	Обозначение	Доступ	Описание
1	BF_RST	RW	Управление линией сброса Сигнального процессора (0-процессор в сброшенным состоянии, 1 - рабочий режим)
3	BF_HWAIT	R	Значение одноименного выхода процессора. Используется для определения готовности процессора к загрузке (должен быть равным 0)
4	DSP_MODE	RW	При установке в 1 модуль переводится в режим работы с DSP
5	DBG_MODE	RW	Включение отладочного режима (должен быть равен 0)
8-11	HDMA_CLK_DIV	RW	Деление частоты клока автомата HostDMA (в штатном режиме всегда должен быть 0!)
0,2,12-31	-	-	Резерв

Таблица 3.3: Биты регистра BF_STATUS

Биты	Обозначение	Доступ	Описание
0	BF_HWAIT	R	Аналогично биту BF_HWAIT в регистре BF_CTL
1	BF_REQ_BUSY	R	1 является признакам, что в настоящее время выполняется команда, код которой был записан в регистр BF_CMD и запускать новую команду нельзя
2-31	-	-	Резерв

Перед началом работы с сигнальным процессором необходимо снять сигнал сброса (делается драйвером при обнаружении платы), установив бит BF_RST = 1, и перевести модуль в режим работы с сигнальным процессором, установив DSP_MODE = 1.

Для выполнения команд по HostDMA предназначен регистр BF_CMD, в который должен быть записан код команды. После этого бит BF_REQ_BUSY регистра BF_STATUS будут установлен в 1 и будет оставаться в 1 до завершения выполнения команды. Его надо проверять, так как записывать новый код команды до завершения предыдущей нельзя. Список команд приведен ниже:

Для чтения или записи в память сигнального процессора BlackFin используется специальный внутренний буфер ПЛИС на 128 32-битных слов, который проецируется на регистры PCI, начиная с адреса регистра BF_REQ_DATA. Обмен идет всегда между этим буфером и памятью BlackFin, т.е. для записи в память нужно сперва записать блок данных в этот буфер, затем задать параметры команды и запустить команду на запись. Для чтения после завершения команды прочитанные данные будут в этом промежуточном буфере и их надо из него считать.

Запись блока данных в память BlackFin выполняется следующим образом:

- Дождаться, пока бит BF_REQ_BUSY регистра BF_STATUS будет равен 0
- Записать от 1 до 128 32-битных слов, которые нужно будет записать в память BlackFin, в регистры PCI начиная с адреса BF_REQ_DATA.
- Записать размер записываемых данных в 32-битных словах в регистр BF_REQ_SIZE

Таблица 3.4: Команды для интерфейса HostDMA

Команда	Код	Описание
L502_BF_CMD_READ	1	Запуск чтения данных из памяти BlackFin по адресу из регистра BF_REQ_ADDR размером, заданным в регистре BF_REQ_SIZE
L502_BF_CMD_WRITE	2	Запуск записи данных в память BlackFin по адресу из регистра BF_REQ_ADDR размером, заданным в регистре BF_REQ_SIZE
L502_BF_CMD_HIRQ	4	Подача команды HOST_IRQ по интерфейсу HostDMA. Используется исключительно для загрузки прошивки перед записью последнего блока данных по HostDMA
L502_BF_CMD_HDMA_RST	8	Сброс автомата для работы с HostDMA в ПЛИС. Используется для случая, если цикл на HostDMA не был завершен (напримемер по причине остановки BlackFin на точке отсанова или зависании прошивки) для приведения автомата в известное состояние

- Записать адрес памяти BlackFin, начиная с которого должны быть записаны данные, в регистр BF_REQ_ADDR
- Записать в регистр BF_CMD код команды L502_BF_CMD_WRITE.
- Дождаться завершения команды — пока бит BF_REQ_BUSY регистра BF_STATUS не станет равен 0

Чтение блока данных в память BlackFin выполняется следующим образом:

- Дождаться, пока бит BF_REQ_BUSY регистра BF_STATUS будет равен 0
- Записать размер данных на чтение в 32-битных словах в регистр BF_REQ_SIZE
- Записать адрес памяти BlackFin, начиная с которого должны быть прочитаны данные, в регистр BF_REQ_ADDR
- Записать в регистр BF_CMD код команды L502_BF_CMD_READ.
- Дождаться завершения команды — пока бит BF_REQ_BUSY регистра BF_STATUS не станет равен 0
- Прочитать считанные данные из регистров, начиная BF_REQ_DATA.

Следует учитывать, что чтение и запись по HostDMA реально выполняются блоками, кратными 8-ми 32-битным словам. Поэтому при записи некратного блока реально будет испорчена часть данных после блока до ближайшей кратной границы. Чтение выполняется аналогично размером, кратным 8-ми 32-битным словам. Хотя в данном случае повреждения данных быть не может, следует следить, чтобы все адреса чтения до ближайшей кратной границы были действительными.

Загрузка прошивки BlackFin выполняется с помощью обычных команд прямой записи в память сигнального процессора, поэтому для записи необходимо вручную разбирать ldr файл прошивки. Запись выполняется следующим образом:

- Сбросить сигнальный процессор путем записи нуля в BF_RST регистра BF_CTL, выждать паузу (1 мс) и снять сброс записью в этот бит 1.
- Дождаться готовности загрузчика BlackFin (бит BF_HWAIT должен стать 0)
- Записать все блоки данных, кроме последнего, по HostDMA
- Перед последним блоком данных нужно подать команду HOST_IRQ путем записи кода L502_BF_CMD_HIRQ в регистр BF_CMD
- Записать последний блок данных

Подробнее о загрузке по HostDMA можно прочитать в специальном Application Note, а о работе интерфейса HostDMA и о формате .ldr в Hardware Manual.

3.4.2 Блок регистров для управления DMA

3.4.2.1 Общее описание

Блок DMA предназначен для организации прямого доступа к памяти ПК из модуля L502 для передачи потоков данных на ввод и на вывод.

В блоке реализовано 2 канала DMA:

- Канал 0 на ввод. Для передачи отсчетов АЦП и состояний цифровых входов в память ПК.
- Канал 1 на вывод. Для чтения отсчетов ЦАП и значений цифровых выходов из памяти ПК.

Каждому каналу DMA соответствует свой блок регистров, в котором хранятся настройки данного канала:

- до 252 дескрипторов страниц памяти ПК
- размер пакета для передачи по PCI-Express
- шаг прерывания (через сколько переданных слов будет генерироваться прерывание)
- режим работы: циклический режим или однократный запуск на один шаг
- разрешение управления потоком, чтобы избежать затирание кольцевого буфера в ПК при переполнении

Список регистров для управления DMA приведен в Таблице 3.9. Эти регистры как правило устанавливаются перед началом сбора данных. Исключением является регистр DMA_CH_PC_POS и дескрипторы станиц, при условии, что данные дескрипторы не используются в момент обновления для передачи.

Кроме того, существует отдельная область регистров, общих для всех каналов, которые нужны для управления каналами DMA и отслеживания их состояния:

- разрешение или запрещение каналов DMA
- разрешение или запрещение генерации прерываний (могут использоваться как прерывания по заданному шагу, так и по завершению обработки заданной страницы)

- чтение флагов прерываний для определения, какие прерывания произошли
- сброс каналов DMA

Для каждого действия реализован свой регистр, каждый бит которого соответствует одному каналу (или одному из источников прерываний), что позволяет изменять состояние нескольких каналов одновременно. Эти регистры всегда доступны как на чтение, так и на запись, независимо от того, какие каналы разрешены.

Распределение адресного пространства между общими регистрами и параметрами каждого канала приведено в Таблице 3.5.

Таблица 3.5: Распределение адресного пространства блока DMA

Адрес	Блок
0x700-0x800	Общие регистры управления каналами DMA
0x800-0xBFF	Параметры канала DMA на ввод
0xC00-0xFFF	Параметры канала DMA на вывод

3.4.2.2 Дескрипторы страниц

Для задания области памяти ПК, в которую модуль будет напрямую записывать данные (или из которой считывать), используется набор дескрипторов страниц. Дескриптор представляет собой набор из 4-х 32-битных регистров, значение которых приведено в Таблице 3.12). Каждый дескриптор страницы задает следующие параметры:

- Физический 64-битный адрес страницы, который могут использовать PCI/PCI-Express устройства
- Размер страницы
- Флаг останова, задающий, нужно ли автоматически остановить передачу после завершения обмена, соответствующего данной странице
- Флаг прерывания, задающий, нужно ли устанавливать соответствующий признак прерывания по завершению обмена, соответствующего данной странице

Следует отметить, что страница может находится в произвольной области 64-битного адресного пространства ПК и должна быть выравнена по границе 32-битного слова (последние 2 бита адреса равны 0). Размер может быть свой для каждой страницы. Область памяти, соответствующая одной странице, должна быть непрерывна (по крайней мере с точки зрения адресного пространства шины PCI/PCI-Express). Для задания не непрерывной области используется несколько дескрипторов страниц.

Количество используемых страниц может быть произвольное в диапазоне от 1 до 252 страниц и задается полем `DMA_CTL_PAGES_CNT` регистра `DMA_CH_CTL`. При этом они используются циклически: по завершению передачи, соответствующей странице с номером `DMA_CTL_PAGES_CNT` - 1, начинается снова передача, соответствующая дескриптору страницы с номером 0.

Дескрипторы страниц как правило задаются до разрешения соответствующего канала DMA, однако при необходимости могут изменяться и во время работы при условии, что гарантированно изменяются только те дескрипторы, которые не используются в данный момент для обмена.

Следует учитывать специфику работы с буферами, доступными для работы как с ПК, так и по DMA для используемой архитектуры (например влияние включенной кэш-памяти). Как правило ОС предоставляет функции, которые переводят буфер либо в режим доступа со стороны ПК, либо в режим доступа со стороны устройства.

3.4.2.3 Запуск и останов канала DMA

Для начального запуска канал DMA, нужно:

- Убедиться, что канал остановлен по соответствующему биту регистра DMA_EN. Если нет, то остановить DMA (см. ниже).
- Запретить прерывания по данному каналу через биты регистра DMA_IRQ_DIS.
- Сбросить все оставшиеся признаки прерываний записью 1 в биты регистра DMA_IRQ, связанные с этим каналом.
- Установить в нулевые значения регистры DMA_CH_CUR_CNTR, DMA_CH_CUR_POS и DMA_CH_PC_POS.
- Выполнить сброс канала, с помощью записи 1 в соответствующий бит регистра DMA_RST
- Заполнить таблицу страниц, записать нужные настройки в DMA_CH_CTL и DMA_CH_CMP_CNTR.
- Разрешить нужные прерывания с помощью битов регистра DMA_IRQ_EN
- Запустить DMA по заданному каналу с помощью бита регистра DMA_EN

Для останова канала DMA нужно:

- Запретить прерывания по данному каналу через биты регистра DMA_IRQ_DIS
- Запретить обмен по заданному каналу с помощью записи 1 в нужный бит регистра DMA_DIS
- Дождаться, пока канал будет остановлен (при чтении из DMA_EN соответствующий бит будет равен 0)
- Сбросить все оставшиеся признаки прерываний записью 1 в биты регистра DMA_IRQ, связанные с этим каналом.

3.4.2.4 Однократный запуск DMA

Если необходимо запустить передачу данных на заданное число отсчетов, после чего остановить передачу и обработать данные, можно воспользоваться режимом однократного сбора. Для этого после сброса канала нужно выполнить следующее:

- Подготовить буфер в памяти ПК на заданное количество слов
- Заполнить дескрипторы страниц параметрами страниц, соответствующими выделенному буферу.
- Записать шаг прерывания, равный передаваемому размеру, в регистр DMA_CH_CMP_CNTR.

- Установить регистр DMA_CH_CTL: PCIE_PACK_SIZE=32, DMA_CTL_PAGES_CNT = кол-во страниц в буфере, DMA_CTL_AUTOSTOP=1.
- Разрешить прерывание с помощью бита DMA_IRQ_STEP_IN или DMA_IRQ_STEP_OUT регистра DMA_IRQ_EN.
- Запустить DMA по заданному каналу
- Дождаться, пока придет прерывание и бит DMA_IRQ_STEP_IN или DMA_IRQ_STEP_OUT регистра DMA_IRQ будет установлен в 1.
- Обработать принятый буфер

3.4.2.5 Циклический запуск DMA

В простейшем случае для запуска циклического режима достаточно выполнить действия, аналогичные однократному запуску, за исключением того, что бит DMA_CTL_AUTOSTOP устанавливается в 0, а шаг прерывания соответствует только части буфера, а не всему.

При этом модуль будет записывать или считывать данные из буфера по кругу, генерируя прерывания при завершении передачи количества слов, кратного заданному в DMA_CH_CMP_CNTR значению.

Однако, если по какой-либо причине, обработка данных программой сильно задержится, то возможно ситуация, что модуль обгонит драйвер на размер буфера и запишет данные в область, где лежали еще необработанные данные с предыдущего круга. Для избежания этого можно использовать режим управления потоком.

3.4.2.6 Управление потоком

При циклическом режиме случай, когда модуль обогнал программу на один цикл, может привести к следующим последствиям:

- При вводе будет испорчена часть необработанных данных, а при выводе будет выведен мусор
- В случае, если к буферу может быть доступ либо только со стороны драйвера, либо только со стороны модуля, это приведет к тому, что модуль будут обращаться к буферу, не настроенному на доступ со стороны устройства.

Для избежания подобной ситуации можно включить режим управления потоком с помощью бита DMA_CTL_PC_WAIT регистра DMA_CH_CTL. В этом случае при подобном сценарии модуль будет ждать, пока драйвер обработает данные для ввода и освободит место или подготовит данные для вывода.

Таким образом драйвер должен сам сообщать о том, какую часть данных он обработал. Для этого используется регистр DMA_CH_PC_POS. В случае ввода драйвер в этот регистр должен записывать позицию сразу за последнем обработанным словом (т.е. номер первого необработанного слова, по адресу которого нельзя писать данные, если модуль обгонит драйвер на цикл), а при выводе — номер слова, идущего за последним подготовленным на вывод словом (т.е. также номер слова, который модуль еще не может считывать).

Модуль выполняет в этом режиме обмен данными до того момента, пока его позиция передаваемых слов из регистра DMA_CH_CUR_POS не станет равна значению в регистре DMA_CH_PC_POS.

Отдельно следует отметить случай, когда буфер либо полностью заполнен, либо полностью пуст. В обоих случаях значение DMA_CH_CUR_POS равно значению DMA_CH_PC_POS. После сброса считается, что в буфере на ввод нет ни одного свободного слова, а в буфере на вывод — ни одного подготовленного. Т.е. после сброса даже при разрешении DMA обмен не начнется, пока не будет изменено значение регистра DMA_CH_CUR_POS. При работе блок DMA отслеживает для каждого канала, какое изменение было последним: обновлен регистр DMA_CH_PC_POS со стороны драйвера или обновлен регистр DMA_CH_CUR_POS в результате завершения передачи пакета по PCI-Express. Если позиции сравнялись в результате первого события, то это означает, что драйвер подготовил или обработал новый блок данных и можно продолжить обмен. Второй случай означает что модуль "догнал" драйвер и нет свободного места на ввод или нет новых данных на вывод. В последнем случае модуль будет ожидать, пока не выполнится следующая запись в регистр DMA_CH_PC_POS.

Следует отметить, что запись в регистр DMA_CH_PC_POS должна производиться драйвером только в момент, когда количество обработанных или подготовленных слов изменилось! В противном случае повторная запись того же значения при не изменившемся значении регистра может привести к тому, что блок DMA распознает это события как факт обработки драйвером сразу всего буфера.

Следует также отметить, что если при вводе модуль полностью заполнит буфер в ПК на передачу, то после этого начинается заполнение внутреннего буфера блока DMA, соответствующего данному каналу. При работе без сигнального процессора при его переполнении будет взведен флаг и отсчеты, для которых не найдется место, будут отброшены. Если в дальнейшем в буфере ПК освободится место, то в точке потери данных будет вставлено в поток сообщение, сигнализирующее о переполнении. В случае работы с сигнальным процессором в таком сценарии начинает заполняться буфер сигнального процессора и прошивка сигнального процессора сама обрабатывает это событие. Штатная прошивка выполняет аналогичные действия, с той лишь разницей, что добавляется внутренний буфер в SDRAM, который должен также переполниться для возникновения события потери данных.

3.4.2.7 Изменение страниц DMA на лету

В самом простейшем случае параметры страниц, соответствующие выделенному буферу, заносятся перед запуском канала DMA и не изменяются в процессе работы. Однако это не всегда возможно, например:

- если сам буфер должен формироваться димнамически
- если предоставленного количества дескрипторов страниц в модуле не достаточно для представления буфера

Последний случай вполне вероятен из-за следующих проблем:

- При больших скоростях ввода буфер должен быть достаточно большим, чтобы не переполнится даже при существенных задержках работы приложения
- Память ПК во время работы как правило сильно фрагментирована, поэтому в общем случае желательно не зависеть от наличия в системе больших сегментов

непрерывного пространства памяти (в идеале работать при размерах страниц равном размеру страниц ПК — 4КБайта)

- На некоторых системах для доступа к памяти через шины (PCI/PCI-Express) могут понадобится специальные ресурсы (mapping-регистры), которые также ограничены

- Размер памяти внутри модуля ограничен, и модуль не может хранить большое количество дескрипторов (максимум — 252 на канал)

В этих условиях в штатных драйверах для общего случая используется динамическое назначение страниц. В каждый момент времени только часть буфера спроецирована на память PCI/PCI-Express и соответствующие только этой части дескрипторы страниц записаны в модуль. Передача разделены на блоки (транзакции). Размер всех транзакций не должен превышать 252 страницы. При этом для непрерывной передачи должно быть не менее 2-х транзакций (чтобы по завершению одной шла передача по другим). По завершению каждой транзакции добавляется новая, которая указывает на не использованную (не поместившуюся до этого) часть общего буфера и может использовать освободившиеся от предыдущей дескрипторы страниц. При этом используется управление потоком, чтобы модуль не использовал повторно уже завершенную транзакцию. Для определения завершения транзакции используется возможность установить флаг прерывания для последней страницы.

3.4.2.8 Значения регистров и их полей

Таблица 3.6: Общие регистры управления каналами DMA

Адрес	Регистр	Доступ	Описание
1	DMA_EN	RW	Разрешение указанных каналов DMA. Состояние каналов каналов, в соответствующие биты которых записывается 0, не изменяется. При чтении возвращается значение, определяющее, какие каналы разрешены. Соответствие битов и каналов приведено в Таблице 3.7.
2	DMA_DIS	W	Запрет указанных каналов DMA. Если в это время передается пакет по этому каналу, то канал будет остановлен после завершения передачи. Чтобы убедиться, что канал остановлен, можно проверить соответствующий бит регистра DMA_EN. Соответствие битов и каналов приведено в Таблице 3.7.
3	DMA_RST	W	Сброс указанных каналов. При сбросе очищается внутренний буфер канала DMA. Соответствие битов и каналов приведено в Таблице 3.7.
4	DMA_IRQ	RW	Биты этого регистра устанавливаются при возникновении соответствующих событий для каналов DMA, независимо от регистра DMA_IRQ_EN. Общий сигнал прерывания становится активным, если установлен хотя бы один бит, для которого также установлена 1 в DMA_IRQ_EN. Общий сигнал прерывания используется для генерации либо MSI-прерываний (по фронту) или эмуляции прерывания INTA (по уровню). При записи сбрасываются те биты, которые в записываемом слове установлены в 1. Соответствие битов регистра и источников прерываний приведено в Таблице 3.8.
5	DMA_IRQ_EN	RW	Разрешение прерываний, для которых биты записанного слова равны 1. Прерывания, для которых соответствующие биты равны 0, остаются разрешенными или запрещенными без изменения. Чтение позволяет определить, какие прерывания разрешены. Соответствие битов регистра и источников прерываний приведено в Таблице 3.8.
6	DMA_IRQ_DIS	W	Запрет прерываний, для которых биты записанного слова равны 1. Соответствие битов регистра и источников прерываний приведено в Таблице 3.8.

