L-CARD

Устройства сбора данных

L-761, L-780 и L-783

Платы АЦП/ЦАП/ТТЛ на шину **PCI 2.1**

Библиотека *PLX_API*. DOS

Руководство программиста.

Москва. Октябрь 2007 г.
Rev. C

ЗАО «Л-КАРД»,
117105, г. Москва, Варшавское шоссе, д. 5, корп. 4, стр. 2.

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

Адреса в Интернет:

WWW: *www.lcard.ru*
FTP: *ftp.lcard.ru*

E-Mail:

Общие вопросы: *lcard@lcard.ru*
Отдел продаж: *sale@lcard.ru*
Техническая поддержка: *support@lcard.ru*
Отдел кадров: *job@lcard.ru*

Представители в регионах:

Украина:	“ХОЛИТ Дэйта Системс, Лтд”	*www.holit.com.ua*	(044) 241-6754
Санкт-Петербург:	ЗАО “АВТЭКС Санкт-Петербург”	*www.autex.spb.ru*	(812) 567-7202
Новосибирск:	ООО “Сектор Т”	*www.sector-t.ru*	(3832) 22-76-20
Екатеринбург:	Группа Компаний АСК	*www.ask.ru*	(343) 371-44-44
Казань:	ООО “Шатл”	*shuttle@kai.ru*	(8432) 38-16-00
Самара:	"АСУ-Самара"	*prosoft-s@jiguli.ru*	(846)-998-29-01

L-761, **L-780** и **L-783**. Платы АЦП/ЦАП/ТТЛ общего назначения на шину **PCI 2.1**.

Оглавление

1. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПОД DOS

1.1. Работа в среде DOS

Данный раздел описания предназначен для программистов, собирающихся писать свои собственные программы в среде DOS для работы с платами серии *L-7xx*, а именно: *L-761*, *L-780* и *L-783*. В качестве базового языка при написании штатной библиотеки подпрограмм (набор *API*-функций) нами был выбран язык *C++*, поскольку он является одним из самых широко распространенных и применяемых языков. Для приобретенных Вами плат *ЗАО «Л-Кард»* поставляет готовую штатную библиотеку подпрограмм, в которую мы попытались включить множество разнообразных функций для облегчения пользователю процедуры написания собственных программ по управлению платами серии *L-7xx*. Данная библиотека позволяет Вам использовать практически все возможности плат, не вдаваясь в тонкости их низкоуровневого программирования. Если Вы все же собираетесь сами программировать платы на низком уровне, то наша библиотека может быть использована Вами в качестве законченного и отлаженного примера, на основе которого Вы можете реализовать свои собственные алгоритмы.

Штатная библиотека содержит функции, позволяющие осуществлять ввод-вывод аналоговой и цифровой информации в асинхронном режиме, вводить и выводить аналоговую информацию, как в одноканальном, так и в многоканальном режимах с произвольной синхронизацией ввода, вводить и выводить данные в программном режиме, в режиме генерации прерываний, осуществлять конфигурацию FIFO буферов для АЦП и ЦАП и т.д. Мы надеемся, что описываемая ниже библиотека упростит и ускорит написание Ваших программ.

Весь пакет штатного программного обеспечения для работы с платами серии *L-7xx* в среде DOS находится на прилагаемом к плате CD-ROM в директории `PCI\L-7XX`. **!!!ВНИМАНИЕ!!!** Везде далее по тексту данного описания все директории указаны относительно неё.

1.2. Подготовка к работе

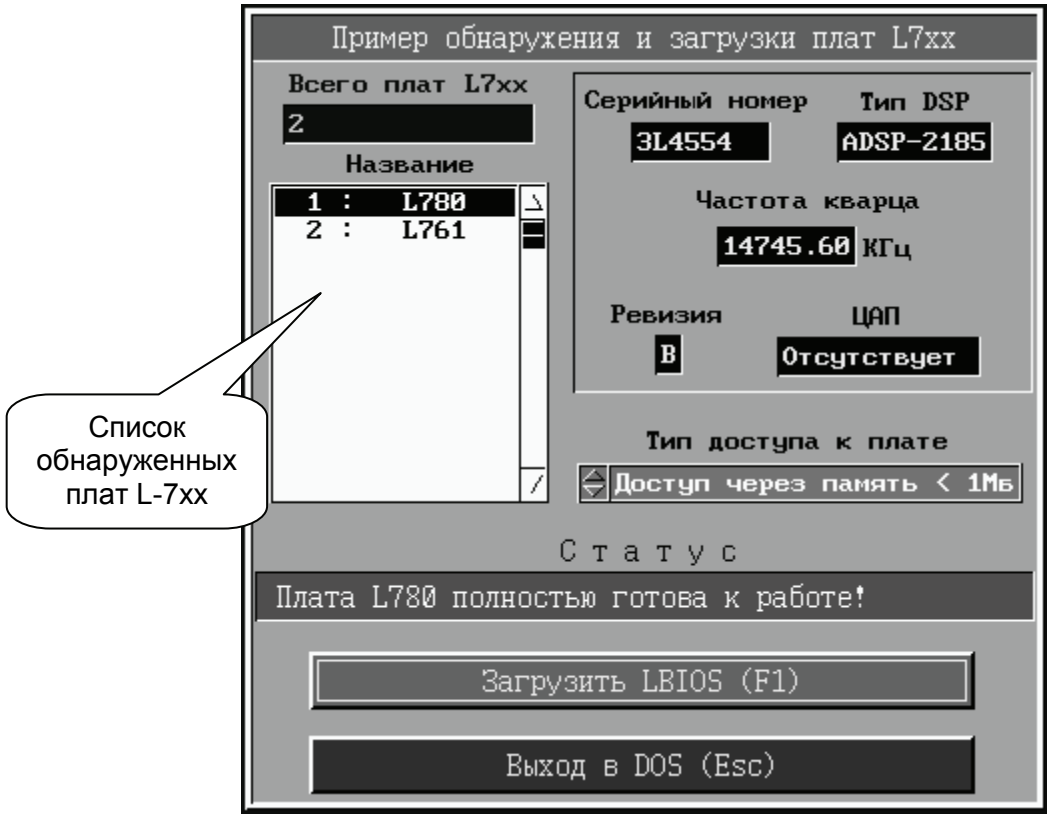
1.2.1. Обнаружение и загрузка плат серии L-7xx

Предположим, что Вы уже установили плату в компьютере и подали на вход платы сигналы. При работе с ней необходимо учитывать, что платы серии *L-7xx* имеют две характерные особенности, отличающие их от простых плат ввода-вывода:

- Платы данной серии разработаны с полным отсутствием каких-то ни было конфигурационных перемычек и переключателей специально для работы с высоко производительной шиной **PCI**. **PCI**-интерфейс обеспечивают микросхемы *PCI9050-1* или *PCI9030* фирмы *PLX Technology, Inc.* (более подробную информацию о ней можно найти на сайте *www.plxtech.com*). В связи с тем, что платы работают с шиной **PCI**, для надлежащей работы и осуществления доступа к плате необходимо выяснить какие базовые адреса, прерывания и т.д. заданы для каждой конкретной платы (в нашем случае эти назначения должны делаться ***BIOS*** компьютера на этапе его загрузки). В штатной библиотеке подпрограмм (набор *API*-функций для плат серии *L-7xx*) для этого имеется соответствующая функция ***INIT_ACCESS_TO_PLX()***, которая аккуратно выполняет процедуры такого рода.
- На платах *L-7xx* установлены современные цифровые сигнальные процессоры *ADSP-2184*/*ADSP-2185*/*ADSP-2185M*/*ADSP-2186* от фирмы *Analog Devices, Inc.* (более подробную информацию можно найти на сайте *www.analog.com*). Перед работой с платой сигнальные процессоры (иначе DSP) необходимо предварительно запрограммировать, т. е. загрузить в него управляющую программу (драйвер, *LBIOS*). В состав штатного программного обеспечения входят законченные управляющие программы для каждого типа плат серии *L-7xx*. Соответствующий драйвер для каждого конкретного типа платы состоит из одного бинарного файла вида `L-7xx.BIO` (т.е. `L761.BIO` для платы *L-761*, `L780.BIO` для платы *L-780* и `L783.BIO` для платы *L-783*), содержащий как выполняемый код управляющей про-

граммы, так и сегмент данных для сигнального процессора. В штатной библиотеке подпрограмм (набор *API*-функций) для загрузки *LBIOS* имеется специальная функция ***LOAD_LBIOS_PLX()***, которая аккуратно выполняет процедуру загрузки *LBIOS*. Для загрузки всех плат, вставленных в Ваш компьютер, можно просто вызвать утилиту загрузки `LOADBIOS\LOADBIOS.EXE`. Программа сама определит, сколько плат *L-7xx* имеется в наличие и попытается **все** их аккуратно загрузить, выдавая соответствующие сообщения на экран. Программа также имеет возможность отключать вывод всех сообщений на экран дисплея с помощью флага –s(creen) (например, для вызова из других программ): `LOADBIOS.EXE -s`. Если же Вы желаете выборочно загружать платы, то можно использовать флаг –m(anual): LOADBIOS.EXE –m. Этот флаг отменяет действие флага –s. Загружать управляющую программу в плату необходимо, как правило, всего один раз после включения питания компьютера (при выключении питания вся память программ сигнального процессора обнуляется). Только **ПОСЛЕ** загрузки *LBIOS* Вы можете полностью управлять платой, т.е. переводить ее в разные режимы ввода-вывода и т. п.

В качестве примера для обнаружения плат серии *L-7xx* (определения базовых адресов доступа, номеров прерываний и т.д.) и проверки процедуры загрузки LBIOS в сигнальный процессор можно просто воспользоваться программой тестирования процедуры загрузки `\Examples\LOAD_PLX\LOAD_PLX.EXE`. При этом в случае обнаружения хотя бы одной платы серии *L-7xx* откроется интуитивно-понятная диалоговая панель, которая представлена на рисунке ниже:



При нажатии клавиши *Загрузить LBIOS (F1)* и в случае успешной загрузки *LBIOS* программа выдаст, например, такое сообщение в статусной строке "Плата L-780 полностью готова к работе!". Иначе выдаст краткое диагностическое толкование ошибки процесса загрузки платы.

Исходный текст программы `LOAD_PLX.EXE` на языке *C++* находится в файле `\Examples\LOAD_PLX.DOS\LOAD_PLX.CPP`. Эти текст можно использовать как пример применения некоторых функций из штатной библиотеки подпрограмм для плат серии *L-7xx*.

1.2.1.1. Библиотека подпрограмм для работы с платами L-7xx

Штатная библиотека подпрограмм для плат серии *L-7xx* написана с использованием широко распространенного языка программирования **Borland C++ 3.1.** Исходный текст данной библиотеки Вы можете найти в файле `LIBRARY\PLX_API.CPP`, откомпилированный вариант в файле `LIBRARY\PLX_API.OBJ`, а заголовочный файл в `LIBRARY\PLX_API.H`. Законченные примеры вызова функций штатной библиотеки и работы с платами *L-7xx* можно найти в директории `\Examples`.

Для вызова функций из языка *C++* Вам необходимо следующее:

- создать файл проектов для языка *C++* (например `test.prj`);
- добавить в него файл `PLX_API.OBJ;`
- создать и добавить в проект Ваш файл с будущей программой (например, `test.cpp`);
- включить в начало вашего файла заголовочный файл `#include "plx_api.h"`, содержащий описания всех доступных функций библиотеки (он должен находится в текущей директории, в которой Вы создали Ваш файл проектов);
- в общем-то, **ВСЁ!** Теперь Вы можете писать свою программу и в любом месте вызывать функции из библиотеки `PLX_API.OBJ`.

Если Вы пользуетесь средой программирования **Borland C++3.1**, то у Вас есть возможность включить в Ваш файл проектов не откомпилированную версию штатной библиотеки, а ее исходный текст `PLX_API.CPP`. При этом Вам необходимо в меню Option\Transfer описать ТурбоАссемблер, добавив строку TASM в ‘Program Path’, /MX /ZI $TASM в ‘Command Line’ и пометить меню ‘Translator’. Это необходимо, т.к. в исходном тексте библиотеки используются ассемблерные вставки.

1.2.2. Используемые термины и форматы данных

1.2.2.1. Термины

Название	Смысл
ADC_Rate	Частота работы АЦП в кГц
Inter_Kadr_Delay	Межкадровая задержка в млс
DAC_Rate	Частота работы ЦАП в кГц
Buffer	Указатель на целочисленный массив для данных
N_Points	Число отсчетов ввода
Channel	Номер канала
Control_Table	Управляющая таблица, содержащая целочисленный массив с логическими номерами каналов для последовательного циклического ввода данных с АЦП
Control_Table_Length	Длина управляющей таблицы
Address	Адрес ячейки в памяти программ или данных DSP

1.2.2.2. Форматы данных

1.2.2.2.1. Формат слова данных с АЦП

Данные, считанные с АЦП, представляются в формате знакового целого двухбайтного числа:
от -8192 до 8191 для 14-ти битных плат *L-761* и *L-780*;
от -2048 до 2047 для 12-ти битных плат *L-783*.

Коды АЦП, соответствующие установленному входному диапазону, приведены в следующей таблице:

Таблица 1. Соответствие кода АЦП напряжению на аналоговом входе

Плата	Усиление	Код	Напряжение, В	Точность, %
L-761, L-780 Ревизия 'B' и 'C'	1; 4; 16; 64	+8000	+MAX	2÷3
		0	0	0.15; 0.2; 0.3; 0.6
		-8000	-MAX	2÷3
L-780 Ревизия 'A'	1; 4; 16; 64	+8191	+MAX	2÷3
		0	0	0.15; 0.2; 0.3; 0.6
		-8192	-MAX	2÷3
L-783	1; 2; 4; 8	+2000	+MAX	2÷3
		0	0	0.15; 0.2; 0.3; 0.6
		-2000	-MAX	2÷3

где MAX - значение установленного диапазона для аналогового канала АЦП (возможные диапазоны для каждого конкретного типа плат см. в **Таблице 5**).

Все выше указанные значения приведены для случая, когда *LBIOS* не корректирует получаемые с АЦП данные с помощью калибровочных коэффициентов, хранящихся в ППЗУ (см. *§ 1.2.2.3 "Формат пользовательского ППЗУ"*). Если же *LBIOS* производит такую корректировку, тогда соответствующие коды АЦП приведены ниже:

Таблица 2. Соответствие кода АЦП напряжению на аналоговом входе при разрешенной корректировке входнных данных

Плата	Усиление	Код	Напряжение, В	Точность, %
L-761, L-780 Ревизия 'B' и 'C'	1; 4; 16; 64	+8000	+MAX	0.1; 0.12; 0.15; 0.2
		0	0	
		-8000	-MAX	
L-780 Ревизия 'A'	1; 4; 16; 64	+8100	+0.975*MAX	
		0	0	
		-8100	-0.975*MAX	

L-783	1; 2; 4; 8	+2000	+MAX	0.1; 0.12; 0.15; 0.2
		0	0	
		-2000	-MAX	

где MAX - значение установленного диапазона для аналогового канала АЦП (возможные диапазоны для каждого конкретного типа плат см. в **Таблице 6**).

1.2.2.2.2. Формат слова данных для ЦАП

Формат $16^{\text{ти}}$ битного слова данных, передаваемого в плату для последующей выдачи на ЦАП, приведен в следующей таблице:

Таблица 3. Формат слова данных ЦАП

Плата	Номер бита	Назначение
L-7xx	0÷11	$12^{\text{ти}}$ битный код ЦАП
	12	Выбор номера канала ЦАП: ✓ ‘0’ – первый канал; ✓ ‘1’ – второй канал.
	13÷15	Не используются

Собственно код, выдаваемый платой на $12^{\text{ти}}$ битный ЦАП, связан с устанавливаемым на внешнем разъёме напряжением в соответствии со следующей таблицей

Таблица 4. Соответствие кода ЦАП’ов напряжению на внешнем разъёме

Плата	Код	Напряжение
L-761, L-780 L-783	+2047	+5.0 Вольт
	0	0 Вольт
	-2048	-5.0 Вольт

1.2.2.2.3. Логический номер канала АЦП

На платах серии *L-7xx* для управления работой АЦП при сборе данных с входных аналоговых каскадов определяется такой параметр, как 8-ми битный логический номер канала АЦП (фактически управляющее слово для АЦП). Именно массив логических номеров каналов АЦП, образующих управляющую таблицу **Control_Table**, задает циклическую последовательность работы АЦП при вводе данных. В состав логического номера канала АЦП входят несколько важных параметров, задающих различные режимы функционирования платы:

- физический номер аналогового канала;
- управление включением режима калибровки нуля, т.е. при этом вход каскада с программируемым коэффициентом усиления (PGA) просто заземляется;
- тип подключения входных каскадов – 16 дифференциальных входных аналоговых каналов или 32 входных канала с общей землёй;
- коэффициент усиления, т.е. для каждого канала можно установить свой индивидуальный коэффициент усиления.

Таблица 5. Формат логического номера канала.

Номер бита	Обозначение	Функциональное назначение
0	**MA0**	0ой бит номера канала
1	**MA1**	1ый бит номера канала
2	**MA2**	2ой бит номера канала
3	**MA3**	3ий бит номера канала
4	**MA4**	Калибровка нуля/4ый бит номера канала
5	**MA5**	16 диф./32 общ.
6	**GS0**	0ой бит коэффициента усиления
7	**GS1**	1ый бит коэффициента усиления

Если **MA5**=0 и **MA4**=0, то **MA0**÷**MA3** – номер выбранной дифференциальной пары входов.
Если **MA5**=0 и **MA4**=1, то калибровка нуля, т.е. измерение собственного напряжения смещения нуля.
Если **MA5**=1, то **MA0**÷**MA4** – номер выбранного входа с общей землей (X1->Вход1, X2-> Вход2, **…**, Y1-> Вход17,**…**, Y16-> Вход32).

Например, логический номер для платы *L-780* равный 0x2 означает дифференциальный режим работы 3его канала с единичным усилением, 0x82 – с усилением равным 16. Если же этот логический номер равен 0x10 или 0x14, то вход каскада PGA просто заземлен (именно PGA, а не входы указанных каналов коммутатора).

Таблица 6. Коэффициент усиления (биты GS0 и GS1)

Плата	Бит GS1	Бит GS0	Усиление	Диапазон, В
L-761, L-780	0	0	1	5.0
	0	1	4	1.25
	1	0	16	0.3125
	1	1	64	0.078
L-783	0	0	1	5.0
	0	1	2	2.5
	1	0	4	1.25
	1	1	8	0.625

Например, при выбранном коэффициенте усиления 16 для платы *L-780* диапазон входного напряжения будет 0.3125 В, а при усилении 8 для платы *L-783* диапазон будет 0.625 В.

1.2.2.3. Формат пользовательского ППЗУ

На платах серии *L-7xx* установлено пользовательское ППЗУ емкостью 64 слова×16 бит. Формат данного ППЗУ представлен на следующем рисунке:

Как видно из рисунка, в первых 20[ти] словах (40 байт) последовательно хранится служебная информация. Порядок расположения в ППЗУ данной информации соответствует структуре *PLATA_DESCR* (см. *§ 1.2.5.1.2 "Структура PLATA_DESCR"*). Для чтения этой информации можно использовать специальную API-функцию ***GET_PLATA_DESCR_PLX()***. Порядок расположения служебной информации имеет последовательный вид:

- ✓ серийный номер платы (9 байт);
- ✓ название платы (5 байт);
- ✓ ревизия платы (1 байт);
- ✓ тип установленного на плате DSP (5 байт);
- ✓ частота установленного на плате кварца в Гц (4 байта);
- ✓ флажок присутствия ЦАП на плате (2 байта);
- ✓ зарезервировано (14 байт);

В следующих 8 словах (16 байт) хранятся коэффициенты, используемые для корректировки данных, получаемых с АЦП. Данные коэффициенты записываются в ППЗУ при наладке плат в *ЗАО «Л-Кард»*. Благодаря этому на платах серии *L-7xx* отсутствуют подстроечные резисторы, что сильно улучшает шумовые характеристики плат и увеличивает их надежность. Формат калибровочных коэффициентов предназначен специально для работы с *LBIOS*. Загрузка этих коэффициентов в память данных DSP осуществляется с помощью API-функции ***LOAD_COEF_PLX()***, а разрешение *LBIOS* корректировать входные данные АЦП – с помощью ***ENABLE_CORRECTION_PLX()***. Коэффициенты хранятся в виде чисел типа **int** языка *C++* (2 байта) и имеют следующий порядок:

- ✓ 20 ячейка - корректировка смещения нуля усиления ‘1’;
- ✓ 21 ячейка - корректировка смещения нуля усиления ‘4’ (*“L761”* и *“L780”*) или ‘2’ (*“L783”)*;
- ✓ 22 ячейка - корректировка смещения нуля усиления ‘16’ (*“L761”* и *“L780”*) или ‘4’ (*“L783”)*;
- ✓ 23 ячейка - корректировка смещения нуля усиления ‘64’(*“L761”* и *“L780”*) или ‘8’ (*“L783”)*;
- ✓ 24 ячейка - корректировка масштаба усиления ‘1’;
- ✓ 25 ячейка - корректировка масштаба усиления ‘4’ (*“L761”* и *“L780”*) или ‘2’ (*“L783”)*;
- ✓ 26 ячейка - корректировка масштаба усиления ‘16’ (*“L761”* и *“L780”*) или ‘4’ (*“L783”)*;
- ✓ 27 ячейка - корректировка масштаба усиления ‘64’ (*“L761”* и *“L780”*) или ‘8’ (*“L783”)*;

В ячейках 28÷31 (8 байт) хранятся коэффициенты, используемые для корректировки кода, выводимого на ЦАП’ы. Данные коэффициенты записываются в ППЗУ при наладке плат в *ЗАО «Л-Кард»*. Преобразование кода, выдаваемого на ЦАП, производится следующим образом:

$$RealDacValue=(DacValue+Offset/10000.)*Scale/10000.,$$

где *RealDacValue* – реальный код, выдаваемый на ЦАП; *DacValue* – код, который желательно установить на выходе ЦАП; *Offset* – значение корректировки нуля, которое хранится в ППЗУ; *Scale* – значение корректировки масштаба, которое также хранится в ППЗУ.

Например, для установки на ЦАПе нулевого выходного напряжения надо вывести код:

$$(0.+Offset/10000.)*Scale/10000.$$

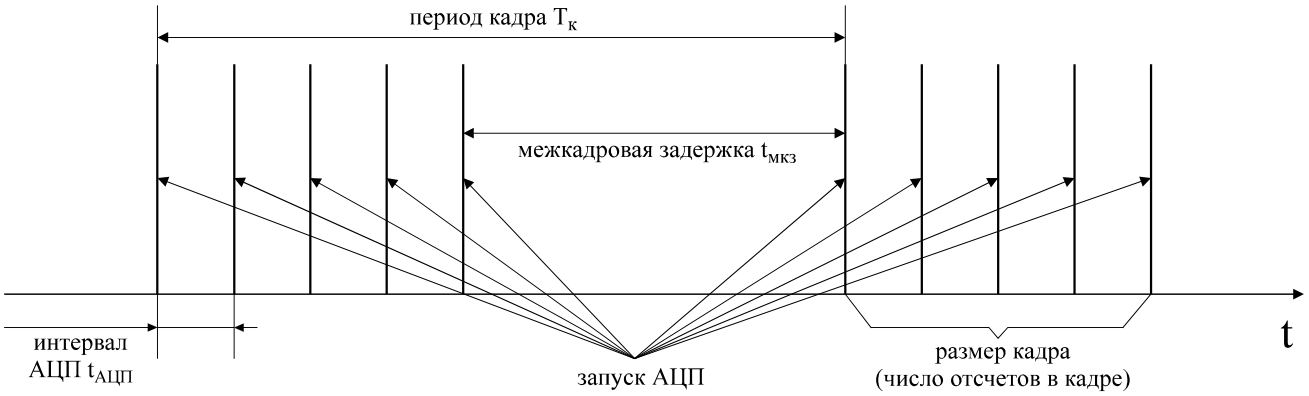
Коэффициенты хранятся в виде чисел типа **int** языка *C++* (2 байта) и имеют следующий порядок:

- ✓ 28 ячейка - корректировка смещения нуля первого ЦАП'а;
- ✓ 29 ячейка - корректировка смещения нуля второго ЦАП'а;
- ✓ 30 ячейка - корректировка масштаба первого ЦАП'а;
- ✓ 31 ячейка - корректировка масштаба второго ЦАП'а;

В пользовательскую область ППЗУ, начиная с 32ой ячейки, Вы можете записывать и считывать любую свою информацию с помощью соответствующих API-функций ***WRITE_FLASH_WORD_PLX()*** и ***READ_FLASH_WORD_PLX()***.

1.2.2.4. Формат кадра отсчетов

Кадр отсчетов представляет себой последовательность отсчетов с логических каналов от **Control_Table[0]** до **Control_Table[Control_Table_Length-1]**, где **Control_Table** - управляющая таблица (массив логических каналов), хранящейся в памяти данных DSP, **Control_Table_Length** определяет размер (длину) этой таблицы. Загрузить таблицу в сигнальный процессор можно с помощью API функции ***LOAD_CONTROL_TABLE_PLX()*** (см. *§ 1.2.5.4.2 "Загрузка управляющей таблицы"*). Временные параметры кадра для **Control_Table_Length=5** приведены на следующем рисунке:



где $\mathbf{T_k}$ – временной интервал между соседними кадрами (фактически частота опроса фиксированного логического номера канала), $\mathbf{t_{мкз}}$=**Inter_Kadr_Delay** – временной интервал между последним отсчетом текущего кадра и первым отсчетом следующего, $\mathbf{t_{АЦП}}$ – интервал запуска АЦП или межканальная задержка. Тогда $1/\mathbf{t_{АЦП}}$=**ADC_Rate** – частота работы АЦП или оцифровки данных, а величина $\mathbf{t_{мкз}}$ не может принимать значения меньшие, чем $\mathbf{t_{АЦП}}$. Если размер кадра, т.е. число отсчетов АЦП в кадре, равен **Control_Table_Length**, то все эти временные параметры можно связать следующей формулой:

$$\mathbf{T_k} = (\mathbf{Control\_Table\_Length}-1) * \mathbf{t_{АЦП}} + \mathbf{t_{мкз}},$$

или

$$\mathbf{T_k} = (\mathbf{Control\_Table\_Length}-1) / \mathbf{ADC\_Rate} + \mathbf{Inter\_Kadr\_Delay}.$$

Временные параметры **ADC_Rate** и **Inter_Kadr_Delay** используются в API функции ***SET_KADR_TIMING_PLX()*** (см. *§ 1.2.5.4.3 "Установка временных параметров работы АЦП"*) для задания необходимого режима работы АЦП и *LBIOS*.

1.2.3. Модели памяти

Все функции библиотеки предполагают использование дальних указателей типа FAR, как для вызова процедур, так и для передачи адресов данных, поэтому компиляцию программы в языке C++ желательно проводить в модели памяти LARGE, в которой по умолчанию используются дальние вызовы процедур и дальние указатели на данные.

1.2.4. Общий подход к работе с платами серии L-7xx

На текущий момент времени существуют следующие модификации плат серии *L-7xx*:

1. платы *L-761 (Rev. B)*;
2. платы *L-780 (Rev. A), L-780 (Rev. B)* и *L-780M (Rev. C)*;
3. платы *L-783 (Rev. B)* и *L-783M (Rev. C)*.

1.2.4.1. Общие особенности работы плат

В последнее время стали появляться мощные компьютеры, оборудованные шиной **PCI** версии 2.3. В принципе было заявлено, что эта спецификация **PCI** является обратно совместимой со старой шиной **PCI** 2.2. Но при этом наши платы серии *L-7xx* иногда начинают некорректно работать с такими компьютерами. Так при их совместном использовании компьютеры вообще могут не запускаться. Один из возможных вариантов решения этой проблемы заключается в необходимости найти компьютер, в котором плата серии *L-7xx* нормально работает, и воспользоваться ДОСовской утилитой с нашего CD-ROM: `\PCI\L7XX\Utils\CHIOMEM\CHIOMEM.EXE`. Также последнюю версию данной утилиты можно скачать с нашего сайти *www.lcard.ru* из раздела *"Библиотека файлов"*. Там из подраздела *"PCI платы серии L-7XX. DOS"* следует выбрать самораспаковывающийся архив *chiomem_util.exe*.

Если данная утилита обнаружит необходимость модификации конфигурационного PCI ППЗУ платы, то она сделает доступной клавишу '*Изменить PCI ППЗУ (F3)*'. Тогда следует обязательно воспользоваться данной опцией для надлежащего исправления содержимого PCI ППЗУ. После успешного выполнения этой процедуры плата должна корректно работать со всеми компьютерами, оснащенными шиной **PCI** версии 2.3.

1.2.4.2. Особенности работы плат "L-780" через порты

Платы *L-780* могут быть трех разновидностей: ревизия 'A', ревизия 'B' и ревизия 'C' (поле **BoardRevision** в структуре *BOARD_INFO* см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*). Их различие состоит в том, что платы ревизии 'A' требуют для своей работы под DOS большее количество ресурсов в виде портов и, в связи с этим, на некоторых материнских платах не функционируют должным образом. Для исключения этого эффекта использование портов на плате, т.е. доступ к плате через порты ввода-вывода, можно полностью запретить с помощью утилиты `\Utils\CHIOMEM\CHIOMEM.EXE` (подробности см. в *Приложении B*). **По умолчанию** платы ревизии 'A' выпускаются с запрещенным доступом через порты I/O. Значение базового порта I/O для доступа к плате после выполнения API-функции ***INIT_ACCESS_TO_PLX()*** находится в поле **IO_BaseAddress** структуры *BOARD_INFO* (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*).

1.2.4.3. Особенности работы плат L-7xx через память ниже 1 Мб

Все платы серии *L-7xx* дают пользователю возможность работы с ними как через порты ввода-вывода (традиционный подход), так и через память, которая расположена либо в диапазоне от 640 Кб до 1 Мб, либо где-нибудь выше 1 Мб. Под работой через память подразумевается работа с платой, когда все ее управляющие регистры "отображены" на обычную память PC и доступ к ней возможен с помощью обычных ассемблерных инструкций передачи данных, например, таких как `MOV` (см. *§ 2.1.1.2 "Интерфейс через память"*). При работе через память скорость обмена платы с компьютером достигает 10 Мб/с. Базовый адрес доступа через память ниже 1 Мб находится в поле **LowMemorySpaceBaseAddress** структуры *BOARD_INFO* (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*). Подробности организации низкоуровневого взаимодействия с платой можно найти в *§ 2.1 "Общий PCI-интерфейс работы с платами серии L-7xx"*. Если Вы желаете работать с платами через память ниже 1 Мб, Вам необходимо предотвратить употребление диапазона памяти выделенного для плат от использования каким то ни было администратором памяти (memory manager), таких как EMM386 или QEMM. Так при работе с EMM386 в файле `config.sys` строчка с `EMM386.EXE` должна содержать параметр `X=mmmm-nnnn`. Например, эта строчка мо-

жет быть такой: DEVICE=C:\WINDOWS\EMM386.EXE X=C800-CA00. Диапазон памяти, выделяемый каждой плате серии *L-7xx*, можно узнать с помощью утилиты `\Utils\PARAMS\PARAMS.EXE` (подробности см. *приложение C*). Если работа через память ниже 1 Мб Вам в принципе не нужна, то можно полностью запретить этот доступ, используя утилиту `\Utils\CHIOMEM\CHIOMEM.EXE` (подробности см. в *приложении B*).

К сожалению, во многих материнских платах (Iwill, Zida и т.д.) доступ к PCI платам через память ниже 1 Мб не работает вообще. Более того, возможно зависание компьютера при его загрузке в зависимости от взаимного расположения платы видеоадаптера и Вашей платы. Все платы серии *L-7xx ЗАО «Л-Кард»* по умолчанию сконфигурированы в режим работы через порты ввода-вывода и через память выше 1 Мб (в 4 Гб адресном пространстве). Программно можно включить еще один тип доступа - через адресное пространство ниже 1 Мб для упрощения работы с платой под DOS. При этом возможны три варианта развития событий:

- Компьютер грузится, и плата нормально работает в режиме доступа через память ниже 1 Мб (например, этот режим гарантированно работает на всех материнских платах ASUS).
- Компьютер грузится, но плата не работает в режиме доступа через память ниже 1 Мб. Увы, Вам не повезло, желательно выключить режим доступа через нижнюю память или попробовать заменить системную плату.
- Компьютер не грузится. Увы, Вам совсем не повезло, но не все еще потеряно. Следует поменять местами плату АЦП и видеокарту, после чего загрузка PC пройдет успешно, но плата через нижнюю память работать не будет. Желательно выключить режим доступа через нижнюю память или попробовать заменить системную плату.

