L-CARD

Внешний модуль АЦП/ЦАП на шину USB 1.1

E-440

Техническое описание
и руководство программиста

Москва. 2003 г.

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

тел. (095) 785-95-25
факс (095) 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*	(3432) 71-44-44
Казань:	ООО “Шатл”	*shuttle@kai.ru*	(8432) 38-1600

E-440. Внешний модуль АЦП/ЦАП на шину **USB 1.1** общего назначения.

1. ОБЩЕЕ ОПИСАНИЕ 7

1.1. Введение 7
1.1.1. Структура описания 8
1.1.2. Общий подход к работе с модулем 8
1.1.3. Назначение модуля E-440 9

1.2. Состав изделия 9
1.2.1. Базовый комплект поставки 9
1.2.2. Дополнительные услуги 9
1.2.3. Штатное программное обеспечение 9
1.2.4. Дополнительное программное обеспечение 9

1.3. Технические параметры модуля E-440 10
1.3.1. Аналого-цифровой преобразователь (АЦП) 10
1.3.2. Цифро-аналоговый преобразователь (ЦАП) 11
1.3.3. Цифровые входы и выходы 11
1.3.4. Кабель USB 11
1.3.5. Энергопотребление 13
1.3.6. Внешние факторы 13

1.4. Подготовка к работе 14
1.4.1. Конфигурирование Setup компьютера 14
1.4.2. Порядок подключение модуля к компьютеру 14
1.4.3. Внешний вид модуля E-440 14
1.4.4. Описание внешнего разъема для подключения аналоговых сигналов 16
1.4.5. Описание разъема для подключения цифровых сигналов 17
1.4.6. Схемы подключения аналоговых сигналов 18

1.5. Характерные неисправности и методы их исправления 20

1.6. Техническое сопровождение 20

2. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 21

2.1. Введение 21

2.2. Общие сведения 22
2.2.1. Подключение модуля E-440 к компьютеру 22
2.2.2. DLL библиотека для работы с модулем E-440 23
2.2.3. Загрузка управляющей программы 24

2.3. Используемые термины и форматы данных 25
2.3.1. Термины 25
2.3.2. Форматы данных 25
2.3.2.1. Формат слова данных с АЦП 25
2.3.2.2. Формат слова данных для ЦАП 26
2.3.2.3. Логический номер канала АЦП 26
2.3.3. Формат пользовательского ППЗУ 28
2.3.4. Формат кадра отсчетов 30

2.4. Общие принципы работы с модулем E-440 30
2.4.1. Общий подход к работе с интерфейсными функциями 30
2.4.2. Общая структура LBIOS 33

2.5. Описание штатной DLL библиотеки 35
2.5.1. Переменные и структуры 35
2.5.1.1. Структура ADC_PARS_E440 35
2.5.1.2. Структура DAC_PARS_E440 35
2.5.1.3. Структура MODULE_DESCR_E440 36
2.5.1.4. Переменные LBIOS 36
2.5.1.5. Номера команд LBIOS 39
2.5.2. Функции общего характера 41
2.5.2.1. Получение версии DLL библиотеки 41
2.5.2.2. Получение указателя на интерфейс модуля 41
2.5.2.3. Функция завершения работы с модулем 41
2.5.2.4. Инициализация доступа к модулю 42
2.5.2.5. Освобождение виртуального слота 42
2.5.2.6. Получение названия модуля 43
2.5.2.7. Загрузка LBIOS 43
2.5.2.8. Проверка загрузки модуля 44
2.5.2.9. Получение версии LBIOS 44
2.5.2.10. Сброс DSP на модуле 45
2.5.2.11. Передача номера команд в драйвер LBIOS 45
2.5.2.12. Получение описания ошибок выполнения функций 46
2.5.3. Функции для доступа к памяти DSP модуля 47
2.5.3.1. Чтение слова из памяти данных DSP 47
2.5.3.2. Чтение слова из памяти программ DSP 47
2.5.3.3. Запись слова в память данных DSP 47
2.5.3.4. Запись слова в память программ DSP 48
2.5.3.5. Чтение массива слов из памяти данных DSP 48
2.5.3.6. Чтение массива слов из памяти программ DSP 48
2.5.3.7. Запись массива слов в память данных DSP 49
2.5.3.8. Запись массива слов в память программ DSP 49
2.5.3.9. Чтение переменной LBIOS 49
2.5.3.10. Запись переменной LBIOS 50
2.5.4. Функции для работы с АЦП 51
2.5.4.1. Разрешение работы АЦП 51
2.5.4.2. Запрещение работы АЦП 51
2.5.4.3. Установка параметров работы АЦП 52
2.5.4.4. Получение текущих параметров работы АЦП 54
2.5.4.5. Получение массива данных с АЦП 55
2.5.4.6. Ввод кадра отсчетов с АЦП 56
2.5.4.7. Однократный ввод с АЦП 56
2.5.5. Функции для работы с ЦАП 57
2.5.5.1. Разрешение работы ЦАП 57
2.5.5.2. Запрещение работы ЦАП 57
2.5.5.3. Установка параметров работы ЦАП 58
2.5.5.4. Получение текущих параметров работы ЦАП 58
2.5.5.5. Передача массива данных в ЦАП 59
2.5.5.6. Однократный вывод на ЦАП 60
2.5.6. Функции для работы с внешними цифровыми линиями 61
2.5.6.1. Разрешение выходных цифровых линий 61
2.5.6.2. Чтение внешних цифровых линий 61
2.5.6.3. Вывод на внешние цифровые линии 61
2.5.7. Функции для работы с пользовательским ППЗУ 62
2.5.7.1. Разрешение/запрещение записи в ППЗУ 62
2.5.7.2. Запись слова в ППЗУ 62
2.5.7.3. Чтение слова из ППЗУ 63

2.5.7.4. Чтение служебной информации из ППЗУ 63
2.5.7.5. Запись служебной информации в ППЗУ 63

3. НИЗКОУРОВНЕВОЕ ОПИСАНИЕ МОДУЛЯ E-440 64

3.1. Структурная схема модуля E-440 64

3.2. Организация USB интерфейса 65
3.2.1. Общие сведения о USB 65
3.2.2. Интерфейс AVR с USB шиной 66
3.2.2.1. Функция DeviceIoControl() 67
3.2.2.2. Запрос V_RESET_DSP_E440 68
3.2.2.3. Запрос V_PUT_ARRAY_E440 68
3.2.2.4. Запрос V_GET_ARRAY_E440 69
3.2.2.5. Запрос V_START_ADC_E440 69
3.2.2.6. Запрос V_START_DAC_E440 70
3.2.2.7. Запрос V_COMMAND_IRQ_E440 71
3.2.2.8. Запрос V_GO_SLEEP_E440 71
3.2.2.9. Запрос V_WAKEUP_E440 71
3.2.2.10. Запрос V_GET_MODULE_NAME_E440 71
3.2.2.11. Потоковые пересылки 72

3.3. Интерфейс AVR с DSP 73

3.4. Интерфейс DSP с периферией модуля 74
3.4.1. Общие сведения 74
3.4.2. Создание управляющей программы 75
3.4.3. Загрузка управляющей программы в DSP 76
3.4.4. Порты управления 77
3.4.5. Конфигурирование флагов DSP 78
3.4.6. АЦП, коммутатор и программируемый усилитель 80
3.4.7. Организация сбора данных с АЦП в LBIOS 83
3.4.8. Корректировка данных с АЦП 84
3.4.9. ЦАП 84
3.4.10. Организация работы с ЦАП в LBIOS 87
3.4.11. Управление ТТЛ линиями 87
3.4.12. ППЗУ 87
3.4.13. Внешняя синхронизация сигнального процессора 88

4. ПРИЛОЖЕНИЯ 89

4.1. ПРИЛОЖЕНИЕ A 89

4.2. ПРИЛОЖЕНИЕ B 90

4.3. ПРИЛОЖЕНИЕ C 91

1. ОБЩЕЕ ОПИСАНИЕ

В Н И М А Н И Е !!!!!

Перед подключением к модулю каких-либо сигналов мы настоятельно рекомендуем Вам изучить *§ 1.4.6. "Схемы подключения аналоговых сигналов"*, в котором описываются схемы подключения сигналов. Как показывает наш опыт, более 80% проблем, возникающих при эксплуатации модуля, связаны с неправильным подключением сигналов.

1.1. Введение

Модуль *E-440* является современным универсальным программно-аппаратным устройством для использования со стандартной последовательной шиной **USB (Rev.1.1)** и предназначен для построения многоканальных измерительных систем ввода, вывода и обработки аналоговой и цифровой информации в составе персональных IBM-совместимых компьютеров. Помимо того, что данный модуль можно рассматривать в качестве достаточно недорогого компактного устройства многоканального сбора информации, он, в тоже время, представляет из себя законченную систему с собственным процессором, позволяющую искушенному пользователю реализовать свои собственные специализированные алгоритмы обработки сигналов на уровне программирования установленного на модуле современного цифрового сигнального процессора (DSP) фирмы **Analog Devices, Inc.** — *ADSP-2185M*.

Применение на модуле *E-440* **USB**-интерфейса предоставляет конечному пользователю целый ряд существенных преимуществ. Так при использовании стандартных устройств АЦП-ЦАП для персональных компьютеров, подключаемых через шину **ISA** либо **PCI**, в процессе решения некоторых задач возникает ряд неудобств, состоящих в необходимости установки платы внутрь компьютера, в конфигурировании **Setup** компьютера для корректной работы плат, в невозможности использования таких плат с портативными компьютерами типа **NoteBook**. К такого рода компьютерам стандартные платы АЦП-ЦАП могут подключаться только при помощи дополнительного специального устройства под названием *Docking Station*, довольно громоздкого агрегата, требующего к тому же отдельного внешнего сетевого электропитания. Одним из возможных вариантов решения таких проблем, когда необходимо иметь устройство, которое можно было бы быстро и удобно подключать к стандартным персональным компьютерам, к компьютерам типа **NoteBook**, к промышленным компьютерам, является использование внешних модулей, подключаемых к шине **USB**. На сегодняшний день все современные компьютеры, в том числе типа **NoteBook**, поддерживают данный тип шины. Кроме того, благодаря встроенным линиям питания, обеспечивающим ток до 500 мА, шина **USB** часто позволяет применять устройства без внешнего блока питания (модуль *E-440* как раз не требует внешнего питания). Все подключаемые к шине **USB** устройства конфигурируются автоматически, т.е. реализован так называемый принцип *Plug&Play*, когда операционная система сама определяет тип подключенного устройства и загружает необходимый для данного устройства драйвер. При этом не возникает никаких вопросов о номере используемого прерывания, адресах портов, DMA и т.д. Кроме того, спецификация шины **USB** допускает "горячее" (т.е. "на лету", при включенном питании компьютера) подключение/отключение устройств.

Пусть Вас не пугает, что на модуле установлен цифровой сигнальный процессор: как правило, большинству пользователей не приходится знакомиться с DSP на уровне программирования сигнальных процессоров, поскольку в комплект поставки модулей входят законченные управляющие программы для процессора *ADSP-2185M*, позволяющие осуществлять ввод-вывод с аналоговых каналов в самых различных режимах. В поставляемых программах были реализованы наиболее часто используемые алгоритмы ввода-вывода, поэтому написание собственных программ для сигнального процессора потребуется только при решении сложных специализированных задач, когда возникает, например, необходимость в перенесении Ваших отлаженных алгоритмов с платформы

PC на сигнальный процессор. Так сигнальный процессор может обеспечить ввод аналоговой информации и её анализ в независимом от компьютера режиме с последующим сообщением о результатах анализа.

В том случае, если Вы приобрели модуль только для обеспечения стандартного качественного многоканального ввода аналоговых сигналов в компьютер, Вам не придется изучать язык ассемблер сигнального процессора. Скорее всего, Вам также не придется изучать низкоуровневое программирование модуля, поскольку к нему поставляется законченный набор штатных функции в виде DLL библиотеки, написанной на языке C++.

Таким образом, для работы в среде *Windows'98/2000/XP* в ***ЗАО "Л-Кард"*** созданы специальная DLL библиотека, а также соответствующий драйвер устройства, максимально упрощающие процесс управления модулем *E-440*, т.е. сбором аналоговых данных в реальном масштабе времени, вывод данных на ЦАП и т.д.

Модуль *E-440* обладает следующими функциональными характеристиками:

- ✓ шина USB (Rev. 1.1);
- ✓ современный цифровой сигнальный процессор *ADSP-2185M* фирмы **Analog Devices, Inc.** с тактовой частотой работы 48 МГц;
- ✓ 16 дифференциальных каналов или 32 канала с общей землей для аналогового ввода с возможностью автоматической корректировки нуля;
- ✓ максимальная частота работы $14^{\text{ти}}$ битного АЦП – 400 кГц;
- ✓ два входа для внешней синхронизации при вводе аналогового сигнала;
- ✓ порт цифрового ввода/вывода, имеющий 16 входных и 16 выходных линий;
- ✓ два канала аналогового вывода $12^{\text{ти}}$ битного ЦАП с максимальной *суммарной* частотой 125 кГц (ЦАП устанавливается по требованию);
- ✓ максимальная пропускная способность по шине **USB (Rev. 1.1) –** не более 500 кСлов/с.

1.1.1. Структура описания

Данное техническое описание состоит из трёх частей:

- В первой части описывается подключение *E-440* к компьютеру, технические параметры функционирования модуля, конфигурирование компьютера, подключение источников сигналов и т.д.
- Во второй части описываются штатная библиотека динимической компоновки (*Dynamic Link Library* – DLL) и, в самом общем виде, управляющий программный драйвер *LBIOS* (т.е. драйвер для цифрового сигнального процессора DSP, установленного на модуле). Библиотека DLL и драйвер *LBIOS*, а также их исходные тексты с довольно подробными комментариями, входят в комплект штатной поставки модуля. Все это можно найти на нашем CD-ROM'е в директории `USB\E440`. Достаточно подробное описание библиотеки функций и программного драйвера *LBIOS* предназначено для программистов, собирающихся использовать штатное программное обеспечение в своих проектах.
- Третья часть предназначена для желающих программировать модуль на самом низком уровне и содержит подробное описание функциональной структуры модуля.

1.1.2. Общий подход к работе с модулем

Ниже описана рекомендуемая последовательность действий при подключении модуля к PC:

I. Модуль подключается к соответствующему разъему компьютера с помощью штатно прилагаемого кабеля **USB**. Модуль можно подключать при включенном питании PC.
II. Распаиваются ответные части разъёмов для подключения аналоговых и/или цифровых сигналов в соответствии с описанием (***!!!особое внимание необходимо уделить заземлению сигналов!!!***).
III. Ответная часть разъёма подсоединяется к модулю.
IV. Включается компьютер (если был выключен).
V. Включаются источники сигналов.

VI. Теперь можно запустить требуемое программное обеспечение и работать с ним.
VII. Выключаются источники сигналов.
VIII. Выключается компьютер.

1.1.3. Назначение модуля E-440

Модуль *E-440* предназначен в основном для преобразования входных аналоговых сигналов в цифровую форму с последующей их передачей в персональный ЭВМ типа IBM PC/AT, обладающий **USB (Rev.1.1)** портом.

Стандартный комплект поставки программного обеспечения позволяет:

- Осуществлять многоканальный ввод аналоговых сигналов с частотой АЦП до 400 кГц.
- Осуществлять двухканальный вывод сигналов на ЦАП с суммарной частотой до 125 кГц.
- Управлять в асинхронном режиме 16тью цифровыми входными и 16тью цифровыми выходными линиями.

1.2. Состав изделия

1.2.1. Базовый комплект поставки

Изделие поставляется в следующей комплектации:

1. Модуль *E-440* (без ЦАП);
2. Ответные части разъемов для подключения аналоговых и цифровых сигналов;
3. Стандартный кабель **USB** длиной 1.8 м;
4. Упаковочная коробка;
5. Штатное программное обеспечение (CD-ROM);
6. Техническое описание и инструкция по эксплуатации.

1.2.2. Дополнительные услуги

Дополнительные услуги предусматривают:

1. Установку микросхемы двухканального 12ти битного ЦАП.

1.2.3. Штатное программное обеспечение

В базовый комплект программного обеспечения входят:

I. Управляющий программный драйвер на языке ассемблер ADSP-218x, реализующий все заложенные в модуль аппаратные возможности.
II. Драйвер устройства и DLL библиотека для управления модулем в операционной среде *Windows'98/2000/XP*.
III. Примеры использования штатных библиотечных функций в различных средах программирования:
   - ✓ **Borland C++ 5.02** – в директории `USB\E440\Examples\BC5`;
   - ✓ **Borland C++ Builder 5.0** – в директории `USB\E440\Examples\BCB5`;
   - ✓ **Delphi 6.0** – в директории `USB\E440\Examples\D6`;
   - ✓ **MS Visual C++ 6.0** – в директории `USB\E440\Examples\MSVC6`.

1.2.4. Дополнительное программное обеспечение

Поддержку модуля *E-440* осуществляет такой законченный программный продукт как **L-Graph**, который был специально разработан для демонстрации возможностей ряда изделий (в том числе модуля *E-440*) производства компании ***ЗАО "Л-Кард"***. Фактически утилита **L-Graph** является многоканальным осциллографом–спектроанализатором–регистратором с достаточно простым, интуитивно-понятным интерфейсом. Подробнее см. последний абзац *§ 2.1. "Введение"*.

1.3. Технические параметры модуля E-440

В данном разделе описаны технические параметры АЦП, ЦАП, цифровых линий и внешние условия работы и хранения модуля *E-440*.

1.3.1. Аналого-цифровой преобразователь (АЦП)

На модуле установлена одна микросхема АЦП, на вход которой при помощи набора коммутаторов может быть подан усиленный сигнал с одного из 16 или 32 аналоговых каналов на внешнем разъеме. Типичные характеристики всего входного аналогового тракта приведены в нижеследующей таблице:

Таблица 1. Параметры аналового тракта.

Тип модуля	**E-440**
Количество каналов	16 дифференциальных или 32 с общей землей
Разрядность АЦП	14 бит
Разрядность, рассчитанная по отношению сигнал/шум на заземленном входе PGA при частоте АЦП 400 кГц	Усиление = 1 13.8 бит Усиление = 4 13.8 бит Усиление = 16 13.5 бит Усиление = 64 13.0 бит
Разрядность, рассчитанная по отношению сигнал/(шум+гармоники) полученная при оцифровке синусоидального сигнала частотой 10 кГц с амплитудой 2.5 В при частоте запуска АЦП 400 кГц	Усиление = 4 13.2 бит
Время преобразования	2.5 мкс
Входное сопротивление при одноканальном вводе	Не менее 1МОм
Диапазон входного сигнала	±10 В, ±2.5 В, ±0.625 В, ±0.15625 В
Максимальная частота преобразования	400 кГц
Защита входов	При включенном питании ±30В При выключенном питании ±10В
Интегральная нелинейность преобразования	макс. ±1.5 МЗР
Дифференциальная нелинейность преобразования	Макс. –1 до +1.5 МЗР
Смещение нуля без калибровки	Макс ±4 МЗР

Межканальное прохождение на частоте сигнала 10 кГц при коэффициенте усиления ‘1’ и макс. частоте запуска АЦП*	-78 дБ

* – Типичные зависимости межканального прохождения в зависимости от частоты запуска АЦП при различных коэффициентах усиления приведены в *Приложении А*.

1.3.2. Цифро-аналоговый преобразователь (ЦАП)

На модуле *E-440* может быть дополнительно установлена микросхема двухканального 12ти битного ЦАП. Т.о. на внешнем разъеме модуля имеется 2 выходных аналоговых канала. Параметры функционирования ЦАП приведены в таблице ниже:

Таблица 2. Параметры ЦАП.

Количество каналов	2
Разрядность	12 бит
Максимальная частота преобразования	125 кГц
Время установления	8 мкс
Выходной диапазон	±5В

1.3.3. Цифровые входы и выходы

На модуле имеются цифровые входные и выходные линии ТТЛ-совместимого уровня, которые могут быть использованы пользователем под свои конкретные задачи, например, для управления внешними устройствами и т.д. Параметры цифровых линий приведены ниже в таблице:

Таблица 3. Параметры цифровых линий.

Входной порт	16 бит КМОП, серия HCT
Выходной порт	16 бит КМОП, серия HCT
Напряжение низкого уровня	мин 0 В, макс 0.4 В
Напряжение высокого уровня	мин 2.4 В, макс 5.0 В
Выходной ток низкого уровня (макс)	6 мА
Выходной ток высокого уровня (макс)	6 мА
Входной ток	10 μA

1.3.4. Кабель USB

Для решения достаточно большого круга задач вполне достаточно штатно поставляемого с модулем кабеля **USB** длиной ~1.8 м. Согласно спецификации **USB** для подключения периферийных устройств используется 4х жильный кабель: питание +5 В, сигнальные линии данных D+ и D-, общий провод (корпус). Общий структурный вид кабеля представлен ниже на рисунке:

Как видно из рисунка, максимально разрешенная длина кабеля может составлять 5 м. Таким образом, при решении с помощью модуля *E-440* ряда задач у пользователя вполне вероятно может возникнуть настоятельная необходимость в расположении собственно модуля на расстояниях от 1.8 м до 5 м (а может быть, и больше) от компьютера (при этом стандартно поставляемым кабелем с очевидностью воспользоваться уже не удастся ☹). В этих условиях перед пользователем естественно встает проблема приобретения дополнительных аксессуаров с целью преодоления ограничений подобного рода. В качестве таковых с успехом могут быть использованы следующие средства:

- ✓ кабель длиной до 5 м;
- ✓ активный удлинитель длиной до 5 м;
- ✓ активный разветвитель (hub).

Т.о., если нет необходимости располагать модуль *E-440* на расстояниях свыше 5 м от PC, то самым разумным выглядит приобретение одного обычного кабеля **USB** соответствующей длины. При этом в процессе выбора подходящего кабеля **USB строго необходимо** учитывать следующие два основных условия:

1. Сигнальные линии кабеля должны быть выполнены в виде *экранированной* витой пары с импедансом 90 Ом.
2. Для надлежащей работы модуля (особенно на максимальных расстояниях от PC) также очень важен такой параметр как толщина (поперечное сечение, калибр) используемых силовых жил (т.е. для питания и корпуса) в применяемом кабеле **USB**. Т.е. поперечное сечение этих проводов должна **строго** соответствовать длине используемого кабеля. Упорядоченный набор толщин проводов задаётся так называемым *American Ware Gauge* стандартом или просто *AWG*. Короче, *AWG* - американский стандарт поперечного сечения проводов. Спецификация **USB (Rev. 1.0)** очень четко определяет соответствие длины кабеля в зависимости от калибра силовых проводов, что и отражено в нижеследующей таблице:

Таблица 4. Таблица зависимости длины кабеля USB от калибра силовых проводов

Калибр силовых проводов, AWG	Максимальное погонное сопротивление, Ом/м	Длина кабеля, м
28	0.232	0.81
26	0.145	1.31
24	0.091	2.08
22	0.057	3.33
20	0.036	5.00

Например. если Вы выбрали для работы кабель длиной 5 м, то калибр силовых жил в нем должен соответствовать 20*AWG*.

Как правило, в маркировке кабеля **USB** (т.е. прямо на поверхности самого шнура) присутствует надпись с калибром (выраженным в *AWG*) отдельно для сигнальных линий и отдельно для силовых жил. В принципе это может выглядеть следующим образом:

‘28AWG/1PR AND 20AWG/2C’

или так

‘2C/28AWG AND 2C/20AWG’.

Здесь 28*AWG* – калибр сигнальных линий кабеля **USB**, а 20*AWG* – силовых жил.

Если же Вам настоятельно необходимо расположить модуль *E-440* на расстояниях свыше 5 м от PC, то возможны два варианта развития событий.

Первый вариант – это дополнительное использование активных разветвителей (хабов). Тогда цепочка подключения модуля *E-440* может выглядеть, например, так:

PC—>кабель 5 м—>активный хаб—>кабель 5 м—>модуль *E-440*.

Вот и получилось удаление устройства от компьютера порядка 10 м. Правда у активных хабов есть один маленький недостаток: они требуют для своей работы внешнего питания, что не всегда и не всем удобно.

Второй вариант – это использование активных **USB** удлинителей. Несмотря на слово ‘активный’ в названии, внешнего питания, в отличие от хабов, такие устройства не требуют. Тогда, последовательно соединяя несколько таких усилителей, можно добиться значительного удаления модуля от PC. Так, используя, например, активный удлинитель марки *UAE016* длиной 5 м, можно без особых проблем построить следующую цепочку подключения модуля:

PC—>UAE016—>UAE016—>кабель 1.8 м—>модуль *E-440*.

Таким образом, легко и просто получилось почти 12 метров удаления прибора от компьютера. В ***ЗАО “Л-Кард"*** тестировался только активный удлинитель типа *UAE016*, подробности о котором можно найти на сайте *www.maxxtro.ru*.

1.3.5. Энергопотребление

1. Рабочий режим (ввод аналоговых сигналов, вывод на ЦАП, управление цифровыми линиями и т.п.). В данном состоянии модуль потребляет не более 350 мА.
2. Режим малого энергопотребления. В данном модуле этот режим не реализован.

