УДК 681.3.06 (075)
У912

КАФЕДРА МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ И ПРИМЕНЕНИЯ ЭВМ

№ 3451-3

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Технологический институт
Федерального государственного образовательного
учреждения высшего профессионального
образования
"Южный федеральный университет"

Н.Ш. Хусаинов

Учебно-методическое пособие
по курсу

АРХИТЕКТУРА И ПРОГРАММИРОВАНИЕ СИГНАЛЬНЫХ ПРОЦЕССОРОВ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРОЦЕССОРОВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

Часть 3

Для студентов специальностей

230105 Программное обеспечение вычислительной техники и автоматизированных систем

010503 Математическое обеспечение и администрирование информационных систем

Таганрог 2009

УДК 681.3.06 (075.8) + 681.325.5. (075.8)

Хусаинов Н.Ш. Учебно-методическое пособие по курсу “Архитектура и программирование сигнальных процессоров". Разработка программного обеспечения процессоров цифровой обработки сигналов. Часть 3. – Таганрог: Изд-во ТТИ ЮФУ, 2009. 86с.

Пособие посвящено вопросам разработки программного обеспечения процессоров цифровой обработки сигналов (ЦОС). Рассматривается архитектура оценочной платы EZ-KIT Lite на базе процессора SHARC ADSP-21160, особенности ее подключения и инсталляции для использования в рамках среды разработки VisualDSP++ 4.5.

Значительное внимание уделяется описанию процесса загрузки пользовательской программы во внутреннюю память процессора при включении питания. Рассматривается алгоритм функционирования загрузочного ядра Boot Kernel, управляющего процессом загрузки. Затрагиваются также вопросы записи программы в энергонезависимую Flash-память с использованием утилиты Flash Programmer среды разработки VisualDSP++.

Отдельный раздел посвящен кодеку AD1881 SoundMax стандарта AC'97, установленному на плате EZ-KIT Lite и обеспечивающему ввод/вывод аудиосигнала. Детально изучаются вопросы управления кодеком и приемы программирования для обмена данными между процессором и кодеком.

Пособие предназначено для студентов специальностей 230105, 010503, изучающих курс "Архитектура и программирование сигнальных процессоров". Представляет интерес для инженеров, разработчиков и программистов в области ЦОС, параллельного программирования, машинно-ориентированного программирования для встроенных систем и систем реального времени, занимающихся проектированием систем ЦОС и разработкой программного обеспечения для них.

Ил. 33. Библиогр.: 20 назв.

Рецензент В.Е. Золотовский, д-р техн. наук, профессор кафедры ВТ ТТИ ЮФУ.

Хусаинов Наиль Шавкятович

Учебно-методическое пособие

по курсу

АРХИТЕКТУРА И ПРОГРАММИРОВАНИЕ

СИГНАЛЬНЫХ ПРОЦЕССОРОВ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРОЦЕССОРОВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

Часть 3

Для студентов специальностей 230105, 010503

Ответственный за выпуск Хусаинов Н.Ш.
Редактор Белова Л.Ф.
Корректор Селезнева Н.И.

ЛР № 020565 от 23.06.1997г. Подписано к печати . .2009г.
Формат 60x84 1/16. Бумага офсетная.
Офсетная печать. Усл.п.л. — 5,3. Уч.-изд. л. — 5,0.
Заказ № Тираж 50 экз.

"С"

Издательство Технологического института Южного федерального университета
347928, ГСП 17А, Таганрог, Некрасовский, 44
Типография Технологического института Южного федерального университета
347928, ГСП 17А, Таганрог, Энгельса, 1

СОДЕРЖАНИЕ

1. АРХИТЕКТУРА ОЦЕНОЧНОЙ ПЛАТЫ ADSP21160 EZ-KIT LITE

1.1. Состав и размещение элементов на плате

Оценочная плата ADSP-21160 EZ-KIT Lite обеспечивает простой способ изучения и оценки возможностей процессоров семейства SHARC, а также разработки собственных приложений для высокоэффективных DSP-процессоров. Плата EZ-KIT Lite представляет собой полнофункциональную расширяемую аппаратную платформу на базе сигнального процессора и простейшей периферии для ввода/вывода данных в реальном масштабе времени. Основное назначение EZ-KIT Lite – обучение программированию для DSP.

Плата ADSP-21160 EZ-KIT Lite ориентирована на разработку приложений, выполняющих обработку 16-битового стереозвука. Состав платы:

- процессор Analog Devices ADSP-21160 DSP с частотой ядра 80 МГц;
- 40 МГЦ осциллятор с переключателем множителя частоты;
- 16-битовый стереокодек AD1881 SoundMAX стандарта AC’97;
- разъем стереоджек 3,5 мм линейного входа (LINE IN) или микрофонного входа (MIC IN) в зависимости от состояния переключателей;
- разъем стереоджек 3.5 мм линейного выхода (LINE OUT);
- Flash память EPROM объемом 4 Мбит (512 К х 8 бит);
- 2 банка расширенной памяти SBSRAM, 64 К х 32 бит каждый (всего 4 Мбит);
- 3 клавиши прерываний IRQs;
- 3 светодиода, соединенных с ножками FLAGs процессора;
- интерфейс USB порта для подключения к персональному компьютеру (ПК) с целью контроля и отладки приложения;
- внешние разъемы для линк-портов № 0 и 4;
- внешний разъем для последовательного порта № 0;
- 14-контактный разъем JTAG для подключения эмулятора.

Плата может работать автономно или в соединении через порт с ПК. Программа слежения (монитор), запущенная на DSP, вместе с основной программой отладки (в рамках среды VisualDSP++), запущенной на ПК, позволяет в интерактивном режиме загружать пользовательскую программу и просматривать содержимое регистров и ячеек памяти ADSP-21160.

Архитектура оценочной платы EZ-KIT Lite приведена на рис. 1.1.

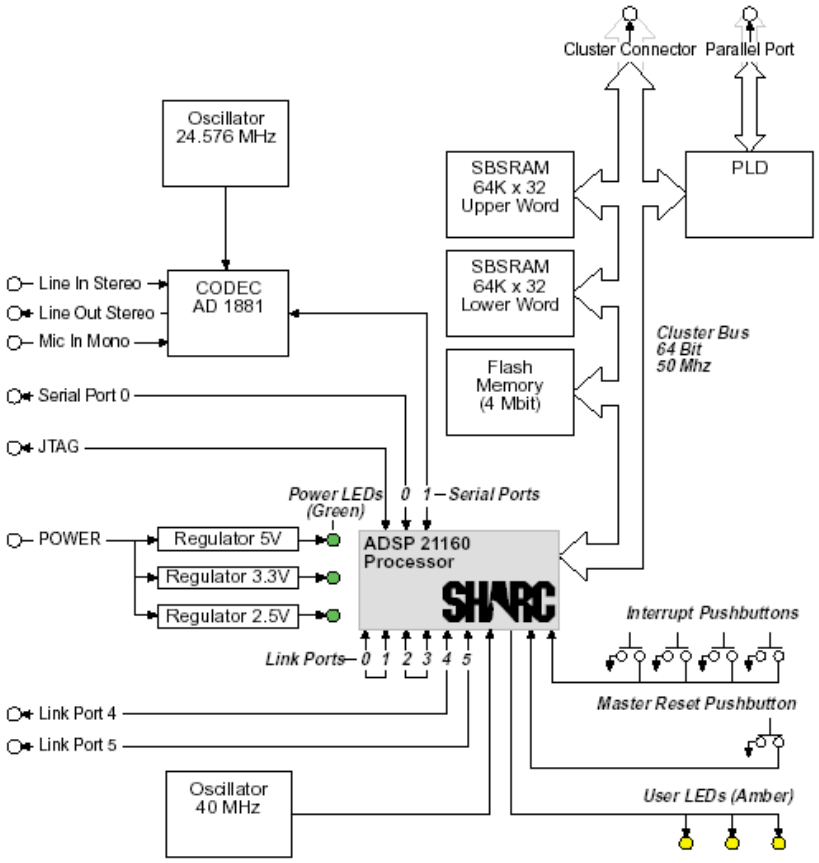


Рис. 1.1. Архитектура платы ADSP-21160 EZ-KIT Lite

Организация взаимодействия между платой, ПК и пользователем, управление выполнением программой, конфигурирование платы осуществляется с использованием разъемов (интерфейсов и портов), переключателей, светодиодов и кнопок.

Разъемы (интерфейсы и порты). На рис. 1.2 показано расположение разъемов для подключения к плате внешних устройств и питания:

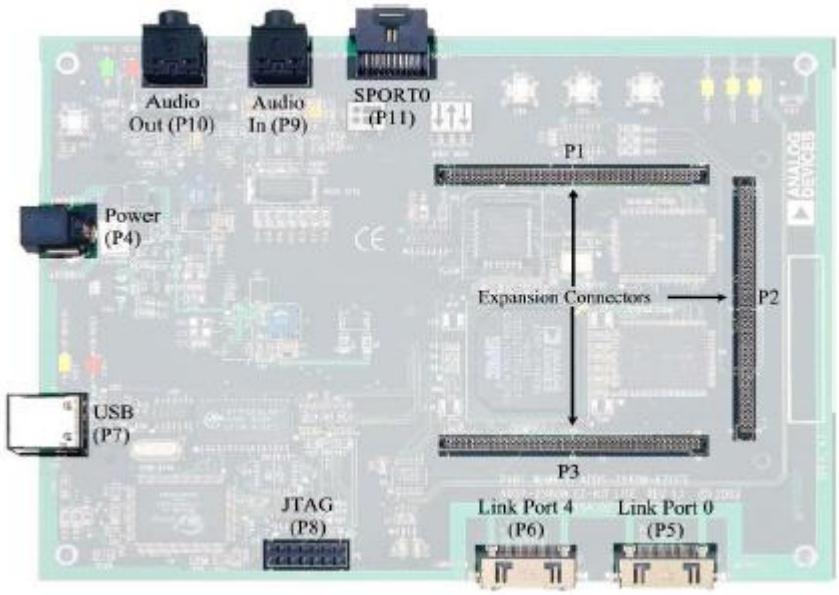


Рис. 1.2. Расположение разъемов на плате EZ-KIT Lite

- разъемы расширения (P1,P2,P3) позволяют подключать внешние устройства для анализа большинства сигналов на внешних линиях процессора (адресных линиях, линиях данных, сигналов FLAG3-0, сигналов IRQ2-0, сигналов линк-порта № 2 и др.) для отладки приложения;

- разъем питания (P4) обеспечивает подключение питания, необходимого для работы платы;

- разъемы линк-портов (P5, P6) для организации взаимодействия между несколькими процессорами (несколькими платами EZ-KIT Lite). Кабель для линк-интерфейса имеет специальные 26-контактные разъемы;

- USB-разъем (P7) используется для взаимодействия между средой разработки VisualDSP++ и системной программой-монитором, исполняемой на процессоре при отладке приложения;

- разъем JTAG (P8) предназначен для подключения средства аппаратной отладки в реальном масштабе времени – эмулятора. Когда эмулятор соединен через JTAG, интерфейс отладки USB недоступен;

- звуковые разъемы (P9-10) для подключения входных и выходных аудиокабелей;

- разъем SPORT0 (P11) для взаимодействия с дополнительными внешними устройстами через последовательный порт SPORT0.

Внешний порт (EP) процессора соединяется с 512 Кбайт (64 К x 32 бита x 2 чипа) SBSRAM. SBSRAM соединяется с переключателем выбора памяти (~MS1), обеспечивая интерфейс 64-битовой памяти.

Внешний порт также связан с микросхемой Flash-памяти объемом 512 Кбайт (512 К x 8 битов). Flash-память соединяется с процессором и по линии ~BMS, и по линии ~MS0. Такая организация соединения позволяет процессору загружаться из Flash-памяти (выставляя сигналы на линию ~BMS), и программировать Flash-память (используя линию ~MS0).

SPORT0 соединяется с кодеком AD1881A SoundMAX (U13). На вход кодеку можно подавать стереосигнал (LINE IN) или моносигнал с микрофона (MIC IN). Тип входного сигнала (разъем P9) выбирается переключателем JP1. SPORT0 также подсоединен к внешнему разъему (P11). Перед началом работы кодек должен быть перезагружен путем сброса-восстановления сигнала на линии FLAG3.

Порт эмуляции JTAG позволяет эмулятору получать доступ к внутренней и внешней памяти процессора.

Переключатели. Схема размещения переключателей на плате EX-KIT Lite приведена на рис. 1.3:

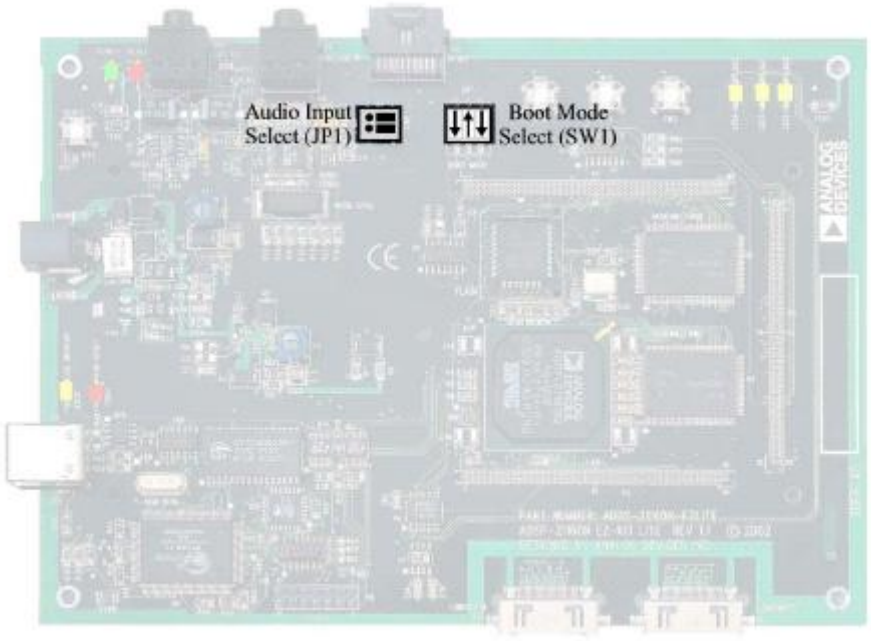


Рис. 1.3. Расположение переключателей на плате EZ-KIT Lite

К аудиокодеку AD1881 через входной разъем (P9) может быть подключен микрофон (MIC IN) или кабель от внешнего аудиоустройства (LINE IN). Выбор типа входного сигнала зависит от состояния перемычки (JP1), как показано на рис. 1.4. Входной сигнал по умолчанию – LINE IN.

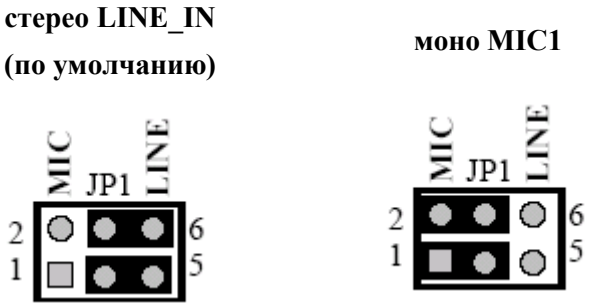


Рис. 1.4. Установки перемычки звукового входа

Переключатель выбора способа загрузки (SW1) определяет источник загрузки ADSP-21160. В табл. 1.1 показаны варианты состояния переключателя способов загрузки. По умолчанию положение переключателей соответствует загрузке из Flash-памяти.

Таблица 1.1

Зависимость способа загрузки от положения переключателей SW1

~BMS контакт 1	*LBOOT контакты 2*	*EBOOT контакты 3*	*Способ загрузки*
Off (выход)	On	Off	Загрузка из 8-битовой Flash-памяти
Off (вход)	On	On	Загрузка с хост-процессора
Off (вход)	Off	On	Загрузка с линк-порта
On (вход)	On	On	Нет загрузки (выполняется с внешней памяти)
On (вход)	Off	On	Зарезервировано
X (вход)	Off	Off	Зарезервировано

Светодиоды и кнопки. На рис. 1.5 показано местоположение светодиодов и кнопок:

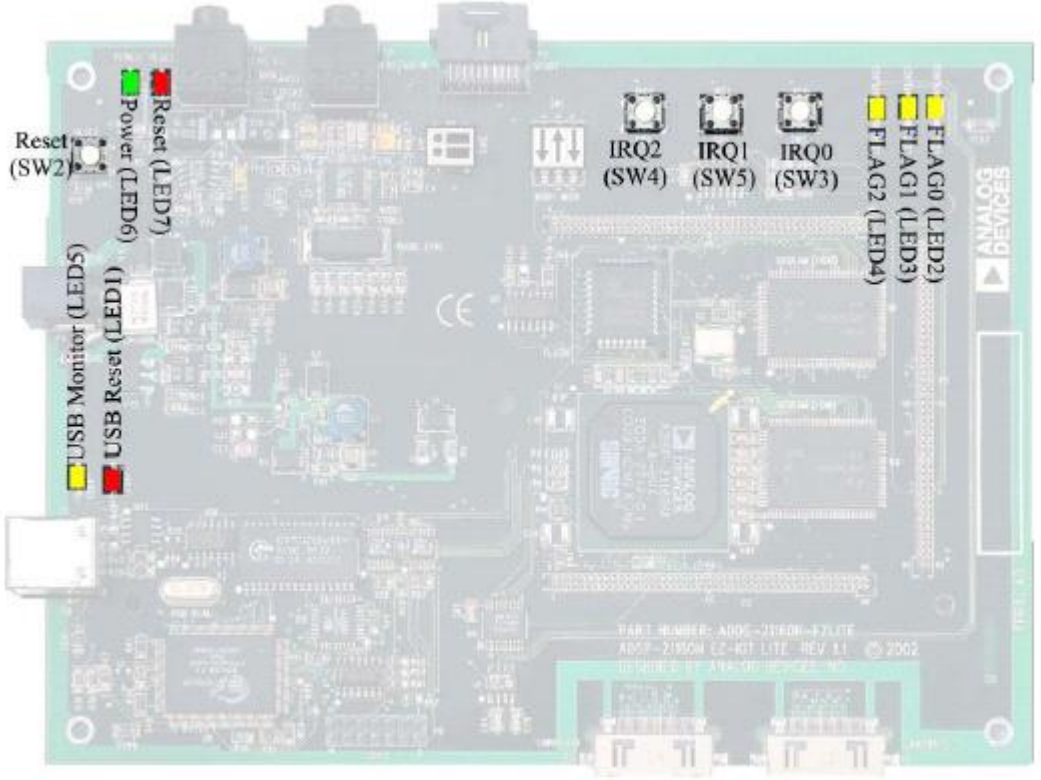


Рис. 1.5. Расположение светодиодов и кнопок на плате EZ-KIT Lite

Когда зажжен светодиод LED1, выполняется процесс перезагрузки всех интегральных схем на плате. Когда зажжен LED7, выполняется сброс чипа USB-интерфейса (U11). USB-интерфейс сбрасывается после включения питания или когда USB-связь не инициализировалась.

*С*ветодиоды LED2..LED4 соединены с ножками FLAG0..FLAG2 процессора соответственно. Светодиоды активны, когда на выходе процессора стоит “1”.

Светодиод монитора USB (LED5) указывает, что USB-связь инициализировалась успешно, и возможно установить соединение с DSP-процессором на плате посредством соответствующей сессии VisualDSP ++. Если светодиод не зажжен приблизительно через 15 секунд после того, как кабель USB соединен с платой, необходимо попробовать отключить и снова подключить питание к плате и/или переустановить USB-драйвер.

Когда зажжен зеленый светодиод LED6, это означает, что на плату подано питание.

Кнопка сброса платы (SW2) сбрасывает все интегральные схемы на плате. Этот сброс не затрагивает чип интерфейса USB (U11), пока связь не инициализирована через ПК. После инициализации связи USB единственная возможность сбросить USB – это выключить питание платы.

Три кнопки прерываний SW3..SW5 соединены соответственно с ножками процессора IRQ3..IRQ2. Нажатие на кнопку приводит к генерации сигнала соответствующего прерывания.

1.2. Интерфейс оценочной платы EZKIT-LITE и среды разработки VisualDSP++

1.2.1. Установка и подключение платы EZ-KIT Lite

При инсталляции среды разработки VisualDSP++ выполняется копирование в подкаталог \Setup каталога установки VisualDSP++ файлов, необходимых для работы драйвера оценочной платы EZ-KIT Lite. При установке VisualDSP++ плата должна быть отсоединена от компьютера.

Перед подключением платы необходимо установить все переключатели на плате в положение "по умолчанию" и затем подключить питание. После подачи питания на плату светодиод LED6 должен загореться зеленым цветом, а индикаторы LED1 (схема USB-интерфейса) и LED7 (сброс) – мигнуть и погаснуть. Затем следует соединить кабелем USB-порт на ПК и разъем P7 на плате ADSP-21160 EZ-KIT Lite.

После подключения платы к компьютеру ОС Windows автоматически обнаружит новое устройство и установит для него необходимый драйвер (с помощью стандартного мастера).

1.2.2. Создание и выбор целевой платформы на базе EZ-KIT Lite для среды разработки VisualDSP++

После запуска VisualDSP++ необходимо создать сессию для отладки пользовательских приложений с использованием EZ-KIT Lite. Начиная с VisualDSP++ версии 4.5, для создания сессии используется мастер, встроенный в среду разработки. Он вызывается при выборе пункта меню Session->New Session.

На экране появляется окно, позволяющее выбрать тип процессора (см. рис. 1.6).

Список доступных семейств процессоров (Processor Family) показывается с учетом установленных лицензий на среду разработки. После выбора конкретной модели процессора, установленного на оценочной плате (SHARC ADSP-21160), следует нажать кнопку "Next>".

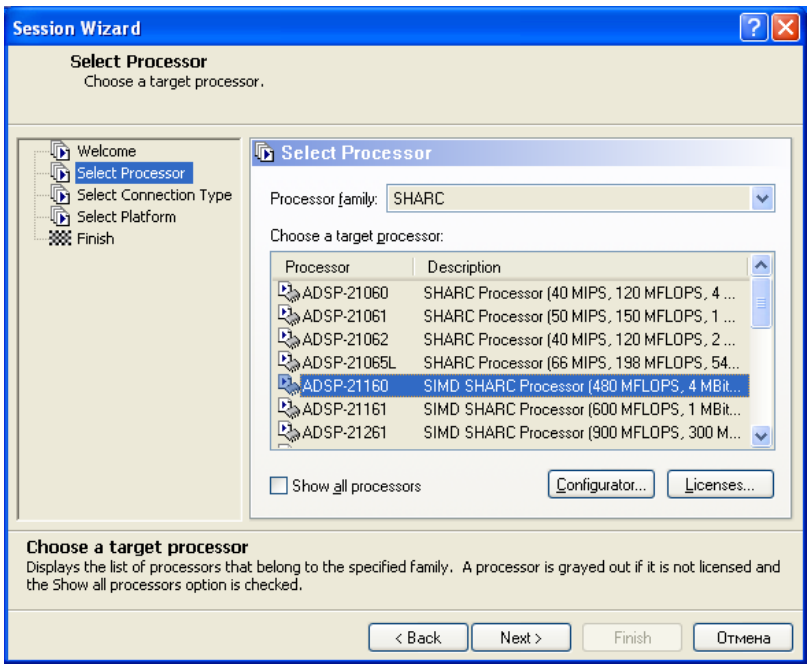


Рис. 1.6. Окно выбора процессора для создания сессии отладки

Далее появляется вкладка, позволяющая выбрать тип отладочной сессии (см. рис. 1.7).

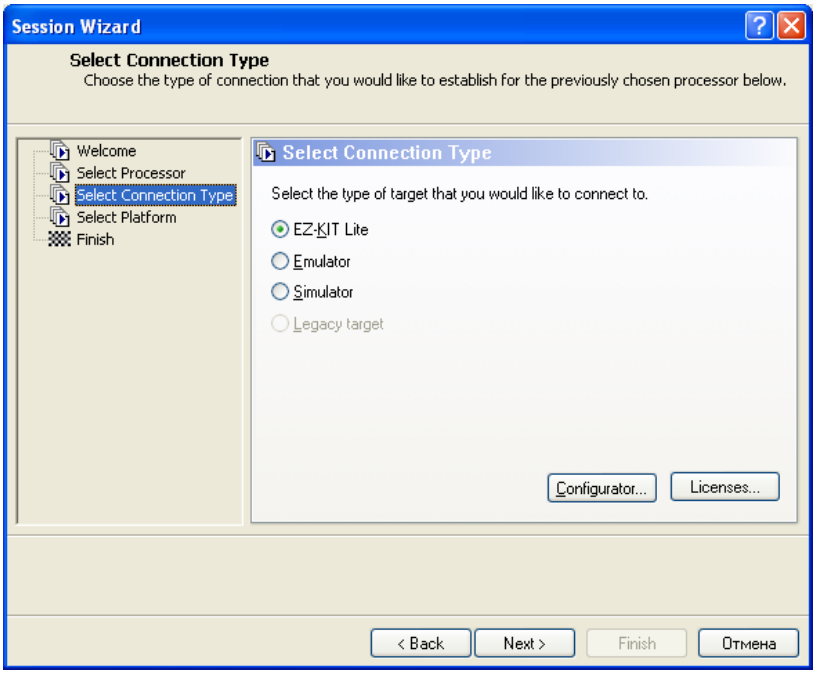


Рис. 1.7. Окно выбора типа отладочной сессии

Как известно, среда VisualDSP++ поддерживает три вида отладки программных приложений: на базе симулятора (без подключения DSP-процессора), с использованием типовой оценочной платы EZ-KIT Lite, и с использованием эмулятора для анализа выполнения программы в реальном масштабе времени на оценочной или целевой архитектуре.

После выбора варианта выполнения и отладки пользовательской программы на плате EZ-KIT Lite и нажатия кнопки "Next>" мастер позволяет выбрать имя для созданной сессии (см. рис. 1.8).