Таблица 3.7: Биты регистров, определяющие используемые каналы DMA

Биты	Обозначение	Описание
0	DMA_CH_IN	Канал DMA на ввод (АЦП/DIN)
1	DMA_CH_OUT	Канал DMA на вывод (ЦАП/DOUT)
31-2	-	Резерв

Таблица 3.8: Биты регистров, определяющие используемые источники прерываний DMA

Биты	Обозначение	Описание
0	DMA_IRQ_STEP_IN	Прерывание, возникающее при завершении записи модулем количества слов, равного установленному шагу, для канала ввода (АЦП/DIN)
1	DMA_IRQ_STEP_OUT	Прерывание, возникающее при завершении чтения модулем количества слов, равного установленному шагу, для канала ввода (ЦАП/DOUT)
8	DMA_IRQ_PAGE_IN	Прерывание, возникающее при завершении записи данных, соответствующих странице с установленным флагом генерации прерываний (канал ввода)
9	DMA_IRQ_PAGE_OUT	Прерывание, возникающее при завершении чтения данных, соответствующих странице с установленным флагом генерации прерываний (канал вывода)
16	DMA_IRQ_FLUSH_IN	Прерывание, возникающее при завершении записи блока данных из сигнального процессора, в дескрипторе которого был установлен признак последнего дескриптора, и одновременному опустошению внутреннего буфера. Также по этому прерыванию сбрасывается флаг разрешения соответствующего канала DMA
31-17, 15-10, 7-2	-	Резерв

Таблица 3.9: Регистры параметров канала DMA

Адрес	Регистр	Доступ	Описание
0	DMA_CH_CTL	RW	Устанавливает режим работы канала DMA. Соответствие битов и каналов приведено в Таблице 3.10.
1	DMA_CH_CMP_CNTR	RW	Задает шаг прерывания для канала в количестве 32-битных отсчетов.
2	DMA_CH_CUR_CNTR	R(W)	Текущее значение счетчика шага канала. Указывает сколько слов из текущего шага передано
3	DMA_CH_CUR_POS	R(W)	Указывает номер страницы и смещение внутри страницы, по которым будет выполнена следующая передача (биты приведены в Таблице 3.11).
4	DMA_CH_PC_POS	W	Указывает номер страницы и смещение внутри страницы за последним обработанным программой ПК словом. Используется только при включенном режиме управления потоком. (биты приведены в Таблице 3.11)
4-15	-	-	Резерв
16-19	DMA_PAGE0	W	Дескриптор первой страницы DMA (состав регистров приведен в Таблице 3.12)
...			
1020-1023	DMA_PAGE251	W	Дескриптор последней (252-ой) страницы DMA

Таблица 3.10: Биты регистра DMA_CH_CTL

Биты	Обозначение	Описание
0-7	PCIE_PACK_SIZE	Максимальный размер пакета при передаче по PCI-Express в 32-битных словах. Должен быть от 1 до 32. В штатном режиме можно всегда задавать 32, т.к. автомат DMA автоматически корректирует размер пакета, если до конца страницы или шага прерываний осталось меньше полного размера пакета
16-23	DMA_CTL_PAGES_CNT	Количество используемых страниц DMA (от 1 до 252). После выполнения обмена по странице с номером DMA_CTL_PAGES_CNT-1, будет выполнен циклический переход к странице с номером 0
30	DMA_CTL_PC_WAIT	Включение управления потоком. В этом режиме канал DMA будет ожидать завершения обработки данных ПК
31	DMA_CTL_AUTOSTOP	Включение однократного режима. По завершению передачи указанного в регистре DMA_CH_CMP_CNTR числа слов, DMA будет автоматически остановлен
8-15, 24-29	-	Резерв

Таблица 3.11: Биты регистров, задающих текущую позицию канала DMA

Биты	Обозначение	Описание
0-23	DMA_POS_OFFSET	Текущее смещение внутри страницы в 32-битных словах
24-31	DMA_POS_PAGE	Текущий номер страницы (от 0 до DMA_CTL_PAGES_CNT-1)

Таблица 3.12: Регистры, задающие дескриптор страницы DMA

Адрес	Регистр	Доступ	Описание
0	DMA_CH_PAGE_ADDRL	W	Младшие 32-бита адреса страницы памяти ПК на шине PCI-Express. Младшие 2 бита должны бить всегда равны 0 (память выровнена по 32-битному слову)
1	DMA_CH_PAGE_ADDRH	W	Старшие 32-бита адреса страницы памяти ПК на шине PCI-Express. Если используется 32-битное адресное пространство, то должен быть записан 0.
2	DMA_CH_PAGE_LEN	W	Размер страницы и флаги (значение битов приведено в Таблице 3.13)
3	-	-	Резерв

Таблица 3.13: Биты регистра DMA_CH_PAGE_LEN страниц DMA

Биты	Обозначение	Описание
0-23	DMA_CH_PAGE_LEN	Размер страницы в 32-битных словах (от 1 до $2^{24}-1$)
24-29	—	Резерв
30	DMA_CH_PAGE_STOP	Если установлен бит, то по завершению передачи, соответствующей данной странице, канал DMA будет автоматически запрещен
31	DMA_CH_PAGE_IRQ	Если установлен бит, то по завершению передачи, соответствующей данной странице, будет установлен бит DMA_IRQ_PAGE_IN или DMA_IRQ_PAGE_OUT

3.4.3 Блок регистров для управления вводом-выводом

Данные регистры содержат настройки ввода-вывода, управление синхронным вводом-выводом, а также регистры для асинхронного вывода на ЦАП/DOUT. К этим регистрам предоставляется доступ либо со стороны ПК через область памяти PCI, либо прошивке BlackFin через интерфейс SPI — в зависимости от режима работы модуля.

При синхронном вводе последовательность инициализации регистров может выглядеть следующим образом:

- Запись нужных настроек синхронного ввода-вывода
- Разрешение синхронных потоков ввода с помощью регистра IN_STREAM_ENABLE
- При синхронном выводе подготовка блока данных для вывода, настройка DMA или SPORT0, и запись 1 в бит OUT_SWAP регистра OUTSWAP_BFCTL.
- При работе из DSP разрешение генерации TFS и/или RFS с помощью битов BF_TFS_EN и BF_RFS_EN регистра OUTSWAP_BFCTL соответственно.
- Запись 1 в PRELOAD_ADC
- По записи 1 в GO_SYNC_IO запускается синхронный ввод-вывод (или ожидание внешнего события старта) и изменения настроек уже не возможно
- По завершению сбора в GO_SYNC_IO записывается 0
- При работе из DSP запрещается генерация TFS и RFS с помощью битов BF_TFS_EN и BF_RFS_EN регистра OUTSWAP_BFCTL соответственно

Регистры управления вводом-выводом условно разделены на два блока — с основными настройками (IO_HARD) и дополнительными операциями (IO_ARITH), такими как калибровка, разрешение потоков ввода и др. Важно при записи настроек учитывать, что запись делителя частоты АЦП должна выполняться в регистр ADC_FREQ_DIV как в блока IO_HARD, так и IO_ARITH.

В Таблице 3.14 приведен список регистров блока IO_HARD, а в Таблице 3.15 — регистров блока IO_ARITH. Все адреса приведены относительна адреса начала блока.

Таблица 3.14: Регистры блока управления вводом-выводом (IO_HARD)

Адрес	Регистр	Доступ	Описание
0x0-0xFF	LTABLE	RW	Таблица настроек логических каналов, определяющих последовательность опроса каналов АЦП. Каждый 32-битный регистр задает настройки одного логического канала, при этом по адресу 0 находятся настройки последнего логического канала, 1 — предпоследнего, N-1 — первого. Формат настроек приведен в Таблице 3.16.
0x100	LCH_CNT	RW	Задает количество логических каналов в управляющей таблице АЦП. Размер логической таблицы равен LCH_CNT + 1.
0x102	ADC_FREQ_DIV	RW	Делитель частоты синхронного ввода с АЦП. Частота ввода с АЦП равна fref/(ADC_FREQ_DIV+1), где fref — опорная частота синхронизации
0x104	ADC_FRAME_DELAY	RW	Значение межкадровой задержки в периодах опорной частоты
0x106	DIGIN_FREQ_DIV	RW	Делитель частоты синхронного ввода с цифровых входов. Частота ввода с DIN равна fref/(DIGIN_FREQ_DIV+1), где fref — опорная частота синхронизации
0x108	IO_MODE	RW	Регистр задает режим синхронизации старта синхронного ввода и делитель ЦАП. Также при чтении может использоваться для проверки захвата частоты синхронизации. Значение битов приведено в Таблице 3.17
0x10A	GO_SYNC_IO	W	Запись 1 в этот регистр приводит к запуску синхронного ввода-вывода, а 0 - к его останову.
0x10C	PRELOAD_ADC	W	Перед записью 1 в регистр GO_SYNC_IO необходимо выполнить запись в этот регистр. По факту записи будет выполнена начальная установка автомата синхронного ввода-вывода. После записи в этот регистр следует выждать несколько милисекунд перед записью в GO_SYNC_IO
0x112	ASYNC_OUT	W	Асинхронный вывод на цифровые линии или на один из каналов ЦАП. Биты приведены в Таблице 3.18
0x114	LED	W	Управление красным цветом светодиода на передней панели модуля при остановленном синхронном вводе-выводе (0 — погашен, 1 — горит). При запущенном синхронном вводе-выводе светодиод всегда горит зеленым цветом.
0x116	DIGIN_PULLUP	W	Включение или отключение подтягивающих резисторов на цифровых входах. Биты приведены в Таблице 3.19
0x118	OUTSWAP_BFCTL	W	Подкачка отсчета на ЦАП и управления режимами SPORT0 для BlackFin. Биты приведены в Таблице 3.20

Таблица 3.15: Регистры блока дополнительной обработки ввода вывода (IO_ARITH)

Адрес	Регистр	Доступ	Описание
0x0	ADC_COEF_B10	W	Коэффициент смещения нуля для диапазона ± 10 В
0x1	ADC_COEF_B5	W	Коэффициент смещения нуля для диапазона ± 5 В
0x2	ADC_COEF_B2	W	Коэффициент смещения нуля для диапазона ± 2 В
0x3	ADC_COEF_B1	W	Коэффициент смещения нуля для диапазона ± 1 В
0x4	ADC_COEF_B05	W	Коэффициент смещения нуля для диапазона ± 0.5 В
0x5	ADC_COEF_B02	W	Коэффициент смещения нуля для диапазона ± 0.2 В
0x8	ADC_COEF_K10	W	Коэффициент шкалы нуля для диапазона ± 10 В
0x9	ADC_COEF_K5	W	Коэффициент шкалы нуля для диапазона ± 5 В
0xA	ADC_COEF_K2	W	Коэффициент шкалы нуля для диапазона ± 2 В
0xB	ADC_COEF_K1	W	Коэффициент шкалы нуля для диапазона ± 1 В
0xC	ADC_COEF_K05	W	Коэффициент шкалы нуля для диапазона ± 0.5 В
0xD	ADC_COEF_K02	W	Коэффициент шкалы нуля для диапазона ± 0.2 В
0x12	ADC_FREQ_DIV	W	Должно быть записано тоже значение, что и в аналогичный регистр управления вводом-выводом
0x19	IN_STREAM_ENABLE	W	Разрешение синхронных потоков на ввод (бит 0 разрешает ввод с АЦП, бит 1 — с цифровых линий)
0x1A	DIN_ASYNC	W	В данном регистре сохраняется последнее введенное значение с цифровых линий при запущенном синхронном вводе (GO_SYNC_IO = 1) не зависимо от разрешения в IN_STREAM_ENABLE. Позволяет эмулировать асинхронный ввод. Биты описаны в Таблице 3.21

Таблица 3.16: Формат настроек логического канала АЦП

Биты	Обозначение	Доступ	Описание
2-0	LCH_RANGE	RW	Диапазон измерения для заданного канала: • ’0’ — ±10 В • ’1’ — ±5 В • ’2’ — ±2 В • ’3’ — ±1 В • ’4’ — ±0.5 В • ’5’ — ±0.2 В • ’6’, ’7’ — резерв
6-3	LCH_CHAN_NUM	RW	Номер физического канала: 0 - 1-ый (17-ый) канал, 15 - 16-ый (32-ой) канал
8-7	LCH_CHAN_MODE	RW	Режим измерения: • ’0’ — дифференциальный • ’1’ — первые 16 каналов с общей землей • ’2’ — вторые 16 каналов с общей землей • ’3’ — собственный ноль
15-9	LCH_AVG	RW	Количество отсчетов для усреднения равно LCH_AVG + 1
31-16	-	-	Резерв

Таблица 3.17: Биты регистра IO_MODE

Биты	Обозначение	Доступ	Описание
2-0	SYNC_MODE	RW	Источник опорной частоты синхронизации • ’0’ — внутренний генератор • ’1’ — от соседнего модуля (через разъем синхронизации) • ’2’ — по фронту сигнала SYN1 • ’3’ — по фронту сигнала SYN2 • ’6’ — по спаду сигнала SYN1 • ’7’ — по спаду сигнала SYN2 • ’4’, ’5’ — резерв
6-3	START_MODE	RW	Источник события запуска синхронного ввода-вывода • ’0’ — старт при записи 1 в регистр GO_SYNC_IO • ’1’ — от соседнего модуля (через разъем синхронизации) • ’2’ — по фронту сигнала SYN1 • ’3’ — по фронту сигнала SYN2 • ’6’ — по спаду сигнала SYN1 • ’7’ — по спаду сигнала SYN2 • ’4’, ’5’, ’8’-’15’ — резерв
8-7	REF_FREQ	RW	Значение опорной частоты синхронизации (при использовании внутреннего генератора) • ’0’ — 2 МГц • ’2’ — 1,5 МГц • ’1’, ’3’ — резерв

Таблица 3.17: Биты регистра IO_MODE

Биты	Обозначение	Доступ	Описание
9	DAC_FREQ	RW	Деление частоты вывода на ЦАП (относительно опорной частоты синхронизации) • ’0’ — не делится (корректно только для внешней опорной частоты меньше 700 кГц) • ’1’ — делится на 2 (режим по-умолчанию)
29-10	-	-	Резерв
30	SLV_CLK_FAIL	R	Признак, что был момент когда частота генератора PLDA не была захвачена (сбрасывается при чтении)
31	SLV_CLK_LOCK	R	Признак, захвачена ли частота генератора PLDA в данный момент

Таблица 3.18: Биты регистра ASYNC_OUT

Биты	Обозначение	Доступ	Описание
31-30	ASYNC_OUT_TYPE	W	Определяет, куда асинхронно будет выведено записанное значение • ’0’ — вывод на DIGOUT • ’1’ — вывод на 1-ый канал ЦАП • ’2’ — вывод На 2-ой канал ЦАП • ’3’ — резерв
29-18	-	-	Резерв
17	ASYNC_DOUT_Z_H	W	Перевод старшей половины выходов в третье состояние при выводе на DOUT
16	ASYNC_DOUT_Z_L	W	Перевод младшей половины выходов в третье состояние при выводе на DOUT
15-0	ASYNC_OUT_DATA	W	Код ЦАП, соответствующий выставляемому значению, или значение цифровых линий

Таблица 3.19: Биты регистра DIGIN_PULLUP

Биты	Обозначение	Доступ	Описание
0	PULLUPS_DI_H	W	Включение подтягивающих резисторов для старшей половины цифровых входов
1	PULLUPS_DI_L	W	Включение подтягивающих резисторов для младшей половины цифровых входов
2	PULLUPS_DI_SYN1	W	Включение подтягивающего резистора для входа SYN1
3	PULLUPS_DI_SYN2	W	Включение подтягивающего резистора для входа SYN2
31-4	-	-	Резерв

Таблица 3.20: Биты регистра OUTSWAP_BFCTL

Биты	Обозначение	Доступ	Описание
0	OUT_SWAP	W	При записи 1 в этот бит, выполняется подкачка одного значения на вывод из буфера блока DMA или из сигнального процессора по SPORT0 в ПЛИС. Должна выполнятся перед запуском синхронного вывода, чтобы первое значение появилось одновременно с признаком запуска синхронного ввода-вывода
1	BF_TFS_EN	W	Разрешение генерации ПЛИС сигналов TFS0. Должен устанавливаться в 1 перед запуском синхронного вывода в режиме работы с DSP. Установка в 0 также приводит к сбросу схемы вывода и очистки буферов и должна выполняться между запусками синхронного сбора. При работе без DSP данный бит всегда должен быть равен 0
2	SPORT0_RING_TEST	W	Включение тестового кольцевого режима для SPORT0, при котором слова переданные в ПЛИС по SPORT0 возвращаются обратно по SPORT0 на ввод. Используется исключительно для проверки работы интерфейса SPORT0. При штатной работе должен быть равен 0
3	BF_RFS_EN	W	Разрешение генерации ПЛИС сигналов RFS0. Должен устанавливаться в 1 перед запуском синхронного ввода в режиме работы с DSP. Установка в 0 также приводит к сбросу буферов на ввод и должна выполняться между запусками синхронного ввода. При работе без DSP данный бит всегда должен быть равен 0
31-4	-	-	Резерв

Таблица 3.21: Биты регистра DIN_ASYNC

Биты	Обозначение	Доступ	Описание
15-0	ASYNC_DI_DATA	R	Сохраненное последнее значение линий цифрового ввода (DI1-DI16)
16	ASYNC_DI_SYN1	R	Сохраненное последнее значение линии SYN1
17	ASYNC_DI_SYN2	R	Сохраненное последнее значение линии SYN2
30-18	-	-	Резерв
31	ASYNC_DI_RDY	R	Признак, обновлялись ли данные с момента последнего считывания из регистра DIN_ASYNC

3.4.4 Блок регистров для доступа к Flash-памяти

3.4.4.1 Общее описание

Данный блок регистров предоставляет доступ к внутренней Flash-паяти модуля SST25VF16B. Полный список регистров приведен в Таблице 3.22. Большинство операций чтения/записи в указанные регистры приводит к выполнению циклов на SPI-интерфейсе для выполнения команд Flash-памяти.

Кроме доступа к Flash-памяти, данный блок содержит регистр HARD_ID, определяющий аппаратные версии и наличие опций для данного модуля.

Пользователю доступна на запись только младшая половина Flash-памяти, размером 1Mb (адреса 0-0xFFFFF), которую пользователь может использовать по своему усмотрению. Старшая половина (адреса 0-0x1FFFFF) защищена и доступна только на чтение. В последнем блоке памяти размером 64КБайт (адреса 0x1EFFFF-0xFFFFF) хранится информация о данном экземпляре (серийный номер, калибровочные коэффициенты). Остальная область используется под резервную и основную копию прошивки ПЛИС (и как правило не должна интересовать пользователя).

Следует также отметить, что по-умолчанию пользовательская область защищена от записи. Выключить защиту можно через статусный регистр памяти EEPROM_RD_STATUS. Кроме того, каждой операция записи или стирания должна предшествовать команда разрешения доступа к памяти (вызывается путем записи в EEPROM_WR_EN). Запись при этом может выполняться реально только в предварительно стертую область.

Типичный алгоритм при изменении содержимого Flash-памяти выглядит следующим образом:

- Снять защиту от записи в пользовательскую область:
  - Разрешить запись в регистр статуса записью в EEPROM_WR_STATUS_EN.
  - Записать в регистр EEPROM_WR_STATUS значение бита EEPROM_STATUS_BP1 = 0.
- Стереть нужную область с помощью одного или нескольких циклов стирания (в зависимости от размера области):
  - Разрешение доступа с помощью записи в регистр EEPROM_WR_EN.
  - Выполнение команды стирания сектора по заданному адресу. Стереть можно либо сектор размером 4Кбайт (запись в регистр EEPROM_ERASE_4K) или 64Кбайт (EEPROM_ERASE_64K).
  - Чтение статусного регистра через чтение EEPROM_RD_STATUS и ожидание, пока бит EEPROM_STATUS_BUSY не станет равным 0.
- Записать данные по одному байту, выполнив нужное количество следующих циклов:
  - Разрешение доступа с помощью записи в регистр EEPROM_WR_EN.
  - Выполнение команды записи байта путем записи нужного адреса и данных в регистр EEPROM_WR_BYTE.
  - Чтение статусного регистра через чтение EEPROM_RD_STATUS и ожидание, пока бит EEPROM_STATUS_BUSY не станет равным 0.

- Снова установить защиту от записи в пользовательскую область (также как и снималась, но EEPROM_STATUS_BP1 = 1)

Чтение может выполняться из любой области в любой момент. Для чтение 32-битного слова нужно выполнить следующие действия:

- записать адрес слова в биты 31-8 регистра EEPROM_SET_RD_ADDR
- прочитать содержимое регистра EEPROM_RD_DWORD. Считанное значение будут соответствовать слову во Flash-памяти.

3.4.4.2 Список регистров

Таблица 3.22: Регистры доступа к Flash-памяти

Адрес	Регистр	Доступ	Описание
0	EEPROM_SET_RD_ADDR	W	Биты 31-8 задают адрес во Flash-памяти при чтении. Адрес должен быть записан до чтения из регистра EEPROM_RD_DWORD. Биты 7-0 не используются
1	EEPROM_RD_DWORD	R	При чтении этого регистра реально выполняется цикл чтения из Flash-памяти по адресу, записанному в регистр EEPROM_SET_RD_ADDR. Прочитанное из памяти значение, возвращается как значение регистра.
2	EEPROM_RD_STATUS	R	Чтение из этого регистра приводит к циклу чтения статуса Flash-памяти. В результате статус Flash-памяти возвращается в битах 31-24 прочитанного слова (формат статуса приведен в Таблице **??**). Биты 23-0 не используются.
3	EEPROM_WR_STATUS_EN	W	Запись в этот регистр разрешает последующую однократную запись в статусный регистр и должна выполняться каждый раз перед записью в EEPROM_WR_STATUS.
4	EEPROM_WR_EN	W	Запись в этот регистр разрешает последующую операцию записи или стирания. Должна выполняться перед каждой записью в регистры EEPROM_WR_BYTE, EEPROM_ERASE_4K и EEPROM_ERASE_64K. После выполнения записи в один из указанных регистров, запись во Flash-память автоматически запрещается.
5	EEPROM_WR_DIS	W	Запись в этот регистр вручную запрещает операцию записи во Flash-память.

Таблица 3.22: Регистры доступа к Flash-памяти

Адрес	Регистр	Доступ	Описание
6	EEPROM_WR_STATUS	W	При записи в этот регистр, младшие 8 бит записанного слова записываются в статусный регистр Flash-памяти. Для того, чтобы эта запись имела эффект, нужно непосредственно перед этим сделать запись в регистр EEPROM_WR_STATUS_EN
7	EEPROM_ERASE_4K	W	Запись в этот регистр запускает операцию стирания по адресу 4K*(значение в битах 31-20). Перед выполнением данной команды должна быть произведена запись в EEPROM_WR_EN. Для ожидания завершения записи, нужно дождаться, пока бит статуса EEPROM_STATUS_BUSY не станет равным 0
8	EEPROM_ERASE_64K	W	Запись в этот регистр запускает операцию стирания по адресу 64K*(значение в битах 31-23). Перед выполнением данной команды должна быть произведена запись в EEPROM_WR_EN. Для ожидания завершения записи, нужно дождаться, пока бит статуса EEPROM_STATUS_BUSY не станет равным 0
9	EEPROM_WR_BYTE	W	Запись в этот регистр запускает операцию записи байта, заданного в битах 7-0, по адресу, указанному в битах 31-8. Перед выполнением данной команды должна быть произведена запись в EEPROM_WR_EN. Для ожидания завершения записи, нужно дождаться, пока бит статуса EEPROM_STATUS_BUSY не станет равным 0
10	HARD_ID	R	В этом регистре содержится информация о аппаратных версиях и наличии опций. Значения битов приведены в Таблице 3.24
11	EEPROM_JEDEC_ID	R	Чтение этого регистра вызывает цикл чтения идентификатора микросхемы Flash-памяти. Должно быть прочитано значение 0x4125BFXX при исправной Flash-памяти

Таблица 3.23: Биты статусного регистра Flash-памяти

Биты	Обозначение	Доступ	Описание
0	EEPROM_STATUS_BUSY	R	Бит служит для определения завершения операций записи или стирания. Если бит равен 1, значит в данный момент выполняется операция записи/стирания.
1	EEPROM_STATUS_WEL	R	Указывает, разрешена ли операция записи во Flash-память. Разрешение выполняется записью в регистр EEPROM_WR_EN
2,3,4	EEPROM_STATUS_BP0, EEPROM_STATUS_BP1, EEPROM_STATUS_BP2	RW	Биты защиты памяти. Определяют какая область защищена от записи (все команды стирания/записи для этих областей игнорируются). Пользователю доступно изменение только бита EEPROM_STATUS_BP1. Если этот бит установлен в 1, то младший 1Mb Flash-памяти защищен от записи, иначе - запись разрешена. Старшая половина памяти всегда защищена от записи.
5	EEPROM_STATUS_BP3	RW	Бит не используется
6	EEPROM_STATUS_AAI	R	Признак, что память находится в режиме автоинкремента адреса (реально этот режим не используется в L502)
7	EEPROM_STATUS_BPL	RW	Запрет изменения битов EEPROM_STATUS_BP0, EEPROM_STATUS_BP1, EEPROM_STATUS_BP2, EEPROM_STATUS_BP3. Если установлен в 1, значения указанных битов реально не изменяются, независимо от записываемого в регистр кода

Таблица 3.24: Биты регистра HARD_ID

Биты	Обозначение	Описание
0	HARD_ID_DAC_PRESENT	Признак наличия ЦАП
1	HARD_ID_GAL_PRESENT	Признак наличия гальваноразвязки
2	HARD_ID_BF_PRESENT	Признак наличия сигнального процессора
3	-	Резерв
7-4	HARD_ID_PLDA_VER	Версия прошивки PLDA
11-8	HARD_ID_BOARD_REV	Номер ревизии платы
15-12	-	Резерв
31-16	HARD_ID_FPGA_VER	Версия прошивки ПЛИС. Младшие 8 бит задают минорную часть, старшие 8 бит — мажорную

3.4.4.3 Формат информации о модуле

В данном разделе приводится формат информации о модуле, сохраненной во Flash-памяти, начиная с адреса 0x1EFFFF. В начале идет фиксированный заголовок размером 128 байт (приведен в Таблице 3.25), затем информация о калибровочных коэффициентах АЦП и ЦАП.