1.3.6. Внешние факторы

Рабочая температура	от +5°C до +55°C
Температура хранения	от –10°C до +90°C
Относительная влажность	от 5% до 90%

1.4. Подготовка к работе

1.4.1. Конфигурирование Setup компьютера

Для надлежащего взаимодействия модуля *E-440* с хост-компьютером Вам, возможно, потребуется разрешить использование прерывания для работы контроллера шины **USB**. Данную настройку следует искать в ***Setup*** компьютера. В различных компьютерах требуемая опция в ***Setup*** может называться по-разному, но нужно найти что-нибудь похожее на меню *"Advanced"* или "*Advanced Chipset Setup"*. Далее могут следовать вложенные подменю (типа "*PCI Configuration*" и т.п.). В этих подменю Вам необходимо найти раздел, содержащий строчку, похожую на:

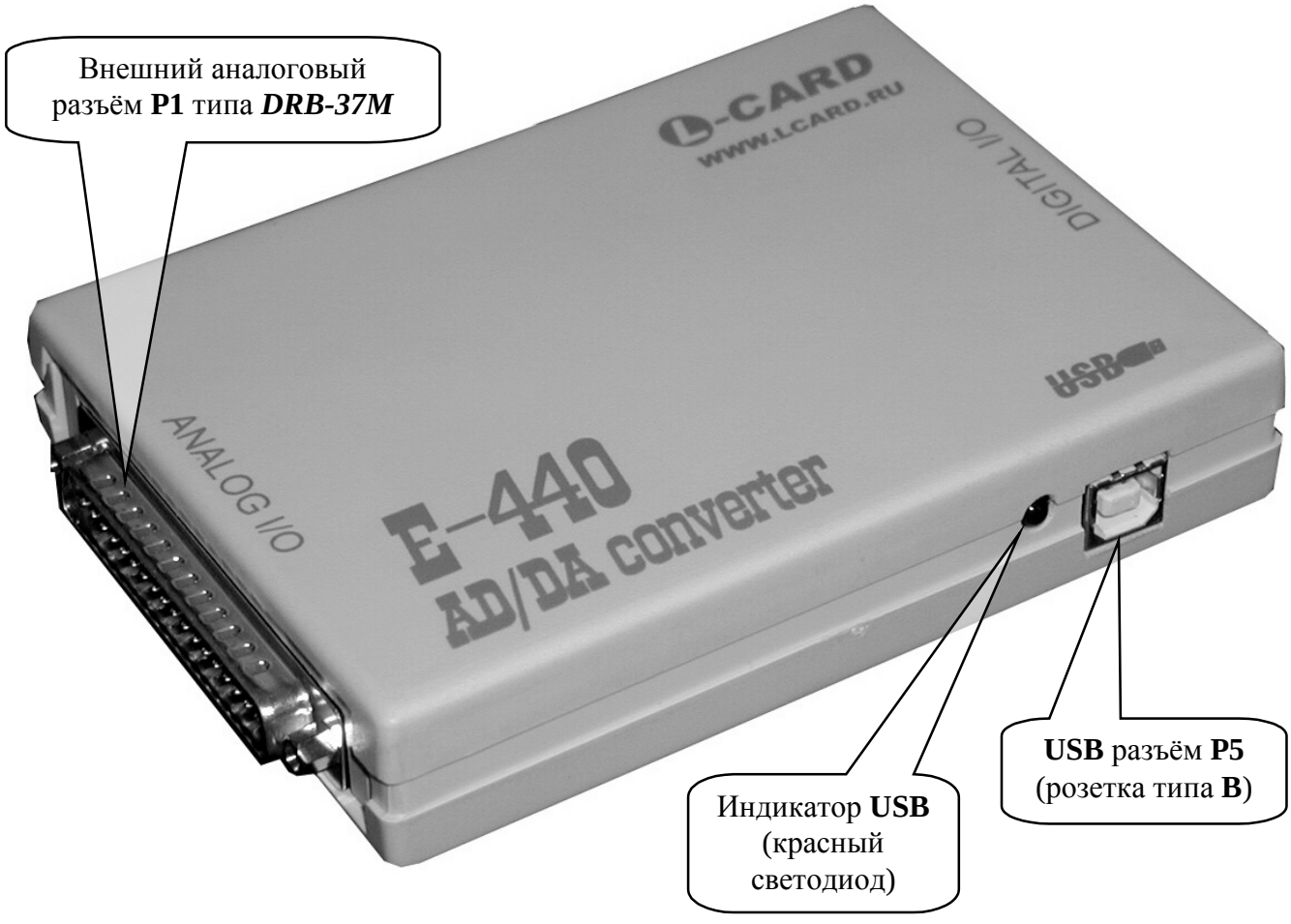
USB IRQ ..... [Enabled]

1.4.2. Порядок подключение модуля к компьютеру

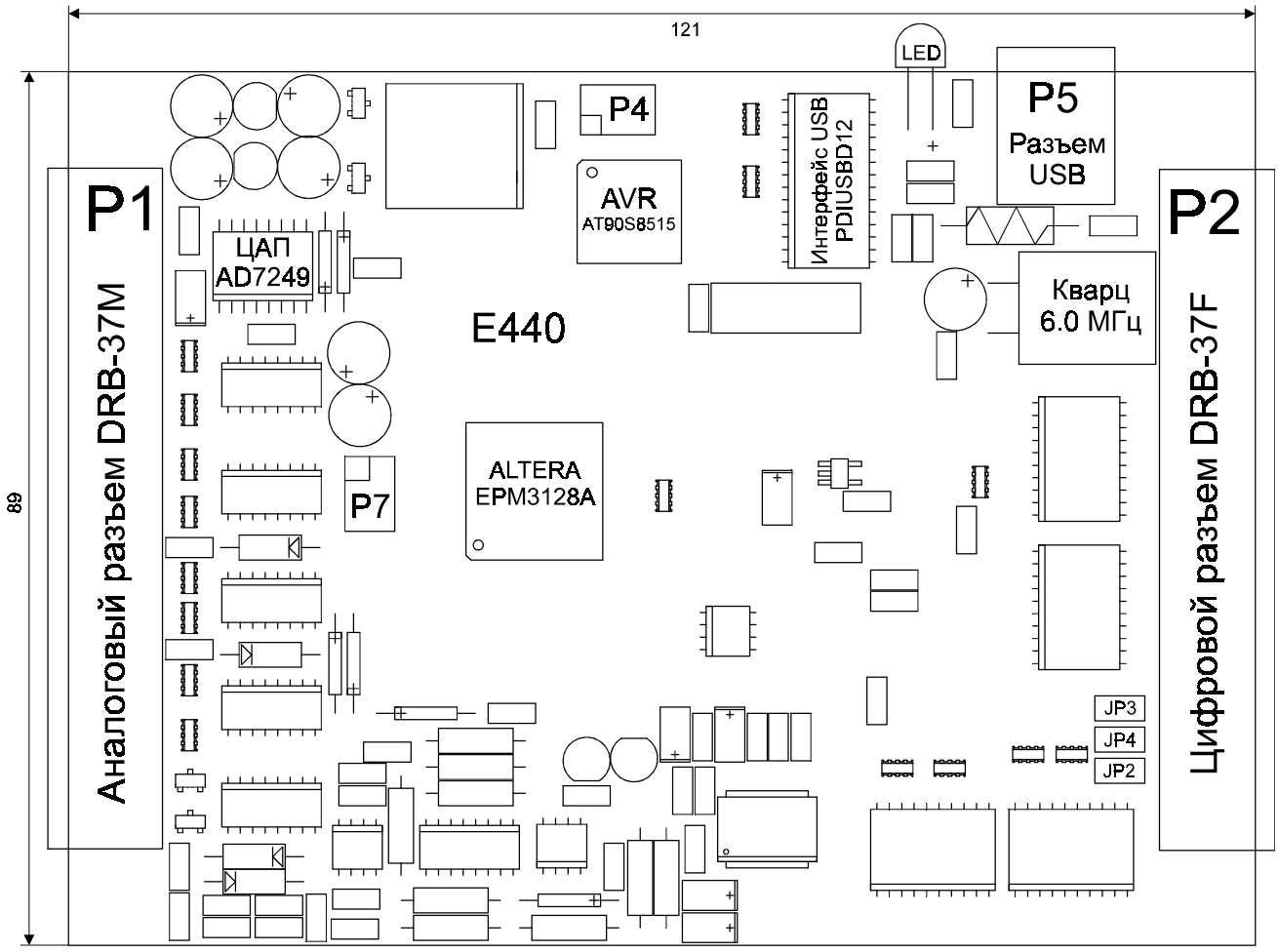
1. Проверьте упаковку и компоненты на отсутствие механических повреждений.
2. Включите питание компьютера, если он был выключен, и загрузите операционную систему *Windows'98/2000/XP*.
3. Теперь с помощью прилагаемого кабеля **USB** можно подключать модуль к компьютеру. Проконтролировать правильность распознавания операционной системой подключенного модуля можно в *"Device Manager"*. Там в разделе *"Universal Serial Bus controllers"* должно появиться устройство *"E-440 Board"*.
4. Установите штатное ПО с прилагаемого CD-ROM'а (библиотека, примеры и т.д.)
5. **ВСЕ!!!** Можно приступать к работе с модулем.

1.4.3. Внешний вид модуля E-440

На рисунке ниже представлен общий внешний вид модуля *E-440*.



А на этом рисунке представлен лицевой вид печатной платы модуля *E-440*.



1.4.4. Описание внешнего разъема для подключения аналоговых сигналов

На внешний разъем ***DRB-37M*** модуля выведены линии аналогового ввода/вывода (**X** означают не инвертирующие входы, **Y** – инвертирующие), описание значений которых приведены в таблице

Таблица 5. Внешний аналоговый разъем DRB-37M

N линии	Назначение	N линии	Назначение
1	**DAC1** – выход первого канала ЦАП	20	**TRIG** – вход внешней ТТЛ синхронизации сигнала
2	**DAC2** – выход второго канала ЦАП	21	**GND32** – общий повод для 32^{x} канального режима
3	**AGND** – аналоговая земля	---	------
4	Вход Y16	22	вход X16
5	вход Y15	23	вход X15
6	вход Y14	24	вход X14
7	вход Y13	25	вход X13
8	вход Y12	26	вход X12
9	вход Y11	27	вход X11
10	вход Y10	28	вход X10
11	вход Y9	29	вход X9
12	вход Y8	30	вход X8
13	вход Y7	31	вход X7
14	вход Y6	32	вход X6
15	вход Y5	33	вход X5
16	вход Y4	34	вход X4
17	вход Y3	35	вход X3
18	вход Y2	36	вход X2
19	вход Y1	37	вход X1

1.4.5. Описание разъема для подключения цифровых сигналов

Выводы внешнего разъема ***DRB-37F*** для подключения цифровых ТТЛ-совместимых сигналов имеют назначения, приведенные в следующей таблице:

Таблица 6. Внешний цифровой разъем DRB-37F.

N линии	Назначение	N линии	Назначение
1	IN1	2	IN2
3	IN3	4	IN4
5	IN5	6	IN6
7	IN7	8	IN8
9	IN9	10	IN10
11	IN11	12	IN12
13	IN13	14	IN14
15	IN15	16	IN16
17	Digital GND	18	+3.3 В
19	**INT** – внешнее, ТТЛ совместимое, прерывание для DSP	20	OUT1
21	OUT2	22	OUT3
23	OUT4	24	OUT5
25	OUT6	26	OUT7
27	OUT8	28	OUT9
29	OUT10	30	OUT11
31	OUT12	32	OUT13
33	OUT14	34	OUT15
35	OUT16	36	Digital GND
37	+5 В	-----	-----

Внимание!!! Подключение к данному разъему можно производить только при выключенном питании модуля *E-440*. Необходимо внимательно следить за тем, чтобы в процессе эксплуатации не было случайных замыканий между контактами разъема *Digital GND* и *+5В* или *+3.3В*, иначе модуль может выйти из строя!!!

1.4.6. Схемы подключения аналоговых сигналов

Перед подключением к модулю каких-либо источников сигнала необходимо обеспечить общий контур заземления Вашего PC и подключаемых к нему приборов. Для этого нужно соединить контакт 3 разъема модуля с контуром заземления Ваших приборов.

При дифференциальной схеме подключения сигнала измеряется разность напряжений между двумя входами канала. При таком подключении обеспечивается подавление шумов, возникающих на соединительных проводах, не менее чем на 60 дБ. Однако следует помнить, что для корректной работы дифференциального усилителя необходимо, чтобы потенциал каждого входа относительно земли (т.н. синфазное напряжение) не превышал установленного входного диапазона. Каждый источник сигнала подключается к соответствующему каналу **ДВУМЯ** проводами. Неинвертирующий вход АЦП подключается к выходной клемме источника, а инвертирующий вход АЦП заземляется непосредственно на корпусе источника сигнала. Общий контур заземления необходимо проводить отдельным проводом. Ниже на рисунках приведены различные схемы подключения сигналов к модулю. При дифференциальном подключении линии **X** означают неинвертирующие входы, а линии **Y** – инвертирующие. Всюду, где речь идет о 32-х канальном подключении сигнала, подразумевается, что линии **X** соответствуют первым 16 каналам модуля, линии **Y** – старшим 16 каналам.

Внимание!!! При работе с модулем следует учитывать, что полоса пропускания входного аналогового тракта намного выше максимальной частоты работы АЦП. Поэтому для получения адекватного преобразования сигнала Вам следует ограничить полосу входного сигнала в соответствии с критерием Найквиста. Т.е. необходимо ограничить полосу сигнала до приемлемого Вам уровня шумов на частоте от $1/(2\ f_{АЦП})$ и выше. Иначе все шумы, лежащие выше $1/(2\ f_{АЦП})$, будут накладываться на полезный сигнал, который должен располагаться ниже $1/(2\ f_{АЦП})$, и не смогут быть отделены от него при последующей обработке.

Внимание!!! При работе с модулем необходимо помнить, что при опросе ‘висячих’ каналов, т.е. каналов, которые не подсоединены ни к сигнальному входу, ни к земле, Вы тем не менее можете получить сигналы, аналогичные сигналам на работающих каналах. Поэтому неподключенные к сигналу аналоговые входы необходимо либо заземлять, либо не опрашивать.

Внимание!!! В случае многоканального ввода сигналов приходится учитывать наличие входной емкости коммутаторов аналогового тракта $C_{вх}\approx100$ пФ. Ошибка установления аналогового тракта не превысит ошибки работы самого АЦП, если выполняется следующий критерий:

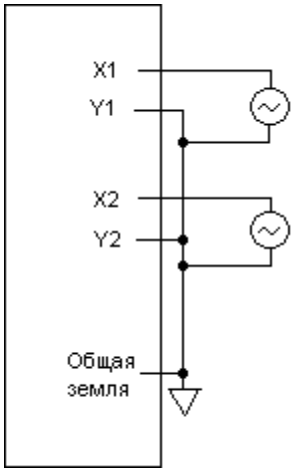
$$R_1 \cdot C_{вх} \le 0.1 \cdot \mathbf{t}_{АЦП},$$

где $R_1 = R_и + R_{защ}$, $R_и$ – выходное сопротивление источника сигнала, $R_{защ}$=1 кОм – сопротивление защиты модуля, $\mathbf{t}_{АЦП}$ – интервал работы АЦП. Максимальное выходное сопротивление источника сигнала должно быть не более 5 кОм.

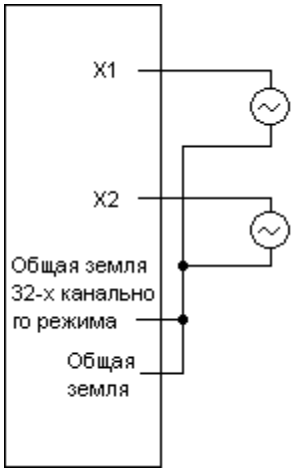
Внимание!!! При работе модуля в псевдодифференциальном режиме (32^{x} канальный режим с общей землей) может наблюдаться некоторое ухудшение подавления синфазной составляющей на частотах сигнала выше 20 кГц. Это может происходить в этом режиме вследствие некоторого разбаланса для положительных входов и общего отрицательного входа.

Источник с плавающим выходом (не заземленным)

16 каналов, дифференциальный режим

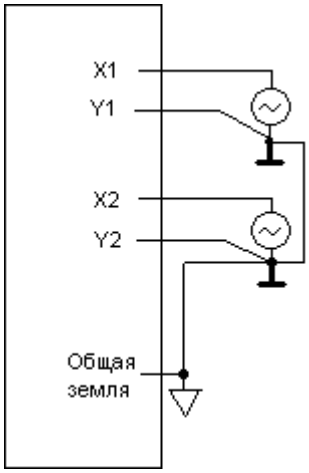


32 канала, псевдодифференциальный режим

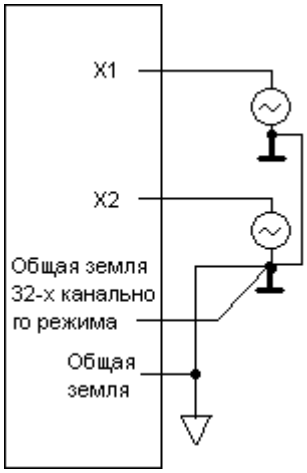


Источник с заземленным выходом

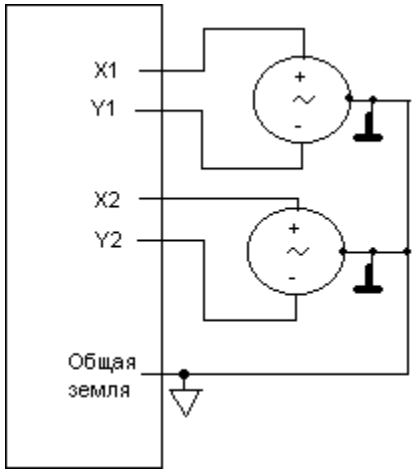
16 каналов, дифференциальный режим



32 канала



Источник с дифференциальным выходом (только для диф. режима)



1.5. Характерные неисправности и методы их исправления

Неисправность, внешнее проявление	Вероятная причина	Метод устранения
Отсутствие сигнала на каналах АЦП	Неправильное подключение к внешнему разъёму модуля	Подключить сигнал в соответствии с описанием внешнего разъема
Повышенный уровень шума	• Неправильное заземление • Неверный номер канала АЦП • Неподключенный канал.	• Обеспечить заземление в соответствии с описанием схем подключения сигналов • Ввести все каналы АЦП и выбрать тот, к которому подключен сигнал
Появление входного сигнала на неподключенных каналах АЦП	-----	Неподключенные к сигналу аналоговые входы необходимо либо заземлить, либо не опрашивать

В случае, если не удается избавиться от неисправности описанными методами, необходимо сообщить об этом в фирму–изготовитель.

1.6. Техническое сопровождение

Любые вопросы и замечания по модулю *E-440* Вы можете задать:

- ✓ по телефону (095) 785-95-25 по рабочим дням с 16.00 до 18.00
- ✓ по e-mail: *lcard@lcard.ru* или *support@lcard.ru*
- ✓ на нашем сайте в разделе ‘Конференция’: *www.lcard.ru/forum.php3?forum=1*

ЗАО “Л-Кард" гарантирует Вам консультации по всем возникшим у Вас вопросам.

2. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

2.1. Введение

Данный раздел описания предназначен для программистов, собирающихся писать свои собственные программы в среде *Windows'98/2000/XP* для работы с модулями *E-440*. В качестве базового языка при написании штатного программного обеспечения нами был выбран язык C++, (а конкретнее, диалект **Borland C++ 5.02**), поскольку он является одним из самых широко распространенных и применяемых языков. Для приобретенного Вами модуля *E-440* ***ЗАО "Л-Кард"*** поставляет драйвер устройства, готовую библиотеку штатных подпрограмм, в виде динамически подключаемой библиотеки (DLL), и ряд законченных примеров. В библиотеку мы попытались включить множество разнообразных полезных функций для облегчения пользователю процедуры написания собственных программ по управлению модулем *E-440*. Данная библиотека позволяет Вам использовать практически все возможности модуля, не вдаваясь в тонкости их низкоуровневого программирования. Если же Вы все-таки собираетесь сами программировать модуль 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, осуществлять конфигурацию FIFO буферов для АЦП и ЦАП и т.д. Мы надеемся, что описываемая ниже библиотека упростит и ускорит написание Ваших собственных программ.

Весь пакет штатного программного обеспечения для модуля *E-440* в среде *Windows'98/2000/XP* находится на прилагаемом к модулю CD-ROM'е в директории `USB\E440` (далее по тексту данного описания все директории указаны относительно нее). Также весь штатный софт можно скачать с нашего *WWW*-сайта *www.lcard.ru* из раздела *"Библиотека файлов"*. Там из подраздела "ПО для внешних модулей" следует выбрать архив `e440v##.exe`, где ## означает номер версии программного обеспечения (на момент написания данного руководства этот архив имеет имя `e440v20.exe`).

Кроме того, при желании можно воспользоваться достаточно широкими возможностями, которые предоставляет законченный программный продукт **L-GRAPH,** для решения ряда общих задач сбора, сохранения и визуализации аналоговой информации**. L-GRAPH** входит в состав *32-разрядной библиотеки* и устанавливается посредством инсталляционной программы `DLL\NEW\Setup.exe` с прилагаемого к модулю компакт-диска. Также дистрибутив программы можно скачать с нашего *WWW*-сайта *www.lcard.ru* из раздела *"Библиотека файлов"* (архив *lgraph.zip*). Данная утилита работает с модулем *E-440* в $16^{ти}$ канальном *дифференциальном* режиме и позволяет, в частности, осуществлять непрерывную регистрацию аналоговой информации в реальном масштабе времени (при этом время ввода ограничено только емкостью Вашего диска). А в общем программа **L-GRAPH** обладает простым, интуитивно-понятным интерфейсом и реализует:

- ✓ $4^{х}$ канальный осциллоскоп;
- ✓ $4^{х}$ канальный спектроанализатор;
- ✓ многоканальный сбор данных в файл;
- ✓ визуализация полученных данных.

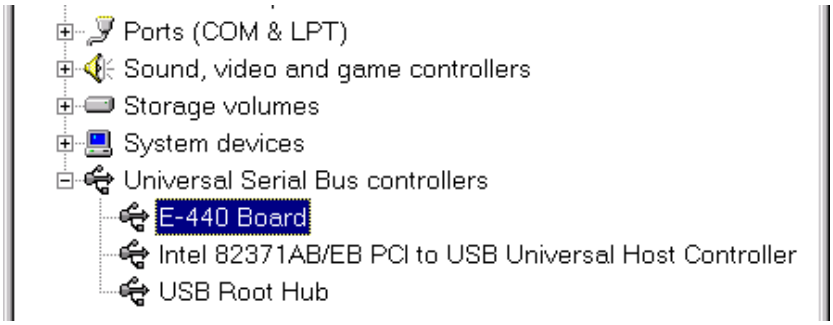
2.2. Общие сведения

2.2.1. Подключение модуля E-440 к компьютеру

Собственно сама процедура аппаратного подключения модуля *E-440* к Вашему компьютеру достаточно тривиальна: необходимо просто соединить **USB** разъем **P5** модуля (см. *§ 1.4.3. "Внешний вид модуля E-440"*) с любым свободным **USB**-портом компьютера при помощи кабеля, входящего в комплект штатной поставки. При этом подразумевается, что на компьютере уже установлена операционная система способная корректно поддерживать функционирование **USB** шины: *Windows'98\2000\XP*. Причем спецификацией **USB** предусматривается как 'горячее' подключение или отключение устройств к/от шины **USB**, так и включение компьютера с уже подключенными устройствами **USB**.

Шина **USB** предоставляет пользователям реальную возможность работать с периферийными устройствами в режиме ***Plug&Play***. Это означает, что стандартом **USB** предусмотрено подключение устройства к работающему компьютеру, автоматическое его распознавание немедленно после подключения и последующая загрузка операционной системой соответствующих данному устройству драйверов.

Т.о. при самом **первом** подсоединении модуля *E-440* к Вашему компьютеру (с помощью прилагаемого стандартного кабеля **USB**) операционная система должна запросить файлы драйвера для впервые подключаемого устройства. Тогда ей необходимо указать *inf*-файл с нашего CD-ROM: `\DRV\Ldevusb.inf`. При этом операционная система сама скопирует файл драйвера в нужное ей место и сделает необходимые записи в своём реестре. После чего операционная система должна произвести так называемую операцию *нумерации* (enumeration, 'переписи'), т.е., грубо говоря, проинициализировать подключенное устройство. Такая процедура нумерации устройств, подключенных к шине **USB**, осуществляется динамически по мере их подключения или отключения без какого-либо вмешательства пользователя или клиентского программного обеспечения. Во время выполнения процесса нумерации индикатор **USB** (красный светодиод расположенный рядом с разъёмом **USB** модуля, см. *§ 1.4.3. "Внешний вид модуля E-440"*) должен непрерывно мигать, а по окончании оной перманентно загореться красным цветом. Это будет говорить о том, что подключенное устройство корректно опознано операционной системой и полностью готово к дальнейшей работе. Дополнительно проконтролировать правильность распознавания операционной системой подключенного модуля можно в *"Device Manager"* (*"Диспетчер устройств"*). Там в разделе *"Universal Serial Bus Controllers"* (*"Контроллеры универсальной последовательной шины USB"*) должно появиться устройство *"E-440 Board"*, как это, например, отображено на рисунке ниже:

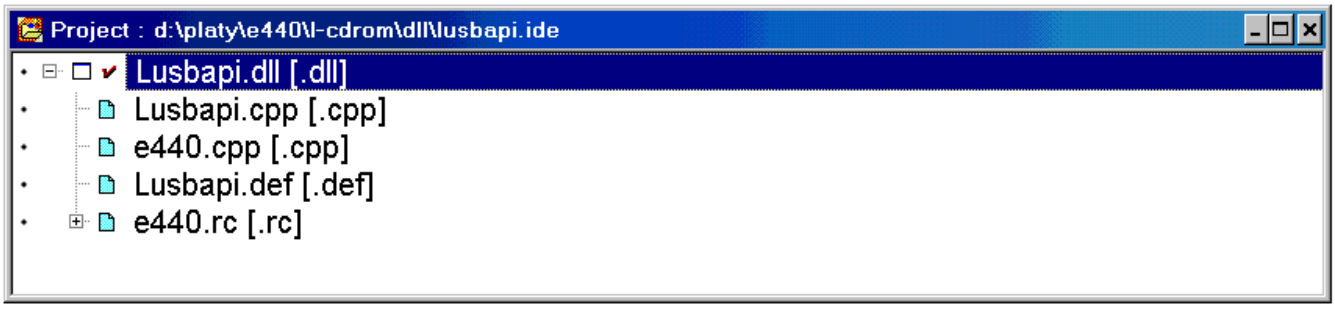


При дальнейшей работе с модулем *E-440* операционная система уже будет знать, где находятся драйвера для данного типа устройства, и будет подгружать их автоматически по мере необходимости.

Кроме того, рекомендуется скопировать файл штатной DLL библиотеки (`Lusbapi.dll`) в директорию `%SystemRoot%\system32`. Это полезно потому, что *Windows'98/2000/XP* при необходимости автоматически производит поиск файлов в указанной директории. Хотя, в принципе, штатная DLL библиотека может находиться в директории Вашего приложения или в одной из директорий, указанных в переменной окружения PATH.

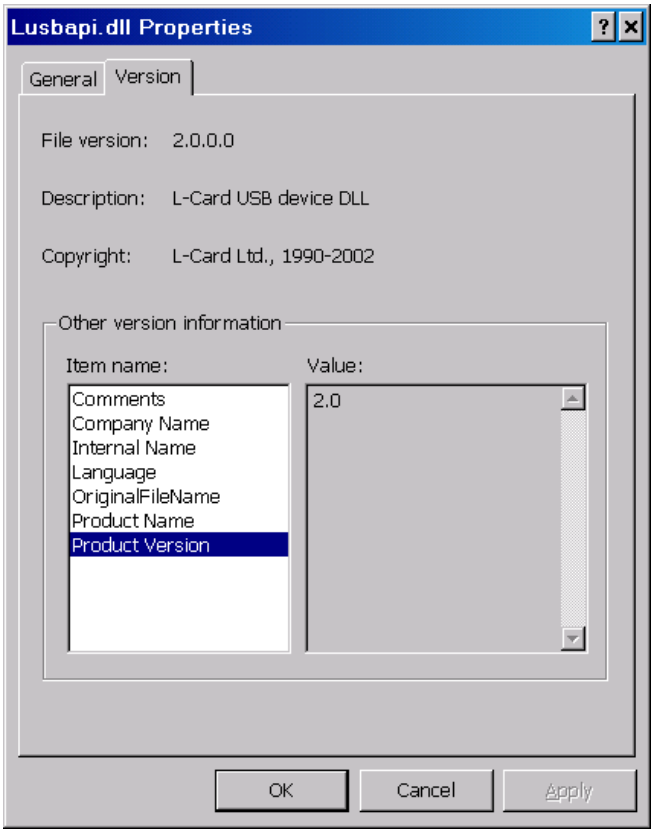
2.2.2. *DLL библиотека для работы с модулем E-440*

Штатная DLL библиотека для модуля *E-440* написана с использованием широко распространенного и весьма доступного языка программирования **Borland C++ 5.02**. Общий вид проекта DLL библиотеки в среде разработки **Borland C++ 5.02** представлен на рисунке ниже:



Собственно, сама библиотека содержит всего две экспортируемые функции, одна из которых *CreateInstance()* возвращает указатель на интерфейс модуля *E-440*. В дальнейшем, используя этот указатель, можно осуществлять доступ ко всем интерфейсным функциям DLL библиотеки (см. исходные тексты примеров). **!!!Внимание!!!** Все интерфейсные функции (кроме *ReadData()* и *WriteData()*), строго говоря, не обеспечивают “потокобезопасную” работу DLL библиотеки. Поэтому, во избежание недоразумений, в многопоточных приложениях пользователь должен сам организовывать, если необходимо, корректную синхронизацию вызовов интерфейсных функций в различных потоках (используя, например, критические участки, мьютексы и т.д.).

В сам файл библиотеки Lusbapi.dll включена информация о текущей версии DLL. Для получения в Вашем приложении сведений о данной версии можно использовать вторую из экспортируемых функций из штатной библиотеки: *GetDllVersion()*. Кроме того, оперативно выявить текущую версию библиотеки можно, используя штатные возможности *Windows*. Например, в *Windows Explorer* щелкните правой кнопкой мышки над файлом DLL библиотеки Lusbapi.dll. Во всплывшем на экране монитора меню следует выбрать опцию ‘*Properties*’, после чего на появившейся панели выбрать закладку ‘*Version*’. На этой закладке в строчке ‘*File version*’ можно без труда прочитать номер версии DLL библиотеки (две старшие цифры). Выглядит это примерно так:



Исходные тексты самой DLL библиотеки Вы можете найти на нашем CD–ROM'е в директории \DLL. Тексты законченных примеров применения интерфейсных функций из штатнгой DLL библиотеки для различных сред разработки приложений можно найти в следующих директориях:

- \Examples\BC5 – для среды **Borland C++ 5.02**;
- \Examples\BCB5 – для среды **Borland C++ Builder 5.0**;
- \Examples\D6 – для среды **Delphi 6.0**;
- \Examples\MSVC6 – для среды **MS Visual C++ 6.0**.

Например, для получения возможности вызова интерфейсных функций в Вашем проекте на **Borland C++** Вам необходимо следующее:

- создать файл проектов (например, для **Borland C++ 5.02**, test.ide);
- добавить в него файл LUSBAPI.LIB;
- создать и добавить в проект Ваш файл с будущей программой (например, test.cpp);
- включить в начало вашего файла заголовочный файл #include "LUSBAPI.H", содержащий описание интерфейса модуля *E-440*;
- в принципе, с помощью функции *GetDllVersion()*, желательно сравнить версию используемой DLL библиотеки с версией текущего программного обеспечения;
- вызвать функцию *CreateInstance()* для получения указателя на интерфейс модуля;
- в общем-то, **ВСЕ!** Теперь Вы можете писать свою программу и в любом месте, используя полученный указатель, вызывать соответствующие интерфейсные функции из штатной DLL библиотеки Lusbapi.dll.

Поклонникам диалекта **Microsoft Visual C++** можно порекомендовать два способа подключения штатной DLL библиотеки к своему приложению:

1. Динамическая загрузка DLL на этапе выполнения приложения (см. исходные тексты примера из директории \Examples\MSVC6\DynLoad)
2. Для статической компоновки штатной DLL предварительно необходимо провести полную перекомпоновку библиотеки в среде разработки **Microsoft Visual C++** с использованием исходных текстов из директории \DLL.