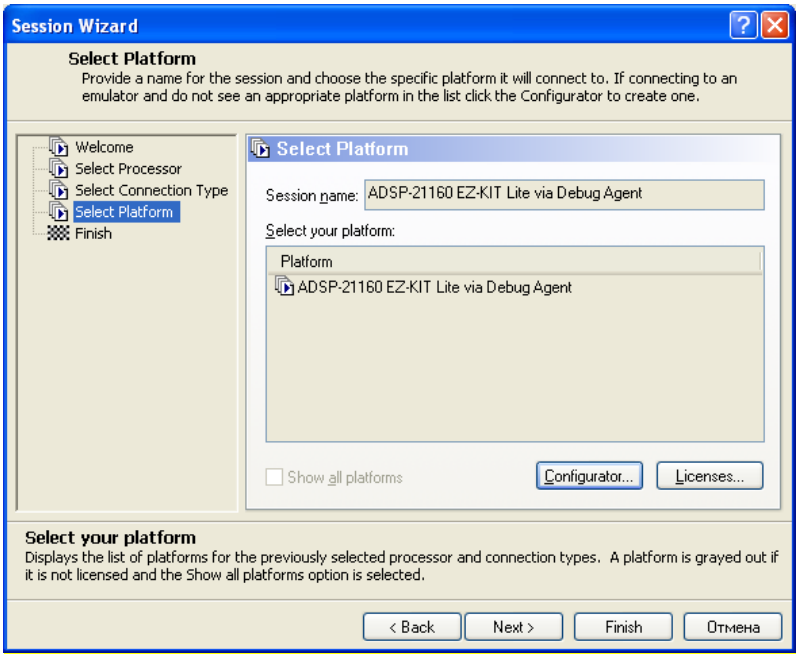


Рис. 1.8. Определение имени отладочной сессии

По умолчанию сессии присваивается имя "ADSP-2160 EZ-KIT Lite via Debug Agent". На этом процесс создания сессии завершается (см. рис. 1.9).

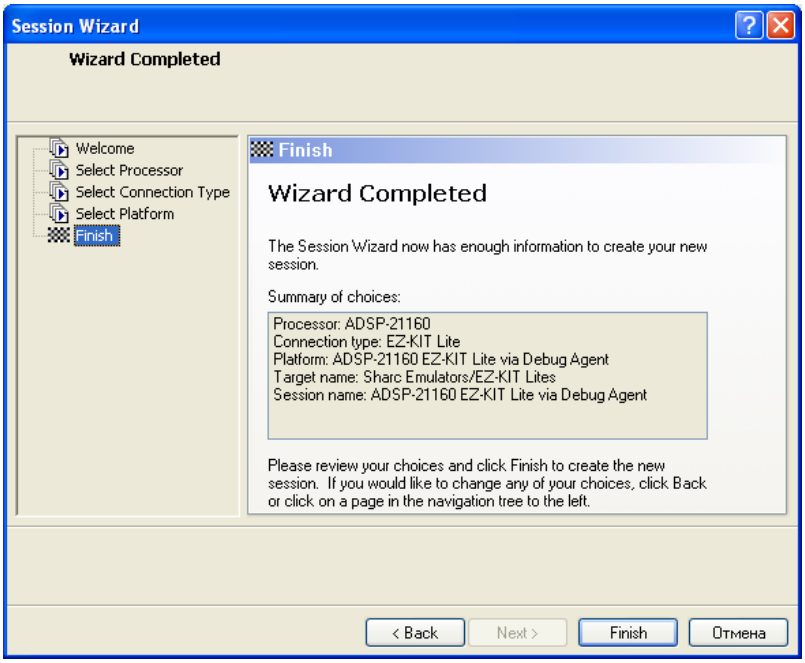


Рис. 1.9. Последний шаг создания отладочной сессии

Если необходимая отладочная сессия уже создана, то ее выбор осуществляется путем вызова списка существующих сессий через меню Session->Session List. Из появившегося списка (рис. 1.10) следует выбрать необходимую сессию и активировать ее (кнопка "Activate") или вызвать мастер для создания новой сессии (кнопка "New Session").

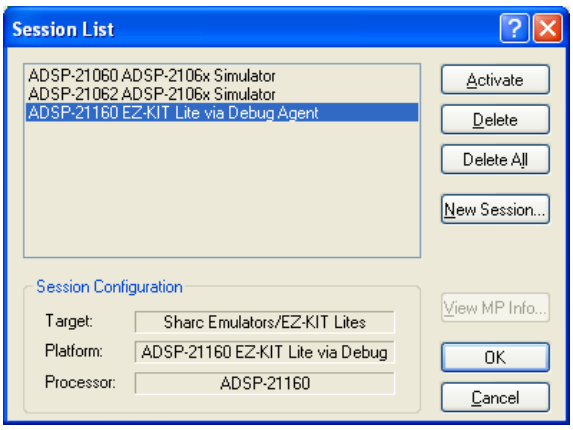


Рис. 1.10. Выбор и активация отладочной сессии

2. ЗАГРУЗКА И ОТЛАДКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ EZ-KIT LITE

2.1. Загрузка программы при включении питания системы ЦОС

На этапе отладки программного кода для системы ЦОС разработчик имеет возможность выполнять программу с использованием отладочных средств, таких как симулятор или эмулятор. В первом случае наличие "физического" DSP-процессора вообще не требуется, поскольку выполнение программы моделируется в интегрированной среде разработки, функционирующей на персональном компьютере. Во втором случае отладочные средства и системное программное обеспечение позволяют загружать инструкции и данные непосредственно в процессор и внешнее ОЗУ из среды разработки.

Но когда приложение написано и отлажено, возникает новая проблема. Функционирование реальной системы ЦОС предполагает автономное выполнение пользовательского кода DSP-процессором. Однако при включении питания энергозависимые внутренняя память (ОЗУ) процессора и внешнее ОЗУ на плате не содержат данных и программного кода, а отсутствие отладочного интерфейса уже не позволит загрузить процессор с персонального компьютера разработчика!

Для решения данной проблемы на плате с DSP-процессором обычно присутствует микросхема ПЗУ (Flash-память), которая содержит пользовательское приложение. Начальное взаимодействие между DSP-процессором и микросхемой ПЗУ зависит от задаваемого разработчиком режима выполнения программы:

- если пользовательское приложение при включении питания должно быть сначала загружено в быстродействующую внутреннюю память процессора (или внешнее ОЗУ) из

"медленной" микросхемы ПЗУ или внешнего устройства, то выполняется "загрузка процессора" (booting), в процессе которого происходит инициализации областей оперативной памяти инструкциями и данными, хранящимися в ПЗУ. Программа, которая загружается и выполняется из внутренней памяти, называется "загружаемой программой" (boot-loadable file);

- если пользовательское приложение (инструкции и данные) хранится и выполняется непосредственно из внешней памяти (внешнего ПЗУ), то такая программа не требует загрузки во внутреннюю память и внешнее ОЗУ процессора и называется "незагружаемой программой" (non-bootable file).

Очевидно, что и в одном, и в другом случае местом хранения пользовательского приложения при выключенном питании является либо ПЗУ (Flash-память), либо иное внешнее устройство, подключаемое через один из портов DSP-процессора. Поэтому после отладки программного кода его необходимо преобразовать в формат, который совместим с форматом хранения данных во внешнем устройстве и обеспечивает одну из двух возможностей:

- возможность считывания инструкций и данных для выполнения программы DSP-процессором непосредственно из внешнего ПЗУ (Flash-памяти);

- возможность предварительной загрузки программы из внешнего устройства во внутреннее или внешнее ОЗУ при включении питания системы ЦОС.

Образ для загрузки (Boot Image) – бинарный файл, сформированный загрузчиком и хранящийся в энергонезависимой микросхеме памяти, откуда он может быть прочитан как самим DSP-процессором, так и внешним хост-процессором. Загружаемый образ имеет особую структуру и может содержать один или несколько подразделов (по одному на каждый процессор).

Первый способ представляет собой простое преобразование исполняемого кода в формат программатора ПЗУ и выполняется утилитой, называемой Splitter (сплиттер).

Второй способ поддерживается утилитой Loader (загрузчик) и предполагает не только преобразование программного кода приложения, но и его модификацию путем дописывания специального программного кода, обеспечивающего загрузку пользовательской программы из внешнего ПЗУ (или внешнего устройства) во внутреннюю память (или внешнее ОЗУ) процессора.

Загрузочное ядро (Boot Kernel) – программный код фиксированного размера (например, 256 48-битных инструкций для SHARC ADSP-2106x/2116x), который автоматически загружается во внутреннюю память DSP-процессора при включении питания и управляет последующей загрузкой пользовательского приложения. Последние (по времени загрузки) секции пользовательской программы перезаписываются "сверху" загрузочного ядра, стирая, таким образом, его из памяти.

В некоторых средах разработки функции сплиттера и загрузчика реализуются одной системной утилитой. В то же время следует отметить, что поскольку загрузка программы может осуществляться не только из внешнего ПЗУ (Flash-памяти), но и через порты DSP-процессора из

какого-либо внешнего устройства (например, хост-процессора), то утилита Loader должна сформировать ядро загрузки таким образом, чтобы обеспечить загрузку данных указанным способом.

Если в качестве энергонезависимой памяти используется Flash-память, то прошивка Flash-памяти может выполняться через отладочный интерфейс средствами среды разработки (утилита Flash Programming Driver).

Поскольку режим работы системы с загрузкой и выполнением программы из внутренней памяти является более предпочтительным для подавляющего большинства практических приложений ЦОС, то рассмотрим более подробно утилиту Loader, структуру загружаемого файла (образа) и механизм его загрузки из ПЗУ (Flash-памяти) во внутреннюю память процессора.

Утилита Loader преобразует сгенерированный компоновщиком исполняемый файл (образ памяти) в один из поддерживаемых форматов, выбор которого зависит от способа загрузки (из ПЗУ, через порты ввода/вывода). При этом каждая секция, определенная в исполняемом файле, предваряется заголовком, где указывается ее размер и место размещения в памяти после загрузки. Затем в начало пользовательского кода дописывается загрузочное ядро. После этого сформированный файл (образ) готов к загрузке.

2.1.1. Способы загрузки программы во внутреннюю память процессоров SHARC ADSP

Процессоры SHARC ADSP поддерживают несколько *режимов загрузки* пользовательского приложения при включении питания:

- *режим без загрузки (No-Boot Mode)* – программа выполняется из внешнего ПЗУ (Flash-памяти). Этот режим не требует использования механизма загрузки;

- *загрузка из ПЗУ (PROM Boot Mode)* – пользовательское приложение загружается из внешнего ПЗУ (Flash-памяти). При включении питания Boot Kernel автоматически загружается из микросхемы ПЗУ (или Flash-памяти) и начинает выполняться в DSP-процессоре, читает из ПЗУ и записывает (переносит) в ОЗУ требуемые данные/инструкции;

- *загрузка через внешнее устройство* – в этом режиме DSP-процессор обычно выполняет роль slave-процессора в многопроцессорной системе. Через один из портов ввода/вывода в DSP-процессор автоматически при включении питания загружается ядро Boot Kernel, которое отвечает за дальнейшее взаимодействие с внешним устройством, получение инструкций/данных и их распределение во внутренней памяти. Процессоры SHARC ADSP поддерживают загрузку через внешний порт от хост-процессора (Host Boot Mode); линк-порт (Link Boot Mode),

последовательный интерфейс периферийных устройств Serial Peripheral Interface (SPI Boot Mode)[1].

Программный код загрузочного ядра входит в комплект среды разработки Visual DSP++ в нескольких модификациях. Нужная модификация выбирается утилитой Loader (и дописывается в начало пользовательского приложения) в зависимости от того, какой режим загрузки программы указан пользователем в опциях проекта. Файл загрузочного ядра в виде исходного текста программы на языке ассемблера находится в подкаталоге LDR каталога, соответствующего типу процессора.

2.1.2. Выбор способа загрузки при включении питания

Способ загрузки программы в оперативную память при включении питания определяется значениями сигналов на ножках процессора EBOOT, LBOOT и BMS в соответствии со следующими правилами:

EBOOT	*LBOOT*	*BMS*	*Режим загрузки*
0	0	0 (вход)	No-Boot Mode (без загрузки)
0	0	1 (вход)	Host Boot Mode (загрузка через хост-процессор)
0	1	1 (вход)	Link Boot Mode (загрузка через линк-порт)
1	0	выход	PROM Boot Mode (загрузка из ПЗУ, в этом режиме линия BMS управляется самим процессором)

2.1.3. Автоматическая загрузка ядра Boot Kernel при включении питания в режиме PROM Boot Mode

Автоматическая загрузка ядра Boot Kernel при включении питания осуществляется через выбранный интерфейс с использованием DMA-пересылки фиксированного размера. Рассмотрим процесс автоматической загрузки ядра Boot Kernel на примере режима загрузки PROM Boot Mode (загрузка из ПЗУ).

Пользовательское приложение и "добавленное" к нему ядро размещаются во внешней загрузочной памяти, временные параметры доступа к которой задаются в полях UBWS и UBWM регистра WAIT. Значение регистра WAIT по умолчанию задает способ доступа к "небанкируемой" (unbanked) памяти с максимальным числом тактов ожидания (значение поля UBWS равно 6) и доступом с подтверждением (значение поля UBWM равно 2). Эти параметры доступа к внешней микросхеме памяти используются только в том случае, когда на линии BMS выставляется активный уровень, а все линии $MS_{2\text{-}0}$ имеют неактивный уровень (т.е. идет обращение к

[1] Для SHARC ADSP 212xx и 213xx.

микросхеме ПЗУ).

Поскольку микросхема ПЗУ является 8-битовой, то пересылка данных осуществляется через внешнюю шину порциями по 8 битов. Поэтому при выполнении DMA-пересылки через внешний порт выполнять упаковку получаемых данных в режиме 8→48. Такой режим упаковки невозможно задать путем записи соответствующего значения в поле PMODE в регистре управления DMA-каналом. Тем не менее в режиме PROM Boot Mode при активном уровне сигнала BMS (доступ к ПЗУ) при DMA-пересылке выполняется автоматическая упаковка 6 последовательно полученных 8-битовых слов в одно 48-битовое слово.

Пересылка данных из внешней памяти (внешнего ПЗУ) во внутреннюю память осуществляется с использованием DMA-канала № 6 (для SHARC ADSP 2106x) или DMA-канала № 10 (SHARC ADSP 2116x).

Выбор микросхемы ПЗУ при доступе к внешней памяти (активное состояние на линии BMS) осуществляется путем установки бита BSO в регистре SYSCON. Если выполняется пересылка по одному из DMA-каналов при включенном бите BSO, то остальные DMA-каналы не могут быть задействованы. Также, если установлен бит BSO, то обращение процессорного ядра к внешней памяти не будет приводить к выставлению активного уровня на линию BMS (к загрузочной памяти обращается только DMA-контроллер через внешний порт). Это позволяет процессорному ядру выполнять запись значений во внешнюю память (при необходимости инициализации секций в пространстве внешней памяти) одновременно с чтением ПЗУ-памяти DMA-контроллером.

Начальные параметры регистров управления DMA-каналом приведены в табл. 2.1.

Таблица 2.1

Параметры памяти и DMA-канала при загрузке процессора из ПЗУ

Значения параметров для режима PROM Boot Mode	*ADSP-21060/61/62*	*ADSP-21160*	*Примечание*
Размер небанкируемой памяти	4M 8-битовых слов	8M 8-битовых слов	Предельный размер ПЗУ
Регистр PC программного секвенсора	0x20004	0x40004	Начальный адрес счетчика PC после сброса
DMA-канал	№ 6	№ 10	
Линии шины DATA	D23-16	D39-32	Линии внешней шины данных DATA, по которым передаются 8-битовые слова из ПЗУ
Адрес размещения в ПЗУ Boot Kernel	0x40 0000	0x80 0000	Пространство ПЗУ накладывается на начальные адреса внешней памяти. Выбор байтового пространства ПЗУ осуществляется при активном уровне сигнала на линии BMS
Адрес размещения в ПЗУ пользовательского приложения	0x40 0600	0x80 0600	Пользовательский код размещается сразу за кодом ядра Boot Kernel. Размер ядра равен 256 инстр. x 6 байтов (48 битов) = 1536 байтов (0x600)

Значения параметров для режима PROM Boot Mode	*ADSP-21060/61/62*	*ADSP-21160*	*Примечание*
Значения регистров параметров DMA-пересылки для ADSP-21060 / ADSP-21160			
DMAC6 / DMAC10	0x2A1	0x4A1	DMA Enable = 1 DType = 1 (instructions) PMode = 2 (not used) MasterMode = 1
II6 / II10	0x20000	0x40000	Начальный адрес пространства внутренней памяти (адрес размещения Boot Kernel)
IM6 / IM10	0x1	1	Модификатор адреса во внутренней памяти
C6 / C10	0x100	0x100	Размер Boot Kernel (256 48-битовых слов)
EI6 / EI10	0x40 0000	0x80 0000	Адрес начала пространства небанкируемой памяти
EM6 / EM10	0x1	0x1	Модификатор адреса во внешней памяти
EC6 / EC10	0x600	0x600	Количество 8-битовых пересылок по внешней шине
Адрес вектора прерывания			
EP0I	0x20040	0x40050	Для обработки прерывания по завершению приема Boot Kernel

При включении питания DSP-процессор в режиме PROM Boot Mode автоматически выполняет следующую последовательность действий:

1. Выставляется активный уровень на линию BMS для выбора пространства памяти, соответствующего микросхеме ПЗУ.

2. Процессор переходит в режим простоя (idle). При этом программный счетчик устанавливается на адрес вектора прерывания по сбросу.

3. Регистры параметров DMA-пересылки инициализируются согласно значениям таблицы (при этом значение поля PMODE регистра DMACx игнорируется). Инициализируются также поля UBWS и UBWM регистра WAIT.

4. DMA-контроллер в цикле читает 8-битовые слова из ПЗУ, упаковывает их в 48-битовые слова (инструкции) и передает во внутреннюю память до тех пор, пока не будет загружено 256 слов.

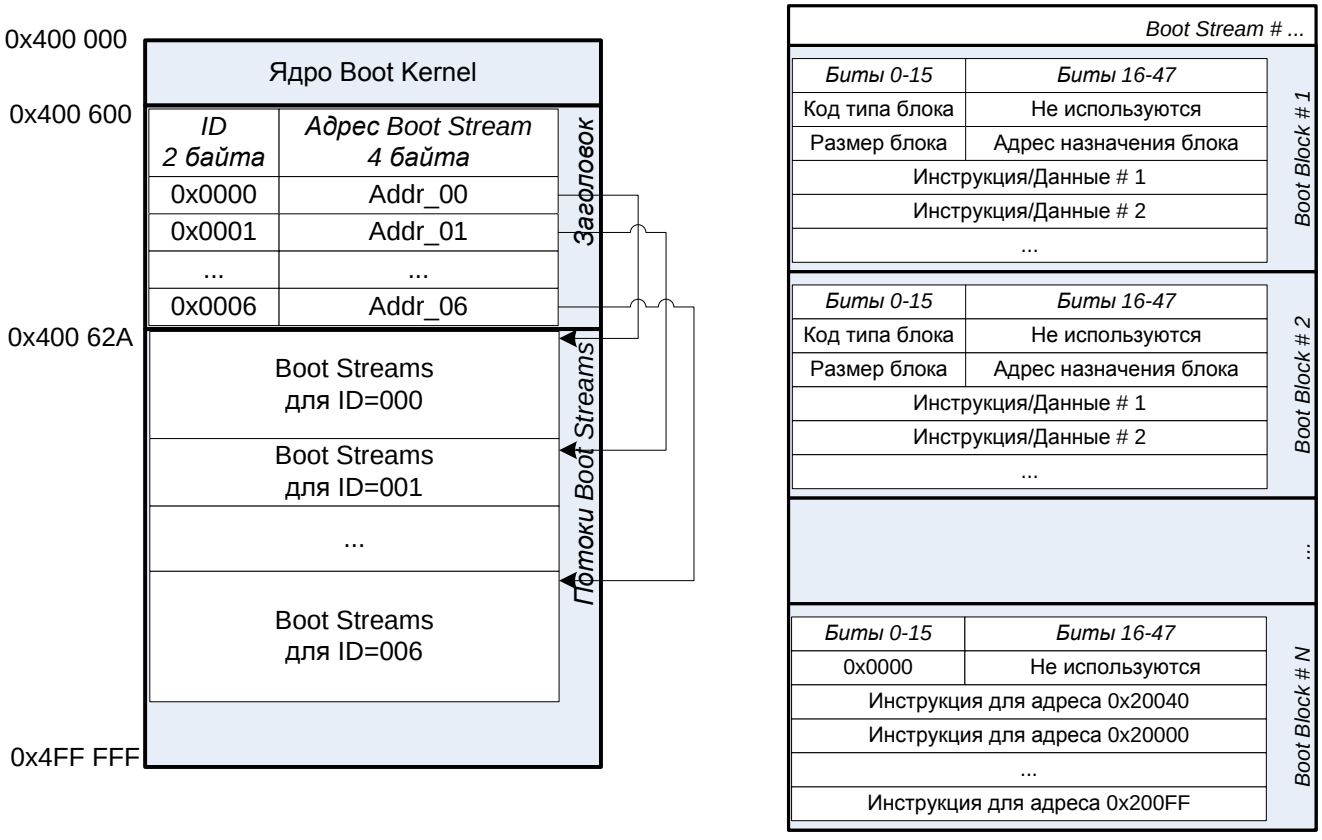
5. После завершения DMA-пересылки бит BSO в регистре SYSCON сбрасывается в 0, что снимает активный уровень с линии BMS и устанавливает режим нормального доступа к банкам внешней памяти.

2.1.4. Формат загружаемых данных

При формировании загружаемого образа (Boot Image) из исполняемого dxe-файла

пользовательской программы удаляется таблица символов (обеспечивающая связь дизассемблированного и текстового файлов при отладке), отладочная информация и карта памяти. Исполняемый файл (образ) записывается в загружаемый образ (Boot Image) в виде потока загрузки.

Поток загрузки (Boot Stream) представляет собой отформатированный определенным образом исполняемый файл (образ) пользовательского приложения, предназначенный для загрузки в процессор. Поток загрузки отличается от привычного набора строк программного кода тем, что содержит дополнительные данные, необходимые для инициализации памяти. Если проект предполагает загрузку нескольких процессоров (т.е. имеется несколько исполняемых файлов), то в загружаемом образе присутствуют несколько потоков. Каждый поток загрузки разбивается на несколько блоков (см. рис. 2.1).



а б

Рис. 2.1. Структура загружаемого образа Boot Image в ПЗУ (а) и потока Boot Stream (б)

Блок загрузки (Boot Block) – минимальная порция загружаемых данных, фактически соответствующая одной секции исполняемого файла (образа). Каждый блок содержит заголовок (тэг), в котором указываются адрес размещения секции и счетчик слов в секции.

Блоки загрузки могут быть различных типов (табл. 2.2). Тип блока задает правило для загрузки инструкций/данных и инициализации памяти (например, блоки, соответствующие пустым (неинициализированным) секциям не заносятся в ldr-файл и не копируются – вместо этого

Boot Kernel выполняет код, который записывает необходимое число нулей в заданную область ОЗУ).

Таблица 2.2

Типы блоков загрузки

Код типа блока	*Тип блока (соответствующей секции)*	*Способ инициализации области в ОЗУ, соответствующей этому блоку*
0x0000	Последний блок инициализации (final init)	Блок перезаписывается поверх Boot Kernel в режиме DMA-пересылки и обычно содержит таблицу векторов прерывания пользовательской программы. Перед запуском DMA-пересылки восстанавливаются значения по умолчанию в регистрах управления (SYSCON, DMAC, LCTL)
0x0001	Блок 16-разрядных нулевых значений (секция неинициализированных данных в программе) в DM-памяти (zero dm16)	Область памяти инициализируется в цикле нулевой константой с помощью косвенной адресации через DAG1. Записываемое значение ("0") хранится в регистре регистрового файла или регистре PX
0x0002	Блок 32-разрядных нулевых значений (секция неинициализированных данных в программе) в DM-памяти (zero dm32)	
0x0003	Блок 40-разрядных нулевых значений (секция неинициализированных данных в программе) в DM-памяти (zero dm40)	
0x0004	Блок 16-разрядных данных в DM-памяти (init dm16)	Указанное количество слов переносится в цикле из Boot Block и записываются в заданную область памяти через DAG1
0x0005	Блок 32-разрядных данных в DM-памяти (init dm32)	
0x0007	Блок 16-разрядных нулевых значений (секция неинициализированных данных в программе) в PM-памяти (zero pm16)	Область памяти инициализируется в цикле нулевой константой с помощью косвенной адресации через DAG2. Записываемое значение ("0") хранится в регистре регистрового файла или регистре PX
0x0008	Блок 32-разрядных нулевых значений (секция неинициализированных данных в программе) в PM-памяти (zero pm32)	
0x0009	Блок 40-разрядных нулевых значений (секция неинициализированных данных в программе) в PM-памяти (zero pm40)	
0x000A	Блок 48-разрядных нулевых значений (секция неинициализированных данных в программе) в PM-памяти (zero pm48)	
0x000B	Блок 16-разрядных данных в PM-памяти (init pm16)	Указанное количество слов переносится в цикле из Boot Block и записываются в заданную область памяти через DAG2
0x000C	Блок 32-разрядных данных в PM-памяти (init pm32)	
0x000E	Блок 48-разрядных данных (инструкций) в PM-памяти (init pm48)	
0x000F	Блок 64-разрядных нулевых значений (секция неинициализированных данных в программе) в DM-памяти (zero dm64)	Только для ADSP-2116x. Обрабатываются аналогично рассмотренным выше секциям zero… и init… в соответствии с пространством памяти
0x0010	Блок 64-разрядных данных в DM-памяти (init dm64)	
0x0011	Блок 64-разрядных нулевых значений (секция неинициализированных данных в программе) в PM-памяти (zero pm64)	
0x0012	Блок 64-разрядных данных в PM-памяти (init pm64)	

2.1.5. Загрузка пользовательского приложения под управлением ядра Boot Kernel

Структура процедуры обработки Boot Image в загрузочном ядре приведена на рис. 2.2. Укрупненно ее можно представить в виде четырех больших фрагментов:

- инициализационная часть ядра Boot Kernel. В начале выполнения инициализируются значения системных регистров и регистров DMA-канала для доступа к ПЗУ, а также определяется номер (ID) процессора для выбора "правильного" потока загрузки Boot Stream;

- загрузка блоков Boot Block, соответствующих выбранному потоку загрузки. При этом в зависимости от типа Boot Block либо выполняется чтение всего блока из ПЗУ, либо инициализация области назначения нулевыми значениями;

- подготовка загрузки последнего блока Boot Block. Обработка последнего блока отличается от остальных, поскольку его загрузка приводит к удалению кода загрузочного ядра. В то же время по завершению загрузки необходимо выполнить несколько обязательных инструкций. Все это требует специальных подготовительных операций, которые и выполняются на данном этапе работы ядра;

- чтение слова из ПЗУ. Поскольку ПЗУ является 8-разрядным, то процедура в режиме DMA-пересылки выполняет упаковку 8-битовых данных в 48-битовые слова, которые затем обрабатываются в главном цикле ядра.