Таблица 3.25: Поля фиксированного заголовка информации о модуле

Смещение (байты)	Размер	Обозначение	Описание
0	4	sign	Признак правильной информации. Должен быть равен 0x4C524F4D (LROM)
4	4	size	Размер всей полезной информации, включая CRC
8	4	format	Целое число, задающее версию формата. Должно быть равно 1
12	32	name	Название устройства (строка “L502” без кавычек)
44	32	serial	Серийный номер устройства (ASCII строка, оканчивающаяся нулем)
76	52	res	Резерв

Последние 4 байта всей информации (по смещению size-4) занимает CRC32, вычисленной по всем size-4 байтам информации. CRC32 можно использовать для проверки целостности записанной информации.

Между фиксированным заголовком и CRC32 может находится произвольное количество дополнительных заголовков. Каждый дополнительный заголовок начинается с двух полей, приведенных в Таблице 3.26.

Таблица 3.26: Общие начальные поля для всех дополнительных заголовков

Смещение (байты)	Размер	Обозначение	Описание
0	4	sign	Код заголовка
4	4	size	Размер заголовка в байтах

В L502 используются два дополнительных заголовка — один для хранения коэффициентов АЦП и один — для коэффициентов ЦАП. Оба имеют одинаковый формат, приведенный в Таблице 3.27.

Следует иметь ввиду, что при разборе следует быть готовым к тому, что в будущем при необходимости могут быть добавлены новые заголовки после перечисленных.

Вы можете посмотреть описания форматов в файле `l502_eeprom.h` исходников l502api.

Таблица 3.27: Дополнительный заголовок для хранения калибровочных коэффициентов

Смещение (байты)	Размер	Обозначение	Описание
0	4	cbr_sign	Код заголовка с калибровочными коэффициентами равен 0x4C434352 (LCBR)
4	4	cbr_size	Размер заголовка в байтах
8	4	format	Версия формата (должна быть равна 2)
12	4	src	Определяет, предназначены эти коэффициенты для АЦП (1) или ЦАП (2)
16	4	flags	Флаги (сейчас не используются)
20	12	reserv	Резерв
32	8	data	Время проведения калибровки (в Unix-time формате)
40	4	channel_cnt	Количество каналов, для которых сохранены коэффициенты. Для АЦП — 1, так как сохраняется один экземпляр коэффициентов на все каналы. Для ЦАП — 2, так как коэффициенты для каждого канала разные.
44	4	range_cnt	Количество диапазонов (6 для АЦП, 1 для ЦАП)
48	8	offs	Код смещения для 1-го диапазона 1-го канала (диапазоны идут последовательно от ± 10В до ± 0.2В). Число с плавающей запятой (соответствует типу double в C)
56	8	k	Код коэффициента шкалы для 1-го диапазона 1-го канала. Число с плавающей запятой (соответствует типу double в C)
64	..	...	Далее последовательно идут коэффициенты offs и k для всех каналов и всех диапазонов

3.4.5 Блок регистров с отладочной информацией

Данный блок регистров используется как правило в отладочных целях.

Таблица 3.28: Регистры блока с отладочной информацией

Адрес	Регистр	Доступ	Описание
0	DBG_EVENTS	RW	Данный регистр фиксирует факт возникновения отладочных событий. Каждый бит отвечает за свое событие. Запись 1 в бит сбрасывает его значение
8	DBG_LAST_ABORT_ADDR	R	Адрес регистра ПЛИС, обращение к которому привело к тому, что не было ответа в течении таймаута и обращение было прервано. Действителен, только если бит DBG_EVENTS_ABORT регистра DBG_EVENTS установлен в 1.
9	DBG_LAST_NACK_ADDR	R	Адрес регистра ПЛИС, обращение к которому вызвало ответ Nack (нет подтверждения). Действителен, только если бит DBG_EVENTS_NACK регистра DBG_EVENTS установлен в 1.
10	DBG_LINK_REPLAY_CNT	R	Данный регистр фиксирует количество ошибок Link-уровня PCI-Express. В младшей половине сохраняется количество повторных передач, а в старшей - количество ошибок приема. Чтение из регистра сбрасывает значение счетчиков. При достижении счетчиком максимального значение (0xFFFF) счетчик дольше не увеличивается до сброса.

Таблица 3.29: Биты регистра DBG_EVENTS

Биты	Обозначение	Описание
0	DBG_EVENTS_ABORT	Признак, что было обращение к регистрам ПЛИС, которое было оборвано по таймауту
1	DBG_EVENTS_NACK	Признак, что было обращение к регистрам ПЛИС, которое было завершено с отрицательным ответом Nack
31-2	-	Резерв

Глава 4

Режим работы с сигнальным процессором

Данный раздел предназначен для программистов, которые будут писать свою прошивку сигнального процессора BlackFin. Для таких программистов “L Card” предоставляет штатную прошивку, которая выполняет те же функции, что доступны в режиме без сигнального процессора. То есть в стандартной прошивке реализовано:

- Инициализация всей необходимой периферии
- Обработка стандартных команд переданных от ПК (настройка параметров сбора, запуск или останов потоков данных и т.п.)
- Прием потока данных АЦП/DIN от ПЛИС по SPORT0 с последующей передачей этих же данных по HDMA в ПЛИС на запись в память ПК.
- Прием по HDMA потока данных ЦАП/DOUT и передача этих же данных в ПЛИС по SPORT0 для вывода.

Дальнейшее описание будет основана на штатной прошивке и работа сигнального процессора будет описана на основе реализованных функций.

Исходный код прошивки входит в состав "L-Card PCI Express SDK") (устанавливается в папку firmware, если был включена установка файлов для разработчика), а также доступен в виде репозитория на bitbucket.org или его можно взять из архива исходных кодов SDK.

В штатной прошивке есть файл l502_user_process.c, в котором реализованы функции, вызываемые при приеме или завершении передачи блока данных, а также при приходе пользовательских команд от ПК. Подразумевается, что в простейшем случае пользователю будет достаточно изменить именно этот файл при добавлении своей обработки, если общая концепция штатной прошивки удовлетворяет задаче. По крайней мере именно такой вариант изменения следует рассмотреть в первую очередь.

Сама штатная прошивка может быть собрана как в среде **VisualDSP**, так и с помощью бесплатного компилятора **GCC** (с использованием GNU Toolchain для BlackFin).

Информацию о том, как работает периферия сигнального процессора ADSP-BF523, можно узнать из Hardware Manual. Данное описание предполагает, что читатель знаком с информацией, изложенной в указанном документе.

4.1 Загрузка и запуск прошивки BlackFin

При включении питания компьютера сигнальный процессор находится в сброшенном состоянии (сброс процессора после подачи питания необходим, чтобы перевести его в известное начальное состояние), а модуль работает в режиме работы без сигнального процессора.

После загрузки драйвера модуля L502 (на одной из стадий загрузки операционной системы), драйвер сразу снимает сигнал сброса в обнаруженных модулях с помощью бита BF_RST регистра BF_CTL. После снятия сигнала сброса, начинает выполняться внутренний загрузчик BlackFin, который ожидает прошивку по интерфейсу HostDMA. Также после снятия сброса процессор становится доступен для управления по интерфейсу JTAG.

При этом требуемые настройки частоты клока сигнального процессора и памяти SDRAM хранятся во внутренней специальной памяти (OTP-память). Таким образом, встроенный загрузчик уже инициализирует SDRAM и частоту клока (используется частота ядра 530 МГц и частота периферии SCLK = 132.5 МГц). Это позволяет при необходимости загружать прошивку сразу в SDRAM как по HostDMA, так и через JTAG, без специального инициализационного кода.

Следует также отметить, что до загрузки прошивки интерфейс HostDMA инициализирован только на запись, а инициализация на чтение происходит уже при запуске самой прошивки.

После того, как сигнал сброса снят, необходимо загрузить прошивку сигнального процессора и перевести модуль L502 в DSP-режим.

Загрузка прошивки может быть осуществлена двумя способами:

- Штатным образом — из .ldr файла с компьютера по интерфейсу HDMA
- Отладочный режим — загрузка программы через JTAG-интерфейс для последующей отладки.

При использовании штатной библиотеки l502api для загрузки прошивки с компьютера достаточно установить связь с устройством и вызвать функцию `L502_BfLoadFirmware()`, которая загружает прошивку, проверяет ее работу с помощью команды L502_BF_CMD_CODE_GET_PARAM с параметром L502_BF_PARAM_FIRM_VERSION (подробнее смотри раздел Командный интерфейс между ПК и сигнальным процессором), передает прошивке необходимую информацию о модуле, после чего переводит модуль в режим работы с DSP.

Для загрузки через JTAG, необходимо подключить JTAG-эмулятор для процессора BlackFin к специальному разъему на плате (смотри "Руководство пользователя"). После этого следует загрузить свой проект по JTAG интерфейсу через эмулятор из среды программирования (описано в следующих двух разделах). Когда прошивка будет загружена и запущена, нужно явно перевести модуль L502 в режим работы с DSP. При использовании прошивки, которая совместима с штатной в плане стандартных команд, для этого можно использовать функцию `L502_BfCheckFirmwareIsLoaded()`, которая проверяет загружена ли прошивка с помощью команд для получения информации о состоянии прошивки, передает прошивке информацию о модуле и переводит модуль в DSP-режим. Чтобы просто перевести модуль в DSP-режим без каких-либо дополнительных действий, можно воспользоваться функцией `L502_SetMode(L502_MODE_DSP)`.

4.2 Сборка и отладка проекта в VisualDSP

В первую очередь необходимо скачать и установить среду VisualDSP 5.0. Оценочную бесплатную версию можно скачать с официального сайта Analog Devices. Крайне рекомендуется устанавливать последние обновления (на момент написания документа - 10.1). Все необходимые драйвера для JTAG-эмулятора (если Вы используете эмулятор от Analog Devices) уже устанавливаются вместе со средой.

Все файлы исходников прошивки, специфичные для сборки в среде VisualDSP, лежат в директории l502-bf/vdsp, включая проект для этой среды — l502-bf.dpj.

Для сборки после установки среды отрываем файл проекта.

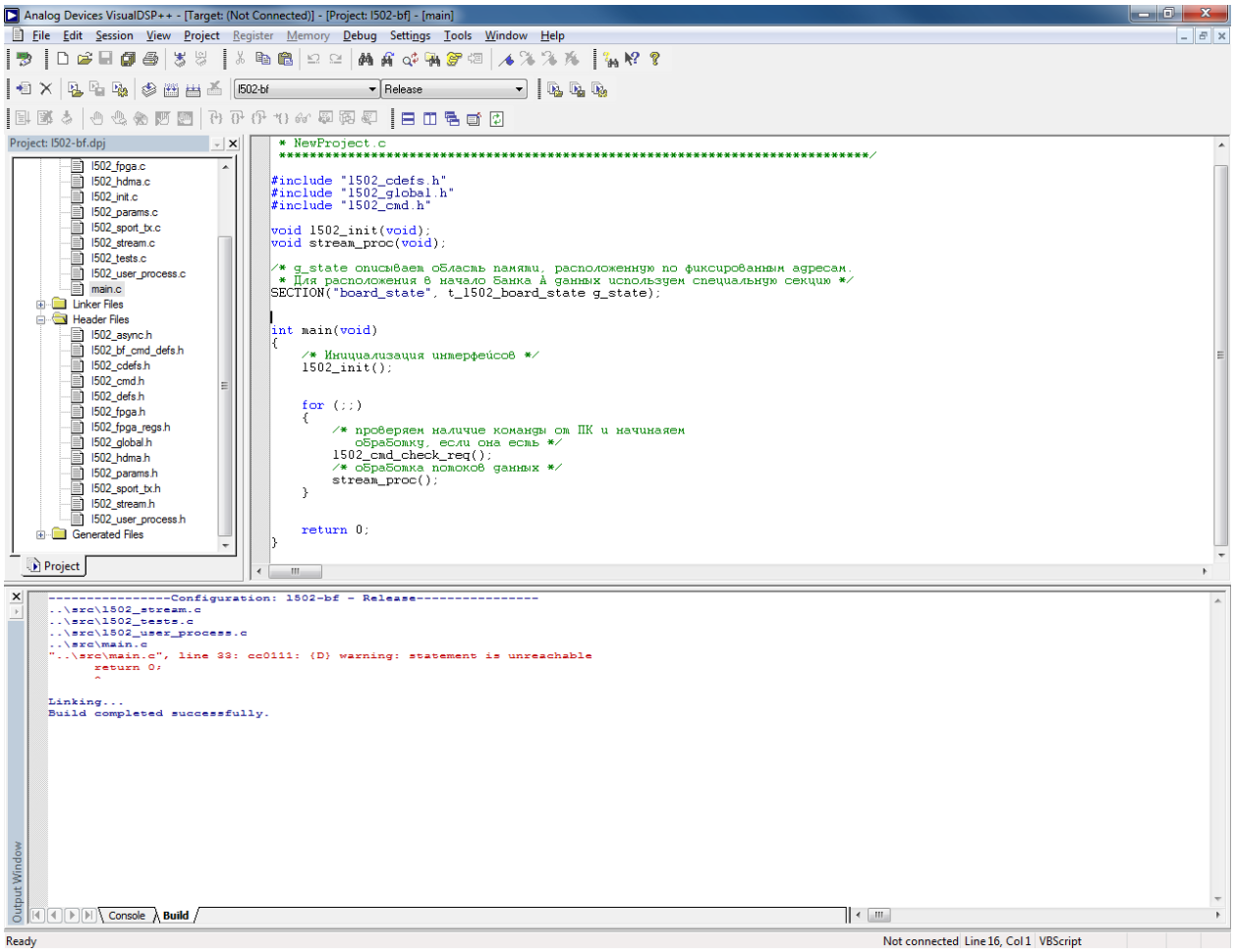

Рис. 4.1: Проект прошивки в VisualDSP

В проекте есть две конфигурации - Debug (с отключенной оптимизацией — для отладки) и Release (с оптимизацией для финальной версии). Для отладки через JTAG нужно скомпилировать исходный код в файл .dxe, для чего в "Project->Project Options..." нужно выбрать "Type: Executable". Чтобы получить файл .ldr для загрузки по HostDMA через `L502_BfLoadFirmware()` нужно будет выбрать "Type: Loader Type".

Проект может быть собран через "Project->Build Project".

Для отладки необходимо создать сессию через "Session->New Session...".

Выбираем процессор ADSP-BF523, на следующей вкладке выбираем Emulator, далее выбираем нужный JTAG-Эмулятор. Работа проверялась на ADI ICE 100B.

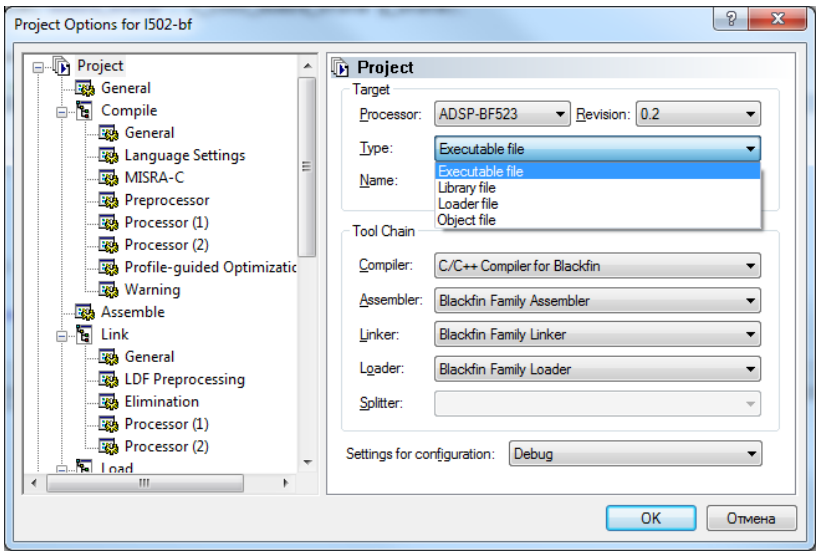

Рис. 4.2: Настройка типа проекта

После этого, если все подключено и настроено правильно, прошивка будет загружена в сигнальный процессор и остановлена в начале главной функции main() для дальнейшей отладки.