Чтобы не быть голословными, ниже приведен ответ одного из импортных консультантов по PCI платам: "Some new PCs does not allow to use the memory below 1 MB. This space is reserved for some specific cards (like video card). If your application is a video card (for example), you should be able to run the PCI9050 in the lower memory < 1MB. Therefore, you can not configure the PCI 9050 using below 1MB memory address space". В заключение можно добавить, что подобных проблем при работе через верхнюю память и через порты ввода-вывода не возникает.

1.2.4.4. Особенности работы плат L-7xx через память выше 1 Мб

Платы серии *L-7xx* дают пользователю возможность работы с ними как через порты ввода-вывода (традиционный подход), так и через память, которая расположена либо в диапазоне от 640 Кб до 1 Мб, либо где-нибудь выше 1 Мб. Под работой через память подразумевается работа с платой, когда все ее управляющие регистры “отображены” на память PC и доступ к ней возможен с помощью обычных ассемблерных инструкций передачи данных, таких как `MOV` (см. *§ 2.1.1.2 "Интерфейс через память"*). При работе через память скорость обмена платы с компьютером достигает 10 Мб/с. Базовый адрес доступа через память выше 1 Мб находится в поле **HighMemorySpaceBaseAddress** структуры *BOARD_INFO* (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*). На данный момент времени функции штатной библиотеки (под DOS) работают надлежащим образом с платами серии *L-7xx* через память выше 1 Мб **только если** Ваша прикладная программа функционирует в “чистом” DOS. Это означает, что Вы должны находиться в реальном режиме работы центрального процессора PC, т.е. не использовать ни какой администратор памяти типа EMM386 или QEMM.

1.2.4.5. Особенности плат Rev. C

Платы *Rev. C* представляют собой продукт основательной модернизации наших старых плат *Rev. B*. С точки зрения конечного пользователя новые платы унаследовала практически без изменений все основные функциональные характеристики старых плат. С другой стороны, данная модификация плат привнесла пользователю, кроме всего прочего, два весьма полезных улучшения, а именно:

1. Возможность организовывать работу потокового вывода на ЦАП, используя при этом дополнительно введённое прерывание от платы в PC;

2. Способность чисто программным образом управлять доступом выходных цифровых линий (т.е. перевод их ‘*третье*’ состояние и обратно). Для того, чтобы иметь возможность воспользоваться указанным режимом, необходимо замкнуть перемычкой контакты 1–2 дополнительно появившегося на плате разъёма **X5** (см. *"Руководство пользователя", § 2.2. "Внешний вид плат серии L-7xx"*). При этом непосредственно после подачи питания на плату выходные линии находятся в ‘*третьем*’ состоянии. Если же у этого разъёма перемычкой замкнуты контакты 2–3, то в части работы выходных цифровых линий плата получается полностью идентичной платам *Rev. A* и *B*. В этом режиме после подачи питания на плату выходные линии находятся в неопределённом состоянии.

1.2.4.6. Общий подход к работе с API функциями

Платы серии *L-7xx* являются устройствами, предназначенными для работы с высоко производительной шиной **PCI**. Стандарт шины PCI предусматривает назначение каждой плате таких параметров как базовые адреса доступа (через порты ввода-вывода и/или память), номер прерываний и т.д. В основном, назначения подобного рода производит либо *BIOS* компьютера, либо WINDOWS. Поэтому на таких платах, как правило, не бывает конфигурационных перемычек и переключателей. Вся назначенная каждой **PCI**-плате информация хранится в так называемом конфигурационном пространстве PCI, прочитать которое можно с помощью надлежащих программных процедур (подробности можно найти, например, в “PCI Local Bus Specification. Revision 2.1”, § 3.7.4.1 “Configuration Mechanism #1”). В штатной библиотеке для этих целей предназначена функция ***READ_PCI_REG_PLX()***.

Другой особенностью плат данной серии является то, что на них установлен мощный цифровой сигнальный процессор (DSP – Digital Signal Processor). Для того, чтобы его “оживить” во внутреннюю память DSP надо записать (загрузить) фирменную управляющую программу, которая входит в комплект штатный поставки (файл с расширением `.BIO`), или свою собственную. Со стороны DSP весь обмен данными с PC осуществляется по так называемому каналу IDMA (Internal Direct Memory Access). Подробнее о работе канала IDMA можно прочитать, например, в оригинальной книге *“ADSP-2100 Family User’s Manual (Includes ADSP-2171, ADSP-2181)”,* Chapter 11 “*DMA Ports*”, *§ 11.3 “IDMA Port”, стр. 11-12, Analog Devices, Inc., Third Edition September 1995* или на сайте www.analog.com. Задачей DSP является управление всей периферией (АЦП, ЦАП, цифровые линии и т.д.) и обработка получаемый данных. Во внутренней памяти DSP расположены FIFO буфера АЦП и ЦАП, а также переменные *LBIOS* (см. *приложение А*). О низкоуровневом взаимодействии DSP с периферией см. *Раздел 2 "НИЗКОУРОВНЕВОЕ ОПИСАНИЕ ПЛАТ СЕРИИ L-7xx"*

Перед началом работы со штатной библиотекой в Вашей программе необходимо сделать объявления следующих переменных:

```
extern int PLX_Board_Quantity;          // определена в штатной библиотеке
                                        // глобальная переменная, содержащая после выполнения
                                        // функции INIT_ACCESS_TO_PLX() кол-во обнаруженных
                                        // плат серии L-7xx
struct BOARD_INFO bi[4];    // массив структур типа BOARD_INFO, описанных в
                            // штатной библиотеке, см. § 1.2.5.1.1 "Структура BOARD_INFO"
```

После того, как необходимые переменные объявлены, Вам следует просканировать все PCI пространство в поисках плат серии *L-7xx*. Эту процедуру можно осуществить с помощью соответствующей API функции ***INIT_ACCESS_TO_PLX(struct BOARD_INFO *bi)***, которая аккуратно заполняет все поля массива структур типа ***BOARD_INFO*** для каждой обнаруженной платы, учитывая при этом служебную информацию, хранящуюся в пользовательском ППЗУ (см. *§ 1.2.2.3 "Формат пользовательского ППЗУ"*). По окончании данной функции в переменной *PLX_Board_Quantity* будет содержаться количество обнаруженный плат серии *L-7xx*.

Теперь, если обнаружена хотя бы одна плата, можно задать режим доступа функций штатной библиотеки к плате: через порты ввода-вывода, через память ниже 1 Мб или через память выше

1 Мб. Процедуру такого рода выполняет функция ***SET_ACCESS_MODE_PLX()*** (подробнее см. *§ 1.2.5.2.3 "Установка режима доступа к плате"*).

Далее необходимо загрузить в сигнальный процессор платы управляющую программу (*LBIOS*). Для этого можно воспользоваться функцией ***LOAD_LBIOS_PLX***(). В случае безошибочного выполнения данной функции, можно проверить работоспособность загруженного *LBIOS* с помощью функции ***PLATA_TEST_PLX()***. Если и эта функция выполнена без ошибки, то это означает, что *LBIOS* успешно загружен и плата полностью готова к работе.

На следующем этапе следует настроить *LBIOS* на работу с тем типом DSP, который у Вас установлен на плате и прописан в ППЗУ (*§ 1.2.2.3 "Формат пользовательского ППЗУ"*). Функция *SET_DSP_TYPE_PLX()* как раз и предназначена для этой цели. Если функция не вернула ошибку, то это означает, что *LBIOS* благополучно настроился на нужный тип DSP.

В общем-то, **ВСЕ!** Теперь можно спокойно управлять всей доступной периферией на плате и самим DSP с помощью соответствующих API функций штатной библиотеки подпрограмм, т.е. задавать различные режимы работы АЦП (программный опрос FIFO буфера АЦП или по прерываниям, конфигурация FIFO буфера АЦП, синхронизация ввода данных с АЦП, частота оцифровки данных и т.д.) и ЦАП (конфигурация FIFO буфера ЦАП, частота выдачи данных на ЦАП и т.д.), обрабатывать входные и выходные цифровые линии, считывать и/или записывать необходимую информацию в/из пользовательского ППЗУ и т.д.

Перед выходом в DOS **необходимо** завершить работу с платами, вызвав функцию ***CLOSE_ACCESS_TO_PLX()*** (см. *§ 1.2.5.2.8 "Функция завершения работы с платам серии L-7xx"*).

В качестве примера приведем исходный текст программы для обнаружения и загрузки **одной** платы серии *L-7xx*:

```
#include <stdlib.h>
#include <stdio.h>
#include "plx_api.h"            // заголовочный файл штатной библиотеки

extern int PLX_Board_Quantity;  // кол-во обнаруженных плат серии L-7xx
struct BOARD_INFO bi[3];        // массив из 3х структур BOARD_INFO

int main(void)
{
    int AccessMode;

    // просканируем все PCI пространство в поисках плат серии L-7xx
    INIT_ACCESS_TO_PLX(bi);
    if(!PLX_Board_Quantity)
    {
      printf("Ни одной платы серии L-7xx не обнаружено!\n");
      CLOSE_ACCESS_TO_PLX(); return 1; //выйдем из программы с ошибкой
    }
    // разрешен ли доступ к первой из обнаруженных плат?
    if(bi[0].BoardAccessMode == NO_ACCESS_MODE)
    {
      printf("Доступ к плате %s (серийный номер %s) ЗАПРЕЩЕН!!!\n",
                           bi[0].Board_Name, bi[0].BoardSerialNumber);
      CLOSE_ACCESS_TO_PLX(); return 1; //выйдем из программы с ошибкой
    }
    else
    {
       // установим режим доступа к плате через память ниже 1 Мб
       AccessMode = LOW_MEM_ACCESS;
```

```
        SET_ACCESS_MODE_PLX(&bi[0], &AccessMode);
        printf("Установлен режим доступа через %s для платы %s\
               (серийный номер %s)!", AccessMode ? "память" : "порты",
              bi[0].Board_Name, bi[0].BoardSerialNumber);
    }
    // теперь попробуем загрузить LBIOS в первую из обнаруженных плат
    if(!LOAD_LBIOS_PLX(&bi[0]))
    {
      printf("Не выполнена функция LOAD_LBIOS_PLX()!");
      CLOSE_ACCESS_TO_PLX(); return 1; //выйдем из программы с ошибкой
    }

    if(!PLATA_TEST_PLX(&bi[0]))
    {
      printf("Не выполнена функция PLATA_TEST_PLX()!");
      CLOSE_ACCESS_TO_PLX(); return 1; //выйдем из программы с ошибкой
    }

    // попробуем настроить LBIOS на нужный тип DSP
    if(!SET_DSP_TYPE_PLX(&bi[0]))
    {
      printf("Не выполнена функция SET_DSP_TYPE_PLX ()!");
      CLOSE_ACCESS_TO_PLX(); return 1; //выйдем из программы с ошибкой
    }

    printf("Плата %s (серийный номер %s) полностью готова к\
                работе!", bi[0].Board_Name, bi[0].BoardSerialNumber);

    // далее можно располагать функции для непосредственного
    // управления платой!
    . . . . . .

    // завершим работу с платами
    CLOSE_ACCESS_TO_PLX();

    // выйдем в DOS
    return 0;
}
```

1.2.4.7. Общая структура LBIOS

На платах серии *L-7xx* устанавливется так называемый цифровой сигнальный процессор (Digital Signal Processor – DSP) фирмы **Analog Devices, Inc.** ADSP-2184/2185/2186. Основное его назначение – это управление различного рода периферийными устройствами, установленными на плате, а также, возможно, обработка данных. Одно из главных преимуществ применения именно цифрового сигнального процессора заключается в том, что можно достаточно гибко менять в широких пределах алгоритмы его работы с периферийными устройствами чисто программным образом (достаточно лишь овладеть очень **не сложным** языком ассемблера DSP). Так в штатном драйвере *LBIOS* реализуются наиболее широко используемые алгоритмы работы с АЦП, ЦАП, входными/выходными ТТЛ линиями и т.д.

Для программиста нужно знать, что DSP обладает двумя независимыми типами памяти:

- ✓ 24 битная память программ (Program Memory – PM), в которой хранятся коды инструкций управляющей программы (драйвера *LBIOS*), а также, возможно, данные;
- ✓ 16 битная память данных (Data Memory – DM), в которой могут находиться только данные.

Карты распределения памяти обоих типов для различных сигнальных процессоров, а также взаимное расположение составных частей *LBIOS*, подробно показаны в *приложении А*. Как видно из указанного приложения, *LBIOS* состоит из:

- ✓ области в PM с исполняемыми кодами инструкций *LBIOS*;
- ✓ области в DM с переменными *LBIOS*;
- ✓ двумя областями под FIFO буфера АЦП и ЦАП.

Исполняемый код *LBIOS* написан с учетом возможности взаимодейтсвия драйвера с Вашей программой в PC по так называемому принципу команд. Это означает следующее:

- ✓ в ячейку данных памяти DSP **L_COMMAND** (см. ниже) необходимо положить номер выполняемой команды с помощью функции ***PUT_DM_WORD_PLX()***;
- ✓ инициировать в DSP прерывания *IRQ2* (см. *§ 2.1. "Общий PCI-интерфейс работы с платами серии L-7xx"*);
- ✓ обработчик указанного прерывания,входящий в состав *LBIOS*, выполнит все необходимые для данной команды действия;
- ✓ по окончании выполнения данной команды в ячейке **L_COMMAND** должно находиться число 0x0.

Адреса переменных в DM, задающих важные параметры функционирования *LBIOS*, а также их краткие толкования, приведены в **Таблице 7.**

В *LBIOS* организованно два циклических буфера: для приема данных с АЦП и для выдачи данных на ЦАП. Штатная библиотека работает с ними как с программно организованными FIFO буферами. Так, например, данные из буфера АЦП можно получать двумя способами: программно (см. *§ 1.2.5.4.5. "Получение массива данных с АЦП"*) и по прерываниям (см. *§ 1.2.5.6. "Функции для работы с прерываниями"*).

1.2.5. Описание библиотеки API функций

В настоящем разделе приведены достаточно подробные описания переменных, структур и функций, входящих в состав штатной досовской библиотеки для плат серии *L-7xx*. Предлагаемое программное обеспечение разрабатывалось с учетом идеологического родства плат данной серии и, следовательно, все штатные функции выполняются однотипно для различных плат. Однако существуют отличительные нюансы определенного характера при использовании штатных функций для плат типа *L-783* на частотах работы АЦП выше 1 МГц. Подробнее эти отличия выявлены в *§ 1.2.6 "Особенности работы штатной библиотеки с платами L-783"*.

1.2.5.1. Переменные и структуры

1.2.5.1.1. Структура BOARD_INFO

Структура *BOARD_INFO* описана в файле `plx_api.h` и представлена ниже:

```
struct BOARD_INFO
{
    char Board_Name[8];       // название платы (макс. 7 символов):
                              // "L761", "L780", "L783" или "Unknown"
    char BoardSerialNumber[9]; // серийный номер платы (8 символов)
    unsigned BoardDspType;    // тип установленного на плате DSP:
                              // 0 или ADSP2184_PLX для ADSP-2184,
                              // 1 или ADSP2185_PLX для ADSP-2185,
                              // 2 или ADSP2186_PLX для ADSP-2186,
                              // где ADSP2184_PLX, ADSP2185_PLX и
                              // ADSP2186_PLX – предопределенные константы.
    double BoardQuartzFrequency;  // частота установленного на плате кварца в кГц:
                                  // 14745.6,
                                  // 16667.0,
                                  // 20000.0.
    char BoardRevision;           // тип ревизии платы: 'A', 'B' или 'C'
    unsigned int ConfigRegsBaseAddress;  // базовый порт доступа к локальным
                                         // регистрам PCI9050-1 или PCI9030
    unsigned int IO_BaseAddress;  // базовый порт (I/O) доступа к плате L-7xx
    unsigned long HighMemorySpaceBaseAddress; // базовый адрес памяти выше
                                              // 1 Мб для доступа к платам L-7xx
    unsigned long LowMemorySpaceBaseAddress;  // базовый адрес памяти ниже
                                              // 1 Мб для доступа к платам L-7xx
    unsigned int IDMA_Word_Access;       // используется в функциях доступа к плате
    unsigned int IDMA_Array_Access;      // используется в функциях доступа к плате
    unsigned int IDMA_Address_Access;    // используется в функциях доступа к плате
    unsigned int DSP_Irq2_Access;        // используется в функциях доступа к плате
    unsigned int DSP_Reset_Access;       // используется в функциях доступа к плате
    unsigned int BoardAccessMode;  // текущий режим доступа к плате под DOS:
                // 0 или IO_ACCESS – через порты ввода-вывода I/O,
                // 1 или LOW_MEM_ACCESS – через память, расположенную ниже 1Мб,
                // 2 или HIGH_MEM_ACCESS – через память, расположенную выше 1Мб,
                // 3 или NO_ACCESS_MODE – доступ к плате вообще запрещен,
                // где IO_ACCESS, LOW_MEM_ACCESS, HIGH_MEM_ACCESS и
                // NO_ACCESS_MODE – предопределенные константы.
    unsigned int IsDacPresented;     // наличие ЦАП'а на плате:
```

```
                                        // 0 – ЦАП отсутствует,
                                        // 1 - ЦАП присутствует,
    unsigned InterruptNumber;   // номер прерывания, назначенный плате
    int IrqVectorNumber;        // вектор назначенного плате прерывания
    int IrqMask;                // маска назначенного плате прерывания
    void interrupt (*OldHandlerPlx)(PARM);      // старый адрес
                                                // обработчика прерывания
    int IsInterruptInProgress;  // флажок нахождения в режиме
                                // работы по прерываниям
    unsigned Bus;               // используется при опросе конфиг. пространства PCI
    unsigned Device;            // используется при опросе конфиг. пространства PCI
    unsigned Function;          // используется при опросе конфиг. пространства PCI
};
```

Перед работой с платами серии *L-7xx* необходимо заполнить поля данной структуры для каждой платы, с которой Вы собираетесь работать, потому что все функции входящие в *API* используют информацию, заложенную в нее. Для каждой платы должен быть свой экземпляр данной структуры и указатель на нее должен передаваться в качестве одного из параметров в функции штатной библиотеки. В штатной библиотеке подпрограмм (набор *API*-функций) для надлежащего заполнения всех полей структуры для каждой из плат серии *L-7xx* вставленных в PC имеется соответствующая функция ***INIT_ACCESS_TO_PLX(struct BOARD_INFO *bi)***. Именно она должна вызываться в начале каждой пользовательской программы. В качестве параметра ей передается указатель на массив структур типа *BOARD_INFO*. Размер массива структур должен быть равен или больше количеству установленных в компьютере плат серии *L-7xx* (можно использовать предопределенную в `plx_api.h` константу **MAXDEVICENUMBER** равную 7). Определить какой именно плате принадлежит конкретный индекс в массиве структур *BOARD_INFO* можно по полю **BoardSerialNumber**, содержащему уникальный серийный номер платы.

1.2.5.1.2. Структура PLATA_DESCR

Структура *PLATA_DESCR* описана в файле `plx_api.h` и представлена ниже:

```
struct PLATA_DESCR
{
    char SerialNumber[9];       // серийный номер платы (8 символов)
    char Name[5];               // название платы (макс. 4 символов):
                                // " L761", "L780" или "L783"
    char Revision;              // ревизия платы: 'A', 'B' или 'C'
    char Dsp_Type[5];           // тип установленного на плате DSP:
                                // "2184" для ADSP-2184,
                                // "2185" для ADSP-2185,
                                // "2186" для ADSP-2186,
    long QuartzFrequency;       // частота установленного на плате кварца в Гц:
                                // 14745600,
                                // 16667000,
                                // 20000000.
    unsigned int IsDacPresented;    // наличие ЦАП'а на плате:
                                    // 0 – ЦАП отсутствует,
                                    // 1 - ЦАП присутствует,
    unsigned int ReservedWord[7];   // зарезервировано
};
```

Данная структура используется в функциях, которые работают со служебной областью пользовательского ППЗУ: *SAVE_PLATA_DESCR_PLX(struct BOARD_INFO *bi, PLATA_DESCR*

pd) и *GET_PLATA_DESCR_PLX(struct BOARD_INFO *bi, PLATA_DESCR *pd)*. Подробности конфигурации этого ППЗУ см. *§ 1.2.2.3 "Формат пользовательского ППЗУ"*. Структура *PLATA_DESCR* используется, например, в функции ***INIT_ACCESS_TO_PLX()*** при чтении служебной информации из пользовательского ППЗУ платы и заполнении соответствующих полей структуры *BOARD_INFO*.

1.2.5.1.3. Глобальные переменные штатной библиотеки

Глобальная переменная *PLX_Board_Quantity* объявлена в файле `plx_api.cpp` и представлена ниже:

```
int PLX_Board_Quantity=0;     // кол-во обнаруженных плат L-7xx
```

В пользовательской программе эту переменную необходимо описать как `extern` и именно в ней после выполнения функции ***INIT_ACCESS_TO_PLX()*** будет находиться количество обнаруженных плат серии *L-7xx*

1.2.5.1.4. Переменные LBIOS

Программист под DOS может напрямую работать с памятью DSP (и программ, и данных), используя библиотечные функции, которые обеспечивают доступ по каналу IDMA процессора ADSP-2184/2185/2186 как к отдельным ячейкам памяти модуля, так и к целым массивам. Эта возможность позволяет программисту работать с платой, непосредственно обращаясь к соответствующим ячейкам памяти программ либо данных. Карты распределения, как памяти программ, так и памяти данных для различных типов DSP, которые могут быть установлены на плате, приведены в *приложении А*.

В **Таблице 7** (см. ниже) приводятся *предопределенные* адреса переменных *LBIOS*, хранящихся в памяти данных DSP, и их краткое описание. Программист может напрямую обращаться к ним, чтобы считать или изменить их содержимое. Собственно сами эти адреса являются $14^{ти}$ битными, но старшие два бита в них могут задаваться в самих *API*-функциях для доступа к *памяти данных* DSP (например, ***GET_DM_WORD_PLX()*** и т.д.) в зависимости от значения поля **BoardDspType** структуры *BOARD_INFO* (тип DSP, на который настроен *LBIOS*). Так при использовании *предопределенных* констант два старших бита в этих адресах задаются как 0x2 для **BoardDspType=0 (**ADSP-2184) и 0x3 для **BoardDspType=1 (**ADSP-2185) и **BoardDspType=2 (**ADSP-2186). Подробности см. *§ 1.2.5.3 "Функции для доступа к памяти DSP"*.

Таблица 7. Адреса управляющих переменных LBIOS

Название переменной	Адрес в Hex формате	Назначение переменной
L_CONTROL_TABLE_PLX	_A00	Управляющая таблица, содержащая последовательность логических номеров каналов (максимум 96). В соответствии с ней DSP производит последовательный циклический сбор данных с АЦП. Размер этой таблицы задается переменной **L_CONTROL_TABLE_ LENGHT_PLX** (см. ниже). По умолчанию – **{ 0, 1, 2, 3, 4, 5, 6, 7}**
L_SCALE_PLX	_D00	Массив с 4 калибровочными коэффициентами, используемый для корректировки масштаба данных с АЦП. По умолчанию – **{ 0x7FFF, 0x7FFF, 0x7FFF, 0x7FFF}**
L_ZERO_PLX	_D04	Массив с 4 калибровочными коэффициентами, используемый для корректировки смещения нуля

		данных с АЦП. По умолчанию – **{ 0x0, 0x0, 0x0, 0x0}**
L_CONTROL_TABLE_ LENGHT_PLX	_D08	Размер управляющей таблицы (максимум 96 логических каналов). По умолчанию – **8**.
L_BOARD_REVISION_ PLX	_D3F	В этой переменной хранится ревизия текущей платы: ‘A’(0x41), ‘B’(0x42) или ‘C’(0x43).
L_READY_PLX	_D40	Флажок готовности платы к дальнейшей работе. После загрузки управляющей программы в DSP, необходимо дождаться установления данного флажка в ‘1’.
L_TMODE1_PLX	_D41	Тестовая переменная. После загрузки драйвера (*LBIOS*) по этому адресу должно читаться число 0x5555.
L_TMODE2_PLX	_D42	Тестовая переменная. После загрузки драйвера (*LBIOS*) по этому адресу должно читаться число 0xAAAA.
L_DAC_IRQ_SOURCE_ PLX	_D43	Содержит информацию об источнике прерываний в PC от ЦАП платы: **0x1** – требуется пересылка очередной порции данных длиной **L_DAC_IRQ_ STEP_PLX** отсчётов в FIFO буфер ЦАП, **0x2** – работа ЦАП остановлена. Только для плат ревизии ‘C’ и выше.
L_DAC_ENABLE_IRQ_ VALUE_PLX	_D44	Переменная, значение которой (при выполнении соответствующей команды) передается в переменную **L_DAC_ENABLE_IRQ_PLX.** Только для плат ревизии ‘C’ и выше.
L_DAC_IRQ_FIFO_ ADDRESS_PLX	_D45	Если произошло прерывание в PC от ЦАП платы, то, начиная с этого адреса можно записать очередных **L_DAC_IRQ_STEP_PLX** отсчетов в FIFO буфер ЦАП. Только для плат ревизии ‘C’ и выше.
L_DAC_IRQ_STEP_PLX	_D46	Переменная, задающая шаг (число отсчетов) в генерировании прерываний в PC от ЦАП платы по мере необходимости в новых данных для FIFO буфера ЦАП. При помощи этой переменной можно сделать так, что прерывания в PC будут генериться, например, через каждые 20 отсчетов, выведенных на ЦАП. По умолчанию – **0x200**. Только для плат ревизии ‘C’ и выше.

L_ENABLE_TTL_OUT_ PLX	_D47	Данная переменная разрешает (0x1) либо запрещает (0x0) использование выходных цифровых линий (т.е. регулирует перевод их в ‘третье’ состояние и обратно). Только для плат ревизии ‘C’ и выше.
L_DSP_TYPE_PLX	_D48	Переменная, передающая драйверу (*LBIOS*) тип установленного на плате DSP. Если она равна 0, то на модуле установлен ADSP-2184 (4 КСлов памяти программ и 4 КСлов памяти данных). Если она равна 1, то – ADSP-2185 (16 КСлов памяти программ и 16 КСлов памяти данных). Если она равна 2, то – ADSP-2186 (8 КСлов памяти программ и 8 КСлов памяти данных). По умолчанию **L_DSP_TYPE_PLX=0.**
L_COMMAND_PLX	_D49	Переменная, при помощи которой драйверу передается номер команды. Краткое описание номеров команд приведено ниже в **Таблице 8**.
L_TTL_OUT_PLX	_D4C	Слово (16 бит), в котором по-битово хранятся значения $16^{\text{ти}}$ выходных цифровых линий для их выставления по команде **C_TTL_OUT_PLX** (см. **Таблицу 8**).
L_TTL_IN_PLX	_D4D	Слово (16 бит), в котором после выполнения команды **C_TTL_IN_PLX** (см. **Таблицу 8**) по-битово хранятся значения $16^{\text{ти}}$ входных цифровых линий.
L_DAC_FIFO_PTR_PLX	_D4F	Переменная, в которой хранится текущий адрес значения, предназначенного для вывода из FIFO буфера ЦАП на сам ЦАП. Данная переменная по мере вывода данных меняет свое значение от **L_DAC_FIFO_BASE_ADDRESS_PLX** до **L_DAC_FIFO_BASE_ADDRESS_PLX** + **L_DAC_FIFO_LENGTH_PLX – 1**.
L_ADC_FIFO_PTR_PLX	_D50	Переменная, в которой хранится текущий адрес заполнения FIFO буфера АЦП в памяти данных DSP. Данная переменная по мере ввода данных с АЦП меняет свое значение от **L_ADC_FIFO_BASE_ADDRESS_PLX** до **L_ADC_FIFO_BASE_ADDRESS_PLX** + **L_ADC_FIFO_LENGTH_PLX – 1**.
L_TEST_LOAD_PLX	_D52	Тестовая переменная.
L_ADC_RATE_PLX	_D53	Переменная, задающая частоту работы АЦП.
L_INTER_KADR_ DELAY_PLX	_D54	Переменная, задающая межкадровую задержку при вводе данных с АЦП.

L_DAC_RATE_PLX	_D55	Переменная, задающая частоту вывода данных с ЦАП'ов.
L_DAC_VALUE_PLX	_D56	Величина, которую требуется однократно установить на выходе ЦАП'а.
L_ENABLE_IRQ_PLX	_D57	Данная переменная разрешает (0x1) либо запрещает (0x0) генерирование прерываний в PC от АЦП платы при соответствующем заполнении FIFO буфера АЦП. По умолчанию – **0x0**.
L_IRQ_STEP_PLX	_D58	Переменная, задающая шаг (число отсчетов) в генерировании прерываний в PC от АЦП платы по мере заполнения FIFO буфера АЦП. При помощи этой переменной можно сделать так, что прерывания в PC будут генериться, например, через каждые 20 отсчетов. По умолчанию – **0x400**.
L_IRQ_FIFO_ ADDRESS_PLX	_D5A	Если произошло прерывание в PC от АЦП платы, то начиная с этого адреса можно считать в PC **L_IRQ_STEP_PLX** отсчетов из FIFO буфера АЦП.
L_ENABLE_IRQ_ VALUE_PLX	_D5B	Переменная, значение которой (при выполнении соответствующей команды) передается в переменную **L_ENABLE_IRQ_PLX.**
L_ADC_SAMPLE_PLX	_D5C	Данная переменная используется при однократном вводе с АЦП, храня считанное значение.
L_ADC_CHANNEL_PLX	_D5D	Данная переменная используется при однократном вводе с АЦП, задавая логический номер канала.
L_CORRECTION_ENABLE _PLX	_D60	Переменная запрещающая (0)/разрешающая (1) корректировку данных аналоговых каналов при помощи калибровочных коэффициентов. По умолчанию **L_CORRECTION_ENABLE_PLX = 0x0**.
L_ADC_ENABLE_PLX	_D62	Переменная запрещающая (0)/разрешающая (1) работу АЦП.
L_ADC_FIFO_BASE_ ADDRESS_PLX	_D63	Текущий базовый адрес FIFO буфера АЦП. По умолчанию **L_ADC_FIFO_BASE_ADDRESS_ PLX = 0x2000**.

L_ADC_FIFO_BASE_ ADDRESS_INDEX_PLX	_D64	Переменная, задающая базовый адрес FIFO буфера АЦП. Может принимать три значения: **0** – базовый адрес начинается с адреса **0x0** только для ADSP-2185 **1** – базовый адрес начинается с адреса **0x2000** только для ADSP-2185 и ADSP-2186 **2** – базовый адрес начинается с адреса **0x3000** для ADSP-2185 и ADSP-2186 и с адреса **0x2000** для ADSP-2184.
L_ADC_FIFO_ LENGTH_PLX	_D65	Текущая длина FIFO буфера АЦП. По умолчанию **L_ADC_FIFO_LENGTH_PLX = 0x800**.
L_ADC_NEW_FIFO_ LENGTH_PLX	_D66	Переменная, задающая длину FIFO буфера АЦП.
L_DAC_ENABLE_ STREAM_PLX	_D67	Переменная запрещающая (0)/разрешающая (1) вывод данных из FIFO буфера ЦАП на сам ЦАП.
L_DAC_FIFO_BASE_ ADDRESS_PLX	_D68	Текущий базовый адрес FIFO буфера ЦАП. Данный буфер расположен в памяти программ DSP. По умолчанию **L_DAC_FIFO_BASE_ ADDRESS_PLX = 0xC00**.
L_DAC_FIFO_ LENGTH_PLX	_D69	Текущая длина FIFO буфера ЦАП. По умолчанию **L_DAC_FIFO_LENGTH_PLX = 0x400**.
L_DAC_NEW_FIFO_ LENGTH_PLX	_D6A	Переменная, задающая длину FIFO буфера ЦАП.
L_DAC_ENABLE_IRQ_ PLX	_D6B	Данная переменная разрешает (0x1) либо запрещает (0x0) генерирование прерываний в PC при работе с ЦАП. По умолчанию – **0x0**. Только для плат ревизии ‘C’ и выше.
L_SYNCHRO_TYPE_PLX	_D70	Переменная, задающая тип синхронизации (аналоговая, цифровая или ее отсутствие).
L_SYNCHRO_AD_ CHANNEL_PLX	_D73	При аналоговой синхронизации задает логический номер канала, по которому происходит синхронизация.
L_SYNCHRO_AD_ POROG_PLX	_D74	При аналоговом типе задает порог синхронизации в кодах АЦП.
L_SYNCHRO_AD_ MODE_PLX	_D75	Переменная, задающая режим аналоговой синхронизации по переходу ‘снизу-вверх’ (**0x0**) или ’сверху-вниз’ (**0x1**).
L_SYNCHRO_AD_ SENSITIVITY_PLX	_D76	Переменная, задающая аналоговую синхронизацию по уровню (**0x0**) либо по переходу (**0x1**).

1.2.5.1.5. Номера команд штатного LBIOS

Фирменный *LBIOS* для плат серии *L-7xx* работает по принципу команд, т.е. в переменной **L_COMMAND** драйверу передается номер команды, которую он должен выполнить. Список доступных номеров команд *LBIOS* приведен ниже в таблице.

