

# L-CARD

Устройства для мобильных систем

---

## E14-440

Внешний модуль на шину **USB 1.1**

*Потоковый ввод с АЦП: 16/32 канала, 440 кГц, 14 бит*

*Потоковый вывод на ЦАП: 2 канала, 12 бит (опция)*

*Асинхронный цифровой ввод-вывод: по 16 линий*

Библиотека *Lusbapi 3.3.*

*Windows'98/Me/2000/XP/Vista/7*

*Руководство программиста*

Москва. Февраль 2010 г.

Ревизия документа А2

## **ЗАО «Л-КАРД»,**

117105, г. Москва, Варшавское шоссе, д. 5, корп. 4, стр. 2.

тел. (495) 785-95-25

факс (495) 785-95-14

### **Адреса в Интернет:**

WWW: [www.lcard.ru](http://www.lcard.ru)

FTP: [ftp.lcard.ru](ftp://ftp.lcard.ru)

### **E-Mail:**

Общие вопросы: [lcard@lcard.ru](mailto:lcard@lcard.ru)

Отдел продаж: [sale@lcard.ru](mailto:sale@lcard.ru)

Техническая поддержка: [support@lcard.ru](mailto:support@lcard.ru)

Отдел кадров: [job@lcard.ru](mailto:job@lcard.ru)

### **Представители в регионах:**

Украина:	“ХОЛИТ Дэйта Системс, Лтд”	<a href="http://www.holit.com.ua">www.holit.com.ua</a>	380 (44) 241-67-54
Санкт-Петербург:	ЗАО “АВТЭКС Санкт-Петербург”	<a href="http://www.autex.spb.ru">www.autex.spb.ru</a>	(812) 567-7202
Санкт-Петербург:	Компания "Ниеншанц-Автоматика"	<a href="http://www.nnz-ipc.ru">www.nnz-ipc.ru</a>	(812) 326-59-24
Новосибирск:	ООО “Сектор Т”	<a href="http://www.sector-t.ru">www.sector-t.ru</a>	(3832) 22-76-20
Екатеринбург:	Группа Компаний АСК	<a href="http://www.ask.ru">www.ask.ru</a>	(343) 371-44-44
Екатеринбург:	ООО “Авеон”	<a href="http://www.aveon.ru">www.aveon.ru</a>	(343) 381-75-75
Казань:	ООО “Шатл”	<a href="mailto:shuttle@kai.ru">shuttle@kai.ru</a>	(8432) 38-16-00
Самара:	"АСУ-Самара"	<a href="mailto:prosoft-s@jiguli.ru">prosoft-s@jiguli.ru</a>	(846)-998-29-01

**E14-440.** Внешний модуль АЦП/ЦАП/ТТЛ общего назначения на шину **USB 1.1.**

© Copyright 1989–2010, **ЗАО “Л-Кард”**. Все права защищены.

# Оглавление

1. Введение.....	7
2. Общие сведения .....	7
2.1. Что нового? .....	7
2.1.1. Библиотека Lusbari 3.3 .....	7
2.2. Подключение модуля к компьютеру.....	8
2.3. Библиотека Lusbari .....	9
2.4. Загрузка управляющей программы.....	10
2.5. Возможные проблемы при работе с модулем .....	11
3. Используемые термины и форматы данных .....	12
3.1. Термины .....	12
3.2. Форматы данных .....	12
3.2.1. Формат слова данных с АЦП .....	12
3.2.2. Формат слова данных для ЦАП .....	13
3.2.3. Логический номер канала АЦП .....	13
3.3. Формат пользовательского ППЗУ .....	15
3.4. Формат кадра отсчетов .....	16
4. Общие принципы работы с модулем E14-440.....	17
4.1. Общий подход к работе с интерфейсными функциями.....	17
4.2. Общая структура драйвера DSP.....	19
5. Описание библиотеки Lusbari .....	21
5.1. Константы .....	21
5.2. Структуры .....	26
5.2.1. Структура MODULE_DESCRIPTION_E440.....	26
5.2.2. Структура ADC_PARS_E440.....	26
5.2.3. Структура DAC_PARS_E440.....	27
5.2.4. Структура IO_REQUEST_LUSBAPI.....	27
5.2.5. Структура LAST_ERROR_INFO_LUSBAPI .....	27
5.3. Драйвер DSP .....	28
5.3.1. Переменные драйвера DSP .....	28
5.3.2. Номера команд драйвера DSP .....	31

5.4. Функции общего характера.....	32
5.4.1. Получение версии библиотеки.....	32
5.4.2. Получение указателя на интерфейс модуля.....	32
5.4.3. Завершение работы с интерфейсом модуля.....	33
5.4.4. Инициализация доступа к модулю .....	33
5.4.5. Завершение доступа к модулю.....	34
5.4.6. Загрузка модуля .....	34
5.4.7. Проверка загрузки модуля .....	35
5.4.8. Получение названия модуля.....	35
5.4.9. Получение скорости работы модуля.....	35
5.4.10. Сброс модуля .....	36
5.4.11. Передача номер команды в драйвер DSP.....	36
5.4.12. Получение дескриптора устройства .....	37
5.4.13. Получение описания ошибок выполнения функций.....	37
5.5. Функции для доступа к памяти DSP модуля .....	38
5.5.1. Чтение слова из памяти данных DSP.....	38
5.5.2. Чтение слова из памяти программ DSP.....	38
5.5.3. Запись слова в память данных DSP .....	38
5.5.4. Запись слова в память программ DSP .....	39
5.5.5. Чтение массива слов из памяти данных DSP.....	39
5.5.6. Чтение массива слов из памяти программ DSP.....	39
5.5.7. Запись массива слов в память данных DSP .....	40
5.5.8. Запись массива слов в память программ DSP .....	40
5.5.9. Чтение переменной драйвера DSP .....	41
5.5.10. Запись переменной драйвера DSP .....	41
5.6. Функции для работы с АЦП.....	42
5.6.1. Корректировка данных АЦП.....	42
5.6.2. Запуск сбора данных АЦП .....	43
5.6.3. Останов сбора данных АЦП.....	43
5.6.4. Установка параметров работы АЦП .....	44
5.6.5. Получение текущих параметров работы АЦП .....	48
5.6.6. Получение массива данных с АЦП .....	49
5.6.7. Ввод кадра отсчетов с АЦП .....	52
5.6.8. Однократный ввод с АЦП.....	52
5.7. Функции для работы с ЦАП.....	53
5.7.1. Корректировка данных ЦАП.....	53
5.7.2. Запуск вывода данных на ЦАП.....	54
5.7.3. Останов вывода данных на ЦАП .....	54
5.7.4. Установка параметров работы ЦАП .....	55

5.7.5. Получение текущих параметров работы ЦАП .....	56
5.7.6. Передача массива данных в ЦАП .....	56
5.7.7. Однократный вывод на ЦАП .....	59
5.8. Функции для работы с внешними цифровыми линиями .....	60
5.8.1. Разрешение выходных цифровых линий .....	60
5.8.2. Чтение внешних цифровых линий .....	60
5.8.3. Вывод на внешние цифровые линии .....	61
5.9. Функции для работы с пользовательским ППЗУ .....	62
5.9.1. Разрешение/запрещение записи в ППЗУ .....	62
5.9.2. Запись слова в ППЗУ .....	62
5.9.3. Чтение слова из ППЗУ .....	63
5.10. Функции для работы со служебной информацией .....	64
5.10.1. Чтение служебной информации.....	64
5.10.2. Запись служебной информации.....	64
5.11. Функции для работы с загрузочным ППЗУ .....	65
5.11.1. Загрузка DSP модуля.....	65
5.11.2. Сброс загрузочного ППЗУ .....	65
5.11.3. Запись загрузочного ППЗУ .....	65
5.11.4. Чтение загрузочного ППЗУ .....	66
Приложение А. Структура памяти драйвера DSP .....	67
Приложение В. Вспомогательные константы и типы .....	68
В.1. Константы .....	68
В.2. Структура VERSION_INFO_LUSBAPI .....	68
В.3. Структура MODULE_INFO_LUSBAPI .....	68
В.4. Структура INTERFACE_INFO_LUSBAPI .....	69
В.5. Структура MCU_INFO_LUSBAPI.....	69
В.6. Структура DSP_INFO_LUSBAPI .....	69
В.7. Структура ADC_INFO_LUSBAPI .....	69
В.8. Структура DAC_INFO_LUSBAPI .....	70
В.9. Структура DIGITAL_IO_INFO_LUSBAPI.....	70



# 1. Введение

Данное описание предназначено для пользователей, собирающихся разрабатывать свои собственные приложения в операционной среде *Windows '98/2000/XP/Vista/7* для работы с модулями *E14-440*. Предварительно настоятельно рекомендуется внимательно ознакомиться с "*E14-440. Руководство пользователя*", где можно найти достаточно подробную информацию по подключению входных сигналов, распиновке внешних разъёмов, о характерных неисправностях и т.п.

В качестве базового языка при написании штатного программного обеспечения нами был выбран язык C++, а конкретнее, старый надёжный **Borland C++ 5.02**. Для модуля *E14-440* фирма ЗАО "А-Кард" предоставляет **USB** драйвер устройства, готовую динамически подключаемую библиотеку *Lusbapi*, а также целый ряд законченных примеров. Причём библиотека и все примеры поставляются вместе с исходными текстами, снабжёнными достаточно подробными комментариями. Библиотека *Lusbapi* позволяет использовать практически все возможности модуля, не вдаваясь в тонкости их низкоуровневого программирования. Если же пользователь все-таки собирается программировать модуль *E14-440* на низком уровне, то библиотека *Lusbapi* может быть использована в качестве законченного и отлаженного руководства, на основе которого можно реализовывать свои собственные алгоритмы, включая алгоритмы реального времени.