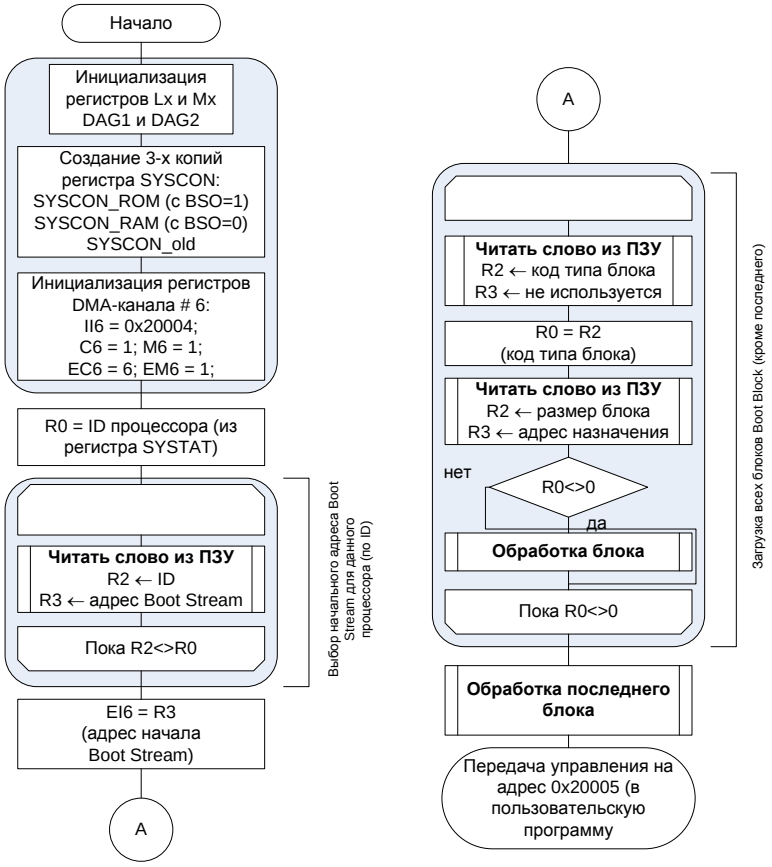


Рис. 2.2. Блок-схема выполнения ядра Boot Kernel

Блок-схема процедуры чтения слова из ПЗУ приведена на рис. 2.3.

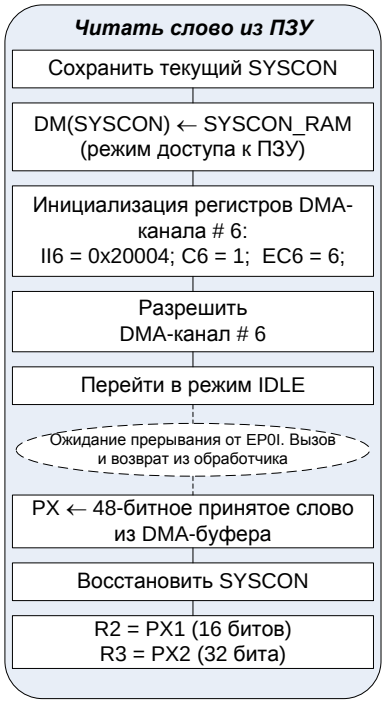


Рис. 2.3. Чтение слова из ПЗУ

При чтении слова из ПЗУ используется DMA-пересылка через внешний порт EOP0I (DMA-канал # 6). Каждое принятое и записанное во внутреннюю память 48-битовое слово формируется из шести 8-битовых слова, пересылаемых по внешней шине из ПЗУ. По завершении приема слова DMA-пересылка запрещается (рис. 2.4).

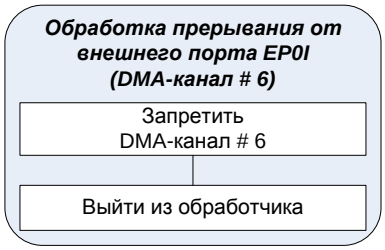


Рис. 2.4. Обработка прерывания EP0I

Процедура обработки каждого блока Boot Block приведена на рис. 2.5.

Если инициализируемый сегмент (секция) не содержит данных (т.е. представляет собой место в памяти для хранения промежуточных значений – xxxx_zero), то инициализация такого сегмента выполняется просто путем записи нулевых значений во все ячейки сегмента. Естественно, что такой сегмент нет необходимости хранить в ldr-файле, поскольку его инициализация может быть выполнена самим загрузочным ядром.

Для ненулевых блоков загрузки (xxxx_init) в цикле выполняется процедура чтения очередного 48-битового слова из ПЗУ и записи его во внутреннюю память с пост-модификацией индексного регистра.

Следует обратить внимание, что при инициализации 16- и 32-разрядных секций

используются регистры регистрового файла (R0 или R3). При записи 40- и 48-битовых данных – регистры PX.

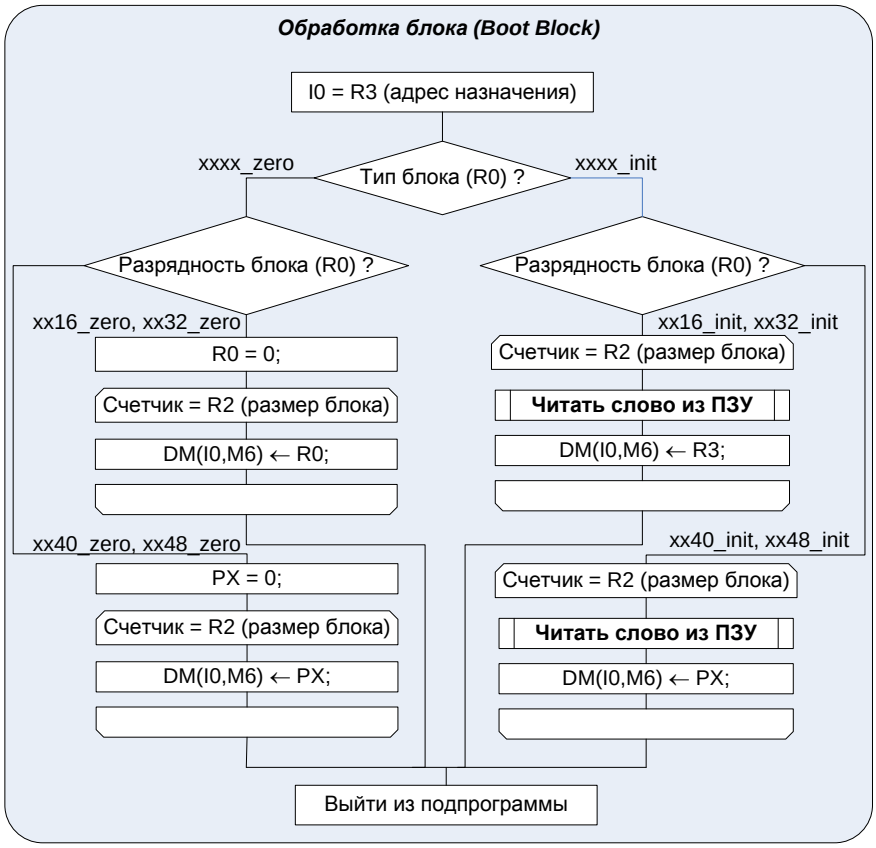


Рис. 2.5. Процедура обработки блоков Boot Block

Особого внимания заслуживает процедура обработки последнего блока Boot Block, которая должна обеспечить выполнение следующих требований:

- восстановление регистра SYSCON в первоначальное состояние (а это можно сделать только после того, как будет завершено чтение образа из ПЗУ);

- передача управления на вектор обработки прерывания по сбросу (только по завершению DMA-пересылки, когда код Boot Kernel уже заменен кодом пользовательской программы);

- замена обработчика прерывания EP0I (инструкции `rti` по адресу 0x20040) после завершения DMA-пересылки на инструкцию программы пользователя.

Данные требования реализуются путем использования довольно сложных программных конструкций, включающих "ручную" организацию цикла для самомодификации кода после завершения переноса из ПЗУ во внутреннюю память (в таблицу векторов прерываний) последнего блока.

Процедура подготовки самомодификации кода при обработке последнего блока Boot Block приведена на рис. 2.6.

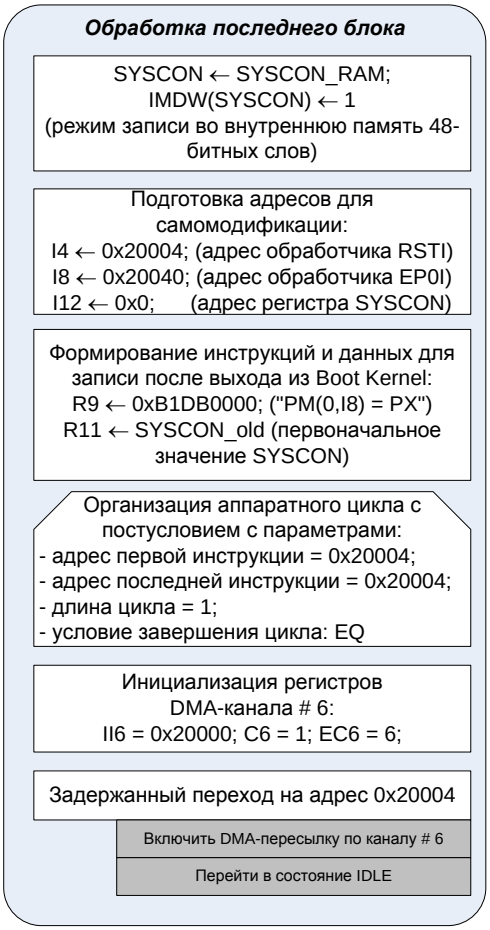


Рис. 2.6. Процедура обработки последнего блока Boot Block

В теле последнего блока Boot Block размером 256 инструкций, который записывается поверх таблицы векторов прерываний (с адреса 0x20000), утилита Loader принудительно вносит две модификации, отличающие его от исходного кода программы пользователя следующим образом:

```
0x20004:    R0 = R0 - R0,  DM(I4,M5) = R9,  PM(I12,M13) = R11;
...
0x20040:    rti;
```

Оригинальная инструкция по адресу 0x20004 нигде не сохраняется (поэтому пользовательская программа должна начинать свою работу с адреса 0x2005). А оригинальная инструкция по адресу 0x20040 хранится во втором слове заголовка последнего блока Boot Block (этот тэг для последнего блока не нужен, т.к. его размер и место размещения в ОЗУ постоянны). При чтении заголовка она сохраняется в регистре PX (регистр PX при копировании последнего блока Boot Block не используется и, следовательно, не модифицируется).

Но система команд SHARC ADSP не позволяет в одной инструкции выполнить и модификацию регистра SYSCON, и модификацию инструкции по адресу 0x20040. А использование 2-х и более принудительно записываемых инструкций в обработчике прерывания (из 4-х возможных) неэффективно. Для решения задачи последовательного выполнения "двух

команд по одному адресу" (причем уже после выхода из Boot Kernel) в процедуре обработки "вручную" организован цикл из одной инструкции по адресу 0x2004. На эту инструкцию и передается управление в конце выполнения загрузочного ядра.

С учетом записанных во время выполнения процедуры "Самомодификация блока" в регистры R9 и R11 значений, выполнение инструкции по адресу 0x20004 приводит сразу к нескольким событиям:

1) сработает условие выхода из цикла, организованного при обработке последнего блока Boot Block и стеки PCSTK и Loop Address Stack освободятся;

2) по адресу 0x20004 будет записана инструкция:

```
PM(0,I8) = PX;
```

т.е. инструкция заменит свой собственный код;

3) поскольку цикл на последней фазе своего выполнения представляет собой цикл из одной инструкции, то значение регистра PC не изменится и останется равным 0x20004.

На следующем шаге (вторая итерация цикла) вновь будет выполнена новая инструкция `PM(0,I8) = PX` по старому адресу 0x20004, что приведет к записи в ячейку по адресу I8 (0x20040) команды пользовательского приложения, хранящейся в PX.

Далее управление передается на первую инструкцию программы пользователя (адрес 0x20005).

2.1.6. Основные возможности загрузки программного кода для процессоров ADSP-2126x и ADSP-2136x

Процессоры SHARC ADSP моделей 2126x и 2136x не поддерживают возможность загрузки DSP-процессора через хост-интерфейс. Вместо этого в дополнение к загрузке из 8-битового стандартного ПЗУ (Flash-памяти) они имеют широкие возможности по загрузке программного кода через Serial Peripheral Interface (SPI), который обеспечивает взаимодействие с устройствами различного типа (ПЗУ, Flash-памятью, внешним процессором) по унифицированному протоколу обмена данными.

Размер ядра Boot Kernel увеличился до 384 инструкций, загружаемых во внутреннюю память при включении питания посредством DMA-пересылки.

Основные изменения в программном коде Boot Kernel вызваны изменением архитектуры внешнего (параллельного) порта по сравнению с SHARC ADSP-2116x и структурой регистров управления.

На кристалле процессоров 2126x и 2136x имеется встроенное ПЗУ, хранящее программный код, которое реализует функции загрузочного ядра Boot Kernel. Использование данного кода позволяет существенно упростить и ускорить процесс загрузки, отказавшись от хранения в внешнем ПЗУ (или Flash-памяти) загрузочного ядра, и выполнения сложных процедур самомодификации кода.

2.2. Загрузка пользовательского приложения в EZ-KIT Lite средствами VisualDSP++ для выполнения и отладки

Как уже отмечалось выше, на этапе отладки программного кода для системы ЦОС разработчик имеет возможность загружать инструкции и данные непосредственно в процессор и внешнее ОЗУ из среды разработки с использованием интерфейса между платой и средой разработки. Для автономного функционирования системы ЦОС пользовательская программа должна быть записана в энергонезависимое ПЗУ (или Flash-память), размещаемое на плате, чтобы при включении питания быть загруженной в процессор.

Комплект средств разработки на базе VisualDSP++ и оценочной платы EZ-KIT Lite позволяет реализовать оба рассмотренных варианта загрузки.

Для реализации "отладочного" варианта достаточно выбрать сессию "ADSP-21160 EZ-KIT Lite" и скомпилировать программу, указав в качестве типа проекта на вкладке "Project Options" тип "Executable". После компиляции среда VisualDSP++ автоматически записывает исполняемый файл (образ) через USB-интерфейс во внутреннюю память процессора и передает управление на первую инструкцию пользовательского кода.

Для реализации варианта "автономной" загрузки тип проекта должен быть установлен в "Loader File", а для прошивки полученного образа в микросхему Flash-памяти следует использовать утилиту Flash Programmer.

2.3. Использование утилиты Flash Programmer для прошивки Flash-памяти на плате EZ-KIT Lite

2.3.1. Взаимодействие между драйвером Flash-памяти и средой VisualDSP++

Взаимодействие между драйвером Flash-памяти и средой VisualDSP++ осуществляется следующим образом. В программе драйвера отводится некоторая область, содержащая

(последовательно) переменные для хранения кода выполняемых команд Flash-интерфейса и буфера для хранения данных, которые необходимо записать во Flash.

Этот набор данных хранится во внутренней памяти ADSP после загрузки программы-драйвера в процессор средой VisualDSP++. Поскольку SHARC ADSP предоставляет доступ к своей внутренней памяти другим процессорам или хост-процессору через общую внешнюю шину (а обмен данными между VisualDSP++ и ADSP осуществляется посредством общей внешней шины, причем USB-интерфейс выступает в роли интерфейса хост-процессора, имеющего максимальный приоритет в системе).

Используя полученные значения, драйвер выполняет запись буфера данных во Flash-память, используя стандартные операции, рассмотренные в приложении А.

2.3.2. Использование утилиты Flash Programmer

ADSP-21160 EZ-KIT Lite содержит утилиту Flash Programmer. Эта утилита позволяет вам программировать Flash память на EZ-KIT Lite. Flash Programmer инсталлируется вместе со средой разработки VisualDSP++ и вызывается при выборе меню Tools/Flash Programmer.

После запуска утилиты необходимо на вкладке Driver выбрать (кнопка Browse...) и загрузить (кнопка Load Driver) в DSP-процессор на плате программный код драйвера, соответствующий модели процессора. Он расположен в каталоге Analog Devices в подкаталоге с именем модели процессора (для рассматриваемой платы файл драйвера имеет имя 21160EZFLASHDRIVER.DXE). После этого окно утилиты Flash Programmer примет вид, приведенный на рис. 2.7.

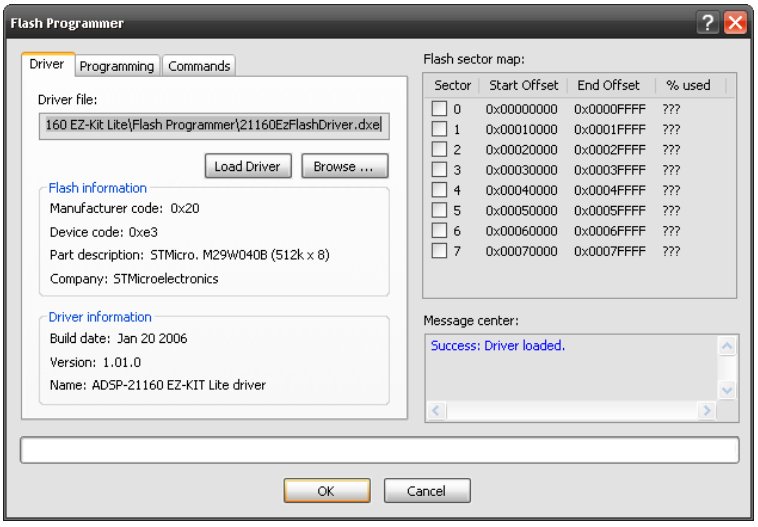


Рис. 2.7. Загрузка драйвера Flash Programmer

В левой части вкладки будет выведена информация о производителе микросхемы (вверху) и разработчике и версии драйвера Flash-памяти (внизу). В правой части окна показаны блоки

(сектора) Flash-памяти с их адресами.

Вкладка Programming содержит интерфейс, позволяющий записать в микросхему Flash-памяти загрузочный образ (Boot Image) пользовательского приложения вместе с ядром загрузки Boot Kernel. Для этого необходимо выбрать ldr-файл с приложением, указать необходимый формат файла для прошивки и затем нажать кнопку Program. По результатам программирования Flash-памяти в правой части вкладки будет показан процент использования каждого из секторов памяти (рис. 2.8). Чтобы сверить записанный во Flash-памяти образ с ldr-файлом на предмет наличия ошибок, следует нажать кнопку Compare (Сравнить) или при записи установить флажок Verify while programming (при этом процесс записи замедляется).

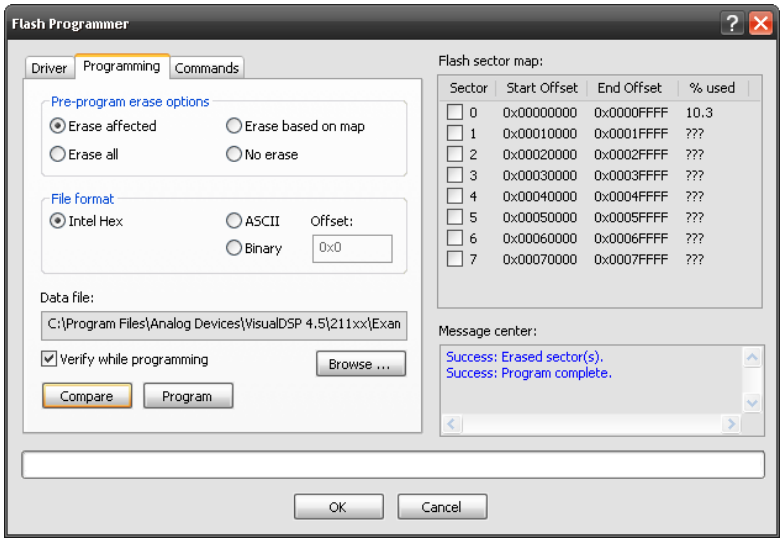


Рис. 2.8. Результат прошивки Flash-памяти

Дополнительные возможности по работе с Flash-памятью представлены на вкладке Commands (рис. 2.9). Интерфейс утилиты Flash Programmer позволяет заполнить содержимое секторов микросхемы необходимым значением, начиная с заданного адреса (верхняя левая часть вкладки) или выполнить требуемую команду (нижняя левая часть вкладки). Наиболее "необходимые" команды (Erase all, Reset Flash) вынесены на отдельные клавиши.

Нажатие каждой из клавиш ведет к вызову соответствующей функции драйвера Flash-памяти. Код операции и ее параметры передаются через интерфейс платы EZ-KIT Lite.

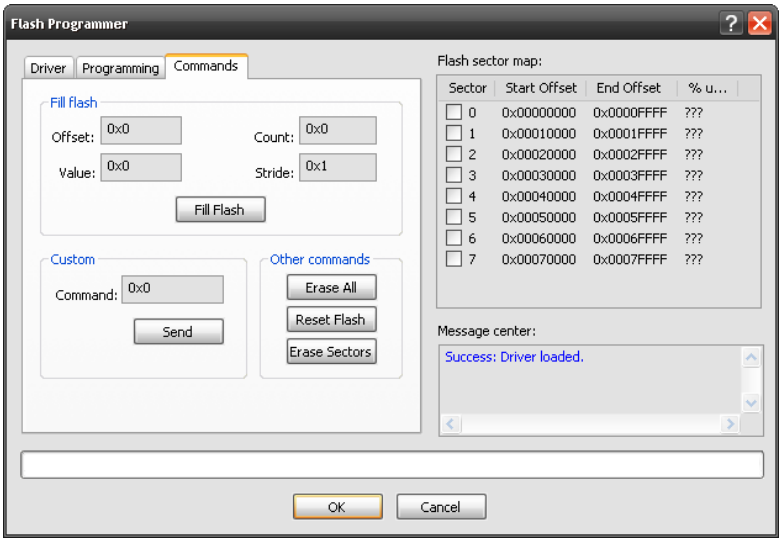


Рис. 2.9. Дополнительные команды по работе с Flash-памятью

Принцип работы драйвера, загруженного в EZ-KIT Lite, заключается в следующем. Утилита Flash Programmer в соответствии со структурой и содержанием ldr-файла записывает в буфер во внутренней памяти процессора (средствами VisualDSP++ и USB-интерфейса платы) структуру данных, содержащую код операции, которую должен выполнить драйвер над Flash-памятью, и необходимые данные (например, подлежащий записи во Flash-память фрагмент кода программы). По завершению пересылки данных из VisualDSP++ в процессор выставляется флаг готовности команды к выполнению. После этого драйвер Flash-памяти выполняет заданную операцию и по ее окончанию выставляет флаг готовности к продолжению работы и т.д.

Если драйвер должен вернуть какую-либо информацию (информацию о производителе микросхемы, содержание ячейки или сектора), то такой обмен также осуществляется через буфер в памяти DSP-процессора.

После программирования Flash-памяти следует отключить и снова включить питание платы EZ-KIT Lite. Пользовательское приложение будет автоматически загружено во внутреннюю память процессора и начнет выполнение даже при отключенном интерфейсе взаимодействия с компьютером.

2.4. Карта памяти EZ-KIT Lite ADSP-21160M

В системе на базе EZ-KIT-Lite могут использоваться следующие "виды" памяти, реализованные на плате:

- внутреннее статическое ОЗУ (Internal SRAM), сконфигурированное в виде двух блоков, расположено на кристалле ADSP. Структура внутренней памяти, позволяющей хранить

инструкции и данные и осуществлять доступ к ним на частоте процессорного ядра (наиболее быстрая память), подробно рассматривается в руководстве по процессору и в лекционном курсе;

- внешняя память системы включает в себя статическое ОЗУ с поддержкой синхронной пакетной передачи данных (SBSRAM) конвейерного (flow-through) типа и Flash-память.

Поддержка SBSRAM отличает ADSP-211xx и последующие поколения SHARC-процессоров от их предшественников. За счет использования внутреннего счетчика переданных слов SBSRAM позволяет получить высокую скорость доступа к словам данных, размещенным в памяти последовательно. Однако производительность такой памяти может снижаться, если доступ к памяти выполняется непоследовательно. SBSRAM на плате содержит 512 Кбайт в виде 2-х микросхем, каждая из которых содержит 64К 32-битовых слов (64К x 32бит x 2). Выбор SBSRAM осуществляется при выставлении (во время доступа к памяти) активного уровня на линию выбора памяти ~MS1. Благодаря внешней 64-битовой шине данных за одно обращение можно прочитать/записать два 32-битовых слова данных. При этом физически младшие 32 бита, передаваемые по внешней шине данных, хранятся в первой микросхеме, а старшие 32 бита – во второй (с точки зрения программиста это не принципиально).

Наряду с SBSRAM на плате содержится одна микросхема Flash-памяти с 512Кбайт памяти (512К x 8бит). Выбор Flash-памяти осуществляется при выставлении активного уровня на линию ~MS0. Основное назначение Flash-памяти – хранение программ и неизменяемых данных при выключении питания платы с целью их загрузки в процессор при включении системы. Поэтому Flash-память может использоваться как загрузочная память благодаря подключению к линии выбора памяти загрузки (~BMS) процессора. С помощью конфигурирования варианта загрузки переключателями на плате (и прошивки Flash-памяти с помощью VisualDSP++) плату можно запустить работать самостоятельно, без интерфейса с ПК.