Таблица 8. Номера команд *LBIOS*.

Название команды	Номер команды	Назначение	Используемые переменные
C_TEST_PLX	0	Проверка загрузки платы и ее работоспособности.	**L_TEST_LOAD_PLX**
C_LOAD_CONTROL_ TABLE_PLX	1	Загрузка управляющей таблицы в память DSP.	**L_CONTROL_ TABLE_PLX, L_CONTROL_TABLE_ LENGHT_PLX**
C _ENABLE_ADC_ PLX	2	Разрешение/запрещение работы АЦП. Если работа разрешена, то заполнение FIFO буфера АЦП начинается заново.	**L_ADC_ENABLE_PLX**
C_ADC_FIFO_ CONFIG_PLX	3	Конфигурирование параметров FIFO буфера АЦП.	**L_ADC_FIFO_BASE_ ADDRESS_PLX, L_ADC_FIFO_BASE_ ADDRESS_INDEX_PLX, L_ADC_FIFO_ LENGTH_PLX, L_ADC_NEW_FIFO_ LENGTH_PLX**
C_SET_ADC_ KADR_PLX	4	Установка временных параметров работы АЦП.	**L_ADC_RATE_PLX, L_INTER_KADR_ DELAY_PLX**
C_ENABLE_DAC_ STREAM_PLX	5	Разрешение/запрещение потокового вывода данных из FIFO буфера ЦАП. А для плат *Rev. C* можно использовать при этом прерывания в PC.	**L_DAC_ENABLE_ STREAM_PLX L_DAC_IRQ_STEP_PLX L_DAC_ENABLE_IRQ_ PLX**
C_DAC_FIFO_ CONFIG_PLX	6	Конфигурирование параметров FIFO буфера ЦАП.	**L_DAC_FIFO_BASE_ ADDRESS_PLX, L_DAC_FIFO_ LENGTH_PLX, L_DAC_NEW_FIFO_ LENGTH_PLX**

C_SET_DAC_ RATE_PLX	7	Установка частоты вывода данных из FIFO буфера ЦАП'а.	**L_DAC_RATE_PLX**
C_ADC_SAMPLE_ PLX	8	Однократный ввод отсчета АЦП с заданного канала.	**L_ADC_SAMPLE_PLX, L_ADC_CHANNEL_PLX**
C_TTL_IN_PLX	9	Получение состояния 16ти внешних цифровых линий.	**L_TTL_IN_PLX**
C_TTL_OUT_PLX	10	Управление 16тью внешними цифровыми линиями.	**L_TTL_OUT_PLX**
C_SYNCHRO_ CONFIG_PLX	11	Управление синхронизацией начала ввода данных с АЦП.	**L_SYNCHRO_TYPE_PLX, L_SYNCHRO_AD_ CHANNEL_PLX, L_SYNCHRO_AD_ POROG_PLX, L_SYNCHRO_AD_ MODE_PLX, L_SYNCHRO_AD_ SENSITIVITY_PLX**
C_ENABLE_IRQ_PLX	12	Разрешение/запрещение работы с АЦП по прерываниям в PC.	**L_ENABLE_IRQ_PLX, L_ENABLE_IRQ_ VALUE_PLX, L_IRQ_STEP_PLX**
C_IRQ_TEST_PLX	13	Тестовая функция для генерирования прерываний в PC с частотой примерно 300 Гц.	**L_ENABLE_IRQ_PLX**
C_SET_DSP_ TYPE_PLX	14	Передает в драйвер *LBIOS* тип установленного на плате DSP и соответствующим образом модифицирует код драйвера.	**L_DSP_TYPE_PLX**
C_ENABLE_TTL_ OUT_PLX	15	Управление доступом выходных цифровых линий. Для плат ревизии 'C' и выше только при наличии перемычки на контактах 1–2 разъёма **X5** платы. См. *"Руководство пользователя", § 2.2. "Внешний вид плат серии L-7xx"*.	**L_ENABLE_TTL_OUT_ PLX**

1.2.5.2. Функции общего характера

1.2.5.2.1. Функция инициализации доступа к платам серии L-7xx

Формат: **int** ***INIT_ACCESS_TO_PLX***(*struct BOARD_INFO *bi*)
Назначение: Данная функция используется для аккуратного заполнения всех полей массива структур типа *BOARD_INFO* для всех плат серии *L-7xx* установленных в PC. Именно она должна вызываться в начале каждой пользовательской программы, с предварительно объявленным массивом структур типа *BOARD_INFO*. При этом каждой установленной в PC плате *L-7xx* будет соответствовать своя заполненная структура *BOARD_INFO* с соответствующим индексом в массиве. Узнать какой именно плате принадлежит конкретный индекс в массиве структур *BOARD_INFO* можно по полю **BoardSerialNumber**, содержащим серийный номер платы. Т.о. на каждую обнаруженную плату будет получена вся информация, необходимая для работы с ней. Поле **BoardDspType** равно нулю (ADSP-2184) после выполнения данной функции для каждого индекса массива структур *BOARD_INFO* (подробности установки этого поля см. *§ 1.2.5.2.4 "Установка типа DSP"*).
Передаваемые параметры: *bi* – указатель на массив структур типа *BOARD_INFO*. Размер этого массива должен быть не меньше количества установленных в компьютере плат серии *L-7xx*.
Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.2.2. Загрузка LBIOS

Формат: **int** ***LOAD_LBIOS_PLX***(*struct BOARD_INFO *bi*)
Назначение: Данная функция выполняет операцию загрузки фирменного драйвера в память программ и данных DSP. Файлы с кодом драйвера `L761.bio`, `L780.bio` или `L783.bio` должны находиться в текущей директории Вашей программы. Функция производит процедуру загрузки из соответствующего файла *LBIOS*, который по умолчанию настроен на работу с ADSP-2184. Поле **BoardDspType** соответствующей структуры *BOARD_INFO* устанавливается в ноль (ADSP-2184) после выполнения данной функции. Для того чтобы настроить *LBIOS* на работу с другим типом DSP, необходимо выполнить функцию ***SET_DSP_TYPE_PLX()*** (см. ниже).
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы.
Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.2.3. Установка режима доступа к плате

Формат: **int** ***SET_ACCESS_MODE_PLX(****struct BOARD_INFO *bi, int *AccessMode****)***

Назначение:

Данная функция позволяет устанавливать режим доступа к плате, а именно через порты ввода-вывода, через память ниже 1 Мб или через память выше 1 Мб. Текущий режим доступа можно узнать, прочитав поле **BoardAccessMode** в соответствующей структуре ***BOARD_INFO***. В соответствии с установленным режимом доступа все функции штатной библиотеки, которые работают с памятью DSP, посылают команды в DSP и т.д., будут функционировать в заданном режиме доступа. При работе через память значительно возрастает скорость обмена данными между компьютером и платой. **После** выполнения данной функции в переменной ***AccessMode*** будет находится значение **действительно установленного** режима доступа к плате. Это значение может быть равно **3** (предопределенная константа **NO_ACCESS_MODE**), если доступ и через порты, и через память (любую) запрещен. Если именно такое значение получено в переменной ***AccessMode*** после выполнения функции, то работать с платой под DOS невозможно. Вообще разрешить или запретить доступ к плате через порты ввода-вывода и/или память ниже 1 Мб можно с помощью утилиты `\Utils\CHIOMEM\CHIOMEM.EXE` (подробности см. в прил. **C**).

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы,
- *AccessMode* – эта переменная может принимать **только** следующие значения:
  - ✓ **0** или предопределенная константа **IO_ACCESS** для доступа через порты ввода-вывода;
  - ✓ **1** или предопределенная константа **LOW_MEM_ACCESS** для доступа через память, расположенную ниже 1Мб;
  - ✓ **2** или предопределенная константа **HIGH_MEM_ACCESS** для доступа через память, расположенную выше 1Мб.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.2.4. Установка типа DSP

Формат: ***int*** ***SET_DSP_TYPE_PLX***(*struct BOARD_INFO *bi*)

Назначение:

Данная функция считывает из соответствующей ячейки пользовательского ППЗУ тип DSP установленного на плате. Указанный тип DSP зашивается в ППЗУ в процессе наладки платы на *ЗАО «Л-Кард»*. В соответствии с этой информацией происходит настройка *LBIOS* (его предварительно необходимо загрузить в DSP с помощью функции ***LOAD_LBIOS_PLX()***) на работу с конкретным типом DSP. При этом полю **`BoardDspType`** в структуре с указателем *bi* присваивается значение в соответствии с установленным типом DSP (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*). Например, если у Вас на плате установлен ADSP-2185, но **после** загрузки драйвера Вы не вызвали данную функцию, то *LBIOS* останется настроенным на работу с ADSP-2184 (как по умолчанию). Тогда, например, максимальный размер FIFO буфера для АЦП будет не более 0x800 (2048) слов вместо возможных 0x3800 (14336). **Внимание!!!** Данная функция работает надлежащим образом **только** после выполнения API функции ***LOAD_LBIOS_PLX()***.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.2.5. Проверка загрузки платы

Формат: **int** ***PLATA_TEST_PLX****(struct BOARD_INFO *bi)*
Назначение: Данная функция проверяет правильность загрузки *LBIOS* в плату и его работоспособность. **Внимание!!!** Данная функция работает надлежащим образом **только** после выполнения API функции ***LOAD_LBIOS_PLX****()*.
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы.
Возвращаемое значение: 1 – *LBIOS* успешно загружен и функционирует надлежащим образом, 0 – произошла ошибка загрузки или функционирования *LBIOS*.

1.2.5.2.6. Сброс DSP на плате

Формат: **void** ***PLATA_RESET_PLX****(struct BOARD_INFO *bi)*
Назначение: Данная функция производит сброс (RESET) DSP на данной плате. Используется при перезагрузки *LBIOS* или остановки работы DSP. **Необходимо помнить, что после выполнения данной функции работа DSP полностью останавливается и для приведения его в рабочее состояние требуется перезагрузить *LBIOS* (например, с помощью функции *LOAD_LBIOS_PLX*()**).
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы.
Возвращаемое значение: нет

1.2.5.2.7. Передача номера команд для LBIOS

Формат: **int** ***SEND_COMMAND_PLX****(struct BOARD_INFO *bi, int Command)*
Назначение: Данная функция записывает в переменную **L_COMMAND** номер команды и вызывает прерывание IRQ2 в DSP. В ответ на это прерывание, в соответствии с номером переданной команды, *LBIOS* выполняет соответствующие этой команде действия.
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы; • *Command* – номер команды, передаваемый в драйвер.
Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.2.8. Функция завершения работы с платами серии L-7xx

Формат: **void** ***CLOSE_ACCESS_TO_PLX****(void)*
Назначение: Данная функция используется для аккуратного завершения сеанса работы с платами серии *L-7xx*. **Внимание!!!** Данная функция **должна обязательно** вызываться в Вашем приложении перед непосредственным выходом в DOS.
Передаваемые параметры: нет.
Возвращаемое значение: нет.

1.2.5.3. Функции для доступа к памяти DSP

Функции данного раздела обеспечивают доступ как к отдельным ячейкам памяти DSP, так и к целым массивам. Эта возможность позволяет программисту работать с платой напрямую, непосредственно обращаясь к соответствующим ячейкам памяти DSP. Т.о. программа пользователя может обращаться ко всей внутренней памяти сигнального процессора, не прерывая работы самого DSP, что исключительно удобно при построении собственных алгоритмов, работающих в реальном масштабе времени.

Внимание!!! При использовании функций, которые предназначены для работы с *памятью данных* DSP (***GET_DM_WORD_PLX()***, ***PUT_DM_WORD_PLX()*** и т.д.), следует учитывать одну маленькую особенность их работы. В зависимости от того, на какой тип DSP настроен его драйвер, он располагает свои переменные в различных местах памяти данных DSP (см. *приложение А*). Т.е. фактически для различных типов DSP *адреса одних и тех же переменных LBIOS могут имеют различные значения*. Это различие заключается в двух старших битах адреса (сами адреса $14^{ти}$ битные). Для ADSP-2184 эти два старших бита равны 0x2, а для ADSP-2185 и ADSP-2186 – 0x3. Если в качестве адреса используются предопределенные константы типа *L-7xx* (см. **Таблицу 7** и объявления в `plx_api.h`), то функции данного раздела сами аккуратно устанавливают два старших бита адреса в зависимости от того, на какой тип DSP настроен *LBIOS*. Предопределенные константы имеют ‘1’ в $15^{ом}$ бите. Поэтому если этот бит равен ‘0’ (т.е. этот адрес не предопределенная константа), то два старших бита адреса не изменяются и происходит доступ к ячейке(ам) с адресом непосредственно указанным в самой функции.

1.2.5.3.1. Чтение слова из памяти данных DSP

Формат: **int** ***GET_DM_WORD_PLX****(struct BOARD_INFO *bi, int Address)*
Назначение: Данная функция считывает значение слова, находящееся по адресу *Address* в памяти данных DSP.
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы; • *Address* – адрес ячейки в памяти данных DSP, значение которой необходимо считать.
Возвращаемое значение: $16^{ти}$ битное слово, прочитанное по указанному адресу.

1.2.5.3.2. Чтение слова из памяти программ DSP

Формат: **long** ***GET_PM_WORD_PLX****(struct BOARD_INFO *bi, int Address)*
Назначение: Данная функция считывает значение слова, находящееся по адресу *Address* в памяти программ DSP.
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы; • *Address* – адрес ячейки в памяти программ DSP, значение которой необходимо считать.
Возвращаемое значение: $24^{х}$ битное слово, прочитанное по указанному адресу.

1.2.5.3.3. Запись слова в память данных DSP

Формат: **void** ***PUT_DM_WORD_PLX****(struct BOARD_INFO *bi, int Address, int DataWord)*

Назначение:
Данная функция записывает значение *DataWord*, в ячейку с адресом *Address* в памяти данных DSP.

Передаваемые параметры:
- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *Address* – адрес ячейки в памяти данных DSP, куда необходимо записать значение *DataWord*;
- *DataWord* – значение записываемого $16^{ти}$ битного слова.

Возвращаемое значение: нет.

1.2.5.3.4. Запись слова в память программ DSP

Формат: **void** ***PUT_PM_WORD_PLX****(struct BOARD_INFO *bi, int Address, long DataWord)*

Назначение:
Данная функция записывает значение *DataWord*, в ячейку с адресом *Address* в памяти программ DSP.

Передаваемые параметры:
- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *Address* – адрес ячейки в памяти программ DSP, куда записывается значение *DataWord*;
- *DataWord* – значение записываемого $24^{х}$ битного слова.

Возвращаемое значение: нет.

1.2.5.3.5. Чтение массива слов из памяти данных DSP

Формат: **void** ***GET_DM_ARRAY_PLX****(struct BOARD_INFO *bi, int base_address, int N_Points, int *Buffer)*

Назначение:
Данная функция считывает массив слов длиной *N_Points* в буфер *Buffer*, начиная с адреса *base_address* в памяти данных DSP. Буфер *Buffer* необходимо заранее определить.

Передаваемые параметры:
- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *base_address* – стартовый адрес в памяти данных DSP, начиная с которого производится чтение массива;
- *N_Points* – длина считываемого массива;
- *Buffer* – указатель на буфер, в который передаются считываемые значения.

Возвращаемое значение: нет.

1.2.5.3.6. Чтение массива слов из памяти программ DSP

Формат: **void** ***GET_PM_ARRAY_PLX****(struct BOARD_INFO *bi, int base_address, int N_Points, int *Buffer)*

Назначение:

Данная функция считывает массив слов длиной *N_Points* в буфер *Buffer*, начиная с адреса *base_address* в памяти программ DSP. Буфер *Buffer* необходимо заранее определить. При использовании этой функцией следует помнить, что одно слово памяти программ DSP является $24^{\text{х}}$ битным. Следовательно, для получения целого слова из памяти программ необходимо **обязательно** прочитать два слова размером 16 бит (подробности см. *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 11 "DMA Ports", § 11.3 "IDMA Port", cmp. 11-12, Analog Devices, Inc., Third Edition September 1995* или на сайте www.analog.com). Например, для чтения одного слова 0x18030F из памяти программ необходимо определить буфер в два $16^{\text{ти}}$ слова. Тогда после выполнения данной функции с *N_Points*=2 буфер будет содержать {0x1803, 0x000F}, а с *N_Points*=1 – {0x1803}.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *base_address* – стартовый адрес в памяти программ DSP, начиная с которого производится чтение массива;
- *N_Points* – число считываемых $16^{\text{ти}}$ битных слов;
- *Buffer* – указатель на буфер, в который передаются считываемые значения.

Возвращаемое значение: нет.

1.2.5.3.7. Запись массива слов в память данных DSP

Формат: **void** ***PUT_DM_ARRAY_PLX****(struct BOARD_INFO *bi, int base_address, int N_Points, int *Buffer)*

Назначение:

Данная функция записывает массив слов длиной *N_Points* из буфера *Buffer* в память данных DSP, начиная с адреса *base_address.*

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *base_address* – стартовый адрес в памяти данных DSP, начиная с которого производится запись массива;
- *N_Points* – длина записываемого массива;
- *Buffer* – указатель на буфер, из которого идет запись.

Возвращаемое значение: нет.

1.2.5.3.8. Запись массива слов в память программ DSP

Формат: **void** ***PUT_PM_ARRAY_PLX****(struct BOARD_INFO *bi, int base_address, int N_Points, int *Buffer)*

Назначение:

Данная функция записывает массив слов длиной *N_Points* из буфера *Buffer* в память программ DSP, начиная с адреса *base_address*. При использовании этой функцией следует помнить, что одно слово памяти программ DSP является $24^{\underline{x}}$ битным. Следовательно, для записи слова в память программ необходимо **обязательно** два слова размером 16 бит (подробности см. *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 11 "DMA Ports", § 11.3 "IDMA Port", cmp. 11-12, Analog Devices, Inc., Third Edition September 1995* или на сайте www.analog.com). Например, для записи одного слова 0x18030F в память программ необходимо сформировать буфер в два $16^{\underline{ти}}$ слова: {0x1803, 0x000F}.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *base_address* – стартовый адрес в памяти программ DSP, начиная с которого производится запись массива;
- *N_Points* – число записываемых $16^{\underline{ти}}$ битных слов;
- *Buffer* – указатель на буфер, из которого идет запись.

Возвращаемое значение: нет.

1.2.5.4. Функции для работы с АЦП

1.2.5.4.1. Разрешение/запрещение работы АЦП

Формат: **int** ***ENABLE_ADC_PLX****(struct BOARD_INFO *bi, int ADC_Enable)*

Назначение:

Данная функция останавливает либо запускает заново АЦП. Если АЦП запускается заново, то при этом заполнение FIFO буфера АЦП, находящегося в памяти данных DSP, начинается сначала. Непосредственно **после** загрузки *LBIOS* работа АЦП разрешена.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *ADC_Enable* – переменная может принимать следующие значения:
  - ✓ если **0**, то работа АЦП останавливается,
  - ✓ если не **0**, то АЦП запускается заново.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.4.2. Загрузка управляющей таблицы

Формат: **int** ***LOAD_CONTROL_TABLE_PLX****(struct BOARD_INFO *bi, int Control_Table_Length, int *Control_Table)*

Назначение:

Данная функция загружает в память данных DSP управляющую таблицу, которая используется *LBIOS* для задания циклической последовательности (длиной *Control_Table_Length*) оцифровки входных аналоговых каналов (логические номера каналов находятся в массиве *Control_Table*). В FIFO буфер АЦП, находящийся в памяти данных DSP, отсчеты будут циклически складываться именно в виде той последовательности, как указанно в массиве *Control_Table*. Например, 1ым элементом в FIFO буфере АЦП будет отсчет с логического канала, находящегося в *Control_Table[0]*, 2ым - *Control_Table[1]* **…** *Control_Table_Length*ым – *Control_Table[Control_Table_Length-1], (Control_Table_Length+1)*ым - *Control_Table[0]* и т.д. Если задаваемая длина управляющей таблицы *Control_Table_Length* не кратна текущей длине FIFO буфера АЦП, то данная функция сама устанавливает корректный размер этого буфера с помощью API функции ***ADC_FIFO_CONFIG_PLX()*** (см *§ 1.2.5.4.6 "Конфигурирование FIFO буфера АЦП"*). **Внимание!!!** Данная функция не должна запускаться в момент работы платы по прерываниям (функция вернет ошибку).

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *Control_Table_Length* – длина управляющей таблицы (максимально 96);
- *Control_Table* – указатель на управляющую таблицу (целочисленный массив), содержащую последовательность логических номеров (кадр).

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.4.3. Установка временных параметров работы АЦП

Формат: **int** ***SET_KADR_TIMING_PLX***(*struct BOARD_INFO *bi, double *ADC_Rate, double *Inter_Kadr_Delay)*

Назначение:

Данная функция выполняет установку таких важных параметров функционирования платы, как частота оцифровки данных (частота работы АЦП, обратная величина межканальной задержки) и межкадровую задержку. **После** выполнения этой функции в переменных *ADC_Rate* и *Inter_Kadr_Delay* находятся **реально установленные** значения межканальной и межкадровой задержек (максимально близкие к задаваемым). Это происходит вследствие того, что реальные значения *ADC_Rate* и *Inter_Kadr_Delay* не являются непрерывными, а принадлежат некоему дискретному ряду (своему для каждого типа плат). Поэтому данная функция просто находит такую дискретную величину (ближайшую к задаваемой), передает ее *LBIOS*, а также возвращает ее значение Вам в соответствующей переменной. При этом под кадром подразумевается совокупность отсчетов с логических каналов от *Control_Table[0]* до *Control_Table[Control_Table_Length-1]*. Формат кадра можно посмотреть в *§ 1.2.2.4 "Формат кадра отсчетов"*.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *ADC_Rate* – определяет частоту работы АЦП (межканальную задержку) в *кГц*;
- *Inter_Kadr_Delay* – определяет величину межкадровой задержки в *млс*. Например, если задать эту переменную равной **0**, то установится межкадровая задержка, соответствующая минимально возможной, т.е. 1/*ADC_Rate*. Именно это значение вернется Вам в данной переменной.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.4.4. Однократный ввод с переустановкой канала АЦП

Формат: **int** ***ADC_SAMPLE_PLX***(*struct BOARD_INFO *bi, int ADC_Channel, int *ADC_Sample)*

Назначение:

Данная функция устанавливает заданный логический канал и осуществляет его однократное аналого-цифровое преобразование. Эта функция удобна для осуществления асинхронного ввода данных с различных входных аналоговых каналов.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *ADC_Channel* – логический номер канала АЦП;
- *ADC_Sample* – результат преобразования по заданному логическому каналу АЦП.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.4.5. Получение массива данных с АЦП

Формат: **int** ***GET_ADC_DATA_PLX****(struct BOARD_INFO *bi, int *Buffer, int N_Points)*

Назначение:

Данная функция обеспечивает последовательное считывание *N_Points* отсчетов из FIFO буфера АЦП, находящегося в памяти данных DSP, в массив *Buffer*, данные в котором находятся в по-кадровом виде (сначала первые отсчеты целого кадра, потом вторые и т.д.). При этом под кадром подразумевается совокупность отсчетов с логических каналов, значения которых хранятся в управляющей таблице в памяти данных DSP. Для того чтобы эта функция нормально функционировала необходимо, чтобы предварительно работа АЦП была разрешена (с помощью функции ***ENABLE_ADC_PLX()***).

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *Buffer* – указатель на массив, в который складываются принимаемые данные;
- *N_Points* – количество отсчетов (не более 32000), которые необходимо положить в *Buffer*.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.4.6. Конфигурирование FIFO буфера АЦП

Формат: **int** ***ADC_FIFO_CONFIG_PLX****(struct BOARD_INFO *bi, int AddressIndex, int Length)*

Назначение:

Данная функция обеспечивает установку параметров FIFO буфера АЦП. Устанавливаемая длина FIFO буфера АЦП должна быть кратна размеру кадра (числу отсчетов в кадре), иначе функция сама подбирает подходящий размер буфера. Минимальный размер FIFO буфера АЦП равен 512 слов. Если функция успешно выполнилась, то **реально** установленный начальный адрес FIFO буфера АЦП и его длину можно прочитать в памяти данных DSP, используя соответственно предопределенные адреса **L_ADC_FIFO_BASE_ADDRESS_PLX** и **L_ADC_FIFO_LENGTH_PLX**. **Внимание!!!** Данная функция не должна запускаться в момент работы платы по прерываниям (функция вернет ошибку).

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *AddressIndex* – эта переменная может принимать три значения
  - ✓ если **0**, то FIFO буфер АЦП начинается с адреса 0x0 в памяти данных (DM) DSP,
    - его максимальный размер - 0x3800 (14336) слов
    - (**!!!!! только для ADSP-2185 !!!!!**);
  - ✓ если **1**, то FIFO буфер АЦП начинается с адреса 0x2000 (8192) в памяти данных (DM) DSP
    - его максимальный размер - 0x1800 (6144) слов
    - (**!!!!! для ADSP-2185 и ADSP-2186 !!!!!**);
  - ✓ если **2**, то FIFO буфер АЦП начинается с адреса 0x2000 (8192) в DM для **ADSP-2184** или
    - с адреса 0x3000 (12288) в DM для **ADSP-2185** и **ADSP-2186**,
    - его максимальный размер - 0x800 (2048) слов
    - (**!!!!! для ADSP-2184, ADSP-2185 и ADSP-2186 !!!!!**);
- *Length* – длина FIFO буфера АЦП (должна быть кратна размеру кадра).

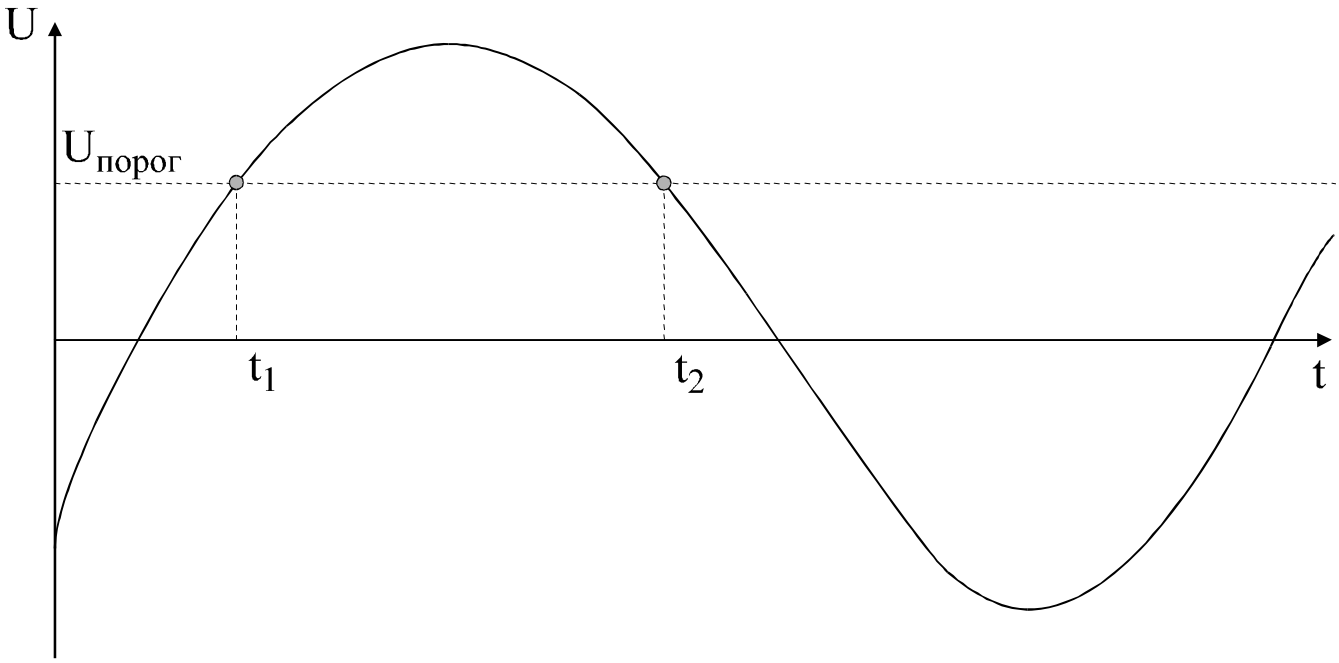
Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.4.7. Синхронизация ввода с АЦП

Формат: **int** ***SYNCHRO_CONFIG_PLX****(struct BOARD_INFO *bi, int SynchroType, int SynchroSensitivity, int SynchroMode, int AdChannel, int AdPorog, int PointRange)*

Назначение:

Данная функция задает режим синхронизации ввода с АЦП: цифровая синхронизация старта, покадровая цифровая синхронизация или аналоговая синхронизация по выбранному логическому каналу АЦП. При цифровой синхронизации старта буфер АЦП начинает заполняться только **после** прихода отрицательного перепада ТТЛ импульса длительностью не менее 50 *нс* на вход **TRIG** внешнего разъёма. При по-кадровой цифровой синхронизации после прихода импульса (см. выше) в FIFO буфер АЦП складываются отсчеты только одного кадра. При аналоговой синхронизации буфер АЦП начинает заполняться после выполнения определенных соотношений между значением отсчета с логического канала синхронизации и заданным пороговым значением (см. ниже). Различные моменты старта аналоговой синхронизации приведены на следующем рисунке:



Если, например, задана аналоговая синхронизация по **переходу** (*SynchroSensitivity*≠0) ‘снизу-вверх’ (*SynchroMode*=0) при пороговом значении равном $U_{порог}$, то FIFO буфер АЦП начнет заполняться данными только после наступления момента времени t_1, т.е. тогда, когда уровень входного сигнала на синхроканале *AdChannel* пересечет пороговую линию в направлении снизу вверх. Аналогично если задан **переход** ‘сверху-вниз’ (*SynchroMode*≠0), то старт заполнения буфера наступит в момент времени t_2, когда уровень входного сигнала на синхроканале пересечет пороговую линию в направлении сверху вниз. Если же синхронизация задается по **уровню** (*SynchroSensitivity*=0), то заполнение буфера начнется, когда уровень входного сигнала на синхроканале либо выше (*SynchroMode*=0), либо ниже (*SynchroMode*≠0) пороговой линии $U_{порог}$.

После наступления условий синхронизации для цифрового старта или аналогового ввода, данные начинают непрерывно поступать в FIFO буфер АЦП. Для повторного выполнения цифровой синхронизации старта или аналоговой синхронизации по выбранному каналу АЦП необходимо снова использовать данную функцию с соответствующими параметрами.

При по-кадровой цифровой синхронизации *LBIOS* находится в постоянном ожидании появления очередного синхроимпульса. Для выхода их этого положения в нормальный режим сбора данных необходимо выполнить данную функцию с переменной *SynchroType* больше двух (см. ниже).