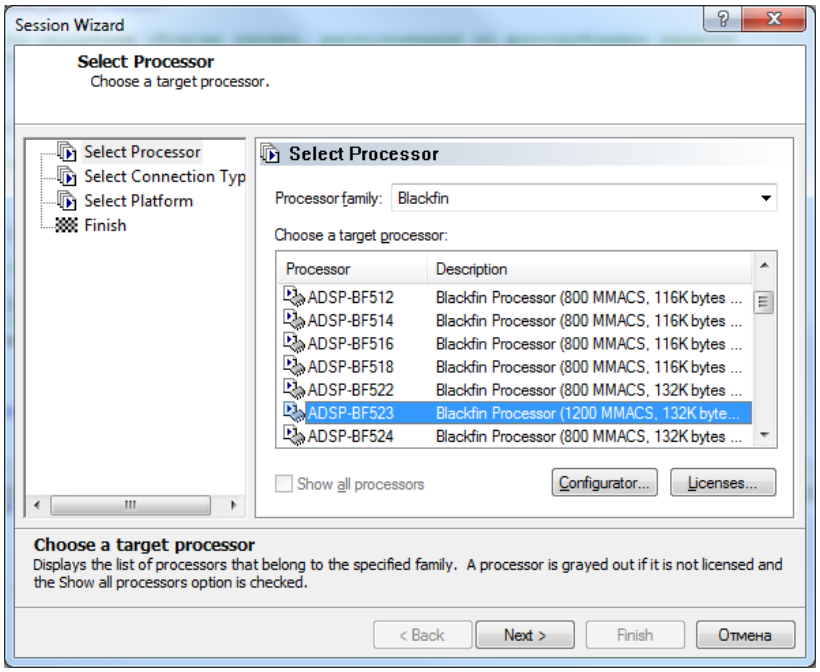

Рис. 4.3: Выбор процессора для отладочной сессии

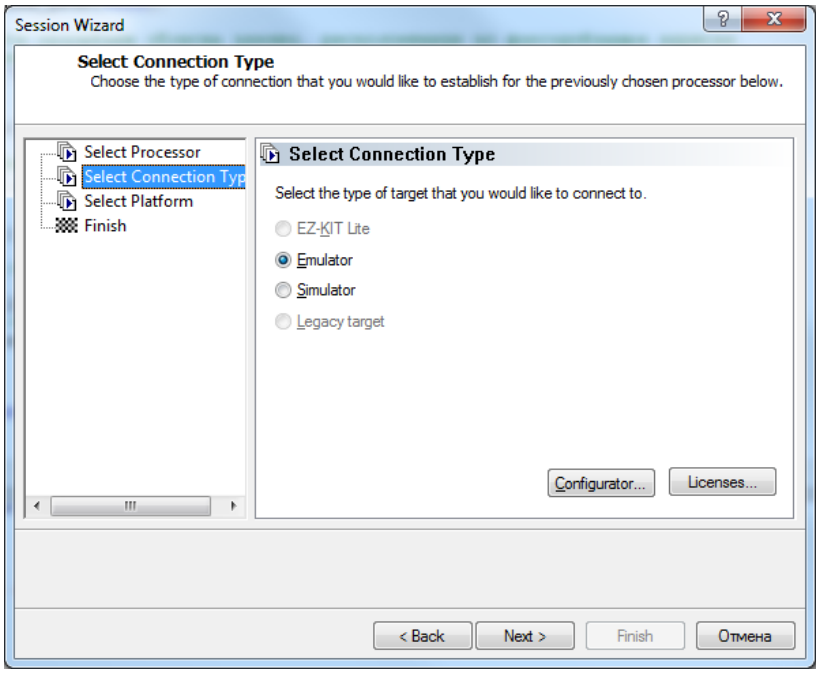

Рис. 4.4: Выбор типа подключения отладочной сессии

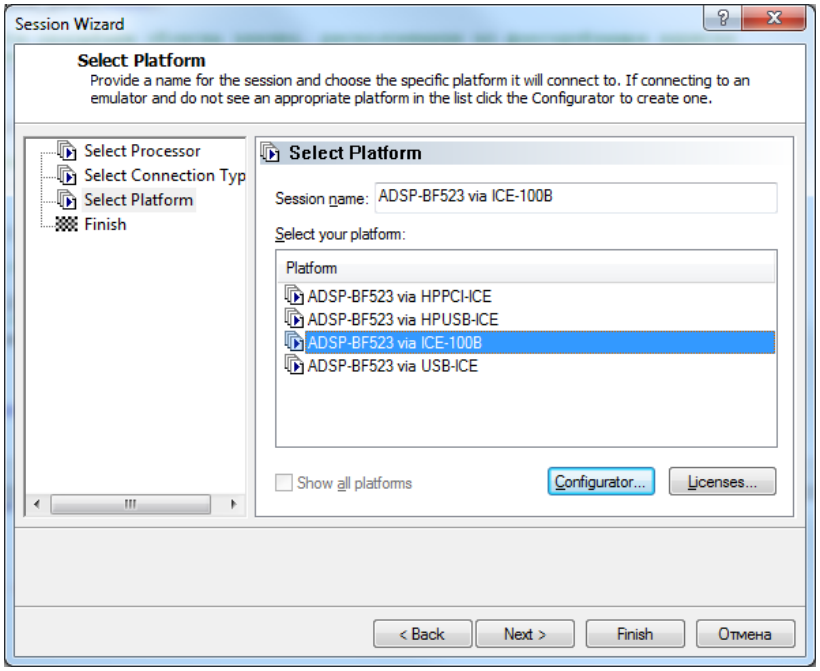

Рис. 4.5: Выбор JTAG-эмулятора отладочной сессии

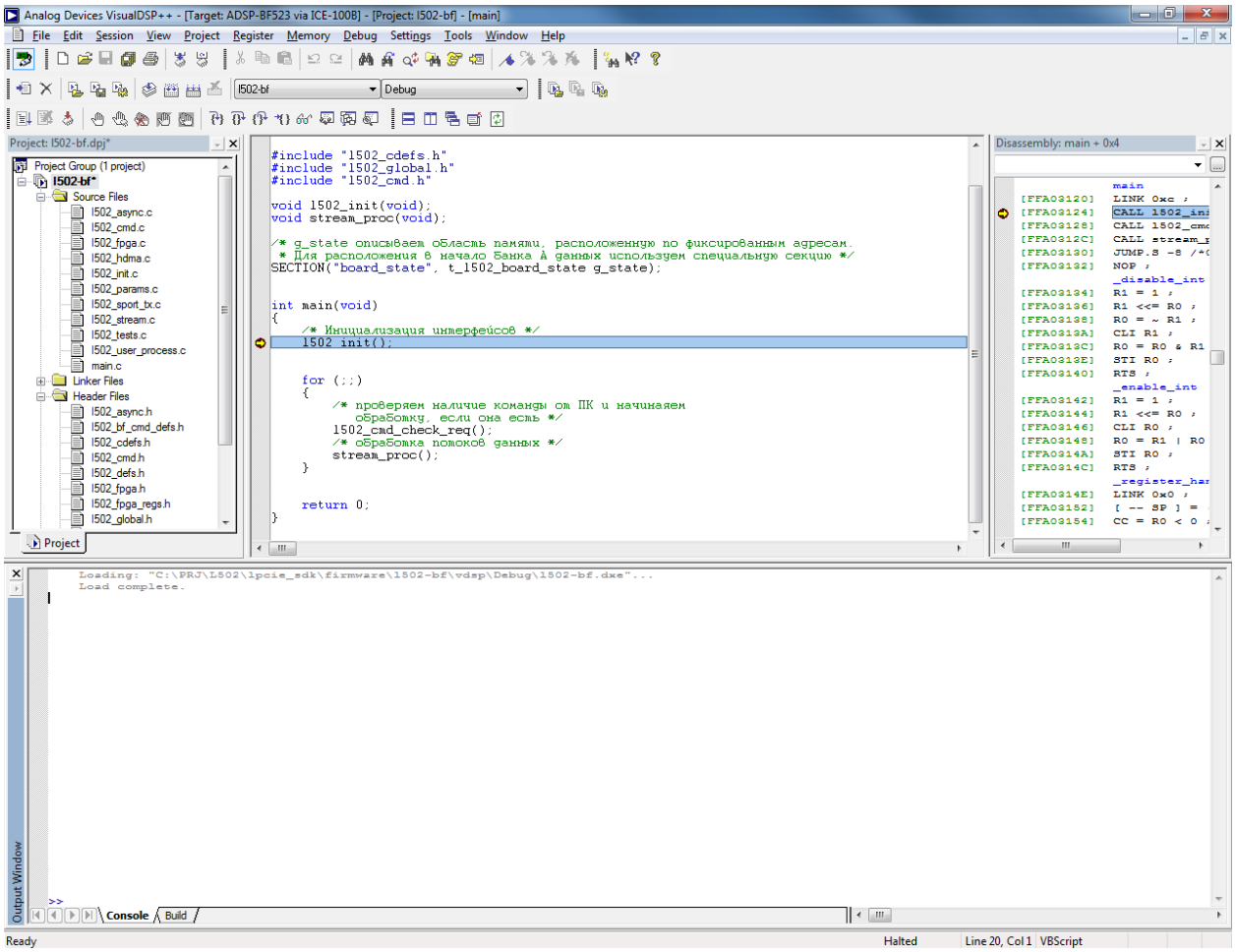

Рис. 4.6: Отладка прошивки в VisualDSP

4.3 Сборка и отладка проекта с использованием компилятора gcc и среды Eclipse

Информацию о использовании открытого программного обеспечения (в частности компилятора GCC) для написания программ для сигнального процессора BlackFin можно получить на сайте http://docs.blackfin.uclinux.org/doku.php. В частности, наибольший интерес представляет раздел Bare Metal Toolchain, описывающий разработку программ под BlackFin без использования операционной системы на сигнальном процессоре.

4.3.1 Сборка проекта

Для начала необходимо скачать и установить необходимый набор инструментов (toolchain) с http://blackfin.uclinux.org/gf/project/toolchain/frs. Для сборки программ без использования ОС нужен toolchain **bfin-elf** . После установки необходимо добавить путь к bfin-elf/bin в переменную окружения PATH.

Также необходима утилита make для сборки проекта по файлу makefile. В Linux ее можно установить из репозитория дистрибутива, а для Windows можно скачать отсюда: http://gnuwin32.sourceforge.net/packages/make.htm.

В исходниках штатной прошивки для L502 в директории gcc хранятся все файлы, специфичные именно для сборки с помощью компилятора gcc (скрипт линкера, startup-код и т.д.). В частности в нем хранится makefile, описывающий процедуру сборки проекта.

Для сборки необходимо зайти в директорию gcc и вызвать:

- для сборки отладочной версии (без оптимизации)

```
make CONFIG:=debug
```

- для сборки финальной версии (с оптимизацией -O2):

```
make CONFIG:=release
```

Результаты сборки будут в директории build/debug/bin или build/release/bin соответственно. Файл l502-bf.elf используется для отладки через интерфейс JTAG, а файл l502-bf.ldr для заливки через HostDMA с помощью `L502_BfLoadFirmware()` из l502api.

4.3.2 Отладка через JTAG из среды Eclipse

Для отладки в первую очередь Вам нужен JTAG-эмулятор, который поддерживается открытым инструментарием, так как не все JTAG-эмуляторы от Analog Devices поддерживаются. Список поддерживаемых JTAG-эмуляторов можно посмотреть в разделе Available JTAG solutions. Работоспособность отладки была проверена для JTAG-эмулятора ADI ICE 100B.

JTAG должен быть подключен к соответствующему разъему на плате L502 (см. "Руководство пользователя") и сигнальный процессор должен быть выведен из аппаратного сброса (см. раздел Загрузка и запуск прошивки BlackFin).

В ОС Linux может понадобится разрешить доступ к JTAG-эмулятору не root-пользователю, для чего необходимо добавить правило для udev. Для ADI ICE 100B

можно скопировать файл gcc/ice-100B.rules в /etc/udev/rules.d (после этого возможно понадобится отключить и подключить снова JTAG-эмулятор). Для других эмуляторов с USB интерфейсом Вам нужно в этом файле изменить параметры idVendor и idProduct на параметры Вашего USB-устройства.

В ОС Windows для отладки через JTAG с использованием открытого ПО Вам понадобится специальный драйвер, который устанавливается вместе с gnu-toolchain, а не тот, что устанавливается вместе с VisualDSP. Для работы с ADI ICE 100B Вам понадобится изменить файл gnICE-drivers/gnICE.inf и добавить в нем в секции в секции [Devices.NTX86] и [Devices.NTAMD64] строчку

```
"ICE-100B" = USB_Install, USB\VID_064b&PID_0225&REV_0100
```

После этого в диспетчере устройств нужно указать явно, что нужно установить драйвер из директории gnICE-drivers.

Следует иметь ввиду, что в прошивке 2.0.0.6 для ADI ICE 100B ввели возможность изменить PID USB-устройства с помощью джампера J2. Если он одет, то эмулятор будет иметь PID=0x1225, а если снят — то PID=0x0255 (по-умолчанию). Это позволяет использовать данный эмулятор на одном компьютере для отладки как в VisualDSP, так и с использованием GNU Toolchain без постоянной перестановки драйверов. Драйвера VisualDSP устанавливаются всегда на устройство с PID=0x0255 и, если в gnICE-drivers/gnICE.inf вместо указанной до этого строчки добавить

```
"ICE-100B" = USB_Install, USB\VID_064b&PID_0225&REV_0100
```

то при одетом джампере J2 можно будет отлаживаться через GNU Toolchain, а со снятым — через VisualDSP (отключая и снова подключая эмулятор по USB).

После того, как JTAG эмулятор подключен, нужно запустить `bfin-gdbproxy` из набора инструментов bfin-elf:

```
bfin-gdbproxy bfin --reset
```

Если вывод будет подобен приведенному ниже, значит доступ к сигнальному процессору через JTAG успешно получен:

```
Found USB cable: ICE-100B
ICE-100B firmware version is 2.0.7
IR length: 5
Chain length: 1
Device Id: 00100010011111100000000011001011 (0x227E00CB)
  Manufacturer: Analog Devices, Inc. (0x0CB)
  Part(0):      BF527 (0x27E0)
  Stepping:     2
  Filename:     /home/user/opt/bfin-elf/bin/../share/urjtag/analog/bf527/bf527
warning:   bfin: no board selected, BF527 is detected
notice:    bfin: jc: waiting on TCP port 2001
notice:    bfin: jc:  (you must connect GDB before using jtag console)
notice:    bfin-gdbproxy: waiting on TCP port 2000
```

Следует отметить, что gdbproxy не различает ADSP-BF523, ADSP-BF525 и ADSP--BF527 и всегда показывает, что видит BF527, как видно из лога выше.

Следующим шагом можно установить среду Eclipse. Работе со средой Eclipse при написании программ для BlackFin посвящена следующая страница http://docs.-blackfin.uclinux.org/doku.php?id=toolchain:eclipse. Вначале необходимо установить саму среду с плагином для разработки на C/C++ — Eclipse CDT. Скачать ее можно с официального сайта или, в случае ОС Linux, установить из репозитория дистрибутива.

Дальше можно нужно установить плагин для работы с BlackFin. Для этого запускаем Eclipse, выбираем "Help->Install New Software", в поле "Work with..." вводим адрес сайта с плагином: http://blackfin.uclinux.org/eclipse, жмем Enter. После соединения в списке плагинов должны появиться "BlackFin Debug" и "BlackFin GNU Toolchain" — выбираем оба и жмем Next, соглашаемся со всем предложенным пока не установим плагин, после чего Eclipse попросит перезагрузиться. Далее добавляем в Eclipse проект

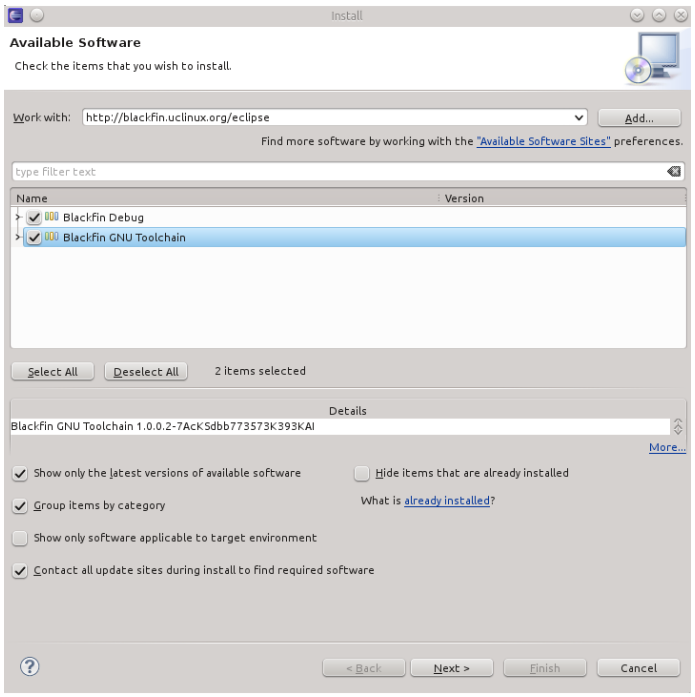

Рис. 4.7: Установка плагина для работы с BlackFin

прошивки с помощью меню "File->Import", выбираем "General-> Existing Projects into Workspace". Указываем путь к проекту прошивки и жмем Finish.

Теперь проект загружен в Eclipse. Открываем представление для редактирования кода (Windows->Open Perspective->Others... C/C++).

Теперь проект можно собирать из Eclipse выбрав проект в Project Explorer и нажав Project->Build Project (Ctrl + B). Следует отметить, что в проекте уже созданы две конфигурации - Debug и Release для сборки соответствующих версий проекта (изменить текущую можно через Project->Build Configurations->Set Active.

Для отладки выбираем Run->Debug Configuration... и в открывшемся окне выбираем "BlackFin GNU Toolchain Bare Metal Configuration (ELF) on JTAG". Должна быть подхвачена конфигурация "l502-bf Debug" с настройками из файла "gcc/l502-bf Debug.-launch".

Убеждаемся что bfin-gdbproxy запущен (его необходимо всегда держать запущенным при отладке) и жмем Debug, после чего выполнится загрузка прошивки и eclipse предложит перейти в Debug Perspective, в которой можно выполнять отладку.