2.2.3. Загрузка управляющей программы

Предположим, что Вы уже успешно подключили модуль к компьютеру и подали на его входы сигналы. При работе с модулем следует учитывать то, что он имеет характерную особенность, отличающюю его от простых устройств ввода-вывода: на нем установлен современный цифровой сигнальный процессор *ADSP-2185M* фирмы **Analog Devices, Inc.** (более подробную информацию на эти DSP можно найти на сайте *www.analog.com*), который необходимо предварительно запрограммировать, т. е. загрузить в него управляющую программу (драйвер, *LBIOS*). В состав штатного программного обеспечения входит законченная управляющая программа, состоящая из одного бинарного файла DSP\E440.bio. Данный файл содержит как выполняемый код управляющей программы, так и сегмент данных для сигнального процессора. В штатной DLL библиотеке для загрузки *LBIOS* в сигнальный процессор модуля имеется специальная интерфейсная функция *LOAD_LBIOS()*, которая аккуратно выполняет процедуру загрузку модуля. Только **ПОСЛЕ** загрузки *LBIOS*а Вы можете полностью управлять модулем, т.е. переводить его в различные режимы работы с АЦП, ЦАП и т. д. Законченные примеры применения интерфейсных функций штатной DLL библиотеки для целей загрузки модуля *E-440* можно найти, например, в директориях \Examples\BCB5\LoadLBios и \Examples\D6\LoadLBios.

2.3. Используемые термины и форматы данных

2.3.1. Термины

Название	Смысл
AdcRate	Частота работы АЦП в *кГц*
ChannelRate	Частота работы аналогового канала в *кГц*
InterKadrDelay	Межкадровая задержка в *млс*
DacRate	Частота работы ЦАП в *кГц*
Buffer	Указатель на целочисленный массив для данных
Npoints	Число отсчетов ввода
AdcChannel	Логический номер аналогового канала АЦП
ControlTable	Управляющая таблица, содержащая целочисленный массив с логическими номерами каналов для последовательного циклического ввода данных с АЦП
ControlTableLength	Длина управляющей таблицы
Address	Адрес ячейки в памяти программ или данных DSP модуля

2.3.2. Форматы данных

2.3.2.1. Формат слова данных с АЦП

Данные, считанные с $14^{\text{ти}}$ битного АЦП модуля *E-440*, представляются в формате знакового целого двухбайтного числа от -8192 до 8191. Точностные пределы кодов АЦП, соответствующие выбранному входному диапазону, приведены в следующей таблице:

Таблица 7. Соответствие кода АЦП напряжению на аналоговом входе

Модуль	Усиление	Код	Напряжение, В	Точность, %
E-440	1; 4; 16; 64	+8000	+MAX	2÷3
		0	0	0.25; 0.3; 0.5; 1.0
		-8000	-MAX	2÷3

где MAX - значение установленного входного диапазона для аналогового канала АЦП (возможные диапазоны для модуля см. в *Таблице 12*).

Вышеуказанные точностные значения приведены для случая, когда *LBIOS* модуля не корректирует поступаемые с АЦП данные с помощью калибровочных коэффициентов (например, хранящихся в ППЗУ самого модуля; см. *§ 2.3.3. "Формат пользовательского ППЗУ"*). Для случая, когда *LBIOS*'у модуля позволено производить такую корректировку входного сигнала, соответствующие точностные параметры АЦП приведены ниже (при температуре 25°C):

Таблица 8. Соответствие кода АЦП напряжению на аналоговом входе при разрешенной корректировке входных данных

Модуль	Усиление	Код	Напряжение, В	Точность, %
E-440	1; 4; 16; 64	+8000	+MAX	0.05; 0.075; 0.1; 0.15
		0	0	
		-8000	-MAX	

где MAX – значение установленного диапазона для входного аналогового канала АЦП (возможные диапазоны для данного модуля см. в *Таблице 12*).

2.3.2.2. Формат слова данных для ЦАП

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

Таблица 9. Формат слова данных ЦАП

Модуль	Номер бита	Назначение
E-440	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 модулем на $12^{\text{ти}}$ битный ЦАП, связан с устанавливаемым на внешнем разъеме напряжением в соответствии со следующей таблицей

Таблица 10. Соответствие кода ЦАП напряжению на внешнем аналоговом разъеме

Модуль	Код	Напряжение
E-440	+2047	+5.0 Вольт
	0	0 Вольт
	-2048	-5.0 Вольт

2.3.2.3. Логический номер канала АЦП

На модуле *E-440* для управления работой входного аналогового каскада определяется такой параметр, как $8^{\text{ми}}$ битный *логический номер канала АЦП* (фактически управляющее слово для АЦП). Именно массив логических номеров каналов АЦП, образующих управляющую таблицу **ControlTable**, задает циклическую последовательность работы АЦП при вводе данных. В состав логического номера канала входят несколько важных параметров, задающих различные режимы функционирования АЦП модуля:

- 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 11. 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 модуля *E-440* 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 12. Коэффициент усиления (биты GS0 и GS1)

Модуль	Бит GS1	Бит GS0	Усиление	Диапазон, В
E-440	0	0	1	±10.0
	0	1	4	±2.5
	1	0	16	±0.625
	1	1	64	±0.15625

Например, для коэффициента усиления 16 диапазон напряжения входного аналогового сигнала будет ±0.625 В.

2.3.3. *Формат пользовательского ППЗУ*

На модуле *E-440* установлено пользовательское ППЗУ емкостью 64 Слова×16 бит. Формат данного ППЗУ представлен на следующем рисунке:

Как видно из рисунка, в первых 32^{x} словах (64 байта) находится служебная область ППЗУ модуля. Порядок расположения в ППЗУ данной информации соответствует структуре *MODULE_DESCR_E440* (см. *§ 2.5.1.3. "Структура MODULE_DESCR_E440"*). Для чтения этой информации можно использовать специальную интерфейсную функцию *GET_MODULE_DESCR()*. Формат расположения служебной информации о модуле (первые 20 ячеек ППЗУ) имеет следующий вид:

- ✓ серийный номер модуля (9 байт);
- ✓ название модуля (7 байт);
- ✓ ревизия модуля (1 байт);
- ✓ тип установленного на модуле DSP (5 байт);
- ✓ флажок присутствия ЦАП на модуле (1 байт);
- ✓ частота установленного на модуле кварца в Гц (4 байта);
- ✓ зарезервировано (13 байт);

В следующих 8 словах (16 байт) хранятся коэффициенты, используемые при корректировке *LBIOS*'ом данных, получаемых с АЦП. Данные коэффициенты записываются в ППЗУ при наладке модуля в ***ЗАО "Л-Кард"***. Благодаря этому на модуле отсутствуют подстроечные резисторы, что сильно улучшает шумовые характеристики модуля и увеличивает их надежность. Формат калибровочных коэффициентов предназначен специально для работы с *LBIOS*. Коэффициенты хранятся в виде чисел типа *WORD* языка C++ (2 байта) и имеют следующий порядок:

- ✓ 20 ячейка – корректировка смещения нуля при усилении '1';
- ✓ 21 ячейка – корректировка смещения нуля при усилении '4';
- ✓ 22 ячейка – корректировка смещения нуля при усилении '16';
- ✓ 23 ячейка – корректировка смещения нуля при усилении '64';
- ✓ 24 ячейка – корректировка масштаба при усилении '1';
- ✓ 25 ячейка – корректировка масштаба при усилении '4';
- ✓ 26 ячейка – корректировка масштаба при усилении '16';
- ✓ 27 ячейка – корректировка масштаба при усилении '64';

В ячейках 28÷31 (8 байт) хранятся коэффициенты, используемые для корректировки кода, выводимого на ЦАП'ы. Данные коэффициенты записываются в ППЗУ при наладке модуля 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 ЦАП нулевого выходного напряжения надо вывести код:

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

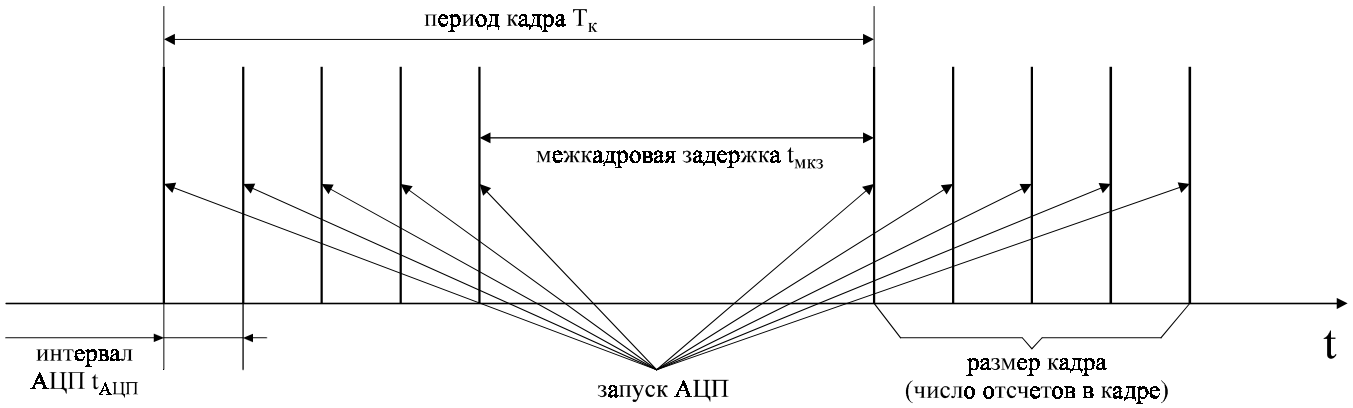
Коэффициенты хранятся в виде чисел типа *WORD* языка C++ (2 байта) и имеют следующий порядок:

- ✓ 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 интерфейсных функций *WRITE_FLASH_WORD()* и *READ_FLASH_WORD()*.

2.3.4. *Формат кадра отсчетов*

Под кадром подразумевается последовательность отсчетов с логических каналов, начиная от **ControlTable[0]** до **ControlTable[ControlTableLength–1]**, где **ControlTable** - управляющая таблица (массив логических каналов), хранящаяся в DSP модуля, а **ControlTableLength** определяет размер (длину) этой таблицы. Загрузить нужную Вам управляющую таблицу в сигнальный процессор модуля можно с помощью интерфейсной функции ***FILL_ADC_PARS()*** (см. *§ 2.5.4. "Установка параметров работы АЦП"*). Временные параметры кадра для **ControlTableLength=5** приведены на следующем рисунке:



где $\mathbf{T_k}$ – временной интервал между соседними кадрами (фактически частота опроса фиксированного логического номера канала *ChannelRate*), $\mathbf{t_{мкз}}$=*InterKadrDelay* – временной интервал между последним отсчетом текущего кадра и первым отсчетом следующего, $\mathbf{t_{АЦП}}$ – интервал запуска АЦП или межканальная задержка. Тогда 1/$\mathbf{t_{АЦП}}$=*AdcRate* – 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 с АЦП в кадре, равен **ControlTableLength**, то все эти временные параметры можно связать следующей формулой:

$$\mathbf{T_k} = 1/\mathbf{ChannelRate} = (\mathbf{ControlTableLength}–1) * \mathbf{t_{АЦП}} + \mathbf{t_{мкз}},$$

или

$$\mathbf{T_k} = 1/\mathbf{ChannelRate} = (\mathbf{ControlTableLength}–1)/\mathbf{AdcRate} + \mathbf{InterKadrDelay}.$$

Временные параметры **AdcRate** и **InterKadrDelay** используются в интерфейсной функции *FILL_ADC_PARS()* при задании необходимого режима работы АЦП.

2.4. *Общие принципы работы с модулем E-440*

2.4.1. *Общий подход к работе с интерфейсными функциями*

Целью штатной DLL библиотеки Lusbapi.dll, поставляемой с модулем *E-440*, является предоставление достаточно наглядного и удобного программного интерфейса при работе с данным устройством. Библиотека содержит в себе определенный набор функций, с помощью которых Вы можете реализовывать многие стандартные алгоритмы ввода/вывода данных в/из модуля.

Перед началом работы с DLL библиотекой в пользовательской программе Вы должны сделать следующие объявление:

```
ILE440 *pE440;                  // указатель на интерфейс модуля E-440
MODULE_DESCR_E440 md;           // структура информации в ППЗУ модуля
```

Первым делом с помощью функции *GetDllVersion()* следует проверить версии используемой DLL библиотеки и текущего программного обеспечения.

Если версии совпадают, то в Вашем приложении необходимо получить указатель на интерфейс модуля, вызвав функцию *CreateInstance()*. В дальнейшем для доступа ко всем интерфейсным функциям модуля необходимо применять именно этот указатель (см. *пример ниже*).

После этого, используя уже полученный указатель на интерфейс модуля, Вам следует проинициализировать доступ к виртуальному слоту, к которому подключён модуль, применяя для этого интерфейсную функцию *OpenLDevice()* (в версии библиотеки 1.0 – *InitLDevice()*). Если ошибки нет, то, в общем случае, какое-то устройство подключено к выбранному виртуальному слоту и можно переходить к этапу его идентификации.

Таким образом, следует проверить с помощью интерфейсной функции *GetModuleName()* название устройства, подключенного к выбранному виртуальному слоту, т.к. к нему, в принципе, может быть присоединено какое-нибудь другое, помимо *E-440*, изделие ***ЗАО "Л-Кард"***, рассчитанное на работу с **USB** шиной. Если все в порядке и подключенное устройство – это модуль *E-440*, то можно переходить к следующей стадии работы.

Важной особенностью модуля *E-440* является то, что на нем установлен мощный современный цифровой сигнальный процессор (DSP – Digital Signal Processor) с фиксированной точкой *ADSP-2185M* фирмы **Analog Devices, Inc.**. Для того, чтобы его "оживить", т.е. заставить работать по требуемому Вам алгоритму, во внутреннюю память DSP надо записать (загрузить) либо фирменную управляющую программу, которая входит в комплект штатной поставки (файл `DSP\E440.BIO`), либо Вашу собственную. Задачей DSP является управление всей установленной на модуле периферией (АЦП, ЦАП, цифровые линии и т.д.), а также сбор и, при необходимости, первичная обработка получаемых данных. Во внутренней памяти DSP расположены программно организованные FIFO буфера АЦП и ЦАП, а также переменные *LBIOS* (см. *Приложение В*). О низкоуровневом взаимодействии компьютера с модулем *E-440*, с одной стороны, и DSP с периферией, с другой, см. *Раздел 3 "НИЗКОУРОВНЕВОЕ ОПИСАНИЕ МОДУЛЯ E-440"*

Т.о., необходимо загрузить в сигнальный процессор модуля управляющую программу (*LBIOS*). Для этого можно воспользоваться интерфейсной функцией *LOAD_LBIOS()*. В случае успешного выполнения данной функции, нужно проверить работоспособность загруженного *LBIOS* с помощью интерфейсной функции *MODULE_TEST()*. Если и эта функция выполнена без ошибки, то это означает, что *LBIOS* успешно загружен и модуль полностью готов к работе.

Далее, *начиная* с версии 2.0 DLL библиотеки Lusbapi.dll, рекомендуется проверять номер текущей версии загруженного в модуль драйвера *LBIOS*. Сделать это несложно с помощью дополнительно введенной интерфейсной функции *GET_LBIOS_VERSION()*.

На следующем этапе Вам следует прочитать служебную информацию, хранящуюся в ППЗУ модуля. Она требуется при работе с некоторыми интерфейсными функциями штатной DLL библиотеки. Интерфейсная функция *GET_MODULE_DESCR()* как раз и предназначена для этой цели. Если функция не вернула ошибку, то это означает, что информация из ППЗУ модуля успешно считана, и можно продолжать работу.

В общем-то, **ВСЁ!** Теперь можно спокойно управлять всей доступной периферией на модуле и самим DSP с помощью соответствующих интерфейсных функций штатной DLL библиотеки, т.е. задавать различные режимы работы АЦП (прием данных с АЦП, 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.

В качестве примера приведем исходный текст (вернее сказать 'скелет') очень простенькой консольной программы для работы с модулем *E-440* (предполагается использование Lusbapi.dll версии не ниже 2.0):

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

ILE440 *pE440;                // указатель на интерфейс модуля
MODULE_DESCR_E440 md;         // структура информации в ППЗУ модуля
```

```
char ModuleName[7];         // название модуля
DWORD LbiosVersion;         // текущий номер версии драйвера LBIOS

int main(void)
{
    // проверим версию DLL библиотеки
    if(GetDllVersion() != LC_CURRENT_VERSION)
    {
      printf("Неправильная версия Dll!");
      return 1;
    }

    // получим указатель на интерфейс модуля
    pE440 = static_cast<ILE440 *>(CreateInstance("e440"));
    if(pE440 == NULL)
    {
      printf("Не могу получить указатель на интерфейс");
      return 1;                 //выйдем из программы с ошибкой
    }

    // попробуем обнаружить какой-нибудь модуль
    // в нулевом виртуальном слоте
    if(!pE440->OpenLDevice(0))
    {
      printf("Не могу получить доступ к модулю!");
      return 1;                 //выйдем из программы с ошибкой
    }

    // прочитаем название модуля в нулевом виртуальном слоте
    if(!pE440->GetModuleName(ModuleName))
    {
      printf("Не могу прочитать название модуля!\n");
      return 1;                 //выйдем из программы с ошибкой
    }

    // проверим: этот модуль - 'E-440'?
    if(strcmp(ModuleName, "E440"))
    {
      printf(" В нулевом виртуальном слоте не 'E-440'\n");
      return 1;                 //выйдем из программы с ошибкой
    }

    // теперь можно попробовать загрузить из соответствующего ресурса
    // библиотеки Lusbapi.dll (версия 2.0) код драйвера LBIOS
    if(!pE440->LOAD_LBIOS())
    {
      printf("Не выполнена функция LOAD_LBIOS()!");
      return 1;                 //выйдем из программы с ошибкой
    }

    // проверим работоспособность загруженного LBIOS
    if(!pE440->MODULE_TEST())
    {
      printf("Не выполнена функция MODULE_TEST()!");
      return 1;                 //выйдем из программы с ошибкой
    }
```

```
    // получим версию загруженного LBIOSa (Lusbapi.dll версии 2.0)
    if(!pE440->GET_LBIOS_VERSION(&LbiosVersion))
    {
      printf("Не выполнена функция GET_LBIOS_VERSION()!");
      return 1;                 //выйдем из программы с ошибкой
    }

    // теперь проверим версию загруженного LBIOS
    if(LbiosVersion != LC_CURRENT_VERSION)
    {
      printf(" Ошибка версии драйвера LBIOS!");
      return 1;                 //выйдем из программы с ошибкой
    }

    // попробуем прочитать информацию, хранящуюся в ППЗУ модуля
    md.size = sizeof(MODULE_DESCR_E440);
    if(!pE440->GET_MODULE_DESCR(&md))
    {
      printf("Не выполнена функция GET_MODULE_DESCR ()!");
      return 1;                 //выйдем из программы с ошибкой
    }

    printf("Модуль E-440 (серийный номер %s) полностью готов к\
                                работе!", md.SerialNumber);

    // далее можно располагать функции для непосредственного
    // управления модулем!

    . . . . . .

    // завершим работу с модулем
    if(!pE440->ReleaseLDevice())
    {
      printf("Не выполнена функция ReleaseLDevice()!");
      return 1;                 //выйдем из программы с ошибкой
    }

    // выйдем из программы
    return 0;
}
```

2.4.2. Общая структура LBIOS

На модуле *E-440* устанавливается так называемый цифровой сигнальный процессор (Digital Signal Processor – DSP) с фиксированной точкой *ADSP-2185M* фирмы **Analog Devices, Inc.** Основное его назначение – это управление различного рода периферийными устройствами, установленными на модуле, а также, возможно, выполнение необходимой предварительной обработки данных. Одно из главных преимуществ применения на модуле именно цифрового сигнального процессора заключается в том, что достаточно гибко чисто программным образом можно изменять в довольно широких пределах алгоритмы работы модуля с периферийными устройствами (достаточно лишь овладеть достаточно несложным языком ассемблера DSP). Так, в штатном драйвере *LBIOS* (исходные тексты которого можно найти в директории DSP\ на прилагаемом к данному модулю CD-ROM'е) реализуются наиболее широко используемые алгоритмы работы с АЦП, ЦАП, входными/выходными ТТЛ линиями и т.д.

В принципе, для написания пользовательских программ достаточно знать, что установленный на модуле DSP обладает двумя независимыми типами памяти, а именно:

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

Для доступа к содержимому ячеек DSP каждого типа памяти существуют штатные интерфейсные функции (см. *§ 2.5.3. "Функции для доступа к памяти DSP модуля"*). Карты распределения памяти обоих типов для различных типов сигнальных процессоров, а также взаимное расположение составных частей штатного *LBIOS*, подробно показаны в *Приложении В*. Как видно из указанного приложения, *LBIOS* состоит из

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

Исполняемый код *LBIOS* написан с учетом возможности взаимодействия данного драйвера с Вашей пользовательской программой в PC по так называемому принципу команд (подробнее см. *§ 3.1. "Структурная схема модуля E-440"*).

Адреса предопределенных переменных в PM DSP модуля, задающих важные параметры функционирования штатного *LBIOS*, а также их краткие толкования, приведены в *Таблице 13*.

В *LBIOS* программно организовано два циклических FIFO буфера: для приема данных с АЦП и для выдачи данных на ЦАП. Передача данных из FIFO буфера АЦП в PC производится порциями по *Длина_FIFO_Буфера_АЦП/2* отсчетов мере их поступления с АЦП. Т.о., фактически чисто программным образом реализован так называемый двойной FIFO буфер. Т.е. при поступлении из PC команды на запуск АЦП, драйвер *LBIOS* ожидает накопления данных в первой половинке FIFO буфера АЦП. После того как первая половинка буфера полностью заполнится готовыми данными с АЦП, дается команда на их передачу в PC (в тоже время **не прекращается** сбор данных во вторую половинку FIFO буфера). После накопления данных во второй половинке FIFO буфера опять дается команда на их передачу в PC и продолжается сбор данных уже в первую половинку. И так до бесконечности по циклу, пока не придет команда из PC на останов работы АЦП. Все то же самое применимо и для алгоритма работы ЦАП.

Драйвер *LBIOS* написан таким образом, что можно независимо управлять работой АЦП, ЦАП и ТТЛ линиями.

2.5. Описание штатной DLL библиотеки

В настоящем разделе приведены достаточно подробные описания переменных, структур и интерфейсных функций, входящих в состав штатной DLL библиотеки для модуля *E-440*.

2.5.1. Переменные и структуры

2.5.1.1. Структура ADC_PARS_E440

Структура *ADC_PARS_E440* описана в файле `Lusbapi.h` и представлена ниже:

```
struct ADC_PARS_E440
{
    WORD size;                      // размер данной структуры в байтах
    bool AdcEnabled;                // состояние работы АЦП (при чтении)
    bool CorrectionEnabled;         // управление корректировкой данных
    WORD InputMode;                 // режим ввода данных с АЦП
    WORD SynchroAdType              // тип аналоговой синхронизации
    WORD SynchroAdMode;             // режим аналоговой синхронизации
    WORD SynchroAdChannel;          // канал АЦП при аналоговой синхронизации
    WORD SynchroAdPorog;            // порог срабатывания АЦП при аналоговой
                                    // синхронизации
    WORD ChannelsQuantity;          // число активных каналов (размер кадра)
    WORD ControlTable[128];         // управляющая таблица с активными каналами
    double AdcRate;                 // частота работы АЦП в кГц
    double InterKadrDelay;          // межкадровая задержка в мс
    double ChannelRate;             // частота одного канала кГц (период кадра)
    WORD AdcFifoBaseAddress;        // базовый адрес FIFO буфера АЦП в DSP модуля
    WORD AdcFifoLength;             // длина FIFO буфера АЦП в DSP модуля
    WORD CalibrKoefAdc[8];          // корректировочные коэф. для АЦП
};
```

Перед началом работы с АЦП необходимо заполнить поля данной структуры и передать ее в модуль с помощью интерфейсной функции *FILL_ADC_PARS()*. При этом в качестве корректировочных коэффициентов для получаемых с АЦП отсчетов можно использовать соответствующую информацию из ППЗУ модуля (см. *§ 2.5.1.3. "Структура MODULE_DESCR_E440"*). Также при необходимости можно считать из модуля текущие параметры функционирования АЦП, используя *GET_CUR_ADC_PARS()*.

2.5.1.2. Структура DAC_PARS_E440

Структура *DAC_PARS_E440* описана в файле `Lusbapi.h` и представлена ниже:

```
struct DAC_PARS_E440
{
    WORD size;                      // размер данной структуры в байтах
    bool DacEnabled;                // состояние работы ЦАП (при чтении)
    double DacRate;                 // частота работы ЦАП в кГц
    WORD DacFifoBaseAddress;        // базовый адрес  FIFO буфера ЦАП в DSP модуля
    WORD DacFifoLength;             // длина FIFO буфера ЦАП в DSP модуля
};
```

Перед началом работы с ЦАП необходимо заполнить поля данной структуры и передать ее в модуль с помощью интерфейсной функции *FILL_DAC_PARS()*. Также при необходимости можно

считать из модуля текущие параметры функционирования ЦАП, используя *GET_CUR_DAC_PARS()*.

2.5.1.3. Структура MODULE_DESCR_E440

Структура *MODULE_DESCR_E440* описана в файле `Lusbapi.h` и представлена ниже:

```
struct MODULE_DESCR_E440
{
    char SerialNumber[9];       // серийный номер модуля (8 символов)
    char Name[7];               // название модуля – "E440"
    char Revision;              // ревизия модуля: 'A' или 'B'
    char Dsp_Type[5];           // тип установленного на модуле DSP:
                                // строка "2184" для ADSP-2184,
                                // строка "2185" для ADSP-2185,
                                // строка "2186" для ADSP-2186,
    char IsDacPresented;        // флажок наличия ЦАП на модуле:
                                // 0 – ЦАП отсутствует,
                                // 1 - ЦАП присутствует,
    long QuartzFrequency;       // тактовая частота входного клока DSP,
                                // равная 24 000 000 Гц.
    char Reserved[13];          // зарезервировано
    WORD CalibrKoefAdc[8];      // корректировочные коэф. для АЦП
    WORD CalibrKoefDac[4];      // корректировочные коэф. для ЦАП
};
```

Данная структура используется в интерфейсных функциях, которые работают со служебной областью пользовательского ППЗУ: *SAVE_MODULE_DESCR()* и *GET_MODULE_DESCR()*. Подробности конфигурации этого ППЗУ см. *§ 2.3.3 "Формат пользовательского ППЗУ"*.

2.5.1.4. Переменные LBIOS

Любой программист при желании может напрямую работать с памятью DSP модуля (и программ, и данных), используя соответствующие интерфейсные функции, которые обеспечивают доступ, как к отдельным ячейкам памяти, так и к целым массивам. Эта возможность позволяет программисту работать с модулем, непосредственно обращаясь к соответствующим ячейкам либо памяти программ, либо памяти данных DSP. Карта распределения, как памяти программ, так и памяти данных для *ADSP-2185M*, который установлен на модуле, приведена в *Приложении В*.

Ниже в *Таблице 13* приводятся *предопределенные* адреса переменных штатного *LBIOS*, расположенных в памяти **программ** DSP, и их краткие описания. Собственно сами переменные *LBIOS* являются $16^{ти}$ битными и располагаются в старших $16^{ти}$ битах $24^{х}$ битного слова памяти программ DSP (при этом младшие 8 из этих $24^{х}$ бит не используются). Программист может напрямую обращаться к переменным штатного *LBIOS*, чтобы при необходимости считать и/или изменить их содержимое с помощью интерфейсных функций *GET_LBIOS_WORD()* и *PUT_LBIOS_WORD()* (см. *§ 2.5.3 "Функции для доступа к памяти DSP модуля"*).

Таблица 13. Адреса управляющих переменных LBIOS

Название переменной	Адрес Hex	Назначение переменной
L_READY_E440	0x31	После загрузки показывает готовность модуля к дальнейшей работе

L_TMODE1_E440	0x32	Тестовая переменная. После загрузки драйвера (*LBIOS*) по этому адресу должно читаться число 0x5555.
L_TMODE2_E440	0x33	Тестовая переменная. После загрузки драйвера (*LBIOS*) по этому адресу должно читаться число 0xAAAA.
L_TEST_LOAD_E440	0x34	Тестовая переменная.
L_COMMAND_E440	0x35	Переменная, при помощи которой драйверу передается номер команды, которую он должен выполнить. Краткое описание номеров команд приведено ниже в *Таблице 14*.
L_DAC_SCLK_DIV_E440	0x37	Текущий делитель для SCLK0. Используется при работе с ЦАП.
L_DAC_RATE_E440	0x38	Переменная, задающая код частоты вывода данных на ЦАП.
L_ADC_RATE_E440	0x39	Переменная, задающая код частоты работы АЦП.
L_ADC_ENABLED_E440	0x3A	Переменная, показывающая текущее состояние АЦП (работает или нет).
L_ADC_FIFO_BASE_ ADDRESS_E440	0x3B	Текущий базовый адрес FIFO буфера АЦП, который всегда должен быть равен **0x0**.
L_CUR_ADC_FIFO_ LENGTH_E440	0x3C	Текущая длина FIFO буфера АЦП. По умолчанию **L_CUR_ADC_FIFO_LENGTH_E440 = 0x3000**.
L_ADC_FIFO_ LENGTH_E440	0x3E	Требуемая длина FIFO буфера АЦП. По умолчанию **L_ADC_FIFO_LENGTH_E440 = 0x3000**.
L_CORRECTION_ ENABLED_E440	0x3F	Переменная запрещающая (0)/ разрешающая (1) корректировку данных аналоговых каналов при помощи калибровочных коэффициентов. По умолчанию **L_CORRECTION_ENABLED = 0x0**.
L_ADC_SAMPLE_E440	0x41	Данная переменная используется при однократном вводе с АЦП, храня оцифрованное значение.
L_ADC_CHANNEL_E440	0x42	Данная переменная используется при однократном вводе с АЦП, задавая логический номер канала.
L_INPUT_MODE_E440	0x43	Переменная, задающая тип синхронизации при вводе данных с АЦП (аналоговая, цифровая или ее отсутствие).
L_SYNCHRO_AD_ CHANNEL_E440	0x46	При аналоговой синхронизации задает логический номер канала, по которому происходит синхронизация.

L_SYNCHRO_AD_ POROG_E440	0x47	При аналоговой синхронизации задает порог срабатывания в кодах АЦП.
L_SYNCHRO_AD_ MODE_E440	0x48	Переменная, задающая режим аналоговой синхронизации.
L_SYNCHRO_TYPE_E440	0x49	При аналоговой синхронизации задает тип срабатывания по уровню (**0x0**) либо по переходу (**0x1**).
L_CONTROL_TABLE_ LENGHT_E440	0x4B	Размер управляющей таблицы (максимум 128 логических каналов). По умолчанию – **0x8**.
L_FIRST_SAMPLE_ DELAY_E440	0x4C	Переменная, задающая код задержки для первого отсчета при старте АЦП
L_INTER_KADR_ DELAY_E440	0x4D	Переменная, задающая код межкадровой задержки при вводе данных с АЦП.
L_DAC_SAMPLE_E440	0x50	Переменная, задающая значение для однократного вывода на ЦАП
L_DAC_ENABLED_E440	0x51	Переменная, показывающая текущее состояние ЦАП (работает или нет).
L_DAC_FIFO_BASE_ ADDRESS_E440	0x52	Текущий базовый адрес FIFO буфера ЦАП, который всегда должен быть равен **0x3000**.
L_CUR_DAC_FIFO_ LENGTH_E440	0x54	Текущая длина FIFO буфера ЦАП. По умолчанию **L_CUR_DAC_FIFO_LENGTH_E440 = 0xFC0**.
L_DAC_FIFO_ LENGTH_E440	0x55	Требуемая длина FIFO буфера ЦАП. По умолчанию **L_DAC_FIFO_LENGTH_E440 = 0xFC0**.
L_FLASH_ENABLED_E440	0x56	Флаг разрешения(1)/запрещения(0) записи в ППЗУ модуля
L_FLASH_ADDRESS_E440	0x57	Номер ячейки ППЗУ (от 0 до 63)
L_FLASH_DATA_E440	0x58	Данные в/из ППЗУ
L_ENABLE_TTL_OUT_E440	0x59	Данная переменная разрешает (0x1) либо запрещает (0x0) использование выходных цифровых линий (перевод их в третье состояние)
L_TTL_OUT_E440	0x5A	Слово (16 бит), в котором по-битово хранятся значения $16^{ти}$ выходных цифровых линий для их выставления по команде **C_TTL_OUT_E440** (см. *Таблицу 14*).

L_TTL_IN_E440	0x5B	Слово (16 бит), в котором после выполнения команды **C_TTL_IN_E440** (см. *Таблицу 14*) побитово хранятся значения 16^{ти} входных цифровых линий.
L_SCALE_E440	0x60	Массив с 4^{мя} калибровочными коэффициентами, используемый для корректировки масштаба данных с АЦП. По умолчанию – **{ 0x8000, 0x8000, 0x8000, 0x8000}**
L_ZERO_E440	0x64	Массив с 4^{мя} калибровочными коэффициентами, используемый для корректировки смещения нуля данных с АЦП. По умолчанию – **{ 0x0, 0x0, 0x0, 0x0}**
L_CONTROL_TABLE_E440	0x80	Управляющая таблица, содержащая последовательность логических номеров каналов (максимум 128 элементов). В соответствии с ней DSP производит последовательный циклический сбор данных с АЦП. Размер этой таблицы задается переменной **L_CONTROL_TABLE_LENGHT_E440** (см. выше). По умолчанию – **{ 0, 1, 2, 3, 4, 5, 6, 7}**

2.5.1.5. Номера команд LBIOS

Фирменный *LBIOS* для модуля *E-440* работает по принципу команд, т.е. драйверу в модуль передается номер команды, которую он должен выполнить. Список доступных номеров команд для штатного *LBIOS* (версия 2.0) приведен ниже в таблице.

Таблица 14. Номера команд штатного LBIOS.

Название команды	Номер команды	Назначение	Используемые переменные
C_TEST_E440	0	Проверка загрузки модуля и его работоспособности.	**L_TEST_LOAD_E440**
C_ENABLE_FLASH_WRITE_E440	1	Разрешение процедуры записи в пользовательское ППЗУ	**L_FLASH_ENABLED_E440**
C_READ_FLASH_WORD_E440	2	Чтение слова из ППЗУ	**L_FLASH_ADDRESS_E440, L_FLASH_DATA_E440**
C_WRITE_FLASH_WORD_E440	3	Запись слова в ППЗУ	**L_FLASH_ADDRESS_E440, L_FLASH_DATA_E440**
C_START_ADC_E440	4	Разрешение работы АЦП	--------
C_STOP_ADC_E440	5	Запрещение работы АЦП	--------

C_ADC_KADR_E440	6	Получение кадра отсчетов с АЦП	--------
C_ADC_ SAMPLE_E440	7	Однократный ввод отсчета с АЦП для заданного канала	**L_ADC_ SAMPLE_E440, L_ADC_ CHANNEL_E440**
C_START_DAC_E440	8	Разрешение работы ЦАП	--------
C_STOP_DAC_E440	9	Запрещение работы ЦАП	--------
C_DAC_ SAMPLE_E440	10	Однократный вывод отсчета на ЦАП	**L_DAC_SAMPLE_E440**
C_ENABLE_TTL_ OUT_E440	11	Разрешение выходных цифровых линий	**L_ENABLE_TTL_ OUT_E440**
C_TTL_IN_E440	12	Считывание состояния 16ти внешних цифровых входных линий.	**L_TTL_IN_E440**
C_TTL_OUT_E440	13	Управление 16тью внешними цифровыми выходными линиями.	**L_TTL_OUT_E440**

2.5.2. *Функции общего характера*

2.5.2.1. *Получение версии DLL библиотеки*

Формат: **LPVOID** ***GetDllVersion****(void)*
Назначение: Данная функция является одной из двух экспортируемых из штатной DLL функцией и возвращает версию используемой DLL библиотеки. Рекомендованную последовательность вызовов интерфейсных функций см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*.
Передаваемые параметры: нет.
Возвращаемое значение: номер версии DLL библиотеки.

2.5.2.2. *Получение указателя на интерфейс модуля*

Формат: **LPVOID** ***CreateInstance****(char *DeviceName)*
Назначение: Данная функция должна обязательно вызываться в начале каждой пользовательской программы, работающей с модулями *E-440*. Она является одной из двух экспортируемых из штатной DLL функцией и возвращает указатель на интерфейс для устройства с названием *DeviceName*. Все интерфейсные функции штатной DLL библиотеки вызываются именно через этот возвращаемый указатель. Рекомендованную последовательность вызовов интерфейсных функций см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*.
Передаваемые параметры: • *DeviceName* – строка с названием устройства (для данного модуля это – "E440").
Возвращаемое значение: В случае успеха — указатель на интерфейс, иначе — **NULL**.

2.5.2.3. *Функция завершения работы с модулем*

Формат: **bool** ***ReleaseLDevice****(void)*
Назначение: Данная интерфейсная функция реализует корректное высвобождение интерфейсного указателя, проинициализированного с помощью интерфейсной функции *CreateInstance()*. Используется для аккуратного завершения сеанса работы с модулем (если предварительно удачно выполнилась функция *CreateInstance()*). **!!!Внимание!!!** Данная функция **должна обязательно** вызываться в Вашем приложении перед непосредственным выходом из него во избежания утечки ресурсов компьютера. Рекомендованную последовательность вызовов интерфейсных функций см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*.
Передаваемые параметры: нет.
Возвращаемое значение: *true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.2.4. Инициализация доступа к модулю

Формат:	**bool**	***InitLDevice****(WORD VirtualSlot)*	(версия 1.0)
	bool	***OpenLDevice****(WORD VirtualSlot)*	(с версии 2.0)

Назначение:

С программной точки зрения, не вдаваясь в излишние тонкости, подсоединенный к компьютеру модуль *E-440* можно рассматривать как устройство, подключённое к некоему виртуальному слоту с сугубо индивидуальным номером. Основное же назначение данной интерфейсной функции – определить, находится ли хоть какое-нибудь устройство в заданном виртуальном слоте. Если функция ***OpenLDevice()*** успешно выполнилась для заданного виртуального слота, то далее следует убедиться, что к этому слоту подключён именно модуль *E-440* (функция *GetModuleName()*). И если это так – можно переходить непосредственно к загрузке модуля и его последующему управлению с помощью соответствующих интерфейсных функций библиотеки Lusbapi.dll.

InitLDevice() является устаревшим названием данной функции, хотя библиотекой по-прежнему поддерживается (кроме **Delphi**). Рекомендованную последовательность вызовов интерфейсных функций см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*.

Передаваемые параметры:

- *VirtualSlot* – номер виртуального слота, к которому, как предполагается, подключен модуль *E-440*.

Возвращаемое значение: *true* – устройство находится в выбранном виртуальном слоте и можно попробовать определить его название с помощью интерфейсной функции *GetModuleName()* (см. ниже);
false – никакого устройства в выбранном виртуальном слоте нет (может, стоит попробовать другой номер виртуального слота).

2.5.2.5. Освобождение виртуального слота

Формат:	**bool**	***CloseLDevice*** *(void)*

Назначение:

Данная интерфейсная функция прерывает, если необходимо, всякое взаимодействие с текущим виртуальным слотом, т.е. выполняет его освобождение (и связанных с ним ресурсов компьютера). После её применения всякий доступ к модулю *E-440* становится невозможным. Для возобновления нормального доступа к устройству необходимо вновь воспользоваться интерфейсной функцией *OpenLDevice()*. Таким образом, эта функция, по своей сути, противоположна интерфейсной функции *OpenLDevice()*. Фактически данная функция используется в таких интерфейсных функциях как *OpenLDevice()* и *ReleaseLDevice()*.

Передаваемые параметры: нет.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.2.6. Получение названия модуля

Формат: **bool** ***GetModuleName***(*char *ModuleName*)

Назначение:

Так как интерфейсная функция *OpenLDevice()* определяет только наличие какого-нибудь устройства **USB** в выбранном виртуальном слоте, то, очевидно, необходимо каким-то образом это устройство идентифицировать. В принципе, к данному виртуальному слоту может быть подключено какое-нибудь другое (помимо модуля *E-440*) изделие ***ЗАО "Л-Кард"***, рассчитанное на работу с шиной **USB**. Данная интерфейсная функция позволяет получить название подключенного к слоту модуля. Массив под название модуля *ModuleName* (не менее 6 символов плюс признак конца строки '\0', т.е. нулевой байт) должен быть заранее определен. Рекомендованную последовательность вызовов интерфейсных функций см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*.

Передаваемые параметры:

- *ModuleName* – возвращается строка, не менее 6 символов, с названием модуля (в нашем случае это должна быть строка `"E440"`).

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.2.7. Загрузка LBIOS

Формат:	**bool**	***LOAD_LBIOS***(*PCHAR FileName*)	(версия 1.0)
	bool	***LOAD_LBIOS***(*PCHAR FileName = NULL*)	(с версии 2.0)

Назначение:

Данная интерфейсная функция выполняет операцию загрузки драйвера (штатного *LBIOS* или Вашего) в DSP модуля. Файл *FileName* с кодом драйвера должен находиться в текущей директории Вашего приложения. С версии 2.0 в DLL библиотеке появилась дополнительная возможность загружать *LBIOS*, содержимое которого хранится в самом теле библиотеки в виде соответствующего ресурса. Для этого достаточно параметр *FileName* задать в виде **NULL**. **NULL** является также значением по умолчанию для параметра *FileName*. Рекомендованную последовательность вызовов интерфейсных функций см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*.

Передаваемые параметры:

- *FileName* – строка с названием файла, содержащим код загружаемой управляющей программы. Например, для штатного *LBIOS* это строка `"E440.BIO"`. Начиная с версии 2.0, если данный параметр задан как **NULL**, то загрузка модуля будет осуществляться тем *LBIOS*'ом, который находится в виде ресурса в теле штатной DLL библиотеки.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.2.8. Проверка загрузки модуля

Формат: **bool** ***MODULE_TEST****(void)*
Назначение: Данная интерфейсная функция проверяет правильность загрузки модуля и его работоспособность. **!!!Внимание!!!** Данная функция работает надлежащим образом **только** после выполнения интерфейсной функции *LOAD_LBIOS()*. Рекомендованную последовательность вызовов интерфейсных функций см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*.
Передаваемые параметры: нет
Возвращаемое значение: *true* – *LBIOS* успешно загружен и функционирует надлежащим образом, *false* – произошла ошибка загрузки или функционирования *LBIOS*.

2.5.2.9. Получение версии LBIOS

Формат: **bool** ***GET_LBIOS_VERSION****(DWORD *LbiosVersion)*
Назначение: С версии 2.0 в DLL библиотеке Lusbapi.dll появилась новая интерфейсная функция, которая позволяет выявлять номер текущей версии уже загруженного в модуль драйвера DSP (он же *LBIOS*). В случае успешного выполнения этой функции в переменной *LbiosVersion* возвращается текущий номер версии *LBIOS*, который **должен обязательно** совпадать с текущим номером версии используемой в данный момент времени DLL библиотеки Lusbapi.dll (т.е. фактически с предопределенной в файле Lusbapi.h константой LC_CURRENT_VERSION). **!!!Внимание!!!** Данная функция работает надлежащим образом **только** после выполнения интерфейсных функций *LOAD_LBIOS()* и *MODULE_TEST()*. Рекомендованную последовательность вызовов интерфейсных функций см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*.
Передаваемые параметры: • *LbiosVersion* – в переменной возвращается текущий номер версии загруженного в модуль драйвера *LBIOS*
Возвращаемое значение: *true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.2.10. Сброс DSP на модуле

Формат:	**bool**	***DSP_RESET****(void)*
Назначение: Данная интерфейсная функция производит сброс (RESET) DSP модуля. Используется при перезагрузке *LBIOS* или для полной остановки работы DSP. **Необходимо помнить, что после выполнения данной функции работа DSP модуля полностью останавливается и для приведения его снова в рабочее состояние требуется перезагрузить *LBIOS* (например, с помощью функции *LOAD_LBIOS()*)**.		
Передаваемые параметры: нет		
Возвращаемое значение:	*true* – функция успешно выполнена; *false*– функция не выполнена.	

2.5.2.11. Передача номера команд в драйвер LBIOS

Формат:	**bool**	***SEND_COMMAND****(WORD Command)*
Назначение: Данная интерфейсная функция записывает в предопределенную переменную *L_COMMAND_E440* номер команды и вызывает командное прерывание *IRQE* в DSP модуля. В ответ на это прерывание *LBIOS* выполняет действия, строго соответствующие номеру переданной команды. **!!!Внимание!!!** Данная функция работает надлежащим образом **только** после выполнения интерфейсных функции *LOAD_LBIOS()* и *MODULE_TEST()*. Рекомендованную последовательность вызовов интерфейсных функций см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*.		
Передаваемые параметры: • *Command* – номер команды, передаваемый в драйвер DSP.		
Возвращаемое значение:	*true* – функция успешно выполнена; *false*– функция не выполнена.	

2.5.2.12. Получение описания ошибок выполнения функций

Формат: **int** ***GetLastErrorString***(*LPTSTR lpBuffer, DWORD nSize*)

Назначение:

Если в процессе работы с DLL библиотекой Lusbapi.dll какая-нибудь интерфейсная функция штатной библиотеки вернула ошибку, то **только** непосредственно после этого с помощью вызова данной интерфейсной функции можно получить краткое толкование произошедшего сбоя. **!!!Внимание!!!** Данная интерфейсная функция не выполняет классификацию ошибок для интерфейсных функций *ReadData()* и *WriteData()*. Т.к. эти функции фактически являются слепком со стандартных *Windows API* функций *ReadData()* и *WriteData()*. Для выявления ошибок их выполнения следует пользоваться классификацией ошибок, присущей системе *Windows*.

Передаваемые параметры:

- *lpBuffer* – указатель на строку, в которой функция вернет описание ошибки;
- *nSize* – длина строки (рекомендуется 128 символов).

Возвращаемое значение: В случае успеха – кол-во скопированных в буфер символов; в противном случае – ноль.

2.5.3. *Функции для доступа к памяти DSP модуля*

Интерфейсные функции данного раздела обеспечивают доступ, как к отдельным ячейкам, так и к целым массивам памяти DSP модуля. Эта возможность позволяет программисту работать с модулем напрямую, непосредственно обращаясь к соответствующим ячейкам памяти. При этом для работы с этими функциями, в принципе, совсем не требуется загруженного в модуль драйвера *LBIOS*.

2.5.3.1. Чтение слова из памяти данных DSP

Формат: **bool** ***GET_DM_WORD****(WORD Address, SHORT *Data)*
Назначение: Данная функция считывает значение слова, находящееся по адресу *Address* в памяти данных DSP модуля.
Передаваемые параметры: • *Address* – адрес ячейки в памяти данных DSP, значение которой необходимо считать; • *Data* – указатель на переменную, куда функция положит считанное $16^{ти}$ битное слово.
Возвращаемое значение: *true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.3.2. Чтение слова из памяти программ DSP

Формат: **bool** ***GET_PM_WORD****(WORD Address, long *Data)*
Назначение: Данная функция считывает значение слова, находящееся по адресу *Address* в памяти программ DSP модуля.
Передаваемые параметры: • *Address* – адрес ячейки в памяти программ DSP, значение которой необходимо считать; • *Data* – указатель на переменную, куда функция положит считанное 24^{x} битное слово.
Возвращаемое значение: *true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.3.3. Запись слова в память данных DSP

Формат: **bool** ***PUT_DM_WORD****(WORD Address, SHORT Data)*
Назначение: Данная функция записывает значение *Data* в ячейку с адресом *Address* в памяти данных DSP модуля.
Передаваемые параметры: • *Address* – адрес ячейки в памяти данных DSP, куда необходимо записать значение *Data*; • *Data* – значение записываемого $16^{ти}$ битного слова.
Возвращаемое значение: *true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.3.4. Запись слова в память программ DSP

Формат: **bool** ***PUT_PM_WORD****(WORD Address, long Data)*

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

Передаваемые параметры:
- *Address* – адрес ячейки в памяти программ DSP, куда записывается значение *Data*;
- *Data* – значение записываемого 24^{x} битного слова.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.3.5. Чтение массива слов из памяти данных DSP

Формат: **bool** ***GET_DM_ARRAY****(WORD BaseAddress, WORD NPoints, SHORT *Buffer)*

Назначение:
Данная функция считывает массив слов длиной *NPoints* в буфер *Buffer*, начиная с адреса ячейки *BaseAddress* в памяти данных DSP модуля. Буфер *Buffer* надлежащей длины необходимо заранее определить.

Передаваемые параметры:
- *BaseAddress* – стартовый адрес в памяти данных DSP, начиная с которого производится чтение массива;
- *NPoints* – длина считываемого массива;
- *Buffer* – указатель на буфер, в который передаются считываемые значения.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.3.6. Чтение массива слов из памяти программ DSP

Формат: **bool** ***GET_PM_ARRAY****(WORD BaseAddress, WORD NPoints, long *Buffer)*

Назначение:
Данная функция считывает массив слов длиной *NPoints* в буфер *Buffer*, начиная с адреса ячейки *BaseAddress* в памяти программ DSP модуля. Буфер *Buffer* надлежащей длины необходимо заранее определить. При использовании этой функции следует помнить, что одно слово памяти программ DSP является 24^{x} битным.

Передаваемые параметры:
- *BaseAddress* – стартовый адрес в памяти программ DSP, начиная с которого производится чтение массива;
- *NPoints* – число считываемых 24^{x} битных слов;
- *Buffer* – указатель на буфер, в который передаются считываемые значения.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.3.7. Запись массива слов в память данных DSP

Формат: **bool** ***PUT_DM_ARRAY****(WORD BaseAddress, WORD Npoints, SHORT *Buffer)*

Назначение:
Данная функция записывает массив слов длиной *NPoints* из буфера *Buffer* в память данных DSP модуля, начиная с адреса *BaseAddress.*

Передаваемые параметры:
- *BaseAddress* – стартовый адрес в памяти данных DSP, начиная с которого производится запись массива;
- *NPoints* – длина записываемого массива;
- *Buffer* – указатель на буфер, из которого идет запись.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.3.8. Запись массива слов в память программ DSP

Формат: **bool** ***PUT_PM_ARRAY****(WORD BaseAddress, WORD NPoints, long *Buffer)*

Назначение:
Данная функция записывает массив слов длиной *NPoints* из буфера *Buffer* в память программ DSP модуля, начиная с адреса *BaseAddress.* При использовании этой функции следует помнить, что одно слово памяти программ DSP является 24^{x} битным.

Передаваемые параметры:
- *BaseAddress* – стартовый адрес в памяти программ DSP, начиная с которого производится запись массива;
- *NPoints* – число записываемых 24^{x} битных слов;
- *Buffer* – указатель на буфер, из которого идет запись.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.3.9. Чтение переменной LBIOS

Формат: **bool** ***GET_LBIOS_WORD****(WORD Address, SHORT *Data)*

Назначение:
Данная функция осуществляет аккуратное считывание $16^{ти}$ битной переменной штатного *LBIOS*, расположенной по адресу *Address* в 24^{x} битной памяти ***программ*** DSP модуля (см. *§ 2.5.1.4. "Переменные LBIOS"*).

Передаваемые параметры:
- *Address* – адрес ячейки переменной *LBIOS* в памяти ***программ*** DSP, значение которой необходимо считать;
- *Data* – указатель на переменную, куда функция положит считанное $16^{ти}$ битное значение переменной штатного *LBIOS*.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.3.10. Запись переменной LBIOS

Формат:	**bool**	***PUT_LBIOS_WORD****(WORD Address, SHORT Data)*

Назначение:

Данная функция осуществляет аккуратную запись $16^{ти}$ битного значения *Data* в переменную штатного *LBIOS*, расположенную по адресу *Address* в $24^{х}$ битной памяти ***программ*** DSP модуля (см. *§ 2.5.1.4. "Переменные LBIOS"*).

Передаваемые параметры:

- *Address* – адрес ячейки переменной *LBIOS* в памяти ***программ*** DSP, куда необходимо записать значение *Data*;
- *Data* – значение записываемого $16^{ти}$ битного значения переменной штатного *LBIOS*.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.4. *Функции для работы с АЦП*

Интерфейсные функции штатной DLL библиотеки позволяют реализовывать разнообразные алгоритмы работы с АЦП (*независимо* от состояния ЦАП). Вообще-то модуль, с точки зрения состояния АЦП, может находиться как бы в двух режимах:

1. режим “покоя”;
2. потоковый (*перманентный*) сбор данных с АЦП.

Функция *START_ADC()* позволяет переводить модуль во второе из этих состояний, а *STOP_ADC()* — в первое. Прежде чем запустить АЦП, необходимо передать в модуль требуемые параметры работы АЦП: тип синхронизации, частота работы АЦП, длина и базовый адрес FIFO буфера АЦП, управляющую таблицу и т.д. Эту операцию можно выполнить с помощью интерфейсной функции *FILL_ADC_PARS()*. Получаемые с АЦП данные модуль складывает в двойной циклический FIFO буфер АЦП, который расположен в памяти данных DSP модуля (см. *§ 2.4.2. “Общая структура LBIOS”*). Для извлечения из модуля полученных с АЦП данных следует пользоваться функцией *ReadData()*. При одновременной работе АЦП и ЦАП необходимо помнить, что максимально возможная пропускная способность шины **USB** для данного модуля не более 500 кСлов/с. Примеры корректного применения интерфейсных функций для работы с АЦП можно найти в директориях `\Example\BC5\ReadData`, `\Example\BC5\ReadWrite` и `\Example\BCB5\Synchro`.

2.5.4.1. Разрешение работы АЦП

Формат:	**bool**	***START_ADC****(void)*
Назначение: Данная функция запускает модуль на перманентный сбор данных с АЦП, расмещая полученные отсчёты в циклическом FIFO буфере АЦП. Параметры работы АЦП предварительно передаются в модуль с помощью интерфейсной функции *FILL_ADC_PARS()*. Извлечение из модуля уже собранных с АЦП данных можно осуществлять с помощью интерфейсной функции *ReadData()*.		
Передаваемые параметры: нет		
Возвращаемое значение:	*true* – функция успешно выполнена; *false*– функция не выполнена.	

2.5.4.2. Запрещение работы АЦП

Формат:	**bool**	***STOP_ADC****(void)*
Назначение: Данная функция запрещает модулю осуществлять сбор данных с АЦП		
Передаваемые параметры: нет		
Возвращаемое значение:	*true* – функция успешно выполнена; *false*– функция не выполнена.	

2.5.4.3. Установка параметров работы АЦП

Формат: **bool** ***FILL_ADC_PARS****(ADC_PARS_E440 *am)*

Назначение:

Данная функция передает в модуль в виде структуры *ADC_PARS_E440* всю необходимую информацию, которая используется при сборе данных с АЦП. Собственно использование модулем **именно** этой переданной информацией начинается **только** после выполнения интерфейсной функции ***START_ADC****()*. Формат структуры *ADC_PARS_E440* приведен в *§ 2.5.1.1. "Структура ADC_PARS_E440"*, а назначение отдельных ее полей описано ниже.

Поле *am->CorrectionEnabled* позволяет модулю по желанию пользователя осуществлять корректировку получаемых с АЦП данных. Т.о. в этом случае из модуля в PC будут поступать уже откорректированные данный с АЦП. Если *am->CorrectionEnabled* равно *true*, то корректировка разрешена, если *false* – нет. При использовании корректировки сами корректировочные коэффициенты должны находиться в поле *am->CalibrKoefAdc* (см. ниже).

Поле *am->InputMode* определяет следующие режимы ввода данных с АЦП:

- ✓ *am->InputMode=0* — отсутствие какой-либо синхронизации ввода;
- ✓ *am->InputMode=1* — цифровая синхронизация начала (старта) сбора;
- ✓ *am->InputMode=2* — покадровая цифровая синхронизация;
- ✓ *am->InputMode=3* — аналоговая синхронизация по логическому каналу АЦП.

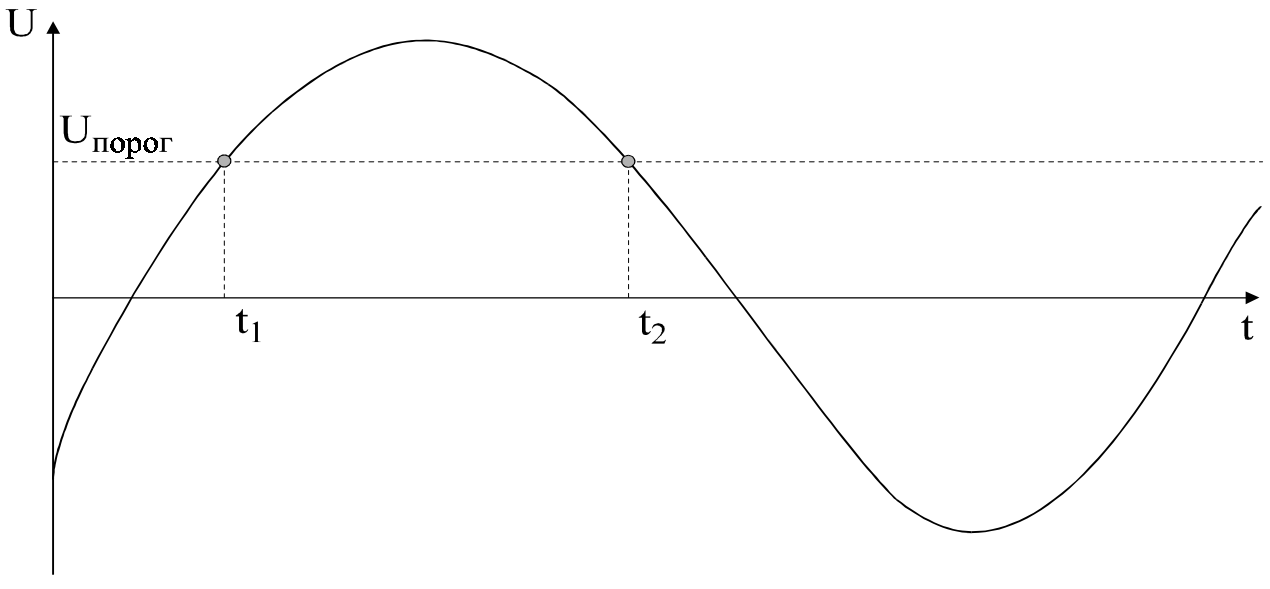
При цифровой синхронизации начала (старта) сбора модуль начинает осуществлять ввод данных с АЦП только **после** прихода отрицательного перепада (⎍) ТТЛ-совместимого одиночного импульса на вход **TRIG** аналогового разъема *DRB-37M*. Длительность этого синхроимпульса должна быть не менее 50 *нс*.

При покадровой цифровой синхронизации после прихода синхроимпульса (см. выше) модуль собирает отсчеты **только** одного кадра. После окончания сбора кадра ожидается приход следующего импульса синхронизации и т.д.

При аналоговой синхронизации модуль начинает собирать данные с АЦП **только** после выполнения определенных соотношений между полученным значением отсчета с заданного аналогового синхроканала и заданным пороговым значением (см. ниже).

Для задания вышеописанных режимов синхронизации ввода данных можно также использовать предопределенные константы: `NO_SYNC_E440`, `TTL_START_SYNC_E440`, `TTL_KADR_SYNC_E440`, `ANALOG_SYNC_E440`.

Поля *am->SynchroAdType*, *am->SynchroAdMode*, *am->SynchroAdChannel*, *am->SynchroAdPorog* используются исключительно при аналоговой синхронизации ввода данных. При этом различные моменты старта аналоговой синхронизации приведены на следующем рисунке:



Если, например, задана аналоговая синхронизация по **переходу** (*am->SynchroAdType*≠0) ‘*снизу-вверх*’ (*am->SynchroAdMode*=0) при пороговом значении (в кодах АЦП) равном *am->SynchroAdPorog*=$U_{порог}$, то модуль начнет собирать данные только после наступления момента времени $\mathbf{t_1}$, т.е. тогда, когда уровень входного сигнала на логическом канале синхронизации *am->SynchroAdChannel* пересечет пороговую линию в направлении снизу вверх. Аналогично если задан **переход** ‘*сверху-вниз*’ (*am->SynchroAdMode*≠0), то старт начало сбора наступит в момент времени $\mathbf{t_2}$, когда уровень входного сигнала на синхроканале пересечет пороговую линию в направлении сверху вниз. Если же аналоговая синхронизация задается по **уровню** (*am->SynchroAdType*=0), то сбор модулем данных начнется, когда уровень входного сигнала на синхроканале окажется либо выше (*am->SynchroAdMode*=0), либо ниже (*am->SynchroAdMode*≠0) пороговой линии $U_{порог}$.

Все вышесказанное относительно синхронизации ввода данных можно вкратце свести к следующему:

- поле *am->InputMode* может принимать следующие значения:
  - ✓ *am->InputMode*=**0** — отсутствие какой-либо синхронизации ввода данных с АЦП (остальные параметры синхронизации не используются);
  - ✓ *am->InputMode*=**1** — цифровая синхронизация начала (старта) сбора (остальные параметры синхронизации не используются);
  - ✓ *am->InputMode*=**2** — покадровая цифровая синхронизация ввода данных с АЦП (остальные параметры синхронизации не используются);
  - ✓ *am->InputMod*=**3** — аналоговая синхронизация по выбранному логическому каналу АЦП;
- поле *am->SynchroAdType* может принимать следующие значения:
  - ✓ *am->SynchroAdType*=**0** — аналоговая синхронизация по **уровню**,
  - ✓ *am->SynchroAdType*≠**0** — аналоговая синхронизация по **переходу**;
- поле *am->SynchroAdMode* может принимать следующие значения:
  - ✓ *am->SynchroAdMode*=**0:**
    - – аналоговая синхронизация по **уровню** — ‘*выше*’,
    - – аналоговая синхронизация по **переходу** — ‘*снизу-вверх*’,
  - ✓ *am->SynchroAdMode*≠**0:**
    - – аналоговая синхронизация по **уровню** — ‘*ниже*’,
    - – аналоговая синхронизация по **переходу** — ‘*сверху-вниз*’;
- поле *am->SynchroAdChannel* – логический номер синхроканала АЦП для аналоговой синхронизации;
- поле *am->SynchroAdPorog* – пороговое значение (в кодах АЦП) для аналоговой синхронизации;

Поле *am->ChannelsQuantity* определяет количество активных логических каналов АЦП для сбора данных с АЦП (не более 128).

Поле *am->ControlTable[]* задает управляющую таблицу (массив) логических каналов, количество которых равно *ChannelsQuantity*. Эта таблица используется модулем при работе с АЦП для задания циклической последовательности отсчетов с входных аналоговых каналов.

При вызове данной интерфейсной функции в полях *am->AdcRate* и *am->InterKadrDelay* должны находиться такие важные параметры функционирования модуля, как частота оцифровки данных *AdcRate* (частота работы АЦП, обратная величина межканальной задержки) и межкадровая задержка *InterKadrDelay*. При этом поле *am->AdcRate* задаётся в кГц, а *am->InterKadrDelay* – в мс. **После** выполнения этой функции в этих полях находятся **реально установленные** значения величин межканальной и межкадровой задержек, **максимально близкие** к изначально задаваемым. Это происходит вследствие того, что реальные значения *AdcRate* и *InterKadrDelay* не являются непрерывными величинами, а принадлежат некоему дискретному ряду. Так, в общем виде, частота работы АЦП определяется по следующей формуле: *AdcRate*=$\mathbf{F}_{clockout}/(2*(\mathbf{N}+1))$, где $\mathbf{F}_{clockout}$ – тактовая частота, установленного на модуле DSP, равная 48000 кГц, **N** – целое число. Поэтому данная функция просто вы-

числяет ближайшую к задаваемой дискретную величину *AdcRate*, передает ее в модуль (в виде целого числа **N**), а также возвращает Вам ее значение в поле *am->AdcRate*. Все то же самое верно и для межкадровой задержки, с той лишь разницей, что она задается в единицах 1/*AdcRate* (причем уже откорректированной *AdcRate*). При этом минимальное значение *AdcRate* составляет 0.366 кГц, максимальное – 400 кГц. Например, если задать *am->AdcRate* = 0, то ***FILL_ADC_PARS()*** установит и возвратит минимально возможную величину для данной переменной, т.е. 0.366 кГц. Аналогично: если задать *am->InterKadrDelay* = 0, то данная функция установит и возвратит минимально возможную межкадровую задержку, т.е. 1/*am->AdcRate*.

Поле *am->ChannelRate* является выходным для данной функции и в нем возвращается частота опроса *ChannelRate* (в кГц) фиксированного канала из управляющей таблицы (фактически это период кадра). Эта частота рассчитывается, исходя из величины *am->ChannelsQuantity*, а также уже скорректированных *am->AdcRate* и *am->InterKadrDelay*. Дополнительно про соотношения между упомянутыми выше величинами *am->ChannelsQuantity*, *am->AdcRate*,*am->InterKadrDelay* и *am->ChannelRate* см. *§ 2.3.4. "Формат кадра отсчетов"*.

Поле *am->AdcFifoBaseAddress* задает базовый адрес FIFO буфера АЦП в DSP модуля. Для данного модуля он всегда равен 0x0.

Поле *am->AdcFifoLength* задает длину FIFO буфера АЦП в DSP модуля. Для данного модуля эта величина может находиться в диапазоне от 0x40 (64) до 0x3000 (12288), а также быть обязательно кратной 0x40(64). Если переданное в функцию значение не удовлетворяет указанным требованиям, то выполняется необходимая корректировка и по завершении даной функции в поле *dm->AdcFifoLength* будет находиться **реально установленное** значение длины FIFO буфера АЦП. Передача данных из FIFO буфера АЦП модуля в PC производится порциями по *am->AdcFifoLength/2* отсчетов (по мере их готовности).

Поле *am->CalibrKoefAdc[]* содержит корректировочные коэффициенты для получаемых с АЦП данных. Управление корректировкой данных осуществляется с помощью поля *am->CorrectionEnabled*. В качестве этих коэффициентов можно использовать либо Ваши собственные, либо штатные, которые хранятся в ППЗУ модуля (см. *§ 2.3.3. "Формат пользовательского ППЗУ"* и *§ 2.5.1.3. "Структура MODULE_DESCR_E440"*).

Передаваемые параметры:

- *am* – адрес структуры типа *ADC_PARS_E440* с параметрами функционирования АЦП.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.4.4. Получение текущих параметров работы АЦП

Формат: **bool** ***GET_CUR_ADC_PARS***(*ADC_PARS_E440 *am*)

Назначение:

Данная функция считывает из модуля всю текущую информацию, которая используется при сборе данных с АЦП.

Передаваемые параметры:

- *am* – адрес структуры типа *ADC_PARS_E440* с полученными из модуля текущими параметрами функционирования АЦП.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.4.5. Получение массива данных с АЦП

Формат: **bool** ***ReadData****(SHORT *Buffer, DWORD *NumberOfWordsToRead, LPDWORD *NumberOfBytesRead, LPOVERLAPPED Overlapped)*

Назначение:

Данная функция обеспечивает ***асинхронный режим*** получения очередных *NumberOfWordsToRead* отсчетов из FIFO буфера АЦП, расположенного в памяти данных DSP модуля. Величина *NumberOfWordsToRead* должна быть в диапазоне от 32 до (1024*1024), а также быть кратной 32. В противном случае интерфейсная функция сама подкорректирует величину переменной *NumberOfWordsToRead* и по возвращении из ***ReadData()*** в ней будет находиться истинное значение количества полученных с АЦП отсчетов.

Поскольку данная функция осуществляет именно ***асинхронный режим*** приёма информации, ***ReadData()*** может вполне законно завершиться перед тем, как прекратится собственно сама операция чтения всего заказанного массива данных. При этом функция ***ReadData()*** должна вернуть *false*, а *NumberOfBytesRead* может быть равно нулю. В этом случае следующим шагом необходимо вызвать *Windows API* функцию ***GetLastError()*** и убедиться, что она вернула ERROR_IO_PENDING. Это будет означать, что все в порядке и собственно операция чтения данных из модуля продолжает успешно выполняться. Окончание же сей асинхронной операции необходимо впоследствии отслеживать с помощью соответствующих *Windows API* функций (***WaitForSingleObject()*** или ***GetOverlappedResult()***), использующих обнаружение события, предварительно указанного в структуре *Overlapped*. Подробнее см. хелп на *Windows API* функцию ***ReadFile()***, а также, например, исходные тексты прилагаемой к модулю законченной программы в директории `\Examples\BC5\ReadData` или `\Examples\BCB5\Synchro`.

Данные в массиве *Buffer*, который следует заранее определить, будут находиться в покадровом 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 модуля.

!!!ВНИМАНИЕ!!! Для того чтобы эта функция корректно функционировала**, строго необходимо**, чтобы предварительно работа АЦП была разрешена с помощью интерфейсной функции *START_ADC()*.

Передаваемые параметры:

- *Buffer* – указатель на массив, в который складываются принимаемые из модуля данные;
- *NumberOfWordsToRead* – количество отсчетов (минимум – 32, максимум –1024*1024), которые необходимо получить из модуля и положить в *Buffer;*
- *NumberOfBytesRead* – количество реально полученных байтов;
- *Overlapped* – указатель на *OVERLAPPED* структуру (см. исходники примеров).

Возвращаемое значение: *true* – функция успешно выполнена;
false – функция не выполнена (см. *замечания в 'Назначении' к этой функции*).

2.5.4.6. Ввод кадра отсчетов с АЦП

Формат: **bool** ***ADC_KADR****(SHORT *Data)*

Назначение:

С версии 2.0 в DLL библиотеке Lusbapi.dll появилась дополнительнпя интерфейсная функция, которая позволяет осуществлять ввод кадра отсчетов с АЦП модуля. Эта функция удобна для осуществления асинхронного ввода целого кадра данных с требуемых входных аналоговых каналов. Такие параметры сбора кадра, как разрешение корректировки данных с АЦП, количество опрашиваемых каналов, частота работы АЦП и т.д., должны предварительно быть заданы с помощью штатной интерфейсной функции *FILL_ADC_PARS()* (при этом информация о синхронизации ввода данных никак не используется, т.е. просто игнорируется). Массив *Data* необходимой длины для получаемых с модуля данных следует заранее определить. Более подробно смотри исходные тексты примера из директории `\Example\BC5\AdcKadr`.

Передаваемые параметры:

- *Data* – указатель на массив, в который складываются получаемые с АЦП модуля данные.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.4.7. Однократный ввод с АЦП

Формат: **bool** ***ADC_SAMPLE****(SHORT *AdcSample, WORD AdcChannel)*

Назначение:

Данная функция устанавливает заданный логический канал и осуществляет его однократное аналого-цифровое преобразование. Эта функция удобна для осуществления асинхронного ввода данных с требуемого входного аналогового канала (см. *§ 2.3.2.3. "Логический номер канала АЦП"*). Предварительно, с помощью штатной интерфейсной функции *FILL_ADC_PARS()*, можно разрешить корректировку данных с заданного канала АЦП. Более подробно смотри исходные тексты примера из директории `\Example\BC5\AdcSample`.

Передаваемые параметры:

- *AdcSample* – результат преобразования по заданному логическому каналу АЦП *AdcChannel*;
- *AdcChannel* – требуемый логический номер канала АЦП.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.5. *Функции для работы с ЦАП*

Интерфейсные функции штатной DLL библиотеки позволяют реализовывать разнообразные алгоритмы работы с ЦАП (*независимо* от состояния АЦП). Вообще-то модуль, с точки зрения работы с ЦАП, может находиться как бы в двух состояниях:

1. режим “покоя”;
2. потоковая (*перманентная*) выдача данных на ЦАП.

Функция *START_DAC()* позволяет переводить модуль во второе из этих состояний, а *STOP_DAC()* — в первое. Перед запуском ЦАП предварительно необходимо передать в модуль требуемые параметры его работы: частота работы, длина и базовый адрес FIFO буфера ЦАП. Эту операцию можно выполнить с помощью интерфейсной функции *FILL_DAC_PARS()*. Данные для выдачи на ЦАП берутся из FIFO буфера ЦАП, который расположен в памяти данных DSP модуля (см. *§ 2.4.2. “Общая структура LBIOS”*). Поэтому очень важно, прежде чем запустить ЦАП, проинициализировать этот буфер требуемыми данными (например, с помощью интерфейсной функции *PUT_DM_ARRAY()*). О формате данных передаваемых в FIFO буфер ЦАП смотри *§ 2.3.2.2. “Формат слова данных для ЦАП”*). Для ‘*подкачки*’ же в модуль *новых* данных уже в процессе работы ЦАП (т.е. после его запуска) следует пользоваться функцией *WriteData()*. При одновременной работе АЦП и ЦАП необходимо помнить, что максимально возможная пропускная способность шины **USB** для данного модуля не более 500 кСлов/с. Примеры корректного применения интерфейсных функций для работы с ЦАП можно найти в директориях `\Examples\BC5\WriteData` и `\Examples\BC5\ReadWrite`.

2.5.5.1. *Разрешение работы ЦАП*

Формат:	**bool**	***START_DAC****(void)*
Назначение: Данная функция запускает модуль на перманентную (непрерывную) выдачу данных на ЦАП. При этом данные из FIFO буфера ЦАП начинают последовательно, с заданной частотой, выводится на ЦАП. Параметры работы ЦАП предварительно передаются в модуль с помощью интерфейсной функции *FILL_DAC_PARS()*. Также предварительно перед запуском ЦАП следует проинициализировать FIFO буфер ЦАП необходимыми начальными значениями с помощью, например, интерфейсной функции *PUT_DM_ARRAY()*. Подробности о начальной инициализации FIFO буфера ЦАП и формате данных для ЦАП см. в прилагаемых к модулю примерах, а также см. *§ 2.3.2.2. “Формат слова данных для ЦАП”*). ‘*Подкачку*’ же в модуль *новых* данных (уже в ходе работы ЦАП) для FIFO буфера ЦАП можно осуществлять с помощью интерфейсной функции *WriteData()*.		
Передаваемые параметры: нет		
Возвращаемое значение:	*true* – функция успешно выполнена; *false*– функция не выполнена.	

2.5.5.2. *Запрещение работы ЦАП*

Формат:	**bool**	***STOP_DAC****(void)*
Назначение: Данная функция запрещает модулю выводить данные на ЦАП.		
Передаваемые параметры: нет		
Возвращаемое значение:	*true* – функция выполнена; *false*– функция не выполнена.	

2.5.5.3. Установка параметров работы ЦАП

Формат: **bool** ***FILL_DAC_PARS***(*DAC_PARS_E440 *dm*)

Назначение:

Данная функция передает в модуль в виде структуры *DAC_PARS_E440* все необходимые для работы ЦАП параметры. Собственно использование модулем этой информации при работе с ЦАП начинается **только** после выполнения интерфейсной функции ***START_DAC()***. Формат структуры *DAC_PARS_E440* приведен в *§ 2.5.1.2. "Структура DAC_PARS_E440"*, а назначение отдельных ее полей описано ниже.

Поле *dm->DacRate* задает частоту работы ЦАП *DacRate* в *кГц*. **После** выполнения данной интерфейсной функции в данном поле находится **реально установленное** значение частоты вывода данных на ЦАП. Это происходит вследствие того, что реальные значения *DacRate* не являются непрерывной величиной, а принадлежат некоему дискретному ряду. Так, в общем виде, частота работы ЦАП определяется по следующей формуле: $DacRate = \mathbf{F}_{clockout}/(12*(\mathbf{N}+1))$, где $\mathbf{F}_{clockout}$ – тактовая частота установленного на модуле DSP равная 48000 кГц, **N** – целое число. Поэтому данная функция просто вычисляет ближайшую к задаваемой дискретную величину *DacRate*, передает ее в модуль (в виде целого числа **N**), а также возвращает Вам ее значение в поле *dm->DacRate*. Минимальное значение *DacRate* равно 0.061 кГц, максимальное – 125 *кГц*. Данный параметр *DacRate* задается в *кГц*.

Поле *dm->DacFifoBaseAddress* задает базовый адрес FIFO буфера ЦАП в DSP модуля. Для данного модуля он всегда равен 0x3000.

Поле *dm->DacFifoLength* задает длину FIFO буфера ЦАП в DSP модуля. Для данного модуля эта величина может находиться в диапазоне от 0x40 (64) до 0xFC0 (4032), а также также быть обязательно кратной 0x40(64). Если переданное в функцию значение не удовлетворяет указанным требованиям, то выполняется необходимая корректировка и по завершении даной функции в поле *dm->DacFifoLength* будет находиться **реально установленное** значение длины FIFO буфера ЦАП. Передача ('*подкачка*') новых данных из PC в FIFO буфер ЦАП модуля производится порциями по *dm->DacFifoLength/2* отсчетов (по мере их необходимости).

Передаваемые параметры:

- *dm* – адрес структуры типа *DAC_PARS_E440* с параметрами функционирования ЦАП.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.5.4. Получение текущих параметров работы ЦАП

Формат: **bool** ***GET_CUR_DAC_PARS***(*DAC_PARS_E440 *dm*)

Назначение:

Данная функция считывает из модуля всю текущую информацию, которая используется в процессе потоковой выдачи данных на ЦАП.

Передаваемые параметры:

- *dm* – адрес структуры *DAC_PARS_E440* с полученными из модуля текущими параметрами функционирования ЦАП.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.5.5. Передача массива данных в ЦАП

Формат: **bool** ***WriteData****(WORD *Buffer, DWORD *NumberOfWordsToWrite, LPDWORD *NumberOfBytesWritten, LPOVERLAPPED Overlapped)*

Назначение:

Данная функция обеспечивает ***асинхронный режим*** передачи *NumberOfWordsToWrite* отсчетов из массива *Buffer* в FIFO буфер ЦАП модуля с целью дальнейшего их вывода на собственно ЦАП. Эта функция предназначена только для передачи *новых порций* данных в FIFO буфер ЦАП в процессе его работы. О структуре данных для ЦАП см. *§ 2.3.2.2. "Формат слова данных для ЦАП"*. Величина *NumberOfWordsToWrite* должна быть в диапазоне от 32 до (1024*1024), а также быть кратна 32. В противном случае интерфейсная функция сама подкорректирует величину переменной *NumberOfWordsToWrite*, и по возвращении из ***WriteData()*** в ней будет находиться истинное значение количества передаваемых в модуль отсчетов.

Поскольку данная функция осуществляет именно ***асинхронный режим*** передачи информации, ***WriteData()*** может вполне законно завершиться перед тем, как прекратится собственно сама операция записи в модуль всего заданного массива данных. При этом функция ***WriteData()*** может вернуть *false*, а *NumberOfBytesWritten* может быть равно нулю. В этом случае следующим шагом необходимо вызвать *Windows API* функцию ***GetLastError()*** и убедиться, что она вернула ERROR_IO_PENDING. Это будет означать, что все в порядке и собственно операция записи данных в модуль продолжает успешно выполняться. Окончание же этой асинхронной операции необходимо впоследствии отслеживать с помощью соответствующих *Windows API* функций (***WaitForSingleObject()*** или ***GetOverlappedResult()***), использующих обнаружение события, предварительно указанного в структуре *Overlapped*. Подробнее см. хелп на *Windows API* функцию ***WriteFile()***, а также исходные тексты прилагаемых к модулю законченных программ в директориях `\Example\BC5\WriteData` и `\Example\BC5\ReadWrite`.

!!!ВНИМАНИЕ!!! Для того, чтобы эта функция корректно функционировала, **строго необходимо**, чтобы предварительно работа ЦАП была разрешена с помощью интерфейсной функции *START_DAC()*.

Передаваемые параметры:

- *Buffer* – указатель на массив, из которого берутся передаваемые в модуль данные для FIFO буфера ЦАП;
- *NumberOfWordsToWrite* – количество отсчетов (минимум – 32, максимум – 1024*1024), которые необходимо передать в модуль;
- *NumberOfBytesWritten* – количество реально переданных байтов;
- *Overlapped* – указатель на *OVERLAPPED* структуру (см. исходники примеров).

Возвращаемое значение: *true* – функция успешно выполнена;
false – функция не выполнена (см. *замечания в 'Назначение' к этой функции*).

2.5.5.6. Однократный вывод на ЦАП

Формат:	**bool**	***DAC_SAMPLE****(WORD DacData, WORD DacChannel)*

Назначение:

Данная функция позволяет однократно устанавливать на указанном канале ЦАП *DacChannel* напряжение в соответствии со значением *DacData* (в кодах ЦАП). О соответствии кода ЦАП величине устанавливаемого на выходе модуля аналогового напряжения см. *§ 2.3.2.2. "Формат слова данных для ЦАП"*

Передаваемые параметры:

- *DacChannel* – требуемый номер канала ЦАП (**0** или **1**).
- *DacData* – устанавливаемое значение напряжения в кодах ЦАП (от –2048 до 2047).

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.6. Функции для работы с внешними цифровыми линиями

2.5.6.1. Разрешение выходных цифровых линий

Формат:	**bool** ***ENABLE_TTL_OUT****(bool EnableTtlOut)*
Назначение: Данная интерфейсная функция позволяет осуществлять управление доступностью всех *выходных* линий внешнего цифрового разъёма *DRB-37F*, т.е. даёт возможность перевода их в третье (высокоимпедансное) состояние и обратно. Непосредственно после подачи на модуль питания выходные цифровые линии находятся в третьем состоянии.	
Передаваемые параметры: • *EnableTtlOut* – флажок, позволяющий (*true*) либо запрещающий (*false*) использование цифровых выходных линий.	
Возвращаемое значение:	*true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.6.2. Чтение внешних цифровых линий

Формат:	**bool** ***TTL_IN****(WORD *TtlIn)*
Назначение: Данная интерфейсная функция осуществляет чтение состояния $16^{ти}$ входных цифровых ТТЛ линий модуля с внешнего цифрового разъёма *DRB-37F*.	
Передаваемые параметры: • *TtlIn* – переменная, содержащая побитовое состояние входных цифровых линий модуля.	
Возвращаемое значение:	*true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.6.3. Вывод на внешние цифровые линии

Формат:	**bool** ***TTL_OUT****(WORD TtlOut)*
Назначение: Данная интерфейсная функция осуществляет установку $16^{ти}$ выходных ТТЛ линий модуля на внешнем цифровом разъёме *DRB-37F* в соответствии с битами передаваемого параметра *TtlOut*. Работа с цифровыми выходами предварительно **должна быть разрешена** с помощью интерфейсной функции *ENABLE_TTL_OUT()*.	
Передаваемые параметры: • *TtlOut* – переменная, содержащая побитовое состояние устанавливаемых выходных цифровых линий модуля.	
Возвращаемое значение:	*true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.7. Функции для работы с пользовательским ППЗУ

На модуле *E-440* установлено последовательное пользовательское ППЗУ емкостью 64 ячейки×16 бит. Первые 32 ячейки данного ППЗУ используются под хранения служебной информации: название модуля, тип DSP, серийный номер, коэффициенты для корректировки отсчётов АЦП и ЦАП и т.д. А оставшеся 32 ячейки предназначены для целей пользователя.

2.5.7.1. Разрешение/запрещение записи в ППЗУ

Формат: **bool** ***ENABLE_FLASH_WRITE****(bool EnableFlashWrite)*

Назначение:

Данная интерфейсная функция разрешает (*true*) либо запрещает (*false*) режим записи в пользовательское ППЗУ модуля с помощью штатной интерфейсной функции *WRITE_FLASH_WORD()*. Следует помнить, что **после** завершения Вами всех требуемых операций записи информации в ППЗУ, **строго необходимо** запретить с помощью данной интерфейсной функции режим записи.

Передаваемые параметры:

- *EnableFlashWrite* – переменная может принимать следующие значения:
  - ✓ если *true*, то режим записи в ППЗУ разрешен,
  - ✓ если *false*, то режим записи в ППЗУ запрещен.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.7.2. Запись слова в ППЗУ

Формат: **bool** ***WRITE_FLASH_WORD****(WORD FlashAddress, SHORT FlashWord)*

Назначение:

Данная интерфейсная функция выполняет запись 16^{ти} битного слова *FlashWord* в ячейку пользовательского ППЗУ с номером *FlashAddress*. Перед началом операции записи в ППЗУ **необходимо** разрешить её с помощью интерфейсной функции *ENABLE_FLASH_WRITE()*. **После** окончания цикла записи всей требуемой информации **строго необходимо** запретить режим записи в ППЗУ с помощью той же функции *ENABLE_FLASH_WRITE()*. Т.к. в первых 32 ячейках ППЗУ находится служебная информация, которая используется штатной DLL в процессе работы с модулем, то для пользователя доступны адреса ячеек **только** с 32 по 63.

Передаваемые параметры:

- *FlashAddress* – номер ячейки, куда будет записано слово *FlashWord*;
- *FlashWord* – слово, значение которого должно быть записано в ППЗУ.

Возвращаемое значение: *true* – функция успешно выполнена;
false– функция не выполнена.

2.5.7.3. Чтение слова из ППЗУ

Формат: **bool** ***READ_FLASH_WORD****(WORD FlashAddress, SHORT *FlashWord)*
Назначение: Данная интерфейсная функция возвращает значения слова, находящегося в ячейке пользовательского ППЗУ с номером *FlashAddress*.
Передаваемые параметры: • *FlashAddress* – номер ячейки, откуда будет считано слово; • *FlashWord* – считанное значение.
Возвращаемое значение: *true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.7.4. Чтение служебной информации из ППЗУ

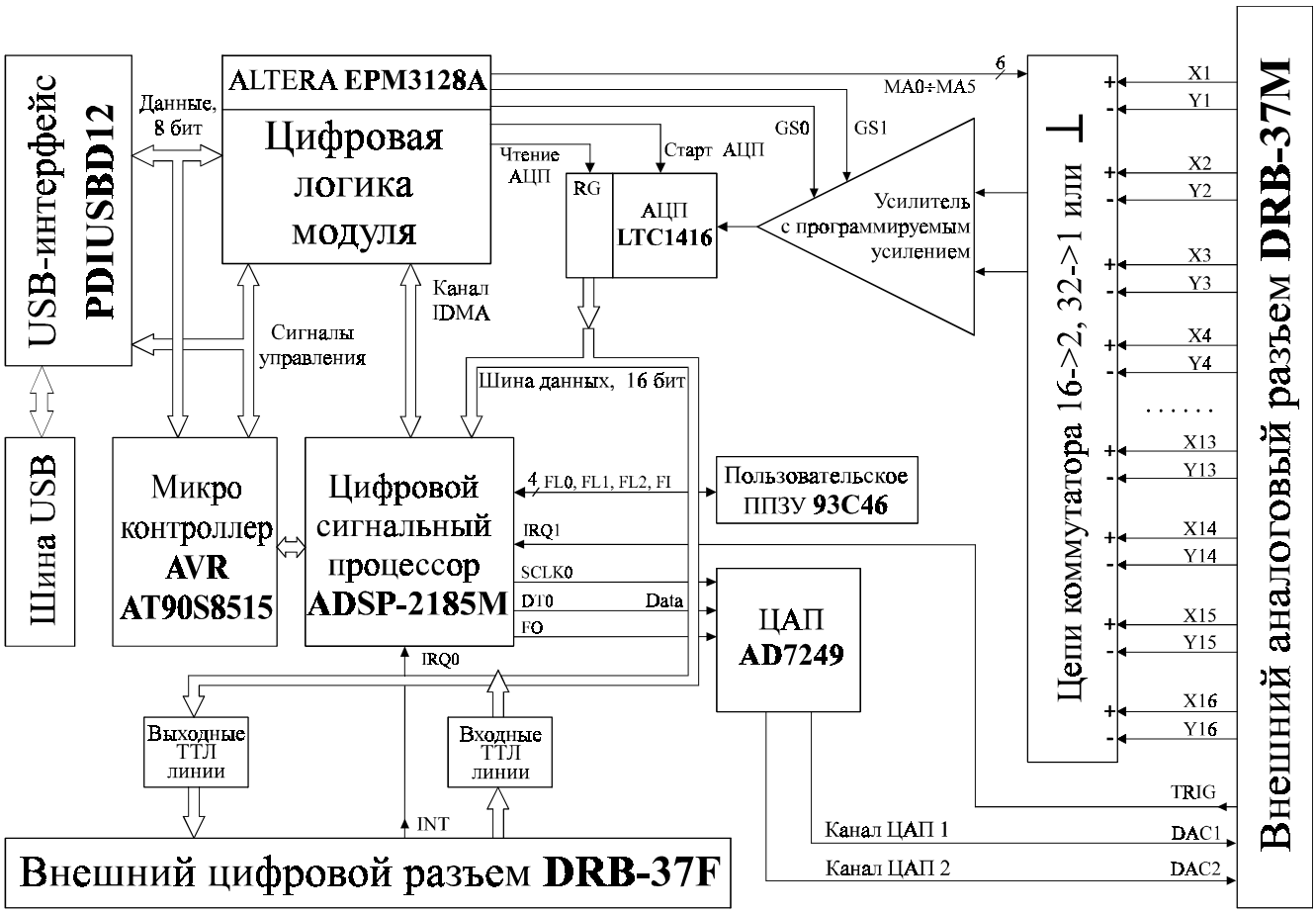
Формат: **bool** ***GET_MODULE_DESCR****(MODULE_DESCR_E440 *md)*
Назначение: Данная интерфейсная функция осуществляет чтение информации из служебной области пользовательского ППЗУ в структуру типа *MODULE_DESCR_E440* (см. *§ 2.5.1.3 "Структура MODULE_DESCR_E440"*). Эта информация требуется при работе с некоторыми интерфейсными функциями штатной DLL библиотеки. Поэтому данную функцию, во избежания непредсказуемого поведения Ваших приложений, следует **обязательно** вызывать непосредственно после загрузки в модуль драйвера *LBIOS* и проверки его работоспособности (см. *§ 2.4.1. "Общий подход к работе с интерфейсными функциями"*)
Передаваемые параметры: • *md* – указатель на структуру типа *MODULE_DESCR_E440*, в которую заносится вся служебная информация из ППЗУ модуля.
Возвращаемое значение: *true* – функция успешно выполнена; *false*– функция не выполнена.

2.5.7.5. Запись служебной информации в ППЗУ

Формат: **bool** ***SAVE_MODULE_DESCR****(MODULE_DESCR_E440 *md)*
Назначение: Данная интерфейсная функция позволяет сохранять всю служебную информацию из структуры типа *MODULE_DESCR_E440* в пользовательском ППЗУ модуля (см. *§ 2.5.1.3. "Структура MODULE_DESCR_E440"*). **!!!Внимание!!! Применять данную функцию нужно только в случае крайней необходимости. Например, когда по тем или иным обстоятельствам испортилось содержимое служебной информации в пользовательском ППЗУ.**
Передаваемые параметры: • *md* – указатель на структуру типа *MODULE_DESCR_E440*, из которой информация заносится в служебную область ППЗУ модуля.
Возвращаемое значение: *true* – функция успешно выполнена; *false*– функция не выполнена.

3. НИЗКОУРОВНЕВОЕ ОПИСАНИЕ МОДУЛЯ E-440

3.1. Структурная схема модуля E-440



Модуль E-440

Как видно из приведенной выше структурной схемы, функционально модуль *E-440* как бы состоит из двух частей. Одна из них полностью отвечает за **USB**-интерфейс модуля с компьютером PC (хост-компьютер) и включает в себя микросхему **USB** интерфейса *PDIUSBD12* (фирмы *Philips Semiconductors*) и микроконтроллер *AVR AT90S8515* (фирмы *Atmel Corporation*). В ППЗУ микроконтроллера при наладке модуля в ***ЗАО "Л-Кард"*** зашивается специальный программный драйвер, всецело отвечающий за корректную работу *E-440* с шиной **USB** и взаимодействие с цифровым сигнальным процессором (DSP). Другая часть находится под полным функциональным контролем цифрового сигнального процессора. Именно с помощью него можно достаточно гибко, чисто программном образом, эффективно управлять фактически всей периферией на модуле, т.е. обеспечивать надлежащее функционирование цепей коммутации и усиления входных аналоговых сигналов, управлять работой АЦП *LTC1416* (фирмы *Linear Technology Corporation*) и ЦАП *AD7249* (фирмы *Analog Devices, Inc.*), контролировать состояния входных и выходных цифровых линий и т.д. Как уже сказано, на данном модуле установлен современный высокопроизводительный цифровой сигнальный процессор фирмы ***Analog Devices, Inc.*** *ADSP-2185M*, работающий на тактовой частоте $\mathbf{F}_{clockout}$=48000.0 кГц. Внутренняя архитектура данного сигнального процессора специальным образом оптимизирована для реализации таких алгоритмов обработки информации, как цифровая фильтрация, спектральный анализ и т.д. Сам процессор имеет достаточно большую внутреннюю оперативную память программ (16 КСлов) и внутреннюю память данных (16 КСлов). Наличие на модуле *E-440* такого мощного современного DSP обеспечивает Вам, при необходимости и определенном усердии, возможность самостоятельного применения чрезвычайно гибких специализиро-

ванных методов (алгоритмов) управления всей доступной периферией. Также можно переносить часть достаточно сложных операций по обработке информации на сам модуль (например, выполнять Быстрое Преобразование Фурье).

На модуле *E-440* также устанавливается микросхема последовательного электрически стираемого программируемого ПЗУ (Serial EEPROM) типа *93C46*, с организацией памяти 64 слов по 16 бит. В ней при наладке в ***ЗАО "Л-Кард"*** прописывается служебная информация, корректировочные коэффициенты для АЦП и ЦАП, а также предусматривается область для пользовательских нужд. Подробнее о формате данных в ППЗУ см *§ 2.3.3 "Формат пользовательского ППЗУ"*.

Применение на модуле *E-440* микросхемы программируемой логической матрицы (ПЛМ) *EPM3128A* (фирмы *Altera Corporation*) позволяет разместить в ней практически всю логическую часть устройства, обеспечивая функциональное взаимодействие основных составных сегментов модуля. Кроме того, использование ПЛМ существенно упрощает процедуру разработки и монтажа изделия, а также максимально способствует достижению минимальных габаритных размеров модуля.

3.2. Организация USB интерфейса

3.2.1. Общие сведения о USB

USB (**U*niversal Serial Bus*** — универсальная последовательная шина, которая является на сегодняшний день широко распространенным промышленным стандартом расширения архитектуры персонального компьютера. В своё время разработка данного стандарта была инициирована весьма авторитетными компьютерными фирмами – Intel, DEC, IBM, NorthenTelecom и Compaq. Версия первого утвержденного варианта стандарта **USB** (спецификация 1.0) появилась по компьютерным мерка довольно давно – 15 января 1996 года. Следующая *спецификация 1.1* была принята 23 сентября 1998 года. И, наконец, действующая на настоящий момент *спецификация 2.0* – 27 апреля 2000 года. Собственно, сами эти спецификации можно совершенно свободно скачать с сайта *www.usb.org*. Таким образом, согласно спецификации шина **USB** может с легкостью использоваться для подключения к компьютеру самых различных устройств, а именно: клавиатур, мышей, джойстиков, модемов, принтеров, сканеров, приводов CD-ROM, аудиоустройств (например, микрофонов и колонок), цифровых фото- и видеокамер, а также множества другой мультимедийной периферии. Т.о. мы видим, что для взаимодействия с компьютером данный стандарт может с успехом применяться для самого широкого и разнообразного спектра периферийных устройств.

Все преимущества применения именно шины **USB** можно описывать довольно долго (очень много интересного можно найти в Интернете). Немного осветим всего лишь один важный аспект применения рассматриваемой шины **USB**. При подключении не **USB**-устройств к компьютеру (например, в процессе его модернизации) пользователям, как правило, все ещё приходится ‘сражаться’ с малопонятными настройками и элементами архитектуры PC, включая запросы прерываний (IRQ), каналы прямого доступа к памяти DMA и адреса портов ввода-вывода. Да и с подсоединением кабелей к разъемам, которых на задней стенке системного блока полным-полно, у многих возникают затруднения. Все эти и другие сложности с подключением к персональному компьютеру необходимых пользователю дополнительных устройств не радуют и самих производителей, поскольку противоречат самым современным требованиям к простоте использования и инсталляции внешней периферии. В том числе, исходя и из этих требований, группа лидирующих в компьютерной индустрии компаний создала и активно продвигает на рынок стандарт, получивший название универсальной последовательной шины, т.е. **USB**. Также одно из главных преимуществ шины **USB** состоит как раз в поддержке так называемого "горячего" подключения и отключения к PC периферийных устройств. Это означает, что пользователи могут подсоединить к компьютеру новое устройство и начать с ним работать, не выключая и не перезагружая систему.

Интерфейс **USB** позволяет осуществлять обмен информацией между хост-компьютером и множеством различных одновременно доступных периферийных устройств, обеспечивая при этом возможность работы на трёх различных скоростях :

- низкая скорость (*Low Speed* – ***LS***) – 1,5 Мбит/с (спецификация *USB 1.1*);
- полная скорость (*Full Speed* – ***FS***) – 12 Мбит/с (спецификация *USB 1.1*);
- высокая скорость (*High Speed* – ***HS***) – 480 Мбит/с (спецификация *USB 2.0*).

Фактически интерфейс соединяет между собой ***хост-контроллер*** **USB** и периферийные устройства. Хост-контроллер **USB** находится внутри персонального компьютера (в сущности, он является программно-аппаратной подсистемой персонального компьютера) и полностью конролирует работу всего интерфейса, т.е. все передачи данных по интерфейсу инициируются именно хост-контроллером. Для того чтобы к одному порту **USB** можно было подключать более одного устройства, применяются ***хабы*** (*hub* – устройство, обеспечивающее подключение к интерфейсу других устройств). ***Корневой хаб*** (*root hub*) находится внутри компьютера и подключен непосредственно к хосту. В интерфейсе **USB** используется специальный термин ***"функция"*** – это логически законченное устройство, выполняющее какую-либо специфическую функцию (в нашем случае это и есть модуль *E-440*).

Всего в интерфейсе **USB** может быть использовано четыре типа пересылок информации, а именно:

- ***управляющая пересылка*** (*control transfer*). Используется для конфигурации устройства, а также для других специфических для конкретного устройства целей.
- ***потоковая пересылка*** (*bulk transfer*). Используется для передачи относительно большого объема информации. Характеризуется гарантированной безошибочной передачей данных между хост-контроллером и устройством посредством обнаружения ошибок в процессе передачи и повторного запроса информации.
- ***пересылка с прерыванием*** (*interrupt transfer*). Используется для передачи относительно небольшого объема информации, для которого особенно важна своевременная его пересылка. Имеет ограниченную длительность и повышенный приоритет относительно других типов пересылок.
- ***изохронная пересылка*** (*isochronous transfer*), также называется потоковой пересылкой *реального* времени. Информация, передаваемая в такой пересылке, требует реального масштаба времени при ее создании, пересылке и приеме.

В модуле *E-440* из всего этого набора используются только ***управляющие*** **и** ***потоковые*** пересылки.

Спецификация **USB** определяет набор стандартных операций, которые должны поддерживать абсолютно все **USB**-устройства. Эти стандартные операции гарантируют некоторую согласованность в базовом поведении устройств, когда они подключаются к шине. Кроме того, разработчики **USB**-периферии при желании могут определять дополнительные операции **USB**-устройств, чтобы адекватно реализовывать необходимые алгоритмы их функционирования.

3.2.2. Интерфейс AVR с USB шиной

В связи с тем, что в интерфейсе **USB** реализован довольно сложный протокол обмена информацией, в устройстве сопряжения с интерфейсом **USB** необходимо применять микропроцессорный блок, обеспечивающий полную поддержку протокола. В качестве такового на модуле *E-440* используется согласованная связка микросхем **USB** интерфейса *PDIUSBD12* (фирмы *Philips Semiconductors*) и микроконтроллера *AVR AT90S8515* (фирмы *Atmel Corporation*). При наладке модуля *E-440* в ***ЗАО "Л-Кард"*** в ППЗУ микроконтроллера AVR зашивается специально разработанный драйвер, одна из основных задач которого — обеспечить корректный интерфейс модуля с хост-компьютером. В данном описании мы не будем глубоко вдаваться в технические подробности реализации **USB** интерфейса, а только в общем виде обрисуем заложенные в драйвер особенности, которые совершенно необходимы для надлежащего программирования модуля при создании своего приложения в PC.

Итак, помимо набора *стандартных запросов*, которые поступают с хост-компьютера и должны пониматься всеми без исключения **USB**-устройствами, в модуле *E-440* организован целый ряд специализированных запросов (‘*Vendor Request*’) для целей реализации требуемых алгоритмов функционирования модуля. Также как и для всех стандартных запросов, для посылки запроса такого рода используется ***управляющая*** пересылка. Каждый специализированный запрос имеет свой уникальный номер и по мере поступления в AVR обрабатывается соответствующим образом, т.е. AVR выполняет действия, однозначно предопределенные номером запроса. Например, в ответ на поступление V_RESET_DSP_E440 запроса, микроконтроллер AVR просто осуществляет сброс DSP модуля. С программной точки зрения посылка из хост-компьютера требуемого запроса типа ‘*Vendor Request*’ в модуль *E-440* осуществляется с помощью обычной *Windows API* функции ***DeviceIoControl()*** (подробнее см. хелп на эту функцию и исходные тексты штатной DLL библиотеки в директории \DLL). Далее мы попробуем чуть-чуть подробнее затронуть описание форматов функции ***DeviceIoControl()*** и запросов типа ‘*Vendor Request*’ применительно к конкретной реализации модуля *E-440*.

3.2.2.1. Функция DeviceIoControl()

Стандартная *Windows API* функция ***DeviceIoControl()*** является одной из самых ключевых функций в деле организации взаимодействия Вашего приложения в PC с модулем *E-440*. Вот как эта функция объявлена в *Windows* (см. хелп на эту функцию):