Карта памяти EZ-KIT Lite, которую следует использовать при разработке собственных приложений, приведена на рис. 2.10. Данная карта памяти соответствует значению размера банка внешней памяти 32M слова (поле MSIZE в регистре SYSCON=1100b): нулевой банк – флэш-память, первый банк – SBSRAM.

	Start Address	End Address	Content
Internal Memory	0x0000 0000	0x0000 FFFF	IOP Registers
	0x0002 0000	0x0003 FFFF	Long Word Addressing
	0x0004 0000	0x0007 FFFF	Normal Word Addressing
	0x0008 0000	0x000F FFFF	Short Word Addressing
Multipro-cessor Space	0x0010 0000	0x001F FFFF	ID = 001 Internal Memory
	0x0020 0000	0x002F FFFF	ID = 010 Internal Memory
	0x0030 0000	0x003F FFFF	ID = 011 Internal Memory
	0x0040 0000	0x004F FFFF	ID = 100 Internal Memory
	0x0050 0000	0x005F FFFF	ID = 101 Internal Memory
	0x0060 0000	0x006F FFFF	ID = 110 Internal Memory
	0x0070 0000	0x007F FFFF	ID = 111 Internal Memory
External Memory	0x0080 0000	0x0087 FFFF	MSO and BMS (Flash memory[1])
	0x0280 0000	0x0281 FFFF	MS1 (SBRAM)
	All other locations		Not Used

Рис. 2.10. Карта памяти EZ-KIT Lite

Пример программного кода для инициализации Flash-памяти и занесения необходимых значений в регистры управления доступом к памяти процессора SHARC ADSP:

```
SetupForFlash:

      bit set MODE2 FLG0O | FLG1O | FLG2O;
      bit clr FLAGS FLG0 | FLG1 | FLG2;

      // MSIZE = log2(banksize)-13
      // размер банка = 32Mb = 200 0000
      // MSIZE = 1100
      // банк по линии MS0  80 0000 - 27F FFFF
      // банк по линии MS1 280 0000 - 47F FFFF
      // банк по линии MS2 480 0000 - 67F FFFF

      ustat1 = dm(SYSCON);
      bit set ustat1 0x0000C000;       // MSIZE <- 1100
      dm(SYSCON) = ustat1;
```

2.5. Выполнение и отладка программы на EZ-KIT Lite под управлением среды VisualDSP++

2.5.1. Программа-монитор

Для загрузки, выполнения и отладки пользовательского приложения на плате EZ-KIT Kite используется программа-монитор, входящая в комплект поставки среды разработки VisualDSP++.

Первоначально ядро программы-монитора загружается на плату во время подачи питания из ПЗУ, размещенного на плате. Во время установления и конфигурирования соединения с VisualDSP++ через USB-интерфейс это ядро позволяет загрузить полноценную программу-монитор, которая управляет взаимодействием между платой и VisualDSP++ и позволяет среде разработки загружать, выполнять и отлаживать пользовательское приложение, а также просматривать регистры процессора и содержимое памяти на плате.

Программа монитор состоит из трех основных компонентов:

- цикл ожидания (Halt Loop);
- обработчик прерывания от последовательного интерфейса (UART ISR);
- ядро обработки команд управления (Command Processing Kernel).

Когда пользовательское приложение не выполняется, программа-монитор удерживает процессор в состоянии idle. В это время среда VisualDSP++ может читать/записывать регистры процессора, ячейки памяти, загружать программу, устанавливать точки останова.

Чтобы перевести плату в состояние цикла ожидания необходимо приостановить выполнение пользовательской программы – либо посредством точки останова, либо выполнив команду Halt в среде отладки. После этого компьютер связывается с программой-монитором и может посылать команды и читать данные. Управляющие команды (не путать с инструкциями процессора!) передаются средой VisualDSP++ через USB-интерфейс побайтово. Каждый байт, записанный в порт интерфейса, генерирует прерывание на ADSP-21160 EZ-KIT Lite. В обработчике прерывания монитор "собирает" команду управления из нескольких последовательных байтов. Когда команда будет сформирована, ядро обработки команд (Command Processing Kernel) выполняет ее, создает соответствующий пакет ответа (например, значения регистров или памяти) для конкретной команды и уведомляет VisualDSP++ о выполнении команды.

VisualDSP++ опрашивает регистр статуса порта, проверяя наличие ответа от платы. Когда ответ обнаружен, VisualDSP++ генерирует прерывание или последовательность прерываний, которые сигнализируют программе-монитору, что можно передавать пакет ответа в среду разработки. Когда пакет ответа послан, монитор возвращается в свое состояние idle.

Если среда VisualDSP++ посылает команду на ADSP-21160 EZ-KIT Lite во время выполнения

кода пользователя, то обработчик прерывания UART ISR обрабатывает каждый байт аналогичным образом, собирая в итоге целую команду управления, однако после обслуживания прерывания управление возвращается программе пользователя.

При переключении между программой-монитором и программой пользователя выполняется сохранение/восстановление контекста задачи.

Поскольку программа-монитор использует те же прерывания, что и пользовательское приложение, то в случае необходимости отладки программы на плате EZ-KIT Lite следует иметь в виду следующие ограничения:

- если запретить вложенные прерывания, то хост-процессор (VisualDSP++) потеряет возможность взаимодействия с программой-монитором во время выполнения приложения и внутри обработчика прерываний;

- если пользовательское приложение запрещает прерывания от последовательного порта, то VisualDSP++ теряет возможность взаимодействия с программой-монитором во время выполнения приложения;

- VisualDSP++ теряет возможность взаимодействия с программой-монитором во время выполнения обработчика прерывания от таймера высокого уровня;

- если запретить бит глобального разрешения прерываний (IRPTEN), то VisualDSP++ не сможет приостанавливать выполнение приложения по команде Halt. Однако точки останова (breakpoints) будут функционировать нормально.

2.5.2. Управление отладкой программы, выполняемой на EZ-KIT Lite, из среды VisualDSP++

Средства загрузки и отладки программы, выполняемой на плате EZ-KIT Lite из среды VisualDSP++, сосредоточены в меню Settings и включают следующие возможности:

- установка аппаратных точек останова (Hardware Breakpoints);

- установка некоторых опций, затрагивающих процесс запуска и останова программы (Target Options);

- генерация сигнала Hard Reset на плате EZ-KIT Lite. Выполнение этой команды (пункт подменю Setting/Boot Load) приводит к аппаратному сбросу платы EZ-KIT Lite и загрузке программного кода (если выбран режим PROM Boot);

- установка точек программного останова (Breakpoints).

Аппаратные точки останова (Hardware Breakpoints) действуют аналогично точкам останова по условию (Watchpoints), которые можно было использовать в среде симулятора при

программной отладке пользовательского приложения. При отладке приложения можно воспользоваться аппаратными точками останова трех типов:

- останов при обращении к данным по DM-шине. Есть возможность отследить доступ в указанном диапазоне адресов к внутренней памяти, IOP-регистрам, внешнему порту и пространству памяти многопроцессорной системы. Условие останова проверяется при доступе к памяти с адресацией через DAG1 или при абсолютной адресации через шину DM. Режим доступа (на чтение, на запись, произвольный) выбирается в поле Mode. Флаг Exclusive устанавливается в том случае, если останов необходим при обращении к памяти только вне указанного диапазона. Есть возможность задать две точки останова данного типа (рис. 2.11);

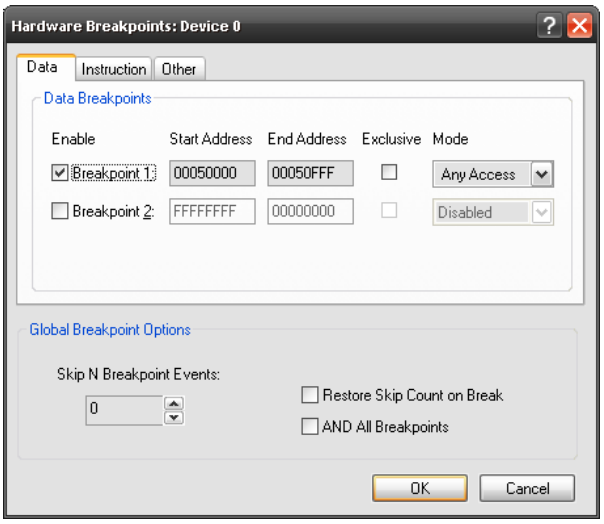


Рис. 2.11. Установка точек аппаратного останова при доступе к данным по шине DM

- останов при выполнении инструкции, попадающей в выбранный диапазон адресов. Останов осуществляется при выполнении инструкции внутри или вне (при включенном флаге Exclusive) указанного диапазона адресов. Есть возможность задать четыре точки останова данного типа (рис. 2.12);

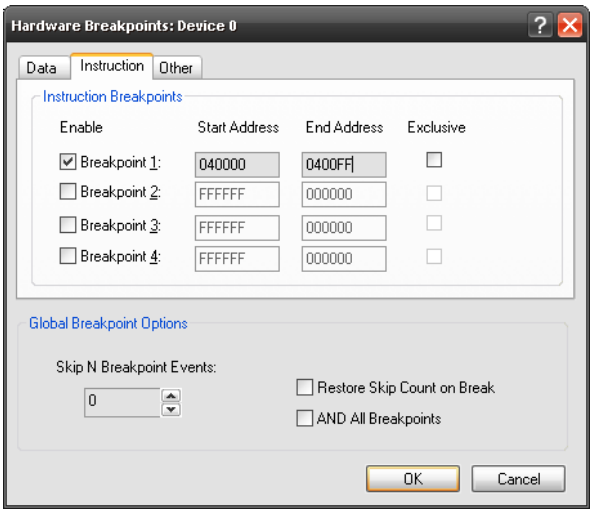


Рис. 2.12. Установка точек аппаратного останова при выполнении инструкций

- останов при доступе к памяти по указанным шинам данных. Имеется возможность остановить выполнение программы при доступе к памяти по шине PM, по шине I/O (например, при DMA-пересылках), или при доступе к данным по общей внешней шине через внешний порт (например, при доступе к пространству внешней памяти) – рис. 2.13.

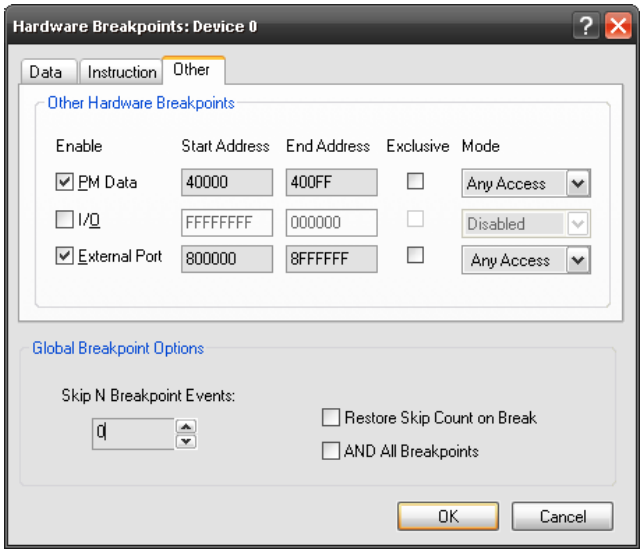


Рис. 2.13. Установка точек аппаратного останова при доступе к памяти по шинам PM, I/O и через внешнюю шину

Останов программы выполняется в том случае, если срабатывает хотя бы одна точка останова (логическое "ИЛИ"). Однако на срабатывание всех точек останова можно установить дополнительные опции (Global Breakpoint Options), касающиеся количества пропущенных событий до срабатывания точки останова, а также возможности группировки точек останова не по логическому "ИЛИ", а по логическому "И".

Точки аппаратного останова рекомендуется устанавливать до загрузки и запуска программы на выполнение.

Окно *Targer Options* используется для контроля особенностей функционирования процессора ADSP-21160 в режиме отладки (рис. 2.14).

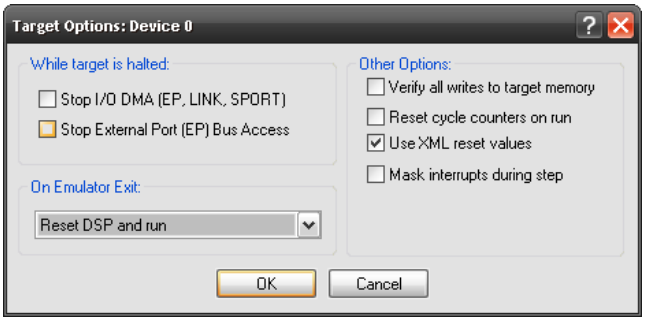


Рис. 2.14. Окно Target Options

Группа флагов "While target is halted" позволяет приостановить DMA-пересылки на время когда DSP-процессор функционирует под управлением программы-монитора (выполнен останов

программы из среды VisualDSP++). Следует помнить, что поскольку передача данных через SPORT не может быть приостановлена, то установка соответствующего флажка приведет к потере данных.

Опция "On Emulator Exit" позволяет выбрать вариант функционирования DSP-процессора после того, как эмулятор передает управление пользовательскому приложению.

Остальные опции (правая часть окна Target Options) также используются в процессе отладки для контроля записи памяти при загрузке программы из VisualDSP++ (Verify all writes to target memory), сброса счетчика точек останова (Reset cycle counter on run), загрузки значений по умолчанию из xml-файла (в каталоге System\ArchDef\ИмяПроцессора.xml) в регистры процессора, маскирования прерываний при пошаговой отладке программы.

Программные точки останова (Breakpoints) используются по аналогии с симулятором. Начиная с версии 4.5, в среде VisualDSP++ появилась возможность задавать автоматическую генерацию точек останова, которая выполняется при загрузке программы в EZ-KIT Lite. Они могут быть условными и безусловными. Принципы работы с программными точками останова аналогичны другим современным средствам отладки ПО.

3. КОДЕК AD1881A SOUND MAX

3.1. Основные функциональные возможности кодека AD1881

Кодек AD1881, установленный на плате EZ-KIT Lite, представляет собой кодек стандарта AC'97 (rev 2.1) с внешним аналоговым интерфейсом, который может быть использован для захвата, обработки и воспроизведения аналоговых сигналов в системе ЦОС (или персональном компьютере). AD1881 широко используется в качестве недорогого интегрированного кодека стандарта AC'97 на современных материнских платах. Базовыми возможностями кодека стандарта AC'97 являются:

- 16-битовый полнодуплексный стереоаудиокодек (АЦП и ЦАП с отношением сигнал/шум при преобразовании "аналог↔цифра" около 85 дБ);
- возможность подключения на аналоговый вход линейного входа (LINE IN) и микрофона (MIC) с усилением сигнала на 20 дБ;
- аналоговый стерео- или моновыход (LINE OUT) с возможностью подключения динамиков;
- возможность переключения типов входного сигнала с помощью переключателей.

Кодек AD1881 имеет дополнительные (по сравнению со стандартом AC'97) возможности,

облегчающие его взаимодействие с процессорами ADSP:

- поддержка специального режима Slot-16 для упрощенного взаимодействия с DSP-процессором через последовательный порт в многоканальном режиме;

- независимая частота дискретизации для АЦП и ЦАП в диапазоне от 7 КГц до 48 КГц с шагом 1 КГц;

- возможность подключения по одному интерфейсу одновременно до трех кодеков AD1881;

- поддержка режима объемного 3D-стерео.

На плате EZ-KIT Lite кодек подключен к последовательному порту SPORT0. Выбор входного сигнала (микрофон или линейный вход) осуществляется переключателем JP1.

3.2. Организация взаимодействия процессора SHARC ADSP с кодеком AD1881

3.2.1. Интерфейс AC-link

Взаимодействие между микросхемой AD1881 и DSP-процессором осуществляется через последовательный порт в режиме с временным мультиплексированием каналов (Time Division Multiplexed Mode, TDM). В этом режиме команды для регистров кодека, информация о состоянии регистров, а также данные от АЦП и для ЦАП передаются и принимаются в различные временные интервалы (слоты) в пределах кадра многоканального режима. В соответствии с функциональным назначением AD1881 будем называть аудиокодеком (или просто кодеком), а ADSP-процессор – контроллером.

Цифровой интерфейс взаимодействия кодека с внешними устройствами в соответствии со стандартом AC'97 называется "AC-link". Он обеспечивает двунаправленную передачу на фиксированной скорости цифрового потока ИКМ-отсчетов сигнала в последовательном виде (побитово).

Для взаимодействия с контроллером кодек использует 5 цифровых линий (рис. 3.1). Все аудиоданные, управляющая и статусная информация передаются по линиям SDATA_IN и SDATA_OUT. Линии SYNC и BIT_CLK используются для передачи тактирующих сигналов. Линия RESET на плате EZ-KIT Lite соединена с выходом FLG2 и используется для программного сброса кодека (для EZ-KIT Lite ADSP-2116x – с выходом FLG3).

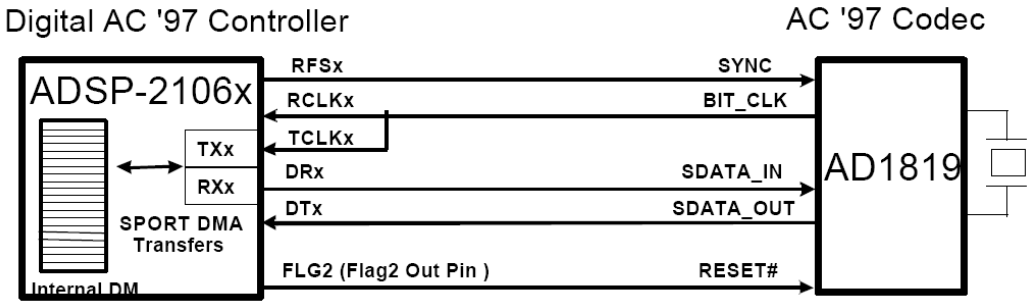


Рис. 3.1. Схема взаимодействия AD1881 и процессора SHARC ADSP

Стандарт AC'97 предполагает обработку аудиоотсчетов и команд в рамках многоканальной схемы с временным разделением каналов (TDM). Стандартный режим работы кодека разделяет входящий и исходящий каждый кадр на двенадцать 20-битовых слотов. Вначале каждого кадра передается специальный 16-битовый слот (# 0), который отражает корректность данных, содержащихся в соответствующем слоте в данном кадре (рис. 3.2). Каждый слот содержит управляющую/статусную информацию или ИКМ-отсчет аудиосигнала. Таким образом, размер каждого кадра составляет 256 битов (1 слот x 16 битов + 12 слотов x 20 битов). Такой формат данных хотя и позволяет передавать 18- и даже 20-битовые аудиоданные, но существенно усложняет низкоуровневое программирование взаимодействия между кодеком и контроллером из-за неодинаковой длины слотов: требуется дополнительная обработка по формированию из принятого потока данных слов правильной размерности.

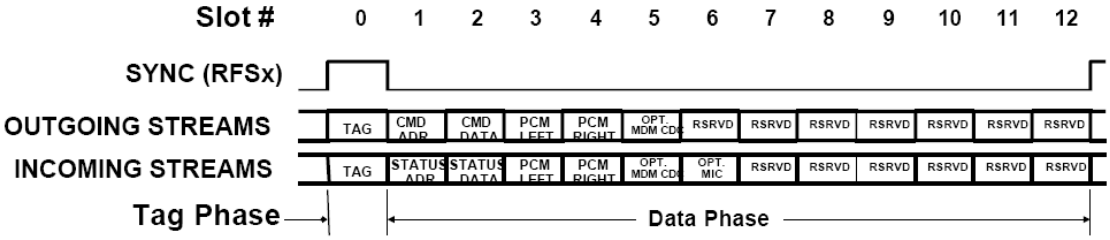


Рис. 3.2. Структура стандартного кадра AC'97

Режим Slot-16 адаптирован под TDM-режим приема/передачи данных через последовательный порт процессора SHARC ADSP. В этом режиме каждый кадр делится на шестнадцать одинаковых 16-битовых слотов, что существенно упрощает интерфейс с процессорами SHARC ADSP (рис. 3.3).

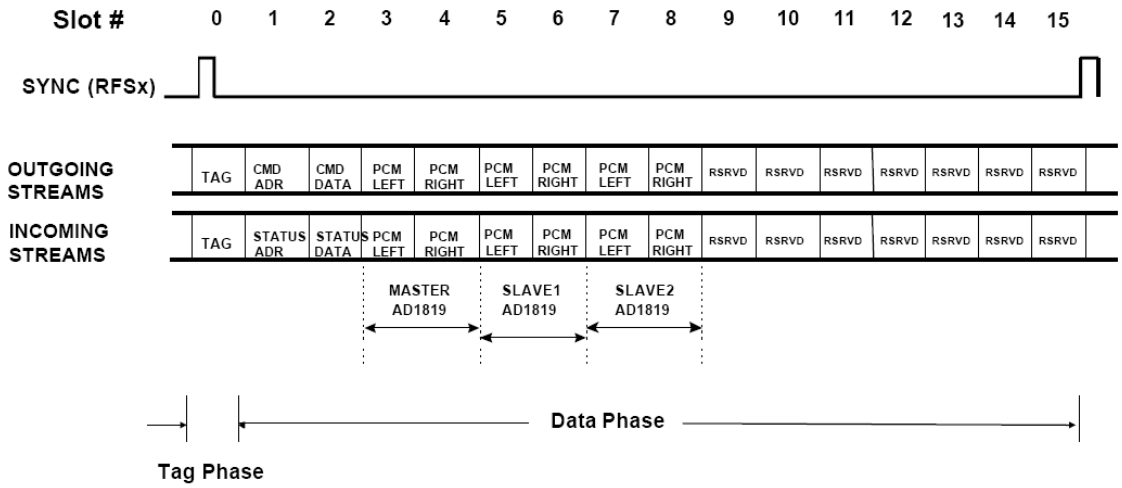


Рис. 3.3. Структура кадра в режиме Slot-16

Режим Slot-16 имеет особенности при адресации регистров кодека вследствие несоответствия границ временных слотов стандарту AC'97. Это следует учитывать в первую очередь при включении питания: в начале работы кодек работает в стандартном режиме AC'97 с 20-битовыми временными слотами, поэтому формирование первого временного слота для переключения в режим Slot-16 должно быть выполнено с учетом этого формата (рис. 3.4).

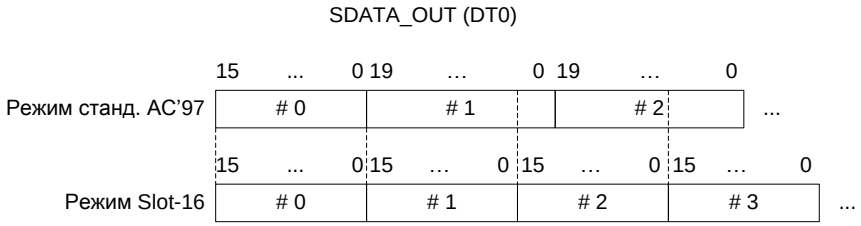


Рис. 3.4. Несоответствие границ временных слотов для стандартного режима кодека AC'97 и режима Slot-16 кодека AD1881

Структуры временных слотов кадра при передаче и при приеме данных для варианта взаимодействия контроллером с единственным кодеком AD1881 приведены в табл. 3.1 и 3.2. Неиспользуемые биты должны быть сброшены в "0".

Таблица 3.1

Структура временных слотов при передаче кадра от контроллера к кодеку для AD1881

Временной слот	*Биты*	*Описание*
# 0 (TAG Phase)	15	Valid Frame – флаг корректности данных в кадре
	14..3	Флаги корректности слотов № 1-12
# 1 (Command Address Slot)	15	Чтение (1) или запись (0) регистра кодека
	14..8	7-битовый номер регистра управления кодека
# 2 (Command Data Slot)	15..0	16-битовое значение для записи в регистр управления (если идет запись значения)
# 3 (PCM Left)	15..0	Значение текущего отсчета для ЦАП левого канала
# 4 (PCM Right)	15..0	Значение текущего отсчета для ЦАП правого канала

Таблица 3.2

Структура временных слотов при приеме кадра от кодека к контроллеру для AD1881

Временной слот	*Биты*	*Описание*
# 0 (TAG Phase)	15	Codec Ready – флаг готовности кодека
	14..3	Флаги корректности слотов № 1-12
# 1 (Status Address Slot) <если в предыдущем отправленном кадре бит включения режима запроса для ЦАП DRQEN=0>	14..8	7-битовый номер запрошенного регистра управления кодека
# 1 (Status Address Slot) <если в предыдущем отправленном кадре бит включения режима запроса для ЦАП DRQEN=1>	14..8	8-битовый номер запрошенного регистра управления кодека. Если в предыдущем отправленном кадре не было команды управления регистрами кодека, то содержит 0x74
	7 6 5..4 3..2	Биты запроса отчета для ЦАП ("0"-требуется, "1"-не требуется): - запрос данных для левого канала (DLRQ0) master-кодека - запрос данных для правого канала (DRRQ0) master-кодека - аналогичный запрос данных для каналов ЦАП кодека Slave1 - аналогичный запрос данных для каналов ЦАП кодека Slave2
# 2 (Status Data Slot) <если в предыдущем отправленном кадре бит включения режима запроса для ЦАП DRQEN=0>	15..0	16-битовое значение запрошенного регистра управления кодека
# 2 (Status Data Slot) <если в предыдущем отправленном кадре бит включения режима запроса для ЦАП DRQEN=1>	15..0	16-битовое значение регистра управления кодека 0x74
# 3 (PCM Left)	15..0	Значение текущего отсчета от АЦП левого канала
# 4 (PCM Right)	15..0	Значение текущего отсчета от АЦП правого канала

Тактирование кодека AC'97 осуществляется внешним генератором с частотой 24,576 МГц. Внутри кодека осуществляется понижение тактовой частоты и на выход BIT_CLK выдается сигнал с тактовой частотой 12,288 МГц. Сигнал тактовой синхронизации BIT_CLK является выходным для AC'97-кодека и входным (внешним) для аудиоконтроллера (DSP-процессора). Он подается на ножки RCLK0 и TCLK0 последовательного порта процессора SHARC ADSP. AD1881 передает данные по переднему фронту сигнала BIT_CLK, а принимает – по заднему фронту.