Модуль *E14-440* разрабатывался с главной целью – обеспечить быстрый бесшумный ввод в компьютер аналоговой информации. Для этого штатная библиотека *Lusbapi* содержит целый ряд функций, позволяющих организовывать многоканальный *непрерывный потоковый* сбор аналоговых данных на частотах АЦП вплоть до **400 кГц**. При вводе данных можно использовать определённый вид синхронизации: цифровую или аналоговую. Наряду с вводом данных, библиотека позволяет организовывать *непрерывный потоковый* вывод аналоговой информации в одно- или двухканальном режиме (только при наличии микросхемы ЦАП на модуле). Кроме того, библиотека содержит также ряд функций, позволяющий осуществлять ввод-вывод аналоговой и цифровой информации в однократном, и поэтому достаточно медленном, режиме. Также пользователь может осуществлять конфигурацию на уровне DSP модуля *FIFO* буферов для АЦП и ЦАП, также многое другое. Мы надеемся, что описываемая ниже библиотека *Lusbapi* упростит и ускорит написание Ваших собственных приложений.

Полный пакет штатного программного обеспечения модуля *E14-440* для работы в среде *Windows '98/2000/XP/Vista/7* находится на прилагаемом к модулю фирменном CD-ROM'e в директории `\USB\Lusbapi`. **!!!ВНИМАНИЕ!!!** Далее по тексту данного описания все директории, касающиеся непосредственно модуля *E14-440*, указаны относительно неё. Также весь пакет штатного программного обеспечения можно скачать с нашего сайта [www.lcard.ru](http://www.lcard.ru) из раздела "*Библиотека файлов*". Там из подраздела "*ПО для внешних модулей*" следует выбрать самораспаковывающийся архив `lusbapiXY.exe`, где **X**, **Y** обозначает номер версии программного обеспечения. На момент написания данного руководства последняя библиотека *Lusbapi* имеет версию **3.3**, а содержащий её архив называется `lusbapi33.exe`.

## 2. Общие сведения

### 2.1. Что нового?

Как правило, в данном параграфе будут приводиться только основные изменения как аппаратного, так и программного характера. За более подробной информацией следует обращаться к:

- "*E14-440. Руководство пользователя*";
- "*E14-440. Библиотека Lusbapi. История дополнений и изменений*".

#### 2.1.1. Библиотека Lusbapi 3.3

В библиотеке *Lusbapi* версии **3.3** реализована поддержка новых функциональных возможностей, появившихся у ревизии **F** модуля *E14-140*. Этот модуль представляет собой продукт основа-

тельной аппаратной модернизации *E14-140*. С точки зрения программиста отметим два основных отличия от предыдущих ревизий модуля:

- Расширены типы тактовых импульсов для АЦП модуля:
- На модуле появилась загрузочное ППЗУ, где можно хранить образ управляющей программы DSP.

## 2.2. Подключение модуля к компьютеру

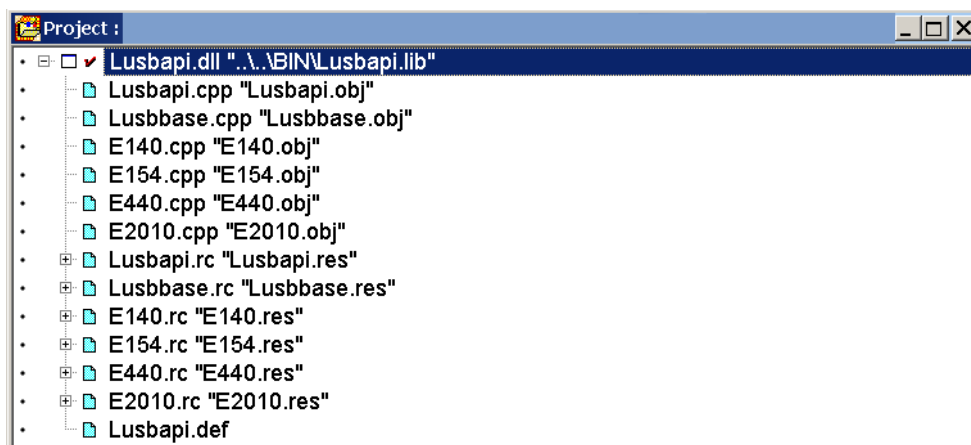
Все подробности по процедуре аппаратного подключения модуля *E14-440* к компьютеру пользователя и надлежащей установке **USB** драйверов можно найти в *"E14-440. Руководство пользователя, § 2 "Инсталляция и настройка"*.

Стоит особо подчеркнуть, что, начиная с версии **3.2** в библиотеке `Lusbapi` изменился основной файл **USB** драйвера. Теперь он называется **Ldevusb.sys** вместо бывшего ранее **Ldevusb.sys**. Т.о. при переходе со старых версий **Lusbapi** на более новую версию **3.22** и выше, конечному пользователю следует через *"Device Manager"* (*"Диспетчер устройств"*) переключить модуль *E14-440* на работу с новым **USB** драйвером.



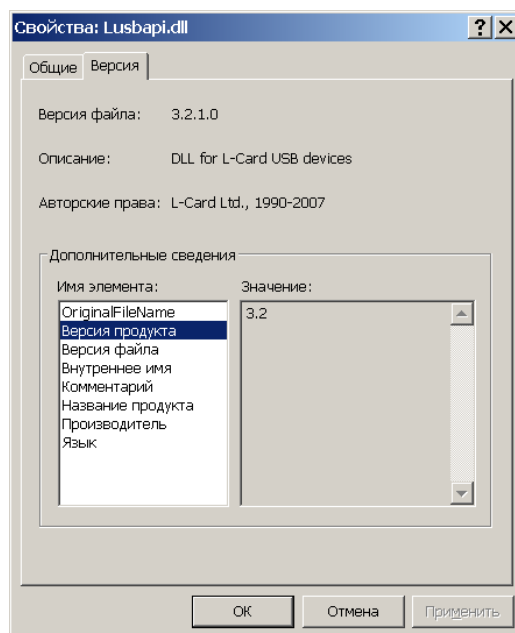
## 2.3. Библиотека Lusbapi

Штатная библиотека Lusbapi написана с использованием весьма доступного языка программирования **Borland C++ 5.02**. Кроме модуля *E14-440* в библиотеке также осуществлена поддержка модулей типа *E14-140*, *E-154* и *E20-10*. Общий вид проекта библиотеки Lusbapi в интегрированной среде разработки **Borland C++ 5.02** представлен на рисунке ниже:



Собственно, сама библиотека содержит всего две экспортируемые функции, одна из которых *CreateLInstance()* возвращает указатель на интерфейс модуля *E14-440*. 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 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 конечном 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 библиотеки Lusbapi.dll. Во всплывшем на экране монитора меню следует выбрать опцию ‘*Properties*’ (‘*Свойства*’), после чего на появившейся панели выбрать закладку ‘*Version*’ (‘*Версия*’). На этой закладке в строчке ‘*File version*’ (‘*Версия файла*’) можно прочитать текущий номер версии библиотеки:



Сам файл штатной библиотеки `Lusbapi.dll` расположен на фирменном CD-ROM в директории `\DLL\BIN`. Её исходные тексты можно найти в директории `\DLL\Source\Lusbapi`. Заголовочные файлы хранятся в директории `\DLL\Include`, а библиотеки импорта и модули объявления для различных сред разработки можно найти в директории `\DLL\Lib`.

Тексты законченных примеров применения интерфейсных функций из штатной DLL библиотеки для различных сред разработки приложений можно найти в следующих директориях:

- `\E14-440\Examples\Borland C++ 5.02`;
- `\E14-440\Examples\Borland C++ Builder 5.0`;
- `\E14-440\Examples\Borland Delphi 6.0`;
- `\E14-440\Examples\Microsoft Visual C++ 6.0`.

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

- создать файл проектов. Например, `test.ide` для среды **Borland C++ 5.02**;
- добавить файл библиотеки импорта `\DLL\Lib\Borland\LUSBAPI.LIB`;
- создать и добавить в проект приложения основной файл с будущей программой, например, `test.cpp`;
- включить в начало вашего файла заголовочный файл `#include "LUSBAPI.H"`, содержащий описание интерфейса модуля **E14-440**;
- далее желательно сравнить версию используемой библиотеки `Lusbapi` с версией текущего программного обеспечения, для чего можно задействовать функцию `GetDllVersion()`;
- вызвать функцию `CreateLInstance()` для получения указателя на интерфейс модуля;
- в общем-то, **ВСЁ!** Теперь Вы можете писать свою программу и в любом месте, используя полученный указатель, вызывать соответствующие интерфейсные функции из штатной библиотеки `Lusbapi.dll`.

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

1. Динамическая загрузка штатной DLL на этапе выполнения приложения. Подробности смотри в исходных текстах примера из директории `\E14-440\Examples\Microsoft Visual C++ 6.0\DynLoad`.
2. При статической компоновке штатной DLL в Вашем проекте использовать файл библиотеки импорта `LUSBAPI.LIB` из директории `\DLL\Lib\Microsoft`.