```
BOOL DeviceIoControl(
HANDLE hDevice,                 // handle to device of interest
DWORD dwIoControlCode,          // control code of operation to perform
LPVOID lpInBuffer,              // pointer to buffer to supply input data
DWORD nInBufferSize,            // size of input buffer (in bytes)
LPVOID lpOutBuffer,             // pointer to buffer to receive output data
DWORD nOutBufferSize,           // size of output buffer (in bytes)
LPDWORD lpBytesReturned,        // pointer to variable to receive output byte count
LPOVERLAPPED lpOverlapped       // pointer to overlapped structure for asynchronous
                                // operation
);
```

Из описания данной функции следует, что её следует применять для передачи управляющих кодов dwIoControlCode непосредственно в драйвер с дескриптором (идентификатором) hDevice. Управляющие коды dwIoControlCode определяют действия драйвера, которые он должен выполнить. Для модуля *E-440* в драйвере предусмотрены три управляющих кода (см. файл DLL\bulkioct.h), мнемоники которых приведены ниже:

1. DIOC_SEND_COMMAND. Данный код предписывает драйверу выполнить соответствующую ***управляющую*** пересылку, т.е. посылку в модуль запроса типа ‘V*endor Request*’ с требуемым номером.

2. DIOC_RESET_PIPE1. Данный код предписывает драйверу выполнить инициализацию (сброс) канала, обеспечивающего ***потоковые пересылки*** для записи данных в модуль (Reset Write Pipe).

3. DIOC_RESET_PIPE3. Данный код предписывает драйверу выполнить инициализацию (сброс) канала, обеспечивающего ***потоковые пересылки*** для чтении данных из модуля (Reset Read Pipe).

При этом переменные lpInBuffer, nInBufferSize, lpOutBuffer, nOutBufferSize, задающие параметры требуемой ***управляющей*** пересылки, имеют смысл только при управляющем коде *DIOC_SEND_COMMAND*, при остальных кодах они должны быть равны NULL. Собственно сам запрос ‘*Vendor Request*’ может состоять как бы из двух различных фаз (стадий), а именно: сначала организуется пересылка *Setup packet*, а затем (если необходимо) *Data stage* (подробное изложение формата запроса ***управляющей*** пересылки можно найти в ***§ 9.3 “USB Device Requests“*** *спецификации 1.1* шины **USB**). Т.о. фаза *Setup packet* всегда присутствует в запросе, в то время как *Data stage* фаза вполне законно может и отсутствовать (в зависимости от требований). Стадия

Setup packet состоит из пересылки по шине **USB** строго структуризированных восьми байтов информации, в которых передаются такие важные параметры как номер запроса, тип запроса (в рассматриваемом случае это всегда *Vendor Request*), направление передачи данных в фазе *Data stage*, количество байт необходимых переслать в фазе *Data stage* (если этот параметр равен нулю, то считается, что фаза *Data stage* полностью отсутствует, т.е. нет потребности в дополнительной передачи данных) и т.д. Массив `lpInBuffer` состоящий из четырех элементов типа *WORD* (параметр `nInBufferSize`) как раз предназначен для организации посылки *Setup packet* с требуемыми параметрами. Массив `lpOutBuffer` длиной `nOutBufferSize` байт, если это необходимо, используется для организации передачи данных в фазе *Data stage*. Подробности этих параметров будут рассмотрены ниже при описании доступных штатной DLL библиотеке номеров запросов типа '*Vendor Request*'.

Переменная `lpBytesReturned` задает обычный счетчик полученных байтов (если оные передаются из драйвера).

Так как функция ***DeviceIoControl()*** **всегда** выполняется в синхронном режиме, переменная `lpOverlapped` **всегда** должна быть установлена в NULL.

3.2.2.2. Запрос V_RESET_DSP_E440

Запрос типа 'V*endor Request*' с номером V_RESET_DSP_E440 (см. файл `E440.h`) предназначен для выполнения процедуры сброса цифрового сигнального процессора, установленного на модуле. Т.е. при поступлении запроса с данным номером микроконтроллер AVR осуществляет аппаратный сброс DSP. С программной точки зрения пересылка указанного запроса выполняется очень просто и выглядит это примерно так:

```
WORD InBuf[4];      // буфер под параметры Setup packet
DWORD cbRet=0;      // байтовый счетчик полученных байтов

InBuf[0] = 0;       // направление передачи данных в фазе Data stage;
                    // в данном запросе не используется, т.к. этой стадии не будет
InBuf[1] = V_RESET_DSP_E440;     // номер запроса
InBuf[2] = 0x0;     // дополнительный параметр (в данном запросе не используется)
InBuf[3] = 0x0;     // дополнительный параметр (в данном запросе не используется)
DeviceIoControl(hDevice,
                DIOC_SEND_COMMAND,
                &InBuf,
                sizeof(InBuf),
                NULL,       // в данном запросе фазы Data stage не будет
                0,          // в данном запросе фазы Data stage не будет
                &cbRet,
                NULL);
```

3.2.2.3. Запрос V_PUT_ARRAY_E440

Запрос типа 'V*endor Request*' с номером V_PUT_ARRAY_E440 (см. файл `E440.h`) предназначен для выполнения процедуры записи массива слов в память цифрового сигнального процессора, установленного на модуле. Т.е. при поступлении запроса с данным номером микроконтроллер AVR осуществляет процедуру записи поступающей информации по каналу *IDMA DSP* в память сигнального процессора (чуть подробнее о IDMA DSP см. *§ 3.4.1 "Общие сведения"*). С программной точки зрения пересылка указанного запроса выполняется достаточно просто и выглядит следующим образом:

```
WORD InBuf[4];      // буфер под параметры Setup packet
DWORD cbRet=0;      // байтовый счетчик полученных байтов
```

```
InBuf[0] = 0;          // будет передача данных в модуль в фазе Data stage;
InBuf[1] = V_PUT_ARRAY_E440;      // номер запроса
InBuf[2] = BaseAddress;       // базовый адрес в памяти DSP, начиная с которого
                              // будет осуществляться запись массива данных
InBuf[3] = 0x0;       // дополнительный параметр (в данном запросе не используется)
DeviceIoControl(hDevice,
                DIOC_SEND_COMMAND,
                &InBuf,
                sizeof(InBuf),
                Data,       // массив с передаваемыми в фазе Data stage данными
                NBytes,     // в фазе Data stage передается NBytes байтов данных
                &cbRet,
                NULL);
```

3.2.2.4. Запрос V_GET_ARRAY_E440

Запрос типа ‘V*endor Request*’ с номером V_GET_ARRAY_E440 (см. файл E440.h) предназначен для выполнения процедуры чтения массива слов из памяти цифрового сигнального процессора, установленного на модуле. Т.е. при поступлении запроса с данным номером микроконтроллер AVR осуществляет процедуру чтения информации по каналу *IDMA DSP* из памяти сигнального процессора (чуть подробнее о IDMA DSP см. *§ 3.4.1 “Общие сведения”*). С программной точки зрения пересылка указанного запроса выполняется достаточно просто и выглядит следующим образом:

```
WORD InBuf[4];          // буфер под параметры Setup packet
DWORD cbRet=0;          // байтовый счетчик полученных байтов

InBuf[0] = 1;          // будет приём данных из модуля в фазе Data stage;
InBuf[1] = V_GET_ARRAY_E440;      // номер запроса
InBuf[2] = BaseAddress;       // базовый адрес в памяти DSP, начиная с которого
                              // будет осуществляться чтение массива данных
InBuf[3] = 0x0;       // дополнительный параметр (в данном запросе не используется)
DeviceIoControl(hDevice,
                DIOC_SEND_COMMAND,
                &InBuf,
                sizeof(InBuf),
                Data,       // массив для получаемых в фазе Data stage данных
                NBytes,     // в фазе Data stage передается NBytes байтов данных
                &cbRet,
                NULL);
```

3.2.2.5. Запрос V_START_ADC_E440

Запрос типа ‘V*endor Request*’ с номером V_START_ADC_E440 (см. файл E440.h) предназначен для инициализации внутренних переменных и структур драйвера микроконтроллера AVR с целью подготовить их для дальнейшей работы AVR по каналу *IDMA DSP* с *FIFO буфером АЦП* (чуть подробнее о IDMA DSP см. *§ 3.4.1 “Общие сведения”*). С программной точки зрения пересылка указанного запроса выполняется достаточно просто и выглядит примерно так:

```
WORD InBuf[4];          // буфер под параметры Setup packet
DWORD cbRet=0;          // байтовый счетчик полученных байтов

InBuf[0] = 0;          // направление передачи данных в фазе Data stage;
                       // в данном запросе не используется, т.к. этой стадии не будет
```

```
InBuf[1] = V_START_ADC_E440;        // номер запроса
InBuf[2] = 0x0 | DM_E440;           // базовый адрес FIFO буфера АЦП
InBuf[3] = AdcFifoLength/2.0;       // половина длины FIFO буфера АЦП
DeviceIoControl(hDevice,
                DIOC_SEND_COMMAND,
                &InBuf,
                sizeof(InBuf),
                NULL,               // в данном запросе фазы Data stage не будет
                0,                  // в данном запросе фазы Data stage не будет
                &cbRet,
                NULL);
```

Разумеется, что сама по себе посылка только данного запроса не обеспечивает собственно запуска АЦП. Для этого необходимо выполнить еще два дополнительных шага:

1. Осуществить посылку с помощью функции ***DeviceIoControl()*** в драйвер устройства ещё одного управляющего кода *DIOC_RESET_PIPE3*, т.е.:

```
DeviceIoControl(hDevice,
                DIOC_RESET_PIPE3/* reset Read Pipe */,
                NULL, NULL, NULL, NULL, &cbRet, NULL);
```

2. Послать с помощью штатной интерфейсной функции *SEND_COMMAND()* в DSP модуля команду запуска АЦП

```
SEND_COMMAND(C_START_ADC_E440);
```

После этих нехитрых манипуляций можно быть уверенным, что запуск АЦП модуля успешно осуществлен. Впоследствии, если в этом есть необходимость, можно организовать передачу из модуля (точнее из *FIFO буфера АЦП*) в хост-компьютер уже собранной с АЦП аналоговой информации. В модуле *E-440* за процедуру подобного рода отвечает *потоковая пересылка* данных из устройства в PC (подробнее см. *§ 3.2.2.11 “Потоковые пересылки”*).

3.2.2.6. Запрос V_START_DAC_E440

Запрос типа ‘V*endor Request*’ с номером V_START_DAC_E440 (см. файл E440.h) предназначен для инициализации внутренних переменных и структур драйвера микроконтроллера AVR с целью подготовить их для дальнейшей работы AVR по каналу *IDMA DSP* с *FIFO буфером ЦАП* (чуть подробнее о IDMA DSP см. *§ 3.4.1 “Общие сведения”*). С программной точки зрения пересылка указанного запроса выполняется достаточно просто и выглядит примерно так:

```
WORD InBuf[4];          // буфер под параметры Setup packet
DWORD cbRet=0;          // байтовый счетчик полученных байтов

InBuf[0] = 0;           // направление передачи данных в фазе Data stage;
                        // в данном запросе не используется, т.к. этой стадии не будет
InBuf[1] = V_START_DAC_E440;        // номер запроса
InBuf[2] = 0x3000 | DM_E440;        // базовый адрес FIFO буфера ЦАП
InBuf[3] = DacFifoLength/2.0;       // половина длины FIFO буфера ЦАП
DeviceIoControl(hDevice,
                DIOC_SEND_COMMAND,
                &InBuf,
                sizeof(InBuf),
                NULL,               // в данном запросе фазы Data stage не будет
                0,                  // в данном запросе фазы Data stage не будет
                &cbRet,
                NULL);
```

Разумеется, что сама по себе посылка только данного запроса не обеспечивает собственно запуска ЦАП. Для этого необходимо выполнить еще два дополнительных шага:

1. Осуществить посылку с помощью функции ***DeviceIoControl()*** в драйвер устройства ещё одного управляющего кода *DIOC_RESET_PIPE1*, т.е.:

```
DeviceIoControl(hDevice,
                DIOC_RESET_PIPE1/* reset Write Pipe */,
                NULL, NULL, NULL, NULL, &cbRet, NULL);
```

2. Послать с помощью штатной интерфейсной функции *SEND_COMMAND()* в DSP модуля команду запуска ЦАП

```
SEND_COMMAND(C_START_DAC_E440);
```

После этих нехитрых манипуляций можно быть уверенным, что запуск ЦАП модуля успешно осуществлен. Впоследствии, если в этом есть необходимость, можно организовать передачу в модуль (точнее в *FIFO буфер ЦАП*) дополнительных данных с целью их дальнейшего вывода на собственно сам ЦАП. В модуле *E-440* за процедуру подобного рода отвечает *потоковая пересылка* данных в устройство (подробнее см. *§ 3.2.2.11 "Потоковые пересылки"*).

3.2.2.7. Запрос V_COMMAND_IRQ_E440

Запрос типа 'V*endor Request*' с номером V_COMMAND_IRQ_E440 (см. файл `E440.h`) предназначен для инициирования так называемого *'командного' прерывания IRQE* в цифровом сигнальном процессоре, установленном на модуле. Т.е. при поступлении запроса с данным номером микроконтроллер AVR осуществляет аппаратное генерирование прерывания *IRQE* в DSP. С программной точки зрения пересылка указанного запроса выполняется очень просто и выглядит следующим образом:

```
WORD InBuf[4];      // буфер под параметры Setup packet
DWORD cbRet=0;      // байтовый счетчик полученных байтов