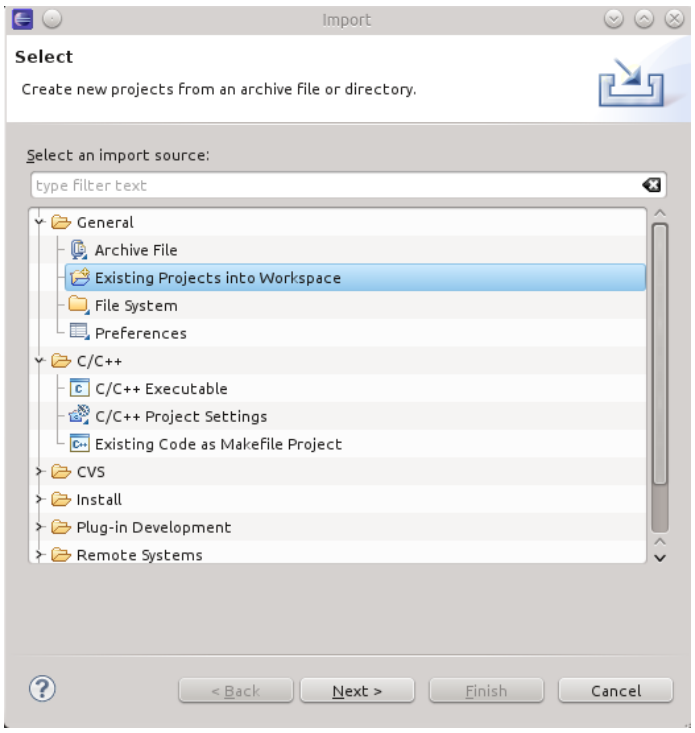

Рис. 4.8: Импорт существующего проекта

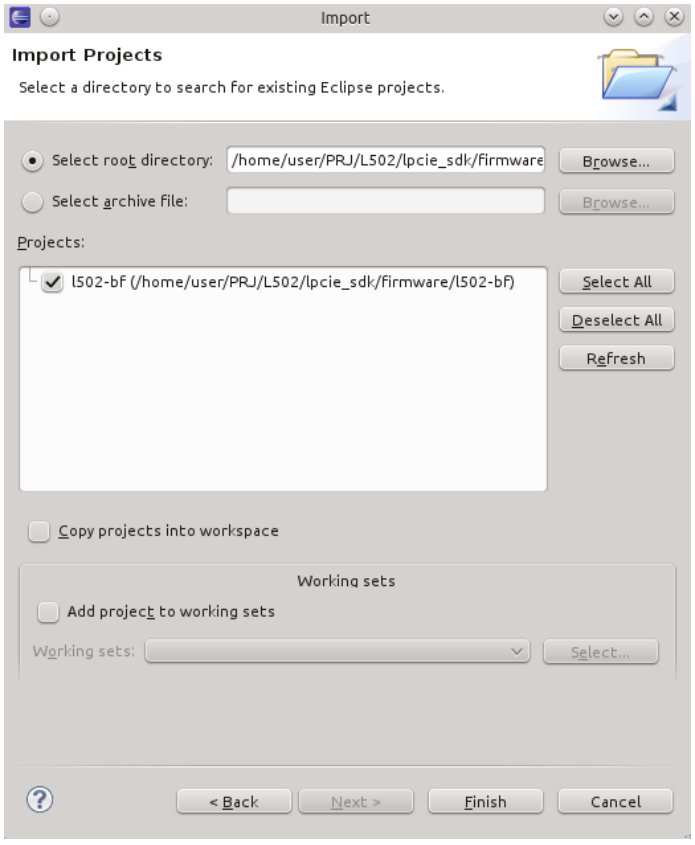

Рис. 4.9: Выбор проекта для импорта

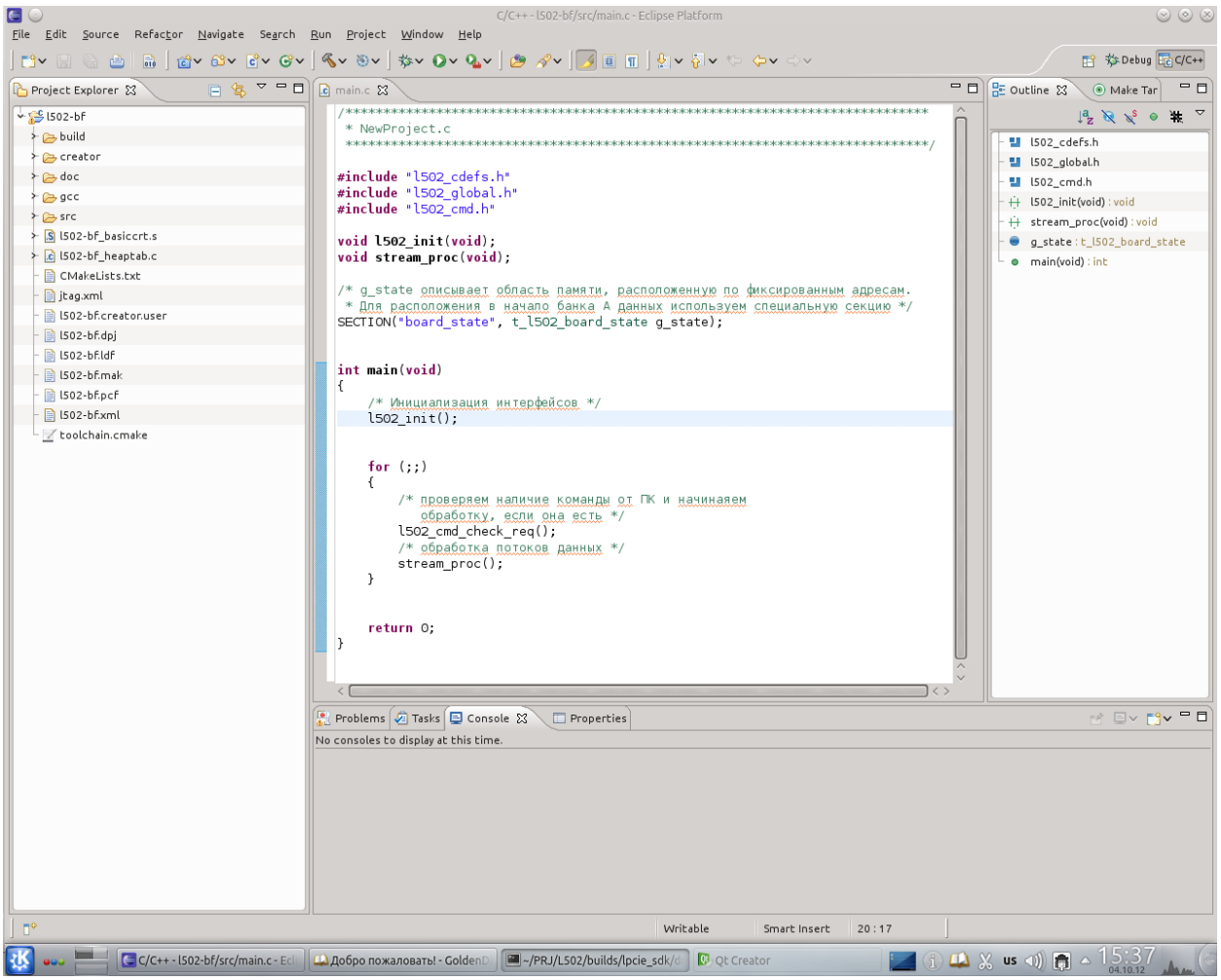

Рис. 4.10: Импортированный проект прошивке в C/C++ перспективе

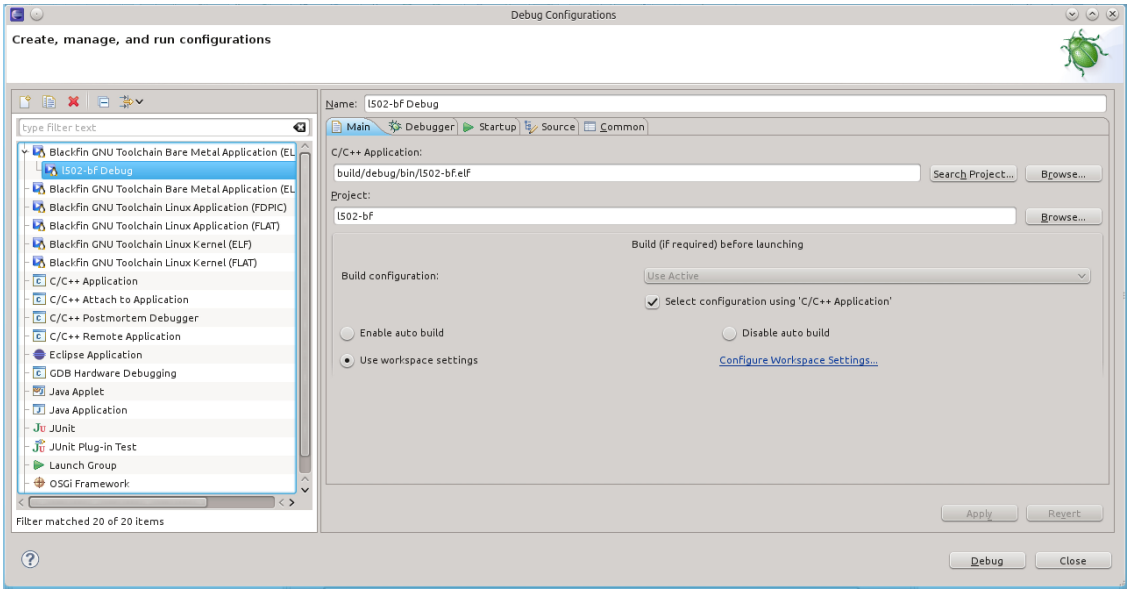

Рис. 4.11: Настройка сеанса отладки

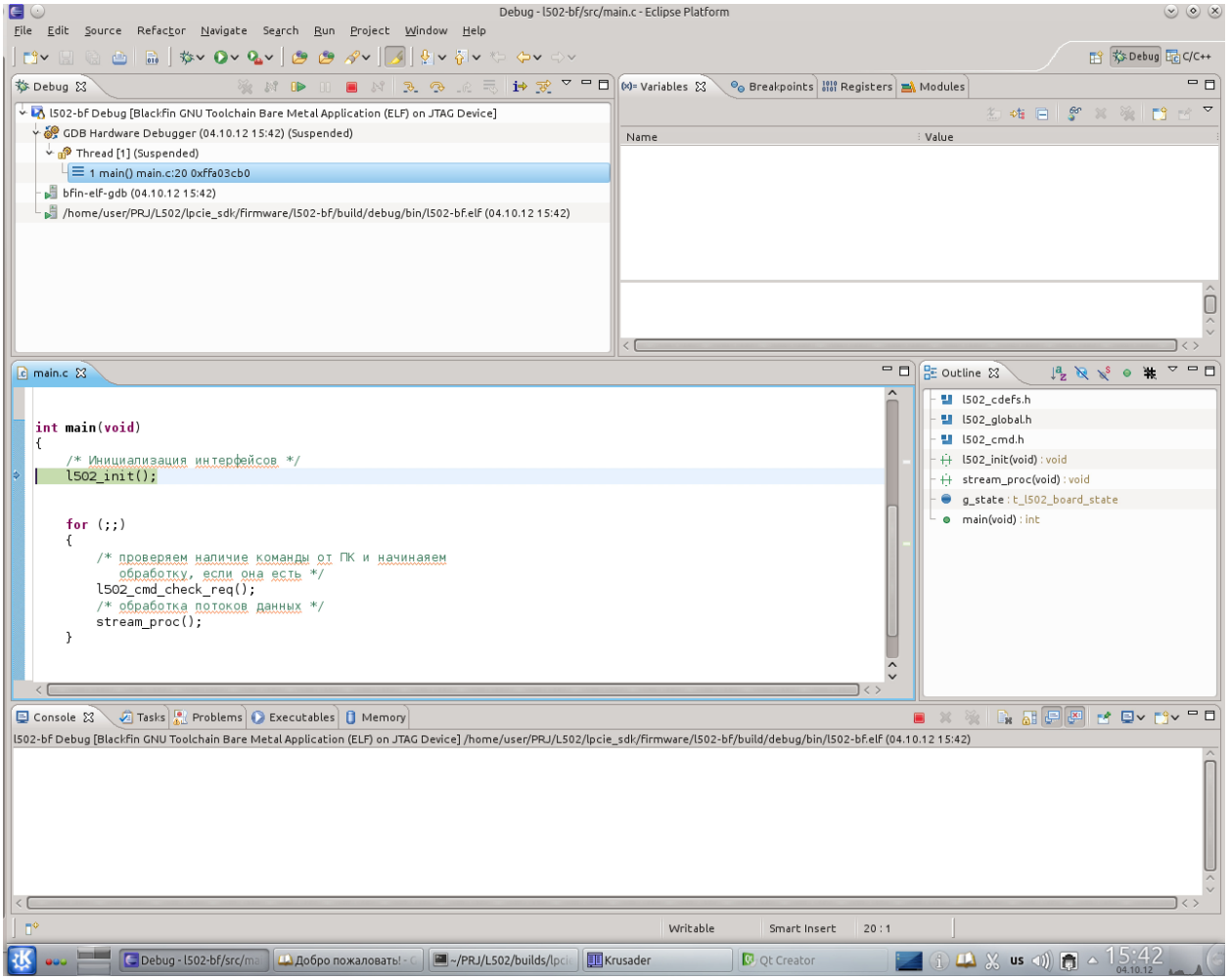

Рис. 4.12: Отладка прошивки l502-bf

4.4 Используемая периферия процессора

Перед штатной работой прошивки выполняется инициализация всей используемой периферии. Инициализация реализована в функции l502_init() в файле l502_init.c, которая внутри себя вызывает процедуры инициализации для каждого программного блока из других файлов.

Как уже было описано в разделе Загрузка и запуск прошивки BlackFin, инициализация PLL и SDRAM в штатном режиме выполняется уже загрузчиком значениями из OTP-памяти. В l502_init() проверяется, что эти значения есть в OTP-памяти, и, если их не окажется, то устанавливает PLL и SDRAM напрямую и записывает нужные настройки в OTP-память (то есть реально это происходит только во время первого запуска, который выполняется уже при наладке изделия в "L Card").

Далее при инициализации настраиваются следующие периферийные узлы сигнального процессора:

Периферия	**Режим**	**Назначение**
SPI	CLK=33.125 МГц, Мастер, 16 бит, CPHA=0, CPOL=0, MSB first	Запись и чтение содержимого регистров IO_HARD и IO_ARITH ПЛИС
SPORT0	CLK=66.25 МГц, внутренний; TFS внешний, активный единицей, ранний; 16-бит, 2 канала, MSB first	Прием данных от АЦП и DIN, передача отсчетов ЦАП и DOUT. Как на прием, так и на передачу используется 2 канала (Primary и Secondary)
HostDMA	16 бит, Acknowledge Mode, Burst	Прямой доступ к памяти BlackFin со стороны ПЛИС как для передачи команд от ПК, так и для приема/записи потоков данных
PF14, PF15, PG5, PG6	Выходы	Управление потоком данных по HostDMA на запись и чтение
DMA1	Настраивается дескриптором от ПЛИС	Передача данных по HostDMA
DMA3	Через регистры, автобуфер, 2D, 16 бит, SYNC	Запись принятых данных от АЦП/DIN по SPORT0 в память BlackFin
DMA4	Список дескрипторов (small), NDSIZE=5, 16 бит, SYNC	Передача данных из памяти BlackFin в SPORT0

В штатной прошивке используются следующие вектора прерываний (не все из них соответствуют настройкам по-умолчанию!):

Прерывание	Периферия	Описание
IVG7	DMA3 (SPORT0 RX)	Завершение приема по DMA данных из интерфейса SPORT0
IVG9	DMA4 (SPORT0 TX)	Завершение передачи по DMA данных в интерфейс SPORT0
IVG10	HostDMA Read	Завершение операции чтения по HostDMA
IVG11	HostDMA Write	Завершение операции записи по HostDMA

4.5 Распределение памяти сигнального процессора

Для хранения данных процессору BlackFin доступно два банка внутренней памяти по 32 КБайта и 32 МБайт внешней памяти SDRAM (Подробнее о адресном пространстве процессора ADSP-BF523 см. `Hardware Manual`, главу 3 Memory).

Большую часть памяти можно использовать по своему усмотрению, однако есть часть фиксированных адресов в начале первого банка внутренней памяти (начиная с адреса 0xFF800000).

Для этой области выделена специальная секция в файле линекра ('board_state') размером 8 КБайт. В эту секцию помещена структура g_state типа t_l502_board_state.

Область адресов 0xFF800000-0xFF8007FF используется для хранения дескрипторов DMA при передаче потока данных из BlackFin в персональный компьютер и обратно через интерфейс HostDMA. Эти адреса заданы в логике работы ПЛИС и всегда должны быть использованы по этому назначению.

Остальная часть фиксированной области памяти используется для передачи команд от ПК через интерфейс HostDMA. Взаимодействие штатной прошивки BlackFin и ПО персонального компьютера осуществляется через специальный командный интерфейс, который подробнее описан в разделе Командный интерфейс между ПК и сигнальным процессором.

Остальную область памяти программист может использовать по своему усмотрению. В штатной прошивке внешняя память SDRAM используется для буферов данных, в то время как все переменные хранятся во внутренней памяти для быстрого доступа. При таком варианте кеш-память, соответственно, не используется.

Следует также учитывать, что при создании больших буферов, начальное значение которых не важно, следует помещать их в секции неинициализируемой памяти, чтобы при заливке прошивки не приходилось загружать эти большие буфера. В штатной прошивке для этого в файле линкера создана специальная секция во внешней SDRAM. К сожалению, в VisualDSP можно указать, что секция находится в неинициализируемой области только через #pragma section(секция, NO_INIT), и нельзя сделать независимый от компилятора #define. Указание того, что массив должен быть в неинициализируемой области SDRAM выполняется с помощью включения файла, в котором объявлена нужная директива для компилятора. В примере ниже это используется для объявления буфера для приема данных по SPORT0 (АЦП/DIN) в файле l502_stream.c. Реально при сборке на VisualDSP включается файл "vdsp/l502_sdram_noinit.h", а при сборке GCC — "gcc/l502_sdram_noinit.h". Включать файл нужно строго перед каждой переменной, которую необходимо поместить в неинициализируемую область SDRAM:

```
#include "l502_sdram_noinit.h"
static volatile uint32_t f_sport_in_buf[L502_SPORT_IN_BUF_SIZE];
```

4.6 Интерфейс HostDMA

Интерфейс HostDMA предоставляет прямой доступ к памяти сигнального процессора BlackFin извне. При этом доступ может быть осуществлен как во внутреннюю память L1, так и во внешнюю SDRAM (доступа к регистрам ядра и периферии по HostDMA нет).

Интерфейс HostDMA в модуле L502 используется для следующих целей:

- Прямой доступ к памяти BlackFin со стороны ПК через регистры PCI блока управления DSP в ПЛИС. В частности через доступ к фиксированной области памяти реализован командных интерфейс между ПК и сигнальным процессором (см. раздел Командный интерфейс между ПК и сигнальным процессором).

- Прямой доступ к памяти BlackFin со стороны блока DMA ПЛИС для осуществления передачи потоков данных между памятью сигнального процессора и памятью ПК. Подробнее этот механизм описан в разделе Передача потоков данных между ПК и сигнальным процессором по интерфейсу HostDMA .

Важно! Запись и чтение по HostDMA всегда осуществляются блоками, кратными восьми 32-битным словам (32 байта). Таким образом при записи массива данных размером, некратным размеру блока, будет испорчена часть данных за записываемой частью до границы, кратной 32-м байтам. При чтении некратного размера также реально читается кратный блок большего размера, лишние данные из которого потом откидываются. Однако и при чтении и при записи некратных блоков необходимо, чтобы все реально читаемые или записываемые адреса кратного блока относились к действительной памяти сигнального процессора!

По завершению обмена одного блока данных по интерфейсу HostDMA в сигнальном процессоре вызывается прерывание. Для обработки завершения чтения используется обработчик hdma_rd_isr(), а для завершения записи hdma_isr(). При этом следует учитывать, что если эти обработчики не выполнят необходимые действия по завершению передачи блока, то следующий обмен по HostDMA не будет разрешен. Таким образом, интерфейс HostDMA в BlackFin не является полностью независимым от прошивки сигнального процессора и, если программа, например, остановлена на точке останова, то в это время обмен по HostDMA может не выполнится.

Для прямого доступа к памяти со стороны ПК в библиотеке l502api введены функции `L502_BfMemRead()` и `L502_BfMemWrite()`. При этом реально обмен осуществляется блоками с максимальным размером 128 32-битных слов (512 байт), т.е. при вызове функции с большим размером, массив данных будет разбит на несколько блоков, которые будут последовательно записаны/прочитаны.

При этом необходимо знать, что находится по адресам, к которым выполняется прямой доступ, и правильно распределять право доступа к разделяемой между ПК и сигнальным процессором памятью, чтобы не допускать чтение частично измененного блока памяти или одновременное изменение блока памяти как со стороны ПК, так и со стороны прошивки BlackFin. В штатной прошивке прямой доступ используется исключительно для организации командного интерфейса, который описан в следующем разделе.

4.7 Командный интерфейс между ПК и сигнальным процессором

Как уже отмечалось, управление сигнальным процессором с ПК осуществляется через интерфейс HostDMA, который предоставляет прямой доступ к памяти сигнального процессора.

В памяти сигнального процессора, начиная с адреса 0xFF800800, выделена специальная область, которая используется для передачи управляющих команд от ПК к сигнальному процессору. Все управление сигнальным процессором в штатной прошивке выполняется с помощью командного интерфейса, описанного в данной главе. Реализация обработки команд содержится в файле l502_cmd.c

Структура фиксированной области памяти, использующейся для передачи команд, описана с помощью типа t_l502_bf_cmd, а доступ к ней можно получить с помощью поля cmd глобальной переменной g_state.

Каждая команда характеризуется:

- Кодом команды, который собственно определяет, какое действие должно выполняться
- 32-битным параметром команды, который уточняет назначение команды. Его интерпретация полностью зависит от кода команды.
- Данными, переданными с командой (от 0 до L502_BF_CMD_DATA_SIZE_MAX 32-битных слов)

По завершению обработки команды прошивка сигнального процессора возвращает:

- Код завершения команды (0 при успехе, код ошибки при неудаче)
- Данные результата (от 0 до L502_BF_CMD_DATA_SIZE_MAX 32-битных слов).

Для проверки, выполняется ли сейчас команда, в t_l502_bf_cmd определено поле статуса команды. Выполнение команды осуществляется следующим образом:

- Изначально статус команды равен L502_BF_CMD_STATUS_IDLE (или L502_BF_CMD_STATUS_DONE, если до этого уже была выполнена команда)
- При необходимости передать команду в DSP программа ПК записывает в фиксированную область все параметры команды. При этом при последней операции записи в поле статуса записывается значение L502_BF_CMD_STATUS_REQ. Т.е. поле статуса записывается не обязательно самым последним, главное что статус изменяется при записи последнего блока по HostDMA. Поэтому проверка статуса команды выполняется именно из обработчика, когда блок полностью записан, а не из произвольного места в прошивке BlackFin.
- В обработчике прерывания на завершения записи по HostDMA hdma_isr() проверяется статус команды. В случае если он равен L502_BF_CMD_STATUS_REQ, то вызывается функция l502_cmd_set_req(), которая устанавливает статус в L502_BF_CMD_STATUS_PROGRESS и устанавливает флаг о том, что пришла команда.

- В основной программе периодически проверяется этот флаг через l502_cmd_check_req(). Если флаг установлен, то вызывается l502_cmd_start(), в которой начинается обработка команды. Таким образом, сама обработка команды выполняется не из обработчика прерываний. Функция l502_cmd_start() анализирует код команды и вызывает соответствующий обработчик команды. Для штатных команд существует таблица соответствия кодов команд и обработчиков. Для кодов команд больше или равных L502_BF_CMD_CODE_USER вызывается пользовательский обработчик usr_cmd_process(). Именно эти коды должен использовать пользователь, если хочет сохранить совместимость с штатной прошивкой.

- Обработчик может обработать команду сразу или отложить ее выполнение на потом, установив например специальный флаг о необходимости завершения и сразу вернув управление.

- По завершению обработки вызывается l502_cmd_done(), которая устанавливает переданный ей код результата, копирует переданные данные в g_state.cmd.data (если они уже не там), устанавливает их размер в g_state.cmd.data_size и в последнюю очередь устанавливает статус равным L502_BF_CMD_STATUS_DONE. Следует следить, чтобы на каждую команду была вызвана l502_cmd_done(). В противном случае программа ПК не сможет дождаться завершения команды.

- Программа ПК периодически читает состояние команды. Когда оно станет L502_BF_CMD_STATUS_DONE, программа определяет, что обработка команды завершена и считывает ее результат.

- После этого программа ПК может передавать следующую команду.

Таким образом, одновременно может выполняться только одна команда. При этом для правильного функционирования интерфейса команд необходимо чтобы как программа на ПК, так и программа сигнального процессора соблюдали правила доступа к разделяемой структуре g_state.cmd и изменяли ее только в те моменты, когда доступ находится у соответствующей программы:

- Изначально доступ к параметром команды находится у ПК и программа ПК может в любой момент передать команду

- После установки поля статуса команды в L502_BF_CMD_STATUS_REQ, доступ к структуре переходит в собственность прошивки сигнального процессора и она может менять ее содержимое по своему усмотрению до завершения обработки

- После установки прошивкой статуса команды в L502_BF_CMD_STATUS_DONE (из l502_cmd_done()) доступ снова переходит к программе ПК, и она может передавать следующую команду.

Штатные команды используются штатной библиотекой l502api внутри стандартных функций, когда модуль находится в DSP режиме. Они соответствуют назначению стандартных функций (установка параметров сбора, запуск, останов синхронного сбора данных, разрешение/запрещение синхронных потоков и т.д.). Пользователь может добавить в прошивку свои команды. Для выполнения пользовательских команд в библиотеке l502api введена функция `L502_BfExecCmd()`. Для совместимости с штатной прошивкой рекомендуется пользователю добавлять свои команды с кодом L502_BF_CMD_CODE_USER или выше, так как в этом случае пользовательские команды не будут пересекаться с штатными (если будут введены новые) и на эти коды всегда вызывается функция usr_cmd_process(), что позволяет не изменять основную прошивку.

4.8 Передача потоков данных между ПК и сигнальным процессором по интерфейсу HostDMA

Через HostDMA также передаются потоки данных от ПК в сигнальный процессор и от сигнального процессора в ПК. Как было уже сказано до этого, направление потока данных определяется относительно ПК (поток ввода IN — АЦП/DIN -> FPGA -> SPORT0 -> BlackFin -> HDMA -> FPGA -> PC, а поток вывода OUT — PC -> FPGA -> HDMA -> BlackFin -> SPORT0 -> FPGA -> ЦАП/DOUT). При этом названия send, recv в функциях даны относительно DSP.

Помимо самого интерфейса HostDMA, для управления потоком дополнительно используются 4 ножки общего назначения (настроенные как выход).

Выводы PF14 и PF15 используются для запрещения/разрешения передачи потока на ввод и вывод соответственно. Когда на них выставлена "1", то в ПЛИС автомат, ответственный за передачу потока данного направления, находится в состоянии сброса. Передача в этом направлении не ведется. Все поставленные в очередь передачи по HostDMA очищаются (однако если идет в данный момент передача блока по HostDMA, то она завершается до конца, так как разрывать передачу посередине нельзя).

Выводы PG5 и PG6 используются, чтобы сообщить ПЛИС о том, что BlackFin готов передавать или принимать блок данных соответственно. Изменение состояния этих выходов (toggle) при нулевом значении на выходах PF14 или PF15 является признаком, что в памяти BlackFin есть готовый новый блок для передачи или есть место для приема очередного блока на вывод. При этом можно сообщить о готовности нового блока до завершения предыдущего, так как ПЛИС сохраняет количество готовых, но не завершенных запросов на обмен. Всего может быть до 31 (определено через L502_IN_HDMA_DESCR_CNT и L502_OUT_HDMA_DESCR_CNT) незавершенных запросов по каждому направлению.

Для описания запросов на обмен используются дескрипторы, которые хранятся в фиксированной области памяти BlackFin (`g_state.hdma.in[]` и `g_state.hdma.out[]`). Состав дескриптора описан типом t_hdma_stream_descr. Каждый дескриптор содержит следующую информацию:

- Полный размер передачи в 16-битных словах
- Адрес буфера в памяти BlackFin из которого данные должны быть переданы или в который должны быть записаны
- Размер непрерывного блока передачи в 16-битных словах! (см. ниже)
- Модификатор указателя (на сколько изменяется адрес при передачи очередного 16-битного слова, для непрерывной передачи равен 2)
- Адрес следующего дескриптора
- Флаги (сейчас определен только один флаг - L502_HDMA_FLAGS_SEND_LAST)
- id дескриптора (используется только для проверки, какой дескриптор завершен)
- valid - признак действительности дескриптора (используется только для проверки завершения)

После сброса автомата ПЛИС ножками PF14 и PF15, автомат ожидает дескриптор по фиксированным адресам: &g_state.in[0] (0xFF800020) для ввода и &g_state.out[0] (0xFF800420) для вывода. Адрес же каждого следующего дескриптора задается в предыдущем дескрипторе.

По завершению чтения или записи данных, соответствующих дескриптору, ПЛИС записывает обработанный дескриптор полностью обратно в память BlackFin по адресам &g_state.in_lb (0xFF800000) и &g_state.out_lb (0xFF800400). При записи назад обработанного дескриптора изменяются следующие поля:

- поле адреса указывает на адрес за последним переданным словом
- поле full_size указывает, сколько осталось передать данных в 16-битных словах (0, если все передано)
- остальные поля сохраняются без изменений

Прошивка использует поле valid, чтобы узнать о завершении запроса. Изначально g_state.in_lb.valid и g_state.out_lb.valid равны нулю. В передаваемых дескрипторах поле valid установлено в единицу. Таким образом, если в g_state.in_lb.valid или g_state.out_lb.valid станет равен единице после завершения записи по HostDMA, то это означает что блок данных, соответствующий дескриптору был передан (если его full_size стал равен нулю).

Один запрос может выполняться за несколько передач. Именно для этого введено поле xcnt в дополнение к full_size. Дело в том, что одна передача по HostDMA не может быть прервана другой, поэтому при использовании больших размеров передач шина будет занята надолго одной передачей и не будет доступна для других. Кроме того, сама передача не начинается, пока во внутреннем буфере DMA в ПЛИС не будет достаточно места (чтобы не получилось, что передача остановлена на середине из-за того, что нет места во внутреннем буфере). При этом, если на каждый небольшой блок автомат ПЛИС будет должен прочитать дескриптор, выполнить обмен и записать дескриптор снова для оповещения о завершении — это приведет к большим дополнительным расходам пропускной способности HostDMA. Таким образом, один дескриптор может описывать задачу на передачу большого блока данных (=full_size), в то время как ПЛИС разобьет их на несколько передач (размер каждой соответствует xcnt). full_size не обязательно должен быть кратен xcnt, так как ПЛИС может выполнить последний обмен меньшего размера. В общем случае xcnt не должен превышать 256, в то время как full_size может достигать размеров всей памяти BlackFin.

Важно! Поля full_size и xcnt задаются в количестве 16-битных слов, а не 32-битных, так как шина HostDMA 16-битная.

В то время как запрос на ввод (передача от BlackFin в ПК) всегда выполняется полностью и в возвращенном дескрипторе full_size = 0, то на вывод ситуация иная. Если ПЛИС выполнит передачу только части блока данных, но при этом от ПК не будет новых данных, то будет записан дескриптор в out_lb с full_size равным количеству непереданных данных. При этом при поступлении новых данных будет продолжен обрабатываться незавершенный запрос и следующая запись в out_lb будет соответствовать тому же дескриптору, но уже с меньшим full_size. Это позволяет пользователю передавать данные на вывод не сплошным потоком, и BlackFin сможет их обработать, не дожидаясь, когда будут записаны все full_size/2 отсчетов.