При работе с модулем типа **E20-10** в среде **Borland Delphi** рекомендуется применять модуль объявлений `LUSBAPI.PAS`, расположенный в директории `\DLL\Lib\Delphi`. Также вместо исходного модуля объявлений вполне можно задействовать уже откомпилированную версию `LUSBAPI.DCU`.

## 2.4. Загрузка управляющей программы

При работе с модулем следует учитывать одну замечательную особенность, отличающую **E14-440** от более простых устройств ввода-вывода: на нем установлен цифровой сигнальный процессор **ADSP-2185M** от фирмы *Analog Devices, Inc.* Для работы с модулем в этот процессор необходимо предварительно загрузить управляющую программу (драйвер DSP, **LBIOS**). В состав штатного программного обеспечения `Lusbapi` входит законченная управляющая программа, состоящая из одного бинарного файла `\E14-440\DSP\E440.bio`. Данный файл содержит как исполняемый код управляющей программы, так и сегмент данных для сигнального процессора.

В штатной библиотеке `Lusbapi` для загрузки **LBIOS** в сигнальный процессор модуля имеется специальная интерфейсная функция `LOAD_MODULE()`, которая аккуратно выполняет процедуру загрузки DSP модуля.

В новой ревизии **F** модуля **E14-140** появилась возможность автоматической загрузки DSP после подачи питания. Для этого на модуль устанавливается специальное загрузочное ППЗУ.

Только **ПОСЛЕ** загрузки **LBIOS**'а приложение может переходить к управлению модулем, т.е. переводить его в различные режимы работы с АЦП, ЦАП, цифровыми линиями и т. д.

## 2.5. Возможные проблемы при работе с модулем

1. Перед началом работы со штатным программным обеспечением модуля *E14-440*, во избежание непредсказуемого его поведения, настоятельно рекомендуется установить драйвера для чипсета используемой материнской платы компьютера. В особенности это касается чипсетов не от *Intel: VIA, SIS, nVidia, AMD+ATI* и т.д. Обычно эти драйвера можно найти на фирменном CD-ROM, который поставляется вместе с материнской платой, или скачать сайта производителя.

2. Компьютеры, у которых материнская плата создана на основе чипсета от фирмы *SIS (Silicon Integrated System Corporation)*, *AMD+ATI (Advanced Micro Devices, Inc.)* или *nVidia (NVIDIA Corporation)*, не совсем корректно работают в среде *Windows '98/2000/XP/Vista/7*. Это проявляется при запросах с большим кол-вом данных в интерфейсных функциях *ReadData()*. Например, при вызове этой функции с параметром *NumberOfWordsToRead = 1024\*1024* операционная система *Windows* вполне может, что называется, 'наглухо' зависнуть вплоть до появления *BSOD (Blue Screen Of Death)*. Решение данной проблемы лежит в русле уменьшения значения *NumberOfWordsToRead*. Причём величина *NumberOfWordsToRead*, при которой всё начинает нормально работать, зависит от конкретного экземпляра компьютера. Так что следует попробовать просто поварьировать величину параметра *NumberOfWordsToRead*.

## 3. Используемые термины и форматы данных

### 3.1. Термины

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

### 3.2. Форматы данных

#### 3.2.1. Формат слова данных с АЦП

Данные, считанные с 14<sup>ти</sup> битного АЦП модуля *E14-440*, представляются в формате знакового целого двухбайтного числа от -8192 до 8191. Точностные пределы кодов АЦП, соответствующие выбранному входному диапазону, приведены в следующей таблице:

Таблица 1. Соответствие кода АЦП напряжению на аналоговом входе

Модуль	Усиление	Напряжение, В	Код	Точность, %
<i>E14-440</i>	1; 4; 16; 64	+MAX	+8000	2÷3
		0	0	0.25; 0.3; 0.5; 1.0
		-MAX	-8000	2÷3

где MAX – значение входного диапазона (см. [Таблица 6](#)).

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. § 3.3. "Формат пользовательского ППЗУ"). Для случая, когда *LBIOS* у модуля предписано производить такую корректировку кодов входного сигнала, соответствующие точностные параметры АЦП приведены ниже (при температуре 25°C):

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

Модуль	Усиление	Напряжение, В	Код	Точность, %
E14-440	1; 4; 16; 64	+MAX	+8000	0.05; 0.075; 0.1; 0.15
		0	0	
		-MAX	-8000	

где MAX – значение входного диапазона (см. Таблица 6).

### 3.2.2. Формат слова данных для ЦАП

На модуле по желанию пользователя может быть установлена микросхема 2<sup>x</sup> канального 12<sup>ти</sup> битного ЦАП. Формат 16<sup>ти</sup> битного слова данных, передаваемого из РС в модуль для последующей выдачи на ЦАП, приведен в следующей таблице:

Таблица 3. Формат слова данных ЦАП

Модуль	Номер бита	Назначение
E14-440	0÷11	12 <sup>ти</sup> битный код ЦАП
	12	Выбор номера канала ЦАП: ✓ '0' – первый канал; ✓ '1' – второй канал.
	13÷15	Не используются

Собственно сам код, выдаваемый модулем на 12<sup>ти</sup> битный ЦАП, связан с устанавливаемым на внешнем разъеме напряжением в соответствии со следующей таблицей

Таблица 4. Соответствие кода ЦАП напряжению на внешнем аналоговом разъеме

Модуль	Код	Напряжение
E14-440	+2047	+5.0 Вольт
	0	0.0 Вольт
	-2048	-5.0 Вольт

### 3.2.3. Логический номер канала АЦП

Для управления работой входного аналогового каскада модуля *E14-440* был введен такой параметр как 8<sup>ми</sup> 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;
- индекс коэффициента усиления, т.е. для каждого логического канала можно установить свой индивидуальный коэффициент усиления (диапазон входного напряжения).

Битовый формат логического номера канала АЦП представлен в таблице ниже:

Таблица 5. Формат логического номера канала.

Номер бита	Обозначение	Функциональное назначение
0	<b>MA0</b>	0 <sup>ой</sup> бит номера канала
1	<b>MA1</b>	1 <sup>ый</sup> бит номера канала
2	<b>MA2</b>	2 <sup>ой</sup> бит номера канала
3	<b>MA3</b>	3 <sup>ий</sup> бит номера канала
4	<b>MA4</b>	Калибровка нуля/4 <sup>ый</sup> бит номера канала
5	<b>MA5</b>	16 диф./32 общ.
6	<b>GS0</b>	0 <sup>ой</sup> бит коэффициента усиления (см. <a href="#">Таблицу 6</a> )
7	<b>GS1</b>	1 <sup>ый</sup> бит коэффициента усиления (см. <a href="#">Таблицу 6</a> )

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 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 **E14-440** равный **0x2** означает дифференциальный режим работы 3<sup>его</sup> канала с единичным усилением, **0x82** – тот же канал с усилением равным 16. Если же логический номер равен **0x10** или **0x14**, то вход каскада PGA будет просто заземлен (именно PGA, а не входы указанных каналов коммутатора).

Таблица 6. Коэффициент усиления (биты GS0 и GS1)

Модуль	Бит GS1	Бит GS0	Усиление	Диапазон, В
<b>E14-440</b>	0	0	1	±10.0
	0	1	4	±2.5
	1	0	16	±0.625
	1	1	64	±0.15625

Например, если в логическом номере канала АЦП установить битовое поле **GS1÷GS0** равным 2, то тем самым будет выбран коэффициент усиления 16, а диапазон входного напряжения при этом составит ±0.625 В.

### 3.3. Формат пользовательского ППЗУ

На модуле *E14-440* установлено пользовательское ППЗУ емкостью 64 Слова×16 бит. Формат данного ППЗУ представлен на следующем рисунке:

Служебная область ППЗУ			Пользовательская область ППЗУ	
Информация о модуле	Калибровочные коэффициенты АЦП	Калибровочные коэффициенты ЦАП		
0	20	28	32	64

Номер ячейки в ППЗУ

Как видно из рисунка, в первых 32<sup>х</sup> словах (64 байта) находится служебная область ППЗУ модуля. 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 драйвером DSP данных, получаемых с АЦП. Данные коэффициенты записываются в ППЗУ при наладке модуля в ЗАО “А-Кард”. Благодаря этому на модуле отсутствуют подстроечные резисторы, что сильно улучшает шумовые характеристики модуля и увеличивает их надежность. Для хранения калибровочных коэффициентов использован специальный дробный формат **1.15**, который предназначен специально для работы с *LBIOS*. Т.о. 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 данных, выводимых на ЦАП. Данные коэффициенты записываются в ППЗУ при наладке модуля в ЗАО “А-Кард”. Коэффициенты хранятся в специальном формате в виде чисел типа *WORD* языка C++ (2 байта) и имеют следующий порядок:

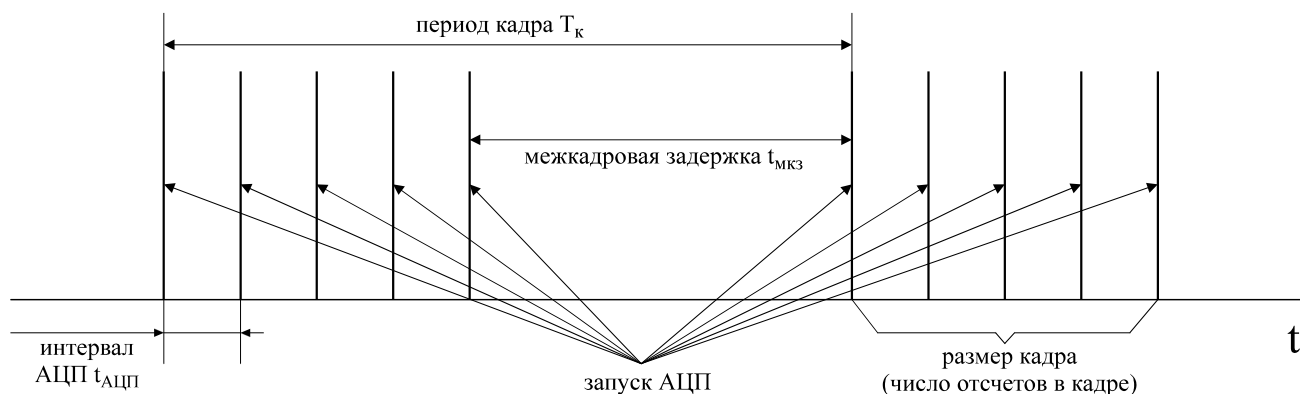
- ✓ 28 ячейка – корректировка смещения нуля первого ЦАП’а;
- ✓ 29 ячейка – корректировка смещения нуля второго ЦАП’а;
- ✓ 30 ячейка – корректировка масштаба первого ЦАП’а;
- ✓ 31 ячейка – корректировка масштаба второго ЦАП’а;

Все приведённые корректировочные коэффициенты преобразуются к более привычному формату типа *double* в соответствующих полях структуры [MODULE\\_DESCRIPTION\\_E440](#), когда служебная область ППЗУ зачитывается интерфейсной функцией [GET\\_MODULE\\_DESCRIPTION\(\)](#).

Пользовательская область ППЗУ начинается с 32<sup>ой</sup> ячейки. Туда совершенно спокойно можно записывать или считывать любую свою информацию. Для этого следует воспользоваться соответствующими интерфейсными функциями: [WRITE\\_FLASH\\_WORD\(\)](#) и [READ\\_FLASH\\_WORD\(\)](#).

### 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 управляющую таблицу в сигнальный процессор модуля можно с помощью штатной интерфейсной функции **SET\_ADC\_PARS()**. Подробнее об этой функции смотри § 5.7.4. "Установка параметров работы АЦП". Временные параметры кадра для случая **ControlTableLength = 5** приведены на следующем рисунке:



где  $T_k$  – временной интервал между соседними кадрами, т.е. это фактически частота опроса фиксированного логического номера канала **KadrRate**;  $t_{МКЗ} = \text{InterKadrDelay}$  – временной интервал между последним отсчетом текущего кадра и первым отсчетом следующего;  $t_{АЦП}$  – интервал запуска АЦП или межканальная задержка. Тогда  $1/t_{АЦП} = \text{AdcRate}$  – частота работы АЦП или оцифровки данных, а величина  $t_{МКЗ}$  не может принимать значения меньше, чем  $t_{АЦП}$ . Если размер кадра, т.е. число отсчетов с АЦП в кадре, равен **ControlTableLength**, то все эти временные параметры можно связать следующей формулой:

$$T_k = 1/\text{KadrRate} = (\text{ControlTableLength}-1) * t_{АЦП} + t_{МКЗ},$$

или

$$T_k = 1/\text{KadrRate} = (\text{ControlTableLength}-1)/\text{AdcRate} + \text{InterKadrDelay}.$$

При выборе необходимого режима функционирования сбора данных с АЦП, такие важные временные параметры как **AdcRate** и **InterKadrDelay** задаются через штатную интерфейсную функцию **SET\_ADC\_PARS()**.



## 4. Общие принципы работы с модулем E14-440

### 4.1. Общий подход к работе с интерфейсными функциями

Целью штатной библиотеки `Lusbapi`, поставляемой с модулем *E14-440*, 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 библиотекой в пользовательской программе необходимо сделать следующие объявления (как минимум):

```
ILE440 *pModule;           // указатель на интерфейс модуля E14-440
MODULE_DESCRIPTION_E440 md; // структура служебной информации о модуле
```

Первым делом с помощью функции `GetDllVersion()` следует сравнить версии используемой библиотеки `Lusbapi.dll` 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, следует проинициализировать доступ к виртуальному слоту, к которому подключён модуль *E14-440*. Для этого предусмотрена интерфейсная функция `OpenLDevice()`. Если нет ошибки при выполнении этой функции, то можно быть уверенным, что устройство типа *E14-440* обнаружено в выбранном виртуальном слоте.

Теперь, в принципе, можно переходить к этапу загрузки обнаруженного модуля, но иногда нелишне бывает определиться с текущей скоростью работы используемого **USB** порта. Для этого предназначена интерфейсная функция `GetUsbSpeed()`. Модуль совместно с **USB** портом должен работать в так называемом **Full-Speed Mode**. Это будет соответствовать пропускной способности модуля по **USB** шины порядка 1 Мбайт/с.

Важной особенностью модуля *E14-440* является то, что на нем установлен достаточно мощный современный цифровой сигнальный процессор (DSP – Digital Signal Processor) с фиксированной точкой *ADSP-2185M* от фирмы *Analog Devices, Inc.* Для того, чтобы его “оживить”, т.е. заставить работать по требуемому алгоритму, во внутреннюю память DSP надо записать (загрузить) либо фирменную управляющую программу, которая входит в комплект штатной поставки (файл `\E14-440\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 штатного драйвера DSP расположены программно организованные *FIFO* буфера АЦП и ЦАП, а также переменные *LBIOS* (см. *Приложение А*). О низкоуровневом взаимодействии компьютера с модулем *E14-440*, с одной стороны, и DSP с периферией, с другой, смотри руководство *"E14-440. Низкоуровневое описание"*

Т.о., необходимо загрузить в сигнальный процессор модуля управляющую программу (*LBIOS*). Для этого можно воспользоваться интерфейсной функцией `LOAD_MODULE()`. 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.

На следующем этапе следует прочитать служебную информацию о модуле. Она требуется при работе с некоторыми интерфейсными функциями библиотеки `Lusbapi`. Интерфейсная функция `GET_MODULE_DESCRIPTION()` как раз и предназначена для этой цели. Если функция не вернула ошибку, то это означает, что вся служебная информация о модуле успешно получена и можно продолжить работу.

В общем-то, **ВСЁ!** Теперь можно спокойно управлять всей доступной периферией на модуле и самим DSP с помощью соответствующих интерфейсных функций штатной библиотеки. Т.е. зада-

вать различные режимы работы АЦП (приём 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, а вернее сказать ‘скелет’, консольной программы для работы с **E14-440**, предполагая использование `Lusbapi` версии не ниже 3.0:

```
#include <stdlib.h>
#include <stdio.h>
#include "Lusbapi.h"           // заголовочный файл библиотеки Lusbapi
ILE440 *pModule;             // указатель на интерфейс модуля
MODULE_DESCRIPTION_E440 md;  // структура с информацией о модуле
char ModuleName[7];          // название модуля
BYTE UsbSpeed;               // скорость работы шины USB

int main(void)
{
    // проверим версию DLL библиотеки
    if(GetDllVersion() != CURRENT_VERSION_LUSBAPI)
    {
        printf("Неправильная версия Dll!");
        return 1;           //выйдем из программы с ошибкой
    }

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

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

    // попробуем получить скорость работы шины USB
    if(!pModule->GetUsbSpeed(&UsbSpeed))
    {
        printf("Не могу узнать скорость работы USB!\n");
        return 1;           //выйдем из программы с ошибкой
    }

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

```

// проверим: этот модуль - 'E14-440'?
if(strcmp(ModuleName, "E440"))
{
    printf(" В нулевом виртуальном слоте не 'E14-440'\n");
    return 1;          //выйдем из программы с ошибкой
}
// теперь можно попробовать загрузить из соответствующего ресурса
// библиотеки Lusbapi код драйвера LBIOS
if(!pModule->LOAD_MODULE())
{
    printf("Не выполнена функция LOAD_MODULE()!");
    return 1;          //выйдем из программы с ошибкой
}
// проверим работоспособность загруженного LBIOS
if(!pModule->MODULE_TEST())
{
    printf("Не выполнена функция MODULE_TEST()!");
    return 1;          //выйдем из программы с ошибкой
}
// попробуем прочитать информацию о модуле
if(!pModule->GET_MODULE_DESCRIPTION(&md))
{
    printf("Не выполнена функция GET_MODULE_DESCRIPTION ()!");
    return 1;          //выйдем из программы с ошибкой
}

printf("Модуль E14-440 (серийный номер %s) полностью готов к\
        работе!", md.Module.SerialNumber);