Поскольку размер кадра и в стандартном режиме, и в режиме Slot-16 равен 256 битам, то сигнал кадровой синхронизации SYNC должен формироваться один раз на 256 принятых/переданных битов. Сигнал кадровой синхронизации является внешним для AC'97-кодека и внутренним для DSP-процессора. Поскольку в многоканальном режиме работы последовательного порта сигнал RFS является сигналом кадровой синхронизации сразу и для передающей, и для приемной частей порта, то линия SYNC кодека соединена с ножкой процессора RFS0. Для генерации сигнала кадровой синхронизации DSP-процессором через каждые 256 битов в регистр RDIV0 необходимо записать значение 0x00FF0000 (поле RFSDIV = 255).

В одном кадре передается только ИКМ-отсчет, соответствующий данному моменту времени (в монорежиме – один отсчет, в стереорежиме – 2 отсчета, для левого и правого канала, в многоканальном режиме – по одному отсчету на каждый канала). Поэтому частота передачи

кадров звукового сигнала определяется количеством кадров аудиокодека в единицу времени и равна[2]:

12,288 МГц / 256 = 48 КГц.

Еще одной особенностью сигналов синхронизации является необходимость генерации сигнала кадровой синхронизации (RFS0→SYNC) до начала передачи информационных битов кадра, причем задержка должна быть равна длительности между двумя последовательными сигналами BIT_CLK. Чтобы общее количество сигналов BIT_CLK при этом осталось неизменным (256), сигнал кадровой синхронизации генерируется во время передачи последнего бита предыдущего кадра. Такой режим реализуется путем записи значения "1" в поле MFD регистра STCTL0.

3.2.2. Протокол взаимодействия с кодеком в режиме Slot-16

Структура кадров при приеме и передаче данных приведена на рис. 3.5. В слоте № 1 располагается адрес регистра управления кодека (CMD_ADR и STATUS_ADR), а в слоте № 2 – либо значение, которое он содержит (в принимаемом от кодека кадре – STATUS_DATA), либо значение для записи в регистр (в передаваемом кадре – CMD_DATA). В следующих двух слотах располагаются ИКМ-отсчеты аудиосигнала для первого (master) кодека AD1881. Если кодек только один, то используются только слоты № 0-4. При этом слоты № 5-15 не используются и все их биты равны нулю. В случае, если параллельно первому кодеку подключен еще один или два кодека (в slave-кодеки), то используются соответственно слоты № 5-6 или 5-8.

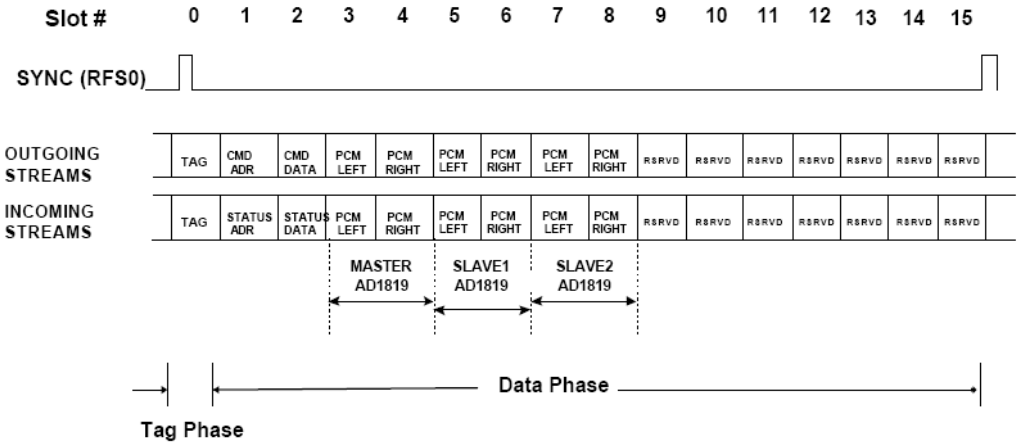


Рис. 3.5. Структура кадра при приеме и передаче данных в режиме Slot-16

[2] Обычно частота дискретизации звукового сигнала (sample rate) соответствует частоте кадров аудиосигнала (frame rate). Однако частота дискретизации может отличаться от частоты кадров аудиосигнала и варьироваться в диапазоне от 7 до 48 КГц.

Протокол обмена данными с кодеком AC'97 предусматривает наличие специального 16-битового временного слота #0 (называемого Tag Phase). Каждый бит этого слота содержит признак корректности соответствующего ему по номеру временного слота в пределах данного кадра. Если признак корректности для какого-либо слота равен "0", то передающая сторона обязана обнулить все биты, передаваемые в данный временной интервал. Больше того, если во временном слоте какие-либо биты не используются, то они также должны быть равны "0" – за выполнение этого требования отвечает передающая сторона.

Данные во всех временных слотах передаются в формате MSB (от старшего бита к младшему).

При *передаче* кадра (ADSP(DT0)→AD1881(SDATA_OUT)) первый переданный бит нулевого слота (бит № 15) является флагом корректности всего кадра ("Valid Frame"). Если он равен "1", то в кадре есть хотя бы один временной слот с корректными данными. Следующие 12 битов нулевого слота являются флагами наличия информации в соответствующих временных слотах (от 1-го до 12-го). Последние (младшие) 3 бита нулевого слота в режиме Slot-16 всегда равны "0", поскольку в режиме Slot-16 общее количество используемых слотов также не должно превышать 13.

При *приеме* кадра (AD1881(SDATA_IN)→ADSP(DR0)) первый принятый бит нулевого слота (бит № 15) является флагом готовности кодека к работе ("Codec Ready") после включения питания. Когда этот флаг установлен, регистры управления и статуса и подсистемы кодека готовы к работе. Поэтому до начала работы с кодеком следует дождаться, пока этот флаг не станет равным единице[3].

Если флаг готовности кодека установлен, то остальные 15 битов нулевого слота показывают содержат ли остальные слоты (с 1-го по 15-й) корректные данные.

3.2.3. Примеры структур данных процессора SHARC ADSP для взаимодействия с кодеком AD1881

Для организации эффективного обмена данными между DSP-процессором и кодеком следует использовать цепочечные DMA-пересылки по каналам № 0 (прием данных от кодека) и № 2 (передача данных кодеку). DMA-канал № 0 связан с буферным регистром RX0 последовательного порта SPORT0 и пересылает блоки данных из RX0 в буфер во внутренней

[3] В дополнение к проверке бита "Codec Ready" полезно проверять готовность всех подсистем кодека. Для этого можно также в цикле запросить и прочитать (в последующих кадрах) значение всех регистров кодеков.

памяти. DMA-канал № 2 связан с буферным регистром передачи порта SPORT0 и используется для пересылки блоков данных из внутренней памяти в TX0.

Для хранения каждого кадра используется буфер во внутренней памяти. Поскольку на плате установлен только один кодек (master-кодек) и количество используемых временных слотов равно 5, то и для приема, и для передачи достаточно использовать по одному буферу размером 5 слов (если частота дискретизации (sample rate) отличается от частоты передачи кадров (frame rate), то рекомендуется использовать другие параметры буфера, как описано в следующих разделах).

Для задания параметров цепочечной DMA-пересылки по каждому из каналов также достаточно одного TCB-блока (рис. 3.6):

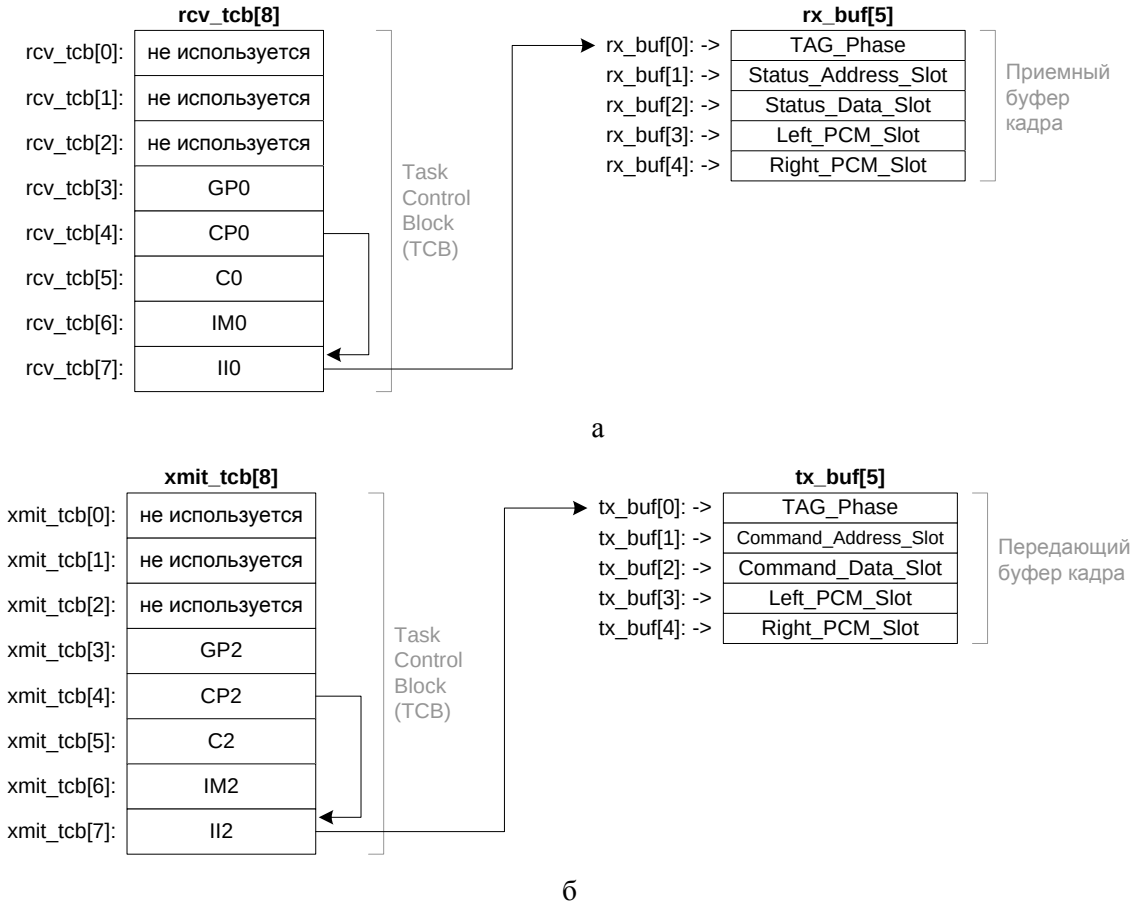


Рис. 3.6. Структуры данных процессора SHARC ADSP для работы с кодеком AD1881: а – вектор параметров DMA-пересылки и буфер приема данных от кодека; б – вектор параметров DMA-пересылки и буфер передачи данных кодеку

Поскольку DMA-пересылка является цепочечной, то после загрузки значений в регистры DMA-контроллера необходимо установить PCI-бит в регистрах CPx (и для канала передачи, и для канала приема), чтобы разрешить генерацию прерывания по завершению пересылки очередного кадра.

3.2.4. Регистры управления и состояния кодека AD1881

После установки бита CodecReady следует записать необходимые значения в регистры кодека. В приложении В приведены регистры кодека стандарта AC'97 с расширениями, реализованными в микросхеме AD1881. В приложении также описаны основные управляющие и статусные биты регистров.

Имена регистров для использования в программе могут быть взяты из файла определений AD1881_Registers.h, находящегося в каталоге VisualDSP++, в подкаталогах с примерами к плате EZ-KIT Lite.

3.2.5. Структура драйвера аудиокодека AD1881

Укрупненная структура типовой подсистемы взаимодействия программы пользователя с кодеком AD1881 приведена на рис. 3.7.

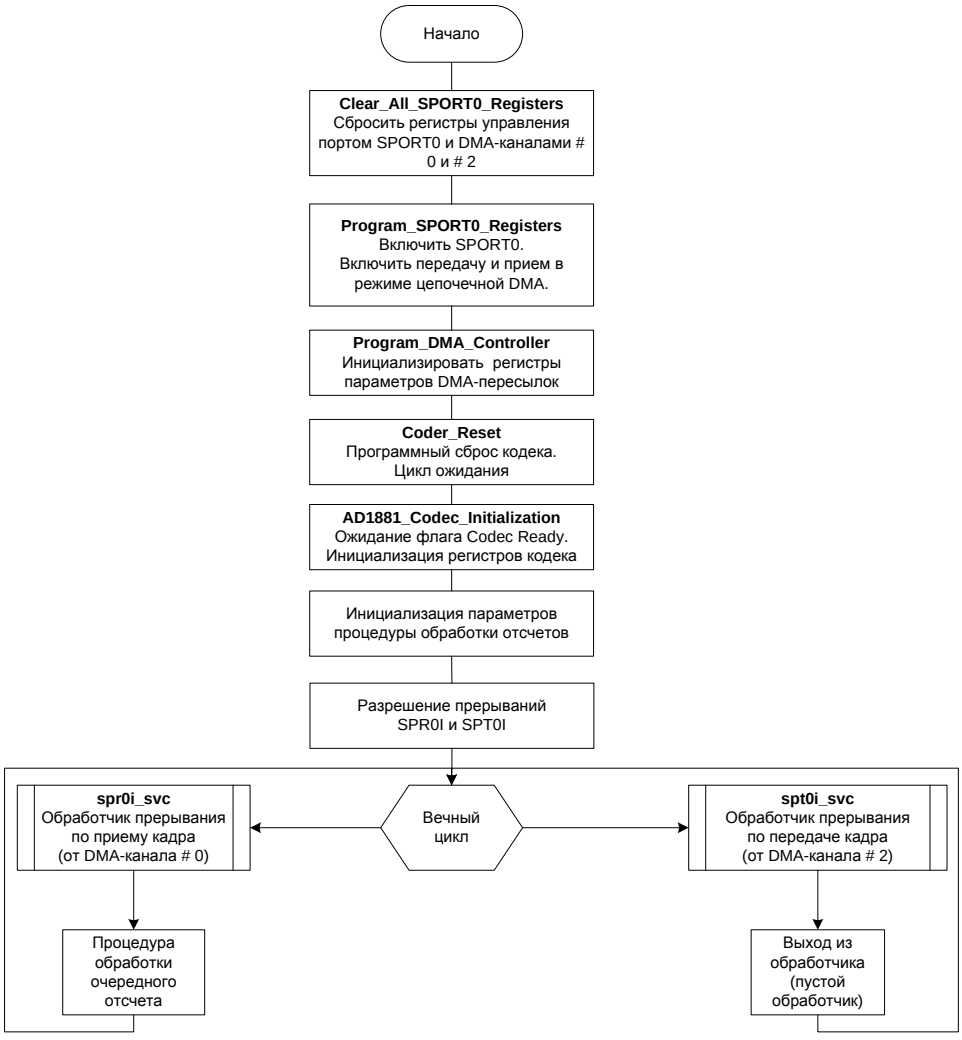


Рис. 3.7. Структура драйвера аудиокодека AD1881

Перед началом работы (и взаимодействия с кодеком) пользовательская программа должна установить параметры последовательного порта и DMA-каналов для приема и передачи кадров данных и проинициализировать регистры управления кодеком.

Вызов подпрограммы сброса регистров управления последовательным портом SPORT0 и DMA-контроллером необходим только в том случае, когда пользовательское приложение выполняется на плате EZ-KIT Lite одновременно с программой-монитором, которая отвечает за взаимодействие между платой и VisualDSP++ и также может использовать порты ввода/вывода.

Сброс кодека осуществляется путем выставления низкого уровня на линию FLG3 (для EZ-KIT Lite ADSP-2116x) с последующим ожиданием (около 3 мс) и выставлением высокого уровня на линию FLG3.

```
  bit clr FLAGS FLG3;
  r0 = 0x40000;
  lcntr = r0, do Reset_End UNTIL LCE;
Reset_End: NOP;
  bit set FLAGS FLG3;
Wait_Codec_Ready:
  r0 = DM(rx_buf+0);            // rx_buf – буфер приема кадра
  r1 = 0x8000;                  // маска с установленным битом № 15
  r0 = r0 AND r1;
  IF sz JUMP Wait_Codec_Ready;  // пока флаг Codec Ready равен "0" – ждать
```

Процедура инициализации содержит цикл по ожиданию активного значения флага Codec Ready в старшем бите временного слота # 0 (в начальном слове приемного буфера rx_buf). Затем в цикле выполняется инициализация основных регистров кодека. Следует отметить, что при программировании регистров кодека целесообразно для отслеживания окончания передачи кадра использовать прерывание по передаче кадра (spt0i). Обработчик этого прерывания не содержит иных команд кроме инструкции возврата.

Код для инициализации очередного регистра кодека может представлять собой следующую конструкцию (при разрешенном прерывании по передаче кадра):

```
   // массив адресов и значений для инициализации регистров кодека
   // в формате: адрес1, значение1, адрес2, значение2, адрес3, значение3,...
   // в режиме Slot-16
.var Init_CodecRegisters[cnNumRegs*2] =
  0x0200, 0x0000,               // MASTER_VOLUME = 0x0000
  0x0600, 0x8000,               // MASTER_VOLUME_MONO = 0x8000 (выкл.)
  0x0A00, 0x8000,               // PC_BEEP_VOLUME = 0x8000 (выкл.)
  ...;
```

```
  i4 = Init_Codec_Registers;
  r12 = 0xE000;                  // включить Valid Frame и слоты # 1 и 2
  lcntr = cnNumRegs, do Codec_Init until lce;
    dm(tx_buf + 0) = r12;
    r1 = dm(i4, 1);              //загрузить инициализационные регистры кодека
    dm(tx_buf + 1) = r1;         // адрес регистра в слот Command Address Slot
    r1 = dm(i4, 1);
    dm(tx_buf + 2) = r1;         // значение в слот Command Data Slot
Codec_Init:  idle;
```

Если кодек должен функционировать с частотой дискретизации (sample rate), отличающейся от частоты кадров (frame rate), то необходимо выключить АЦП и ЦАП и затем вновь включить их. Время выключения АЦП и ЦАП составляет 1 нс. Если тактовая частота процессора (MIPS) составляет 80 МГц, то количество тактов ожидания составляет 60 процессорных циклов. Время включения схем АЦП и ЦАП составляет 1 мс, поэтому количество тактов ожидания составляет 80000 процессорных циклов.

```
PowerDown_DACs_ADCs:
  r12 = 0xE000;                  // флаг Valid Frame и слоты # 1 и 2
  dm(tx_buf + 0) = r12;
  r0 = 0x2600;                   // регистр POWERDOWN_CTRL_STAT
  dm(tx_buf + 1) = r0;
  r0 = 0x0300;                   // выключить ЦАП и АЦП
  dm(tx_buf + 2) = r0;
  idle;                          // ждать 2 кадра
  idle;
  LCNTR = 80, DO reset_loop UNTIL LCE;
reset_loop: NOP;                 // 60 циклов для выключения ЦАП/АЦП
  idle;
  r12 = 0xE000;                  // флаг Valid Frame и слоты # 1 и 2
  dm(tx_buf + 0) = r12;
  r0 = 0x2600;                   // регистр POWERDOWN_CTRL_STAT
  dm(tx_buf + 1) = r0;
  r0=0;                          // включить ЦАП и АЦП
  dm(tx_buf + 2) = r0;
  idle;                          // ждать 2 кадра
  idle;
  LCNTR = 80000, DO warmup_loop UNTIL LCE;
warmup_loop: NOP;                // 80000 циклов для включения ЦАП/АЦП
```

Пример инициализации кодека AD1881 приведен в приложении С.

3.2.6. Особенности обработки кадров в режиме многоканального приема/передачи по сигналам прерываний от DMA-контроллера

Выше было показано, что минимально необходимое количество слов, соответствующее числу временных слотов для варианта единственного подключенного к AC-link кодека AD1881, равно пяти. Обработка события, связанного с завершением получения или передачи очередного кадра, может выполняться по сигналам прерывания от DMA-контроллера по приему (прерывание spr0i от DMA-канал № 0) или передаче (прерывание spt0i от DMA-канала № 2).

Реализация обменов данными между внутренней памятью и портами ввода/вывода через DMA-контроллера в процессорах SHARC ADSP имеет некоторые особенности. Они проявляются в рассогласовании моментов генерации сигналов прерываний по приему буфера и по передаче буфера. На рис. 3.8 приведен пример, иллюстрирующий несоответствие моментов прерываний по передаче и приему данных для буферов данных из 5 элементов.

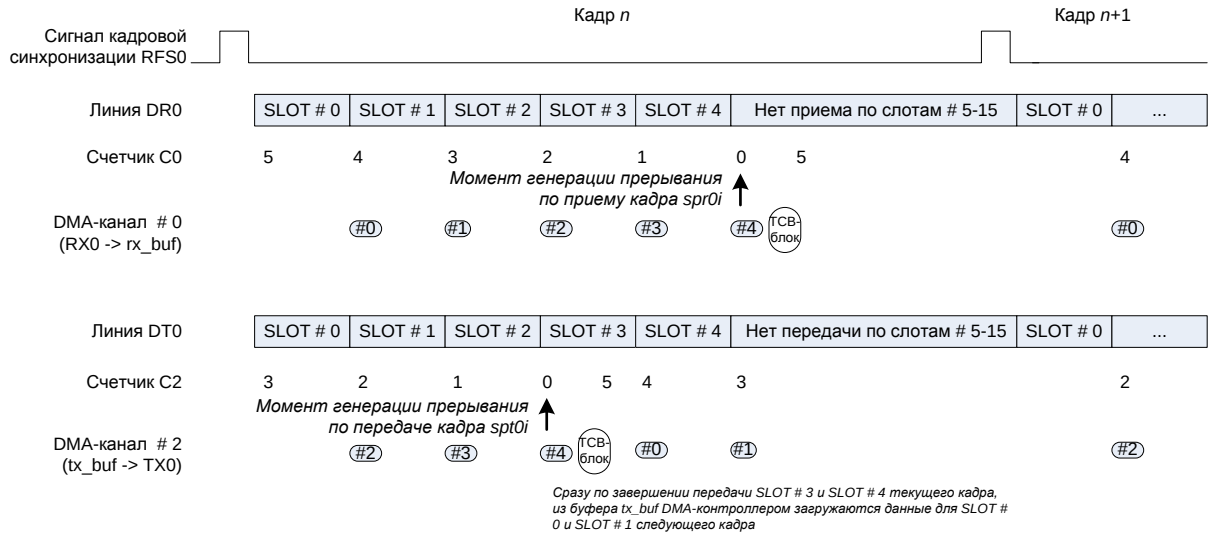


Рис. 3.8. Рассогласование моментов генерации сигналов прерываний по приему и передаче кадра. Формирование и пересылка в порт управляющих тэгов следующего кадра осуществляется до завершения приема текущего кадра. Практически отсутствует время на обработку ИКМ-отсчетов

Следует отметить, что в рассматриваемом режиме цепочечной DMA прерывание по приему текущего кадра от DMA-канала № 0 всегда происходит на два временных слота позже, чем прерывание по передаче текущего кадра от DMA-канала № 2[4]. Это объясняется тем, что генерация DMA-прерывания *при приеме* происходит после фактического получения последнего слова блока данных и его передачи во внутреннюю память. А генерация прерывания от DMA-контроллера *при*

[4] С учетом временных соотношений стандарта AC'97 и тактовой частоты процессора (80 МГц) можно оценить количество тактов (возможных инструкций) между этими сигналами (1 временной слот):

(1 / 12,288 МГц) x (16 бит/слот) x (1 слот) ≈ 1,302 мс. 1 / 80 МГц = 0,0125 мс. 1,302 мс / 0,0125 мс ≈ 104 такта.

передаче происходит в момент переноса последнего слова по I/O-шине из внутренней памяти в буферный регистр порта. В этот момент в двухуровневом FIFO-буфере порта находятся два слова, которые будут передаваться в течение двух последующих временных слотов. После прерывания spt0i происходит перезагрузка TCB-блока для DMA-канала № 2 и затем в буферные регистры TX0 из буфера передачи tx_buf во внутренней памяти начинают копироваться данные, соответствующие слоту #0 (TAG_PHASE) и слоту # 1 следующего кадра. Таким образом, слова данных нулевого и первого слотов для следующего кадра передаются в буфер порта SPORT0 еще до того момента, как будет закончено получение ИКМ-отчетов текущего кадра. И хотя передаваться кодеку эти данные начнут только после генерации очередного сигнала кадровой синхронизации RFS0, изменить, например, статусные биты временного слота # 0 следующего передаваемого кадра по итогам анализа полученных ИКМ-отсчетов текущего кадра уже невозможно.

Наиболее простым и универсальным решением данной проблемы является использование 16-словного буфера для передачи данных и, соответственно, 16-словной DMA-пересылки (рис. 3.9)[5].

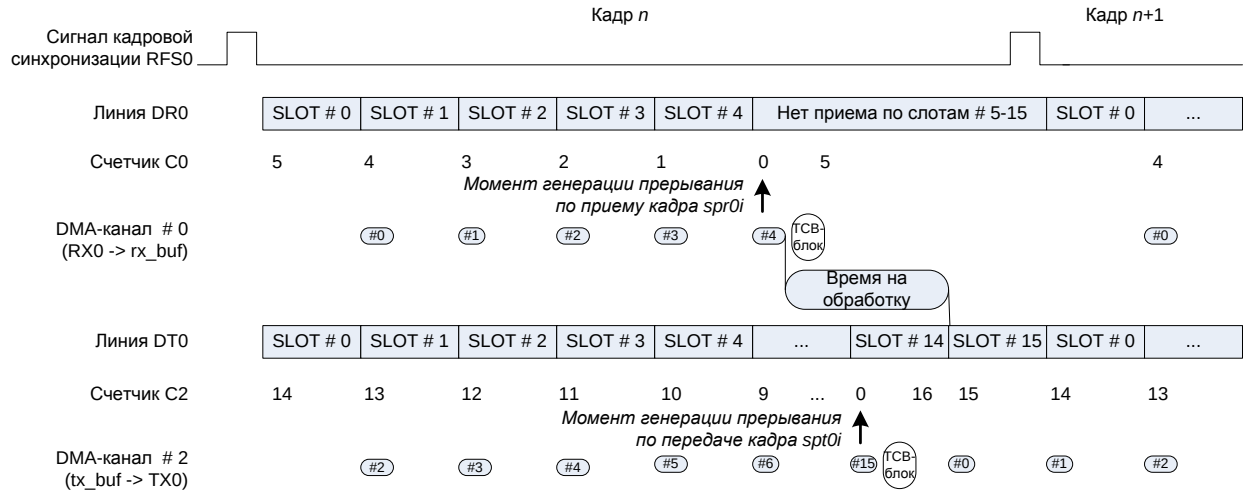


Рис. 3.9. Решение проблемы рассогласования моментов генерации прерываний по приему и передаче кадров данных. За счет активного использования большего числа временных слотов и более длинной DMA-пересылки появляется временной интервал между получением текущего кадра и формированием следующего

Передача кадров между кодеком и контроллером всегда осуществляется с частотой 48 КГц (sample rate – 48 кадров в секунду). В то же время кодек AD1881 позволяет организовать обработку отсчетов в ЦАП и АЦП с частотой дискретизации, отличающейся от 48 КГц (sample rate от 7 КГц до 48 КГц), причем частота дискретизации ЦАП может отличаться от частоты дискретизации АЦП. Если частота дискретизации аудиосигнала в АЦП и ЦАП совпадает с частотой кадров (48 КГц), то передача и прием ИКМ-отсчетов всегда происходит в каждом кадре.