Таким образом, алгоритм приема потока вывода выглядит следующим образом:

- При запуске потока прошивка вызывает функцию hdma_recv_start(), которая устанавливает выход PF15 в 0 — разрешается поток на вывод

- При наличии места для приема данных вызывается функция hdma_recv_req_start() с указанием буфера — функция заполняет дескриптор g_state.out[0] нужными значениями (full_size=size*2, xcnt=min(256, size*2), addr, valid=1, id) и изменяет состояние выхода PG6, чтобы сообщить ПЛИС о готовности нового запроса
- ПЛИС по этому сигналу, когда будет свободна шина HostDMA, считывает дескриптор из g_state.out[0]
- При наличии данных в памяти ПК на передачу и свободной шине HostDMA идет запись в память BlackFin по адресу g_state.out[0].addr данных из памяти ПК блоками размером не больше xcnt 16-битных слов.
- При передаче всех full_size слов или при отсутствии данных в памяти ПК на передачу записывается считанный до этого дескриптор с измененными полями full_size и addr по адресу &g_state.out_lb.
- По завершению этой записи в обработчике hdma_isr() прошивка определяет по g_state.out_lb.valid, что записан дескриптор, устанавливает g_state.out_lb.valid в 0, вычисляет обработанный размер и вызывает hdma_recv_done().
- hdma_recv_done() в l502_stream.c помечает данные как готовые к обработке. После из уже не из обработчика прерываний, а из основной программы, будет вызвана usr_out_proc_data() для обработки принятых данных.
- Если full_size=0, то дескриптор полностью обработан и можно переходить к следующему, иначе — продолжается обмен, соответствующий данному дескриптору.
- По мере необходимости добавляются новые дескрипторы через hdma_recv_req_start(). Дожидаться завершения предыдущего не обязательно — задания на прием данных ставятся в очередь. Только нужно проверять, что есть место в очереди через hdma_recv_req_rdy().
- При завершении потокового вывода вызывается hdma_recv_stop(), которая устанавливает PF15 в 1, что приводит к сбросу автомата передачи потока на вывод по HostDMA в ПЛИС.

Алгоритм передачи данных для потока ввода выглядит схожим образом:

- При запуске потока прошивка вызывает функцию hdma_send_start(), которая устанавливает выход PF14 в 0 — разрешается поток на ввод
- При наличии блока данных на передачу необходимо вызвать hdma_send_req_start() с указанием буфера с данными — функция заполняет дескриптор g_state.in[0] нужными значениями (full_size=size*2, xcnt=min(256, size*2), addr, valid=1, id) и изменяет состояние выхода PG5, чтобы сообщить ПЛИС о готовности нового запроса.
- ПЛИС по этому сигналу, когда будет свободна шина HostDMA, считывает дескриптор из g_state.in[0].
- При наличии места в памяти ПК на прием и свободной шине HostDMA идет чтение данных из памяти BlackFin, начиная с адреса g_state.in[0].addr, блоками размером не больше xcnt 16-битных слов с последующей записью в память ПК.

- При передаче всех full_size слов записывается считанный до этого дескриптор с измененными полями full_size и addr по адресу &g_state.in_lb.
- По завершению этой записи в обработчике hdma_isr() прошивка определяет по g_state.in_lb.valid, что записан дескриптор, устанавливает g_state.in_lb.valid в 0, вычисляет обработанный размер и вызывает hdma_send_done().
- hdma_send_done() реализовано в l502_user_process.c.
- При передаче на вывод в обратно записанном дескрипторе full_size=0, что соответствует полностью обработанному дескриптору.
- По мере необходимости добавляются новые дескрипторы через hdma_send_req_start(). Дожидаться завершения предыдущего не обязательно — задания на передачу данных ставятся в очередь. Только нужно проверить, что есть место в очереди через hdma_send_req_rdy().
- При завершении потокового ввода вызывается hdma_send_stop(), которая устанавливает PF14 в 1, что приводит к сбросу автомата приема потока на ввод по HostDMA в ПЛИС.

4.9 Прием потока данных по SPORT0

В отличии от остальных интерфейсов для приема-передачи потоков данных, где обмен осуществляется через дескрипторы и может быть приостановлен, пока новый дескриптор не будет поставлен в очередь, прием синхронных данных от АЦП/DIN всегда осуществляется непрерывно со скоростью ввода этих данных. Прием осуществляется 32-битными словами. При этом каждое 32-битное слово передается как два 16-битных, передаваемых параллельно — одно по первичному каналу SPORT0, второе — по вторичному. При приеме используется внешний сигнал начала кадра (RFS), который генерирует ПЛИС каждый раз при наличии нового принятого отсчета АЦП или DIN. Сигнал RFS должен быть явно разрешен с помощью установки бита `BF_RFS_EN` регистра `OUTSWAP_BFCTL`.

Для приема используется третий канал DMA BlackFin в режиме автобуфера. Прием данных происходит постоянно в буфер `f_sport_in_buf` размером L502_SPORT_IN_BUF_SIZE. Сам буфер разделен на блоки, размер каждого из которых равен шагу прерывания. Этот шаг может быть установлен с помощью функции sport_in_set_step_size(). В штатной прошивке установка этого параметра осуществляется с ПК через команду L502_BF_CMD_CODE_SET_PARAM с параметром L502_BF_PARAM_IN_STEP_SIZE, но ничего не мешает изменять его из самой прошивки пока синхронные потоки не запущены.

Для реализации приема данных с указанным шагом используется 2D режим DMA, где размер по X равен размеру шага (умноженному на 2, т.к. используется 16-битный режим DMA), а по Y - количеству шагов, которое помещается в буфере на прием (должно быть не меньше 3). Таким образом, если размер буфера на прием не кратен шагу, то реально используется не весь буфер, а наибольший размер кратный шагу, но не превышающий размер буфера.

После приема данных, размер которых соответствует шагу прерывания, будет вызван обработчик isr_sport_dma_rx(), в котором обновляется размер принятых данных

в буфере и проверяется переполнение. Т.к. прерывание возникает при завершении приема блока данных, когда уже начат прием следующего блока, то, чтобы не допустить порчи данных из-за переполнения буфера, необходимо, чтобы в момент прерывания было свободно для приема как минимум два блока. В противном случае, если к следующему прерыванию хотя бы еще один блок не освободится, то о переполнении прошивка узнает только в момент, когда часть буфера уже была повреждена (что может привести к тому, что будут обработаны испорченные данные). Таким образом, для надежного приема данных, если в прерывании обнаруживается, что свободно меньше двух блоков, то это считается переполнением.

По переполнению останавливается прием по интерфейсу SPORT0 и взводится специальный флаг. После этого обрабатываются все принятые, но необработанные данные в буфере, после чего в ПК посылается слово о том, что в данном месте было переполнение.

Управление приемом по SPORT0 осуществляется из функций управления потоками: при запуске синхронного ввода-вывода и разрешенных потоках на ввод выполняется функция sport_rx_start(), которая настраивает DMA, сбрасывает указатели в буфере на прием и разрешает прием по SPORT0. При останове синхронного ввода-вывода или запрете всех потоков на ввод выполняется останов приема по SPORT0 через sport_rx_stop().

4.10 Передача потока данных по SPORT0

Передача по SPORT0 данных для вывода на ЦАП и DOUT осуществляется во многом подобно тому, как происходит передача по HostDMA. Для этого также используется набор дескрипторов, которые можно поставить в очередь с помощью sport_tx_start_req() и узнать количество свободных дескрипторов в очереди с помощью sport_tx_req_rdy().

В отличие от HostDMA для этого используются стандартные дескрипторы DMA процессора BlackFin. Также не требуется отдельно запускать передачу по SPORT0 — разрешение на передачу выполняется при добавлении первого дескриптора. Но для останова всех данных и освобождения всех дескрипторов существует функция sport_tx_stop().

Передача по SPORT0 аналогично приему осуществляется 32-битными словами в параллель двумя половинами по 16-бит — одна по первичному каналу, вторая по вторичному. Также как и для приема необходимо разрешение генерации внешних TFS. Причем это выполняется в логике управления потоками при разрешении потоков на вывод. А постановка новых дескрипторов на передачу выполняется из пользовательской обработки.

4.11 Общая логика управления потоками

Логика управления потоками в штатной прошивке аналогична этой же логике в l502api:

- Каждый источник ввода-вывода (АЦП, DIN, 1-ый канал ЦАП, 2-ой канал ЦАП, DOUT) может быть настроен на синхронный потоковый режим с помощью функций stream_enable() и запрещен синхронный режим по нему с помощью stream_disable().

- Все разрешенные потоки запускаются одновременно с помощью функции streams_start(). Эта функция разрешает генерацию синхрочастоты, прием и передачу по HostDMA, прием и передачу по SPORT0 (если нужные потоки разрешены).
- Данные на вывод могут быть предварительно загружены, для чего функция stream_out_preload() инициализирует передачу по HostDMA без запуска синхронных сбора-выдачи.
- Во время запущенного синхронного ввода-вывода изменять настройки ввода-вывода нельзя (логическую таблицу, частоту сбора и т.д.).
- Во время сбора данных можно дополнительно разрешать или запрещать синхронные потоки (но пока не все варианты реализованы...)
- По вызову streams_stop() останавливается синхронный ввод-вывод и запрещаются все потоки данных по SPORT0 и HostDMA.

Как для потока ввода, так и для потока вывода в l502_stream.c выделен буфер для приема данных. Для отслеживания состояния данных для каждого буфера используются 3 указателя:

- Индекс, указывающий на слово сразу за последним принятым словом (f_sport_in_put_pos/f_hdma_out_put_pos). Обновляется в обработчике прерывания при приеме очередного блока данных.
- Индекс, указывающий на следующее слово для обработки (f_sport_in_proc_pos/f_hdma_out_proc_pos). Вместе с предыдущем определяет количество принятых, но еще не обработанных данных. Фоновая функция stream_proc() проверяет наличие необработанных данных и вызывает функцию пользовательской обработки: usr_in_proc_data() или usr_out_proc_data(). Эти функции возвращают количество обработанных данных, на которое и увеличивается индекс слова для обработки. Эти данные считаются обработанными, но буфер все еще занятым.
- Индекс, указывающий на следующее занятое слово (f_sport_in_get_pos/f_hdma_out_get_pos). Используется, чтобы определить, сколько места в буфере свободно для приема новых данных. Этот размер нужен для постановки новых дескрипторов на прием по HostDMA или для определения факта переполнения при приеме по SPORT0. Эти индексы обновляются при вызове функций stream_in_buf_free()/stream_out_buf_free() из пользовательских обработчиков.

Разграничение обработанных и занятых данных можно использовать, например в следующих случаях:

- Случай, когда еще нет возможности обработать данные, функция обработки может вернуть 0. Тогда в последующем она будет вызвана снова для обработки этих же данных (возможно с дополнительными данными, принятыми за интервал между вызовами). При этом функции обработки не нужно копировать данные для гарантии, что они не будет перезаписаны. Так например происходит в штатной прошивке, когда нет свободных дескрипторов для передачи принятых данных.
- Если обработка ведется заданными блоками, то функция может возвращать обработанный размер всегда кратный размеру блоку. При этом в дальнейшем необработанный неполный блок будет снова передан в функцию обработки при следующем вызове.

4.12 Форматы данных в потоке на ввод и на вывод

Существует только один поток на ввод и один на вывод. Данные в потоке передаются в виде 32-битных слов. При этом в 32-битном слове, помимо самих данных закодировано и что это за данные (АЦП или DIN).

Определить, что за данные, можно по старшему байту принятого слова. Ниже приведена таблица для определения какому типу данных принадлежит принятое слово из потока ввода:

Биты 31-24	**Значение**
1xxxxxxx	Данные от АЦП
00000000	Данные с цифровых входов
00000001	Сообщения (от BF в ПК)
001xxxxx	Резерв
01xxxxxx	Пользовательские данные

Непосредственно от ПЛИС по SPORT0 приходят только данные от АЦП и цифровых входов. Сообщения могут генерироваться в прошивке BlackFin для уведомления программы ПК о событиях. Сейчас используется только одно сообщение — сообщение о переполнении буфера. Штатная прошивка передает это сообщение при переполнении буфера SPORT0 на прием. При работе в режиме без DSP это сообщение генерируется аппаратно ПЛИС при переполнении внутреннего буфера DMA на ввод.

Данные от АЦП приходят в виде расширенного 24-битного формата в дополнительном коде, так как к ним уже аппаратно применены калибровочные коэффициенты. Код L502_ADC_SCALE_CODE_MAX соответствует напряжению равному границе данного диапазона (например, +10В, +2В и т.д.).

Данные от АЦП приходят в следующем формате:

31-24	**23-0**
11mmcccc	ddddddddddddddddddddddddd

- d — 24-битный скорректированный дополнительный код АЦП.
- c — номер канал: "0" — 1-ый или 17-ый, "15" — 16-ый или 32-ой канал (в зависимости от режима).
- m — режим измерения:
  - 0 — дифференциальный
  - 1 — первые 16 каналов с общей землей
  - 2 — вторые 16 с общей землей
  - 3 — собственный ноль

С цифровых линий данные приходят в следующем формате:

31-24	**23-16**	**15 - 8**	**7 - 0**
00000000	000000ss	dddddddd	dddddddd

- d — значения цифровых линий
- s — значение линий DI_SYN1 (16-ый бит) и DI_SYN2 (17-ый бит)

Для слов потока на вывод схожим образом по старшем битам определяется, для чего предназначены данные:

Биты 31-24	23-16	15 - 8	7 - 0	Значение
00000000	000000ee	dddddddd	dddddddd	Цифровой вывод (e - разрешение старшей и младшей половин)
01000000	00000000	dddddddd	dddddddd	код для ЦАП1
10000000	00000000	dddddddd	dddddddd	код для ЦАП2
110xxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	Резерв
111xxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	Пользовательские данные

Коды ЦАП в отличие от кодов АЦП не откалиброваны. Калибровку необходимо выполнять либо в ПО на ПК (что доступно в l502api), либо использовать коэффициенты из g_module_info.

Комбинации, помеченные как пользовательские данные, пользователь может использовать для передачи своих данных в общем потоке от DSP в ПК или от ПК в DSP. Эти комбинации не должны передаваться в ПЛИС по SPORT0.

4.13 Доступ к регистрам ПЛИС

Прошивка имеет доступ к части регистров ПЛИС для настройки параметров ввода-вывода. BlackFin имеет доступ только к регистрам IO_HARD и IO_ARITH при включенном режиме DSP (доступ к этим регистрам с ПК по PCI-Express при этом запрещается). Все адреса при этом остаются теми же как и при работе с ними со стороны ПК (используются именно адреса 32-битных регистров, приведенные в данном документе, а не адреса памяти) и эти значения описаны в файле `l502_fpga_regs.h`.

Список регистров приведен в разделе 3.4.3.

Для чтения и записи регистров используются функции fpga_reg_write() и fpga_reg_read().

При этом сам доступ идет по SPI-интерфейсу и осуществляется за четыре 16-битных цикла. BlackFin при этом является мастером. Ниже приведены данные, передаваемые и принимаемые при циклах записи и чтения регистра ПЛИС.

Цикл записи регистра выглядит следующим образом:

№ цикла SPI	MOSI	MISO	Описание
1	11aaaaaaaaaaaaa	-	Признак начала записи и адрес регистра
2	00000000dddddddd	-	Данные (биты 31-24)
3	00000000dddddddd	-	Данные (биты 15-8)
4	00000000dddddddd	-	Данные (биты 7-0)

Цикл чтения регистра выглядит следующим образом:

№ цикла SPI	MOSI	MISO	Описание
1	10aaaaaaaaaaaaaa	-	Признак начала чтения и адрес регистра
2	-	-	Пустой цикл
3	-	dddddddddddddddd	Данные (биты 31-16)
4	-	dddddddddddddddd	Данные (биты 15-0)

4.14 Настройка параметров сбора данных

Как и для API верхнего уровня, прошивка BlackFin может устанавливать параметры сбора данных только в момент, когда не запущен синхронный ввод-вывод. В штатной прошивке, как и в API верхнего уровня, сперва задаются все параметры, после чего вызывается специальная функция (configure()), которая их непосредственно записывает в регистры ПЛИС. Сами параметры хранятся в структуре g_set. Они могут быть заданы как командами из программы ПК при вызове `L502_Configure()`, так и изменены внутри самой прошивкой. Для изменения можно как напрямую изменять поля структуры g_set, так и использовать функции из l502_param.h (params_set_lch(), params_set_adc_freq_div() и т.д.), которые дополнительно проверяют значения параметров. Сами параметры аналогичны используемым в API верхнего уровня (правда частоты сбора необходимо устанавливать вручную с помощью делителей и межкадровой задержки).

После изменения параметров из прошивки необходимо перед запуском сбора не забыть вызвать configure().

4.15 Информация о модуле

Так как BlackFin не имеет доступа к регистрам EEPROM, то из прошивки нельзя напрямую узнать информацию о модуле, которая может понадобится:

- наличие опций (в данном случае может интересовать наличие ЦАП, т.к. DSP и гальваноразвязка должны быть)
- версия ПЛИС и PLDA (чтобы знать, можно ли полагаться на аппаратные возможности, которые могут быть введены в дальнейшем)
- калибровки ЦАП, чтобы выставить на ЦАП откалиброванные значения из прошивки BlackFin по каким либо условиям. В отличие от данных АЦП, которые приходят уже после аппаратной коррекции, к данным ЦАП нужно применять калибровочные коэффициенты программно. В случае передачи значений из программы ПК, калибровка может быть уже применена в ПК. В случае, если значения формируются внутри прошивки BlackFin, код должен быть скорректирован самой прошивкой.

Вся эта информация передается прошивке BlackFin программой ПК с помощью команд сразу после загрузки прошивки с помощью `L502_BfLoadFirmware()` или при вызове `L502_BfCheckFirmwareIsLoaded()`. Эта информация сохраняется в глобальной структуре g_module_info.

Глава 5

Константы, типы данных и функции стандартной прошивки BlackFin

5.1 Фиксированная область памяти

5.1.1 Файлы.

Файл	Описание
l502_global.h	Файл содержит определение структур, описывающих область памяти BlackFin с фиксированными адресами:

5.1.2 Глобавльные переменные

Переменная	**Тип**	**Описание**
`g_state`	t_l502_board_state	Структура, содержащая данные фиксированной памяти

5.1.3 Константы и макроопределения.

Константа	**Значение**	**Описание**
`L502_BF_FIRM_VERSION`	`0x00000001`	Версия прошивки BlackFin
`L502_BF_FIRM_FEATURES`	`0x0`	Дополнительные возможности, реализованные в прошивке
`L502_IN_HDMA_DESCR_CNT`	`(31)`	Максимальное количество дескрипторов на передачу данных (Bf->ПК)
`L502_OUT_HDMA_DESCR_CNT`	`(31)`	Максимальное количество дескрипторов на прием данных (ПК->Bf)

5.1.4 Структуры.

5.1.4.1 Структура дескриптора для передачи потока по HostDMA

Тип: t_hdma_stream_descr		
Описание: Структура дескриптора для передачи потока по HostDMA		
Поле	**Тип**	**Описание поля**
flags	uint32_t	Флаги
full_size	uint32_t	Полный размер передачи
addr	void *	Начальный адрес данных для передачи
xcnt	uint16_t	Кол-во 16-разрядных слов в одном передаваемом блоке
xmod	uint16_t	Модификатор указателя
ycnt	uint16_t	Не используется
ymod	uint16_t	Не используется
next_descr	void *	Указатель на следующий дескриптор
id	uint16_t	id-номер дескриптора
valid	uint16_t	Признак действительности
udata	uint32_t	Доп данные

5.1.4.2 Описание массива дескрипторов HDMA.

Тип: t_hdma_descr_arr		
Описание: Структура описывает расположение дескрипторов HostDMA для организации потока данных между BlackFin и персональным компьютером как на передачу, так и на прием		
Поле	**Тип**	**Описание поля**
in_lb	t_hdma_stream_descr	Последний выполненный дескриптор на ввод
in	t_hdma_stream_descr	Список дескрипторов на ввод
out_lb	t_hdma_stream_descr	Последний выполненный дескриптор на вывод
out	t_hdma_stream_descr	Список дескрипторов на вывод

5.1.4.3 Описание фиксированной области памяти

Тип: t_l502_board_state		
Описание: Структура описывает данные в памяти сигнального процессора, которые должны быть расположены по фиксированным адресам		
Поле	**Тип**	**Описание поля**
hdma	t_hdma_descr_arr	0xFF800000 - дескрипторы для обмена потоком по HDMA
cmd	t_l502_bf_cmd	0xFF800800 - команда обмена с PC

5.2 Обработка команд от ПК

5.2.1 Файлы.

Файл	Описание
l502_cmd.h	Файл содержит описания функций, которые используются для обработки команд от ПК.
l502_cmd.c	Файл содержит логику обработки команд от ПК, переданных через HostDMA в BlackFin. Для каждого кода команды в таблице f_cmd_tbl задана функция для обработки команды. Если в таблице код команды не найден, то возвращается ошибка. Для пользовательских команд всегда вызывается usr_cmd_process().
l502_bf_cmd_defs.h	Файл содержит определения, которые используются для передачи команд от ПК в DSP (определения команд, их параметров, результатов выполнения)

5.2.2 Константы и макроопределения.

Константа	Значение	Описание
L502_BF_CMD_DATA_SIZE_MAX	(1024)	Максимальный размер данных, передаваемых с командой в 32-битных словах
L502_BF_CMD_CODE_USER	0x80000000UL	Код, с которого начинаются пользовательские команды

5.2.3 Перечисления.