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

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

// выйдем из программы
return 0;
}

```

## 4.2. Общая структура драйвера DSP

На модуле *E14-440* устанавливается так называемый цифровой сигнальный процессор (Digital Signal Processor – DSP) с фиксированной точкой *ADSP-2185M* от 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. Для этого достаточно лишь овладеть достаточно несложным языком ассемблера DSP. Так, в штат-

ном драйвере *LBIOS*, исходные тексты которого можно найти в директории `\E14-440\DSP\` на прилагаемом к данному модулю фирменном CD-ROM, реализуются наиболее широко используемые алгоритмы работы с АЦП, ЦАП, входными/выходными ТТЛ линиями и т.д.

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

- ✓ 24<sup>х</sup> битная память программ (**Program Memory – PM**), в которой хранятся коды инструкций управляющей программы (драйвера *LBIOS*), а также, возможно, данные;
- ✓ 16<sup>тн</sup> 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, которым посвящён [§ 5.5. "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 А](#). 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 пользовательским приложением в РС по так называемому принципу команд. Об этом подробнее можно посмотреть в руководстве ["E14-440. Низкоуровневое описание § 1.2. Структурная схема модуля"](#).

Адреса предопределенных переменных в **PM** DSP модуля, задающих важные параметры функционирования штатного *LBIOS*, а также их краткие толкования, приведены в [Таблице 7](#).

В *LBIOS* программно организовано два циклических *FIFO* буфера: для приёма данных с АЦП и для вывода данных на ЦАП. Передача данных из *FIFO* буфера АЦП в РС производится порциями по *Длина\_FIFO\_Буфера\_АЦП/2* отсчётов по мере их поступления с АЦП. Т.е., фактически чисто программным образом реализован так называемый двойной *FIFO* буфер. Т.е. при поступлении из РС команды на запуск АЦП, драйвер *LBIOS* ожидает накопления данных в первой половине *FIFO* буфера АЦП. После того как первая половина буфера полностью заполнится готовыми данными с АЦП, даётся команда на их передачу в РС. При этом **не прекращается** сбор данных во вторую половину *FIFO* буфера. После накопления данных во второй половине *FIFO* буфера опять дается команда на их передачу в РС и продолжается сбор данных уже в первую половину. И так до бесконечности по циклу, пока не придет команда из РС 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.

## 5. Описание библиотеки *Lusbari*

В настоящем разделе приведены достаточно подробные описания констант, структур и интерфейсных функций, входящих в состав штатной библиотеки *Lusbari* для модуля *E14-440*.

### 5.1. Константы

Приведённые ниже основные константы настоятельно рекомендуется использовать в исходных текстах приложения при работе с модулем *E14-440*. Это весьма повышает *читаемость* и *понимаемость* исходных текстов, а также значительно облегчает сопровождение программ. Рассматриваемые константы расположены в файле `\DLL\Include\Lusbari.h`.

1. Модулю *E14-440* доступны несколько режимов сброса устройства. Местом использования этих констант, как правило, является аргумент интерфейсной функции [RESET\\_MODULE\(\)](#).

Константа	Значение	Назначение
<b>INIT_E440</b>	0	Модуль подвергается полной переинициализации. После чего DSP модуля снова готов к загрузке.
<b>RESET_E440</b>	1	Это константа действует на модуль <i>E14-440</i> с ревизией 'E' и выше. При этом модуль подвергается процедуре полного сброса. После чего DSP находится в состоянии сброса, вторичное питание отключено. Это состояние с пониженным энергопотреблением.
<b>INVALID_RESET_TYPE_E440</b>	2	Неправильный тип сброса модуля.

2. У модуля *E14-440* есть четыре возможных диапазона входных напряжений, каждый из которых можно задать данными константами. Местом использования этих констант, как правило, являются битовые поля **GS0÷GS1** *логического номера канала АЦП*. Массив этих логических каналов образуют управляющую таблицу поля *ControlTable* структуры [ADC\\_PARS\\_E440](#).



Константа	Значение	Назначение
<b>ADC_INPUT_RANGE_10000mV_E440</b>	0	Данная константа задаёт входной диапазон, равный $\pm 10000$ мВ. Также можно эту константу можно использовать в качестве индекса для доступа к первому элементу константного массива <a href="#">ADC_INPUT_RANGES_E440</a> .
<b>ADC_INPUT_RANGE_2500mV_E440</b>	1	Данная константа задаёт входной диапазон, равный $\pm 2500$ мВ. Также можно эту константу можно использовать в качестве индекса для доступа ко второму элементу константного массива <a href="#">ADC_INPUT_RANGES_E440</a> .

<b>ADC_INPUT_RANGE_625mV_E440</b>	2	Данная константа задаёт входной диапазон, равный $\pm 625$ мВ. Также можно эту константу можно использовать в качестве индекса для доступа к третьему элементу константного массива <a href="#">ADC_INPUT_RANGES_E440</a> .
<b>ADC_INPUT_RANGE_156mV_E440</b>	3	Данная константа задаёт входной диапазон, равный $\pm 156$ мВ. Также можно эту константу можно использовать в качестве индекса для доступа к четвёртому элементу константного массива <a href="#">ADC_INPUT_RANGES_E440</a> .

3. У модуля *E14-440* ревизии **F** и выше появилась возможность задавать источник тактовых импульсов АЦП: внутренние без трансляции, внутренние с трансляцией и внешние. Данные константы определяют этот источник. Местом использования этих констант, как правило, является поле *AdcClockSource* структуры [ADC\\_PARS\\_E440](#).

Константа	Значение	Назначение
<b>INT_ADC_CLOCK_E440</b>	0	<b>Внутренние тактовые импульсы АЦП без трансляции.</b> При этом тактовые импульсы аппаратно генерируются посредством DSP модуля без трансляции на внешние разъёмы. Т.о. в этом режиме линия <b>IN16</b> внешнего цифрового разъёма может использоваться как обычная входная цифровая линия.
<b>INT_ADC_CLOCK_WITH_TRANS_E440</b>	1	<b>Внутренние тактовые импульсы АЦП с трансляции.</b> При этом тактовые импульсы аппаратно генерируются посредством DSP модуля и дополнительно транслируются на линию <b>IN16</b> внешнего цифрового разъёма. Т.о. в этом режиме линия <b>IN16</b> может использоваться <b>только</b> как выходная линия тактовых импульсов АЦП.
<b>EXT_ADC_CLOCK_E440</b>	2	<b>Внешние тактовые импульсы АЦП.</b> В этом режиме используется внешний источник тактовых импульсов, которые подключается к линии <b>IN16</b> цифрового разъёма модуля. Т.о. в этом режиме линия <b>IN16</b> внешнего цифрового разъёма должна использоваться <b>только</b> как входная цифровая линия для тактовых импульсов АЦП.
<b>INVALID_ADC_CLOCK_E440</b>	3	Неправильный тип источника тактовых импульсов АЦП.

4. Модуль *E14-440* в состоянии осуществлять различную синхронизацию ввода данных с АЦП. Данные константы определяют тип требуемой синхронизации. Местом использования этих констант, как правило, является поле *InputMode* структуры *ADC\_PARS\_E440*.

Константа	Значение	Назначение
<b>NO_SYNC_E440</b>	0	<i>Отсутствие синхронизации ввода.</i> Сбор данных с АЦП модуль начинает непосредственно после выполнения функции <i>START_ADC()</i> .
<b>TTL_START_SYNC_E440</b>	1	<i>Цифровая синхронизация начала ввода.</i> Непосредственно после выполнения функции <i>START_ADC()</i> модуль переходит в состояние ожидания прихода отрицательного перепада (  ) у TTL-совместимого одиночного импульса на входе <b>TRIG</b> аналогового разъёма <i>DRB-37M</i> . Длительность этого синхроимпульса должна быть не менее 50 нс. Только после этого модуль приступает к сбору данных.
<b>TTL_KADR_SYNC_E440</b>	2	<i>Цифровая покадровая синхронизация ввода.</i> Непосредственно после выполнения функции <i>START_ADC()</i> модуль переходит в состояние ожидания прихода отрицательного перепада (  ) у TTL-совместимого одиночного импульса на входе <b>TRIG</b> аналогового разъёма <i>DRB-37M</i> . Длительность этого синхроимпульса должна быть не менее 50 нс. После прихода синхроимпульса (см. выше) модуль собирает отсчеты только одного кадра. Во время ввода кадра данных, вся активность на линии <b>TRIG</b> игнорируется. Только после окончания сбора текущего кадра данных ожидается приход следующего импульса синхронизации и т.д.
<b>ANALOG_SYNC_E440</b>	3	<i>Аналоговая синхронизация начала ввода.</i> Непосредственно после выполнения функции <i>START_ADC()</i> модуль переходит в состояние ожидания выполнения условий аналоговой синхронизации. Модуль начинает собирать данные с АЦП только после выполнения заданных соотношений между полученным значением с заданного аналогового синхроканала и заданным пороговым значением.
<b>INVALID_SYNC_E440</b>	4	Неправильный вид синхронизации ввода данных.

5. На модуле *E14-440* по желанию пользователя может быть установлена микросхема двунального ЦАП. Состояние поля *Dac.Active* структуры служебной информации *MODULE\_DESCRIPTION\_E440* отражает факт наличия ЦАП на борту модуля.

Константа	Значение	Назначение
<b>DAC_INACCESSIBLE_E440</b>	0	На модуле полностью отсутствует микросхема ЦАП.
<b>DAC_ACCESSIBLE_E440</b>	1	На модуле присутствует микросхема ЦАП.

6. Ревизия модуля *E14-440* отражает определённые конструктивные особенности модуля. Она задаётся одной заглавной латинской буквой. Например, *первая* ревизия модуля обозначается как 'A'.

Константа	Значение	Назначение
<b>REVISION_A_E440</b>	0	Данные константы могут использоваться в качестве индекса для доступа к соответствующему элементу константного массива <i>REVISIONS_E440</i> .
<b>REVISION_B_E440</b>	1	
<b>REVISION_C_E440</b>	2	
<b>REVISION_D_E440</b>	3	
<b>REVISION_E_E440</b>	4	
<b>REVISION_F_E440</b>	5	

7. Различные константы для работы с модулем *E14-440*.

Константа	Значение	Назначение
<b>CURRENT_VERSION_LUSBAPI</b>	—	Версия используемой библиотеки <i>Lusbapi</i> . Как правило, используется совместно с функцией <i>GetDllVersion()</i> .
<b>MAX_CONTROL_TABLE_LENGTH_E440</b>	128	Максимально возможное кол-во логических каналов в управляющей таблице <b>ControlTable</b> .
<b>ADC_CALIBR_COEFS_QUANTITY_E440</b>	4	Кол-во корректировочных коэффициентов для данных АЦП. По одному на каждый входной диапазон. Корректировке подвергаются как смещение, так и масштаб данных АЦП.
<b>ADC_INPUT_RANGES_QUANTITY_E440</b>	4	Кол-во входных диапазонов.
<b>MAX_ADC_FIFO_SIZE_E440</b>	12288	Максимальный размер FIFO буфера АЦП в DSP модуля.
<b>DAC_CHANNELS_QUANTITY_E440</b>	2	Кол-во физических каналов ЦАП на модуле (при условии наличия микросхемы ЦАП на модуле).



<b>DAC_CALIBR_COEFS_QUANTITY_E440</b>	2	Кол-во корректировочных коэффициентов для данных ЦАП. По одному на каждый канал. Корректировке подвергаются как смещение, так и масштаб данных ЦАП.
<b>MAX_DAC_FIFO_SIZE_E440</b>	4032	Максимальный размер FIFO буфера ЦАП в DSP модуля.
<b>TTL_LINES_QUANTITY_E440</b>	16	Кол-во входных и выходных цифровых линий.
<b>REVISIONS_QUANTITY_E440</b>	6	Кол-во ревизий (модификаций) модуля.

8. Различные *константные массивы* для работы с модулем *E14-440*:

a. Массив доступных диапазонов входного напряжения АЦП в Вольтах:

```
const double
    ADC_INPUT_RANGES_E440 [ADC_INPUT_RANGES_QUANTITY_E440] =
    {
        10.0, 10.0/4.0, 10.0/16.0, 10.0/64.0
    };
```

b. Диапазон выходного напряжения ЦАП в Вольтах:

```
const double DAC_OUTPUT_RANGE_E440 = 5.0;
```

c. Ревизия модуля отражает определённые конструктивные особенности модуля. Она задаётся одной заглавной латинской буквой. Например, *первая* ревизия модуля обозначается через букву 'A'. Текущая ревизия модуля содержится в поле *Module.Revision* структуры служебной информации *MODULE\_DESCRIPTION\_E440*. Массив доступных ревизий модуля задаётся следующим образом.

```
const BYTE REVISIONS_E440 [REVISIONS_QUANTITY_E440] =
    {
        'A', 'B', 'C', 'D', 'E', 'F'
    };
```

## 5.2. Структуры

В данном разделе приведены основные типы структур, которые применяются в библиотеке `Lusbapi` при работе с модулем *E14-440*.

### 5.2.1. Структура `MODULE_DESCRIPTION_E440`

Структура `MODULE_DESCRIPTION_E440` описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct MODULE_DESCRIPTION_E440
{
    MODULE_INFO_LUSBAPI      Module;           // общая информация о модуле
    INTERFACE_INFO_LUSBAPI   Interface;        // информация об интерфейсе
    MCU_INFO_LUSBAPI<VERSION_INFO_LUSBAPI> Mcu; // информация о MCU
    DSP_INFO_LUSBAPI         Dsp;              // информация о DSP
    ADC_INFO_LUSBAPI         Adc;              // информация о АЦП
    DAC_INFO_LUSBAPI         Dac;              // информация о ЦАП
    DIGITAL_IO_INFO_LUSBAPI DigitalIo;         // информация о цифровом вводе-выводе
};
```

В данной структуре представлена самая общая служебная информация об используемом экземпляре модуля *E14-440*. Эта структура используется при работе с интерфейсными функциями [SAVE\\_MODULE\\_DESCRIPTION\(\)](#) и [GET\\_MODULE\\_DESCRIPTION\(\)](#). В определении этой структуры применяются вспомогательные константы и типы данных, описанные в [Приложении В](#).

### 5.2.2. Структура `ADC_PARS_E440`

Структура `ADC_PARS_E440` описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct ADC_PARS_E440
{
    BOOL IsAdcEnabled;           // состояние работы АЦП (только на чтение)
    BOOL IsCorrectionEnabled;    // управление коррекцией данных
    WORD AdcClockSource;         // источник тактовых импульсов АЦП
    WORD InputMode;              // режим синхронизации ввода данных с АЦП
    WORD SynchroAdType           // тип аналоговой синхронизации
    WORD SynchroAdMode;          // режим аналоговой синхронизации
    WORD SynchroAdChannel;       // канал АЦП при аналоговой синхронизации
    SHORT SynchroAdPorog;        // порог срабатывания аналоговой синхронизации
    WORD ChannelsQuantity;       // число активных каналов (размер кадра)
    WORD ControlTable[128];      // управляющая таблица с активными лог. каналами
    double AdcRate;              // частота работы АЦП в кГц
    double InterKadrDelay;        // межкадровая задержка в мс
    double KadrRate;             // частота кадра в кГц
    WORD AdcFifoBaseAddress;      // базовый адрес FIFO буфера АЦП в DSP модуля
    WORD AdcFifoLength;          // длина FIFO буфера АЦП в DSP модуля
    double AdcOffsetCoefs[ADC\_CALIBR\_COEFS\_QUANTITY\_E440]; // коррективка смещения АЦП: 4диапазона
    double AdcScaleCoefs[ADC\_CALIBR\_COEFS\_QUANTITY\_E440]; // коррективка масштаб АЦП: 4диапазона
};
```

Перед началом работы с АЦП необходимо заполнить поля данной структуры и передать её в модуль с помощью интерфейсной функции [SET\\_ADC\\_PARS\(\)](#). В описании этой функции подробно прокомментированы смысл и назначение всех полей данной структуры. При этом в качестве

корректировочных коэффициентов можно использовать значения из соответствующих полей структуры [MODULE\\_DESCRIPTION\\_E440](#). При необходимости можно считать из модуля текущие параметры функционирования АЦП с помощью интерфейсной функции [GET\\_ADC\\_PARS\(\)](#).

### 5.2.3. Структура DAC\_PARS\_E440

Структура *DAC\_PARS\_E440* описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct DAC_PARS_E440
{
    BOOL DacEnabled;           // состояние работы ЦАП (только на чтение)
    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 [SET\\_DAC\\_PARS\(\)](#). В описании этой функции подробно прокомментированы смысл и назначение всех полей данной структуры. Также при необходимости можно считать из модуля текущие параметры функционирования ЦАП, используя [GET\\_DAC\\_PARS\(\)](#).

### 5.2.4. Структура IO\_REQUEST\_LUSBAPI

Структура *IO\_REQUEST\_LUSBAPI* описана в файле `\DLL\Include\LusbapiTypes.h` и представлена ниже:

```
struct IO_REQUEST_LUSBAPI
{
    SHORT * Buffer;           // буфер для передаваемых данных
    DWORD NumberOfWordsToPass; // кол-во отсчетов, которые требуется передать
    DWORD NumberOfWordsPassed; // кол-во реально переданных отсчетов
    OVERLAPPED * Overlapped; // для синхронного запроса – NULL, а для асинхронного
                             // запроса – указатель на стандартную WinAPI
                             // структуру типа OVERLAPPED
    DWORD Timeout;          // для синхронного запроса – таймаут в мс, а для
                             // асинхронного запроса не используется
};
```

Данная структура используется функциями [ReadData\(\)](#) и [WriteData\(\)](#) при организации *поточных* передач данных между модулем и компьютером. В описании этих функций подробно прокомментированы смысл и назначения полей данной структуры.

### 5.2.5. Структура LAST\_ERROR\_INFO\_LUSBAPI

Структура *LAST\_ERROR\_INFO\_LUSBAPI* описана в файле `\DLL\Include\LusbapiTypes.h` и представлена ниже:

```
struct LAST_ERROR_INFO_LUSBAPI
{
    BYTE ErrorString[256];   // строка с кратким описанием последней ошибки
                             // библиотеки Lusbapi
    DWORD ErrorNumber;       // номер последней ошибки библиотеки Lusbapi
};
```

Данная структура используется функцией [GetLastErrorInfo\(\)](#) при выявлении ошибок выполнения интерфейсных функций библиотеки *Lusbapi*.

## 5.3. Драйвер DSP

### 5.3.1. Переменные драйвера DSP

Любой программист при желании может напрямую работать с памятью DSP модуля *E14-440* (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 памяти программ и данных для *ADSP-2185M*, который установлен на модуле, приведена в *Приложении А*.

Ниже в *Таблице 7* приводятся *предопределенные* адреса переменных штатного *LBIOS*, расположенных в памяти **программ** DSP, и их краткие описания. Собственно сами переменные *LBIOS* являются 16<sup>ти</sup> битными и располагаются в старших 16<sup>ти</sup> битах 24<sup>х</sup> битного слова памяти программ DSP (при этом младшие 8 из этих 24<sup>х</sup> 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. § 5.5. "Функции для доступа к памяти DSP модуля").

Таблица 7. Адреса управляющих переменных драйвера DSP

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 ( <i>LBIOS</i> ) по этому адресу должно читаться число 0x5555.
L_TMODE2_E440	0x33	Тестовая переменная. После загрузки драйвера ( <i>LBIOS</i> ) 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 <i>Таблице 8</i> .
L_DAC_SCLK_DIV_E440	0x37	Текущий делитель для <i>SCLK0</i> . 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.

<b>L_CUR_ADC_FIFO_LENGTH_E440</b>	0x3C	Текущая длина FIFO буфера АЦП. По умолчанию <b>L_CUR_ADC_FIFO_LENGTH_E440 = 0x3000</b> .
<b>L_ADC_FIFO_LENGTH_E440</b>	0x3E	Требуемая длина FIFO буфера АЦП. По умолчанию <b>L_ADC_FIFO_LENGTH_E440 = 0x3000</b> .
<b>L_CORRECTION_ENABLED_E440</b>	0x3F	Переменная запрещающая (0)/ разрешающая (1) корректировку данных аналоговых каналов при помощи калибровочных коэффициентов. По умолчанию <b>L_CORRECTION_ENABLED = 0x0</b> .
<b>L_ADC_SAMPLE_E440</b>	0x41	Данная переменная используется при однократном вводе с АЦП, храня оцифрованное значение.
<b>L_ADC_CHANNEL_E440</b>	0x42	Данная переменная используется при однократном вводе с АЦП, задавая логический номер канала.
<b>L_INPUT_MODE_E440</b>	0x43	Переменная, задающая тип синхронизации при вводе данных с АЦП (аналоговая, цифровая или ее отсутствие).
<b>L_SYNCHRO_AD_CHANNEL_E440</b>	0x46	При аналоговой синхронизации задает логический номер канала, по которому происходит синхронизация.
<b>L_SYNCHRO_AD_POROG_E440</b>	0x47	При аналоговой синхронизации задает порог срабатывания в кодах АЦП.
<b>L_SYNCHRO_AD_MODE_E440</b>	0x48	Переменная, задающая режим аналоговой синхронизации.
<b>L_SYNCHRO_TYPE_E440</b>	0x49	При аналоговой синхронизации задает тип срабатывания по уровню ( <b>0x0</b> ) либо по переходу ( <b>0x1</b> ).
<b>L_CONTROL_TABLE_LENGTH_E440</b>	0x4B	Размер управляющей таблицы (максимум 128 логических каналов). По умолчанию – <b>0x8</b> .
<b>L_FIRST_SAMPLE_DELAY_E440</b>	0x4C	Переменная, задающая код задержки для первого отсчета при старте АЦП
<b>L_INTER_KADR_DELAY_E440</b>	0x4D	Переменная, задающая код межкадровой задержки при вводе данных с АЦП.
<b>L_DAC_SAMPLE_E440</b>	0x50	Переменная, задающая значение для однократного вывода на ЦАП
<b>L_DAC_ENABLED_E440</b>	0x51	Переменная, показывающая текущее состояние ЦАП (работает или нет).
<b>L_DAC_FIFO_BASE_ADDRESS_E440</b>	0x52	Текущий базовый адрес FIFO буфера ЦАП, который всегда должен быть равен <b>0x3000</b> .
<b>L_CUR_DAC_FIFO_</b>	0x54	Текущая длина FIFO буфера ЦАП. По умолчанию

<b>LENGTH_E440</b>		<b>L_CUR_DAC_FIFO_LENGTH_E440 = 0xFC0.</b>
<b>L_DAC_FIFO_LENGTH_E440</b>	0x55	Требуемая длина FIFO буфера ЦАП. По умолчанию <b>L_DAC_FIFO_LENGTH_E440 = 0xFC0.</b>
<b>L_FLASH_ENABLED_E440</b>	0x56	Флаг разрешения(1)/запрещения(0) записи в ППЗУ модуля
<b>L_FLASH_ADDRESS_E440</b>	0x57	Номер ячейки ППЗУ (от 0 до 63)
<b>L_FLASH_DATA_E440</b>	0x58	Данные в/из ППЗУ
<b>L_ENABLE_TTL_OUT_E440</b>	0x59	Данная переменная разрешает (0x1) либо запрещает (0x0) использование выходных цифровых линий (перевод их в третье состояние)
<b>L_TTL_OUT_E440</b>	0x5A	Слово (16 бит), в котором по-битово хранятся значения 16 <sup>ти</sup> выходных цифровых линий для их выставления по команде <b>C_TTL_OUT_E440</b> (см. <i>Таблице 8</i> ).
<b>L_TTL_IN_E440</b>	0x5B	Слово (16 бит), в котором после выполнения команды <b>C_TTL_IN_E440</b> (см. <i>Таблице 8</i> ) по-битово хранятся значения 16 <sup>ти</sup> входных цифровых линий.
<b>L_SCALE_E440</b>	0x60	Массив с 4 <sup>мя</sup> калибровочными коэффициентами, используемый для корректировки масштаба данных с АЦП. По умолчанию – { <b>0x8000, 0x8000, 0x8000, 0x8000</b> }
<b>L_ZERO_E440</b>	0x64	Массив с 4 <sup>мя</sup> калибровочными коэффициентами, используемый для корректировки смещения нуля данных с АЦП. По умолчанию – { <b>0x0, 0x0, 0x0, 0x0</b> }
<b>L_CONTROL_TABLE_E440</b>	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 <b>L_CONTROL_TABLE_LENGTH_E440</b> (см. выше). По умолчанию – { <b>0, 1, 2, 3, 4, 5, 6, 7</b> }
<b>L_DSP_INFO_STRUCTURE_E440</b>	0x200	Информация о драйвере DSP, включая версию.

### 5.3.2. Номера команд драйвера DSP

Фирменный *LBIOS* для модуля *E14-440* 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 и выше) приведен ниже в таблице.

Таблица 8. Номера команд штатного драйвера DSP.

Название команды	Номер команды	Назначение	Используемые переменные
<b>C_TEST_E440</b>	0	Проверка загрузки модуля и его работоспособности.	<b>L_TEST_LOAD_E440</b>
<b>C_ENABLE_FLASH_WRITE_E440</b>	1	Разрешение процедуры записи в пользовательское ППЗУ	<b>L_FLASH_ENABLED_E440</b>
<b>C_READ_FLASH_WORD_E440</b>	2	Чтение слова из ППЗУ	<b>L_FLASH_ADDRESS_E440,</b> <b>L_FLASH_DATA_E440</b>
<b>C_WRITE_FLASH_WORD_E440</b>	3	Запись слова в ППЗУ	<b>L_FLASH_ADDRESS_E440,</b> <b>L_FLASH_DATA_E440</b>
<b>C_START_ADC_E440</b>	4	Разрешение работы АЦП	-----
<b>C_STOP_ADC_E440</b>	5	Запрещение работы АЦП	-----
<b>C_ADC_KADR_E440</b>	6	Получение кадра отсчетов с АЦП	-----
<b>C_ADC_SAMPLE_E440</b>	7	Однократный ввод отсчета с АЦП для заданного канала	<b>L_ADC_SAMPLE_E440,</b> <b>L_ADC_CHANNEL_E440</b>
<b>C_START_DAC_E440</b>	8	Разрешение работы ЦАП	-----
<b>C_STOP_DAC_E440</b>	9	Запрещение работы ЦАП	-----
<b>C_DAC_SAMPLE_E440</b>	10	Однократный вывод отсчета на ЦАП	<b>L_DAC_SAMPLE_E440</b>
<b>C_ENABLE_TTL_OUT_E440</b>	11	Разрешение выходных цифровых линий	<b>L_ENABLE_TTL_OUT_E440</b>
<b>C_TTL_IN_E440</b>	12	Считывание состояния 16 <sup>ти</sup> внешних цифровых входных линий.	<b>L_TTL_IN_E440</b>
<b>C_TTL_OUT_E440</b>	13	Управление 16 <sup>тью</sup> внешними цифровыми выходными линиями.	<b>L_TTL_OUT_E440</b>

## 5.4. Функции общего характера

### 5.4.1. Получение версии библиотеки

<b>Формат:</b>	<b>DWORD</b>	<i>GetDllVersion(void)</i>						
<b>Назначение:</b>	<p>Данная функция является одной из <i>двух</i> экспортируемых из штатной библиотеки <code>Lusbapi</code> функцией. Она возвращает текущую версию используемой библиотеки. Формат номера версии следующий:</p> <table border="1"><thead><tr><th>Битовое поле</th><th>Назначение</th></tr></thead><tbody><tr><td>&lt;31..16&gt;</td><td>Старшее слово версии библиотеки</td></tr><tr><td>&lt;15..0&gt;</td><td>Младшее слово версии библиотеки</td></tr></tbody></table> <p>Рекомендованную последовательность вызовов интерфейсных функций смотри <a href="#">§ 4.1. "Общий подход к работе с интерфейсными функциями"</a>.</p>		Битовое поле	Назначение	<31..16>	Старшее слово версии библиотеки	<15..0>	Младшее слово версии библиотеки
Битовое поле	Назначение							
<31..16>	Старшее слово версии библиотеки							
<15..0>	Младшее слово версии библиотеки							
<b>Передаваемые параметры:</b>	нет							
<b>Возвращаемое значение:</b>	номер версии библиотеки <code>Lusbapi</code> .							

### 5.4.2. Получение указателя на интерфейс модуля

<b>Формат:</b>	<b>LPVOID</b>	<i>CreateInstance(const char *DeviceName)</i>	(версия 1.0)
	<b>LPVOID</b>	<i>CreateLInstance(PCHAR const DeviceName)</i>	(с версии 3.0)