[5] С учетом временных соотношений стандарта AC'97 и тактовой частоты процессора (80 МГц) можно оценить предельную длительность процедуры обработки кадра (10 временных слотов):
(1 / 12,288 МГц) x (16 бит/слот) x (10 слотов) ≈ 13,02 мс. 1 / 80 МГц = 0,0125 мс. 13,02 мс / 0,0125 мс ≈ 1040 тактов.

Если же частота дискретизации меньше частоты кадров, то контроллер должен использовать дополнительные механизмы для определения момента поступления от кодека очередного отсчета и момента необходимости передачи в кодек очередного отсчета.

Вариант 1. Частота дискретизации (sample rate) и частота кадров (frame rate) совпадают и равны частоте кадров (48 КГц). Данный вариант является наиболее простым для программной реализации. ИКМ-отсчеты левого и правого каналов передаются и принимаются в каждом кадре и биты корректности соответствующих слотов всегда установлены, поэтому их не нужно проверять при получении очередного кадра. Процедура обработки очередного полученного отсчета вызывается из обработчика прерывания по приему кадра (spr0i_svc). Для реализации данного варианта вначале достаточно использовать 5 активных временных слотов и для приема, и для передачи данных, задать буферы для хранения данных размером по 5 слов каждый и TCB-блоки на 5 пересылок в каждом блоке DMA. При этом по рассмотренным выше причинам возможно возникновение постоянной задержки между входными и выходными отсчетами размером в один кадр, что, однако, не оказывает сколько-нибудь заметного влияния на слуховое восприятие сигнала.

Массив tx_buf может состоять из 5 слов и первоначально содержит значения кадра, передача которого необходима для перевода кодека в необходимое состояние:

```
rx_buf[5];
tx_buf[5] = 0xE000,   // сначала Valid Frame = 1, корректные слоты # 0, 1 и 2
            0x7400,    // адрес регистра – 0x74. Запись регистра
            0x0F80,    // 0xF8 – для записи в регистр 0x74
            0x0000,
            0x0000;
rcv_tcb[8] = 0, 0, 0, 0, 0, 5, 1, rx_buf;
xmit_tcb[8] = 0, 0, 0, 0, 0, 5, 1, tx_buf;
```

Первая пересылка буфера tx_buf выполняется, когда кодек еще функционирует в стандартном режиме AC'97, и нулевой слот имеет размер 16 битов, а остальные – по 20 битов. Поэтому старшие 4 бита значения tx_buf[2] фактически будут переданы как младшие 4 бита первого временного слота. В соответствии с приложением B, значение 0xF8, которое записывается в регистр 0x74 конфигурации порта, соответствует переводу кодека в режим Slot-16, активации всех кодеков и разрешения режима DRQEN, в котором в каждом кадре в контроллер от кодека в слотах № 1 и 2 передается текущее значение регистра 0x74.

В регистры управления портами следует занести значения:

```
SRCTL0 ←  0x1F8C40F0; // MCE = 1 (многокан. режим приема/передачи вкл.)
```

```
                        // NCH = 15 (16 каналов)
                        // SLEN = 15 (16 битов каждое слово)
                        // IRFS = 1 (внутренний сигнал кадровой синхронизации)
                        // SDEN = 1 (разрешение DMA)
                        // SCHEN = 1 (режим цепочечной DMA)
RDIV0 ← 0x00FF0000;  // RFSDIV0 = 255 (256 битов в кадре)

STCTL0 ← 0x001C00F0; // SLEN = 15 (16 битов на каждое слово)
                        // MFD = 1 (задержка в один BIT_CLK после SYNC)
                        // SDEN = 1 (разрешение DMA)
                        // SCHEN = 1 (режим цепочечной DMA)

MRCS0, MTCS0 ← 0x0000001F;      // включить прием и передачу по каналам 0..4
MRCCS0, MTCCS0 ← 0x00000000;    // откл. компандир. при приеме и передаче
```

После получения сигнала Codec Ready в tx_buf[1] и tx_buf[2] можно раз и навсегда записать нулевые значения, а в tx_buf[0] – значение 0xB000, соответствующее установленному флагу Valid Frame и "разрешенным" слотам № 2 и 3 (PCM Left и PCM Right), либо выполнять эти обновления перед передачей каждого кадра.

Вариант 2. Метод "ADC Valid Bits" для случая, когда частота дискретизации ЦАП совпадает с частотой дискретизации АЦП и меньше 48 КГц. В данном случае контроллер должен отправлять кодеку (в ЦАП) кадр с выходными ИКМ-отсчетами с той же частотой, что и принимает корректные ИКМ-отсчеты от кодека (с АЦП). Поскольку частота дискретизации (sample rate) меньше частоты кадров (frame rate), то получение/передача ИКМ отсчетов происходит не в каждом кадре. Заданная частота дискретизации в АЦП и ЦАП поддерживается кодеком. Формирование входных отсчетов в АЦП и их вставку в "правильные" (с учетом заданной частоты дискретизации) кадры выполняет кодек. Задача контроллера – обеспечить вставку выходных отсчетов для ЦАП также в "правильные" кадры.

Для этого в процедуре обработки проверяется наличие ИКМ-отсчетов от АЦП (путем проверки статусных битов в слоте № 0 – "ADC Valid Bits") в последнем полученном кадре. Если эти слоты содержат информацию, то контроллер должен заполнить выходные слоты № 3 и 4 (и соответствующие статусные биты в слоте № 0) для формируемого выходного кадра. Если же в последнем полученном кадре ИКМ-отсчеты от АЦП не получены, то контроллер сбрасывает статусные биты в слоте № 0, соответствующие ИКМ-слотам № 3 и 4 и обнуляет эти слоты.

Поскольку дискретизация левого и правого канала может осуществляться с различными частотами, то проверку ADC Valid Bits по каждому из каналов (и формирование ИКМ-отсчетов

для следующего кадра) следует выполнять независимо.

В данном варианте также можно использовать 5 временных слотов, но для минимизации задержки между получением и формированием кадра рекомендуется использовать большее число слотов. В приведенном ниже примере массив tx_buf состоит из 16 слов и первоначально содержит значения кадра, передача которого необходима для перевода кодека в состояние, соответствующее рассматриваемому режиму:

```
rx_buf[5];
tx_buf[16] = 0xE000,  // флаг Valid Frame = 1, корректные слоты # 0, 1 и 2
           0x7400,    // адрес регистра – 0x74. Запись регистра
           0x0F80,    // 0xF8 – для записи в регистр 0x74
           0x0000,
           0x0000,
           0x0000, 0x0000, 0x0000, 0x0000, 0x0000,    // 11 пустых слотов
           0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000;
rcv_tcb[8] = 0, 0, 0, 0, 0, 5, 1, rx_buf;
xmit_tcb[8] = 0, 0, 0, 0, 0, 16, 1, tx_buf;
```

В регистры управления портами следует занести значения:

```
SRCTL0 ←  0x1F8C40F0; // MCE = 1 (многоканальный режим приема/передачи вкл.)
                      // NCH = 15 (16 каналов)
                      // SLEN = 15 (16 битов каждое слово)
                      // IRFS = 1 (внутренний сигнал кадровой синхронизации)
                      // SDEN = 1 (разрешение DMA)
                      // SCHEN = 1 (режим цепочечной DMA)
RDIV0 ← 0x00FF0000;  // RFSDIV0 = 255 (256 битов в кадре)

STCTL0 ← 0x001C00F0; // SLEN = 15 (16 битов на каждое слово)
                      // MFD = 1 (задержка в один BIT_CLK после SYNC)
                      // SDEN = 1 (разрешение DMA)
                      // SCHEN = 1 (режим цепочечной DMA)

MRCS0 ← 0x0000001F;  // включить прием по каналам 0..4
MTCS0 ← 0x0000FFFF;  // включить передачу по каналам 0..15
MRCCS0, MTCCS0 ← 0x00000000;    // откл. компандир. при приеме и передаче
```

В подпрограмме обработки полученного кадра выполняются следующие действия:

1. Проверяется наличие ИКМ-данных в слотах № 3 и 4 путем анализа слота № 0 последнего полученного кадра, например:

```
check_ADCs_for_valid_data:
  r0 = dm(rx_buf+0);  // прочитать слово из слота # 0
  r1 = 0x1800;        // маска = 0001100..0 (биты соотв. слотам # 3 и 4)
```

```
  r2 = r0 and r1;     // взять биты корректности слотов # 3 и 4 из слота # 0
  dm(ADC_valid_bits) = r2; // сохранить биты ADC Valid Bits для двух каналов
```

2. По результатам проверки устанавливаются биты в тэге передаваемого кадра (слот № 0) биты корректности для слотов № 3 и 4. Этот шаг необходимо выполнить до переноса tx[0] в буфер последовательного порта:

```
set_tx_slot_valid_bits:
  r1 = dm(tx_buf+0);       // прочитать значение по умолчанию для слота # 0
  r3 = r2 or r1;           // наложить биты ADC Valid Bits (в те же слоты)
  dm(tx_buf+0) = r3;       // положить в буфер для передачи
  r3 = 0;
  dm(tx_buf+1) = r3;       // обнулить все слоты
  dm(tx_buf+2) = r3;
  dm(tx_buf+3) = r3;
  dm(tx_buf+4) = r3;
  dm(Left_Channel_Flag) = r3;   // сбросить внутренние флаги наличия
  dm(Right_Channel_Flag) = r3;  // ИКМ-отсчетов левого и правого каналов
```

3. При необходимости читаются полученные от кодека ИКМ-отсчеты и сохраняются в отдельные переменные (или массивы) для последующей обработки:

```
get_ADC_left:
  BTST r2 BY 12;           // проверка ADC Valid Bit для левого канала
  IF sz JUMP get_ADC_right;// если не установлен, то перейти к правому каналу
    r6 = dm(rx_buf+3);     // прочитать ИКМ-отсчет для левого канала (16 бит)
    r6 = lshift r6 by 16;  // сдвинуть в старшие 16 битов (формат 1.31)
    dm(Left_Channel_In) = r6; // записать в отдельную переменную
    r4 = 1;
    dm(Left_Channel_Flag) = r4; // установить флаг наличия ИКМ левого канала
get_ADC_right:
  BTST r2 BY 11;           // проверка ADC Valid Bit для правого канала
  IF sz rti;               // если не установлен, то завершить разбор кадра
    r6 = dm(rx_buf+4);     // прочитать ИКМ-отсчет для левого канала (16 бит)
    r6 = lshift r6 by 16;  // сдвинуть в старшие 16 битов (формат 1.31)
    dm(Right_Channel_In) = r6; // записать в отдельную переменную
    r4 = 1;
    dm(Right_Channel_Flag) = r4;// установить флаг наличия ИКМ правого канала
```

4. Вызывается алгоритм обработки полученных отсчетов:

```
user_applic:
  call DSP_Audio_Routine;
```

5. Формируется выходной кадр (с учетом битов ADC Valid Bits):

```
playback_audio_data:
  r2 = dm(ADC_valid_bits); // прочитать ADC Valid Bits
put_DAC_left:
  BTST r2 BY 12;                // (проверка ADC Valid Bit для левого канала)
  IF sz JUMP put_DAC_right;     // если сброшен, то не записывать левый канал
    r15 = dm(Left_Channel_Out); // взять результат обработки
    r15 = lshift r15 by -16;    // сдвинуть в младшие 16 битов регистра
    dm(tx_buf+3) = r15;         // записать в слот № 3
put_DAC_right:
  BTST r2 BY 11;                // проверка ADC Valid Bit для правого канала
  IF sz JUMP tx_done;           // если сброшен то не записывать правый канал
    r15 = dm(Right_Channel_Out); // взять результат обработки
    r15 = lshift r15 by -16;    // сдвинуть в младшие 16 битов регистра
    dm(tx_buf+4) = r15;         // записать в слот № 4
tx-done:
  ...
```

Способ взаимодействия контроллера с кодеком на основе проверки ADC Valid Bits позволяет обеспечивать обработку и воспроизведение аудиосигналов во всех случаях, когда частота дискретизации (sample rate) ЦАП равна частоте дискретизации (sample rate) АЦП и отличается от частоты кадров (frame rate).

Примеры программ, организующих взаимодействие с кодеком с использованием метода ADC Valid Bits, находятся в подкаталоге 211xx\Examples\ADSP-2116x EZ-KIT Lite\ каталога установки среды VisualDSP++.

Вариант 3. Метод "DAC Request Bits" для случая, когда частота дискретизации ЦАП не равна частоте дискретизации АЦП. Данный вариант немного более сложен для реализации и касается процедуры формирования выходного кадра. Это связано с тем, что контроллер должен уметь помещать обработанные ИКМ-отсчеты в слоты № 3 и 4 передаваемого кадра только в те моменты, когда ЦАП кодека готов их принять. В противном случае возможны искажения в процессе цифро-аналогового преобразования.

Кодек AD1881 выставляет бит запроса DRRQx ИКМ-отсчета в i-м кадре, если ЦАП кодека по соответствующему каналу готов принять этот отсчет в (i+1)-м кадре. Это позволяет освободить контроллер от функций контроля частоты дискретизации выходного сигнала. Задачей контроллера является своевременное обнаружение этого бита и формирование следующего кадра с включенными в него ИКМ-отсчетами для вывода (воспроизведения).

Для отслеживания активного уровня флага DRRQx контроллер должен анализировать биты регистра конфигурации кодека 0x74. Запрос для левого канала ЦАП соответствует единичному

биту DLRQ0 (бит № 8 в регистре 0x74), для правого – единичному биту DRRQ0 (№ 0 в регистре 0x74).

Возможность для эффективного покадрового отслеживания этих битов (без специального запроса контроллера) предоставляется кодеком, если в его регистре конфигурации 0x74 установлен флаг DRQEN. В этом случае в соответствии со структурой кадра информация о флагах DRRQx (DAC Request Bits) передается в слотах Status Address Slot и Status Data Slot следующим образом:

- слот Status Data Slot содержит значение регистра 0x74, причем активным уровнем (запрос есть), как обычно, считается единичное значение битов DLRQx и DRRQx;

- слот Status Address Slot содержит в битах № 14..8 номер регистра (0x74), а в битах № 7..2 – флаги DLRQx и DRRQx, причем активным состоянием (запрос есть) является нулевое значение этих битов.

Примерные этапы обработки данных в процессоре SHARC ADSP для метода передачи данных по запросу могут быть записаны следующим образом:

1. Проверяется активность битов DLRQ0 и DRRQ0 во временном слоте № 1 (этот слот содержит биты DAC Request Bits для левого и правого каналов в позициях № 7 и 6 соответственно, причем активное состояние этих битов – нулевое!).

```
check_DAC_request_bits:
  r1 = dm(rx_buf+1);            // взять Status Address Slot
  r0 = 0x00C0;                  // маска для выделения позиций № 7 и 6
  r2 = r1 AND r0;               // выделить DAC Request Bits (DLRQ0 и DRRQ0)
  dm(DAC_request_bits) = r2;    // сохранить DAC Request Bits
```

2. По результатам проверки в тэге передаваемого кадра (слот № 0) устанавливаются биты корректности для слотов № 3 и 4. Этот шаг необходимо выполнить до переноса tx[0] в буфер последовательного порта:

```
set_tx_slot_valid_bits:
     // установить биты корректности слотов № 3 и 4
     // (по значениям DAC Request Bits)
  r2 = r2 XOR r0;     // инвертировать DAC Request Bits (активное сост. = 1)
  r2 = LSHIFT r2 BY 5; // на места битов корректности слотов № 3 и 4
  r0 = 0x8000;        // установить флаг Valid Frame для передачи
  r2 = r2 or r0;      // сформировать тэг для временного слота # 0
  dm(tx_buf+0) = r2;  // положить в буфер для передачи
  r3 = 0;
  dm(tx_buf+1) = r3;  // обнулить все остальные слоты
  dm(tx_buf+2) = r3;
  dm(tx_buf+3) = r3;
```

```
  dm(tx_buf+4) = r3;
```

3. Как и в предыдущем варианте проверяется наличие ИКМ-данных в слотах № 3 и 4 путем анализа слота № 0 последнего полученного кадра, например:

```
check_ADCs_for_valid_data:
  r0 = dm(rx_buf+0);  // прочитать слово из слота # 0
  r1 = 0x1800;        // маска = 0..011000000 (биты соотв. слотам # 3 и 4)
  r2 = r0 and r1;     // взять биты корректности слотов # 3 и 4 из слота # 0
  dm(ADC_valid_bits) = r2; // сохранить биты ADC Valid Bits для двух каналов
```

4. При необходимости читаются полученные от кодека ИКМ-отсчеты и сохраняются в отдельные переменные (или массивы) для последующей обработки:

```
get_ADC_left:
  r3 = 0;
  dm(Left_Channel_Flag) = r3;   // сбросить внутренние флаги наличия
  dm(Right_Channel_Flag) = r3;  // ИКМ-отсчетов левого и правого каналов
  BTST r2 BY 12;           // проверка ADC Valid Bit для левого канала
  IF sz JUMP get_ADC_right;// если не установлен, то перейти к правому каналу
    r6 = dm(rx_buf+3);     // прочитать ИКМ-отсчет для левого канала (16 бит)
    r6 = lshift r6 by 16;  // сдвинуть в старшие 16 битов (формат 1.31)
    dm(Left_Channel_In) = r6; // записать в отдельную переменную
    r4 = 1;
    dm(Left_Channel_Flag) = r4; // установить флаг наличия ИКМ левого канала
get_ADC_right:
  BTST r2 BY 11;           // проверка ADC Valid Bit для правого канала
  IF sz rti;               // если не установлен, то завершить разбор кадра
    r6 = dm(rx_buf+4);     // прочитать ИКМ-отсчет для левого канала (16 бит)
    r6 = lshift r6 by 16;  // сдвинуть в старшие 16 битов (формат 1.31)
    dm(Right_Channel_In) = r6; // записать в отдельную переменную
    r4 = 1;
    dm(Right_Channel_Flag) = r4;// установить флаг наличия ИКМ правого канала
```

4. Вызывается подпрограмма обработки полученных отсчетов (в зависимости от алгоритма обработки вызов подпрограммы может осуществляться либо из обработчика прерывания по приему кадра, либо из обработчика прерывания по передаче кадра):

```
user_applic:
  call DSP_Audio_Routine;
```

5. Формируется выходной кадр (с учетом битов DAC Request Bits по каждому каналу, их активное состояние – нулевое!):

```
playback_audio_data:
```

```
  r2 = dm(DAC_request_bits);     // прочитать DAC Request Bits

put_DAC_left:
  BTST r2 BY 7;                  // проверка DAC Request Bit для левого канала
  IF not sz JUMP put_DAC_right; // если =1, то не записывать левый канал
    r15 = dm(Left_Channel_Out); // взять результат обработки
    r15 = lshift r15 by -16;     // сдвинуть в младшие 16 битов регистра
    dm(tx_buf+3) = r15;          // записать в слот № 3
put_DAC_right:
  BTST r2 BY 11;                 //проверка DAC Request Bit для правого канала
  IF not sz JUMP tx_done;        // если =1, то не записывать правый канал
    r15 = dm(Right_Channel_Out); // взять результат обработки
    r15 = lshift r15 by -16;     // сдвинуть в младшие 16 битов регистра
    dm(tx_buf+4) = r15;          // записать в слот № 4
tx-done:
  ...
```

Рассмотренный подход к реализации драйвера позволяет передавать кодеку обработанные ИКМ-отсчеты по активному состоянию битов DAC Request Bits (по запросу).

Если частота дискретизации (sample rate) кратна частоте кадров (frame rate), например, 8 КГц, 12 КГц, 16 КГц, то биты ADC Valid Bits и биты запросов DAC Request Bits появляются с определенным периодом и процедура обработки ИКМ-отсчетов аудиосигнала также должна вызываться строго через заданное количество аудиокадров (соответственно, через 6, 4, 3 кадров). Если же частота дискретизации не удовлетворяет этому условию (например, 44,1 КГц, 22,05 КГц), то моменты появления битов ADC Valid Bits и DAC Request Bits определяются самим кодеком и трудно прогнозируемы. Пример распределения по кадрам битов наличия входных данных (ADC Valid Bits) и битов-запросов на передачу выходных данных (DAC Request Bits) по нескольким последовательно получаемым кадрам приведен в табл. 3.3.

Таблица 3.3

Пример распределения входных и выходных ИКМ-отсчетов по слотам кадров

Номер кадра	*N*	*N+1*	*N+2*	*N+3*	*N+4*	*N+5*	*N+6*
ADC Valid Bits	да	да	нет	да	нет	нет	да
DAC Request Bits	нет	нет	да	да	нет	да	да
ИКМ-левый	данные	данные	0x0000	данные	0x0000	0x0000	данные
ИКМ-правый	данные	данные	0x0000	данные	0x0000	0x0000	данные

Такое чередование входных и выходных данных связано с двумя потенциальными проблемами:

- если в нескольких последовательных кадрах поступают входные отсчеты (активны ADC Valid Bits) при отсутствии запросов DAC Request Bits, то возможна потеря данных вследствие

наложения последнего обработанного отсчета на предыдущие;

- если требуется передавать кодеку выходные отсчеты (активны DAC Request Bits) в нескольких последовательных кадрах при отсутствии входных данных (неактивны ADC Valid Bits), то возможно повторная передача одного и того же обработанного отсчета.

Для решения этих проблем рекомендуется использовать для хранения полученного и обработанного отсчетов не отдельные переменные в памяти, а массив, организованный по принципу циклического буфера. Поскольку частоты дискретизации ЦАП и АЦП по каждому каналу могут быть различны, то для хранения данных по каждому каналу целесообразно использовать раздельные массивы. По массиву хранения данных (обычно длиной от 4 до 8 слов) перемещаются два круговых указателя. Первый отвечает за помещение в массив очередного принятого ИКМ-отсчета (смещается каждый раз, когда контроллер обнаруживает бит ADC Valid Bit для соответствующего канала). Второй используется для выдачи очередного (еще не переданного) обработанного отсчета (смещается каждый раз, когда контроллер обнаруживает бит DAC Request Bit для соответствующего канала). Рекомендуется, чтобы указатель чтения из массива "отставал" от указателя записи в массив на 2-3 слова. В режиме обработки сигнала в реальном масштабе времени такая задержка является незаметной для слухового восприятия.

Пример программы, организующей взаимодействие с кодеком с использованием метода DAC Request Bits, приведен в приложении D.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Материалы по плате EZ-KIT Lite, процессору ADSP-21160 и интегрированной среде разработки VisualDSP++:

- ADSP-21160 EZ-KIT Lite Evaluation System Manual. Revision 3.0 January 2003. – www.analog.com

- VisualDSP++ 4.5. Getting Started Guide. – www.analog.com

- VisualDSP++ 4.5. Loader and Utilities Manual. – www.analog.com

- VisualDSP++ 4.5. User's Guide. – www.analog.com

- ADSP-21160 SHARC DSP. Hardware Reference. – www.analog.com

- ADSP-21160 SHARC DSP. Instruction Set Reference. – www.analog.com

2. Материалы по загрузке процессора SHARC ADSP при включении питания:

- VisualDSP++ 4.5. Loader and Utilities Manual. – www.analog.com

- EE-56. Tips & Tricks On Eprom / Host Booting. – www.analog.com

- EE-117. In-System-Programming (ISP) of ADSP-2106x boot-image into FLASH-Memories. – www.analog.com

- EE-199. Link Port Booting the ADSP-21161 SHARC DSP. – www.analog.com

3. Материалы по архитектуре и программированию Flash-памяти:

- EE-55. Interfacing Byte Programmed Flash Memories to the ADSP-2106x SHARC series. – www.analog.com

- STMicroelectronics. M29W040B Data Sheet. – www.st.com

4. Материалы по работе программы-монитора на плате EZ-KIT Lite:

- ADSP-21061 EZ-KIT Lite Evaluation System Manual. Revision 3.0. January, 2003. – www.analog.com

- EE-296. Using the UART Port Controller on SHARC Processors. – www.analog.com

- EE-87. Update to SHARC EZ-KIT Lite Message Packet Protocols. – www.analog.com

5. Материалы по аудиокодеку AD1881:

- Tomarakos J. Interfacing the ADSP-21065L SHARC DSP to the AD1819 'AC-97' SoundPort Codec. Version 2.4A – www.analog.com

- Audio Codec '97. Component Specification. Revision 2.3 Rev. 1.0. – www.intel.com

- How To Use AD1819A Variable Sample Rate Support. EE-note # 54. – www.analog.com

- EE-74. Analog Devices Serial Port Development and Troubleshooting Guide. – www.aanlog.com

- Tomarakos J. Interfacing the ADSP-21161 SIMD SHARC DSP to the AD1836 (24-bit/96kHz) Multichannel Codec. Example Interface Drivers in Assembly and C For Use With The 21161 EZ-KIT LITE. Version 1.0A – www.analog.com

- AC'97 SoundMAX Codec AD1881. Data Sheet. – www.analog.com

Приложение А

ОРГАНИЗАЦИЯ И КОМАНДНЫЙ ИНТЕРФЕЙС FLASH-ПАМЯТИ

А.1. Организация Flash-памяти

Плата EZKIT-Lite содержит микросхему Flash-памяти M29W040B компании STMicroelectronics объемом 512K 8-разрядных слов (байтов). Большинство микросхем Flash-памяти, предназначенных для использования в мобильных и встроенных системах, имеют унифицированный стандартизованный интерфейс для обмена данными между памятью и вычислительными устройствами и могут отличаться внутренней, невидимой для внешних устройств, организацией, емкостью и, возможно, конфигурацией выводов микросхемы.