5.2.3.1 Статус команд управления сигнальным процессором

Тип: t_l502_bf_cmd_status		
Описание: Статус команд управления сигнальным процессором		
Константа	**Значение**	**Описание**
L502_BF_CMD_STATUS_IDLE	= 0x0	Начальное состояние (команда вообщен не выполнялась)
L502_BF_CMD_STATUS_REQ	= 0x5A01	Передан запрос на обработку команды от ПК
L502_BF_CMD_STATUS_PROGRESS	= 0x5A02	Сигнальный процессор начал обработку команды
L502_BF_CMD_STATUS_DONE	= 0x5A03	Команда выполнена. Результат выполнения в поле ret_code

5.2.3.2 Коды команд управления сигнальным процессором

Тип: t_l502_bf_cmd_code
Описание: Коды команд управления сигнальным процессором

Константа	Значение	Описание
L502_BF_CMD_CODE_TEST	= 0x01	Запуск теста (параметр определяет тип теста)
L502_BF_CMD_CODE_SET_PARAM	= 0x02	Установить параметр (код параметра в поле param)
L502_BF_CMD_CODE_GET_PARAM	= 0x03	Прочитать текущее значение параметра
L502_BF_CMD_CODE_CONFIGURE	= 0x04	Сконфигурировать модуль в соответствии с ранее установленными параметрами
L502_BF_CMD_CODE_STREAM_EN	= 0x05	Разрешение потоков ввода/вывода
L502_BF_CMD_CODE_STREAM_DIS	= 0x06	Запрещение потоков ввода/вывода
L502_BF_CMD_CODE_STREAM_START	= 0x07	Запуск потоков ввода/вывода
L502_BF_CMD_CODE_STREAM_STOP	= 0x08	Останов потоков ввода/вывода
L502_BF_CMD_CODE_PRELOAD	= 0x09	Предзагрузка данных на ЦАП
L502_BF_CMD_CODE_ASYNC_OUT	= 0x10	Асинхронный вывод (куда - зависит от параметра)
L502_BF_CMD_CODE_ASYNC_DIG_IN	= 0x11	Асинхронный ввод с цифровых линий
L502_BF_CMD_CODE_ADC_GET_FRAME	= 0x12	Асинхронный ввод карда АЦП

5.2.3.3 Варианты тестов

Тип: t_l502_bf_test_code

Описание: Коды тестов, передающиеся в параметре команды L502_BF_CMD_CODE_TEST

Константа	Значение	Описание
L502_BF_CMD_TEST_STOP	= 0x00	Останов выполняемого теста
L502_BF_CMD_TEST_GET_RESULT	= 0x01	Получение результата теста
L502_BF_CMD_TEST_ECHO	= 0x10	Тест эхо - возвращает те же данные что передавались
L502_BF_CMD_TEST_SPORT	= 0x11	Тест интерфейса SPORT в кольцевом режиме
L502_BF_CMD_TEST_SDRAM	= 0x12	Тест SDRAM памяти
L502_BF_CMD_TEST_SPI	= 0x13	Тест интерфейса SPI

5.2.3.4 Устанавливаемые параметры

Тип: t_l502_bf_params

Описание: Коды пареметров, устанавливаемых командой L502_BF_CMD_CODE_SET_PARAM или получаемых с помщью команды L502_BF_CMD_CODE_GET_PARAM

Константа	Значение	Описание
L502_BF_PARAM_FIRM_VERSION	= 0x00	Версия прошивки - 4 байта
L502_BF_PARAM_STREAM_MODE	= 0x01	Режим работы (запущен поток или нет)

L502_BF_PARAM_ENABLED_STREAMS	= 0x02	Какие потоки разрешены
L502_BF_PARAM_MODULE_INFO	= 0x03	Запись информации о модуле
L502_BF_PARAM_IN_BUF_SIZE	= 0x10	Размер буфера на преием
L502_BF_PARAM_CYCLE_BUF_SIZE	= 0x11	Размер буфера для записи циклического сигнала
L502_BF_PARAM_LCH_CNT	= 0x20	Количество логических каналов в таблице
L502_BF_PARAM_LCH	= 0x21	Параметры логического канала
L502_BF_PARAM_ADC_FREQ_DIV	= 0x22	Делитель частоты АЦП
L502_BF_PARAM_REF_FREQ_SRC	= 0x23	Выбор опорной частоты
L502_BF_PARAM_ADC_FRAME_DELAY	= 0x24	Значение межкадровой задержки
L502_BF_PARAM_SYNC_MODE	= 0x25	Режим синхронизации
L502_BF_PARAM_SYNC_START_MODE	= 0x26	Условие запуска синхронных потоков сбора данных
L502_BF_PARAM_ADC_COEF	= 0x27	Установка коэффициентов для заданного диапазона АЦП
L502_BF_PARAM_DAC_COEF	= 0x28	Установка коэффициентов для заданного канала ЦАП
L502_BF_PARAM_DIN_FREQ_DIV	= 0x30	Делитель частоты цифрового ввода
L502_BF_PARAM_DAC_FREQ_DIV	= 0x31	Делитель частоты вывода на ЦАП
L502_BF_PARAM_IN_STEP_SIZE	= 0x32	Шаг для обработки входных данных
L502_BF_PARAM_IN_STREAM_MODE	= 0x100	Режим работы потока на ввод

5.2.3.5 Тип асинхронного вывода

Тип: t_l502_bf_cmd_async_type		
Описание: Код, задающий в параметре команды команды L502_BF_CMD_CODE_ASYNC_OUT, куда должно выводится передаваемое значение		
Константа	**Значение**	**Описание**
L502_BF_CMD_ASYNC_TYPE_DOUT	= 0x0	Вывод на цифровые линии
L502_BF_CMD_ASYNC_TYPE_DAC1	= 0x1	Вывод на первый канал ЦАП
L502_BF_CMD_ASYNC_TYPE_DAC2	= 0x2	Вывод на второй канал ЦАП

5.2.3.6 Коды завершения команд

Тип: t_l502_bf_err_code		
Описание: Коды завершения команд		
Константа	**Значение**	**Описание**
L502_BF_ERR_SUCCESS	= 0	Команда выполнена успешно
L502_BF_ERR_FIRST_CODE	= -512	Код ошибки, с которого начинаются отсальные коды. Используется, чтобы разделить на верхнем уровне ошибки библиотеки и возвращенные сигнальным процессором
L502_BF_ERR_UNSUP_CMD	= -512	Неизвестный код команды
L502_BF_ERR_CMD_OVERRUN	= -513	Пришла команда до того, как была завершена предыдущая
L502_BF_ERR_INVALID_CMD_PARAMS	= -514	Неверное значение параметра команды
L502_BF_ERR_INSUF_CMD_DATA	= -515	Недостаточное кол-во данных передано с командой
L502_BF_ERR_STREAM_RUNNING	= -516	Команда не допустима при запущеном сборе, а сбор запущен
L502_BF_ERR_STREAM_STOPPED	= -517	Команда допустима только при запущеном сборе, а сбор остановлен
L502_BF_ERR_NO_TEST_IN_PROGR	= -518	Сейчас не выполняется никакого теста
L502_BF_ERR_TEST_VALUE	= -519	Считано неверное значение при выполнении теста

5.2.3.7 Режим работы сигнального процессора

Тип: t_l502_bf_mode		
Описание: Режим работы сигнального процессора		
Константа	**Значение**	**Описание**
L502_BF_MODE_IDLE	= 0	Ждущий режим, поток не запущен
L502_BF_MODE_STREAM	= 1	Запущены потоки ввода-вывода
L502_BF_MODE_TEST	= 2	Тестовый режим

5.2.4 Функции.

5.2.4.1 Запуск обработки команды от ПК.

Формат: `void l502_cmd_start(t_l502_bf_cmd *cmd)`
Описание: Функция начинает обработку команды от ПК, переданной по HDMA, вызывая нужный обработчик по коду команды. Вызывается из `l502_cmd_check_req()` при наличии запроса, но можно запустить и из самой прошивки выполнение команды, заполнив поля в g_state.cmd и вызвав `l502_cmd_start()`. Команда считается завершенной, когда будет вызвана `l502_cmd_done()`, которая может быть вызвана как из `l502_cmd_start()`, так и позже из другого места.
Параметры: **cmd** — Структура, описывающая команду

5.2.4.2 Завершение выполнения команды от ПК

Формат: `void l502_cmd_done(int32_t result, uint32_t *data, uint32_t size)`
Описание: Пометить комнду как завершенную, записав ее реультат и возвращаемые данные в фиксированную область. Если data уже указывает на g_state.cmd.data, т.е. данные уже находятся в нужной памяти, то они копироваться не будут
Параметры: **result** — Код завершения команды **data** — Данные для ПК, возвращаемые командой **size** — Размер данных, возвращаемых командой (в 32-битных словах)

5.2.4.3 Проверка наличия команды от ПК

Формат: `void l502_cmd_check_req(void)`
Описание: Функция проверяет, есть ли команда от ПК для которой не была начата обработка (был вызов `l502_cmd_set_req()`, но еще не был соответствующий вызов `l502_cmd_start()`). Если есть необработанная команда, то для нее вызывается `l502_cmd_start()`.

5.2.4.4 Установка флага, что есть необработанная команда

Формат: `void l502_cmd_set_req(void)`
Описание: Функция вызвается из прерывания от HostDMA в случае, если при этой операции HostDMA была записана команда. Функция устанавливает флаг, который будет проверен уже из основной программы с помощью `l502_cmd_check_req()`.

5.2.5 Структуры.

5.2.5.1 Параметры команды

Тип: t_l502_bf_cmd		
Описание: Структура описывает расположение полей в области памяти BlackFin, используемой для передачи команд между персональным компьютером и сигнальным процессором		
Поле	**Тип**	**Описание поля**
code	uint16_t	Код команды из t_l502_bf_cmd_code
status	uint16_t	Статус выполнения - в обработчике не изменяется
param	uint32_t	Параметр команды
result	int32_t	Код результата выполнения команды
data_size	uint32_t	Количество данных, переданных с командой или возвращенных с ответом в 32-битных словах
data	uint32_t	Данные, передаваемые с командой и/или в качестве результата

5.2.5.2 Результат выполнения теста

Тип: t_l502_bf_test_res		
Описание: Структура описывает параметры выполняемого теста, возвращаемые в качестве данных на команду L502_BF_CMD_CODE_TEST с параметром L502_BF_CMD_TEST_GET_RESULT		
Поле	**Тип**	**Описание поля**
test	uint32_t	Номер выполняемого теста
run	uint32_t	Признак, запущен ли сейчас тест
stage	uint32_t	Этап выполнения теста
cntr	uint32_t	Счетчик - сколько раз прошел тест
err	int32_t	Код ошибки выполнения теста
last_addr	uint32_t	Последний используемый адрес
last_wr	uint32_t	Последнее записанное значение
last_rd	uint32_t	Последнее считанное значение

5.3 Пользовательская обработка данных

5.3.1 Файлы.

Файл	Описание
l502_user_process.h	Файл содержит описания функций, которые предназначены для изменения пользователем для написания своих алгоритмов обработки данных и реализации пользовательских команд.
l502_user_process.c	Файл содержит простейший вариант реализации пользовательских функций, в котором потоки данных передаются без изменения и никакие пользовательские команды не обрабатываются. Пользователь может изменить этот файл и добавить здесь свою обработку.

5.3.2 Функции.

5.3.2.1 Обработка принятого массива данных АЦП/DIN.

Формат: `uint32_t usr_in_proc_data(uint32_t *data, uint32_t size)`
Описание: Функция вызывается каждый раз, когда обнаружены новые данные от АЦП/цифровых входов, пришедшие по SPORT0. Функция должна обработать данные и вернуть количество обработанных данных, однако эти данные все еще считаются использованными (не могут быть переписаны новыми пришедшими данными) до тех пор пока не будет вызвана функция `stream_in_buf_free()`). Если функция вернет значение меньше чем size, то функция будут вызванна при следующем проходе еще раз с указателем на необработанные данные. В текущей реализации просто запускается передача данных по HDMA в ПК
Параметры: **data** — Указатель на массив с принятыми данными **size** — Количество принятых данных в 32-битных словах
Возвращаемое значение: Функция возвращает количество обработанных данных (от 0 до size). На эти данные не будет вызываться повторно `usr_in_proc_data()`, но они считаются еще используемыми

5.3.2.2 Обработка принятого массива с данными ЦАП/DOUT.

Формат: `uint32_t usr_out_proc_data(uint32_t *data, uint32_t size)`

Описание: Функция вызывается каждый раз, когда обнаружены новые данные, принятые от ПК по HDMA. Функция должна обработать данные и вернуть количество обработанных данных, однако эти данные все еще считаются использованными (не могут быть переписаны новыми пришедшими данными) до тех пор пока не будет вызвана функция `stream_out_buf_free()`).
Если функция вернет значение меньше чем size, то функция будут вызвана после еще раз с указателем на необработанные данные.
В текущей реализации просто запускается передача данных по SPORT для вывода на ЦАП/цифровые выходы.

Параметры:

data — Указатель на массив с принятыми данными

size — Количество принятых данных в 32-битных словах

Возвращаемое значение:
Функция возвращает количество обработанных данных (от 0 до size). На эти данные не будет вызываться повторно `usr_out_proc_data()`, но они считаются еще используемыми

5.3.2.3 Обработка пользовательских команд.

Формат: `void usr_cmd_process(t_l502_bf_cmd *cmd)`

Описание: Функция вызывается при приеме команды от ПК с кодом большим или равным `L502_BF_CMD_CODE_USER`.
По завершению обработки необходимо обязательно вызвать `l502_cmd_done()`, указав код завершения команды и при необходимости передать данные с результатом

Параметры:

cmd — Структура с описанием принятой команды

5.3.2.4 Обработка завершения передачи по HostDMA.

Формат: `void hdma_send_done(uint32_t *addr, uint32_t size)`

Описание: Функция вызывается из обработчика прерывания, когда завершилась передача блока данных по HDMA в ПК, поставленного до этого на передачу с помощью `hdma_send_req_start()`.

Параметры:

addr — Адрес слова, сразу за последним переданным словом

size — Размер переданных данных в 32-битных словах

5.3.2.5 Обработка завершения передачи по SPORT.

Формат: `void sport_tx_done(uint32_t *addr, uint32_t size)`
Описание: Функция вызывается из обработчика прерывания при завершении передачи блока данных по SPORT'у на цифровые выходы/ЦАП, поставленного до этого на передачу с помощью `sport_tx_start_req()`.
Параметры: **addr** — Адрес слова, сразу за последним переданным словом **size** — Размер переданных данных в 32-битных словах

5.4 Работа с интерфейсом HostDMA

5.4.1 Файлы.

Файл	Описание
l502_hdma.c	Файл содержит логику работы с hdma на прием и на передачу данных как из, так и в BlackFin.
l502_hdma.h	Файл содержит описания функций для работы с интерфейсом HostDMA

5.4.2 Перечисления.

5.4.2.1 Флаги для передачи по t_hdma_send_flags

Тип: `t_hdma_send_flags`		
Описание: Флаги для передачи по `t_hdma_send_flags`		
Константа	**Значение**	**Описание**
`L502_HDMA_FLAGS_SEND_LAST`	`= 0x1`	Флаг, указывающий, что передается последний пакет данных. По завершению передачи будет остановлен поток и сгенерировано прерывания для ПК

5.4.3 Функции.

5.4.3.1 Обработка завершения передачи по HostDMA.

Формат: `void hdma_send_done(uint32_t *addr, uint32_t size)`
Описание: Функция вызывается из обработчика прерывания, когда завершилась передача блока данных по HDMA в ПК, поставленного до этого на передачу с помощью `hdma_send_req_start()`.
Параметры: **addr** — Адрес слова, сразу за последним переданным словом **size** — Размер переданных данных в 32-битных словах

5.4.3.2 Обработка завершения приема по HostDMA.

Формат: `void hdma_recv_done(uint32_t *addr, uint32_t size)`
Описание: Функция вызывается из обработчика прерывания, когда завершился прием блока данных по HDMA в ПК, поставленного до этого на передачу с помощью `hdma_recv_req_start()`. Функция просто обновляет счетчик принятых данных (а обработка будет уже из фоновой функции `stream_proc()`.
Параметры: **addr** — Адрес слова, сразу за последним принятым словом. **size** — Количество принятых 32-битных слов

5.4.3.3 Обработчик прерывания на завершения записи в память BF по HDMA.

Формат: `void hdma_isr(void)`
Описание: Обработчик вызывается по завершению приема блока по HostDMA. Обработчик выполняет установку необходимых флагов для разрешения приема следующего блока и, кроме того, проверяет наличие новой команды и завершения передачи или приема блока из потока данных

5.4.3.4 Обработчик прерывания на завершение чтения по HDMA.

Формат: `void hdma_rd_isr(void)`
Описание: Данный обработчик вызывается по завершению передачи блока данных по HostDMA. Выполняет только установку необходимых флагов для разрешения следующей передачи

5.4.3.5 Инициализация интерфейса HostDMA.

Формат: `void hdma_init(void)`
Описание: Настройка параметров HostDMA и инициализация неизменяемых полей дескрипторов для организации потока по данных по HostDMA

5.4.3.6 Запуск потока на передачу по HostDMA.

Формат: `void hdma_send_start(void)`
Описание: Функция сбрасывает логику обработки заданий на передачу данных по HostDMA и разрешает передачу. Должна вызываться до добавления первого задания с помощью `hdma_send_req_start()`

5.4.3.7 Останов потока на передачу по HostDMA.

Формат: `void hdma_send_stop(void)`
Описание: Запрет передачи по HostDMA с остановом всех текущих заданий

5.4.3.8 Запуск потока на прием по HostDMA.

Формат: `void hdma_recv_start(void)`
Описание: Функция сбрасывает логику обработки заданий на прием данных по HostDMA и разрешает прием. Должна вызываться до добавления первого задания с помощью `hdma_recv_req_start()`

5.4.3.9 Останов потока на прием по HostDMA.

Формат: `void hdma_recv_stop(void)`
Описание: Запрет приема по HostDMA с остановом всех текущих заданий

5.4.3.10 Получить количество свободных запросов на передачу.

Формат: `int hdma_send_req_rdy(void)`
Описание: Фунция позволяет узнать, сколько запросов можно еще поставить в очередь на передачу с помощью `hdma_send_start()`.
Возвращаемое значение: Количество запросов на передачу, которое можно поставить в очередь

5.4.3.11 Получить количество свободных запросов на прием

Формат: `int hdma_recv_req_rdy(void)`
Описание: Фунция позволяет узнать, сколько запросов можно еще поставить в очередь на прием с помощью `hdma_recv_start()`.
Возвращаемое значение: Количество запросов на прием, которое можно поставить в очередь

5.4.3.12 Поставить запрос на передачу по HostDMA.

Формат: `int hdma_send_req_start(const uint32_t *buf, uint32_t size, uint32_t flags)`
Описание: Функция ставит запрос на передачу указанных данных. Сами данные не копируются, т.е. буфер нельзя будет использовать до того, как данные не будут переданы! Для постановки запроса необходимо, чтобы был свободный дескриптор (можно узнать через `hdma_send_req_rdy()`)
Параметры: **buf** — Указатель на массив на передачу. **size** — Количество 32-битных слов на передачу **flags** — Флаги из `t_hdma_send_flags`
Возвращаемое значение: < 0 при ошибке, $>= 0$ - id передачи при успехе

5.4.3.13 Поставить запрос на передачу по HostDMA.

Формат: `int hdma_recv_req_start(uint32_t *buf, uint32_t size)`
Описание: Функция ставит запрос на прием данных в указанный буфер. Сами данные будут в буфере только по завершению запроса. Для постановки запроса необходимо, чтобы был свободный дескриптор (можно узнать через `hdma_recv_req_rdy()`)
Параметры: **buf** — Указатель на массив на передачу. **size** — Количество 32-битных слов на передачу
Возвращаемое значение: < 0 при ошибке, $>= 0$ - id передачи при успехе

5.5 Передача потока данных по SPORT0

5.5.1 Файлы.

Файл	Описание
l502_sport_tx.c	Файл содержит логику для управления передачей по SPORT0. Изначально DMA должен быть проинициализирован с помощью `sport_tx_init()`, Передача начинается с помщью `sport_tx_start_req()`. Можно поставить до 16 запросов одновременно. Остановить все текущие передачи можно с помощью `sport_tx_stop()`.
l502_sport_tx.h	Файл содержит описания функций для управления передачи потока по SPORT0

5.5.2 Глобавльные переменные

Переменная	**Тип**	**Описание**
`g_mode`	int	Режим работы - определяет, запущен ли синхронный сбор или тест

5.5.3 Функции.

5.5.3.1 Обработка завершения передачи по SPORT.

Формат: `void sport_tx_done(uint32_t *addr, uint32_t size)`
Описание: Функция вызывается из обработчика прерывания при завершении передачи блока данных по SPORT'у на цифровые выходы/ЦАП, поставленного до этого на передачу с помощью `sport_tx_start_req()`.
Параметры: **addr** — Адрес слова, сразу за последним переданным словом **size** — Размер переданных данных в 32-битных словах

5.5.3.2 Начальная инициализация канала DMA на передачу по SPORT.

Формат: `void sport_tx_init(void)`
Описание: Функция устанавливает параметры DMA которые не меняются при последующей работе прошивки

5.5.3.3 Останов сбора по SPORT0.

Формат: `void sport_tx_stop(void)`
Описание: Функция запрещает прием по SPORT0 и соответствующий канал DMA на прем

5.5.3.4 Получить количество свободных дескрипторов на передачу

Формат: `int sport_tx_req_rdy(void)`
Описание: Функция возвращает количество запросов, которое можно поставить в очередь на передачу с помощью `sport_tx_start_req()`.
Возвращаемое значение: Количество запросов на передачу, которое можно поставить в очередь

5.5.3.5 Поставить запрос на передачу по SPORT0.

Формат: `void sport_tx_start_req(uint32_t *buf, uint32_t size)`
Описание: Функция ставит запрос на передачу указанных данных. Сами данные не копируются, т.е. буфер нельзя будет использовать до того, как данные не будут переданы! Для постановки запроса необходимо, чтобы был свободный дескриптор (можно узнать через `sport_tx_req_rdy()`). При добавление первого дескриптора автомтически разрешается передача по DMA и SPORT0.
Параметры: **buf** — Указатель на массив на передачу. **size** — Количество 32-битных слов на передачу

5.5.3.6 Обработчик прерывания по SPORT0 на завершение передачи.

Формат: `void isr_sport_dma_tx(void)`
Описание: Прерывание возникает, когда был передан блок данных по SPORT0, соответствующий одному дескриптору. Функция рассчитывает количество переданных данных и вызывает `sport_tx_done()`. Также функция помечает дескриптор как свободный и при необходимости снова запускает DMA (если он был остановлен, но при этом уже добавлен новый дескриптор на передачу)

5.6 Прием потока данных по SPORT0

5.6.1 Файлы.

Файл	Описание
l502_sport_rx.h	Файл содержит описания функций для управления приемом потока по SPORT0

5.6.2 Функции.

5.6.2.1 Установка шага прерывания для према по SPORT0.

Формат: `int32_t sport_in_set_step_size(uint32_t size)`
Описание: Функция устанавливает шаг прерываний для DMA, использующегося для приема данных синхронного ввода. При этом размер шага должен быть как имнимум в 4 раза меньше размера буфера L502_SPORT_IN_BUF_SIZE. После установки шага определяется реально используемый размер буфера, как наибольшее число кратное шагу и не превышающее L502_SPORT_IN_BUF_SIZE.
Параметры: **size** — Размер шага прерывания в 32-битных словах
Возвращаемое значение: Код ошибки

5.6.2.2 Запуск сбора данных по SPORT0.

Формат: `void sport_rx_start(void)`
Описание: Функция настраивает DMA3 на режим автобуфера с 2D, размер шага выбирается равным f_sport_in_block_size. После чего разрешается канал DMA и прием по SPORT0

5.6.2.3 Останов сбора данных по SPORT0.

Формат: `void sport_rx_stop(void)`
Описание: Функция запрещает прием по SPORT0 и останавливает DMA

5.6.2.4 Обработчик прерывания по SPORT0 на прием.

Формат: `void isr_sport_dma_rx(void)`
Описание: Прерывание возникает, когда был принят блок данных по SPORT0. Обновляем указатель принятых данных и проверяем переполнение

5.7 Управление потоками сбора данных

5.7.1 Файлы.

Файл	Описание
l502_stream.c	Файл содержит функции обработки потоков от АЦП/DIN в ПК и от ПК в ЦАП/DOUT. Также в этот файл внесена реализация функций приема по SPORT0, так как они связаны с логикой управления потоков.
l502_stream.h	Файл содержит описания функций для работы с интерфейсом HostDMA

5.7.2 Глобавльные переменные

Переменная	Тип	Описание
g_stream_in_state	t_in_stream_state	Состояние потока на ввод
g_stream_out_state	t_out_stream_state	Состояние потока на вывод
g_mode	int	Режим работы - определяет, запущен ли синхронный сбор или тест
g_streams	int	Флаги, обозначающие какие потоки разрешены

5.7.3 Константы и макроопределения.