Данная функция, если выполнены условия синхронизации, разрешает работу АЦП, даже если до этого его функционирование было запрещено с помощью функции ***ENABLE_ADC_PLX()***.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *SynchroType* – переменная может принимать следующие значения:
  - ✓ если **0**, то цифровая синхронизация старта (остальные параметры не используются);
  - ✓ если **1**, то по-кадровая цифровая синхронизация (остальные параметры не используются);
  - ✓ если **2**, то аналоговая синхронизация по выбранному каналу АЦП;
  - ✓ если **>2**, то выход из режима синхронизация (остальные параметры не используются);
- *SynchroSensitivity* – переменная может принимать следующие значения:
  - ✓ если **0**, то аналоговая синхронизация по **уровню**,
  - ✓ если не **0**, то аналоговая синхронизация по **переходу**;
- *SynchroMode* - переменная может принимать следующие значения:
  - ✓ если **0**, то
    - аналоговая синхронизация по **уровню** ‘выше’,
    - аналоговая синхронизация по **переходу** ‘снизу-вверх’,
  - ✓ если не **0**, то
    - аналоговая синхронизация по **уровню** ‘ниже’,
    - аналоговая синхронизация по **переходу** ‘сверху-вниз’;
- *AdChannel* – логический номер канала АЦП для аналоговой синхронизации;
- *AdPorog* – пороговое значение (в кодах АЦП) для аналоговой синхронизации;
- *PointRange* – не используется.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.4.8. Загрузка калибровочных коэффициентов АЦП

Формат: **int** ***LOAD_COEF_PLX****(struct BOARD_INFO *bi)*

Назначение:

Данная функция обеспечивает загрузку калибровочных коэффициентов АЦП из соответствующей области пользовательского ППЗУ в надлежащие места в памяти данных сигнального процессора. Для того, чтобы *LBIOS* начал корректировку входных данных АЦП с помощью этих коэффициентов, необходимо воспользоваться API функцией ***ENABLE_CORRECTION_PLX()*** (см. ниже). При этом в FIFO буфере АЦП будут находится уже откорректированные значения входных сигналов.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.4.9. Управление корректировкой данных с АЦП

Формат: **int** ***ENABLE_CORRECTION_PLX****(struct BOARD_INFO *bi, int Correction_Enable)*

Назначение:

Данная функция запрещает/разрешает корректировку входных данных с АЦП. Предварительно необходимо загрузить калибровочные коэффициенты по соотвествующим адресам в памяти данных DSP с помощью функции ***LOAD_COEF_PLX()*** (см. выше). Если использование калибровочных коэффициентов разрешено, то в FIFO буфере АЦП будут находится уже откорректированные значения входных сигналов.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *Correction_Enable* – переменная может принимать следующие значения:
  - ✓ если **0**, то корректировка запрещена,
  - ✓ если **1**, то корректировка разрешена.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.5. Функции для работы с ЦАП

1.2.5.5.1. Управление работой ЦАП

Формат: **int** ***ENABLE_DAC_STREAM_PLX****(struct BOARD_INFO *bi, int DAC_Control, int Enable_IRQ, int IRQ_Step)*

Назначение:

Данная функция позволяет управляеть выводом данных из циклического буфера ЦАП, находящегося в памяти **программ** DSP, на сам $12^{ти}$ битный ЦАП. Непосредственно **после** загрузки *LBIOS* вывод из циклического буфера ЦАП запрещен. Следует помнить, что сам FIFO буфер ЦАП расположен в памяти программ DSP и, следовательно, ячейки этой памяти являются $24^{х}$ битными. Из этих $24^{х}$ битов при непосредственном выводе на ЦАП задействованы только старшие 16 бит. Т.е. биты 0÷11 в слове данных задают собственно сам код ЦАП, бит 12 осуществляет выбор номера канала, а биты 13, 14 и 15 не используются. Перед разрешением вывода из циклического FIFO буфера ЦАП необходимо корректно заполнить его требуемыми данными, например с помощью функции ***PUT_PM_ARRAY_PLX()***. Данные формируются в кодах ЦАП, с возможной предварительной корректировкой этих кодов с помощью коэффициентов записанных в ППЗУ (см. *§ 1.2.2.3 "Формат пользовательского ППЗУ"*). Начальный адрес в памяти программ и длину буфера ЦАП можно прочитать из памяти данных DSP, используя соответственно предопределенные адреса **L_DAC_FIFO_BASE_ADDRESS_PLX** и **L_DAC_FIFO_LENGTH_PLX**. Адрес текущего отсчёта, предназначенного для вывода на ЦАП, можно узнать, прочитав содержимое ячейки с предопределенным адресом **L_DAC_FIFO_PTR_PLX**. Данная функция позволяет, например, реализовывать двухканальный генератор сигналов произвольной формы. В частности, можно обеспечивать вывод стереоаудио файлов (например, типа *.wav).

Для плат *Rev. C* данная функция предоставляет дополнительные возможности при работе с ЦАП. Так плата данной ревизии вполне может работать в режиме генерации прерываний в PC по мере возникновения необходимости передачи новых данных в FIFO буфер ЦАП. Реально установленный шаг прерываний *IRQ_Step* можно узнать, прочитав после успешного выполнения данной функции, переменную *LBIOS* с предопределенным адресом **L_DAC_IRQ_STEP_PLX**. При получении прерывания в PC можно записать в FIFO буфер ЦАП *IRQ_Step* новых отсчетов, начиная с адреса, значение которого находится в памяти данных DSP в переменной с предопределенным адресом **L_DAC_IRQ_FIFO_ADDRESS_PLX**. Например, если разрешить работу ЦАП и прерывания от него, а также задать переменную *IRQ_Step* равной половине длины FIFO буфера ЦАП, то после выполнения функции ***ENABLE_DAC_STREAM_PLX()*** драйвер *LBIOS* начнет последовательно выводить данные из FIFO буфера с частотой, заданной функцией ***SET_DAC_RATE_PLX()***. Как только завершится вывод на ЦАП первой половинки FIFO буфера *LBIOS* сгенерирует прерывание в PC, сигнализируя о том, что первую половинку FIFO буфера можно заполнять новыми данными. Операцию пересылки следующей части данных для ЦАП можно осуществить с помощью штатной функции ***PUT_PM_ARRAY_PLX()***. При этом *LBIOS* будет продолжать вывод данных на ЦАП из второй половинки FIFO буфера. Далее вся эта процедура циклически повторяется. Т.о. данный подход позволяет реализовать алгоритм непрерывного потока данных на ЦАП с заданной частотой вывода.

Для плат *Rev. C* существует одна интересная особенность при работе с ЦАП: существует возможность остановить работающий ЦАП не выполняя ф. ***ENABLE_DAC_STREAM_PLX()***. Для этого *LBIOS* должен обнаружить в качестве очередного выводимого на ЦАП отсчёта число-признак равное 0xFFFF. При этом дальнейшая работа ЦАП прекращается и генерируется информационное прерывание в PC.

Т.о. при описании данной функции упоминались два различных источника прерывания от ЦАП платы в PC: нужны новые данные в буфер ЦАП и работа ЦАП приостановлена. Поскольку прерывние одно, то необходимо их как-то различить в Вашей пользовательской программе на PC. Для этого используется предопределённая переменная **L_DAC_IRQ_SOURCE_PLX**. Так если в Вашем обработчике прерываний в PC по адресу **L_DAC_IRQ_SOURCE_PLX** считалось:

- ✓ 0x1 – то это прерывание от ЦАП, сигнализирующее о необходимости передачи в его FIFO буфер очередных **L_DAC_IRQ_STEP_PLX** отсчётов;
- ✓ 0x2 – то это прерывание от ЦАП, сигнализирующее о приостановке его работы.

Достаточно подробно прокомментированные исходные тексты примера по использованию функций штатной библиотеки для целей работы с ЦАП как в программном режиме, так и по прерываниям можно найти на фирменном CD-ROM в директории `\Examples\WAVPLAY.DOS`.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *DAC_Control* – переменная может принимать следующие значения:
  - ✓ если **0**, то вывод значений из циклического буфера ЦАП запрещается,
  - ✓ если не **0**, то вывод значений из циклического буфера ЦАП разрешен.
- *Enable_IRQ* – разрешение/запрещение использования прерываний при работе с ЦАП
- *IRQ_Step* – шаг, с которым генерируется прерывание по мере вывода данных из FIFO буфера ЦАП. При помощи этой переменной можно сделать так, что плата будет генерировать прерывания в PC каждый раз по мере возникновения необходимости передачи очередных *IRQ_Step* отсчётов в FIFO буфер ЦАП.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.5.2. Установка частоты работы ЦАП

Формат: **int** ***SET_DAC_RATE_PLX****(struct BOARD_INFO *bi, double *DAC_Rate)*

Назначение:

Данная функция выполняет установку частоты работы ЦАП. **После** выполнения функции в переменной *DAC_Rate* находится **реально установленное** значение частоты вывода данных с ЦАП (максимальное значение 125 *кГц*).

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *DAC_Rate* – задает частоту работы ЦАП (частота вывода данных с ЦАП) в *кГц*.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.5.3. Конфигурирование FIFO буфера ЦАП

Формат: **int** ***DAC_FIFO_CONFIG_PLX****(struct BOARD_INFO *bi, int DacFifoLength)*

Назначение:

Данная функция обеспечивает установку параметров FIFO буфера ЦАП, находящегося в памяти программ (PM) DSP. Если функция успешно выполнилась, то **реально** установленный начальный адрес FIFO буфера ЦАП и его длину можно прочитать в памяти данных DSP, используя соответственно предопределенные адреса **L_DAC_FIFO_BASE_ADDRESS_PLX** и **L_DAC_FIFO_LENGTH_PLX**.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *DacFifoLength* – длина FIFO буфера ЦАП, максимальное значение которой в зависимости от типа DSP может быть:
  - ✓ для ADSP-2184 – 0x400 (1024), а начальный адрес FIFO – 0xC00 (3072) в PM DSP;
  - ✓ для ADSP-2185 – 0x1000 (4096), а начальный адрес FIFO – 0x3000 (12288) в PM DSP;
  - ✓ для ADSP-2186 – 0x800 (2048), а начальный адрес FIFO – 0x1800 (6144) в PM DSP.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.5.4. Однократный вывод на ЦАП

Формат: **int** ***SET_DAC_SAMPLE_PLX****(struct BOARD_INFO *bi, int DAC_Number, int *DAC_Value)*

Назначение:

Данная функция однократно устанавливает на указанном канале ЦАП *DAC_Number* напряжение в соответствии со значением *DAC_Value* (в кодах ЦАП).

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *DAC_Number* – номер канала ЦАП (**0** или **1**).
- *DAC_Value* – устанавливаемое значение в кодах ЦАП (от –2048 до 2047).

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.6. Функции для работы с прерываниями

Общая идея приема данных по прерываниям достаточно проста. DSP вводит заданный ряд отсчетов с соответствующих каналов АЦП в свой FIFO буфер и, после этого, генерит прерывание в PC. В Вашей программе в PC предварительно должен быть организован соответствующий обработчик данного прерывания, который должен считывать из платы вводимый ряд отсчетов. Если у Вас плата *Rev. C*, то аналогичную работу можно организовать для потокового вывода данных на ЦАП. Т.е. после вывода на ЦАП заданного ряда отсчетов, DSP сгенерит прерывание в PC, сигнализируя о необходимости передачи очередной порции новых данных в свой FIFO буфер ЦАП.

Пример корректной работы с АЦП по прерываниям можно найти в директории `Examples\EXAMPLE.DOS` (файл `disk.cpp`) или `Examples\WriteDat.DOS` (файл `WRITEDAT.CPP`). Аналогично для ЦАП – в директории `Examples\WAVPLAY.DOS` (файл `WAVPLAY.CPP`).

1.2.5.6.1. Установка обработчика прерываний

Формат: **int** ***INIT_INTERRUPT_PLX****(struct BOARD_INFO *bi, void interrupt IRQ_Handler_PLX(PARM))*

Назначение:

Данная функция программирует надлежащим образом контроллер прерываний компьютера, устанавливает обработчик прерываний, адрес которого передается в параметре *IRQ_Handler_PLX* и разрешает микросхеме PCI-интерфейса аппаратно генерировать прерывания. **После** успешного выполнения данной функции необходимо разрешить генерирование прерывания *LBIOS* (функция ***ENABLE_IRQ_PLX()*** и/или ***ENABLE_DAC_STREAM_PLX()***). Обработчик прерываний **обязательно должен** позаботиться о сбросе, как контроллера прерываний компьютера, так и триггера прерываний на самой плате, используя функцию ***RESET_IRQ_PLX()*** (см. ниже).

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *IRQ_Handler_PLX* – адрес обработчика прерываний.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.6.2. Использование прерываний при работе с АЦП

Формат: **int** ***ENABLE_IRQ_PLX****(struct BOARD_INFO *bi, int Enable_IRQ, int IRQ_Step)*

Назначение:

Данная функция задает параметры работы платы в режиме генерации прерываний в PC по мере заполнения FIFO буфера АЦП. Реально установленный шаг прерываний *IRQ_Step* можно узнать, прочитав после успешного выполнения данной функции, переменную *LBIOS* с предопределенным адресом **L_IRQ_STEP_PLX**. При получении прерывания в PC можно считать из FIFO буфера АЦП *IRQ_Step* отсчетов, начиная с адреса, значение которого находится в памяти данных DSP в переменной с предопределенным адресом **L_IRQ_FIFO_ADDRESS_PLX**.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы;
- *Enable_IRQ* – переменная может принимать следующие значения:
  - ✓ Если **0**, то плате запрещено генерировать прерывания в PC,
  - ✓ Если не **0**, то плате разрешено генерировать прерывания в PC по мере заполнения FIFO буфера АЦП (см. ниже);
- *IRQ_Step* – шаг, с которым генерируется прерывание по мере заполнения FIFO буфера АЦП.

	При помощи этой переменной можно сделать так, что плата будет генерировать прерывания в PC каждый раз по мере получения очередных *IRQ_Step* заполнений FIFO буфера АЦП.
Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.	

1.2.5.6.3. Останов прерываний

Формат: **int** ***STOP_INTERRUPT_PLX****(struct BOARD_INFO *bi)*
Назначение: Данная функция осуществляет восстановление старого обработчика прерывания и запрещает микросхеме PCI-интерфейса генерировать прерывания в PC. Перед вызовом данной функции следует запретить *LBIOS* генерировать прерывания в PC (функция ***ENABLE_IRQ_PLX()*** и/или ***ENABLE_DAC_STREAM_PLX()***).
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы.
Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.6.4. Сброс прерываний

Формат: **void** ***RESET_IRQ_PLX****(struct BOARD_INFO *bi, unsigned int IrqSource)*
Назначение: Данная функция осуществляет сброс контроллера прерываний компьютера и триггера прерываний на самой плате. Эта функция **должна обязательно** вызываться в обработчике прерываний.
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы. • *IrqSource* – содержит источник прерывания, запрос которого необходимо сбросить. Может принимать два значения: **ADC_INTR** – для сброса прерывания от АЦП и **DAC_INTR** – для сброса прерывания от ЦАП.
Возвращаемое значение: нет.

1.2.5.7. Функции для работы с внешними цифровыми линиями

1.2.5.7.1. Разрешение выходных цифровых линий

Формат: **int** ***ENABLE_TTL_OUT_PLX****(struct BOARD_INFO *bi, int EnableTtlOut)* **(только для плат *Rev. C* и выше)**
Назначение: Данная функция позволяет для плат *Rev. C* осуществлять управление доступом выходных цифровых линий, т.е. перевод их в '*третье*' состояние и обратно. **ВНИМАНИЕ!!!** Для того, чтобы иметь возможность воспользоваться таким режимом, необходимо с помощью перемычки замкнуть контакты 1–2 на разъёме **X5** платы (см. *§ 1.2.4.5. "Особенности плат L-780M (Rev. C)"*).
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы. • *EnableTtlOut* – флажок, позволяющий (0x1) либо запрещающий (0x0) использование цифровых выходов.
Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.7.2. Чтение внешних цифровых линий

Формат: **int** ***TTL_IN_PLX****(struct BOARD_INFO *bi, unsigned int *Ttl_In)*
Назначение: Данная функция осуществляет чтение состояния 16^{ти} входных цифровых TTL линий.
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы. • *Ttl_In* – переменная, содержащая побитовое состояние цифровых входов.
Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.7.3. Вывод на внешние цифровые линии

Формат: **int** ***TTL_OUT_PLX****(struct BOARD_INFO *bi, unsigned int *Ttl_Out)*
Назначение: Данная функция осуществляет установку 16^{ти} выходных цифровых TTL линий в соответствии с битами параметра *Ttl_Out*.
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы. • *Ttl_Out* – переменная, содержащая побитовое состояние устанавливаемых цифровых выходов
Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.8. Функции для работы с пользовательским ППЗУ

1.2.5.8.1. Разрешение/запрещение записи в ППЗУ

Формат: **void** ***ENABLE_FLASH_WRITE_PLX****(struct BOARD_INFO *bi, int Flag)*

Назначение:

Данная функция разрешает либо запрещает процедуру записи в пользовательское ППЗУ с помощью функции ***WRITE_FLASH_WORD_PLX()***. **После** разрешения записи в ППЗУ и окончания самой процедуры записи **необходимо** запретить запись в ППЗУ.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы.
- *Flag* – переменная может принимать следующие значения:
  - ✓ Если **0**, то процедура записи в ППЗУ запрещена,
  - ✓ Если не **0**, то процедура записи в ППЗУ разрешена.

Возвращаемое значение: нет.

1.2.5.8.2. Запись слова в ППЗУ

Формат: **int** ***WRITE_FLASH_WORD_PLX****(struct BOARD_INFO *bi, int FlashAddress, int FlashWord)*

Назначение:

Данная функция выполняет процедуру записи слова *FlashWord* в ячейку пользовательского ППЗУ с адресом *FlashAddress*. Перед началом процедуры записи в ППЗУ **необходимо** разрешить ее с помощью функции ***ENABLE_FLASH_WRITE_PLX()***. **После** окончания цикла записи всей информации **необходимо** запретить запись в ППЗУ с помощью той же функции ***ENABLE_FLASH_WRITE_PLX()***. Т.к. в первых 32 ячейках ППЗУ находятся служебная информация, а также калибровочные коэффициенты АЦП и ЦАП, то для пользователя доступны адреса ячеек **только** с 32 по 63.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы.
- *FlashAddress* – адрес ячейки, куда будет записано слово *FlashWord*;
- *FlashWord* – слово, значение которого должно быть записано в ППЗУ.

Возвращаемое значение: 1 – функция успешно выполнена, 0 – функция не выполнена.

1.2.5.8.3. Чтение слова из ППЗУ

Формат: **int** ***READ_FLASH_WORD_PLX****(struct BOARD_INFO *bi, int FlashAddress)*

Назначение:

Данная функция возвращает значения слова, находящегося в ячейке пользовательского ППЗУ с адресом *FlashAddress*.

Передаваемые параметры:

- *bi* – указатель на структуру типа *BOARD_INFO* для данной платы.
- *FlashAddress* – адрес ячейки, откуда будет считано слово.

Возвращаемое значение: значение слова из ППЗУ.

1.2.5.8.4. Чтение служебной информации из ППЗУ

Формат: **void** ***GET_PLATA_DESCR_PLX(****struct BOARD_INFO *bi, PLATA_DESCR *pd****)***
Назначение: Данная функция осуществляет чтение служебной информации из пользовательского ППЗУ в структуру типа *PLATA_DESCR* (см. *§ 1.2.5.1.2 "Структура PLATA_DESCR"*) .
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы. • *pd* – указатель на структуру типа *PLATA_DESCR*, в которую заносится вся служебная информация.
Возвращаемое значение: нет.

1.2.5.8.5. Запись служебной информации в ППЗУ

Формат: **void** ***SAVE_PLATA_DESCR_PLX(****struct BOARD_INFO *bi, PLATA_DESCR *pd****)***
Назначение: Данная функция позволяет сохранять всю служебную информацию из структуры типа *PLATA_DESCR* (см. *§ 1.2.5.1.2 "Структура PLATA_DESCR"*) в пользовательском ППЗУ. **Внимание!!! Пользоваться данной функцией нужно только в случае крайней необходимости. Например, когда по тем или иным обстоятельствам испортилось содержимое служебной информации в пользовательском ППЗУ.**
Передаваемые параметры: • *bi* – указатель на структуру типа *BOARD_INFO* для данной платы. • *pd* – указатель на структуру типа *PLATA_DESCR*, из которой служебная информация заносится в ППЗУ.
Возвращаемое значение: нет.

1.2.6. Особенности работы штатной библиотеки с платами L-783

Высокоскоростная плата *L-783* при использовании функций штатной библиотеки обладает некоторыми ограничениями в своем функционировании на частотах работы АЦП выше 1000 кГц. Фактически этих ограничений три, а именно:

- Если задаваемая в функции ***SET_KADR_TIMING_PLX()*** частота АЦП выше 1000 кГц, то параметр межкадровой задержки $\mathbf{t_{мкз}}$=**Inter_Kadr_Delay** жестко устанавливается равным интервалу запуска АЦП (межканальной задержки) $\mathbf{t_{АЦП}}$=1/**ADC_Rate** (см. *§ 1.2.2.4 "Формат кадра отсчетов"*). Т.е. на таких частотах управление величиной межкадровой задержки фактически отсутствует.
- При работе с платой на частоте АЦП выше 1000 кГц одновременное функционирование АЦП (сбор данных в FIFO буфер) и ЦАП (выдача данных из FIFO буфера) не возможна. Т.е. если на таких частотах при работающем АЦП подается команда разрешения вывода данных из FIFO буфера ЦАП с помощью функции ***ENABLE_DAC_STREAM_PLX()***, то функционирование АЦП прекращается, а выдача данных на ЦАП разрешается. И на оборот, если при разрешенном выводе данных на ЦАП подается команда разрешения работы АЦП с помощью функции ***ENABLE_ADC_PLX()***, то выдача данных на ЦАП прекращается, а функционирование АЦП (сбор данных в FIFO буфер) разрешается.

- При работе с платой на частоте АЦП выше 1000 кГц отсутствует возможность корректировки входных данных с АЦП с помощью калибровочных коэффициентов. Т.е. функция ***ENABLE_CORRECTION()*** (см. *§ 1.2.5.4.9 "Управление корректировкой данных с АЦП"*) на таких частотах работы АЦП не оказывает никакого влияния на работу платы.

2. НИЗКОУРОВНЕВОЕ ОПИСАНИЕ ПЛАТ СЕРИИ L-7xx

1.3. Общий PCI-интерфейс работы с платами серии L-7xx

Весь интерфейс плат серии *L-7xx* с компьютером можно осуществлять через традиционные порты ввода-вывода, память ниже1 Мб или память выше 1 Мб. Такое взаимодействие осуществляется через посредство высоко производительной шины PCI, всю работу с которой обеспечивает специализированные микросхемы *PCI9050-1* или *PCI9030* от фирмы *PLX Technology, Inc.* Подробную информацию об особенностях функционирования микросхем PCI-интерфейса можно найти на их сайте *www.plxtech.com*. Конкретная техническая реализация конфигурации адресов доступа к платам через порты и память имеет некоторое отличие плат *L-780* (ревизия 'A') от всех остальных плат серии *L-7xx*. Эти различия описаны в двух последующих параграфах.

1.3.1. Интерфейс с платами L-761, L-780 (Rev. 'B' и 'C') и L783

1.3.1.1. Интерфейс через порты ввода-вывода

Для взаимодействия с платой через традиционные порты ввода-вывода используется всего четыре адреса, конфигурация которых представлена в следующей таблице:

Адрес	Доступ	Назначение
BASE+0	Чтение/Запись	Чтение/запись данных по каналу IDMA
BASE+2	Запись	Запись адреса для канала IDMA
BASE+4	Запись	Инициирование прерывания IRQ2 в DSP
BASE+6	Запись	Сброс на плате локального триггера прерываний АЦП
BASE+8 (только для плат *Rev. 'C'*)	Запись	Сброс на плате от локального триггера прерываний ЦАП

где **BASE** – базовый адрес для доступа к плате через порты, значение которого можно найти, например, в поле **IO_BaseAddress** структуры *BOARD_INFO* после выполнения функции ***INIT_ACCESS_TO_PLX()*** (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*).

В платах *Rev. 'C'* предусмотрена дополнительная возможность сброса локальных тригеров прерываний. В качестве этого используются цифровые линии общего назначения микросхемы PLX PCI9030, а именно локальный регистр 0x54 – ***General Purpose I/O Control (*GPIOC**)**:

- ✓ флаг ***GPIO1*** – используется для сброса на плате локального триггера прерываний АЦП;
- ✓ флаг ***GPIO3*** – используется для сброса на плате локального триггера прерываний ЦАП.

Следует заметить, что со стороны платы весь обмен данными с PC осуществляется по так называемому каналу IDMA (Internal Direct Memory Access), по которому может работать установленный на плате DSP. Подробнее о работе канала IDMA можно прочитать, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 11 "DMA Ports", § 11.3 "IDMA Port", стр. 11-12, Analog Devices, Inc., Third Edition September 1995* или на сайте *www.analog.com*.

Т.о. для того, чтобы считать/записать **одно** слово в/из памяти DSP необходимо:

1. по адресу **BASE+2** записать адрес ячейки памяти DSP, с которой будет производиться операция обмена;
2. по адресу **BASE+0** осуществить либо чтение содержимого данной ячейки памяти, либо запись заданного значения.

Для обмена же **массивами** данных с памятью DSP необходимы следующие шаги:

1. по адресу **BASE+2** записать стартовый адрес ячейки памяти DSP, начиная с которой будет производиться операция обмена массивами данных;
2. по адресу **BASE+0** осуществить нужное количество операций чтения или записи самих данных, при этом, начиная с указанного в п.1 стартового адреса, будет последовательно считано или записано соответствующее количество смежных ячеек в/из памяти DSP.

Т.к. фирменный драйвер *LBIOS* работает по принципу команд, то для реализации данной возможности используется прерывание IRQ2 сигнального процессора. Сначала в соответствующую ячейку памяти данных DSP (предопределенная константа **L_COMMAND**, см. *§ 1.2.5.1.4 "Переменные LBIOS"*) заносится номер команды, которую драйвер должен выполнить. Затем с помощью операции записи по адресу **BASE+4** в DSP инициируется прерывание IRQ2, в ответ на которое, обработчик данного прерывания, содержащийся в самом *LBIOS,* выполняет соответствующие данной команде действия.

Платы серии *L-7xx* имеют возможность генерировать (по мере необходимости) прерывания в PC. Номер прерывания, назначенный каждой конкретной плате, можно найти, например, в поле **InterruptNumber** структуры *BOARD_INFO* после выполнения функции ***INIT_ACCESS_TO_PLX()*** (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*). При этом в Вашей программе, в обработчике прерываний от платы, **необходимо** сбрасывать так называемый локальный триггер прерываний, установленный на самой плате. Выполнение такой процедуры обеспечивается с помощью операции чтения по адресу **BASE+6**.

1.3.1.2. Интерфейс через память

Для взаимодействия с платой через память используется набор адресов, конфигурация которых представлена в следующей таблице:

Адрес	Доступ	Назначение
BASE+0	Чтение/Запись	Чтение/запись данных по каналу IDMA
BASE+2	Запись	Запись адреса для канала IDMA
BASE+4	Запись	Инициирование прерывания IRQ2 в DSP
BASE+6	Запись	Сброс на плате локального триггера прерываний АЦП
BASE+8 (только для плат *Rev. 'C'*)	Запись	Сброс на плате локального триггера прерываний ЦАП
Диапазон от **BASE+4096** до **BASE+8192**	Чтение/Запись	Область для быстрого чтения/записи массивов данных по каналу IDMA

где **BASE** – базовый адрес для доступа к плате через память, значение которого можно найти, например, в полях **LowMemorySpaceBaseAddress** или **HighMemorySpaceBaseAddress** структуры *BOARD_INFO* после выполнения функции ***INIT_ACCESS_TO_PLX()*** (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*).

В платах *Rev. 'C'* предусмотрена дополнительная возможность сброса локальных тригеров прерываний. В качестве этого используются цифровые линии общего назначения микросхемы **PLX PCI9030**, а именно локальный регистр 0x54 – ***General Purpose I/O Control (*GPIOC**):

- ✓ флаг ***GPIO1*** – используется для сброса на плате локального триггера прерываний АЦП;
- ✓ флаг ***GPIO3*** – используется для сброса на плате локального триггера прерываний ЦАП.

Следует заметить, что со стороны платы весь обмен данными с PC осуществляется по так называемому каналу IDMA (Internal Direct Memory Access), по которому может работать установленный на плате DSP. Подробнее о работе канала IDMA можно прочитать, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 11 "DMA Ports", § 11.3 "IDMA Port", стр. 11-12, Analog Devices, Inc., Third Edition September 1995* или на сайте *www.analog.com*.

Только при работе с платой через **память** микросхема PCI9050-1 или PCI9030 фирмы **PLX Technology, Inc.** обеспечивает при чтении массивов так называемую предварительную выборку запрашиваемых данных (prefetching), что увеличивает скорость обмена до 10 Мб/с (подробности можно найти на *www.plxtech.com*). Именно эта особенность используется для быстрого чтения массивов данных из памяти DSP.

Т.о. для того, чтобы считать/записать **одно** слово в/из памяти DSP необходимо:

1. по адресу **BASE+2** записать адрес ячейки памяти DSP, с которой будет производиться операция обмена;
2. по адресу **BASE+0** осуществить либо чтение содержимого данной ячейки памяти, либо запись заданного значения.

Для **быстрого** обмена **массивами** данных с памятью DSP необходимы следующие шаги:

1. по адресу **BASE+2** записать стартовый адрес ячейки памяти DSP, начиная с которой будет производиться операция обмена массивами данных;
2. начиная с адреса **BASE+4096** осуществить чтение/запись массива данных из/в память PC длиной не более 2048 16^{ти} битных слов, при этом, начиная с указанного в п.1 стартового адреса, будет последовательно считано/записано соответствующее количество смежных ячеек в/из памяти DSP.

С целью повышения эффективности работы при чтении/записи массивов в/из памяти PC нужно использовать ассемблерную конструкцию строкового чтения `REP MOVSD`. Например, для чтения 1024 слов из памяти данных DSP, начиная с адреса 0x2000, в буфер Buffer, находящийся в памяти PC через память ниже 1 Мб, можно использовать следующую ассемблерную вставку:

```
Buffer    dw  1024 dup(?) ; выделим память под Buffer
full_ptr  dd  0C0009000h  ; пусть плате выделена память с базовым
            ; адресом 0xC8000, тогда полный указатель на область
            ; быстрого чтения платы будет, например, 0xC0009000
idma_addr dd  0C0008002h  ; полный указатель на регистр записи
                          ; адреса для канала IDMA

. . . . .

les   di, idma_addr   ; загрузим в пару ES:DI указатель на регистр
                      ; платы для последующей записи стартового
                      ; адреса ячейки в памяти данных DSP
mov   word ptr es:[di], 02000h  ; загрузим сам стартовый адрес,
                                ; начиная с которого будем читать
                                ; массив из памяти данных DSP в
                                ; память PC
les   di, Buffer    ; загрузим в пару ES:DI указатель на Buffer
lds   si, full_ptr  ; загрузим в пару DS:SI указатель на область
                    ; быстрого чтения платы
mov   cx, 1024/2    ; загрузим счетчик для movsd
rep   movsd         ; выполним строковое чтение массива из 1024
                    ; слов
```

Т.к. фирменный драйвер *LBIOS* работает по принципу команд, то для реализации данной возможности используется прерывание IRQ2 сигнального процессора. Сначала в соответствующую ячейку памяти данных DSP (предопределенная константа **L_COMMAND**, см. *§ 1.2.5.1.4 "Переменные LBIOS"*) заносится номер команды, которую драйвер должен выполнить. Затем с помощью операции записи по адресу **BASE+4** в DSP инициируется прерывание IRQ2, в ответ на которое обработчик данного прерывания, содержащийся в самом *LBIOS,* выполняет соответствующие данной команде действия.

Платы серии *L-7xx* имеют возможность генерировать (по мере необходимости) прерывания в PC. Номер прерывания, назначенный каждой конкретной плате, можно найти, например, в поле **InterruptNumber** структуры *BOARD_INFO* после выполнения функции ***INIT_ACCESS_TO_PLX()*** (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*). При этом в Вашей программе, в обработчике прерываний от платы, **необходимо** сбрасывать так называемый локальный триггер прерываний, установленный на самой плате. Выполнение такой процедуры обеспечивается с помощью операции чтения по адресу **BASE+6**.

1.3.2. Интерфейс с платой L-780 (Rev. 'A')

Для взаимодействия с платой, как через порты, так и через память используется набор адресов, конфигурация которых представлена в следующей таблице:

Адрес	Доступ	Назначение
Диапазон от **BASE+0** до **BASE+4095**	Чтение/Запись	Чтение/запись (в том числе и быстрая) данных по каналу IDMA
BASE+4096	Запись	Запись адреса для канала IDMA
BASE+8192	Запись	Инициирование прерывания IRQ2 в DSP
BASE+12288	Запись	Сброс локального триггера прерываний в PC

где **BASE** – базовый адрес для доступа к плате, как через порты, так и через память. Значение этого базового адреса можно найти соответственно, например, в поле **IO_BaseAddress** или в полях **LowMemorySpaceBaseAddress** или **HighMemorySpaceBaseAddress** структуры *BOARD_INFO* после выполнения функции ***INIT_ACCESS_TO_PLX()*** (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*).

Следует заметить, что со стороны платы весь обмен данными с PC осуществляется по так называемому каналу IDMA (Internal Direct Memory Access), по которому может работать установленный на плате DSP. Подробнее о работе канала IDMA можно прочитать, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 11 "DMA Ports", § 11.3 "IDMA Port", стр. 11-12, Analog Devices, Inc., Third Edition September 1995* или на сайте *www.analog.com*.

Только при работе с платой через **память** микросхема PCI9050-1 или PCI9030 фирмы **PLX Technology, Inc.** обеспечивает при чтении массивов так называемую предварительную выборку запрашиваемых данных (prefetching), что увеличивает скорость обмена до 10 Мб/с (подробности можно найти на сайте *www.plxtech.com*). Именно эта особенность используется для быстрого чтения массивов данных из памяти DSP.

Для того, чтобы считать/записать **одно** слово в/из памяти DSP при работе с платой, как через порты, так и через память необходимо выполнить шаги аналогичные описанным, например, в *§ 2.1.1.1 "Интерфейс через порты ввода-вывода"* с учетом конфигурации базовых адресов, указанных в данном параграфе, а именно:

1. по адресу **BASE+4096** записать адрес ячейки памяти DSP, с которой будет производиться операция обмена;

2. по адресу **BASE+0** осуществить либо чтение содержимого данной ячейки памяти, либо запись заданного значения.

Для обмена **массивами** данных с памятью DSP при работе с платой через **порты** необходимо выполнить шаги аналогичные описанным в *§ 2.1.1.1 "Интерфейс через порты ввода-вывода"* с учетом конфигурации базовых адресов, указанных в данном параграфе, а именно:

1. по адресу **BASE+4096** записать стартовый адрес ячейки памяти DSP, начиная с которой будет производиться операция обмена массивами данных;
2. по адресу **BASE+0** осуществить нужное количество операций чтения или записи самих данных, при этом, начиная с указанного в п.1 стартового адреса, будет последовательно считано/записано соответствующее количество смежных ячеек в/из памяти DSP.