Внутренняя организация Flash-памяти M29W040B представляет собой набор блоков одинакового размера, размещенных последовательно в адресном пространстве микросхемы. Каждый блок содержит 64K слов. Разрядность Flash-памяти (размер одной адресуемой ячейки памяти) – 8 битов.

№ блока	Размер (Кб)	Диапазон внутренних адресов
0	64	00000h – 0FFFFh
1	64	10000h – 1FFFFh
2	64	20000h – 2FFFFh
3	64	30000h – 3FFFFh
4	64	40000h – 4FFFFh
5	64	50000h – 5FFFFh
6	64	60000h – 6FFFFh
7	64	70000h – 7FFFFh

Адресная шина, соединяющая память с другими устройствами (например, DSP) – 19-разрядная, шина данных – 8-разрядная. Кроме того, для указания вида операции и разрешения доступа к микросхеме используются дополнительные сигнальные линии и линии питания.

Каждый из блоков Flash-памяти может быть заблокирован для предотвращения случайной операции записи (программирования) или стирания. При этом операции чтения могут выполняться без ограничений. Операции блокирования/разблокирования блока выполняются исключительно внешними устройствами через командный интерфейс микросхемы памяти.

Принципиальной особенностью программирования (записи) Flash-памяти является то, в "чистом" блоке памяти все биты имеют единичные значения, а команда записи может (в соответствии с записываемым словом) сбросить отдельные биты в "0". Обратите внимание, что команда записи не может изменить нулевое значение бита на единичное – для этого нужно стирание блока (или всего чипа памяти).

Взаимодействие DSP с Flash-памятью выполняется с использованием набора управляющих, адресных линий и линий данных. Следует помнить, что слова данных при обменах между Flash и DSP передаются по линиям D[32:39] внешней шины данных.

A.2. Командный интерфейс Flash-памяти

Командный интерфейс (Command Interface) Flash-памяти, используемый внешними устройствами для выполнения всех операций над памятью, соответствует стандартам JEDEC. Команда представляет собой последовательность нескольких 8-разрядных слов, содержащих управляющую, информацию и, при необходимости, адресную информацию и данные. Поступившая команда (последовательность слов) обрабатывается контроллером программирования/стирания памяти (Program/Erase Controller).

Командный интерфейс поддерживает выполнение 10 стандартных команд, основные из которых приведены в таблице. Для выполнения команды DSP должен последовательно выставлять на шины адреса и данных, соединяющих его с Flash-памятью, соответствующие комбинации адресов и значений (управляющих кодов или слов данных).

Команда	**Длина**	**Элементарные операции (обращения к памяти)**											
		1-я		**2-я**		**3-я**		**4-я**		**5-я**		**6-я**	
		Адрес	**Данные**	**Адрес**	**Данные**	**Адрес**	**Данные**	**Адрес**	**Данные**	**Адрес**	**Данные**	**Адрес**	**Данные**
Сброс	1	X	F0										
	3	555	AA	2AA	55	X	F0						
Автовыбор	3	555	AA	2AA	55	555	90						
Записать 1 слово (программирование)	4	555	AA	2AA	55	555	A0	PA	PD				
Разблокировать память	3	555	AA	2AA	55	555	20						
Записать в разблокированную память	2	X	A0	PA	PD								
Заблокировать блок	2	X	90	X	00								
Стирание чипа	6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10
Стирание блока	6+	555	AA	2AA	55	555	80	555	AA	2AA	55	BA	30
Приостановка стирания	1	X	B0										
Возобновление стирания	1	X	30										

X – произвольный адрес (не имеет значения); PA – адрес ячейки; PD – данные для записи; BA – произвольный адрес в пределах нужного блока

Обратите внимание, что при обработке элементарных операций с командными тэгами используются только линии A0-A10 адресной шины, а при обработке операций с пересылкой данных – все 19 линий адресной шины. Распознавание операций, в которых должна выполняться пересылка данных, осуществляется микросхемой памяти по уже полученным тэгам команд.

Чтение данных (!) из Flash-памяти выполняется как из обычной памяти (не нужны специальные управляющие команды), но только в том случае, если микросхема находится в режиме чтения. Программируется операция чтения с использованием PX-регистра:

```
ERROR_CODE ReadFlash( unsigned long ulOffset, int *pnValue )
{
 int nValue = 0x0;          // временная переменная
           // сформировать адрес в пространстве памяти системы
  asm ("r0 = 0x800000;");  // r0 – начало flash-памяти в пространстве ADSP
  asm ("r1 = %0;": : "k" (ulOffset) );     // r1 – смещение от начала flash
  asm ("r0 = r0 + r1;");   // адрес ячейки в пространстве памяти
  asm ("l1 = 0x0;");       // L1 = 0x0;   M1 = 0x0;  I1 =  адрес ячейки;
  asm ("m1 = 0x0;");
  asm ("i1 = r0;");
     // чтение значения с использованием PX-регистра
  asm ("px = dm(i1, m1);"); // PX[63:0] = значение. Фактически [39:32]
  asm ("r1 = px2;");       // r1 = PX[63:32]. Фактически младшие 8 битов
     // поместить во временную переменную
  asm ("%0 = r1;": "=k" (nValue) : );
     // возвращаем через указатель
  *pnValue = nValue;
  return NO_ERR;
}
```

Команда записи (программирования) значения (Program Command) записывает одно значение по заданному адресу во Flash-память. Если адрес попадает в защищенный блок, то команда игнорируется, а данные остаются неизменными. До завершения команды записи нельзя прочитать значения из памяти – в это время при чтении будет возвращаться только значение регистра статуса. После выполнения команды, если не было ошибок, микросхема автоматически переходит в режим чтения, иначе – продолжает выдавать значение регистра статуса до команды сброса.

Выполнение элементарной операции во Flash-память должно выполняться обязательно с использованием PX-регистра, например, следующим образом:

```
ERROR_CODE WriteFlash(unsigned long ulOffset,    // адрес
                      int nValue)           // значение (только младшие 8 бит)
{
     // сформировать адрес в пространстве памяти системы
  asm ("r0 = 0x800000;");  // r0 – адрес flash-памяти в пространстве ADSP
  asm ("r1 = %0;": : "k" (ulOffset) );     // r1 – смещение от начала flash
  asm ("r0 = r0 + r1;");   // адрес ячейки в пространстве памяти
  asm ("l1 = 0x0;");       // L1 = 0x0;   M1 = 0x0;  I1 =  адрес ячейки;
```

```
  asm ("m1 = 0x0;");
  asm ("i1 = r0;");
          // запись значения с использованием PX-регистра
  asm ("r1 = %0;": : "k" (nValue) );  // r1 = значение
  asm ("px2 = r1;");        // PX[63:32] = значение. Фактически [39:32]
  asm ("dm(i1, m1) = px;");      // на шину данных – 64-битное значение
  return NO_ERR;
}
```

Перед вызовом функции WriteFlash Flash-память обязательно должна быть разблокирована, например, следующим образом:

```
ERROR_CODE UnlockFlash()
{
     // часть команды Program Command, которая разблокирует память
  WriteFlash( 0x5555, 0xaa );
  WriteFlash( 0x2aaa, 0x55 );
  WriteFlash( 0x5555, 0xa0 );
  return NO_ERR;
}
```

Успешность выполнения операции записи может быть проверена путем проверки соответствующих статусных битов (см. ниже описание функции PollToggleBit) и (при необходимости) чтения только что записанного значения. Тогда полная последовательность действий для записи одного слова во Flash память можно записать следующим образом:

```
  UnlockFlash();
  WriteFlash( ulOffset, Value );
  ErrorCode = PollToggleBit();
  ReadFlash( ulOffset, &nCompare );
  if( nCompare != ( pnData[i] & 0x0000FFFF ) )
  {
     bVerifyError = TRUE;
     break;
  }
```

Команда сброса – перевода микросхемы в режим чтения (Read/Reset Command) переводит микросхему в режим чтения, в котором Flash-память ведет себя как обычное ПЗУ. Чтение значений из микросхемы в данном режиме с точки зрения процессора идентично чтению данных из памяти любого другого типа: DSP выставляет на адресную шину адрес ячейки и затем считывает с шины данных значение. Никаких дополнительных управляющих команд для доступа к ячейкам, если микросхема находится в режиме чтения, не нужно.

Пример программного кода для выполнения команды сброса:

```
ERROR_CODE ResetFlash()
{
// выполнить последовательность управляющих команд для перевода микросхемы в
// режим чтения
  WriteFlash( 0x5555, 0xaa );
  WriteFlash( 0x2aaa, 0x55 );
  WriteFlash( 0x5555, 0xf0 );
  return NO_ERR;
}
```

Команда выбора (Auto Select Command) используется для чтения кода производителя и проверки состояния блокировки блока. После выполнения управляющих команд чтение информации выполняется по фиксированным адресам (реальные значения, которые записаны по тем же адресам путем программирования flash-памяти не перезаписываются – они будут доступны вновь только после перевода микросхемы в режим чтения). При попытке чтения статуса блока (заблокирован он или нет от записи) в 16-18 биты адреса записывается 3-битовый номер блока (от 0 до 7), а в бит № 1 – единица: например для чтения статуса блока № 2 адрес должен выглядеть следующим образом: 0x20002. Если в возвращаемом слове младший бит равен 1, то блок защищен от записи/стирания, иначе – запись/стирание блока разрешена.

```
ERROR_CODE GetCodes()
{
 int *AFP_ManCode;
 int *AFP_DevCode;

     // последовательность управляющих команд
  WriteFlash( 0x5555, 0xaa );
  WriteFlash( 0x2aaa, 0x55 );
  WriteFlash( 0x5555, 0x90 );
     // теперь читаем требуемые данные по определенным адресам
  ReadFlash( 0x0000, &AFP_ManCode );  // STMicroelectronics = 0x20
  ReadFlash( 0x0001, &AFP_DevCode );  // код M29W040B = 0x3E
     // вернуть микросхему в нормальный рабочий режим чтения
     // чтобы можно было читать данные
  ResetFlash();
     // ok
  return NO_ERR;
}
```

Команда стирания чипа памяти (Chip Erase Command) используется для очистки всех Flash-

памяти, которые не защищены от записи/стирания. При этом значения всех битов устанавливаются равными "1". После выполнения команды микросхема автоматически переводится в режим чтения. Для контроля выполнения операции используется проверка статусных битов (описание функции PollToggleBit приведено ниже).

```
ERROR_CODE EraseFlash()
{
  ERROR_CODE ErrorCode = NO_ERR;// флаг: нет ошибки
           // последовательность управляющих команд
  WriteFlash( 0x5555, 0xaa );
  WriteFlash( 0x2aaa, 0x55 );
  WriteFlash( 0x5555, 0x80 );
  WriteFlash( 0x5555, 0xaa );
  WriteFlash( 0x2aaa, 0x55 );
  WriteFlash( 0x5555, 0x10 );
      // проверить статусные биты
  ErrorCode = PollToggleBit();
  return ErrorCode;
}
```

Команда стирания блока памяти (Block Erase Command) позволяет очистить один или несколько блоков памяти. Результат выполнения операции проверяется по статусным битам. Пример функции для стирания одного блока памяти приведен ниже.

```
ERROR_CODE EraseBlock( int nBlock )
{
 int nSectorOff = 0x0;
 ERROR_CODE ErrorCode = NO_ERR;             // код ошибки
      // если номер блока недопустим
  if ( (nBlock < 0) || (nBlock > AFP_NumSectors) )
      return INVALID_BLOCK;

      // последовательность управляющих команд
  WriteFlash( 0x5555, 0xaa );
  WriteFlash( 0x2aaa, 0x55 );
  WriteFlash( 0x5555, 0x80 );
  WriteFlash( 0x5555, 0xaa );
  WriteFlash( 0x2aaa, 0x55 );
      // в конце – вычислить любой адрес, попадающий в блок
  nSectorOff = (nBlock * 0x10000);
  WriteFlash( nSectorOff, 0x30 );
```

```
    // проверить успешность выполнения операции
  ErrorCode = PollToggleBit();
  return ErrorCode;
}
```

A.3. Контроль выполнения операций над Flash-памятью

По результатам выполнения операций записи, чтения или стирания памяти устанавливаются соответствующие биты в регистре статуса. Если DSP пытается прочитать содержимое какой-либо ячейки памяти во время выполнения операций записи или стирания памяти (именно во время выполнения этих операций), то вместо значения ячейки, интерфейс памяти возвращает значение регистра статуса Flash-памяти. По этому значению можно определить успешность выполнения операции записи или стирания.

Наибольший интерес в регистре статуса представляют следующие биты:

- бит № 7 (DQ7, *Data Polling Bit*). В течение циклов выполнения операции записи (программирования), пока она не завершена, ячейка DQ7 имеет значение, обратное к тому, которое DSP пытается записать в слово данных в 7-м бите. После успешного выполнения операции записи бит DQ7 принимает значение бита № 7, записываемого в слово данных. Таким образом, выполнив операцию записи и тут же после нее операцию чтения (а после успешной записи микросхема автоматически возвращается в режим чтения) и сравнив записываемое и прочитанное значения 7-го бита слова данных, можно обнаружить ошибку;

- бит № 6 (DQ6, *Toggle Bit*). Также используется для контроля успешного завершения операции записи или стирания памяти. Если во время выполнения цикла записи или стирания DQ6 DSP будет пытаться производить операции чтения из памяти, то при выполнении каждой следующей такой операции чтения значение этого бита будет меняться на противоположное;

- бит № 5 (DQ5, *Error Bit*). Бит устанавливается в "1", если операция записи или стирания памяти была выполнена с ошибкой (например, при попытке перезаписать память новыми значениями без стирания – как говорилось выше, операция записи может изменять бит только с "1" на "0").

Стандартным способом проверки успешности завершения операции записи/стирания памяти является чтение после этой операции двух подряд слов и сравнение их на идентичность. При необходимости дополнительно может выполняться чтение только что записанного слова по соответствующему адресу памяти и сравнение его значения с истинным.

```
ERROR_CODE PollToggleBit()
{
```

```
  ERROR_CODE ErrorCode = NO_ERR;

  asm ("r7 = 0;");              // счетчик тайм-аута
  asm ("r5 = 0xffffff;");       // время тайм-аута
  asm ("r2 = 0xff;");           // маска для выделения 8-битового значения

asm ("POLL_TOGGLE_BIT:");
  asm ("px = dm(0x800000);");   // прочитать
  asm ("r8 = px2;");            // сохранить
  asm ("px = dm(0x800000);");   // прочитать еще раз
  asm ("r9 = px2;");            // сохранить
  asm ("r8 = r8 and r2;");      // взять младшие 8 битов первого статуса
  asm ("r9 = r9 and r2;");      // взять младшие 8 битов второго статуса
  asm ("comp(r8,r9);");         // сравнить
  asm ("if eq jump DONE_TOGGLE_BIT;"); // если слово не изменилось все
     // продолжать парные чтения и сравнения
     // до истечения тайм-аута или окончания операции
asm("Check:");
  asm ("r7 = r7 + 1;");         // увеличить счетчик тайм-аута
  asm ("px = dm(0x800000);");   // прочитать первое слово
  asm ("r8 = px2;");            // сохранить
  asm ("px = dm(0x800000);");   // прочитать еще раз
  asm ("r9 = px2;");            // тоже сохранить
  asm ("r8 = r8 and r2;");      // взять младшие 8 битов первого статуса
  asm ("r9 = r9 and r2;");      // взять младшие 8 битов второго статуса
  asm ("comp(r8,r9);");         // сравнить
  asm ("if eq jump DONE_TOGGLE_BIT;"); // если слово не изменилось все
  asm ("comp (r5, r7);");       // не завершился ли тайм-аут?
  asm ("if ne jump Check;");    // если нет – еще раз попробовать
  ErrorCode = POLL_TIMEOUT;     // установить флаг ошибки
  ResetFlash();                 // сбросить память (перевести в режим чтения)
     // метка завершения проверки
asm ("DONE_TOGGLE_BIT:");
  return ErrorCode;
}
```

Приложение В

ОСНОВНЫЕ РЕГИСТРЫ УПРАВЛЕНИЯ И СТАТУСА КОДЕКА AD1881

Регистры, выделенные жирным шрифтом, устанавливаются DSP-процессором (контроллером) при инициализации кодека. Контроллер может записать новые значения в ряд и других регистров (см. столбец "Изменяются контроллером?"). Остальные регистры не могут быть изменены процессором.

Таблица С.1

Регистры кодека AD1881

Адрес	*Регистр*	*Используемое имя в программе (#define)*	*Значение*	*Изменяются контроллером?*
0x00	Reset	REGS_RESET	0x0400	нет
0x02	**Master Volume**	**MASTER_VOLUME**	**0x0000**	**да**
0x04	Зарезервировано	RESERVED_REG1	0xXXXX	нет
0x06	Master Volume Mono	MASTER_VOLUME_MONO	0x8000	да
0x08	Зарезервировано	RESERVED_REG2	0xXXXX	нет
0x0A	PC_Beep Mono	PC_BEEP_VOLUME	0x8000	да
0x0C	Phone Volume	PHONE_VOLUME	0x8008	да
0x0E	**Microphone Volume**	**MIC_VOLUME**	**0x8008**	**да**
0x10	**Line In Volume**	**LINE_IN_VOLUME**	**0x8808**	**да**
0x12	CD Volume	CD_VOLUME	0x8808	да
0x14	Video Volume	VIDEO_VOLUME	0x8808	да
0x16	Aux Volume	AUX_VOLUME	0x8808	да
0x18	PCM Out Volume	PCM_OUT_VOLUME	0x8808	да
0x1A	**Record Select**	**RECORD_SELECT**	**0x0404**	**да**
0x1C	**Record Gain**	**RECORD_GAIN**	**0x0F0F**	**да**
0x1E	Зарезервировано	RESEFRVED_REG3	0xXXXX	нет
0x20	**General Purpose**	**GENERAL_PURPOSE**	**0x8000**	**да**
0x22	3D Control	THREE_D_CONTROL_REG	0x0000	да
0x24	Зарезервировано	RESERVED_REG4	0xXXXX	нет
0x26	**Power-Down Control/Status**	**POWER_DOWN_CNTL_STAT**	**0x000X**	**нет**
0x28	Зарезервировано	RESERVED_REG5	0xXXXX	нет
0x74[6]	***Serial Configuration***	***SERIAL_CONFIGURATION***	***0xFF80***	***да***
0x76	***Miscellaneous Control Bits***	***MISC_CONTROL_BITS***	***0x0000***	***да***
0x78	***Sample Rate 0***	***SAMPLE_RATE_GENERATE_0***	***0xBB80***	***да***
0x7A	*Sample Rate 1*	*SAMPLE_RATE_GENERATE_1*	*0xBB80*	*да*
0x7C	*Vendor ID1*	*VENDOR_ID1*	*0x4144*	*нет*
0x7E	*Vendor ID2*	*VENDOR_ID2*	*0x5300*	*нет*

Как видно, для адресации регистра достаточно 7-битового поля. Жирным шрифтом выделены регистры, явную установку значений которых рекомендуется выполнять в программе.

[6] В соответствии со стандартом AC'97 регистры с адреса 0x70 используются производителями конкретной модели кодека.

C.1. Регистры управления громкостью воспроизведения

Регистр 02h (Master Volume) позволяет изменять громкость левого и правого каналов. Регистр 06h (Master Volume Mono) изменяет громкость на выходе MONO_OUT (если к этому выходу подключено внешнее устройство).

Адрес	Регистр	Биты																Значение по умолчанию
		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
0x02	Master Volume	Mute	X	ML5	ML4	ML3	ML2	ML1	ML0	X	X	MR5	MR4	MR3	MR2	MR1	MR0	0x8000
0x06	Master Volume Mono	Mute	X	X	X	X	X	X	X	X	X	MM5	MM4	MM3	MM2	MM1	MM0	0x8000

Биты ML5..ML0 соответствуют громкости (затуханию) левого канала, биты MR5..MR0 – громкости (затуханию) правого канала, биты MM5..MM0 – громкости (затуханию) на моно-выходе. В обоих регистрах младший разряд уровня соответствует изменению уровня громкости на 1,5 dB. Если старший бит (D15, Mute) в каком-либо из регистров установлен в "1", то звуковой сигнал не воспроизводится.

Таким образом, уровень "00000" соответствует максимальной громкости (затухание 0 dB), а уровень "111111" – минимальной громкости (максимальному затуханию – 94,5 dB).

C.2. Регистры усиления уровня входного аналогового сигнала

Усиление уровня входного сигнала осуществляется путем записи соответствующих значений в регистры, лежащие в диапазоне адресов 0x0C – 0x18 кодека.

Адрес	Регистр	Биты																Значение по умолчанию
		D15	D14	D13	D12	D11	D10	D 9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
0x0C	Phone Volume	Mute	X	X	X	X	X	X	X	X	X	X	GN4	GN3	GN2	GN1	GN0	0x8008
0x0E	Mic Volume	Mute	X	X	X	X	X	X	X	X	20dB	X	GN4	GN3	GN2	GN1	GN0	0x8008
0x10	Line In Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	0x8808
0x12	CD Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	0x8808
0x14	Video Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	0x8808
0x16	Aux In Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	0x8808
0x18	PCM Out Volume	Mute	X	X	GL4	GL3	GL2	GL1	GL0	X	X	X	GR4	GR3	GR2	GR1	GR0	0x8808

Путем изменения значения поля Gx4..Gx0 можно варьировать уровень сигнала (усиление/затухание) левого и правого каналов по каждому из аналоговых входов. Вес младшего разряда соответствует уровню примерно в 1,5 дБ. Влияние значения поля на уровень сигнала можно описать следующим образом:

Поле Mute	Поле Gx4..Gx0	Усиление / Затухание
0	00000	+12 дБ (усиление)
0	10000	0 дБ
0	11111	-34,5 дБ (затухание)
1	xxxxx	-∞ дБ (полное затухание)

C.2. Регистры выбора источника входного сигнала

Регистры, используемые для выбора источника сигнала и установки уровня усиления сигнала, расположены в диапазоне адресов кодека 0x1A-0x1E.

Адрес	Регистр	Биты																Значение по умолчанию
		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
0x1A	Record Select	X	X	X	X	X	SL2	SL1	SL0	X	X	X	X	X	SR2	SR1	SR0	0x0000
0x1C	Record Gain	Mute	X	X	X	GL3	GL2	GL1	GL0	X	X	X	X	GR3	GR2	GR1	GR0	0x8000
0x1E	Record Gain Mic	Mute	X	X	X	X	X	X	X	X	X	X	X	GM4	GM2	GM1	GM0	0x8000

Регистр 0x1A (Record Select) используется для выбора источника сигнала, причем выбор осуществляется независимо для правого и левого каналов. Поля SLx и SRx задают источник сигнала по следующей таблице соответствий:

Поле Sx3..Sx0	Источник для правого (R) канала (SR3..SR0)	Источник для левого (L) канала (SL3..SL0)
0	Mic	Mic
1	CD In (R)	CD In (L)
2	Video In (R)	Video In (L)
3	Aux In (R)	Aux In (L)
4	Line In (R)	Line In (L)
5	Stereo Mix (R)	Stereo Mix (L)
6	Mono Mix	Mono Mix
7	Phone	Phone

Регистры 0x1C и (при необходимости) 0x1E задают уровень усиления сигнала от выбранного источника. Значение "1111" в поле Gx3..Gx0 соответствует максимальному уровню усиления +22,5 dB, значение "0000" – отсутствию усиления. Бит № 15 (Mute) действует аналогично рассмотренным выше регистрам.

C.3. Регистр общего назначения

Регистр 0x20 (General Purpose) используется для управления некоторыми опциональными возможностями компонент кодека AC'97. Поддержка работы с кодеком посредством битов данного регистра зависит от конкретной модели кодека и требует обязательной проверки.

Адрес	Регистр	Биты																Значение по умолчанию
		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
0x20	General Purpose	POP	ST	3D	LD	DRSS1	DRSS0	MIX	MS	LPBK	X	X	X	X	X	X	X	0x0000

При работе с платой EZ-KIT Lite могут быть полезными следующие биты:

- "3D" – стерео в режиме имитации 3D (3D Enhancement);

- "LD" – режим усиления низких частот (Bass Boost);

- "LPBK" – режим зацикливания АЦП-ЦАП, когда данные с выхода АЦП передаются напрямую на ЦАП, без задействования цифрового интерфейса AC-link.

Назначение остальных битов данного регистра приведено в спецификации кодека AC'97.

C4. Регистр управления воспроизведением в режиме 3D Enhancement

Стандарт AC'97 позволяет воспроизводить стереозвук в режиме имитации трехмерного звучания (3D Enhancement). Если конкретная модель кодека поддерживает данный режим, то управление им осуществляется через регистр 0x22 (3D Control Register).

Адрес	Регистр	Биты																Значение по умолчанию
		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
0x22	3D Control	X	X	X	X	CR3	CR2	CR1	CR0	X	X	X	X	DP3	DP2	DP1	DP0	0x0000

Параметрам 3D-воспроизведения являются положение центра (CR3..CR0) и глубина (DP3..DP0) источника звука. Нулевое значение поля соответствует величине соответствующего параметра 0%, значение 15 – величине 100%.

Перед использованием данного регистра необходимо проверить, поддерживает данная модель кодека возможность варьирования параметров 3D-воспроизведения. Если при чтении регистра выяснилось, что его значение по умолчанию не равно нулю, то положение центра и

глубина источника звука – фиксированные величины и не могут быть изменены контроллером. Только если при включении питания кодека регистр имеет нулевое значение, то можно варьировать положение источника звука в режиме 3D Enhancement.