Константа	Значение	Описание
L502_SPORT_IN_BUF_SIZE	(2048*1024)	Размер буфера на прием данных по SPORT0 в 32-битных словах
L502_HDMA_OUT_BUF_SIZE	(1024*1024)	Размер буфера для приема данных по HostDMA на вывод в 32-битных словах
L502_DEFAULT_SPORT_RX_BLOCK_SIZE	(32*1024)	Шаг прерываний для приема данных по SPORT0 по-умолчанию

5.7.4 Перечисления.

5.7.4.1 Состояние потока на ввод

Тип: t_in_stream_state		
Описание: Состояние потока на ввод		
Константа	**Значение**	**Описание**
IN_STREAM_STOP	=0	Поток на ввод остановлен
IN_STREAM_RUN	=2	Поток на ввод в рабочем режиме
IN_STREAM_OV_ALERT	=4	Произошло переполнение - нужно сообщить о ошибке
IN_STREAM_ERR	=3	Поток остановлен из-за ошибки/переполнения

5.7.4.2 Состояние потока на вывод

Тип: t_out_stream_state		
Описание: Состояние потока на вывод		
Константа	**Значение**	**Описание**
OUT_STREAM_STOP	=0	Поток на вывод остановлен
OUT_STREAM_PRELOAD	=1	Идет предзагрузка данных - данные принимаются от ПК, но еще не запущен синхронных сбор/выдача
OUT_STREAM_RUN	=2	
OUT_STREAM_ERR	=3	Поток запущен на выдачу
OUT_STREAM_CYCLE	=4	Поток остановлен по ошибке (сейчас не используется)

5.7.5 Функции.

5.7.5.1 Начальная инициализация параметров для синхронных потоков

Формат: `void l502_stream_init(void)`
Описание:

5.7.5.2 Запуск предзагрузки данных на вывода

Формат: `int32_t stream_out_preload(void)`
Описание: Данная функция запускает предзагрузку данных потока на вывод. Используется, чтобы загрузить данные в буфер BlackFin до запуска синхронного ввода-вывода.
Возвращаемое значение: Код ошибки

5.7.5.3 Разрешение указанных синхронных потоков

Формат: `int32_t stream_enable(uint32_t streams)`
Описание:
Параметры: **streams** — Битовая маска из `t_l502_streams`, указывающая какие потоки должны быть разрешены (в дополнения к уже разрешенным)
Возвращаемое значение: Код ошибки

5.7.5.4 Запрещение указанных синхронных потоков

Формат: `int32_t stream_disable(uint32_t streams)`
Описание:
Параметры: **streams** — Битовая маска из `t_l502_streams`, указывающая какие потоки должны быть запрещены
Возвращаемое значение: Код ошибки

5.7.5.5 Запуск синхронного ввода-вывода

Формат: `int32_t streams_start(void)`
Описание: Функция запускает синхронный ввод-вывод платы. При этом начинается передача по всем ранее разрешенным потокам с помощью `stream_enable()`. После вызова этой функции изменять настройки модуля уже нельязя, однако можно дополнительно разрешать или запрещать потоки через `stream_enable()` или `stream_disable()`.
Возвращаемое значение: Код ошибки.

5.7.5.6 Останов синхронных потоков ввода-вывода.

Формат: `int32_t streams_stop(void)`
Описание: По этой функции останавливаются все синхронные потоки. Запрещается передача потоков по SPORT и по HostDMA
Возвращаемое значение: Код ошибки

5.7.5.7 Фоновая обработка потокой ввода-вывода

Формат: `void stream_proc(void)`
Описание: Функция переодически вызывается из основного цикла программы. При рабочем режиме, проверяются, есть ли необработанные данные пришедшие от АЦП/DIGIN и/или пришедшие от ПК данные на ЦАП или DOUT. При их наличии вызывается соответствующая функция пользовательской обработки данных. Также, если было переполнение и все данные до переполнения были обработаны, то в ПК посылается слово о том, что в этом месте произошло переполнение

5.7.5.8 Освобождение size слов из буфера приема по SPORT0.

Формат: `void stream_in_buf_free(uint32_t size)`
Описание: Функция помечает, что size слов из начала той части буфера SPORT0, в которую были приняты данные, но не освобождены, как освобожденные. Т.е. в эту область снова можно будет принимать данные со SPORT0. При этом надо всегда следить, чтобы количество освобожденных данных не привышало количество обработанных!
Параметры: **size** — Размер освобожденных данных в 32-битных словах

5.7.5.9 Освобождение size слов из буфера приема по HostDMA.

Формат: `void stream_out_buf_free(uint32_t size)`
Описание: Функция помечает, что size слов из начала той части буфера HostDMA, в которую были приняты данные от ПК, но не освобождены, как освобожденные. Т.е. в эту область снова можно будет принимать данные от ПК по HostDMA. При этом надо всегда следить, чтобы количество освобожденных данных не привышало количество обработанных!
Параметры: **size** — Размер освобожденных данных в 32-битных словах

5.7.5.10 Обработка завершения приема по HostDMA.

Формат: `void hdma_recv_done(uint32_t *addr, uint32_t size)`
Описание: Функция вызывается из обработчика прерывания, когда завершился прием блока данных по HDMA в ПК, поставленного до этого на передачу с помощью `hdma_recv_req_start()`. Функция просто обновляет счетчик принятых данных (а обработка будет уже из фоновой функции `stream_proc()`.
Параметры: **addr** — Адрес слова, сразу за последним принятым словом. **size** — Количество принятых 32-битных слов

5.7.5.11 Размер буфера на прием.

Формат: `uint32_t sport_in_buffer_size(void)`
Описание: Функция возвращает размер буфера на прием по SPORT0
Возвращаемое значение: размер буфера на прием в 32-битных словах

5.7.5.12 Установка шага прерывания для према по SPORT0.

Формат: `int32_t sport_in_set_step_size(uint32_t size)`
Описание: Функция устанавливает шаг прерываний для DMA, использующегося для приема данных синхронного ввода. При этом размер шага должен быть как имнимум в 4 раза меньше размера буфера `L502_SPORT_IN_BUF_SIZE`. После установки шага определяется реально используемый размер буфера, как наибольшее число кратное шагу и не превышающее `L502_SPORT_IN_BUF_SIZE`.
Параметры: **size** — Размер шага прерывания в 32-битных словах
Возвращаемое значение: Код ошибки

5.8 Настройки сбора данных модуля

5.8.1 Файлы.

Файл	Описание
l502_defs.h	Файл содержит определения, которые используются как программой BF, так и l502api, и кроме того видны программам, которые используют l502api
l502_params.h	Файл содержит определение структур, описывающих текущие настройки сбора данных для модуля и объявляет внешнюю переменную g_set, содержащую эти параметры
l502_params.c	Файл содержит функции по обработке команд от ПК на установку параметров конфигурации сбора данных и других параметров. Только в данном файле должны быть изменения полей структуры общих настроек - g_set. Так же здесь

5.8.2 Глобавльные переменные

Переменная	**Тип**	**Описание**
`g_set`	t_settings	Структура, содержащая текущие настройки сбора данных
`g_module_info`	t_module_info	Структура, содержащая информацию о модуле

5.8.3 Константы и макроопределения.

Константа	**Значение**	**Описание**
`L502_LTABLE_MAX_CH_CNT`	`256`	Максимальное количество логических каналов в таблице
`L502_ADC_RANGE_CNT`	`6`	Количество диапазон для измерения напряжений
`L502_ADC_SCALE_CODE_MAX`	`6000000`	Код АЦП, соответствующий максимальному значению шкалы
`L502_LCH_AVG_SIZE_MAX`	`128`	Максимальное значение для аппаратного устреднения по логическому каналу
`L502_ADC_FREQ_DIV_MAX`	`(1024*1024)`	Максимальное значения делителя частоты АЦП
`L502_DIN_FREQ_DIV_MAX`	`(1024*1024)`	Максимальное значение делителя частоты синхронного цифрового ввода
`L502_ADC_INTERFRAME_DELAY_MAX`	`(0x1FFFFF)`	Максимальное значение межкадровой задержки для АЦП

L502_DAC_RANGE	5.	Диапазон ЦАП в вольтах
L502_DAC_CH_CNT	2	Количество каналов ЦАП
L502_STREAM_IN_MSG_OVERFLOW	0x01010000	Слово в потоке, означающее, что произошло переполнение
L502_STREAM_OUT_WORD_TYPE_DOUT	0x0	Тип передаваемого отсчета - цифровой вывод
L502_STREAM_OUT_WORD_TYPE_DAC1	0x40000000	Тип передаваемого отсчета - Код для 1-го канала ЦАП
L502_STREAM_OUT_WORD_TYPE_DAC2	0x80000000	Тип передаваемого отсчета - Код для 2-го канала ЦАП

5.8.4 Перечисления.

5.8.4.1 Флаги для управления цифровыми выходами.

Тип: t_l502_digout_word_flags		
Описание: Флаги управления цифровыми выходами. Могут быть объединены через логическое "ИЛИ" со значениями цифровых выходов при асинхронном выводе с помощью L502_AsyncOutDig() или переданы в L502_PrepareData() при синхронном выводе.		
Константа	**Значение**	**Описание**
L502_DIGOUT_WORD_DIS_H	= 0x00020000	Запрещение (перевод в третье состояние) старшей половины цифровых выходов
L502_DIGOUT_WORD_DIS_L	= 0x00010000	Запрещение младшей половины цифровых выходов

5.8.4.2 Константы для выбора опорной частоты

Тип: t_l502_ref_freq		
Описание: Константы для выбора опорной частоты		
Константа	**Значение**	**Описание**
L502_REF_FREQ_2000KHZ	= 2000000	Частота 2МГц
L502_REF_FREQ_1500KHZ	= 1500000	Частота 1.5МГц

5.8.4.3 Диапазоны измерения для канала АЦП

Тип: t_l502_adc_range		
Описание: Диапазоны измерения для канала АЦП		
Константа	**Значение**	**Описание**
L502_ADC_RANGE_10	= 0	Диапазон +/-10V
L502_ADC_RANGE_5	= 1	Диапазон +/-5V
L502_ADC_RANGE_2	= 2	Диапазон +/-2V
L502_ADC_RANGE_1	= 3	Диапазон +/-1V
L502_ADC_RANGE_05	= 4	Диапазон +/-0.5V
L502_ADC_RANGE_02	= 5	Диапазон +/-0.2V

5.8.4.4 Режим измерения для логического канала

Тип: t_l502_lch_mode		
Описание: Режим измерения для логического канала		
Константа	**Значение**	**Описание**
L502_LCH_MODE_COMM	= 0	Измерение напряжения относительно общей земли
L502_LCH_MODE_DIFF	= 1	Дифференциальное измерение напряжения
L502_LCH_MODE_ZERO	= 2	Измерение собственного нуля

5.8.4.5 Режимы синхронизации.

Тип: t_l502_sync_mode		
Описание: Режимы задания источника частоты синхронизации и признака начала синхронного ввода-вывода		
Константа	**Значение**	**Описание**
L502_SYNC_INTERNAL	= 0	Внутренний сигнал
L502_SYNC_EXTERNAL_MASTER	= 1	От внешнего мастера по разъему синхронизации
L502_SYNC_DI_SYN1_RISE	= 2	По фронту сигнала DI_SYN1
L502_SYNC_DI_SYN2_RISE	= 3	По фронту сигнала DI_SYN2
L502_SYNC_DI_SYN1_FALL	= 6	По спаду сигнала DI_SYN1
L502_SYNC_DI_SYN2_FALL	= 7	По спаду сигнала DI_SYN2

5.8.4.6 Флаги для обозначения синхронных потоков данных

Тип: t_l502_streams		
Описание: Флаги для обозначения синхронных потоков данных		
Константа	**Значение**	**Описание**
L502_STREAM_ADC	= 0x01	Поток данных от АЦП
L502_STREAM_DIN	= 0x02	Поток данных с цифровых входов
L502_STREAM_DAC1	= 0x10	Поток данных первого канала ЦАП
L502_STREAM_DAC2	= 0x20	Поток данных второго канала ЦАП
L502_STREAM_DOUT	= 0x40	Поток данных на цифровые выводы
L502_STREAM_ALL_IN	= L502_STREAM_ADC | L502_STREAM_DIN	Объединение всех флагов, обозначающих потоки данных на ввод
L502_STREAM_ALL_OUT	= L502_STREAM_DAC1 | L502_STREAM_DAC2 | L502_STREAM_DOUT	Объединение всех флагов, обозначающих потоки данных на вывод

5.8.4.7 Режим работы модуля L502

Тип: t_l502_mode		
Описание: Режим работы модуля L502		
Константа	**Значение**	**Описание**
L502_MODE_FPGA	= 0	Все потоки данных передаются через ПЛИС минуя сигнальный процессор BlackFin
L502_MODE_DSP	= 1	Все потоки данных передаются через сигнальный процессор, который должен быть загружен прошивкой для обработки этих потоков
L502_MODE_DEBUG	= 2	Отладочный режим

5.8.4.8 Номера каналов ЦАП.

Тип: t_l502_dac_ch		
Описание: Номер каналов ЦАП для указания в L502_AsyncOutDac()		
Константа	**Значение**	**Описание**
L502_DAC_CH1	= 0	Первый канал ЦАП
L502_DAC_CH2	= 1	Второй канал ЦАП

5.8.4.9 Флаги, описывающие модуль

Тип: t_l502_devinfo_flags		
Описание: Флаги, описывающие модуль		
Константа	**Значение**	**Описание**
L502_DEVFLAGS_DAC_PRESENT	= 0x0001	Признак наличия ЦАП
L502_DEVFLAGS_GAL_PRESENT	= 0x0002	Признак наличия гальваноразвязки
L502_DEVFLAGS_BF_PRESENT	= 0x0004	Признак наличия сигнального процессора
L502_DEVFLAGS_FLASH_DATA_VALID	= 0x00010000	Признак, что во Flash-памяти присутствует информация о модуле
L502_DEVFLAGS_FLASH_ADC_CALIBR_VALID	= 0x00020000	Признак, что во Flash-памяти присутствуют действительные калибровочные коэффициенты АЦП
L502_DEVFLAGS_FLASH_DAC_CALIBR_VALID	= 0x00040000	Признак, что во Flash-памяти присутствуют действительные калибровочные коэффициенты ЦАП

5.8.5 Функции.

5.8.5.1 Запись параметров сбора в регистры ПЛИС

Формат: `int32_t configure(void)`
Описание: Функция выполняет запись всех параметров из структуры `g_set` в регистры ПЛИС. Функция может вызываться только когда сбор данных остановлен.
Возвращаемое значение: Код ошибки

5.8.5.2 Установка количества логических каналов

Формат: `int32_t params_set_lch_cnt(uint32_t lch_cnt)`
Описание: Проверка и запись в поле g_set.lch_cnt значение кол-ва каналов в логической таблице АЦП.
Параметры: **lch_cnt** — Количество логических каналов (от 1 до `L502_LTABLE_MAX_CH_CNT`)
Возвращаемое значение: Код ошибки

5.8.5.3 Установить параметры логического канала

Формат: `int32_t params_set_lch(uint32_t index, uint32_t ch, t_l502_lch_mode mode, t_l502_adc_range range, uint32_t avg, uint32_t flags)`
Описание: Функция проверяет входные параметры и записывает их в соответствующее поле таблицы g_set.lch[]
Параметры: **index** — Номер логического канала [0, L502_LTABLE_MAX_CH_CNT-1] **ch** — Номер физического канала (от 0 до 15 или 31) **mode** — Режим измерения для данного лог. канала **range** — Диапазон измерения для данного лог. канала **avg** — Коэф. усреднения по данному лог. каналу **flags** — Дополнительные флаги
Возвращаемое значение: Код ошибки

5.8.5.4 Установка делителя частоты АЦП

Формат: `int32_t params_set_adc_freq_div(uint32_t div)`
Описание: Установка делителя частоты АЦП
Параметры: **div** — Значение делителя
Возвращаемое значение: Код ошибки

5.8.5.5 Установка значения межкадровой задержки

Формат: `int32_t params_set_adc_interframe_delay(uint32_t delay)`
Описание: Установка значения межкадровой задержки
Параметры: **delay** — Значение межкадровой задержки (от 0 до L502_ADC_INTERFRAME_DELAY_MAX)
Возвращаемое значение: Код ошибки

5.8.5.6 Установка значения опорной частоты

Формат: `int32_t params_set_ref_freq(uint32_t freq_code)`
Описание: Установка значения опорной частоты
Параметры: **freq_code** — Значение частоты. Для внутренней может быть только L502_REF_FREQ_2000KHZ или L502_REF_FREQ_1500KHZ
Возвращаемое значение: Код ошибки

5.8.5.7 Установка источника опроной частоты синхронизации

Формат: `int32_t params_set_sync_mode(t_l502_sync_mode sync_mode)`
Описание: Установка источника опроной частоты синхронизации
Параметры: **sync_mode** — Значение из t_l502_sync_mode
Возвращаемое значение: Код ошибки

5.8.5.8 Установка источника синхронизации старта сбора данных

Формат: `int32_t params_set_sync_start_mode(t_l502_sync_mode sync_mode)`
Описание: Установка источника синхронизации старта сбора данных
Параметры: **sync_mode** — Значение из `t_l502_sync_mode`
Возвращаемое значение: Код ошибки

5.8.5.9 Установка делителя частоты синхронного ввода цифровых линий

Формат: `int32_t params_set_din_freq_div(uint32_t div)`
Описание: Установка делителя частоты синхронного ввода цифровых линий
Параметры: **div** — Значение делителя
Возвращаемое значение: Код ошибки

5.8.5.10 Установка делителя частоты вывода на ЦАП

Формат: `int32_t params_set_dac_freq_div(uint32_t div)`
Описание: Установка делителя частоты вывода на ЦАП
Параметры: **div** — Значение делителя (1 или 2)
Возвращаемое значение: Код ошибки

5.8.6 Структуры.

5.8.6.1 Калибровочные коэффициенты ЦАП

Тип: `t_dac_cbr_coef`		
Описание: Калибровочные коэффициенты ЦАП		
Поле	**Тип**	**Описание поля**
`offs`	float	Смещение нуля
`k`	float	Коэффициент шкалы

5.8.6.2 Информация о используемом модуле

Тип: t_module_info		
Описание: Информация о используемом модуле		
Поле	**Тип**	**Описание поля**
devflags	uint32_t	Флаги с информацией о наличии опций и сост. модуля
fpga_ver	uint16_t	Версия FPGA
plda_ver	uint8_t	Версия PLDA
dac_cbr	t_dac_cbr_coef	Коэффициенты ЦАП

5.8.6.3 Параметры логического канала

Тип: t_lch		
Описание: Параметры логического канала		
Поле	**Тип**	**Описание поля**
phy_ch	uint8_t	Номер физического канала
mode	uint8_t	Режим сбора
range	uint8_t	Диапазон
avg	uint8_t	Коэф. усреднения
flags	uint32_t	Доп. флаги

5.8.6.4 Настройки сбора данных

Тип: t_settings		
Описание: Настройки сбора данных		
Поле	**Тип**	**Описание поля**
lch	t_lch	Таблица логических каналов
lch_cnt	uint16_t	Количество каналов в лог. таблице
adc_freq_div	uint32_t	Делитель частоты АЦП
din_freq_div	uint32_t	Делитель частоты для цифрового входа
adc_frame_delay	uint32_t	Межкадровая задержка
ref_freq	uint32_t	Значение опорной частоты (2 или 1.5 МГц) или внешняя
dac_freq_div	uint16_t	Делитель частоты ЦАП (1 или 2)
sync_mode	uint8_t	Режим синхронизации для опорной частоты
sync_start_mode	uint8_t	Режим запуска сигнала синхронизации

5.9 Доступ к регистрам ПЛИС

5.9.1 Файлы.

Файл	Описание
l502_fpga.h	Файл содержит функции для записи и чтения регистров ПЛИС по интерфейсу АЦП
l502_fpga.c	Файл содержит логику передачи команд для чтения/записи регистров ПЛИС по SPI. Изначально SPI должен быть проинициализирован с помощью `fpga_spi_init()`. После этого можно осуществлять запись с помощью `fpga_reg_write()`, а чтение с помощью `fpga_reg_read()`.

5.9.2 Функции.

5.9.2.1 Инициализация SPI интерфейса

Формат: `void fpga_spi_init(void)`
Описание: Инициализация SPI интерфейса

5.9.2.2 Запись регистра в регистр ПЛИС значения по интерфейсу SPI

Формат: `void fpga_reg_write(uint16_t addr, uint32_t value)`
Описание: Запись регистра в регистр ПЛИС значения по интерфейсу SPI
Параметры: **addr** — Адрес регистра ПЛИС **value** — Записываемое значение

5.9.2.3 Чтение значения из регистра ПЛИС по интерфейсу SPI

Формат: `uint32_t fpga_reg_read(uint16_t addr)`
Описание: Чтение значения из регистра ПЛИС по интерфейсу SPI
Параметры: **addr** — Адрес регистра ПЛИС
Возвращаемое значение: Прочитанное значение

5.10 Асинхронный ввод-вывод

5.10.1 Файлы.

Файл	Описание
l502_async.h	Файл содержит описания функций асинхронного ввода-вывода
l502_async.c	Файл содержит реализацию функций для асинхронного ввода/вывода (пока только вывода)

5.10.2 Константы и макроопределения.

Константа	Значение	Описание
`L502_DAC_CH1`	`0`	Первый канал ЦАП при выводе через `async_dac_out()`
`L502_DAC_CH2`	`1`	Второй канал ЦАП при выводе через `async_dac_out()`

5.10.3 Функции.

5.10.3.1 Асинхронный вывод на один из каналов ЦАП

Формат: `void async_dac_out(uint8_t ch, int32_t val)`

Описание: Функция выполняет асинхронный вывод кода на ЦАП

Параметры:

ch — Канал ЦАП (`L502_DAC_CH1` или `L502_DAC_CH2`)

val — Код ЦАП (действительны только младшие 16 бит)

5.10.3.2 Асинхронный вывод на цифровые линии

Формат: `void async_dout(uint32_t val, uint32_t msk)`

Описание: Функция выполняет асинхронный вывод кода на цифровые линии с возможностью указать линии, которые не должны изменяться

Параметры:

val — биты 0-15 - значения линий на вывод + возможно перевести половину портов в 3-е состояние с помощью флагов из `t_l502_digout_word_flags`

msk — Если в маске установлены некоторые биты, то соответствующие биты из val не влияют, а сохраняются предыдущие значения