Для **быстрого** обмена **массивами** данных с памятью DSP при работе с платой через **память** необходимо выполнить шаги аналогичные описанным в *§ 2.1.1.2 "Интерфейс через память"* с учетом конфигурации базовых адресов, указанных в данном параграфе, а именно:

1. по адресу **BASE+4096** записать стартовый адрес ячейки памяти DSP, начиная с которой будет производиться операция обмена массивами данных;
2. начиная с адреса **BASE+0** осуществить чтение/запись массива данных из/в память PC длиной не более 2048 16^{ти} битных слов, при этом, начиная с указанного в п.1 стартового адреса, будет последовательно считано/записано соответствующее количество смежных ячеек в/из памяти DSP.

С целью повышения эффективности работы с платой через **память** при чтении/записи массивов в/из памяти PC нужно использовать ассемблерную конструкцию строкового чтения `REP MOVSD`. Например, для чтения 1024 слов из памяти данных DSP, начиная с адреса 0x2000, в буфер Buffer, находящийся в памяти PC через память ниже 1 Мб, можно использовать следующую ассемблерную вставку:

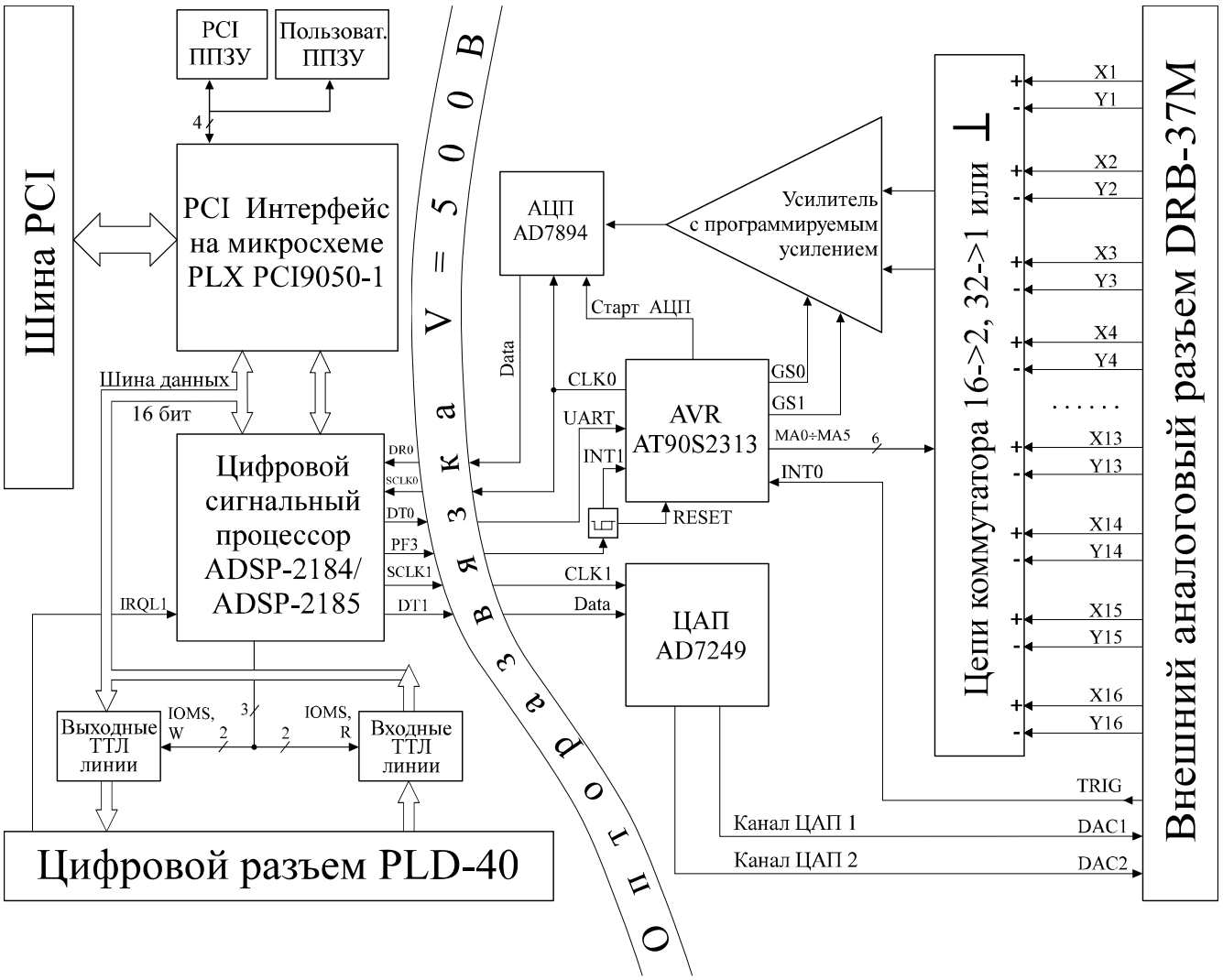
```
Buffer    dw  1024 dup(?) ; выделим память под Buffer
full_ptr  dd  0C0008000h  ; пусть плате выделена память с базовым
            ; адресом 0xC8000, тогда полный указатель на область
            ; быстрого чтения платы будет, например, 0xC0008000
idma_addr dd  0C0009000h  ; полный указатель на регистр записи
                          ; адреса для канала IDMA

. . . . .
les   di, idma_addr   ; загрузим в пару ES:DI указатель на регистр
                      ; платы для последующей записи стартового
                      ; адреса ячейки в памяти данных DSP
mov   word ptr es:[di], 02000h  ; загрузим сам стартовый адрес,
                                ; начиная с которого будем читать
                                ; массив из памяти данных DSP в
                                ; память PC
les   di, Buffer    ; загрузим в пару ES:DI указатель на Buffer
lds   si, full_ptr  ; загрузим в пару DS:SI указатель на область
                    ; быстрого чтения платы
mov   cx, 1024/2    ; загрузим счетчик для movsd
rep   movsd         ; выполним строковое чтение массива из 1024
                    ; слов
```

Т.к. фирменный драйвер *LBIOS* работает по принципу команд, то для реализации данной возможности используется прерывание IRQ2 сигнального процессора. Сначала в соответствующую ячейку памяти данных DSP (предопределенная константа **L_COMMAND**, см. *§ 1.2.5.1.4 "Переменные LBIOS"*) заносится номер команды, которую драйвер должен выполнить. Затем с помощью операции записи по адресу **BASE+8192** в DSP инициируется прерывание IRQ2, в ответ на которое, обработчик данного прерывания, содержащийся в самом *LBIOS,* выполняет соответствующие данной команде действия.

Платы серии *L-7xx* имеют возможность генерировать (по мере необходимости) прерывания в PC. Номер прерывания, назначенный каждой конкретной плате, можно найти, например, в поле **InterruptNumber** структуры *BOARD_INFO* после выполнения функции ***INIT_ACCESS_TO_PLX()*** (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*). При этом в Вашей программе, в обработчике прерываний от платы, **необходимо** сбрасывать так называемый локальный триггер прерываний, установленный на самой плате. Выполнение такой процедуры обеспечивается с помощью операции чтения по адресу **BASE+12288**.

1.4. Плата L-761

1.4.1. Структурная схема платы L-761



Как видно из приведенной выше структурной схемы, функционально плата *L-761* как бы разделена на две гальванически развязанные части. На опторазвязанной с компьютером стороне находятся микроконтроллер AVR 90S2313, который обеспечивает надлежащее функционирование последовательного АЦП AD7894 и цепей коммутации входных сигналов, и ЦАП AD7294, управляемый непосредственно DSP. В ППЗУ микроконтроллера при наладке на *ЗАО «Л-Кард»* зашивается соответствующая программа, отвечающая за его корректную работу. На другой, гальванически не развязанной с компьютером, стороне расположены микросхема *PCI9050-1*, полностью обеспечивающая PCI-интерфейс платы с PC, цифровой сигнальный процессор (DSP), фактически управляющий всей периферией на плате, а также разъём для внешних цифровых линий. Итак, на данной плате установлен современный высокопроизводительный сигнальный процессор фирмы **Analog Devices, Inc.** *ADSP-2184* (возможна замена на *ADSP-2185*), работающий на частоте $2*f_q$, где f_q=14745.6 кГц – частота кварца. Внутренняя архитектура процессора оптимизирована для реализации таких алгоритмов, как цифровая фильтрация, спектральный анализ и т.д. Сам процессор имеет внутреннюю память программ на 4 КСлов и внутреннюю память данных 4 КСлов (процессор ADSP-2185 обладает 16 КСлов памяти программ и 16 КСлов памяти данных). Наличие такого мощного сигнального процессора обеспечивает Вам, при необходимости, возможность самостоятельного применения чрезвычайно гибких методов управления всей периферией платы и позволяет переносить часть операций по обработке данных на саму плату (например, Быстрое Преобразование Фурье). Процессор обладает своим собственным контроллером ПДП (канал IDMA: Internal Direct Memory Access; подробнее о работе IDMA см. *"ADSP-2100 Family User's Manual*

(Includes ADSP-2171, ADSP-2181)", § 11.3 "IDMA Port", Analog Devices, Inc., Third Edition September 1995 или на сайте www.analog.com) для доступа к любой ячейки внутренней памяти. Благодаря этому Ваша программа может обращаться к любой ячейки памяти процессора, не прерывая работы самого DSP, что исключительно удобно при построении алгоритмов, работающих в реальном масштабе времени. Максимальная пропускная способность обмена данными между сигнальным процессором и компьютером составляет приблизительно 10 Мб/с. Весь обмен данными с центральным компьютером DSP осуществляет через свой канал IDMA. Протокол работы с каналом IDMA предусматривает также возможность загрузки в сигнальный процессор управляющей программы (драйвера), которая будет осуществлять требуемые алгоритмы ввода-вывода (процедуру загрузки см. *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 11.3.5 "Boot Loading Through the IDMA Port", стр. 11-24, Analog Devices, Inc., Third Edition September 1995*). Фирменный драйвер *LBIOS* работает по принципу команд и для реализации такой возможности используется прерывание *IRQ2* сигнального процессора. Сначала в соответствующую ячейку памяти данных DSP (предопределенная константа **L_COMMAND**, см. *§ 1.2.5.1.4 "Переменные LBIOS"*) заносится номер команды, которую драйвер должен выполнить. Затем инициируется прерывание *IRQ2*, в ответ на которое, обработчик данного прерывания, содержащийся в самом *LBIOS,* выполняет соответствующие данной команде действия. DSP осуществляет получение данных с АЦП и управление работой микроконтроллера AVR с помощью одного из своих последовательных (сериальных) портов (SPORT0). Микроконтроллер AVR, получающий параметры своего функционирования от DSP по каналу UART, управляет цепями коммутатора входных сигналов, коэффициентом усиления программируемого усилителя, частотой запуска АЦП и, при необходимости, синхронизацией ввода данных по линии **TRIG** внешнего разъёма ***DRB-37M***. Сигнальный процессор обеспечивает также взаимодействие с микросхемой двухканального ЦАП через посредство другого своего последовательного порта (SPORT1). Управление внешними цифровыми линиями на разъёме ***PLD-40*** осуществляется DSP с помощью чтения/записи своего пространства ввода/вывода (I/O Memory Space). Описание этого пространства можно найти, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 10.6.4 "ADSP-2181 I/O Memory Space", стр. 10-32, Analog Devices, Inc., Third Edition September 1995*.

1.4.2. Создание управляющей программы

У пользователя, как правило, не появляется необходимость в написаниии своих собственных управляющих программ для данной платы, т.к. все наиболее часто требуемые алгоритмы работы уже реализованы в фирменном драйвере, находящимся в файле `L761.bio`. Однако если же у Вас все-таки возникла необходимость в создании собственной управляющей программы (например, для формирования какого-либо специализированного алгоритма действия процессора), то для этого придется освоить достаточно несложный язык ассемблера для сигнального процессора. В качестве законченного примера программирования платы на таком языке можно использовать исходные тексты фирменного драйвера, хранящиеся в файлах `DSP\L761\L761.DSP` и `DSP\L761\*.H`. Эти исходные тексты достаточно подробно прокомментированы.

Процесс формирования собственной управляющей программы для DSP потребует от Вас приложения определенных усилий:

1. Вам необходимо будет изучить несложную архитектуру процессора ADSP-218x, а также освоить довольно простой язык его программирования (ассемблер для DSP). Всю подробную информацию об этом можно найти в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Analog Devices, Inc., Third Edition, September 1995* или в русском переводе *"Руководство пользователя по сигнальным микропроцессорам семейства ADSP-2100", под редакцией А.Д.Викторова, Санкт-Петербург, 1997*. Оба эти издания можно приобрести в *ЗАО «Л-Кард»*. Описание и примеры программ для DSP с исходными текстами приводятся в двухтомном справочнике *"Digital Signal Processing Applications Using the ADSP-2100 Family", Analog Devices, Inc.*, который можно найти у официальных российских дистрибьюторов компании ***Analog Devices, Inc.*** (например, фир-

мы ***Autex Ltd.*** или ***Argussoft Co.***). Много полезного в дополнение к указанной документации можно обнаружить также на сайте *www.analog.com*.

2. Процессоры семейства ADSP-218x поддерживаются полным набором программных средств отладки. Этот набор включает в себя несколько программ: построитель системы (`bld21.exe`), ассемблер (`asm21.exe`), линкер или редактор связей (`ld21.exe`) и т.д. Все эти программы очень подробно описываются в оригинальной книге *"ADSP-2100 Family Assembler Tools & Simulator Manual", Analog Devices, Inc., Second Edition, November 1994*, которую можно найти у официальных российских дистрибьюторов компании ***Analog Devices, Inc.*** (например, фирмы ***Autex Ltd.*** или ***Argussoft Co.***). Сам пакет разработчика программ для сигнальных процессоров семейства ADSP-21xx, содержащий все выше указанные средства отладки (кроме `bld21.exe`), можно приобрести в *ЗАО «Л-Кард»*. В качестве архитектурного файла нужно использовать `L761.ACH`.
3. Итак, если у Вас не возникло проблем с первыми двумя этапами, то теперь можно приступать к собственно написанию Вашей управляющей программы. Для начала надо создать соответствующие файлы с исходным кодами Вашей программы на языке ассемблер DSP. Затем эти файлы необходимо оттранслировать (`asm21.exe`) и скомпоновать с помощью редактора связей (`ld21.exe`), формируя, таким образом, выполняемую программу типа `.EXE`, так называемый файл отображения в памяти (memory image file). **!!! Не путать этот файл с обычной выполняемой DOS программой типа `.EXE`!!!** Формат сформированного файла отображения в памяти очень подробно описан в *"ADSP-2100 Family Assembler Tools & Simulator Manual", Appendix **B** "File Format", **B.2** "Memory Image File (.EXE)", Analog Devices, Inc., Second Edition, November 1994*. Именно в этом файле содержатся все коды инструкций Вашей программы с соответствующими адресами их расположения в памяти программ DSP, а также инициализирующие значения Ваших переменных и адреса их нахождения в памяти данных. Зная всю эту информацию нужно просто аккуратно загрузить ее в память DSP по надлежащим адресам. Для упрощения процедуры загрузки полученный файл отображения в память `.EXE` преобразуется с помощью утилиты `DSP\L761\BIN3PCI.EXE` в файл `.BIO` (подробности см. *приложение D*).

Вообще-то, всю эту последовательность создания файла `.BIO` можно проследить по содержанию файлу `DSP\L761\L761.BAT`. Созданный таким образом файл с управляющей программой `.BIO` затем используется, например, в штатной функции загрузки сигнального процессора ***LOAD_LBIOS_PLX()*** (см. *§ 1.2.5.2.2 "Загрузка LBIOS"*).

1.4.3. Загрузка управляющей программы в DSP

Перед началом работы с платой необходимо ее "оживить", т.е. загрузить в DSP либо фирменный драйвер, находящийся в файле `L761.bio`, либо Вашу собственную управляющую программу. **Только** после выполнения такой процедуры плата будет корректно работать с функциями штатной или Вашей (если создадите) библиотеки. Формат файла `L761.bio`, содержащий все коды инструкций управляющей программы, подробно описан в *приложении D*. Коды инструкций сигнального процессора из этого файла необходимо расположить по соответствующим адресам памяти программ, также как данные, инициализирующие переменные драйвера, в памяти данных и запустить управляющую программу на выполнение. Все это и называется загрузка управляющей программы в DSP.

Сама процедура начальной загрузки управляющей программы во внутреннюю память DSP достаточно проста и хорошо описана в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 11.3.5 "Boot Loading Through the IDMA Port", стр. 11-24, Analog Devices, Inc., Third Edition September 1995*. Она предполагает выполнение следующих несложных шагов:

- произведите сброс (RESET) DSP на данной плате, например, с помощью API функции ***PLATA_RESET_PLX()*** (см. *§ 1.2.5.2.6 "Сброс DSP на плате"*);
- заполните необходимыми данными по каналу IDMA память данных и программ, кроме ячейки с адресом PM(0x0000), для чего можно воспользоваться соответствующими функциями

PUT_DM_ARRAY_PLX() (см. *§ 1.2.5.3.7 "Запись массива слов в память данных DSP"*), ***PUT_PM_WORD_PLX()*** (см. *§ 1.2.5.3.8 "Запись массива слов в память программ DSP"*) и т.д.

– запишите данные в ячейку памяти программ по адресу PM(0x0000) для старта Вашей управляющей программы (при этом ее выполнение начнется с инструкции, находящейся в PM(0x0000)).

Внимание!!! Необходимо обязательно убедиться, что Вы загрузили всю нужную информацию в память DSP **до** записи данных по адресу PM(0x0000).

В общем-то **ВСЕ!** Теперь можно спокойно приступать к управлению платой с помощью функций штатной или Вашей библиотеки.

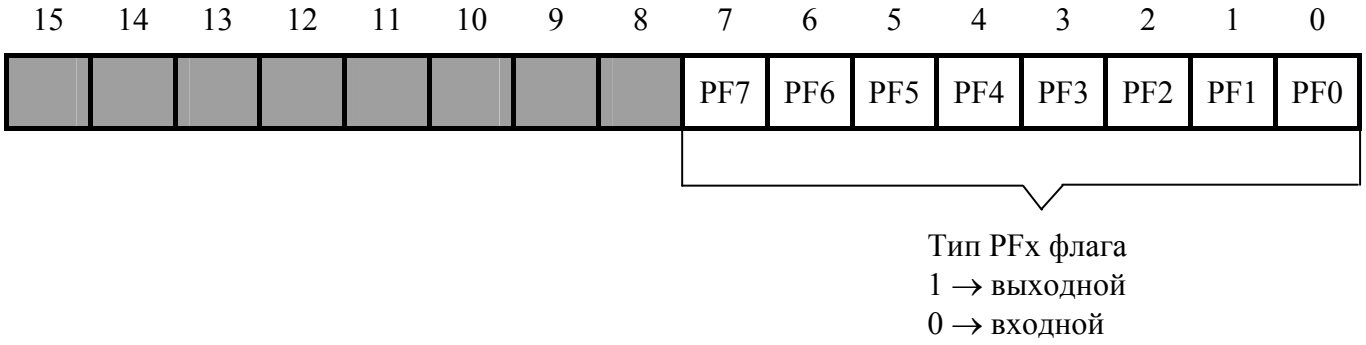
1.4.4. Установка флагов PFx сигнального процессора

Микросхемы сигнальных процессоров ADSP-2184/2185/2186 имеют восемь дополнительных неспециализированных выводов, так называемых флагов, обозначаемых обычно **PF0÷PF7**. Данные флаги могут быть индивидуально запрограммированы как на вход, так и на выход. На указанных типах процессоров выводы этих флагов обладают также и дополнительными функциями. Так флаг **PF6** совмещен с входом прерывания *IRQL1*, а флаг **PF7** - с *IRQ2* и т.д. (подробности можно найти в техническом описании этих DSP на сайте www.analog.com). Т.о. флаги **PFx** необходимо запрограммировать на вход-выход надлежащим образом, т.е. флаги **PF0÷PF2**, **PF6**, **PF7** как входные, а флаги **PF3**, **PF4** и **PF5** как выходные. Программирование флагов осуществляется с помощью двух управляющих регистров DSP:

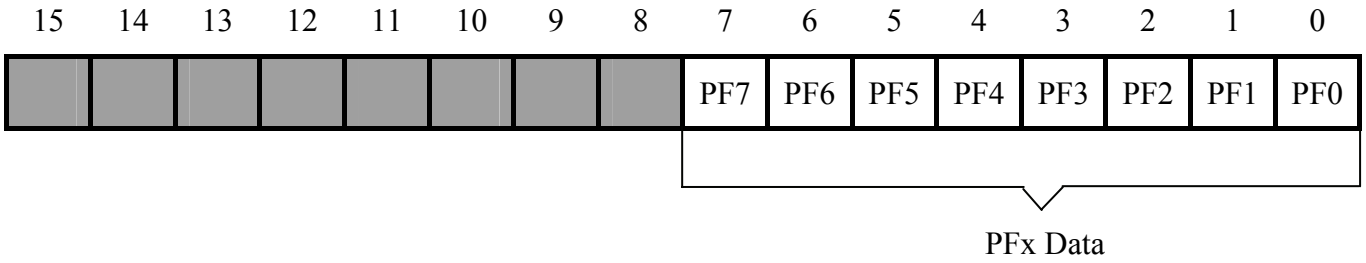
- ✓ регистр Programmable Flag & Composite Select Control, находящий по адресу DM(0x3FE6) в памяти данных, управляет направлением флага
  - – 1 – выход,
  - – 0 – вход;
- ✓ регистр Programmable Flag Data, расположенный по адресу DM(0x3FE5), используется для записи и считывания значений флагов.

Формат данных регистров приведен ниже и подробно описан в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 9.5 "Flag Pins", cmp. 9-15, Analog Devices, Inc., Third Edition, September 1995.*

Programmable Flag & Composite Select Control (DM(0x3FE6))



Programmable Flag Data (DM(0x3FE5))



Следовательно, для надлежащего программирования флагов PFx необходимо написать следующий код:

```
      const Prog_Flag_Comp_Sel_Ctrl = 0x3FE6;
      . . . . .
{сделаем флаги PFx: PF0÷PF2, PF6, PF7 - входные, остальные - выходные}
      AR=0x38;
      DM(Prog_Flag_Comp_Sel_Ctrl)=AR;
```

1.4.5. Микроконтроллер AVR

На плате *L-761* микроконтроллер **AVR AT93S2313**, находящийся на гальванически развязанной с компьютером стороне (см. *§ 2.2.1 "Структурная схема платы L-761"*), в соответствии с информацией, получаемой от DSP, задает режим работы и временные диаграммы АЦП, а также управляет цепями входного коммутатора и программируемого усилителя (PGA). Полное техническое описание на микроконтроллер можно найти на сайте *www.atmel.com*. Два порта данного микроконтроллера сконфигурированы следующим образом:

Порт B

Вывод	Назначение
PB0	**MA0** – 0ой бит адреса канала
PB1	**MA1** – 1ый бит адреса канала
PB2	**MA2** – 2ой бит адреса канала
PB3	**MA3** – 3ий бит адреса канала
PB4	**MA4** – заземление входа программируемого усилителя/ 4ый бит номера канала
PB5	**MA5** – 16 диф./32 общ. режим подключения входных каналов
PB6	**GS0** – 0ой бит коэффициента усиления
PB7	**GS1** – 1ый бит коэффициента усиления

смысл назначений данного порта подробно описан в *§ 1.2.2.2.3 "Логический номер канала АЦП"*.

Порт D

Вывод	Назначение
PD0	Вход данных UART
PD1	Не используется
PD2	Прерывание INT0 используется для внешней синхронизации АЦП (отрицательный импульс ⎍ длительностью не менее

	200 нс)
PD3	Прерывание INT1 управляется от DSP (флаг **PF3**)
PD4	-CONV – запуск АЦП
PD5	Не используется
PD6	-FRAME используется при передаче данных из АЦП в DSP

В ППЗУ микроконтроллера на *ЗАО «Л-Кард»* зашивается специализирована программа, которая через выше указанные порты осуществляет управление АЦП, коммутатором и PGA. Основные параметры этого управления AVR получает по UART (1 стартовый бит, 9 бит данных, 2 стоповых бита; скорость 625 кГц) из DSP, использующий для передачи последовательный порт SPORT0. Посылки по UART бывает двух типов: адрес и данные. Так при посылке адресного типа $9^{\text{ый}}$ бит передаваемых данных должен быть установлен в ‘1’, иначе в ‘0’. Работа с AVR по UART предполагает выполнение следующих шагов:

- сначала в AVR необходимо передать адрес, т.е. номер ячейки памяти микроконтроллера;
- теперь можно осуществлять посылки самих данных, которые будут последовательно располагаться в ячейках памяти, начиная с уже переданного адреса.

При получении посылки любого типа AVR прерывает выполнение своей основной программы, фактически останавливая работу АЦП. После того, как все необходимые данные (параметры) приняты, основную программу AVR, а ,следовательно, и АЦП, необходимо заново запустить в обычном режиме сбора данных. Для этого надо вызвать прерывание *INT1*, сформировав короткий отрицательный импульс (⎍ менее 1 мкс) на выходе флага **PF3** сигнального процессора. Можно осуществить и полный сброс (RESET) микроконтроллера, когда все параметры управления в программе AVR устанавливаются по умолчанию. Делается это также с помощью флага **PF3**, но формируемый отрицательный импульс (⎍) на его выводе должен иметь длительность не менее 10 мкс. Сама процедура сброса AVR достаточно длительная (не менее 0.2 с) и не рекомендуется ничего посылать в AVR, пока она не завершится.

В памяти микроконтроллера расположены следующие переменные, значения которых задают все доступные режимы работы АЦП, коммутатора и PGA:

Адрес (Hex)	Обозначение	Назначение
0x5	adsynchro	логический номер канала для аналоговой синхронизации
0x6	rate_low	младший байт $16^{\text{ти}}$ битного значения межканальной задержки соответствующего $t_{\text{мкз}}$=Inter_Kadr_Delay (само $16^{\text{ти}}$ битное значение межканальной задержки задается в единицах 0.1 мкс)
0x7	rate_high	старший байт $16^{\text{ти}}$ битного значения межканальной задержки соответствующего $t_{\text{мкз}}$=Inter_Kadr_Delay (само $16^{\text{ти}}$ битное значение межканальной задержки задается в единицах 0.1 мкс)
0x8	kadr_low	младший байт $16^{\text{ти}}$ битного значения интервала запуска АЦП соответствующего $t_{\text{АЦП}}$=1/ADC_Rate (само $16^{\text{ти}}$ битное значение интервала запуска АЦП задается в единицах 0.1 мкс)
0x9	kadr_high	старший байт $16^{\text{ти}}$ битного значения интервала запуска АЦП соответствующего $t_{\text{АЦП}}$=1/ADC_Rate (само $16^{\text{ти}}$ битное значение интер-

		вала запуска АЦП задается в единицах 0.1 мкс)
0xA	topaddr	Адрес ячейки с последним байтом управляющей таблицы + 1, т.е. topaddr=0x60+**Control_Table_Length**
0xB	mode	Установленные в ‘1’ соответствующие биты данного параметра определяют режимы синхронизации работы АЦП: ✓ $5^{ый}$ бит задает режим аналоговой синхронизации ввода данных с АЦП по заданному в ячейке 0x5 логическому каналу; ✓ $6^{ый}$ бит задает режим ждущей по-кадровой синхронизации; ✓ $7^{ый}$ бит задает режим ждущей синхронизации старта АЦП.
0x60÷0xC0	control_table	96 байтовая управляющая таблица , каждый элемент которой содержит значение для последовательного управления портом B (фактически это **Control_Table**, переданная в AVR)

Смысл $\mathbf{t_{АЦП}}$, **ADC_Rate,** $\mathbf{t_{МКЗ}}$, **Inter_Kadr_Delay**, **Control_Table** и **Control_Table_Length** подробно описан, например, в *§ 1.2.2.4. "Формат кадра отсчетов"*.

При начале работы с AVR первым делом необходимо произвести его сброс. **Только** после завершения сей процедуры можно загрузить в AVR управляющую таблицу и адрес ее последнего байта. При желании можно задать также частоту работы АЦП и межкадровую задержку (эти два параметра определяются в единицах 0.1 мкс.). Теперь надо запустить, приостановленную записью параметров, основную программу AVR в обычном режиме сбора данных с АЦП, сгенерировав для этого прерывание *INT1*. Все! Остается только ждать когда АЦП начнет передавать данные в последовательном виде (их должен принимать DSP с помощью своего сериального порта SPORT0). Первым поступят данные с аналогового канала, заданного первым элементом управляющей таблицы, следующим– заданный вторым элементом и т.д. циклически повторяясь.

В режим синхронизации ввода с АЦП основную программу AVR можно перевести, записав в ячейку с адресом 0xB надлежащее значение. При ждущей синхронизации старта АЦП, данные с него начнут поступать только после прихода прерывания *INT0* с внешнего разъёма. В случае покадровой синхронизации данные **одного** кадра с АЦП поступят только после прихода прерывания *INT0*. Перед переходом в режим аналоговой синхронизации Вам необходимо прежде прописать по адресу 0x5 логический номер канала синхронизации (синхроканал). После перехода в этот режим с АЦП начнут поступать данные только с заданного синхроканала. Эти данные теперь можно анализировать на предмет выполнения каких-то заданных Вами критериев синхронизации. После реализации условий таких критериев можно запустить основную программу AVR в обычном режиме работы сбора данных, сгенерировав прерывание *INT1*.

Для выхода из режима синхронизации не дождавшись момента его наступления, Вы должны прописать в ячейке с адресом 0xB значение равное 0x0, с последующим генерированием прерывания *INT1*.

1.4.6. Передача параметров из DSP в AVR

Передачу параметров в AVR сигнальный процессор осуществляет с помощью своего сериального порта SPORT0. Для этого необходимо его запрограммировать надлежащим образом, а именно:

- ✓ длина слова (word length) – 12 бит;
- ✓ тактовые синхроимпульсы (serial clocks) –внутренние с частотой 614.4 кГц;
- ✓ кадровая синхронизация передачи каждого слова;
- ✓ внутренняя кадровая синхронизация передачи;
- ✓ альтернативная кадровая синхронизация передачи;
- ✓ кадровый сигнал передачи активен по низкому уровню.

Очень подробное описание конфигурирования сериальных портов DSP можно найти в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 5 "Serial Ports", стр. 5-1, Analog Devices, Inc., Third Edition, September 1995.*

Для посылки параметра в AVR его надо соответствующим образом преобразовать, с учетом обязательного наличия одного стартового и двух стоповых битов. После чего передаваемое значение можно записать в регистр передачи TX0 и порт SPORT0 сам осуществит эту посылку.

Приведем пример подготовки порта SPORT0 для передачи параметров в AVR:

```
{ Для начала отконфигурируем SPORT0 на передачу                    }
{ SPORT0 - disable, SPORT1 - enable, SPORT1 - serial port          }
    AR = 0x0C00;              { 0000 1100 0000 0000                }
    DM(Sys_Ctrl_Reg)=AR;      { 0x3FFF - System Control Register   }

{ Serial Clock Divide Modulus                                      }
    AR = 23;                  { SCLK0 - internal                   }
    DM(Sport0_Sclkdiv) = AR;  { 0x3FF5 - Serial Clock Divide Modulus }

{ Receive Frame Sync Divide Modulus                                }
    AR = 1000;      { may be any number: receive frame is external   }
    DM(Sport0_Rfsdiv) = AR; {0x3FF4-Receive Frame Sync Divide Modulus}

{ Control word for SPORT0: SCLK0 - internal                        }
{ low level, alternate external receive frame on each word(12 bit) - not used     }
{ low level, alternate internal transmit frame on each word(12 bit)  }
    AR = 0x7ECB;              { 0111 1110 1100 1011                }
    DM(Sport0_Ctrl_Reg) = AR; { 0x3FF6 - SPORT0 Control Register   }

{ SPORT0 enable, SPORT1 enable, SPORT1 - serial port               }
    AR = 0x1C00;              { 0001 1100 0000 0000                }
    DM(Sys_Ctrl_Reg) = AR;    { 0x3FFF - System Control Register   }

{ Все! Теперь нужно подготовить данные для передачи их в формате UART}
{ и загрузить их в регистр передачи TX0. Эти данные портом SPORT0  }
{ будут переданы в AVR самостоятельно                              }
```

1.4.7. АЦП

Прием данных, поступающих в последовательном виде с АЦП **AD7894** (техническое описание можно найти на сайте www.analog.com), сигнальный процессор осуществляет через посредство своего сериального порта SPORT0. Для этого необходимо его запрограммировать надлежащим образом, а именно:

- ✓ длина слова (word length) – 14 бит;
- ✓ тактовые синхроимпульсы (serial clocks) – внешние с периодом 0.1 мкс;
- ✓ кадровая синхронизация приема каждого слова;
- ✓ внешняя кадровая синхронизация приема;
- ✓ нормальная кадровая синхронизация приема;
- ✓ кадровый сигнал приема активен по низкому уровню.