C.5. Регистр управления и контроля состояния подсистем кодека

Регистр 0x26 (Powerdown Control/Status) используется для мониторинга готовности к работе отдельных подсистем кодека.

Адрес	Регистр	Биты																Значение по умолчанию
		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
0x26	Powerdown Ctrl/Stat	EAPD	PR6	PR5	PR4	PR3	PR2	PR1	PR0	X	X	X	X	REF	ANL	DAC	ADC	na

Младшие 8 битов этого регистра – статусные биты (только для чтения), единичное значение каждого из которых подтверждает состояние готовности соответствующей подсистемы. При записи нового значения в этот регистр биты № 7..0 буду проигнорированы кодеком. После обнаружения контроллером единичного значения флага CodecReady в очередном кадре интерфейса AC-link, он может проверить состояние готовности некоторых подсистем:

- "PR0" – бит включения ("=0") и выключения ("=1") АЦП;
- "PR1" – бит включения ("=0") и выключения ("=1") ЦАП;
- "ANL" – микшер аналоговых сигналов;
- "DAC" – готовность ЦАП к приему данных;
- "ADC" – готовность АЦП к передаче данных.

Из старших битов для использования с платой EZ-KIT Lite могут представлять интерес биты PR0 (выключение АЦП) и PR1 (выключение ЦАП). Назначение остальных битов приведено в спецификации стандарта AC'97.

C.6. Регистр конфигурации кодека

Регистр 0x74 (Serial Configuration) используется для программирования и анализа состояния кодека AD1881.

Адрес	Регистр	Биты																Значение по умолчанию
		D15	D14	D13	D12	D11	D10	D9	D8	D 7	D 6	D 5	D 4	D 3	D2	D1	D0	
0x74	Serial Configuration	SLOT16	REGM2	REGM1	REGM0	DRQEN	DLRQ2	DLRQ1	DLRQ0	X	X	X	X	X	DRRQ2	DRRQ1	DRRQ0	X

Регистр 0x74 является специфическим для кодека AD1881 и играет ключевую роль при управлении кодеком. Он позволяет выполнять следующие основные функции:

- переводит кодек в режим Slot-16 или AC'97;

- выбирает активный кодек для управления или чтения статуса в том случае, если по одному интерфейсу AC-link подключены несколько микросхем;

- содержит биты запросов на передачу данных для ЦАП.

Эти функциональные возможности реализуются путем установки/сброса битов регистра:

- SLOT16 – управляющий бит переключения кодека в режим Slot-16 ("=1") или в стандартный режим AC'97 ("=0");

- REGMx – маски выбора кодека при записи или чтения значения в системный регистр кодека. Если используется только одна микросхема AD1881, то рекомендуется всегда записывать единицы во все биты REGMx;

- DRQEN – управляющий бит включения/выключения режима принудительной передачи контроллеру в каждом кадре в слотах № 1(Status Address Slot) и № 2 (Status Data Slot) статусной информации из регистра 0x74 (битов DxRQx) без специального запроса на чтение данного регистра от контроллера;

- DLRQx и DRRQx – статусные биты запросов от ЦАП на передачу в следующем кадре от контроллера кодеку обработанных ИКМ-отсчетов для вывода (воспроизведения) соответственно по левым и правым каналам кодеков (если подключен только один кодек, то следует рассматривать только биты DLRQ0 и DRRQ0).

С.7. Регистры управления частотой дискретизации аудиосигнала (sample rate)

Три регистра кодека управляют частотой дискретизации двух встроенных генераторов тактовой частоты. Регистр 0x76 содержит биты соответствия генератора тому или иному каналу АЦП и ЦАП, а регистры 0x78 и 0x7A содержат значения частот дискретизации отсчетов аудиосигналов для генератора № 0 и генератора № 1 в герцах.

Адрес	Регистр	Биты																Значение по умолчанию
		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
0x76	Miscellaneous Control Bits	X	X	X	X	X	DLSR	X	ALSR	X	X	X	X	X	DRSR	X	ARSR	X

Биты DxSR используются для выбора генератора для левого и правого каналов ЦАП, а биты AxSR – для левого и правого каналов АЦП. Если значение бита равно нулю, то в качестве генератора частоты дискретизации по соответствующему каналу будет использоваться генератор № 0 (регистр 0x78), а если единице – то генератор № 1 (регистр 0x7A). Остальные биты регистра используются для других целей.

Значения частот дискретизации записываются в регистры 0x78 и 0x7A в виде 16-разрядного беззнакового числа, лежащего в диапазоне от 7000 до 48000 Гц. Если записанное число выходит за границы диапазона, то работа генератора (и соответствующей подсистемы кодека) будет непредсказуемой.

Приложение С

ПРИМЕР ПРОГРАММНОГО КОДА ДЛЯ ИНИЦИАЛИЗАЦИИ КОДЕКА AD1881

```
#include    "def21160.h"
//-----------------------------------------------------------------------------
// Определения констант – номеров регистров, временных слотов и масок
//-----------------------------------------------------------------------------
      // Номера временных слотов в режиме TDM
#define           TAG_PHASE               0
#define           COMMAND_ADDRESS_SLOT    1
#define           COMMAND_DATA_SLOT       2
#define           STATUS_ADDRESS_SLOT     1
#define           STATUS_DATA_SLOT        2
#define           LEFT                    3
#define           RIGHT                   4
      // Биты ADC Valid Bits и маски DAC Request Bits
#define           M_Left_ADC              12
#define           M_Right_ADC             11
#define           DAC_Req_Left            0x80
#define           DAC_Req_Right           0x40
      // Адреса системных регистров кодека AD1881
#define           REGS_RESET              0x0000
#define           MASTER_VOLUME           0x0200
#define           RESERVED_REG_1          0x0400
#define           MASTER_VOLUME_MONO      0x0600
#define           RESERVED_REG_2          0x0800
#define           PC_BEEP_Volume          0x0A00
#define           PHONE_Volume            0x0C00
#define           MIC_Volume              0x0E00
#define           LINE_IN_Volume          0x1000
#define           CD_Volume               0x1200
#define           VIDEO_Volume            0x1400
#define           AUX_Volume              0x1600
#define           PCM_OUT_Volume          0x1800
#define           RECORD_SELECT           0x1A00
#define           RECORD_GAIN             0x1C00
#define           RESERVED_REG_3          0x1E00
#define           GENERAL_PURPOSE         0x2000
#define           THREE_D_CONTROL_REG     0x2200
#define           RESERVED_REG_4          0x2400
#define           POWERDOWN_CTRL_STAT     0x2600
#define           SERIAL_CONFIGURATION    0x7400
#define           MISC_CONTROL_BITS       0x7600
#define           SAMPLE_RATE_GENERATE_0  0x7800
#define           SAMPLE_RATE_GENERATE_1  0x7A00
```

```
#define           VENDOR_ID_1             0x7C00
#define           VENDOR_ID_2             0x7E00
      // число системных регистров для инициализации в программе
#define           cnNumRegisters          17
      // размер буфера передачи и число активных слотов в режиме TDM
#define           cnTX_Size               16
      // Доступные функции
.GLOBAL           Clear_All_SPT0_Regs;
.GLOBAL           Program_SPORT0_Registers;
.GLOBAL           Program_DMA_Controller;
.GLOBAL           AD1881_Codec_Initialization;
.GLOBAL           Codec_Reset;
      // Глобальные переменные
.GLOBAL           tx_buf;
.GLOBAL           rx_buf;
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
.segment /dm    seg_dmda;

.var rx_buf[5];                     // приемный буфер

.var tx_buf[cnTX_Size] =            // передающий буфер
      0xE000,                       // флаг Valid Frame и слоты № 1 и 2
      0x7400,                       // регистр Serial Configuration
      0x0F80,                       // режим Slot-16, REGM2..0 = 1111, DRQEN = 1
      0x0000,                       // остальные слоты обнулить
      0x0000,
      0x0000,
      0x0000,
      0x0000,
      0x0000,
      0x0000,
      0x0000,
      0x0000,
      0x0000,
      0x0000,
      0x0000,
      0x0000,
      0x0000;

      // TCB_блоки для DMA-пересылок на прием и передачу кадров
      // с установленным битом PCI для CPx регистра (цепочечная DMA)
.var rcv_tcb[8]  = 0, 0, 0, 0, rcv_tcb+7+0x40000, 5, 1, rx_buf;             // TCB-блок для приема
.var xmit_tcb[8] = 0, 0, 0, 0, xmit_tcb+7+0x40000, cnTX_Size, 1, tx_buf;    // TCB-блок для передачи
```

```
      // номера и значения системных регистров кодека AD1881 для инициализации
.var Init_Codec_Registers[cnNumRegisters*2] =
      MASTER_VOLUME,          0x0000,     // режим Mute выключен
      MASTER_VOLUME_MONO,     0x8000,     // Mono Volume отключен (Mute)
      PC_BEEP_Volume,         0x8000,     // PC Volume отключен
      PHONE_Volume,           0x8008,     // Phone Volume отключен
      MIC_Volume,             0x8048,     // усиление 20 дБ, выход с MIC IN отключен
      LINE_IN_Volume,         0x8808,     // усиления нет, выход с LINE IN откл.
      CD_Volume,              0x8808,     // CD Volume отключен (Mute)
      VIDEO_Volume,           0x8808,     // Video Volume отключен (Mute)
      AUX_Volume,             0x8808,     // AUX Volume отключен (Mute)
      PCM_OUT_Volume,         0x0808,     / усиление от ЦАП 12,5 дБ по каждому каналу
      RECORD_SELECT,          0x0404,     // Входной сигнал LINE IN для обоих каналов
      RECORD_GAIN,            0x0000,     // Нет усиления входа для обоих каналов
      GENERAL_PURPOSE,        0x0000,     // нет никаких 3D и т.п. эффектов
      THREE_D_CONTROL_REG,    0x0000,     // нет параметров 3D стерео
      MISC_CONTROL_BITS,      0x0000,     // генератор № 0 для АЦП и ЦАП
      SAMPLE_RATE_GENERATE_0, 8000,       // Генератор № 0 - частота дискретизации
      SAMPLE_RATE_GENERATE_1, 48000;      // Генератор № 1 - частота дискретизации

      // массив для размещения номеров и значений регистров при чтении
      // (для проверки инициализации)
.var Codec_Init_Results[34] =
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0,
      0,          0;

.endseg;
//------------------------------------------------------------------------

//------------------------------------------------------------------------
.SEGMENT /pm      seg_pmco;
```

```
//-------------------------------------------------------------------------
// Очистка регистров (для совместимости с программой-монитором)
//-------------------------------------------------------------------------
Clear_All_SPT0_Regs:
      IRPTL = 0x00000000;     // снять все имеющиеся запросы на прерывания
      bit clr imask SPT0I;    // запретить прерывания по передачи (TX0)

      r0 = 0x00000000;
      dm(SRCTL0) = r0;
      dm(RDIV0) = r0;
      dm(STCTL0) = r0;
      dm(MRCS0) = r0;
      dm(MTCS0) = r0;
      dm(MRCCS0) = r0;
      dm(MTCCS0) = r0;

            // Установка параметров DMA-каналов, связанных со SPORT0
            // в значения "по умолчанию после сброса"
      r1 = 0x1FFFF;     dm(II0) = r1;
      r1 = 0x0001;      dm(IM0) = r1;
      r1 = 0xFFFF;      dm(C0)  = r1;
      r1 = 0x00000;     dm(CP0) = r1;
      r1 = 0x1FFFF;     dm(GP0) = r1;
      r1 = 0x1FFFF;     dm(II2) = r1;
      r1 = 0x0001;      dm(IM2) = r1;
      r1 = 0xFFFF;      dm(C2)  = r1;
      r1 = 0x00000;     dm(CP2) = r1;
      r1 = 0x1FFFF;     dm(GP2) = r1;
      rts;
Clear_All_SPT0_Regs.end:
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------
// Программирование регистров управления SPORT0
//-------------------------------------------------------------------------
Program_SPORT0_Registers:
            // Регистр управления приемом через SPORT0:
            // TDM-16 каналов, RFS-внутр., RCLK-внешн., Длина слова – 16 битов,
            // DMA-вкл., Цепочечная DMA-вкл.
      r0 = 0x1F0C40F0;
      dm(SRCTL0) = r0;

            // Делитель кадровой частоты (12.288 МГц / 48 КГц - 1 = 255)
      r0 = 0x00FF0000;
      dm(RDIV0) = r0;
```

```
            // Регистр управления передачей через SPORT0:
            // MFD – 1 бит, Длина слова – 16 битов, DMA-вкл., Цепочечная DMA-вкл.
            // многоканальный режим пока не включен!
      r0 = 0x001C00F0;
      dm(STCTL0) = r0;

            // определить активные каналы передачи и приема в режиме TDM
      r0 = 0x0000001F;        // для приема – каналы 0-4
      dm(MRCS0) = r0;
      r0 = 0x0000FFFF;        // для передачи – каналы 0-15
      dm(MTCS0) = r0;

            // отключить компандирование при приеме и передаче
      r0 = 0x00000000;
      dm(MRCCS0) = r0;
      dm(MTCCS0) = r0;

      rts;
Program_SPORT0_Registers.end:
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Программирование регистров параметров DMA контроллера
// для передачи и приема данных через SPORT0
//-----------------------------------------------------------------------------
Program_DMA_Controller:
      // инициировать загрузку TCB_блока и начало DMA-пересылки на передачу кадра
      r0 = dm(xmit_tcb + 4);
      dm(CP2) = r0;           // CP2 <- xmit_tcb+7 (PCI=1)

//-----------------------------------------------------------------------
// После начала DMA-пересылки сразу пересылается 2 слова из tx_buf,
// поскольку двухуровневый буферный регистр TX0 свободен
//-----------------------------------------------------------------------
      // инициировать загрузку TCB_блока и начало DMA-пересылки на прием кадра
      r0 = dm(rcv_tcb + 4);
      dm(CP0) = r0;           // CP0 <- rcv_tcb+7 (PCI=1)

      rts;
Program_DMA_Controller.end:
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Выключение и включение кодека AD1881 с помощью линии FLAG3
//-----------------------------------------------------------------------------
```

```
Codec_Reset:
      bit clr FLAGS FLG3;     //выключить кодек
      r0 = 0x40000;           // ждать
      lcntr = r0, do Reset_End UNTIL LCE;
Reset_End:  nop;

      bit set FLAGS FLG3;     // включить кодек
      rts;
Codec_Reset.end:
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
// Инициализировать регистры кодека AD1881
//------------------------------------------------------------------------------
AD1881_Codec_Initialization:
            // разрешить прерывание по передаче кадра
            // (с пустым обработчиком)
      bit set imask SPT0I;

Enable_SPORT0_Transfers:
            // перевести SPORT0 в многоканальный режим (TDM)
            // бит многоканального режима в SRCTL0 отвечает и за прием, и за передачу
      r0 = dm(SRCTL0);
      r0 = bset r0 by 23;
      dm(SRCTL0) = r0;

Wait_Codec_Ready:             // ждать готовности кодека
      r0 = dm(rx_buf + 0);    // для этого проверять бит № 15 в слоте № 0
      r2 = 0x8000;
      r0 = r0 and r2;
      if eq jump Wait_Codec_Ready;  // если Codec Ready=0, то снова ждать
      idle;                   // ждать еще два кадра (отсчет по переданным кадрам)
      idle;

            // кодек готов к работе. Инициализировать регистры управления
Initialize_1881_Registers:
      i4 = Init_Codec_Registers;    // указатель на адреса регистров и значения
      r12 = 0xE000;                // Флаг Valid Frame = 1, слоты № 1 и 2 активны

      lcntr = cnNumRegisters, do Codec_Init until lce;
            dm(tx_buf + TAG_PHASE) = r12;       // временной слот № 0 одинаковый
            r1 = dm(i4, 1);                     // взять адрес регистра из массива
            dm(tx_buf + COMMAND_ADDRESS_SLOT) = r1; // записать во временной слот № 1
            r1 = dm(i4, 1);                     // взять значение регистра из массива
            dm(tx_buf + COMMAND_DATA_SLOT) = r1;  // записать во временной слот № 2
Codec_Init: idle;                               // ждать один кадр
```

```
      // Для поддержки частоты дискретизации, отличающейся от 48 КГЦ,
      // целесообразно выключить и включить АЦП и ЦАП чтобы они работали синхронно
PowerDown_DACs_ADCs:
      idle;                         // синхронизировать передачу и прием кадров
            // выключение АЦП и ЦАП
      r12 = 0xE000;                 // Флаг Valid Frame = 1, слоты № 1 и 2 активны
      dm(tx_buf + TAG_PHASE) = r12;
      r0 = POWERDOWN_CTRL_STAT;     // регистр выключения/включения подсистем
      dm(tx_buf + COMMAND_ADDRESS_SLOT) = r0;
      r0 = 0x0300;                  // выключить АЦП и ЦАП
      dm(tx_buf + COMMAND_DATA_SLOT) = r0;
      idle;                         // ждать два кадра
      idle;
            // цикл ожидания
      lcntr = 80, do reset_loop1 until lce;
reset_loop1: nop;

      idle;                         // синхронизировать передачу и прием кадров
            // включение АЦП и ЦАП
      r12 = 0xE000;                 // Флаг Valid Frame = 1, слоты № 1 и 2 активны
      dm(tx_buf + TAG_PHASE) = r12;
      r0 = POWERDOWN_CTRL_STAT;     // регистр выключения/включения подсистем
      dm(tx_buf + COMMAND_ADDRESS_SLOT) = r0;
      r0 = 0x0000;                  // включить АЦП и ЦАП
      dm(tx_buf + COMMAND_DATA_SLOT) = r0;
      idle;                         // ждать два кадра
      idle;
            // цикл ожидания для перехода АЦП и ЦАП в установившийся режим
      lcntr = 80000, do reset_loop2 until lce;
reset_loop2: nop;

      bit clr imask SPT0I;          // запретить прерывания по передаче кадра от TX0
      rts;
AD1881_Codec_Initialization.end:
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// Главная функция программы. Вызывается из обработчика прерывания сброса
//-------------------------------------------------------------------------
_main:
      IRPTL = 0x00000000;           // сбросить все запросы на обработку прерываний
      bit set MODE1 IRPTEN;         // глобально разрешить прерывания

      call Clear_All_SPT0_Regs;
      call Program_SPORT0_Registers;
```

```
      call Program_DMA_Controller;
      call Codec_Reset;
      call AD1881_Codec_Initialization;

      bit set IMASK SPR0I;          // Разрешить прерывание по приему кадра от RX0
                                    // для обработки аудиоданных
.endseg;
//-------------------------------------------------------------------------
```

Приложение D

ПРИМЕР ГЕНЕРАЦИИ АУДИОСИГНАЛОВ РАЗЛИЧНЫХ ЧАСТОТ С ИСПОЛЬЗОВАНИЕМ ПЛАТЫ EZ-KIT LITE

```
//-------------------------------------------------------------------------
// Программный код выполняет генерацию звукового стереосигнала заданной
// частоты (тона). Форма синусоиды задается в виде коэффициентов массива
// для фиксированной частоты дискретизации (8000 Гц).
// Поддерживаются несколько тонов: 100 Гц (80 амплитуд для полного периода)
//                                 200 Гц (40 амплитуд)
//                                 400 Гц (20 амплитуд)
//                                 800 Гц (10 амплитуд)
// Переключение к тону большей частоты осуществляется при нажатии на кнопку SW3
// (прерывание IRQ0). Переключение к тону меньшей частоты – при нажатии на
// кнопку SW2 (прерывание IRQ1). Переключение тонов является циклическим.
//
// Входные данные (ИКМ-отсчеты) от кодека не обрабатываются. Моменты выдачи
// ИКМ-отсчетов синусоиды текущего тона определяются путем анализа битов
// DAC Request Bits. Предполагается, что левый и правый канал ЦАП работают
// синхронно и на одной частоте дискретизации.
//
// Приведены фрагменты таблицы векторов прерываний, подпрограмм обработки
// прерывания по приему кадра и обработчики прерываний IRQ0 и IRQ1.
//-------------------------------------------------------------------------


//-------------------------------------------------------------------------
// Таблица векторов прерываний
//-------------------------------------------------------------------------
.segment /pm   seg_rth;
RSTI_scv:   jump _main;
...
IRQ1I_svc: jump Process_IRQ1; rti; rti; rti;
IRQ0I_svc: jump Process_IRQ0; rti; rti; rti;
...
SPR0I_svc: jump Process_AD1881_Audio_Samples; rti; rti; rti;
...
SPT0I_svc:  rti;  rti;  rti; rti;
...
.ENDSEG;
//-------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------
.segment /dm    seg_dmda;

.var        Left_Channel_Out;       // выходные ИКМ-отсчеты для вывода в кодек
.var        Right_Channel_Out;

.var        DAC_request_bits;       // флаги запросов от ЦАП

.var        ToneChange = 0;         // Флаг смены тона

      // адреса и размеры массивов синусоид
.var        Tone_Tags[8] =    Sinusoid100, 80,
                              Sinusoid200, 40,
                              Sinusoid400, 20,
                              Sinusoid800, 10;

.var Sinusoid100[80] =  "sin100.dat";     // синусоида 100 Гц
.var Sinusoid200[40] =  "sin200.dat";     // синусоида 200 Гц
.var Sinusoid100[20] =  "sin400.dat";     // синусоида 400 Гц
.var Sinusoid100[10] =  "sin800.dat";     // синусоида 800 Гц
.endseg;
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------
.segment /pm seg_pmco;
//-------------------------------------------------------------------------------
// Перед началом работы создаем два буфера:
// B0 – буфер начальных адресов и длин синусоид
// B1 – буфер текущей синусоиды
//-------------------------------------------------------------------------------
_main:
      ...
            // Буфер Tone_Tags. Модификатор равен 2.
      B0 = Tone_Tags;
      I0 = Tone_Tags + 2;           // начальная синусоида – 200 Гц
      L0 = 8;
      M0 = 2;
            // Буфер текущего тона
      r0 = dm(0, I0);               // адрес начала синусоиды
      B1 = r0;
      r0 = dm(1, I0);               // длина синусоиды
      L1 = r0;
      M1 = 1;
      ...
_main.end:
```

```
//-------------------------------------------------------------------------------
// Обработчик прерывания IRQ0.
// Переходим к синусоиде с большей частотой и устанавливаем флаг изменения тона
//-------------------------------------------------------------------------------
Process_IRQ0:
      modify(I0,2);
      r8 = dm(ToneChange);
      r8 = 1;
      dm(ToneChange) = r8;
      rti;
Process_IRQ0.end:
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------
// Обработчик прерывания IRQ1.
// Переходим к синусоиде с меньшей частотой и устанавливаем флаг изменения тона
//-------------------------------------------------------------------------------
Process_IRQ1:
      modify(I0,-2);
      r8 = dm(ToneChange);
      r8 = 1;
      dm(ToneChange) = r8;
      rti;
Process_IRQ1.end:
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------
// Обработчик прерывания SPR0I по приему кадра.
// Анализируем наличие запроса ЦАП на ИКМ-отсчеты и формируем выходной кадр
//-------------------------------------------------------------------------------
Process_AD1881_Audio_Samples:
            // формировать кадр
      r0 = 0x8000;              // Флаг Valid Frame = 1
      dm(tx_buf + TAG_PHASE) = r0;
      r0 = 0;                   // остальные слоты обнулить (с 5-го по 15-й всегда = 0)
      dm(tx_buf + COMMAND_ADDRESS_SLOT) = r0;
      dm(tx_buf + COMMAND_DATA_SLOT) = r0;
      dm(tx_buf + LEFT) = r0;
      dm(tx_buf + RIGHT) = r0;
check_DAC_request_bits:
      r1 = dm(rx_buf+1);        // взять Status Address Slot
      r0 = 0x00C0;              // маска для выделения позиций № 7 и 6
      r2 = r1 AND r0;           // выделить DAC Request Bits (DLRQ0 и DRRQ0)
      dm(DAC_request_bits) = r2;    // сохранить DAC Request Bits
set_tx_slot_valid_bits:
      r2 = r2 XOR r0;           // инвертировать DAC Request Bits (активное сост. = 1)
```

```
      r2 = LSHIFT r2 BY 5;    // на места битов корректности слотов № 3 и 4
      r0 = 0x8000;            // установить флаг Valid Frame для передачи
      r2 = r2 or r0;          // изменить тэг для временного слота # 0
      dm(tx_buf + TAG_PHASE) = r2;  // положить в буфер для передачи
do_audio_processing:
      r2 = dm(DAC_request_bits);    // прочитать DAC Request Bits
      BTST r2 BY 7;                 // проверка DAC Request Bit для левого канала
      IF not sz JUMP playback_audio_data; // если =1, то не записывать левый канал
            // доходим сюда только если надо выводить данные
      r0 = dm(ToneChange);
      r0 = pass r0;
      if eq jump get_audio_data;    // если тон не изменялся
            // Буфер текущего тона
      r0 = dm(0, I0);               // адрес начала синусоиды
      B1 = r0;
      r0 = dm(1, I0);               // длина синусоиды
      L1 = r0;
      r0 = 0;
      dm(ToneChange) = r0;          // тон изменился флаг сбросить
get_audio_data:                     // берем следующее значение для того же тона
      r0 = dm(I1,M1);
      r2 = r0;
      DM(Left_Channel_Out) = r0;    // записать для левого канала
      DM(Right_Channel_Out) = r2;    // записать для правого канала
playback_audio_data:
      r2 = dm(DAC_request_bits);    // прочитать DAC Request Bits
put_DAC_left:
      BTST r2 BY 7;                 // проверка DAC Request Bit для левого канала
      IF not sz JUMP put_DAC_right; // если =1, то не записывать левый канал
      r15 = dm(Left_Channel_Out);   // взять результат обработки
      dm(tx_buf + LEFT) = r15;      // записать в слот № 3
put_DAC_right:
      BTST r2 BY 11;                //проверка DAC Request Bit для правого канала
      IF not sz JUMP tx_done;       // если =1, то не записывать правый канал
    r15 = dm(Right_Channel_Out);    // взять результат обработки
    dm(tx_buf + RIGHT) = r15;       // записать в слот № 4
tx_done:
      rti;
Process_AD1881_Audio_Samples.end:
.endseg;
//---------------------------------------------------------------------------
```