InBuf[0] = 0;       // направление передачи данных в фазе Data stage;
                    // в данном запросе не используется, т.к. этой стадии не будет
InBuf[1] = V_COMMAND_IRQ_E440;      // номер запроса
InBuf[2] = 0x0;     // дополнительный параметр (в данном запросе не используется)
InBuf[3] = 0x0;     // дополнительный параметр (в данном запросе не используется)
DeviceIoControl(hDevice,
                DIOC_SEND_COMMAND,
                &InBuf,
                sizeof(InBuf),
                NULL,       // в данном запросе фазы Data stage не будет
                0,          // в данном запросе фазы Data stage не будет
                &cbRet,
                NULL);
```

3.2.2.8. Запрос V_GO_SLEEP_E440

В данном модуле этот запрос не реализован ☹.

3.2.2.9. Запрос V_WAKEUP_E440

В данном модуле этот запрос не реализован ☹.

3.2.2.10. Запрос V_GET_MODULE_NAME_E440

Запрос типа 'V*endor Request*' с номером V_GET_MODULE_NAME_E440 (см. файл `E440.h`) предназначен для получения названия опрашиваемого устройства. Т.е. при поступлении запроса с данным номером микроконтроллер AVR модуля осуществляет в фазе *Data stage* передачу по шине **USB** семибайтового массива данных, содержащего строку `'E440'`. С программной точки зрения пересылка указанного запроса выполняется очень просто и выглядит следующим образом:

```
WORD InBuf[4];          // буфер под параметры Setup packet
DWORD cbRet=0;          // байтовый счетчик полученных байтов

InBuf[0] = 1;           // будет приём данных из модуля в фазе Data stage;
InBuf[1] = V_GET_MODULE_NAME_E440;      // номер запроса
InBuf[2] = 0x0;         // дополнительный параметр (в данном запросе не используется)
InBuf[3] = 0x0;         // дополнительный параметр (в данном запросе не используется)
DeviceIoControl(hDevice,
                DIOC_SEND_COMMAND,
                &InBuf,
                sizeof(InBuf),
                ModuleName,     // массив для получаемых в фазе Data stage данных
                7,              // в фазе Data stage передается 7 байтов данных
                &cbRet,
                NULL);
```

3.2.2.11. Потоковые пересылки

Пересылки такого рода на модуле *E-440* используются исключительно для целей передачи массивов данных из *FIFO буфера АЦП* в хост-компьютер, а также из хост-компьютера в *FIFO буфер ЦАП*.

В общем виде принципы работы здесь достаточно проста. Допустим, пользователь осуществил запуск АЦП. Драйвер сигнального процессора *LBIOS* начинает располагать поступающие отсчеты с АЦП в так называемом *FIFO буфере АЦП*. По мере накопления определенной порции аналоговых данных DSP через посредство соответствующего прерывания (см. *§ 3.3. "Интерфейс AVR с DSP"*), даёт знать микроконтроллеру AVR, что пора бы что-то делать с накопившейся информацией. В этой ситуации AVR проверяет, поступил ли запрос из хост-компьютера на ***потоковую*** передачу аналоговой информации в PC. Если да – AVR осуществляет такую операцию пересылки данных в PC из соответствующей половинки *FIFO буфера АЦП*, если нет – AVR терпеливо ждет поступления требуемого запроса. Пользователь из своего приложения в PC может легко организовать так необходимый ему для нормального сбора данных с АЦП запрос в модуль на ***потоковую*** пересылку (естественно только после того, как он обеспечил запуск АЦП с помощью интерфейсной функции *START_ADC()*). Для этих целей и служит штатная интерфейсная функция *ReadData()* (по сути это своего рода слепок с *Windows API* функции ***RadFile()***). Т.е. данная функция обеспечивает формирование запроса на ***потоковую*** пересылку, передачу его в модуль и приём из модуля требуемого (заказанного) количества данных с аналоговых каналов.

Всё то же самое верно и для случая работы ЦАП. За тем исключением, что за формирование ***потоковой*** пересылки на передачу данных в модуль для *FIFO буфер ЦАП* предназначена штатная интерфейсная функция *WriteData()* (по сути это своего рода слепок с *Windows API* функции ***WriteFile()***).

3.3. *Интерфейс AVR с DSP*

Помимо ответственности за организацию и поддержание в надлежащем виде **USB**-интерфейса модуля *E-440* с хост-компьютером, на микроконтроллер AVR ложится довольно непростая задача по корректному взаимодействию с цифровым сигнальным процессором, расположенным на модуле.

Драйвер микроконтроллера AVR написан таким образом, что разрешает пользователю из приложения в PC путём посылки в модуль соответствующих запросов **USB** (см. *§ 3.2. "Организация USB интерфейса"*) задавать функциональное состояние модуля *E-440*. При этом аппаратно модуль реализован так, что вся периферия (АЦП, ЦАП, ТТЛ линии и .т.д.) находится исключительно под управлением цифрового сигнального процессора. Поэтому от AVR настоятельно требуется обладать возможностью оперативного вмешательства в любой момент времени в текущую работу драйвера DSP без остановки в его функционировании (кроме сброса DSP). С этой целью весь интерфейс микроконтроллера AVR с DSP реализован двояким образом:

1. Набор выделенных цифровых линий, которые в штатном программном обеспечении (как в драйвере AVR, так и в *LBIOS*'е) используются следующим образом:
   - ✓ По поступлении из хост-компьютера запроса *V_RESET_DSP_E440* AVR выполняет полный аппаратный сброс DSP.
   - ✓ По поступлении из хост-компьютера запроса *V_COMMAND_IRQ_E440* AVR инициирует в DSP так называемое *'командное' прерывания IRQE*, которое должно надлежащим образом отрабатываться соответствующим обработчиком прерывания в *LBIOS*'е.
   - ✓ Две цифровые линии заведены с флагов *PF1* и *PF3* DSP (см. *§ 3.4.5. "Конфигурирование флагов DSP"*) на внешние прерывания AVR, а именно входы *INT0* и *ANA_COMP* (см. описание *AVR AT90S8515*). Штатное программное обеспечение DSP использует данные линии с целью извещения AVR о настоятельной необходимости обеспечить передачу требуемых данных либо в хост-компьютер (собранные данные из *FIFO буфера АЦП*), либо из хост-компьютера (новые данные для *FIFO буфера ЦАП*).
2. Возможность доступа микроконтроллера AVR к **любой** ячейке памяти DSP обеспечивается через посредство такой полезной архитектурной особенности сигнального процессора, как наличие в микросхеме DSP канала *IDMA* (чуть подробнее о канале IDMA DSP см. *§ 3.4.1 "Общие сведения"*). Именно эта отличительная черта архитектуры сигнального процессора даёт возможность пользователю из своего приложения при помощи организации посылок запросов типа *V_PUT_ARRAY_E440* и *V_GET_ARRAY_E440* считать или изменить по своему усмотрению содержимое **любой** области памяти DSP. Также именно благодаря этой архитектурной особенности сигнального процессора микроконтроллер AVR в произвольные моменты времени имеет возможность обращаться к содержимому любой области как в *FIFO буфере АЦП*, так и в *FIFO буфере ЦАП*, обеспечивая, таким образом, необходимые *потоковые пересылки* информации в/из хост-компьютера.

3.4. *Интерфейс DSP с периферией модуля*

В данном разделе хотелось бы достаточно подробно осветить различные аспекты взаимодействия установленного на модуле *E-440* цифрового сигнального процессора с периферией, а именно АЦП, ЦАП, цифровые линии, ППЗУ и т.д.

3.4.1. *Общие сведения*

Основными функциональными обязанностями цифрового сигнального процессора *ADSP-2185M*, установленного на модуле *E-440*, являются управление и полный контроль над всей периферией устройства, а также, при необходимости, первичная обработка информации. DSP функционирует достаточно автономно, в том смысле, что даже и не "подозревает" о существовании микроконтроллера AVR, не говоря уже о наличии **USB** шины, все взаимодействие с которой взял на себя AVR (см. *§ 3.2. "Организация USB интерфейса"*). Именно и только AVR, в соответствии с поступающими по **USB** с хост-компьютера запросами (командами), обеспечивает всё необходимое функциональное согласование Вашего приложения с сигнальным процессором модуля, заставляя его выполнять те или иные алгоритмы работы.

Можно сказать, что микроконтроллер AVR оперативно связан с DSP двумя различными способами (методами). Первый из них заключается во взаимном использовании трех отдельных цифровых линий. Так, AVR имеет возможность инициировать в DSP прерывание *IRQE*. В штатном *LBIOS*'е данный цифровой сигнал используется в качестве так называемого "командного прерывания" (см. *ниже*). DSP же, со своей стороны, может генерировать в AVR целых два прерывания *INT0* и *ANA_COMP* (см. описание *AVR AT90S8515*). В штатном *LBIOS*'е данные прерывания используются для управления потоками данных с АЦП и на ЦАП соответственно.

Второй способ взаимодействия AVR с сигнальным процессором основывается на том факте, что DSP обладает такой архитектурной особенностью, как свой собственный контроллер ПДП (так называемый канал IDMA: Internal Direct Memory Access) для доступа к любой ячейке своей внутренней памяти (подробнее о работе IDMA см. книгу *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", § 11.3 "IDMA Port", Analog Devices, Inc., Third Edition September 1995*, которую можно без проблем скачать с сайта фирмы ***Analog Devices, Inc.:*** *www.analog.com/library/dspManuals/ADSP-2100_fum_books.html*). Микроконтроллер AVR способен по запросам (командам) из хост-компьютера осуществлять те или иные операции прямого доступа к памяти DSP. Таким образом, благодаря указанной возможности Ваше приложение в PC через посредство AVR может обращаться к любой ячейке памяти сигнального процессора, не прерывая при этом работу самого DSP. Это исключительно удобно при построении алгоритмов, работающих в реальном масштабе времени. Кроме возможности собственно прямого доступа к памяти, протокол работы с каналом IDMA DSP предусматривает также такую важную процедуру как начальная загрузка сигнального процессора управляющей программой (драйвером), которая и будет осуществлять требуемые алгоритмы обработки и ввода-вывода информации (подробности загрузки DSP 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*). Т.е. микроконтроллер AVR (по командам из хост-компьютера) способен осуществлять загрузку всей памяти сигнального процессора требуемым программным кодом.

Фирменный драйвер *LBIOS* работает по принципу команд, и для реализации такой возможности используется прерывание *IRQE* сигнального процессора ("командное прерывание"). Сначала в соответствующую ячейку памяти программ DSP (предопределенная константа *L_COMMAND_E440*, см. *§ 2.5.1.4 "Переменные LBIOS"*) заносится номер команды (см. *§ 2.5.1.5 "Номера команд LBIOS"*), которую драйвер *LBIOS* впоследствии должен выполнить. Затем в DSP инициируется командное прерывание *IRQE*, в ответ на которое обработчик данного прерывания, содержащийся в самом *LBIOS*, выполняет соответствующие данной команде действия. Названия штатных команд для DSP говорят сами, для чего каждое из них предназначено, а более полное понимание их функционирования можно при желании проследить по исходным текстам *LBIOS* (см. директорию \DSP).

Цифровой сигнальный процессор получает данных с АЦП, контролирует цепи коммутатора входных сигналов, коэффициенты усиления программируемого усилителя, частоту запуска АЦП и, при необходимости, синхронизирует ввод данных с АЦП либо по сигналу с цифровой линии **TRIG** внешнего разъема *DRB-37M*, либо по выбранному аналоговому каналу. Также DSP обеспечивает управление микросхемой двухканального ЦАП через посредство своего последовательного порта (*SPORT0*). Функциональное состояние внешних цифровых линий на разъеме *DRB-37F* определяется DSP с помощью простых операций чтения/записи нулевой ячейки своего пространства ввода/вывода (I/O Memory Space), а также с помощью флага *PF0* управляет доступностью выходных линий. И наконец, сигнальный процессор обеспечивает взаимодействие с микросхемой ППЗУ, предоставляя полный доступ к любой его ячейке.

В штатном пакете программного обеспечения, прилагаемого к модулю *E-440*, поставляются все исходные тексты драйвера *LBIOS*, написанные на языке ассемблер цифрового сигнального процессора. Их можно найти в директории `\DSP`. Исходники достаточно подробно прокомментированы и могут быть с успехом использованы в качестве законченного примера программирования данного модуля при модификации штатного или создании собственного драйвера устройства.

3.4.2. Создание управляющей программы

У пользователя, как правило, не появляется необходимость в написании своих собственных управляющих программ для данного модуля, т.к. все наиболее часто требуемые алгоритмы работы уже реализованы в фирменном драйвере, находящимся в файле `E440.bio`. Если у Вас все-таки возникла необходимость в создании собственной управляющей программы (например, для реализации какого-либо специализированного алгоритма действия DSP), то для этого придется освоить достаточно несложный язык ассемблера для сигнального процессора. В качестве готового примера программирования модуля на таком языке можно использовать исходные тексты фирменного драйвера, хранящиеся в файлах `DSP\E440.DSP` и `DSP\*.H`.

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

1. Вам необходимо будет изучить достаточно несложную архитектуру процессоров семейства ADSP-218x, а также освоить довольно простой язык программирования (ассемблер 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* (которую можно без проблем скачать с сайта фирмы ***Analog Devices, Inc.:*** *www.analog.com/library/dspManuals/ADSP-2100_fum_books.html*) или в русском переводе *"Руководство пользователя по сигнальным микропроцессорам семейства ADSP-2100", под редакцией А.Д.Викторова, Санкт-Петербург, 1997* (нечто похожее на эту книжку можно найти по следующей ссылке: *vadis7.chat.ru/book.htm*). 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-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 вышеуказанные средства отладки (кроме `bld21.exe`), можно приобрести в ***ЗАО "Л-Кард"***. В качестве архитектурного файла нужно использовать `E440.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 с обычной исполняемой DOS/Windows 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\BIN3PCI.EXE` в файл `*.BIO` (подробности см. *Приложение С*).

Вообще-то, всю эту последовательность создания файла `*.BIO` можно проследить по содержанию файлу `DSP\E440.BAT`. Созданный таким образом файл с управляющей программой `*.BIO` затем используется, например, в штатной функции загрузки сигнального процессора ***LOAD_LBIOS()*** (см. *§ 2.5.2.7. "Загрузка LBIOS"*).

При желании, для создания собственного драйвера сигнального процессора для модуля *E-440* можно воспользоваться такой продвинутой интегрированной средой разработки как *VisualDSP++* (на сегодняшний день доступна версия 3.0). Подробности смотри на сайте *www.analog.com*.

3.4.3. Загрузка управляющей программы в DSP

Перед началом работы с устройством *E-440* необходимо его "оживить", т.е. загрузить в DSP модуля либо фирменный драйвер, находящийся в файле `E440.bio`, либо Вашу собственную управляющую программу. **Только** после выполнения такой процедуры модуль будет корректно работать с функциями штатной или Вашей (если создадите) библиотеки. Формат файла `E440.bio`, содержащего все коды инструкций управляющей программы, а также переменных *LBIOS*, подробно описан в *Приложении С*. Эти коды инструкций драйвера *LBIOS* и начальные значения его переменных необходимо расположить по соответствующим адресам памяти программ сигнального процессора и запустить управляющую программу на выполнение (в штатном драйвере память данных DSP изначально пуста и, следовательно, в неё ничего загружать не надо). Все это и называется загрузка управляющей программы в DSP или просто загрузка DSP.

Сама процедура начальной загрузки управляющей программы во внутреннюю память DSP модуля 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 данном модуле, например, с помощью интерфейсной функции ***DSP_RESET()*** (см.*§ 2.5.2.10. "Сброс DSP на модуле"*);
- заполните требуемой информацией память программ и/или данных DSP, кроме ячейки памяти программ с адресом 0x0000 (или короче PM(0x0000)), для чего можно, например, воспользоваться соответствующими интерфейсными функциями ***PUT_DM_ARRAY()*** (см. *§ 2.5.3.7. "Запись массива слов в память данных DSP"*) или ***PUT_PM_ARRAY()*** (см. *§ 2.5.3.8. "Запись массива слов в память программ DSP"*).
- 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, **ВСЁ!** Теперь можно спокойно приступать к управлению модулем с помощью интерфейсных функций штатной или Вашей библиотеки.

3.4.4. Порты управления

В качестве интерфейса цифрового сигнального процессора с некоторой частью периферии модуля (а конкретнее АЦП и цифровые линии) используются так называемые порты управления, которые отображены на пространство ввода-вывода DSP (I/O Memory Space). Подробное 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.*

На модуле аппаратно организованы четыре порта управления (см. *Таблицу 15*). При взаимодействии с ними дешифрируется только младший бит адреса при обращении к ячейке пространства ввода-вывода DSP (следовательно, адреса портов можно устанавливать кратным двум). Цифровой сигнальный процессор рассматривает все свое пространство ввода-вывода (всего 2048 ячеек), как 4 области по 512 ячеек и обладает возможностью устанавливать для каждого такого сегмента индивидуальное количество тактов ожидания (*waitstates*) для обращения к нему. За конфигурацию тактов ожидания для I/O Memory Space отвечает служебный регистр сигнального процессора Waitstate Control Register, расположенный по адресу DM(0x3FFE). Таким образом, часть портов управления можно расположить в одной области пространства ввода-вывода, а часть – в другой и т.д. Это позволяет, установив задержку на одну область, не замедлить доступ к портам в другой. В штатном *LBIOS*'е четыре порта управления расположены в первой области пространства ввода-вывода и для любой операции обращения к ним устанавливается один дополнительный такт ожидания. В штатном *LBIOS*'е это выглядит так:

```
{ Константа для обозначения адреса регистра Waitstate Control Register      }
     const Dm_Wait_Reg = 0x3FFE;

     . . . . .

{ Set 1 Waitstate for I/O Memory Space                                        }
    AR = 0x0001;              { 0000 0000 0000 0001                           }
    DM(Dm_Wait_Reg) = AR;     { 0x3FFE – Waitstate Control Register           }
```

Краткое описание используемых портов управления приведено в нижеследующей таблице.

Таблица 15. Порты управления DSP

Название порта	Доступ	Назначение	Адрес
TTL_IN	Чтение	Чтение состояния 16[ти] входных цифровых линий на внешнем разъёме *DRB-37F*.	xxx0H
TTL_OUT	Запись	Установка 16[ти] выходных цифровых линий на внешнем разъёме *DRB-37F*.	xxx0H
READ_ADC	Чтение	Чтение данных из АЦП	xxx1H
SET_ADC_CHANNEL	Запись	Управление входным коммутатором и коэффициентом усиления канала АЦП	xxx1H

3.4.5. *Конфигурирование флагов DSP*

Микросхемы цифровых сигнальных процессоров типа *ADSP-2185M* имеют в своём составе восемь дополнительных неспециализированных выводов, так называемых флагов, обозначаемых обычно **PF0÷PF7**. Данные флаги, при желании, могут быть индивидуально запрограммированы как на вход, так и на выход. На микросхеме указанного типа сигнального процессора выводы этих флагов обладают совмещенными функциями. Так, флаг **PF6** совмещен с входом прерывания *IRQL1*, а флаг **PF7 —** с *IRQ2* и т.д. (все необходимые подробности можно найти в техническом описании (*datasheet*) данного DSP на сайте *www.analog.com*). Все флаги и краткое описание их назначения для модуля *E-440* приведены в следующей таблице:

Таблица 16. Флаги сигнального процессора.

Название флага	*Направ-ление*	*Назначение*
PF0	Выход	При **PF0**=1 переводятся в третье (высокоимпедансное) состояние все 16^{ть} выходных цифровых линий на разъёме *DRB-37F* модуля.
PF1	Выход	Прерывание в **AVR**. В штатном *LBIOS*'е используется для инициализации передачи данных, полученных с АЦП, в хост-компьютер. Т.е., когда DSP собрал соответствующую половину FIFO буфера АЦП, следует 'дернуть' этим флажком вверх-вниз. При этом сгенерится прерывание в **AVR**, обработчик которого начнет передачу этих готовых данных в хост-компьютер. Нормальное состояние – 0.
PF2	-------	Не используется.
PF3	Выход	Прерывание в **AVR**. В штатном *LBIOS*'е используется для инициализации передачи данных, полученных с хост-компьютера, в соответствующую половинку FIFO буфера ЦАП. Т.е., когда DSP отослал очередную половинку FIFO буфера, следует 'дернуть' этим флажком вверх-вниз. При этом сгенерится прерывание в **AVR**, обработчик которого начнет передачу новых данных из хост-компьютера в память DSP. Нормальное состояние – 0.
PF4/IRQE	Вход	*Командное прерывание* для DSP.
PF5	Выход	Прямая запись в выходной регистр номера/усиления канала.
PF6	-------	Не используется.
PF7/IRQ2	Вход	Прерывание от АЦП (конфигурируется для работы по фронту).
FL0	Выход	**FL0**=1 выбор микросхемы пользовательского ППЗУ (*Chip Select*).
FL1	Выход	Клоки пользовательского ППЗУ.
FL2	Выход	Данные в пользовательское ППЗУ.
FI	Вход	Данные из пользовательского ППЗУ.
FO	Выход	Разрешение загрузки данных в ЦАП. Активный уровень — низкий

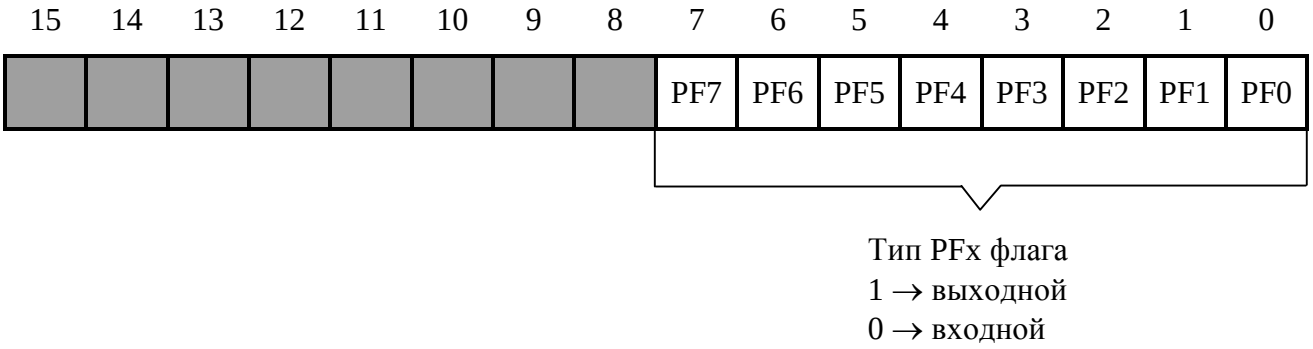
Таким образом, флаги **PFx** необходимо запрограммировать на вход-выход надлежащим образом, т.е. флаги **PF2**, **PF4**, **PF6** и **PF7** как входные, а флаги **PF0**, **PF1**, **PF3** и **PF5** как выходные.

Программирование флагов осуществляется с помощью двух управляющих регистров DSP:

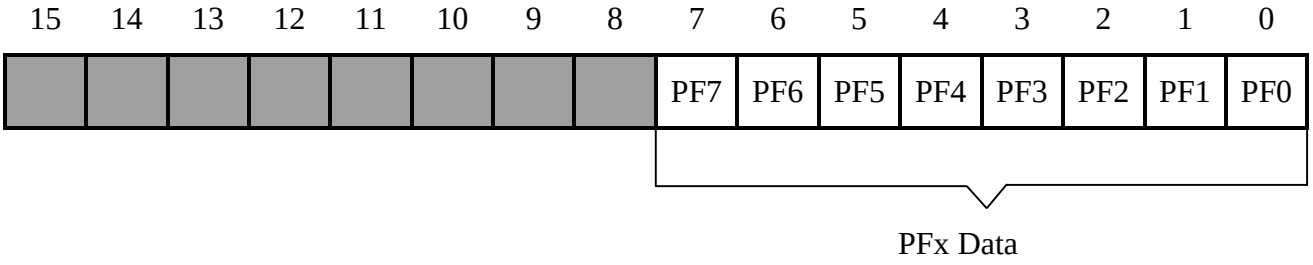
- ✓ регистр Programmable Flag & Composite Select Control, находящийся по адресу DM(0x3FE6) в памяти данных DSP, управляет направлением флага
  - 1 → выход,
  - 0 → вход;
- ✓ регистр Programmable Flag Data, расположенный по адресу DM(0x3FE5), используется для записи и считывания значений флагов.

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))



Например, в штатном *LBIOS*'е для надлежащего программирования флагов **FO**, **FLx** и **PFx** написан следующий код:

```
{ Введем константы для обозначения адресов регистров DSP,          }
{ отвечающих за управление флагами PF0÷PF7                         }
     const Prog_Flag_Data = 0x3FE5;
     const Prog_Flag_Comp_Sel_Ctrl = 0x3FE6;

     . . . . .