Очень подробное описание конфигурирования сериальных портов DSP можно найти в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 5 "Serial Ports", стр. 5-1, Analog Devices, Inc., Third Edition, September 1995.*

После приема одного целого слова, сериальный порт, запрограммированный, как указано выше, генерирует прерывание приема SPORT0 Receive. Теперь в регистре приема этого порта RX0 находится значение с АЦП. В фирменном драйвере `L761.bio` прием данных портом SPORT0 осуществляется в режиме автобуферизации (AutoBuffering). Детали автобуферизации описаны в*"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 5.11 "AutoBuffering", стр. 5-26, Analog Devices, Inc., Third Edition, September 1995.*

Попробуем проиллюстрировать сказанное примером:

```
{ Для начала откнфигурируем SPORT0 на прием                          }
{ SPORT0 - disable, SPORT1 - disable, SPORT1 - serial port           }
    AR=0x0400;
    DM(Sys_Ctrl_Reg)=AR;      { 0x3FFF - System Control Register     }

{ ********************************************************************* }
{ * Set SPORT0 for receive of ADC samples                           * }
{ * SCLK0 and Receive Frame - external, word = 14 bits              * }

{ Serial Clock Divide Modulus                                        }
    AR = 9;                   { may be any: SCLK0 - external         }
    DM(Sport0_Sclkdiv) = AR;  { 0x3FF5 - Serial Clock Divide Modulus }

{ Receive Frame Sync Divide Modulus                                  }
    AR = 1000;      { may be any number: receive frame is external   }
    DM(Sport0_Rfsdiv) = AR; {0x3FF4-Receive Frame Sync Divide Modulus}

{ Control word for SPORT0: SCLK0 - external                          }
{ low level, normal external receive frame on each word(14 bit)      }
{ low level, alternate internal transmit frame on each word(14 bit) - not used    }
    AR = 0x2EDD;              { 0010 1110 1101 1101                  }
    DM(Sport0_Ctrl_Reg) = AR; { 0x3FF6 - SPORT0 Control Register     }
{ ********************************************************************* }

{ SPORT0 - enable, SPORT1 - enable, SPORT1 - serial port             }
    AR = 0x1C00;              { 0001 1100 0000 0000                  }
    DM(Sys_Ctrl_Reg) = AR;    { 0x3FFF - System Control Register     }

{ Все! Теперь можно ждать прихода прерывания SPORT0 Receive          }
{ и при его появлении в регистре приема RX0 будет отсчет с АЦП       }
```

1.4.8. ЦАП

Управление микросхемой двухканального 12[ти] битного ЦАП **AD7249** (техническое описание можно найти на сайте www.analog.com), которая может быть установлена на плате по Вашему желанию, сигнальный процессор осуществляет с помощью последовательного порта SPORT1. Для этого необходимо его запрограммировать надлежащим образом, а именно:

- ✓ длина слова (word length) – 16 бит;
- ✓ тактовые синхроимпульсы (serial clocks) – внутренние с периодом не менее 0.4 мкс;
- ✓ кадровая синхронизация приема каждого слова;
- ✓ внутренняя кадровая синхронизация приема;
- ✓ альтернативная кадровая синхронизация приема;
- ✓ кадровый сигнал приема активен по высокому уровню;
- ✓ кадровая синхронизация передачи каждого слова;
- ✓ внешняя кадровая синхронизация передачи;
- ✓ альтернативная кадровая синхронизация передачи;
- ✓ кадровый сигнал передачи активен по высокому уровню.

Выводы сигнального процессора для кадровой синхронизации приема и передачи (выводы RFS1 и TFS1 соответственно) объединены, т.е. внутренне генерируемые сигналы кадровой синхронизации приема являются сигналами внешней кадровой синхронизации передачи. Частоту генерирования сигналов на выводе RFS1 можно изменять в достаточно широких пределах, устанавливая таким образом частоту вывода данных на ЦАП. Очень подробное описание конфигурирования сериальных портов DSP можно найти в *“ADSP-2100 Family User’s Manual (Includes ADSP-2171, ADSP-2181)”, Chapter 5 “Serial Ports”, стр. 5-1, Analog Devices, Inc., Third Edition, September 1995*

Формат передаваемых в микросхему ЦАП данных приведен в следующей таблице:

Номер бита	Назначение
0÷11	$12^{\text{ти}}$ битный код ЦАП
12	Выбирает номер ЦАП: ✓ если ‘0’, то запись в первый канал ЦАП; ✓ если ‘1’, то запись во второй канал ЦАП.
13÷15	Не используются

Приведем пример подготовки порта SPORT1 для управления ЦАП’ами:

```
{ Для начала отконфигурируем SPORT1 на передачу данных                }
{ SPORT0 - disable, SPORT1 - disable, SPORT1 - serial port            }
    AR=0x0400;
    DM(Sys_Ctrl_Reg)=AR;      { 0x3FFF - System Control Register      }

{ ************************************************************** }
{   Set SPORT1 for transmit digital codes in DAC                      }
{   SCLK1 and Receive Frame - internal, word = 16 bits                }
{   Transmit Frame - external                                         }

{   Serial Clock Divide Modulus                                       }
    AR=5;                     { SCLK1 – internal, T=406.9 ns          }
    DM(Sport1_Sclkdiv) = AR;   { 0x3FF1 - Serial Clock Divide Modulus }

{Receive Frame Sync Divide Modulus                                    }
    AR = 49;            { Определяет частоту вывода отсчетов с ЦАП'а  }
    DM(Sport1_Rfsdiv)=AR;   {0x3FF0-Receive Frame Sync Divide Modulus }

{ Control word for SPORT1: SCLK1 - internal                           }
{ high level, alternate internal receive frame on each word(16 bit)   }
{ high level, alternate external transmit frame on each word(16 bit) }
    AR = 0x7D0F;              { 0111 1101 0000 1111                   }
    DM(Sport1_Ctrl_Reg) = AR; { 0x3FF2 - SPORT1 Control Register      }
{ ************************************************************** }

{ SPORT0 - enable, SPORT1 - enable, SPORT1 - serial port              }
    AR = 0x1C00;              { 0001 1100 0000 0000                   }
    DM(Sys_Ctrl_Reg) = AR;    { 0x3FFF - System Control Register      }

{ Мы установили частоту вывода данных на ЦАП равной                   }
{ fq/((5+1)*(49+1))=49.512 Кгц, где fq=14745.6 кГц - частота кварца   }
{ Теперь с этой частотой будут приходить прерывания SPORT1 Transmit, }
{ обработчик, которого каждый раз должен записывать в регистр        }
{ передачи TX1 очередное значение, посылаемое в микросхему ЦАП       }

{ Мы же сейчас просто установим нулевой уровень на первом ЦАП’е       }
    TX1=0x0;

    . . . . .

{ А сейчас установим уровень 5.0 В (код 2047) на втором ЦАП’е         }
    TX1=0x17FF;
```

1.4.9. Управление ТТЛ линиями

Управление цифровыми линиями осуществлять достаточно просто. Любая запись какого-либо числа в пространство ввода-вывода (I/O Memory Space) сигнального процессора устанавливает состояния всех 16[ти] выходных линий на разъёме ***PLD-40*** в соответствие с битами записываемого значения. Любое чтение из пространства ввода-вывода (I/O Memory Space) сигнального процессора считывает состояние всех 16[ти] входных линий на разъёме ***PLD-40*** платы. Описание разъёма см. *"Руководство пользователя", § 3.3.2. "Внутренний разъём для подключения цифровых сигналов"*. Описание I/O Memory Space можно найти, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 10.6.4 "ADSP-2181 I/O Memory Space", стр. 10-32, Analog Devices, Inc., Third Edition September 1995*.

Рассмотрим простенький пример работы с цифровыми линиями:

```
{ считаем входные цифровые линии                                    }
    AR=IO(0x0);
{ установим прочитанные состояния на выходных линиях                }
    IO(0x0)=AR;
```

1.4.10. Генерирование прерываний в PC

На плате предусмотрена возможность генерирования прерываний в PC. Эту способность можно использовать, когда Вам требуется проинформировать компьютер о наступлении какого-либо события. Например, в фирменном драйвере генерирование прерывания в PC применяется по мере соответствующего заполнения FIFO буфера АЦП, информируя Вас о готовности данных. Номер прерывания для каждой платы назначается BIOS компьютера или WINDOWS и его можно узнать, прочитав, например, поле **InterruptNumber** в соответствующей структуре типа *BOARD_INFO* (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*).

Для генерации прерывания используется флаг сигнального процессора **FL2**. Так при работе с прерываниями необходимо, чтобы флаг **FL2** постоянно находился в состоянии '1', а переход '1'-'0'-'1' осуществлялся только тогда, когда надо вызвать прерывание в компьютере. **Внимание!!!** Прерывание в компьютере будет сгенерировано только в случае, если со стороны PC оно будет разрешено с помощью API функции ***INIT_INTERRUPT_PLX()*** (см. *§ 1.2.5.6.1 "Установка обработчика прерываний"*).

Приведем пример генерирования одного прерывания в компьютер:

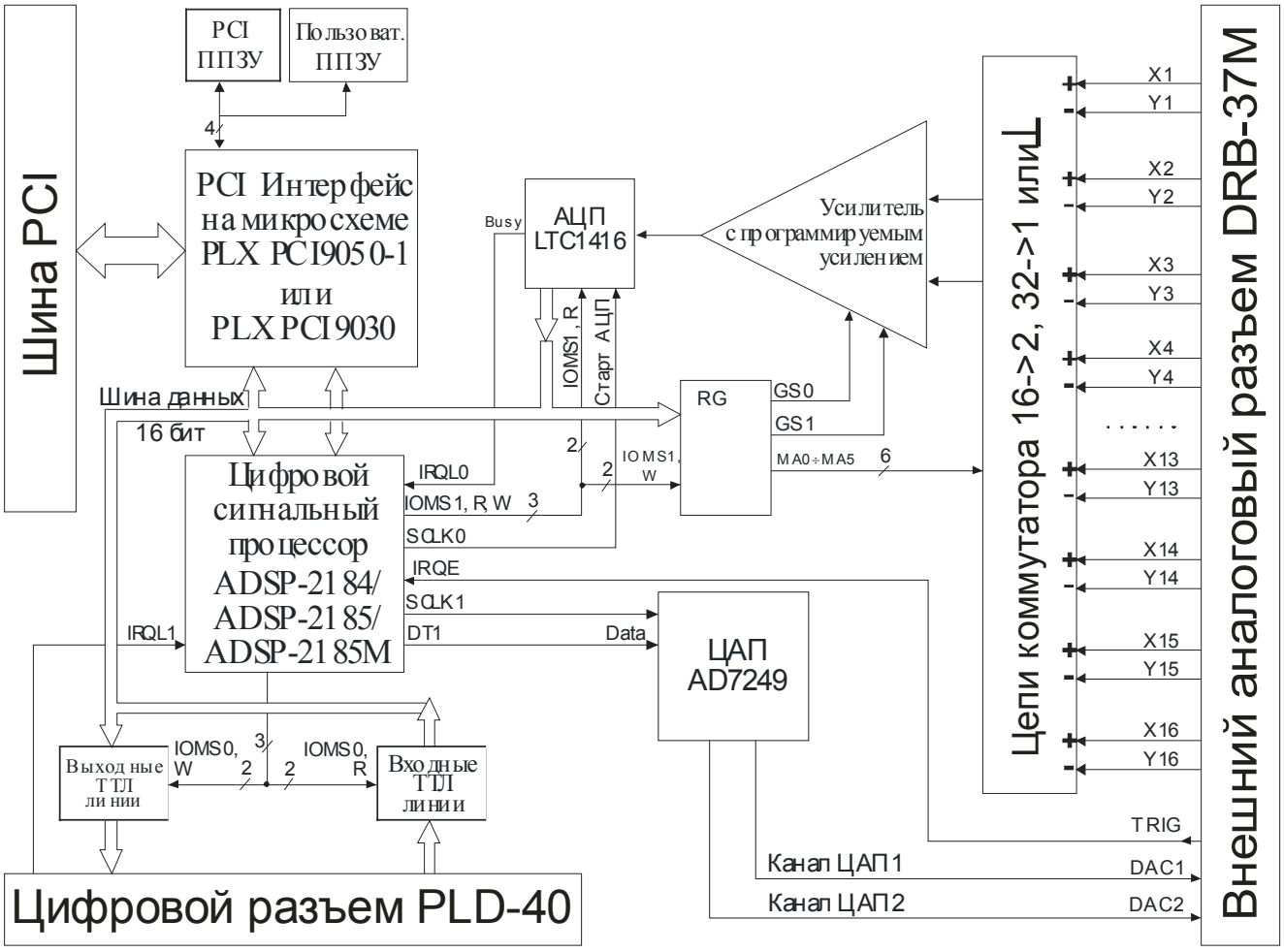
```
{ Сбросим флаг FL2 (предполагается, что он установлен в '1')       }
    RESET FL2;
    NOP; NOP;
    SET FL2;       { вернем флаг FL2 в исходное состояние, т.е. в '1' }
```

1.4.11. Внешняя синхронизация сигнального процессора

При необходимости дополнительной внешней синхронизации можно использовать ТТЛ совместимую линию **INT** разъёма ***PLD-40***, которая подключена к ножке прерывания *IRQL1* сигнального процессора. Данное прерывание работает по уровню, т.е. линия *IRQL1* должна оставаться в активном низком уровне до тех пор, пока процессор не начнет обслуживание прерывания. В обработчике прерывания линию *IRQL1* **обязательно** надо сбрасывать в исходное высокое состояние, чтобы это прерывание не обрабатывалось повторно. В фирменном драйвере эта линия не используется.

1.5. Плата L-780

1.5.1. Структурная схема платы L-780



Как видно из приведенной выше структурной схемы, функционально плата *L-780* выглядит достаточно не сложно. На плате расположены микросхема *PCI9050-1* или *PCI9030* (в зависимости от ревизии платы), полностью обеспечивающая PCI интерфейс платы с PC, и цифровой сигнальный процессор (DSP), управляющий всей периферией на плате, а именно: АЦП, коммутатором, программируемым усилителем, ЦАП и цифровыми линиями. В зависимости от ревизии на данной плате установлен современный высокопроизводительный сигнальный процессор фирмы **Analog Devices, Inc.** *ADSP-2184*/*ADSP-2185* или *ADSP-2185M*, работающий на частоте $2*f_q$, где f_q=14745.6 кГц – частота кварца. Внутренняя архитектура процессора оптимизирована для реализации таких алгоритмов, как цифровая фильтрация, спектральный анализ и т.д. Сам процессор имеет внутреннюю память программ на 4 КСлов и внутреннюю память данных 4 КСлов (процессор ADSP-2185 обладает 16 КСлов памяти программ и 16 КСлов памяти данных). Наличие такого мощного сигнального процессора обеспечивает Вам возможность самостоятельного применения чрезвычайно гибких методов управления всей периферией платы и позволяет переносить часть операций по обработке данных на саму плату (например, Быстрое Преобразование Фурье). Процессор обладает своим собственным контроллером ПДП (канал IDMA: Internal Direct Memory Access; подробнее о работе IDMA см. *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 11.3 "IDMA Port", Analog Devices, Inc., Third Edition September 1995* или на сайте www.analog.com) для доступа к любой ячейки внутренней памяти. Благодаря этому Ваша программа может обращаться к любой ячейки памяти процессора, не прерывая работы самого DSP, что исключительно удобно при построении алгоритмов, работающих в реальном масштабе времени. Максимальная пропускная способность обмена данными между сигнальным процессором и

компьютером составляет приблизительно 10 Мб/с. Весь обмен данными с центральным компьютером DSP осуществляет через свой канал IDMA. Протокол работы с каналом IDMA предусматривает также возможность загрузки в сигнальный процессор управляющей программы (драйвера), которая будет осуществлять требуемые алгоритмы ввода-вывода (процедуру загрузки см. *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 11.3.5 "Boot Loading Through the IDMA Port", стр. 11-24, Analog Devices, Inc., Third Edition September 1995*). Фирменный драйвер *LBIOS* работает по принципу команд и для реализации такой возможности используется прерывание *IRQ2* сигнального процессора. Сначала в соответствующую ячейку памяти данных DSP (предопределенная константа **L_COMMAND**, см. *§ 1.2.5.1.4 "Переменные LBIOS"*) заносится номер команды, которую драйвер должен выполнить. Затем инициируется прерывание *IRQ2*, в ответ на которое обработчик данного прерывания, содержащийся в самом *LBIOS,* выполняет соответствующие данной команде действия. DSP осуществляет получение данных с АЦП, управляет цепями коммутатора входных сигналов, коэффициентом усиления программируемого усилителя, частотой запуска АЦП и, при необходимости, синхронизацией ввода данных по линии **TRIG** внешнего разъёма ***DRB-37M***. Сигнальный процессор обеспечивает также взаимодействие с микросхемой двухканального ЦАП через посредство своего последовательного порта (SPORT1). Управление внешними цифровыми линиями на разъёме ***PLD-40*** осуществляется DSP с помощью чтения/записи нулевой ячейки своего пространства ввода/вывода (I/O Memory Space). Описание этого пространства можно найти, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 10.6.4 "ADSP-2181 I/O Memory Space", стр. 10-32, Analog Devices, Inc., Third Edition September 1995*.

1.5.2. Создание управляющей программы

У пользователя, как правило, не появляется необходимость в написаниии своих собственных управляющих программ для данной платы, т.к. все наиболее часто требуемые алгоритмы работы уже реализованы в фирменном драйвере, находящимся в файле `L780.bio`. Однако если же у Вас все-таки возникла необходимость в создании собственной управляющей программы (например, для формирования какого-либо специализированного алгоритма действия процессора), то для этого придется освоить достаточно несложный язык ассемблера для сигнального процессора. В качестве законченного примера программирования платы на таком языке можно использовать исходные тексты фирменного драйвера, хранящиеся в файлах `DSP\L780\L780.DSP` и `DSP\L780\*.H`. Эти исходные тексты достаточно подробно прокомментированы.

Процесс формирования собственной управляющей программы для DSP потребует от Вас приложения определенных усилий:

1. Вам необходимо будет изучить несложную архитектуру процессора ADSP-218x, а также освоить довольно простой язык его программирования (ассемблер для DSP). Всю подробную информацию об этом можно найти в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Analog Devices, Inc., Third Edition, September 1995* или в русском переводе *"Руководство пользователя по сигнальным микропроцессорам семейства ADSP-2100", под редакцией А.Д.Викторова, Санкт-Петербург, 1997*. Оба эти издания можно приобрести в *ЗАО «Л-Кард»*. Описание и примеры программ для DSP с исходными текстами приводятся в двухтомном справочнике *"Digital Signal Processing Applications Using the ADSP-2100 Family", Analog Devices, Inc.*, который можно найти у официальных российских дистрибьюторов компании ***Analog Devices, Inc.*** (например, фирмы ***Autex Ltd.*** или ***Argussoft Co.***). Много полезного в дополнение к указанной документации можно обнаружить также на сайте *www.analog.com*.
2. Процессоры семейства ADSP-21xx поддерживаются полным набором программных средств отладки. Этот пакет включает в себя несколько программ: построитель системы (`bld21.exe`), ассемблер (`asm21.exe`), линкер или редактор связей (`ld21.exe`) и т.д. Все эти программы очень подробно описываются в оригинальной книге *"ADSP-2100 Family Assembler Tools & Simulator Manual", Analog Devices, Inc., Second Edition, November 1994*, которую можно найти у официальных российских дистрибьюторов компании ***Analog***

Devices, Inc. (например, фирмы ***Autex Ltd.*** или ***Argussoft Co.***). Сам пакет разработчика программ для сигнальных процессоров семейства ADSP-21xx, содержащий все выше указанные средства отладки (кроме `bld21.exe`), можно приобрести в *ЗАО «Л-Кард»*. В качестве архитектурного файла нужно использовать `L780.ACH`.

3. Итак, если у Вас не возникло проблем с первыми двумя этапами, то теперь можно приступать к собственно написанию Вашей управляющей программы. Для начала надо создать соответствующие файлы с исходным кодами Вашей программы на языке ассемблер DSP. Затем эти файлы необходимо оттранслировать (`asm21.exe`) и скомпоновать с помощью редактора связей (`ld21.exe`), формируя, таким образом, выполняемую программу типа `.EXE`, так называемый файл отображения в памяти (memory image file). **!!! Не путать этот файл с обычной выполняемой DOS программой типа `.EXE`!!!** Формат сформированного файла отображения в памяти очень подробно описан в *"ADSP-2100 Family Assembler Tools & Simulator Manual", Appendix **B** "File Format", **B.2** "Memory Image File (.EXE)", Analog Devices, Inc., Second Edition, November 1994*. Именно в этом файле содержатся все коды инструкций Вашей программы с соответствующими адресами их расположения в памяти программ DSP, а также инициализирующие значения Ваших переменных и адреса их нахождения в памяти данных. Зная всю эту информацию нужно просто аккуратно загрузить ее в память DSP по надлежащим адресам. Для упрощения процедуры загрузки полученный файл отображения в память `.EXE` преобразуется с помощью утилиты `DSP\L780\BIN3PCI.EXE` в файл `.BIO` (подробности см. *приложение D*).

Вообще-то, всю эту последовательность создания файла `.BIO` можно проследить по содержанию файлу `DSP\L780\L780.BAT`. Созданный таким образом файл с управляющей программой `.BIO` затем используется, например, в штатной функции загрузки сигнального процессора ***LOAD_LBIOS_PLX()*** (см. *§ 1.2.5.2.2 "Загрузка LBIOS"*).

1.5.3. Загрузка управляющей программы в DSP

Перед началом работы с платой необходимо ее "оживить", т.е. загрузить в DSP либо фирменный драйвер, находящийся в файле `L780.bio`, либо Вашу собственную управляющую программу. **Только** после выполнения такой процедуры плата будет корректно работать с функциями штатной или Вашей (если создадите) библиотеки. Формат файла `L780.bio`, содержащий все коды инструкций управляющей программы, подробно описан в *приложении D*. Коды инструкций сигнального процессора из этого файла необходимо расположить по соответствующим адресам памяти программ, также как данные, инициализирующие переменные драйвера, в памяти данных и запустить управляющую программу на выполнение. Все это и называется загрузка управляющей программы в DSP.

Сама процедура начальной загрузки управляющей программы во внутреннюю память DSP достаточно проста и хорошо описана в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 11.3.5 "Boot Loading Through the IDMA Port", стр. 11-24, Analog Devices, Inc., Third Edition September 1995*. Она предполагает выполнение следующих несложных шагов:

- произведите сброс (RESET) DSP на данной плате, например, с помощью API функции ***PLATA_RESET_PLX()*** (см. *§ 1.2.5.2.6 "Сброс DSP на плате"*);
- заполните необходимыми данными по каналу IDMA память данных и программ, кроме ячейки с адресом PM(0x0000), для чего можно воспользоваться соответствующими API функциями ***PUT_DM_ARRAY_PLX()*** (см. *§ 1.2.5.3.7 "Запись массива слов в память данных DSP"*), ***PUT_PM_WORD_PLX()*** (см. *§ 1.2.5.3.8 "Запись массива слов в память программ DSP"*) и т.д.
- запишите данные в ячейку памяти программ по адресу PM(0x0000) для старта Вашей управляющей программы (при этом ее выполнение начнется с инструкции, находящейся в PM(0x0000)).

Внимание!!! Необходимо обязательно убедиться, что Вы загрузили всю нужную информацию в память DSP **до** записи данных по адресу PM(0x0000).

В общем-то **ВСЕ!** Теперь можно спокойно приступать к управлению платой с помощью функций штатной или Вашей библиотеки.

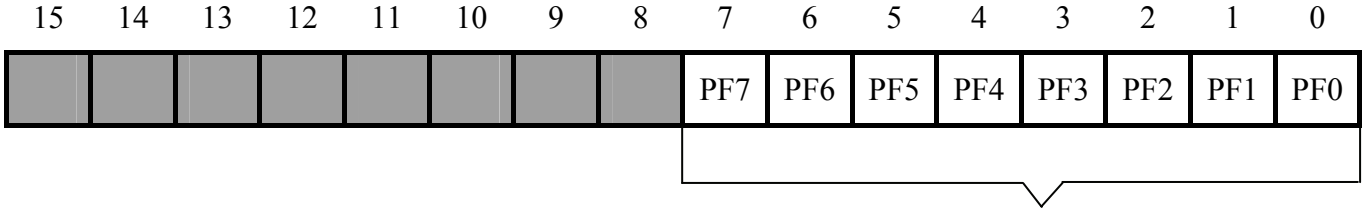
1.5.4. Установка флагов PFx сигнального процессора

Микросхемы сигнальных процессоров ADSP-2184/2185/2186 имеют восемь дополнительных неспециализированных выводов, так называемых флагов, обозначаемых обычно **PF0÷PF7**. Данные флаги могут быть индивидуально запрограммированы как на вход, так и на выход. На указанных типах процессоров выводы этих флагов обладают также и дополнительными функциями. Так флаг **PF4** совмещен с входом прерывания *IRQE*, **PF5** – с *IRQL0*, **PF6** – с *IRQL1*, а **PF7** – с *IRQ2* и т.д. (подробности можно найти в техническом описании этих DSP на сайте www.analog.com). Т.о. флаги **PFx** необходимо запрограммировать на вход-выход надлежащим образом, т.е. флаги **PF1÷PF2** и **PF4÷PF7** как входные, а флаги **PF0** и **PF3** как выходные. Программирование флагов осуществляется с помощью двух управляющих регистров DSP:

- ✓ регистр Programmable Flag & Composite Select Control, находящий по адресу DM(0x3FE6) в памяти данных, управляет направлением флага
  - – 1 – выход,
  - – 0 – вход;
- ✓ регистр Programmable Flag Data, расположенный по адресу DM(0x3FE5), используется для записи и считывания значений флагов.

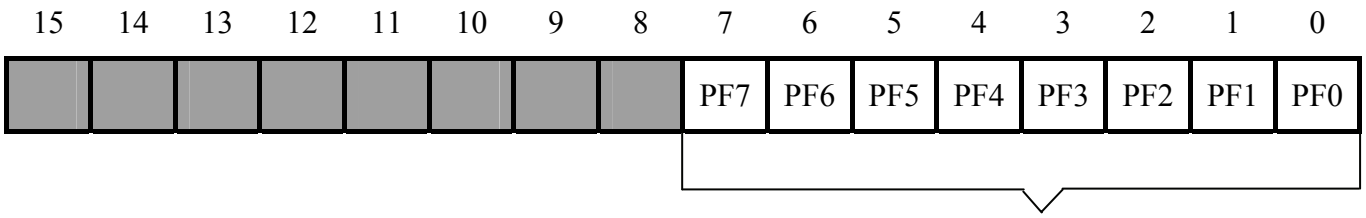
Формат данных регистров приведен ниже и подробно описан в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 9.5 "Flag Pins", cmp. 9-15, Analog Devices, Inc., Third Edition, September 1995.*

Programmable Flag & Composite Select Control (DM(0x3FE6))



Тип PFx флага
1 → выходной
0 → входной

Programmable Flag Data (DM(0x3FE5))



PFx Data

Следовательно, для надлежащего программирования флагов PFx необходимо написать следующий код:

```
     const Prog_Flag_Comp_Sel_Ctrl  =  0x3FE6;
     . . . . .

{ флаги PFx: PF1, PF2, PF4÷PF7 - входные, а PF0 и PF3 – выходные     }
     AR=0x09;
     DM(Prog_Flag_Comp_Sel_Ctrl)=AR;
```

1.5.5. АЦП, коммутатор и программируемый усилитель

Все взаимодействие сигнального процессора с 14[ти] битным АЦП **LTC1416** (техническое описание можно найти на сайте www.linear-tech.com.com), коммутатором и программируемым усилителем осуществляется через посредство первой ячейки пространства ввода-вывода DSP (I/O Memory Space) и прерывания *IRQL0*. Описание I/O Memory Space можно найти, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 10.6.4 "ADSP-2181 I/O Memory Space", стр. 10-32, Analog Devices, Inc., Third Edition September 1995*.

Управление адресом (номером) канала и коэффициентом усилением сигнала происходит с использованием двух управляющих регистров:

- буферный регистр адреса/усиления канала;
- выходной регистр адреса/усиления канала.

Информация, находящаяся в выходном регистре, определяет адрес канала и коэффициент усиления для текущего преобразований АЦП. Формат этой информации полностью соответствует формату логического канала, описанному в *§ 1.2.2.2.3 "Логический номер канала АЦП"*. Данные из буферного регистра переписываются в выходной регистр по возрастающему фронту сигнала #BUSY АЦП. Таким образом, данные, записанные в буферный регистр при обработки прерывания от выборки N (см. ниже), будут действовать для выборки N+1. Запись данных в буферный и выходной регистры производится через первую ячейку пространства ввода-вывода `IO(1)`, при учете состояния флага **FL0**:

- если **FL0**=0 во время записи в `IO(1)`, то грузится буферный регистр;
- если **FL0**=1 во время записи в `IO(1)`, то грузится выходной регистр.

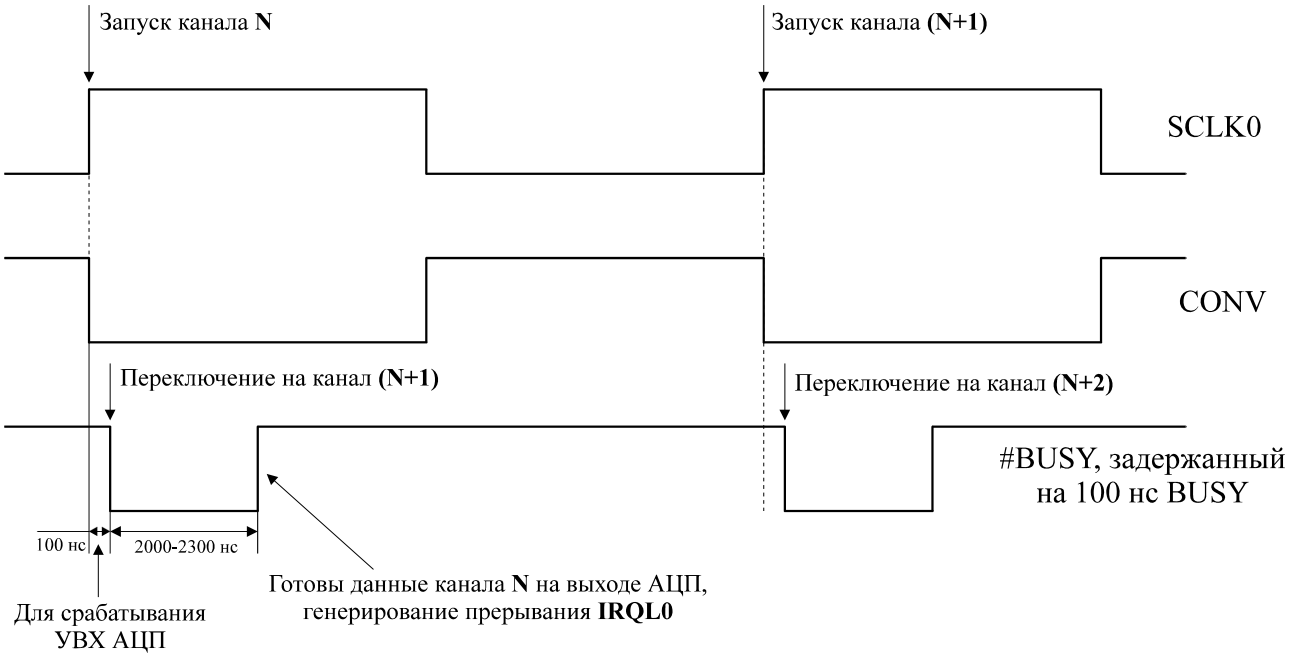
В качестве сигналов запуска преобразования АЦП на данной плате используются инвертированные внутренние тактовые синхроимпульсы последовательного порта SPORT0. С помощью регистров управления работой SPORT0, адреса которых в памяти данных DM(0x3FF6) и DM(0x3FF5) соответственно, Вы можете разрешить либо запретить генерацию этих импульсов, а также задавать частоту их следования, определяя, таким образом, частоту работы АЦП. Так частота работы АЦП определяется по следующей формуле:

$$\mathbf{ADC\_Rate}=\mathbf{f_q}/(\mathbf{SCLKDIV}+1)\ \text{кГц},$$

где $\mathbf{f_q}$=14745.6 кГц – частота установленного на плате кварца, **SCLKDIV** – содержимое регистра DM(0x3FF5). Очень подробное описание работы сериальных портов DSP можно найти в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 5 "Serial Ports", стр. 5-1, Analog Devices, Inc., Third Edition, September 1995*

Для обслуживания АЦП предлагается использовать прерывание *IRQL0*. Запрос на это прерывание устанавливается по возрастающему фронту сигнала #BUSY АЦП. Готовые данные с АЦП считываются в обработчике *IRQL0* через первую ячейку пространства ввода-вывода `IO(1)`. Т.к. *IRQL0* работает по уровню, в обработчике **обязательно** надо сбрасывать запрос на это прерывание чтением данных АЦП через `IO(1)`. При заходе в обработчик первым делом надо записать в буферный регистр адреса/усиления следующее значение логического канала, а потом уже считывать данные с АЦП и т.д.

Временные диаграммы работы АЦП приведены на следующем рисунке:



Теперь попробуем проиллюстрировать все выше сказанное примером:

```
{ Для начала отконфигурируем SPORT0                                    }
{ SPORT0 - disable, SPORT1 - disable, SPORT1 - serial port             }
    AR=0x0400;
    DM(Sys_Ctrl_Reg)=AR;      { 0x3FFF - System Control Register       }
{ ************************************************************** }
{ Set SPORT0 for start of ADC chip                                     }
{ Serial Clock Divide Modulus                                          }
    AR = 99;                  { частота запуска АЦП (SCLK0)            }
    DM(Sport0_Sclkdiv) = AR;  { 0x3FF5 - Serial Clock Divide Modulus   }
{ Receive Frame Sync Divide Modulus                                    }
    AR = 0xF;              { may be any number: - not used             }
    DM(Sport0_Rfsdiv)=AR;  { 0x3FF4-Receive Frame Sync Divide Modulus }
{ Control word for SPORT0: SCLK0 – external (остановим АЦП)            }
{ high level, alternate external receive frame on each word(16 bit) - not used  }
{ high level, alternate external transmit frame on each word(16 bit) - not used }
    AR = 0x3C1F;              { 0111 1100 0001 1111                    }
    DM(Sport0_Ctrl_Reg) = AR; { 0x3FF6 - SPORT0 Control Register       }
{ ************************************************************** }
{ задержка на 2.7 мкс, чтобы оцифрился последний предыдущий отсчет    }
    cntr=80;
    DO DelayLoop UNTIL CE;
DelayLoop: NOP;
{ Сброс запроса прерывания IRQL0 (на всякий случай)                    }
    AR = IO(1);
{ очистим все запросы на прерывания                                    }
    IFC = 0xFF; NOP;
{ разрешим прерывания IRQL0                                            }
    IMASK=0x80; NOP;
```

```
{ зададим усиление и номер канала для текущего и следующего отсчетов,}
{ предполагая наличия циклического буфера с управляющей таблицей     }
{ (I2, M2, L2)                                                        }
    AR=DM(I2, M2);
    SET FL0;
    IO(1)=AR;
    RESET FL0;
    AR=DM(I2, M2);
    IO(1)=AR;

{ задержка на 2.7 мкс, чтобы установился аналоговый тракт             }
    cntr=80;
    DO DelayLoop1 UNTIL CE;
DelayLoop1: NOP;

{ включим клоки, запустив АЦП, т.е. сделаем SCLK внутренним           }
    AR = 0x7C1F;                { 0111 1100 0001 1111                 }
    DM(Sport0_Ctrl_Reg) = AR;   { 0x3FF6 - SPORT0 Control Register    }

{ Все! Теперь можно ждать прихода прерывания IRQL0.                   }
{ Частота работы АЦП: ADC_Rate=14745.6/(99+1)=147.456 кГц             }

    . . . . .

{ Обработчик прерывания IRQL0 может содержать следующие строчки       }

    AR=DM(I2, M2);      { зададим усиление и номер канала для         }
    IO(1)=AR;           { следующего отсчета                          }
    . . . . .
    AR=IO(1);           { теперь в AR находится отсчет с АЦП          }
```

1.5.6. ЦАП

Управление микросхемой двухканального 12[ти] битный ЦАП **AD7249** (техническое описание можно найти на сайте www.analog.com), которая может быть установлена на плате по Вашему желанию, сигнальный процессор осуществляет с помощью последовательного порта SPORT1. Для этого необходимо его запрограммировать надлежащим образом, а именно:

- ✓ длина слова (word length) – 16 бит;
- ✓ тактовые синхроимпульсы (serial clocks) – внутренние с периодом не менее 0.4 мкс;
- ✓ кадровая синхронизация приема каждого слова;
- ✓ внутренняя кадровая синхронизация приема;
- ✓ альтернативная кадровая синхронизация приема;
- ✓ кадровый сигнал приема активен по низкому уровню;
- ✓ кадровая синхронизация передачи каждого слова;
- ✓ внешняя кадровая синхронизация передачи;
- ✓ альтернативная кадровая синхронизация передачи;
- ✓ кадровый сигнал передачи активен по низкому уровню.

Выводы сигнального процессора для кадровой синхронизации приема и передачи (выводы RFS1 и TFS1 соответственно) объединены, т.е. внутренне генерируемые сигналы кадровой синхронизации приема являются сигналами внешней кадровой синхронизации передачи. Частоту генерирования сигналов на выводе RFS1 можно изменять в достаточно широких пределах, устанавливая таким образом частоту вывода данных на ЦАП. Очень подробное описание конфигурирования сериальных портов DSP можно найти в *“ADSP-2100 Family User’s Manual (Includes ADSP-2171, ADSP-2181)”, Chapter 5 “Serial Ports”, стр. 5-1, Analog Devices, Inc., Third Edition, September 1995*

Формат передаваемых в микросхему ЦАП данных приведен в следующей таблице:

Номер бита	Назначение
0÷11	$12^{\text{ти}}$ битный код ЦАП
12	Выбирает номер ЦАП: ✓ если ‘0’, то запись в первый канал ЦАП; ✓ если ‘1’, то запись во второй канал ЦАП.
13÷15	Не используются

Приведем пример подготовки порта SPORT1 для управления ЦАП’ами:

```
{ Для начала отконфигурируем SPORT1 на передачу данных                }
{ SPORT0 - disable, SPORT1 - disable, SPORT1 - serial port            }
    AR=0x0400;
    DM(Sys_Ctrl_Reg)=AR;      { 0x3FFF - System Control Register      }

{ ************************************************************** }
{   Set SPORT1 for transmit digital codes in DAC                      }
{   SCLK1 and Receive Frame - internal, word = 16 bits                }
{   Transmit Frame - external                                         }

{   Serial Clock Divide Modulus                                       }
    AR=5;                     { SCLK1 – internal, T=406.9 ns          }
    DM(Sport1_Sclkdiv) = AR;   { 0x3FF1 - Serial Clock Divide Modulus }

{Receive Frame Sync Divide Modulus                                    }
    AR = 49;            { Определяет частоту вывода отсчетов с ЦАП'а  }
    DM(Sport1_Rfsdiv)=AR;   {0x3FF0-Receive Frame Sync Divide Modulus }

{ Control word for SPORT1: SCLK1 - internal                           }
{ low level, alternate internal receive frame on each word(16 bit)    }
{ low level, alternate external transmit frame on each word(16 bit)   }
    AR = 0x7DCF;              { 0111 1101 1100 1111                   }
    DM(Sport1_Ctrl_Reg) = AR; { 0x3FF2 - SPORT1 Control Register      }
{ ************************************************************** }

{ SPORT0 - enable, SPORT1 - enable, SPORT1 - serial port              }
    AR = 0x1C00;              { 0001 1100 0000 0000                   }
    DM(Sys_Ctrl_Reg) = AR;    { 0x3FFF - System Control Register      }

{ Мы установили частоту вывода данных на ЦАП равной                   }
{ fq/((5+1)*(49+1))=49.512 Кгц, где fq=14745.6 кГц - частота кварца   }
{ Теперь с этой частотой будут приходить прерывания SPORT1 Transmit,  }
{ обработчик, которого каждый раз должен записывать в регистр        }
{ передачи TX1 очередное значение, посылаемое в микросхему ЦАП       }

{ Мы же сейчас просто установим нулевой уровень на первом ЦАП’е       }
    TX1=0x0;

    . . . . .

{ А сейчас установим уровень 5.0 В (код 2047) на втором ЦАП’е         }
    TX1=0x17FF;
```

1.5.7. Управление ТТЛ линиями

Управление цифровыми линиями осуществлять достаточно просто. Любая запись какого-либо числа по адресу 0x0 в пространстве ввода-вывода (I/O Memory Space) сигнального процессора устанавливает состояния всех 16[ти] выходных линий на разъёме ***PLD-40*** в соответствие с битами записываемого значения. Любое чтение по адресу 0x0 из пространства ввода-вывода (I/O Memory Space) сигнального процессора считывает состояние всех 16[ти] входных линий на разъёме ***PLD-40*** платы. Подробности разъёма см. *"Руководство пользователя", § 3.3.2. "Внутренний разъём для подключения цифровых сигналов"*. Описание I/O Memory Space можно найти, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 10.6.4 "ADSP-2181 I/O Memory Space", стр. 10-32, Analog Devices, Inc., Third Edition September 1995*.

В платах *Rev. C* имеется возможность программным образом управлять доступом к выходным цифровым линиям. (подробнее см. *§ 1.2.4.5. "Особенности плат Rev. C"*). Для этого используется флаг сигнального процессора **PF0**, который должен быть заблаговременно настроен как выходной. Сброс флажка **PF0** в состояние лог. '0' разрешает выходные линии платы, а установка в лог. '1' – переводит их в '*третье*' стостояние.

Рассмотрим простой пример работы с цифровыми линиями:

```
{ прочитаем текущее состояние флагов PFx                    }
    AR=DM(Prog_Flag_Data);
{ разрешим доступ к выходным линиям, сбросив флажок PF0     }
    AR=CLRBIT 0 OF AR; DM(Prog_Flag_Data)=AR;
{ теперь считаем входные цифровые линии                     }
    AR=IO(0x0);
{ установим прочитанные состояния на выходных линиях        }
    IO(0x0)=AR;
```

1.5.8. Генерирование прерываний в PC

На плате предусмотрена возможность генерирования прерываний в PC. Эту способность можно использовать, когда Вам требуется проинформировать компьютер о наступлении какого-либо события, требующего срочной обработки. Номер прерывания для каждой платы назначается либо BIOS компьютера, либо *Windows* и его можно узнать, прочитав, например, поле **InterruptNumber** в соответствующей структуре типа *BOARD_INFO* (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*). Для генерации прерываний из платы в PC можно использовать следующие флаги сигнального процессора:

- ✓ **FL2** (платы *Rev. A* и *B*).
- ✓ **FL2** и **FL1** (платы *Rev. C*).

В штатном *LBIOS* флаг **FL2** используется для генерации прерываний по мере заполнения FIFO буфера АЦП, а флаг **FL1** – FIFO буфера ЦАП, информируя PC о готовности тех или иных данных. При работе с прерываниями необходимо, чтобы соответствуюший флаг **FLx** постоянно находился в состоянии лог. '1', а переход '1'-'0'-'1' осуществлялся только тогда, когда надо вызвать прерывание в компьютере. **Внимание!!!** Прерывание в компьютере будет сгенерировано только в случае, если со стороны PC оно будет разрешено с помощью API функции ***INIT_INTERRUPT_PLX()*** (см. *§ 1.2.5.6.1 "Установка обработчика прерываний"*).

Приведем пример генерирования одного прерывания в компьютер:

```
{ Сбросим флаг FL2 (предполагается, что он находился в лог. '1')   }
    RESET FL2;
{ организуем небольшую задержку                                     }
    NOP; NOP;
{ вернем флаг FL2 в исходное состояние, т.е. в лог '1'              }
    SET FL2;
```

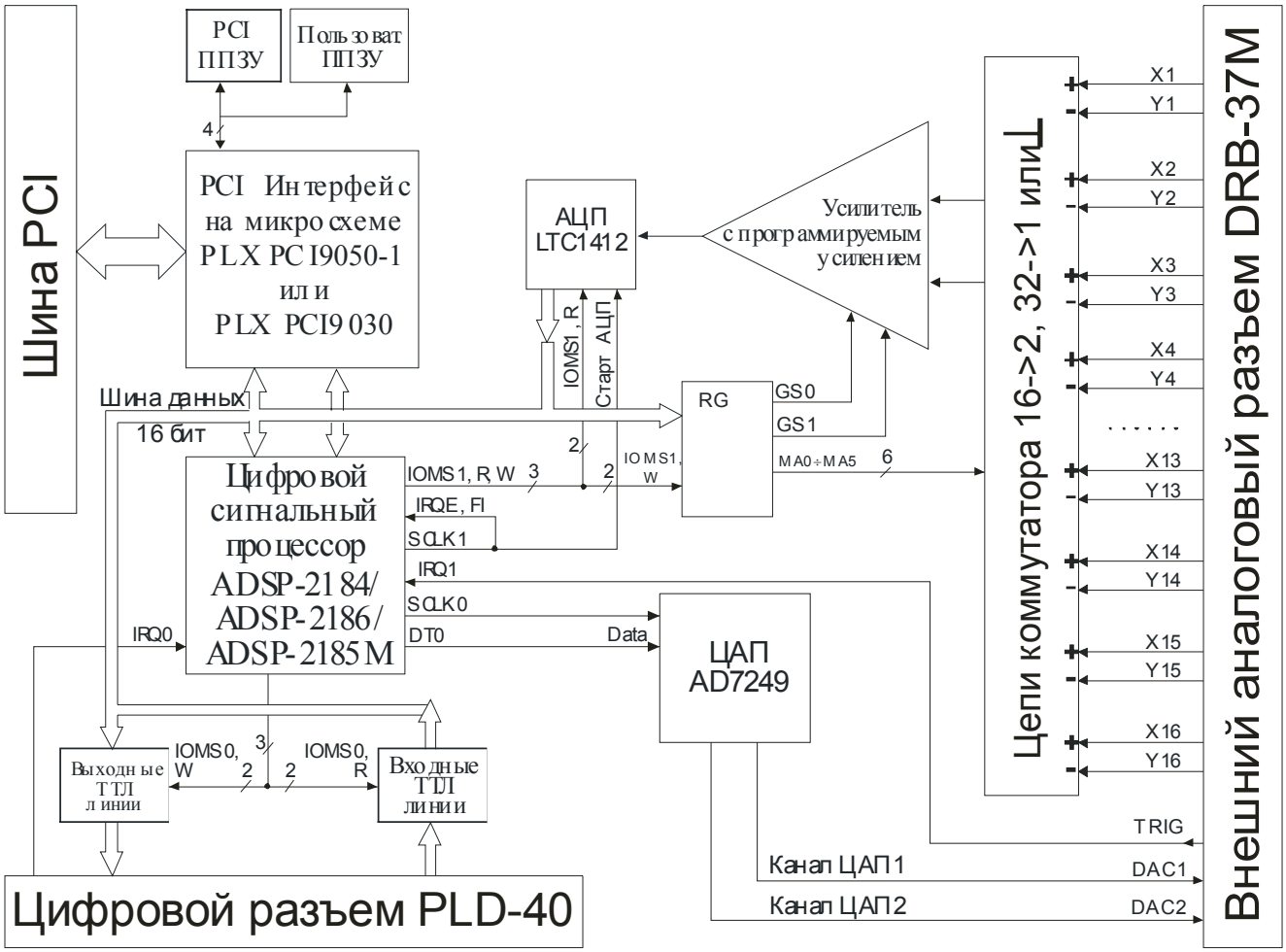
1.5.9. Внешняя синхронизация сигнального процессора

При необходимости дополнительной внешней синхронизации DSP можно использовать следующие ТТЛ совместимые сигналы на плате:

- ✓ линия **TRIG** на внешнем разъёме ***DRB-37M***, которая подключена к ножке прерывания *IRQE* сигнального процессора (по фронту). Данное прерывание *IRQE* генерируется при отрицательном перепаде импульса (⎍) длительностью не менее 50 нс. В фирменном драйвере эта линия используется для цифровой синхронизации ввода данных с АЦП.
- ✓ линия **INT** на разъёме ***PLD-40***, которая подключена к ножке прерывания *IRQL1* сигнального процессора (по уровню). Т.к. данное прерывание работает по уровню, то линия *IRQL1* должна оставаться в активном низком уровне до тех пор, пока процессор не начнет обслуживание прерывания. В обработчике прерывания линию *IRQL1* **обязательно** надо сбрасывать в исходное высокое состояние, чтобы это прерывание не обрабатывалось повторно. В фирменном драйвере эта линия не используется.

1.6. Плата L-783

1.6.1. Структурная схема платы L-783



Как видно из приведенной выше структурной схемы, функционально плата *L-783* выглядит достаточно не сложно. На плате расположены микросхема *PCI9050-1* или *PCI9030* (в зависимости от ревизии платы), полностью обеспечивающая PCI-интерфейс платы с PC, и цифровой сигнальный процессор (DSP), управляющий всей периферией на плате, а именно: АЦП, коммутатором, программируемым усилителем, ЦАП и цифровыми линиями. В зависимости от ревизии на данной плате установлен современный высокопроизводительный сигнальный процессор фирмы **Analog Devices, Inc.** *ADSP-2184*/*ADSP-2186* или *ADSP-2185M*, работающий на частоте $2*f_q$, где f_q=20000.0 кГц – частота кварца. Внутренняя архитектура процессора оптимизирована для реализации таких алгоритмов, как цифровая фильтрация, спектральный анализ и т.д. Сам процессор имеет внутреннюю память программ на 4 КСлов и внутреннюю память данных 4 КСлов (процессор ADSP-2186 обладает 8 КСлов памяти программ и 8 КСлов памяти данных). Наличие такого мощного сигнального процессора обеспечивает Вам возможность самостоятельного применения чрезвычайно гибких методов управления всей периферией платы и позволяет переносить часть операций по обработке данных на саму плату (например, Быстрое Преобразование Фурье). Процессор обладает своим собственным контроллером ПДП (канал IDMA: Internal Direct Memory Access; подробнее о работе IDMA см. *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 11.3 "IDMA Port", Analog Devices, Inc., Third Edition September 1995* или на сайте www.analog.com) для доступа к любой ячейки внутренней памяти. Благодаря этому Ваша программа может обращаться к любой ячейки памяти процессора, не прерывая работы самого DSP, что исключительно удобно при построении алгоритмов, работающих в реальном масштабе времени. Максимальная пропускная способность обмена данными между сигнальным процессором и

компьютером составляет приблизительно 10 Мб/с. Весь обмен данными с центральным компьютером DSP осуществляет через свой канал IDMA. Протокол работы с каналом IDMA предусматривает также возможность загрузки в сигнальный процессор управляющей программы (драйвера), которая будет осуществлять требуемые алгоритмы ввода-вывода (процедуру загрузки см. *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 11.3.5 "Boot Loading Through the IDMA Port", стр. 11-24, Analog Devices, Inc., Third Edition September 1995*). Фирменный драйвер *LBIOS* работает по принципу команд и для реализации такой возможности используется прерывание *IRQ2* сигнального процессора. Сначала в соответствующую ячейку памяти данных DSP (предопределенная константа **L_COMMAND**, см. *§ 1.2.5.1.4 "Переменные LBIOS"*) заносится номер команды, которую драйвер должен выполнить. Затем инициируется прерывание *IRQ2*, в ответ на которое обработчик данного прерывания, содержащийся в самом *LBIOS,* выполняет соответствующие данной команде действия. DSP осуществляет получение данных с АЦП, управляет цепями коммутатора входных сигналов, коэффициентом усиления программируемого усилителя, частотой запуска АЦП и, при необходимости, синхронизацией ввода данных по линии **TRIG** внешнего разъёма ***DRB-37M***. Сигнальный процессор обеспечивает также взаимодействие с микросхемой двухканального ЦАП через посредство своего последовательного порта (SPORT0). Управление внешними цифровыми линиями на разъёме ***PLD-40*** осуществляется DSP с помощью чтения/записи нулевой ячейки своего пространства ввода/вывода (I/O Memory Space). Описание этого пространства можно найти, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 10.6.4 "ADSP-2181 I/O Memory Space", стр. 10-32, Analog Devices, Inc., Third Edition September 1995*.

1.6.2. Создание управляющей программы

У пользователя, как правило, не появляется необходимость в написаниии своих собственных управляющих программ для данной платы, т.к. все наиболее часто требуемые алгоритмы работы уже реализованы в фирменном драйвере, находящимся в файле `L783.bio`. Однако если же у Вас все-таки возникла необходимость в создании собственной управляющей программы (например, для формирования какого-либо специализированного алгоритма действия процессора), то для этого придется освоить достаточно несложный язык ассемблера для сигнального процессора. В качестве законченного примера программирования платы на таком языке можно использовать исходные тексты фирменного драйвера, хранящиеся в файлах `DSP\L783\L783.DSP` и `DSP\L783\*.H`. Эти исходные тексты достаточно подробно прокомментированы.

Процесс формирования собственной управляющей программы для DSP потребует от Вас приложения определенных усилий:

1. Вам необходимо будет изучить несложную архитектуру процессора ADSP-218x, а также освоить довольно простой язык его программирования (ассемблер для DSP). Всю подробную информацию об этом можно найти в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Analog Devices, Inc., Third Edition, September 1995* или в русском переводе *"Руководство пользователя по сигнальным микропроцессорам семейства ADSP-2100", под редакцией А.Д.Викторова, Санкт-Петербург, 1997*. Оба эти издания можно приобрести в *ЗАО «Л-Кард»*. Описание и примеры программ для DSP с исходными текстами приводятся в двухтомном справочнике *"Digital Signal Processing Applications Using the ADSP-2100 Family", Analog Devices, Inc.*, который можно найти у официальных российских дистрибьюторов компании ***Analog Devices, Inc.*** (например, фирмы ***Autex Ltd.*** или ***Argussoft Co.***). Много полезного в дополнение к указанной документации можно обнаружить также на сайте *www.analog.com*.
2. Процессоры семейства ADSP-21xx поддерживаются полным набором программных средств отладки. Этот пакет включает в себя несколько программ: построитель системы (`bld21.exe`), ассемблер (`asm21.exe`), линкер или редактор связей (`ld21.exe`) и т.д. Все эти программы очень подробно описываются в оригинальной книге *"ADSP-2100 Family Assembler Tools & Simulator Manual", Analog Devices, Inc., Second Edition, November 1994*, которую можно найти у официальных российских дистрибьюторов компании

Analog Devices, Inc. (например, фирмы ***Autex Ltd.*** или ***Argussoft Co.***). Сам пакет разработчика программ для сигнальных процессоров семейства ADSP-21xx, содержащий все выше указанные средства отладки (кроме `bld21.exe`), можно приобрести в *ЗАО «Л-Кард»*. В качестве архитектурного файла нужно использовать `L783.ACH`.

3. Итак, если у Вас не возникло проблем с первыми двумя этапами, то теперь можно приступать к собственно написанию Вашей управляющей программы. Для начала надо создать соответствующие файлы с исходным кодами Вашей программы на языке ассемблер DSP. Затем эти файлы необходимо оттранслировать (`asm21.exe`) и скомпоновать с помощью редактора связей (`ld21.exe`), формируя, таким образом, выполняемую программу типа `.EXE`, так называемый файл отображения в памяти (memory image file). **!!! Не путать этот файл с обычной выполняемой DOS программой типа `.EXE`!!!** Формат сформированного файла отображения в памяти очень подробно описан в *"ADSP-2100 Family Assembler Tools & Simulator Manual", Appendix **B** "File Format", **B.2** "Memory Image File (.EXE)", Analog Devices, Inc., Second Edition, November 1994*. Именно в этом файле содержатся все коды инструкций Вашей программы с соответствующими адресами их расположения в памяти программ DSP, а также инициализирующие значения Ваших переменных и адреса их нахождения в памяти данных. Зная всю эту информацию нужно просто аккуратно загрузить ее в память DSP по надлежащим адресам. Для упрощения процедуры загрузки полученный файл отображения в память `.EXE` преобразуется с помощью утилиты `DSP\L783\BIN3PCI.EXE` в файл `.BIO` (подробности см. *приложение D*).

Вообще-то, всю эту последовательность создания файла `.BIO` можно проследить по содержанию файлу `DSP\L783\L783.BAT`. Созданный таким образом файл с управляющей программой `.BIO` затем используется, например, в штатной функции загрузки сигнального процессора ***LOAD_LBIOS_PLX()*** (см. *§ 1.2.5.2.2 "Загрузка LBIOS"*).

1.6.3. Загрузка управляющей программы в DSP

Перед началом работы с платой необходимо ее "оживить", т.е. загрузить в DSP либо фирменный драйвер, находящийся в файле `L783.bio`, либо Вашу собственную управляющую программу. **Только** после выполнения такой процедуры плата будет корректно работать с функциями штатной или Вашей (если создадите) библиотеки. Формат файла `L783.bio`, содержащий все коды инструкций управляющей программы, подробно описан в *приложении D*. Коды инструкций сигнального процессора из этого файла необходимо расположить по соответствующим адресам памяти программ, также как данные, инициализирующие переменные драйвера, в памяти данных и запустить управляющую программу на выполнение. Все это и называется загрузка управляющей программы в DSP.

Сама процедура начальной загрузки управляющей программы во внутреннюю память DSP достаточно проста и хорошо описана в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 11.3.5 "Boot Loading Through the IDMA Port", стр. 11-24, Analog Devices, Inc., Third Edition September 1995*. Она предполагает выполнение следующих несложных шагов:

- произведите сброс (RESET) DSP на данной плате, например, с помощью API функции ***PLATA_RESET_PLX()*** (см. *§ 1.2.5.2.6 "Сброс DSP на плате"*);
- заполните необходимыми данными по каналу IDMA память данных и программ, кроме ячейки с адресом PM(0x0000), для чего можно воспользоваться соответствующими API функциями ***PUT_DM_ARRAY_PLX()*** (см. *§ 1.2.5.3.7 "Запись массива слов в память данных DSP"*), ***PUT_PM_WORD_PLX()*** (см. *§ 1.2.5.3.8 "Запись массива слов в память программ DSP"*) и т.д.
- запишите данные в ячейку памяти программ по адресу PM(0x0000) для старта Вашей управляющей программы с инструкции, находящейся в PM(0x0000).

Внимание!!! Необходимо обязательно убедиться, что Вы загрузили всю нужную информацию в память DSP **до** записи данных по адресу PM(0x0000).

В общем-то **ВСЁ!** Теперь можно спокойно приступать к управлению платой с помощью функций штатной или Вашей библиотеки.

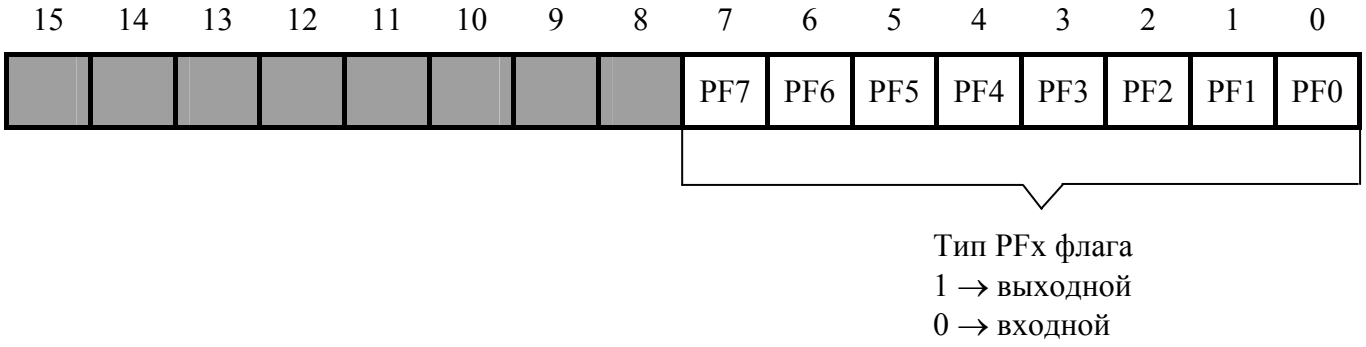
1.6.4. Установка флагов PFx сигнального процессора

Микросхемы сигнальных процессоров ADSP-2184/2185/2186 имеют восемь дополнительных неспециализированных выводов, так называемых флагов, обозначаемых обычно **PF0÷PF7**. Данные флаги могут быть индивидуально запрограммированы как на вход, так и на выход. На указанных типах процессоров выводы этих флагов обладают также и дополнительными функциями. Так флаг **PF4** совмещен с входом прерывания *IRQE*, а **PF7** – с *IRQ2* и т.д. (подробности можно найти в техническом описании этих DSP на сайте www.analog.com). Т.о. флаги **PFx** необходимо запрограммировать на вход-выход надлежащим образом, т.е. флаги **PF1÷PF2** и **PF4÷PF7** как входные, а флаги **PF0** и **PF3** как выходные. Программирование флагов осуществляется с помощью двух управляющих регистров DSP:

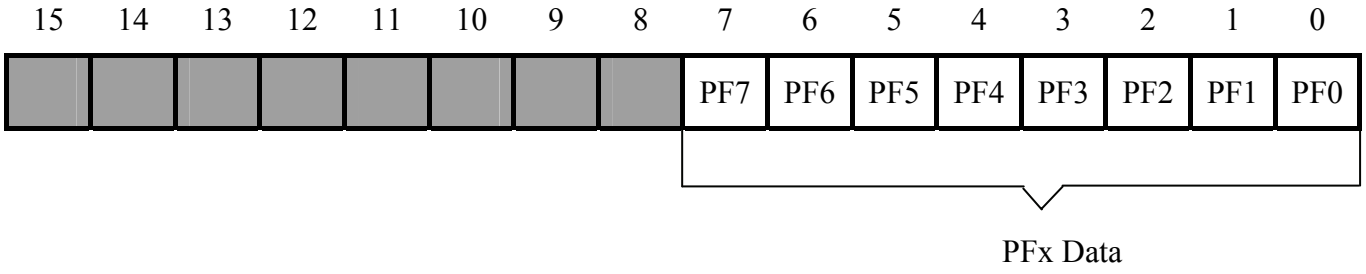
- ✓ регистр Programmable Flag & Composite Select Control, находящий по адресу DM(0x3FE6) в памяти данных, управляет направлением флага
  - 1 – выход,
  - 0 – вход;
- ✓ регистр Programmable Flag Data, расположенный по адресу DM(0x3FE5), используется для записи и считывания значений флагов.

Формат данных регистров приведен ниже и подробно описан в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 9.5 "Flag Pins", стр. 9-15, Analog Devices, Inc., Third Edition, September 1995.*

Programmable Flag & Composite Select Control (DM(0x3FE6))



Programmable Flag Data (DM(0x3FE5))



Следовательно, для надлежащего программирования флагов PFx необходимо написать следующий код:

```
     const Prog_Flag_Comp_Sel_Ctrl  =  0x3FE6;
     . . . . .
{ конфигурируем флаги PFx: PF1, PF2, PF4÷PF7 - входные,                }
{ а PF0 и PF3 – выходные                                                }
     AR=0x09;
     DM(Prog_Flag_Comp_Sel_Ctrl)=AR;
```

1.6.5. АЦП, коммутатор и программируемый усилитель

Все взаимодействие сигнального процессора с 12[ти] битным АЦП **LTC1412** (техническое описание можно найти на сайте www.linear-tech.com), коммутатором и программируемым усилителем осуществляется через посредство первой ячейки пространства ввода-вывода DSP (I/O Memory Space) и прерывания *IRQE*, сигнал с которого одновременно заведен на входной флаг FI. Описание I/O Memory Space можно найти, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 10.6.4 "ADSP-2181 I/O Memory Space", стр. 10-32, Analog Devices, Inc., Third Edition September 1995*.

Для обслуживания АЦП предлагается использовать прерывание *IRQE* или входной флаг FI. Запрос на прерывание устанавливается по возрастающему фронту сигнала SCLK1 последовательного порта SPORT1. Готовые данные с АЦП можно считывать, например, в обработчике *IRQE* через первую ячейку пространства ввода-вывода `IO(1)`. При заходе в обработчик первым делом надо записать в буферный регистр адреса/усиления следующее значение логического канала, а потом уже считывать данные с АЦП и т.д. Готовность данных с АЦП, кроме прерывания, можно также отслеживать с помощью входного флага FI, используя ассемблерные конструкции сигнального процессора `IF FLAG_IN JUMP/CALL xxx` или `IF NOT FLAG_IN JUMP/CALL xxx`.

Управление адресом (номером) канала и коэффициентом усилениея сигнала происходит с использованием двух управляющих регистров:

- буферный регистр адреса/усиления канала;
- выходной регистр адреса/усиления канала.