{ установим флаг FO в низкое состояние - разрешение загрузки ЦАП   }
     RESET FLAG_OUT;

{ установим флаги FLx в исходное низкое состояние                  }
     RESET FL0, RESET FL1, RESET FL2;

{ Флаг PF0 в 1 (переводим все линии TTL OUT на внешнем цифровом    }
{ разъёме в третье (высокоимпедансное) состояние)                  }
{ Флаг PF5 в 0 (запись буферный регистр адреса/усиления канала)    }
     AR=0x01;
     DM(Prog_Flag_Data)=AR;
```

```
{ Отконфигурируем все флаги PFx:                                    }
{                    PF0,PF1,PF3 и PF5 - выходные,                  }
{                    PF2,PF4,PF6 и PF7 - входные                    }
     AR=0x2B;                              { 0010 1011               }
     DM(Prog_Flag_Comp_Sel_Ctrl)=AR;
```

3.4.6. АЦП, коммутатор и программируемый усилитель

На модуле *E-440* установлен 14[ти] битный АЦП *LTC1416* с параллельным выходным портом производства фирмы *Linear Technology Corporation*. Все взаимодействие цифрового сигнального процессора с этим АЦП, а также коммутатором и программируемым усилителем осуществляется через посредство портов *READ_ADC* и *SET_ADC_CHANNEL*, прерывания *IRQ2* (предварительно данное прерывание должно быть **обязательно** сконфигурировано на работу по фронту) и тактовых синхроимпульсов (клоков) *SCLK1* последовательного порта *SPORT1*. 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*. Конфигурирование внешних прерываний DSP подробно описано в той же книге в *§ 3.4.2 "Configuring Interrupt", стр. 3-14* и *§ 9.5 "EXTERNAL INTERRUPTS", стр. 9-14*.

Как упоминалось выше, для обслуживания взаимодействия с АЦП предлагается использовать внешнее прерывание DSP *IRQ2*. Запрос на это прерывание устанавливается по спадающему фронту сигнала –BUSY, а в качестве задатчика сигнала запуска преобразования АЦП (–CONV) используются тактовые клоки *SCLK1* последовательного порта *SPORT1*. Как правило, готовые данные с АЦП считываются в обработчике *IRQ2* путем выполнения операции чтения из порта *READ_ADC*, т.е. первой ячейки пространства ввода-вывода `IO(0x1)`. При заходе в данный обработчик прерывания первым делом **необходимо** записать в так называемый буферный регистр адреса/усиления очередное значение логического канала, потом считывать готовые данные с АЦП, а далее уже может следовать Ваш алгоритм обработки. Также очень важно, чтобы от момента прихода прерывания до чтения данных из порта *READ_ADC* проходило не более 1.4 мкс. Иначе могут быть получены уже не (N-1)[ые], а N[ые] данные с АЦП (подробнее см. диаграммы сигналов для работы с АЦП на *рисунке ниже*).

Управление адресом (номером) канала и коэффициентом усиления сигнала происходит с помощью двух управляющих регистров:

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

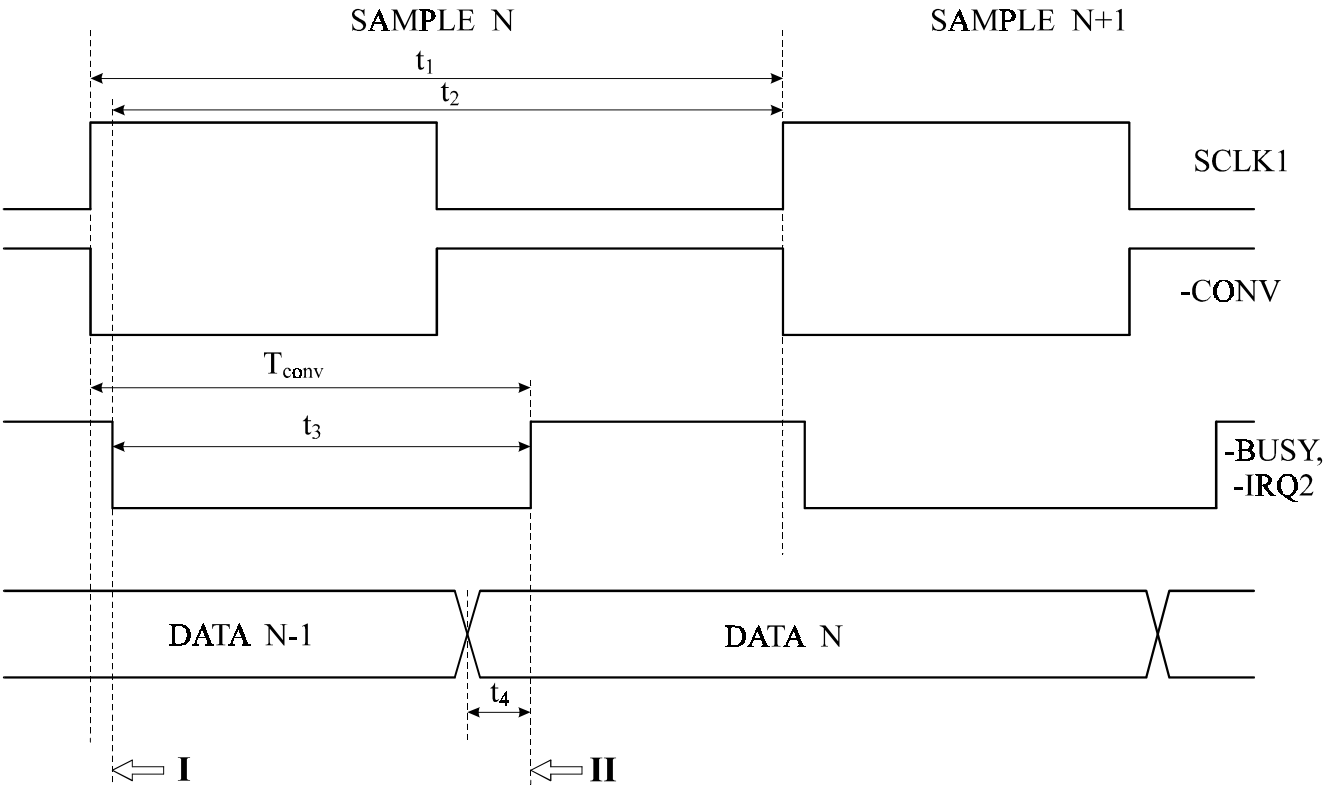
Информация, находящаяся в *выходном* регистре, определяет адрес канала и коэффициент усиления для **текущего** преобразования АЦП. Формат этой информации полностью соответствует формату логического канала, описанному в *§ 2.3.2.3 "Логический номер канала АЦП"* Данные из *буферного* регистра переписываются в *выходной* регистр по спадающему фронту сигнала –BUSY. Таким образом, данные адреса/усиления, записанные в *буферный* регистр при обработке прерывания от выборки N (см. рисунок ниже), будут действовать при преобразовании АЦП для выборки N+2. Запись информации в *буферный* и *выходной* регистры производится через операцию записи в порт *SET_ADC_CHANNEL* (т.е. в первую ячейку пространства ввода-вывода `IO(0x1)`), при надлежащем использовании состояния флага *PF5*. Порядок записи в *выходной* регистр аппаратно определен следующим образом:

1. Сбросить флаг **PF5** в ноль, т.е. **PF5**=0.
2. Произвести операцию записи в порт *SET_ADC_CHANNEL* требуемого логического номера канала АЦП. Это фактически означает запись в *буферный* регистр адреса/усиления.
3. Установить **PF5** в единицу (**PF5**=1). При этом. информация из *буферного* регистра будет скопирована в *выходной*.
4. Сбросить флаг **PF5** в ноль, т.е. **PF5**=0.
5. Произвести операцию записи в порт *SET_ADC_CHANNEL* следующего логического номера канала АЦП. Это опять же означает запись в *буферный* регистр адреса/усиления.

Таким образом, мы прописали необходимой информацией сразу оба буфера и *буферный* и *выходной*. При выполнении этой процедуры следует помнить о двух важных моментах, а именно:

- ✓ Запись в *выходной* регистр должен производиться за время не менее 2 мкс до запуска преобразования АЦП (необходимо время на установления аналогового тракта).
- ✓ Запрещена запись в *выходной* регистр во время активного сигнала, т.е. от момента **I** до момента **II** (см. ниже *рисунок с диаграммами*).

Временные диаграммы основных сигналов, используемых для работы с АЦП, приведены на следующем рисунке:



На приведенном рисунке через **I** обозначен момент времени:

- ✓ прихода N$^{\text{ого}}$ прерывания *IRQ2*, в обработчике которого необходимо считать готовые (N-1)$^{\text{ые}}$ данные с АЦП и записать в *буферный* регистр (N+2)$^{\text{ой}}$ логический канал;
- ✓ аппаратного копирования содержимого *буферного* регистра с (N+1)$^{\text{ым}}$ логическим каналом в *выходной* регистр;

В момент времени **II** аппаратно защелкиваются N$^{\text{ые}}$ данные с АЦП в порту *READ_ADC*.

Основные временные характеристики, показанные на рисунке с диаграммами, приведены в таблице ниже:

Таблица 17. Временные характеристики сигналов АЦП

Обозначение	Длительность			Единицы
	min	typ	max	
T_{conv}	1.5	1.9	2.2	мкс
t_1	2.5			мкс
t_2		2.475		мкс

t_3	1.475	1.875	2.175	мкс
t_4	75	100		нс

Частота запуска преобразования АЦП на данном модуле фактически определяется частотой следования внутренних тактовых синхроимпульсов *SCLK1* последовательного порта *SPORT1*. Для надлежащего конфигурирования порта *SPORT1* используются соответствующие регистры сигнального процессора: SPORT1 Control Register и SPORT1 SCLKDIV, адреса которых в памяти данных определены как DM(0x3FF2) и DM(0x3FF1) соответственно. С помощью данных регистров Вы можете разрешить либо запретить генерацию синхроимпульсов, а также напрямую задавать период их следования, однозначно определяя частоту работы АЦП. Исходя из архитектуры сигнального процессора, частота запуска преобразования АЦП определяется по следующей формуле:

$$\mathbf{AdcRate} = \mathbf{F}_{clockout}/(2\cdot(\mathbf{SCLKDIV1}+1)),$$

где $\mathbf{F}_{clockout}$ – тактовая частота установленного на модуле DSP, равная 48000 кГц, **SCLKDIV1** – коэффициент деления частоты $\mathbf{F}_{clockout}$, который определяется содержимым регистра SPORT1 SCLKDIV. Очень подробное описание работы сериальных портов DSP можно найти в книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 5 "Serial Ports", cmp. 5-1, Analog Devices, Inc., Third Edition, September 1995.*

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;                  { задаем частоту 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 2.0 мкс, чтобы оцифрился последний предыдущий отсчет    }
    cntr=95;
    DO DelayLoop UNTIL CE;
DelayLoop: NOP;
{ очистим все запросы на прерывания                                    }
    IFC = 0xFF; NOP;
{ разрешим прерывания IRQ2                                             }
    IMASK=0x200; NOP;
```

```
{ зададим усиление и номер канала для текущего и следующего отсчетов,}
{ предполагая наличия циклического буфера с управляющей таблицей     }
{ в памяти данных с указателем (I5, M5, L5)                           }

{ PF5 в ноль - промежуточный (буферный) регистр                       }
    AR=DM(Prog_Flag_Data);
    AR=CLRBIT 5 OF AR;
    DM(Prog_Flag_Data)=AR;

{ запись первого канала в буферный регистр                            }
    AR=PM(I5, M5);
    IO(SET_ADC_CHANNEL)=AR;

{ PF5 в единицу - копирование из буферного в выходной регистр         }
    AR=DM(Prog_Flag_Data);
    AR=SETBIT 5 OF AR;
    DM(Prog_Flag_Data)=AR;

{ PF5 в ноль - опять промежуточный (буферный) регистр                 }
AR=DM(Prog_Flag_Data);
AR=CLRBIT 5 OF AR;
DM(Prog_Flag_Data)=AR;

{ запись второго канала в буферный регистр                            }
AR=PM(I5, M5);
IO(SET_ADC_CHANNEL)=AR;

{ задержка на 2.0 мкс, чтобы установился аналоговый тракт             }
    cntr=95;
    DO DelayLoop1 UNTIL CE;
DelayLoop1: NOP;

{ включим клоки SCLK1, 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 прерываний IRQ2                     }
{ Первое пришедшее прерывание IRQ2 будет содержать 'левый' отсчет с   }
{ АЦП и его не надо принимать в расчёт, а вот при последующих         }
{ прерываниях мы будем получать уже то, что надо ☺                    }
{ Частота работы АЦП: AdcRate=48000.0/(2*(99+1))=240.0 кГц            }

    . . . . .

{ Обработчик прерывания IRQ2 может содержать следующие строчки        }
    AR=DM(I5, M5);              { зададим усиление и номер канала для }
    IO(SET_ADC_CHANNEL)=AR;     { следующего отсчета                  }
    . . . . .
    AR=IO(READ_ADC);            { теперь в AR находится отсчет с АЦП  }
```

3.4.7. Организация сбора данных с АЦП в LBIOS

В штатном *LBIOS*'е для работы с АЦП программно организован циклический двойной FIFO буфер АЦП достаточно большого размера (до 12288 слов), расположенный в памяти данных DSP. При этом всё базисное взаимодействие с АЦП (чтение готовых данных, их корректировка, размещение в буфере, установление очередного логического канала и т.д.) сосредоточено в обработчике прерывания *IRQ2*. Основной же программе остается только периодически (в фоновом режиме) отслеживать степень заполнения текущей половинки FIFO буфера готовыми данными с АЦП. Как только это событие произошло (т.е. текущая половинка буфера полностью заполнилась) DSP сиг-

нализирует об этом факте в микроконтроллер AVR через посредство флага *PF1*. Это является признаком того, что данная половинка FIFO буфера полностью готова к передаче в хост-компьютер по шине **USB**. AVR, используя свою возможность работы с DSP по *каналу IDMA*, должен осуществить данную процедуру передачи, в то время как DSP продолжает непрерывный сбор данных с АЦП в другую половинку FIFO буфера. Для целей приёма данных посылаемых модулем в хост-компьютере можно, например, воспользоваться штатной интерфейсной функцией *ReadData()*.

3.4.8. Корректировка данных с АЦП

Помимо собственно сбора данных *LBIOS* может осуществлять (по желанию пользователя) первичную обработку получаемой информации, заключающуюся в корректировке отсчетов с АЦП. Управление корректировкой в штатном *LBIOS*'е осуществляется через посредство предопределенной переменной *L_CORRECTION_ENABLED_E440*. В качестве корректировочных коэффициентов можно использовать либо Ваши собственные, либо штатные, которые хранятся в ППЗУ модуля (см. *§ 2.3.3. "Формат пользовательского ППЗУ"*). Штатные коэффициенты прошиваются в ППЗУ при наладке модуля в ***ЗАО "Л-Кард"***. Вся процедура первичной обработки располагается в обработчике прерывания *IRQ2* и обеспечивает корректировку смещения и масштаба получаемых с АЦП данных. Обычная формула для корректировки выглядит следующим образом:

$$\mathbf{U = (V + A)\cdot B},$$

где **V** – полученный отсчет с АЦП, **A** – кор. коэффициент смещения, **B** – кор. коэффициент масштаба, **U** – откорректированное значение отсчета с АЦП. В нашем случае коэффициент **A** это обычное *знаковое* целое $16^{\text{ти}}$ битное число. Коэффициент **B** это *беззнаковое* дробное число по величине близкое к 1.0 (например, 0.956 или 1.17). Чтобы произвести процедуру умножения в DSP, коэффициент **B** приходится немного 'причесать' к виду, удобному для работы сигнального процессора. Для этого необходимо привести его к известному в DSP дробному формату 1.15 (подробнее о форматах см. книгу *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)"*, *Appendix C "Numeric Format", стр. C-1, Analog Devices, Inc., Third Edition, September 1995*.), используя следующую формулу:

$$\mathbf{B'} = 32768\cdot\mathbf{B} + 0.5,$$

где **B'** – $16^{\text{ти}}$ битное представление коэффициента **B** в дробном формате 1.15. Именно коэффициенты **A** и **B'** хранятся в ППЗУ модуля и применяются в процедуре корректировки данных с АЦП. Необходимо также учитывать, что для каждого *коэффициента усиления* в ППЗУ модуля хранится соответствующая пара коэффициентов **A** и **B'** (см. *§ 2.3.3. "Формат пользовательского ППЗУ"*).

На языке DSP ассемблер операция корректировки выглядит примерно так:

```
{ чтение отсчета с АЦП                                              }
    AR=IO(READ_ADC);
{ корректировка смещения; в регистре AY1 коэффициент A              }
    AY1 = -4; AR=AR+AY1;
{ корректировка масштаба; в регистре MY1 коэффициент B'             }
    MY1 = 32832; MR=AR*MY1(SU);
{ округление результата                                             }
    MR=MR(RND);
{ теперь в регистре MR1 находится откорректированный отсчет с АЦП  }
```

3.4.9. ЦАП

Взаимодействие с микросхемой двухканального $12^{\text{ти}}$ битного ЦАП *AD7249* (техническое описание (*datasheet*) можно найти на сайте *www.analog.com*), которая по Вашему желанию может быть установлена на модуле *E-440*, цифровой сигнальный процессор осуществляет с помощью своего последовательного порта *SPORT0*. Для этого *LBIOS* программирует *SPORT0* надлежащим образом, а именно:

- ✓ длина слова (*word length*) – 16 бит;
- ✓ тактовые синхроимпульсы (*serial clocks*) – внутренние с периодом не менее 0.2 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 порта *SPORT0* сигнального процессора для кадровой синхронизации приема и передачи (выводы *RFS0* и *TFS0* соответственно) аппаратно объединены. Таким образом, получается, что внутренне генерируемые сигналы кадровой синхронизации приема являются сигналами внешней кадровой синхронизации для передачи. Частоту же генерирования сигналов на выводе *RFS0* можно программным способом 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*.

В штатном *LBIOS*'е период тактовых синхроимпульсов *SCLK0* установлен равным 250 нс с использованием для этого регистр делителя тактовой частоты SPORT0 SCLKDIV, который расположен в памяти данных по адресу DM(0x3FF5). Именно этот период определяет скорость последовательной передачи $16^{ти}$ битного слова данных в ЦАП. Собственно частоту передачи этих слов данных в ЦАП (т.е. частоту работы ЦАП) задаёт другой регистр – SPORT0 RFSDIV, расположенный в памяти данных по адресу DM(0x3FF4). Итого для определения частоты вывода информации на ЦАП можно написать следующую формулу:

$$\mathbf{DacRate} = \mathbf{F}_{clockout}/(2\cdot(\mathbf{SCLKDIV0}+1)\cdot(\mathbf{RFSDIV0}+1)),$$

где $\mathbf{F}_{clockout}$ – тактовая частота установленного на модуле DSP равная 48000 кГц, **SCLKDIV0** – коэффициент деления частоты $\mathbf{F}_{clockout}$, который определяется содержимым регистра SPORT0 SCLKDIV (штатно равен 5), **RFSDIV0** –задает периодичность следования внутренне генерируемых сигнальным процессором сигналов *RFS0*, и следовательно *TFS0* (определяется содержимым регистра SPORT0 RFSDIV).

Формат $16^{ти}$ битного слова данных, передаваемого по последовательному порту *SPORT0* в микросхему ЦАП, приведен в следующей таблице:

Номер бита	Назначение
0÷11	$12^{ти}$ битный код ЦАП
12	Выбор номера канала ЦАП: ✓ '0' – первый канал; ✓ '1' – второй канал.
13÷15	Не используются

Теперь, научившись корректно организовывать передачу требуемой информации в ЦАП, необходимо сделать еще один шаг и заставить его обновлять напряжение на своих выходах в нужный момент времени в соответствии с уже поступившими данными. За это отвечает флаг *FO* (линия *LDAC* микросхемы ЦАП). Т.е. последовательно переданная информация только защелкивается в соответствующих регистрах микросхемы ЦАП, а реальное обновление напряжения на его выходах наступает только при определенных состояниях флага *FO*. При этом возможны два варианта:

1. Если в момент прихода сигнала *TFS0* (линия *SYNC* микросхемы ЦАП) уровень флага *FO* низкий, то происходит *автоматическое* обновление выходных напряжений ЦАП сразу по получении последнего бита данных.
2. Если в момент прихода сигнала *TFS0* (линия *SYNC* микросхемы ЦАП) уровень флага *FO* высокий, то обновление выходных напряжений ЦАП происходит при переводе флага *FO* в низкое состояние в любой момент времени после окончания передачи данных.

В штатном *LBIOS*'е флаг *FO* раз и навсегда устанавливается в *низкое состояние*, т.е. реализуется первый вариант *автоматической* загрузки в ЦАП получаемой информации.

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=5;                     { SCLK0 – internal, T=250 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                   }
{ 48000/(2*(5+1)*(49+1))=80 Кгц                                       }
{ 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;
```

3.4.10. Организация работы с ЦАП в LBIOS

В штатном *LBIOS*'е работа порта *SPORT0* при *непрерывном* выводе на ЦАП организована с использованием так называемого режима *autobuffering* (подробнее о данном режиме можно прочитать в книге *"ADSP-2100 Family User's Manual (Includes ADSP-2171, ADSP-2181)", Chapter 5 "Serial Ports", § 5.11 "AutoBuffering", стр. 5-26, Analog Devices, Inc., Third Edition, September 1995*.). Для этого в памяти данных сигнального процессора программно организован циклический двойной FIFO буфер ЦАП достаточно большого размера (до 4032 слов). Параметры этого буфера и разрешение использовать режим *autobuffering* передаются последовательному порту при его инициализации. При этом сразу после запуска ЦАП (фактически при разрешении последовательного порта) *SPORT0* будет сам, в фоновом для основной программы режиме и без прерывания ее работы, передавать всю требуемую информацию из FIFO буфера на ЦАП с предварительно заданной частотой. Тогда задача основной программы резко упрощается и будет заключаться только в периодическом отслеживании количества переданных в ЦАП слов. По факту завершения вывода данных из очередной половинки FIFO буфера ЦАП DSP сигнализирует об этом в микроконтроллер AVR через посредство флага *PF3*. Это является признаком того, что вся информация на ЦАП из текущей половинки буфера уже успешно выведена и её можно перезаписывать новыми данными. Из хост–компьютера по шине **USB** должны поступать требуемые порции новых данных для последующего их вывода на ЦАП (для этого, например, можно воспользоваться штатной интерфейсной функцией *WriteData()*), и именно AVR (отвечающий за **USB** интерфейс) по мере необходимости осуществляет подгрузку вновь поступивших данных в надлежащую половинку FIFO буфера ЦАП (используя при этом свою возможность работы с DSP по *каналу IDMA*). При этом в процессе этой подгрузки собственно сам вывод на ЦАП продолжается из другой половинки FIFO буфера.

3.4.11. Управление ТТЛ линиями

Управление цифровыми линиями *внешнего разъёма DRB-37F* на модуле *E-440* осуществляется достаточно просто. Флаг *PF0* позволяет контролировать нахождение всех $16^{ти}$ **выходных** линий разъёма в третьем (высокоимпедансном) или рабочем состоянии. Любая операция записи какого-либо числа в порт *TTL_OUT*, т.е. по адресу 0x0 в пространстве ввода-вывода (I/O Memory Space) сигнального процессора, приводит к установке всех $16^{ти}$ выходных линий на разъёме в состояние, соответствующее битам записываемого значения. Аналогично любая операция чтения из порта *TTL_IN*, т.е. по адресу 0x0 из пространства ввода-вывода (I/O Memory Space) сигнального процессора, позволяет получить состояние всех $16^{ти}$ входных линий на разъеме *DRB-37F*. Подробности расположения выводов разъема и краткое описание их назначения см. *§ 1.4.5. "Описание разъема для подключения цифровых сигналов"*. Подробное 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*.

Теперь можно рассмотреть простенький пример работы с цифровыми линиями:

```
{ считаем входные цифровые линии                                    }
    AR=IO(TTL_IN);
{ установим полученные состояния на выходных линиях                 }
    IO(TTL_OUT)=AR;
```

3.4.12. ППЗУ

Как уже отмечалось во *введении к данной главе*, на модуле *E-440* устанавливается микросхема электрически стираемого программируемого ПЗУ (Serial EEPROM) типа ***93C46***. Организация памяти данного устройства представлена форматом: 64 Слова×16 бит. Очень подробные тех-

нические описания (*datasheet*) данного устройства можно найти в Интернете, например, по нижеприведенным ссылкам:

- *www.microchip.com/1000/pline/memory/memdvice/micro/devices/93c46b/index.htm*;
- *www.atmel.com/atmel/products/prod162.htm*.

На модуле *E-440* все программное взаимодействие цифрового сигнального процессора и микросхемы ППЗУ ***93C46*** организовано по следующим линиям:

1. Флаг *FL0* DSP позволяет осуществлять выбор микросхемы ППЗУ. Данный флаг заведен на ножку *Chip Select* (*CS*) микросхемы ***93C46***.
2. Флаг *FL1* DSP используется в качестве тактовых синхроимпульсов (клоков, clocks) при обмене информацией между сигнальным процессором и ППЗУ. Данный флаг заведен на ножку *Serial Data Clock* (*CLK*) микросхемы ***93C46***.
3. Флаг *FL2* используется для последовательной передачи информации (команд, адресов ячеек и собственно данных) из сигнального процессора в ППЗУ. Данный флаг заведен на ножку *Serial Data In* (*DI*) микросхемы ***93C46***.
4. Флаг *FI* используется для последовательного приёма данных из ППЗУ. Данный флаг заведен на ножку *Serial Data Out* (*DO*) микросхемы ***93C46***.

Всю дополнительную информацию, касающуюся работы с микросхемой ППЗУ типа ***93C46***, т.е. временны**ы**е диаграммы, набор и формат команд управления, инструкции по работе с устройством и многое, многое другое, можно с лихвой обнаружить в упомянутых выше технических описаниях. А практическое воплощение взаимодействия DSP и микросхемы ППЗУ ***93C46*** при необходимости нетрудно найти в исходных текстах драйвера *LBIOS* (см. файл `\DSP\flash.h`)

3.4.13. Внешняя синхронизация сигнального процессора

Для задач, которые требуют дополнительной синхронизации работы DSP с внешними устройствами, предусмотрено использование следующих ТТЛ совместимых выводов на разъёмах модуля:

- ✓ вывод **TRIG** на внешнем аналоговом разъеме *DRB-37M*, который подключен к линии прерывания *IRQ1* цифрового сигнального процессора. В фирменном драйвере *LBIOS* это прерывание отконфигурировано для работы по фронту и используется для различных режимов цифровой синхронизации ввода данных с АЦП.
- ✓ вывод **INT** на цифровом разъеме *DRB-37F*, который подключен к линии прерывания *IRQ0* цифрового сигнального процессора (прерывание может быть отконфигурировано для работы по уровню или по фронту). В фирменном драйвере эта линия не используется.

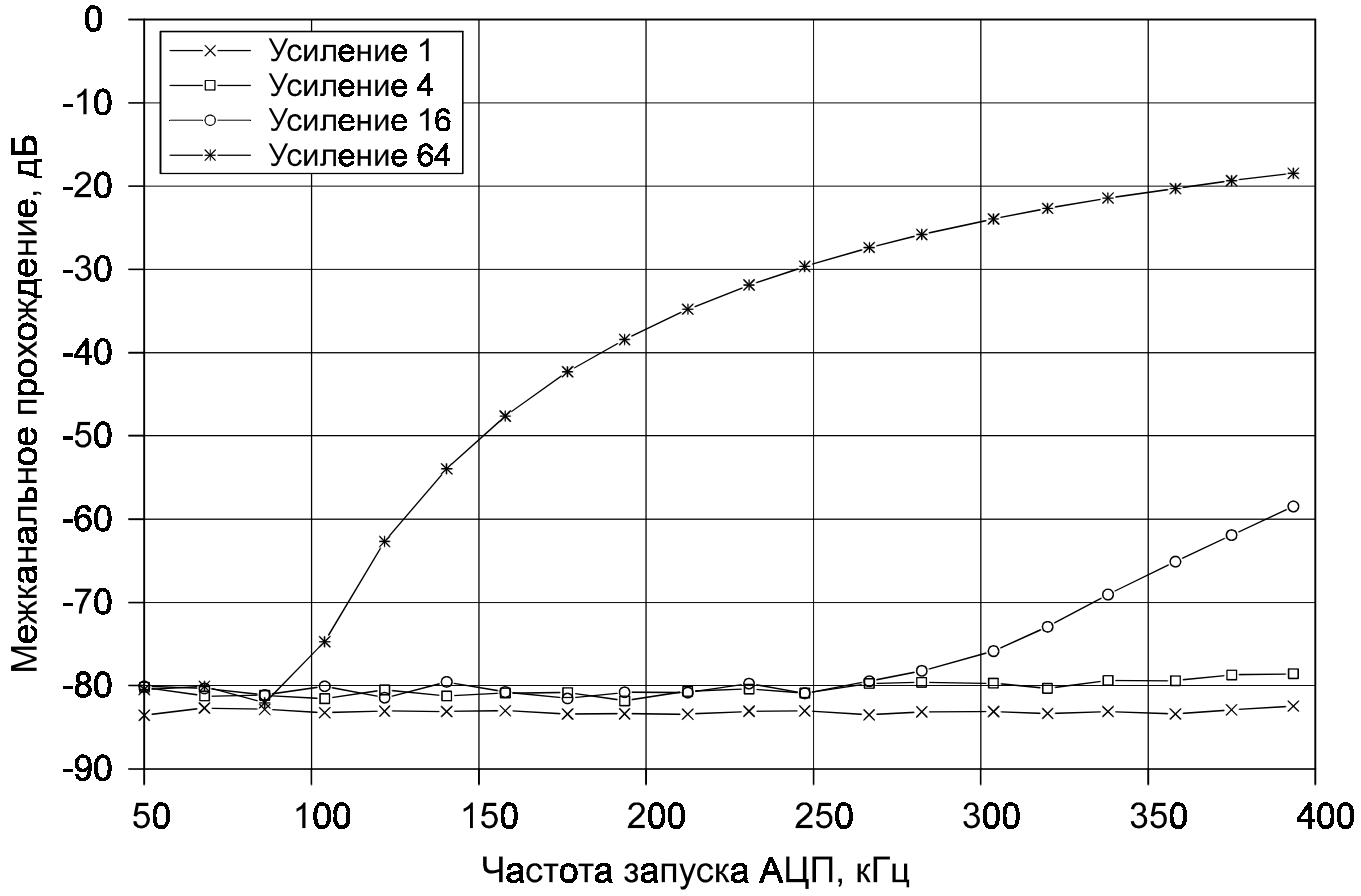
Каждая из указанных линий прерывания может быть индивидуально сконфигурированы на работу, как по фронту, так и по уровню (за это отвечает системный регистр DSP под названием *ICNTL*). Если прерывание работает по фронту, то оно генерируется при отрицательном перепаде импульса (⎍) длительностью не менее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

4. ПРИЛОЖЕНИЯ

4.1. ПРИЛОЖЕНИЕ А

Межканальное прохождение для модуля E-440

В данном приложении приводятся типичные зависимости межканального прохождения для модуля *E-440* на частоте входного синусоидального сигнала 10 кГц при различных частотах запуска АЦП и коэффициентах усиления.



4.2. ПРИЛОЖЕНИЕ В

Структура памяти сигнального процессора ADSP-2185M

Карта распределения памяти программ и памяти данных для цифрового сигнального процессора *ADSP-2185M* и расположение в ней программных блоков драйвера *LBIOS* приведена на следующем рисунке:

Карта распределения памяти для ADSP-2185

4.3. ПРИЛОЖЕНИЕ С

Утилита BIN3PCI.EXE и формат файла .BIO

Досовская утилита BIN3PCI.EXE предназначена исключительно для целей преобразования стандартного формата файла отображения в памяти (memory image file), сформированного редактором связей DSP ld21.exe, в формат .BIO, применяемый в ***ЗАО "Л-Кард"*** и являющийся более удобным для процесса загрузки цифрового сигнального процессора. Стандартный формат файла отображения в памяти (memory image file) в деталях описан в оригинальной книге *"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 E440.exe в файл E440.bio, в командной строке необходимо набрать следующую строчку:

```
bin3pci E440.exe
```

Файл .BIO содержит в обычном бинарном виде массив слов типа **WORD** (в C++) в, формат которого представлен в нижеследующей таблице:

Таблица 18. Формат файла BIO

Индекс	Назначение
0	Общее количество слов (**NPM**) типа **WORD** (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 **WORD** (в C++), которые надо грузить в память данных DSP, начиная с адреса DM(0x0)
NPM+2	Первое 16^{ти} битное слово, загружаемое по адресу DM(0x0)
NPM+3	Второе 16^{ти} битное слово, загружаемое по адресу DM(0x01)

. . . . .	
NPM+NDM+1	Последнее $16^{ти}$ битное слово, загружаемое по адресу DM(0x0+(**NDM**–1))

Следует помнить, что штатно загрузка памяти данных для штатного *LBIOS*'а не происходит поскольку, он написан таким образом, что изначально память данных DSP не инициализирована.