Информация, находящаяся в выходном регистре, определяет адрес канала и коэффициент усиления для текущего преобразований АЦП. Формат этой информации полностью соответствует формату логического канала, описанному в *§ 1.2.2.2.3 "Логический номер канала АЦП"*. Данные из буферного регистра переписываются в выходной регистр по спадающему фронту сигнала #BUSY АЦП. Таким образом, данные, записанные в буферный регистр при обработки прерывания от выборки N (см. ниже), будут действовать для выборки N+2. Запись данных в буферный и выходной регистры производится через первую ячейку пространства ввода-вывода `IO(1)`, при учете состояния флага **FL0**:

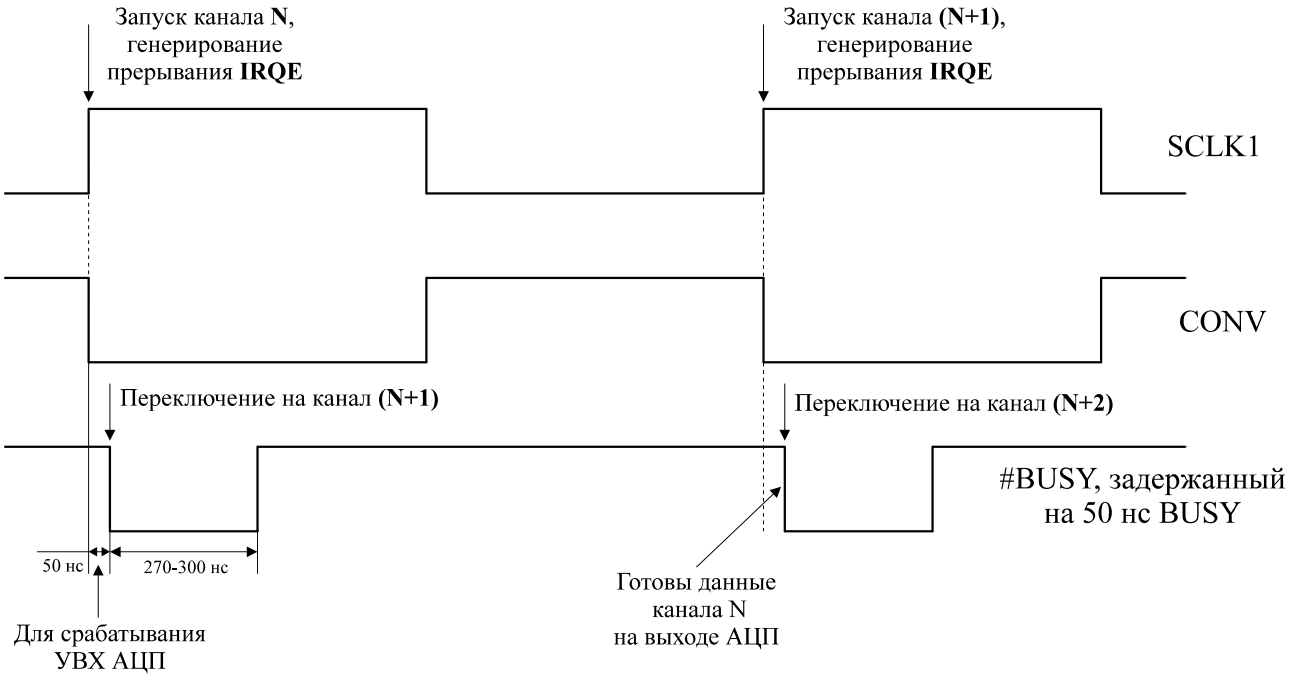
- если **FL0**=0 во время записи в `IO(1)`, то грузится буферный регистр;
- если **FL0**=1 во время записи в `IO(1)`, то грузится выходной регистр.

В качестве сигналов запуска преобразования в АЦП на данной плате используются инвертированные внутренние тактовые синхроимпульсы SCLK1 последовательного порта SPORT1. С помощью регистров управления работой SPORT1, адреса которых в памяти данных DM(0x3FF2) и DM(0x3FF1) соответственно, Вы можете разрешить либо запретить генерацию этих импульсов, а также задавать частоту их следования, определяя, таким образом, частоту работы АЦП. Так частота работы АЦП определяется по следующей формуле:

$$\textbf{ADC\_Rate}=\mathbf{f_q}/(\textbf{SCLKDIV}+1)\ \text{кГц},$$

где $\mathbf{f_q}$=20000.0 кГц – частота установленного на плате кварца, **SCLKDIV** – содержимое регистра DM(0x3FF1). Очень подробное описание работы сериальных портов DSP можно найти в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 5 "Serial Ports", стр. 5-1, Analog Devices, Inc., Third Edition, September 1995*.

Временные диаграммы работы АЦП приведены на следующем рисунке:



Теперь попробуем проиллюстрировать все выше сказанное примером:

```
{ Для начала отконфигурируем SPORT1                                    }
{ SPORT0 - disable, SPORT1 - disable, SPORT1 – not serial port         }
    AR=0x0000;
    DM(Sys_Ctrl_Reg)=AR;      { 0x3FFF - System Control Register       }
{ ********************************************************************* }
{ Set SPORT1 for start of ADC chip                                     }
{ Serial Clock Divide Modulus                                          }
    AR = 99;                  { частота запуска АЦП (SCLK1)            }
    DM(Sport1_Sclkdiv) = AR;  { 0x3FF1 - Serial Clock Divide Modulus   }
{ Receive Frame Sync Divide Modulus                                    }
    AR = 0xF;               { may be any number: - not used            }
    DM(Sport1_Rfsdiv)=AR;  { 0x3FF0-Receive Frame Sync Divide Modulus }
{ Control word for SPORT1: SCLK1 – external (остановим АЦП)            }
{ high level, alternate external receive frame on each word(16 bit) - not used  }
{ high level, alternate external transmit frame on each word(16 bit) - not used }
    AR = 0x3C1F;              { 0111 1100 0001 1111                    }
    DM(Sport1_Ctrl_Reg) = AR; { 0x3FF2 – SPORT1 Control Register       }
{ ********************************************************************* }
{ задержка на 0.4 мкс, чтобы оцифрился последний предыдущий отсчет    }
    cntr=16;
    DO DelayLoop UNTIL CE;
DelayLoop: NOP;
{ очистим все запросы на прерывания                                    }
    IFC = 0xFF; NOP;
{ разрешим прерывания IRQE                                             }
    IMASK=0x10; NOP;
{ зададим усиление и номер канала для текущего и следующего отсчетов, }
{ предполагая наличия циклического буфера с управляющей таблицей      }
```

```
{ (I2, M2, L2)                                                          }
    AR=DM(I2, M2);
    SET FL0;
    IO(1)=AR;
    RESET FL0;
    AR=DM(I2, M2);
    IO(1)=AR;

{ задержка на 0.4 мкс, чтобы установился аналоговый тракт               }
    cntr=16;
    DO DelayLoop1 UNTIL CE;
DelayLoop1: NOP;

{ включим клоки, запустив АЦП, т.е. сделаем SCLK1 внутренним            }
    AR = 0x7C1F;                { 0111 1100 0001 1111                   }
    DM(Sport1_Ctrl_Reg) = AR;   { 0x3FF2 - SPORT1 Control Register      }

{ Все! Теперь можно ждать прихода прерывания IRQE                       }
{ Первое пришедшее прерывание IRQE будет левым и его надо пропустить    }
{ Частота работы АЦП: ADC_Rate=20000.0/(99+1)=200.0 кГц                 }

    . . . . .

{ Обработчик прерывания IRQE может содержать следующие строчки          }

    AR=DM(I2, M2);      { зададим усиление и номер канала для           }
    IO(1)=AR;           { следующего отсчета                            }
    . . . . .
    AR=IO(1);           { теперь в AR находится отсчет с АЦП            }
```

1.6.6. ЦАП

Управление микросхемой двухканального 12[ти] битного ЦАП **AD7249** (техническое описание можно найти на сайте www.analog.com), которая может быть установлена на плате по Вашему желанию, сигнальный процессор осуществляет с помощью последовательного порта SPORT0. Для этого необходимо его запрограммировать надлежащим образом, а именно:

- ✓ длина слова (word length) – 16 бит;
- ✓ тактовые синхроимпульсы (serial clocks) – внутренние с периодом не менее 0.4 мкс;
- ✓ кадровая синхронизация приема каждого слова;
- ✓ внутренняя кадровая синхронизация приема;
- ✓ альтернативная кадровая синхронизация приема;
- ✓ кадровый сигнал приема активен по низкому уровню;
- ✓ кадровая синхронизация передачи каждого слова;
- ✓ внешняя кадровая синхронизация передачи;
- ✓ альтернативная кадровая синхронизация передачи;
- ✓ кадровый сигнал передачи активен по низкому уровню.

Выводы сигнального процессора для кадровой синхронизации приема и передачи (выводы RFS0 и TFS0 соответственно) объединены, т.е. внутренне генерируемые сигналы кадровой синхронизации приема являются сигналами внешней кадровой синхронизации передачи. Частоту генерирования сигналов на выводе RFS0 можно изменять в достаточно широких пределах, устанавливая, таким образом, частоту вывода данных на ЦАП. Очень подробное описание конфигурирования сериальных портов DSP можно найти в *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 5 "Serial Ports", стр. 5-1, Analog Devices, Inc., Third Edition, September 1995*

Формат передаваемых в микросхему ЦАП данных приведен в следующей таблице:

Номер бита	Назначение
0÷11	$12^{\text{ти}}$ битный код ЦАП
12	Выбирает номер ЦАП: ✓ если ‘0’, то запись в первый канал ЦАП; ✓ если ‘1’, то запись во второй канал ЦАП.
13÷15	Не используются

Приведем пример подготовки порта SPORT0 для управления ЦАП’ами:

```
{ Для начала отконфигурируем SPORT0 на передачу данных                }
{ SPORT0 - disable, SPORT1 - disable, SPORT1 – not serial port        }
    AR=0x0;
    DM(Sys_Ctrl_Reg)=AR;      { 0x3FFF - System Control Register      }

{ ************************************************************** }
{   Set SPORT0 for transmit digital codes in DAC                      }
{   SCLK0 and Receive Frame - internal, word = 16 bits                }
{   Transmit Frame - external                                         }

{   Serial Clock Divide Modulus                                       }
    AR=7;                     { SCLK0 – internal, T=400 ns            }
    DM(Sport0_Sclkdiv) = AR;  { 0x3FF5 - Serial Clock Divide Modulus  }

{Receive Frame Sync Divide Modulus                                    }
    AR = 49;            { Определяет частоту вывода отсчетов с ЦАП'а  }
    DM(Sport0_Rfsdiv)=AR;   {0x3FF4-Receive Frame Sync Divide Modulus}

{ Control word for SPORT0: SCLK0 - internal                           }
{ low level, alternate internal receive frame on each word(16 bit)    }
{ low level, alternate external transmit frame on each word(16 bit)   }
    AR = 0x7DCF;              { 0111 1101 1100 1111                   }
    DM(Sport0_Ctrl_Reg) = AR; { 0x3FF6 - SPORT1 Control Register      }
{ ************************************************************** }

{ SPORT0 - enable, SPORT1 - disable, SPORT1 – not serial port         }
    AR = 0x1000;              { 0001 1100 0000 0000                   }
    DM(Sys_Ctrl_Reg) = AR;    { 0x3FFF - System Control Register      }

{ Мы установили частоту вывода данных на ЦАП равной                   }
{ fq/((7+1)*(49+1))=50 Кгц, где fq=20000 кГц - частота кварца         }
{ Теперь с этой частотой будут приходить прерывания SPORT0 Transmit,  }
{ обработчик, которого каждый раз должен записывать в регистр         }
{ передачи TX0 очередное значение, посылаемое в микросхему ЦАП        }

    . . . . .

{ Мы же сейчас просто установим нулевой уровень на первом ЦАП’е       }
    TX0=0x0;

    . . . . .

{ А сейчас установим уровень 5.0 В (код 2047) на втором ЦАП’е         }
    TX0=0x17FF;
```

1.6.7. Управление ТТЛ линиями

Управление цифровыми линиями осуществлять достаточно просто. Любая запись какого-либо числа по адресу 0x0 в пространстве ввода-вывода (I/O Memory Space) сигнального процессора устанавливает состояния всех 16[ти] выходных линий на разъёме ***PLD-40*** в соответствие с битами записываемого значения. Любое чтение по адресу 0x0 из пространства ввода-вывода (I/O Memory Space) сигнального процессора считывает состояние всех 16[ти] входных линий на разъёме ***PLD-40*** платы. Подробности разъёма см. *"Руководство пользователя", § 3.3.2. "Внутренний разъём для подключения цифровых сигналов"*. Описание I/O Memory Space можно найти, например, в оригинальной книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 10.6.4 "ADSP-2181 I/O Memory Space", стр. 10-32, Analog Devices, Inc., Third Edition September 1995*.

В платах *Rev. C* имеется возможность программным образом управлять доступом к выходным цифровым линиям. (подробнее см. *§ 1.2.4.5. 'Особенности плат Rev. C'*). Для этого используется флаг сигнального процессора **PF0**, который должен быть заблаговременно настроен как выходной. Сброс флажка **PF0** в состояние лог. '0' разрешает выходные линии платы, а установка в лог. '1' – переводит их в '*третье*' стостояние.

Рассмотрим простой пример работы с цифровыми линиями:

```
{ прочитаем текущее состояние флагов PFx                    }
    AR=DM(Prog_Flag_Data);
{ разрешим доступ к выходным линиям, сбросив флажок PF0     }
    AR=CLRBIT 0 OF AR; DM(Prog_Flag_Data)=AR;
{ теперь считаем входные цифровые линии                     }
    AR=IO(0);
{ установим прочитанные состояния на выходных линиях        }
    IO(0)=AR;
```

1.6.8. Генерирование прерываний в PC

На плате предусмотрена возможность генерирования прерываний в PC. Эту способность можно использовать, когда Вам требуется проинформировать компьютер о наступлении какого-либо события. Например, в фирменном драйвере генерирование прерывания в PC применяется по мере соответствующего заполнения FIFO буфера АЦП, информируя Вас о готовности данных. Номер прерывания для каждой платы назначается BIOS компьютера или *Windows* и его можно узнать, прочитав, например, поле **InterruptNumber** в соответствующей структуре типа *BOARD_INFO* (см. *§ 1.2.5.1.1 "Структура BOARD_INFO"*). Для генерации прерываний из платы в PC можно использовать следующие флаги сигнального процессора:

- ✓ **FL2** (платы *Rev. B*).
- ✓ **FL2** и **FL1** (платы *Rev. C*).

В штатном *LBIOS* флаг **FL2** используется для генерации прерываний по мере заполнения FIFO буфера АЦП, а флаг **FL1** – FIFO буфера ЦАП, информируя PC о готовности тех или иных данных. При работе с прерываниями необходимо, чтобы соответствуюший флаг **FLx** постоянно находился в состоянии лог. '1', а переход '1'-'0'-'1' осуществлялся только тогда, когда надо вызвать прерывание в компьютере. **Внимание!!!** Прерывание в компьютере будет сгенерировано только в случае, если со стороны PC оно будет разрешено с помощью API функции ***INIT_INTERRUPT_PLX()*** (см. *§ 1.2.5.6.1 "Установка обработчика прерываний"*).

Приведем пример генерирования одного прерывания в компьютер:

```
{ Сбросим флаг FL2 (предполагается, что он находился в лог. '1')   }
    RESET FL2;
{ организуем небольшую задержку                                     }
    NOP; NOP;
{ вернем флаг FL2 в исходное состояние, т.е. в лог '1'              }
    SET FL2;
```

1.6.9. Внешняя синхронизация сигнального процессора

При необходимости дополнительной внешней синхронизации DSP можно использовать следующие ТТЛ совместимые сигналы на плате:

- ✓ линия **TRIG** на внешнем разъёме ***DRB-37M***, которая подключена к ножке прерывания *IRQ1* сигнального процессора. В фирменном драйвере это прерывание отконфигурировано для работы по фронту и используется для цифровой синхронизации ввода данных с АЦП.
- ✓ линия **INT** на разъёме ***PLD-40***, которая подключена к ножке прерывания *IRQ0* сигнального процессора (по уровню или по фронту). В фирменном драйвере эта линия не используется.

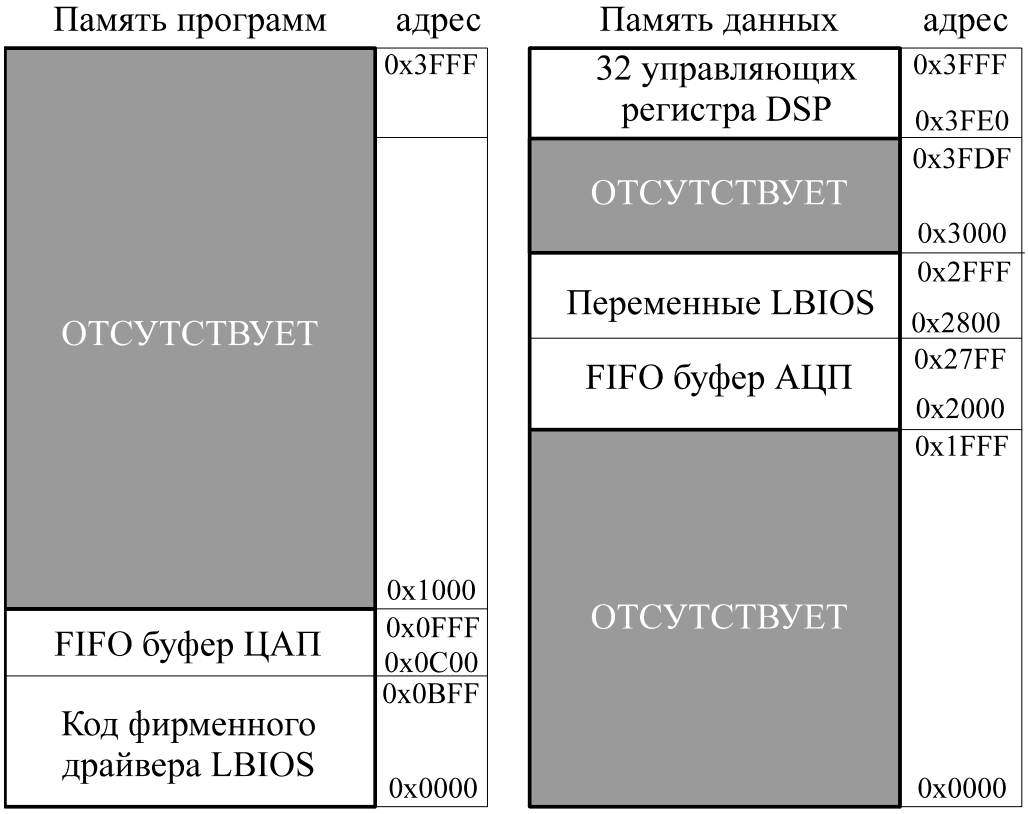
Данные прерывания могут быть сконфигурированы на работу, как по фронту, так и по уровню (за это отвечает регистр DSP под названием *ICNTL*). Если прерывание работает по фронту, то оно генерируется при отрицательном перепаде импульса (⎍⎽⎍) длительностью не менее 50 нс. В случае же конфигурации по уровню, соответствующая линия прерывания должна оставаться в активном низком уровне до тех пор, пока процессор не начнет обслуживание данного прерывания. В обработчике прерывания соответствующую ему линию **обязательно** надо сбрасывать в исходное высокое состояние, чтобы это прерывание не обрабатывалось повторно.

3. ПРИЛОЖЕНИЯ

1.7. ПРИЛОЖЕНИЕ А

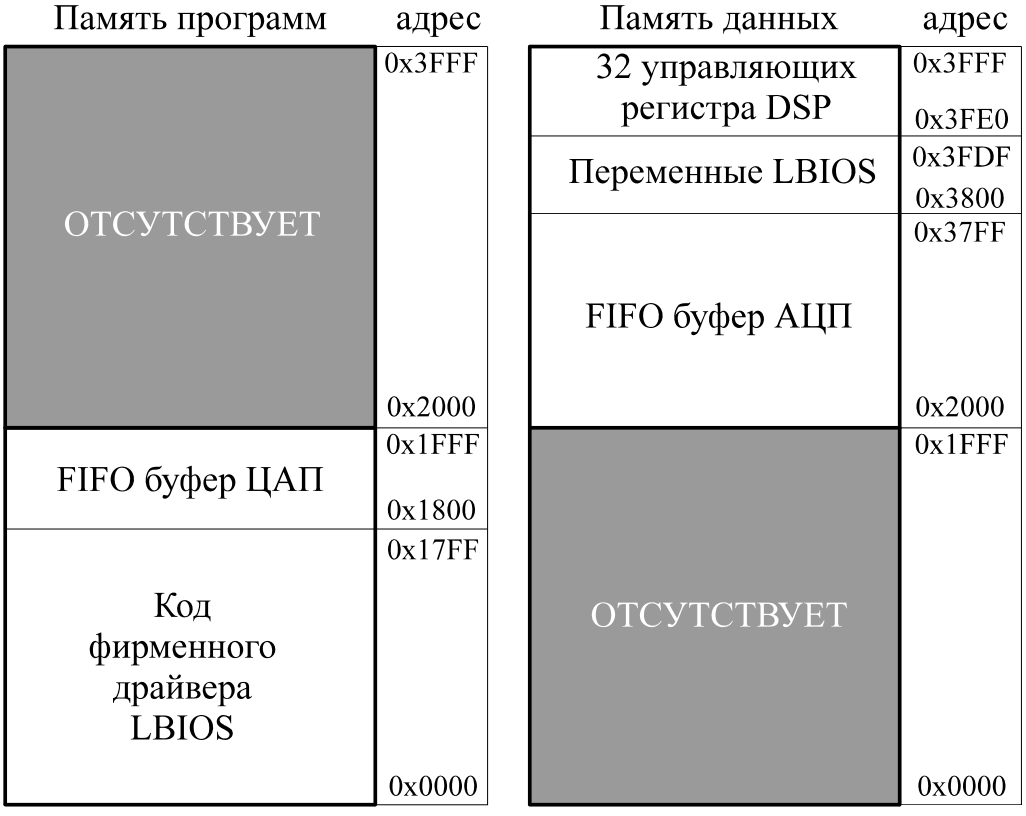
Структура памяти сигнальных процессоров семейства ADSP-218x

Карты распределения памяти программ и памяти данных для различных типов сигнальных процессоров и расположение программных блоков *LBIOS*, настроенного на соответствующий тип DSP, приведены на следующих рисунках:



Карта распределения памяти для ADSP-2184

Карта распределения памяти для ADSP-2185

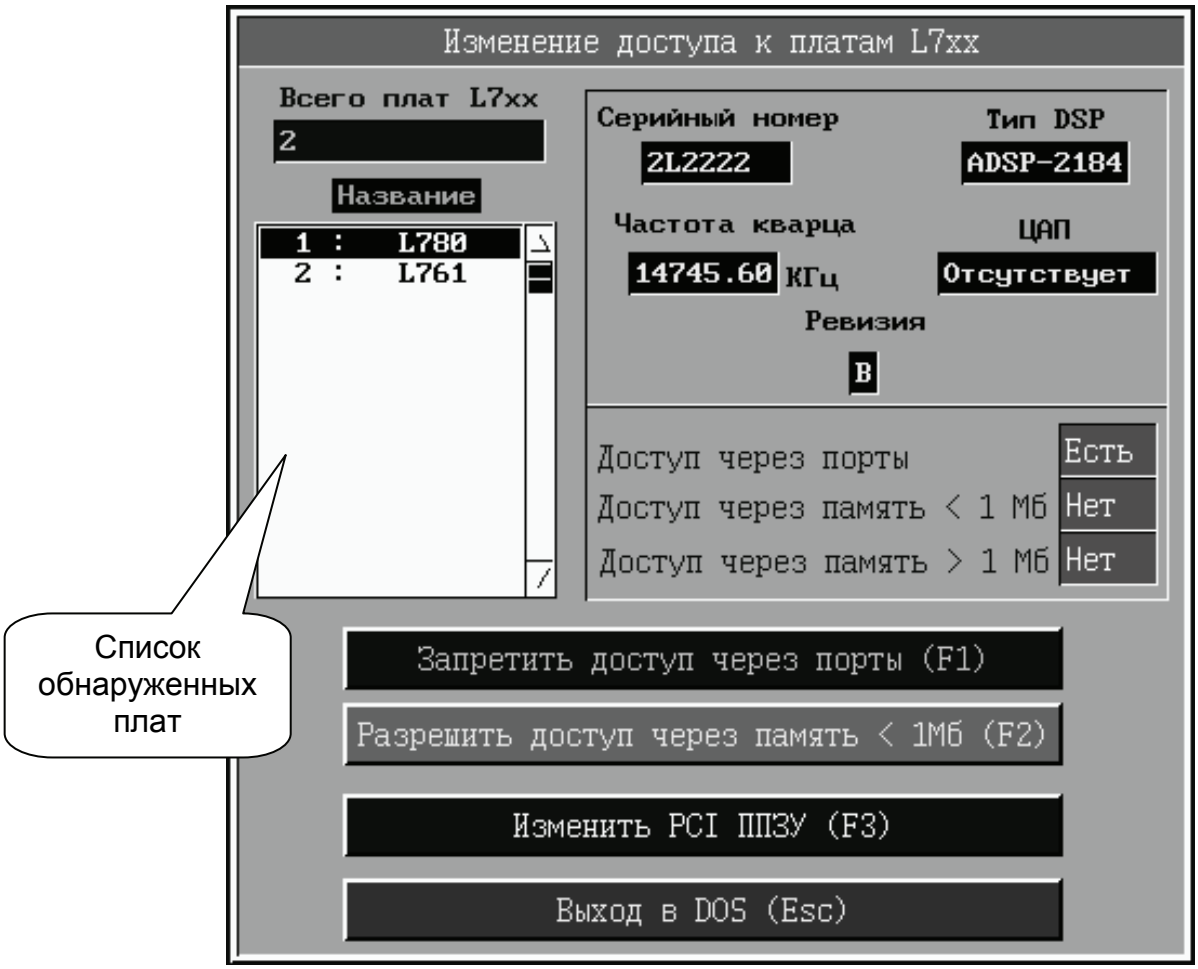


Карта распределения памяти для ADSP-2186

1.8. ПРИЛОЖЕНИЕ В

Утилита CHIOMEM.EXE

Доступом к платам серии *L-7xx* как через порты ввода-вывода, так и через память ниже 1 Мб можно полностью управлять с помощью утилиты `\Utils\CHIOMEM\CHIOMEM.EXE`. Предварительно настоятельно рекомендуется прочитать файл `Readme`. Главная панель программы представлена ниже на рисунке:



Для начала Вам необходимо выбрать нужную плату из списка обнаруженных плат. Идентифицировать эту плату можно по серийному номеру, отображаемому в соответствующем окошке. Тогда в окошках *Доступ через порты*, *Доступ через память < 1Мб* и *Доступ через память > 1Мб* будет информация о возможности доступа через соответствующие ресурсы. Управлять (разрешать и/или запрещать) доступом к плате через порты или память ниже 1 Мб можно с помощью "горячих" клавиш F1 и F2 соответственно.

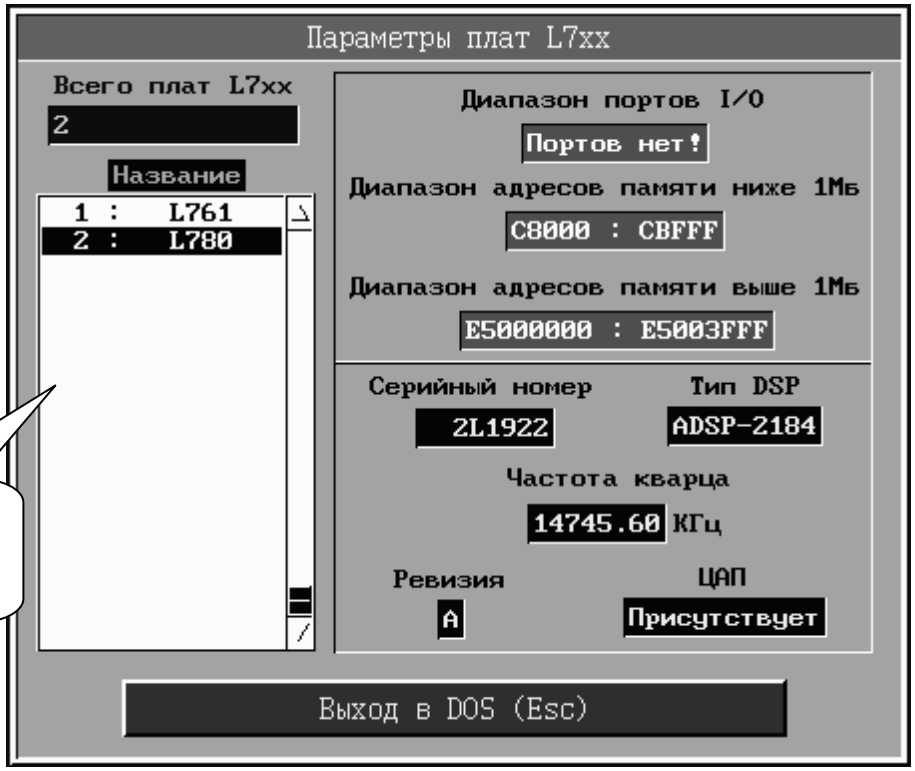
Если утилита обнаружит необходимость в модификации PCI ППЗУ, где хранятся параметры работы микросхемы **PCI**-интерфейса с самой шиной PCI, то появится доступ к клавише *Изменить PCI ППЗУ (F3)*. В этом случае следует **обязательно** воспользоваться данной опцией для надлежащего исправления содержимого PCI ППЗУ.

Для того, чтобы любые изменения сделанные с помощью данной утилиты возымели действие **необходимо** осуществить перезапуск Вашего компьютера с помощью кнопки *RESET* на системном блоке.

1.9. ПРИЛОЖЕНИЕ C

Утилита PARAMS.EXE

Диапазоны ресурсов, выделяемые платам серии *L-7xx*, можно узнать, запустив на выполнение утилиту `\Utils\PARAMS\PARAMS.EXE`. Предварительно очень рекомендуется прочитать файл `Readme`. Диалоговая панель этой программы представлена на следующем рисунке:



Для начала Вам необходимо выбрать нужную плату из списка обнаруженных плат. Идентифицировать эту плату можно по серийному номеру, отображаемому в соответствующем окошке. Тогда в соответствующих окошках *Диапазон портов I/O*, *Диапазон адресов памяти ниже 1Мб* и *Диапазон адресов памяти выше 1Мб* будет информация о диапазонах ресурсов, выделенных данной плате.

1.10. ПРИЛОЖЕНИЕ D

Утилита BIN3PCI.EXE и формат файла .BIO

Утилита BIN3PCI.EXE предназначена для преобразования формата файла отображения в памяти (memory image file), сформированного редактором связей DSP ld21.exe, в формат .BIO, более удобный для процесса загрузки в сигнальный процессор. Формат файла отображения в памяти (memory image file) а деталях описан в *"ADSP-2100 Family Assembler Tools & Simulator Manual", Appendix B "File Format", B.2 "Memory Image File (.EXE)", Analog Devices, Inc., Second Edition, November 1994*. Для выполнения процедуры преобразования формата, например файла отображения в памяти L761.exe в файл L761.bio, в командной строке необходимо набрать

```
bin3pci L761.exe
```

Файл .BIO содержит массив слов типа **int** (в *C++*), формат которого следующий:

Индекс	Назначение
0	Общее количество слов (**NPM**) типа **int** (в *C++*), которые надо грузить в память программ DSP, начиная с адреса PM(0x0)
1	Старшие 16 бит из 24ого битного слова памяти программ, загружаемого по адресу PM(0x0)
2	Младшие 8 бит из 24ого битного слова памяти программ, загружаемого по адресу PM(0x0)
3	Старшие 16 бит из 24ого битного слова памяти программ, загружаемого по адресу PM(0x1)
4	Младшие 8 бит из 24ого битного слова памяти программ, загружаемого по адресу PM(0x1)
. . . . .	
NPM-1	Старшие 16 бит из 24ого битного слова памяти программ, загружаемого по адресу PM((**NPM**-2)/2)
NPM	Младшие 8 бит из 24ого битного слова памяти программ, загружаемого по адресу PM((**NPM**-2)/2)
NPM+1	Общее количество слов (**NDM**) типа **int** (в *C++*), которые надо грузить в память данных DSP, начиная с адреса DM(0x2000)
NPM+2	Первое 16^{ти} битное слово, загружаемое по адресу DM(0x2000)
NPM+3	Второе 16^{ти} битное слово, загружаемое по адресу DM(0x2001)
. . . . .	
NPM+NDM+1	Последнее 16^{ти} битное слово, загружаемое по адресу DM(0x2000+(**NDM**-1))

Вся выше перечисленная информация приводится для работы с сигнальным процессором ADSP-2184, у которого 4 КСлов памяти программ и 4 КСлов памяти данных. Если же Вы пишите управляющую программу для другого типа DSP (ADSP-2185 или ADSP-2186), то нужно раском-

ментировать соответствующий раздел в файле `L7??.SEG`, используемый в своей работе утилитой `bin3pci`, и, при желании, изменить в нем параметры BASE и/или SIZE. Тогда может потребоваться изменить начальные адреса загрузки управляющей программы, указанные в таблице выше.