<b>Назначение:</b>	<p>Данная функция должна обязательно вызываться в начале каждой пользовательской программы, работающей с модулями <i>E14-440</i>. Она является одной из <i>двух</i> экспортируемых из штатной библиотеки <code>Lusbapi</code> функцией и возвращает указатель на интерфейс для устройства с названием <i>DeviceName</i>. Все последующие интерфейсные функции штатной библиотеки вызываются именно через этот возвращаемый указатель.</p> <p>Рекомендованную последовательность вызовов интерфейсных функций смотри <a href="#">§ 4.1. "Общий подход к работе с интерфейсными функциями"</a>.</p>		
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li><i>DeviceName</i> – строка с названием устройства (для данного модуля это – “E440”).</li></ul>		
<b>Возвращаемое значение:</b>	В случае успеха — указатель на интерфейс, иначе — <b>NULL</b> .		



### 5.4.3. Завершение работы с интерфейсом модуля

<b>Формат:</b>	<b>bool</b> <b>BOOL</b>	<i>ReleaseLDevice(void)</i> <i>ReleaseLInstance(void)</i>	(версия 1.0) (с версии 3.0)
<b>Назначение:</b>	<p>Данная интерфейсная функция реализует корректное высвобождение интерфейсного указателя, проинициализированного с помощью интерфейсной функции <a href="#">CreateLInstance()</a>. Используется для аккуратного завершения сеанса работы с модулем, если предварительно удачно выполнялась функция <a href="#">CreateLInstance()</a>. <b>!!!Внимание!!!</b> Данная функция <b>должна обязательно</b> вызываться в Вашем приложении перед непосредственным выходом из него во избежание утечки ресурсов компьютера.</p> <p>Рекомендованную последовательность вызовов интерфейсных функций смотри <a href="#">§ 4.1. "Общий подход к работе с интерфейсными функциями"</a>.</p>		
<b>Передаваемые параметры:</b>	нет.		
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.		

### 5.4.4. Инициализация доступа к модулю

<b>Формат:</b>	<b>bool</b> <b>BOOL</b>	<i>InitLDevice(WORD VirtualSlot)</i> <i>OpenLDevice(WORD VirtualSlot)</i>	(версия 1.0) (с версии 2.0)
<b>Назначение:</b>	<p>С программной точки зрения, не вдаваясь в излишние тонкости, подсоединенный к компьютеру модуль <b>E14-440</b> можно рассматривать как устройство, подключённое к некоему виртуальному слоту с сугубо индивидуальным номером. Основное назначение данной интерфейсной функции как раз в том и состоит, чтобы определить, что именно модуль <b>E14-440</b> находится в заданном виртуальном слоте. Если функция <i>OpenLDevice()</i> успешно выполнялась для заданного виртуального слота, то можно переходить непосредственно к загрузке модуля и его последующему управлению с помощью соответствующих интерфейсных функций библиотеки <i>Lusbapi</i>.</p> <p><i>InitLDevice()</i> является устаревшим названием данной функции, хотя библиотекой по-прежнему поддерживается, кроме <b>Delphi</b>.</p> <p>Рекомендованную последовательность вызовов интерфейсных функций смотри <a href="#">§ 4.1. "Общий подход к работе с интерфейсными функциями"</a>.</p>		
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>VirtualSlot</i> – номер виртуального слота, к которому, как предполагается, подключен модуль <b>E14-440</b>.</li></ul>		
<b>Возвращаемое значение:</b>	<i>TRUE</i> – модуль <b>E14-440</b> находится в выбранном виртуальном слоте и можно переходить непосредственно к загрузке модуля; <i>FALSE</i> – устройства типа модуль <b>E14-440</b> в выбранном виртуальном слоте нет. Следует попробовать другой номер виртуального слота.		

### 5.4.5. Завершение доступа к модулю

<b>Формат:</b>	<b>BOOL</b>	<i>CloseLDevice (void)</i>
<b>Назначение:</b>		
<p>Данная интерфейсная функция прерывает всякое взаимодействие с <i>текущим</i> виртуальным слотом, к которому подключён модуль. При этом данный виртуальный слот аккуратно закрывается и выполняется освобождение связанных с ним ресурсов <i>Windows</i>. После её применения всякий доступ к модулю <i>E14-440</i> становится невозможным. Для возобновления нормального доступа к устройству необходимо вновь воспользоваться интерфейсной функцией <i>OpenLDevice()</i>. Таким образом, эта функция, по своей сути, противоположна интерфейсной функции <i>OpenLDevice()</i>. Фактически данная функция используется в таких интерфейсных функциях как <i>OpenLDevice()</i> и <i>ReleaseInstance()</i>.</p>		
<b>Передаваемые параметры:</b> нет.		
<b>Возвращаемое значение:</b>		
<p><i>TRUE</i> – функция успешно выполнена;  <i>FALSE</i> – функция выполнена с ошибкой.</p>		

### 5.4.6. Загрузка модуля

<b>Формат:</b>	<b>bool</b>	<i>LOAD_LBIOS(PCHAR FileName)</i>	(версия 1.0)
	<b>bool</b>	<i>LOAD_LBIOS(PCHAR FileName = NULL)</i>	(с версии 2.0)
	<b>BOOL</b>	<i>LOAD_MODULE(PCHAR const FileName = NULL)</i>	(с версии 3.0)
<b>Назначение:</b>			
<p>Данная интерфейсная функция выполняет операцию загрузки драйвера (штатного <i>LBIOS</i> или Вашего) в DSP модуля. Файл <i>FileName</i> с кодом драйвера DSP должен находиться в текущей директории приложения.</p> <p>В библиотеке <i>Lusbapi</i> с версии 2.0 появилась дополнительная возможность загружать <i>LBIOS</i>, содержимое которого хранится в самом теле библиотеки в виде соответствующего ресурса. Для этого достаточно параметр <i>FileName</i> задать в виде <b>NULL</b>. <b>NULL</b> является также значением по умолчанию для параметра <i>FileName</i>.</p> <p>Модуль <i>E14-440</i>, начиная с ревизии <b>F</b>, может храниться штатный <i>LBIOS</i> в <i>загрузочном ППЗУ</i>. Этот <i>LBIOS</i> может либо по запросу пользователя, либо автоматически после подачи питания загружаться в DSP модуля. В библиотеке <i>Lusbapi</i> с версии 3.3 была введена возможность загрузки <i>LBIOS</i> по запросу пользователя посредством передачи в функцию <i>LOAD_MODULE()</i> предопределённой строки "flash".</p> <p>Рекомендованную последовательность вызовов интерфейсных функций смотри § 4.1. "Общий подход к работе с интерфейсными функциями".</p>			
<b>Передаваемые параметры:</b>			
<ul style="list-style-type: none"> <li><i>FileName</i> – строка с названием файла, содержащим код загружаемой управляющей программы. Например, для штатного драйвера DSP это будет строка "E440.BIO". Начиная с версии 2.0 библиотеки <i>Lusbapi</i>, если данный параметр задан как <b>NULL</b>, то загрузка модуля будет осуществляться тем <i>LBIOS</i>, который находится в виде ресурса в теле штатной библиотеки. Если этот параметр содержит предопределённую строку "flash", то загрузка DSP будет производиться из <i>загрузочного ППЗУ</i> (для модуля ревизии <b>F</b> и выше)</li> </ul>			
<b>Возвращаемое значение:</b>			
<p><i>TRUE</i> – функция успешно выполнена;  <i>FALSE</i> – функция выполнена с ошибкой.</p>			

#### 5.4.7. Проверка загрузки модуля

<b>Формат:</b>	<b>bool</b> <b>BOOL</b>	<i>MODULE_TEST(void)</i> <i>TEST_MODULE(void)</i>	(версия 1.0) (с версии 3.0)
<b>Назначение:</b>	Данная интерфейсная функция проверяет правильность загрузки модуля и его работоспособность. <b>!!!Внимание!!!</b> Данная функция работает надлежащим образом <b>ТОЛЬКО</b> после выполнения интерфейсной функции <i>LOAD_MODULE()</i> . Рекомендованную последовательность вызовов интерфейсных функций смотри <a href="#">§ 4.1. "Общий подход к работе с интерфейсными функциями"</a> .		
<b>Передаваемые параметры:</b>	нет		
<b>Возвращаемое значение:</b>	<i>TRUE</i> – драйвер DSP успешно загружен и функционирует надлежащим образом, <i>FALSE</i> – произошла ошибка загрузки или функционирования драйвера DSP.		

#### 5.4.8. Получение названия модуля

<b>Формат:</b>	<b>BOOL</b>	<i>GetModuleName(PCHAR const ModuleName)</i>
<b>Назначение:</b>	Данная вспомогательная интерфейсная функция позволяет получить название подключенного к слоту модуля. Массив под название модуля <i>ModuleName</i> (не менее 6 символов плюс признак конца строки ‘\0’, т.е. нулевой байт) должен быть заранее определен. Рекомендованную последовательность вызовов интерфейсных функций смотри <a href="#">§ 4.1. "Общий подход к работе с интерфейсными функциями"</a> .	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>ModuleName</i> – возвращается строка, не менее 6 символов, с названием модуля (в нашем случае это должна быть строка "E440").</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

#### 5.4.9. Получение скорости работы модуля

<b>Формат:</b>	<b>BOOL</b>	<i>GetUsbSpeed(BYTE * const UsbSpeed)</i>
<b>Назначение:</b>	Данная функция позволяет определить, на какой скорости шина <b>USB</b> работает с модулем.	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>UsbSpeed</i> – возвращаемое значение этой переменной может принимать следующее:<ul style="list-style-type: none"><li>✓ 0 – модуль работает с <b>USB</b> шиной в режиме <i>Full-Speed Mode</i> (12 Мбит/с)</li><li>✓ 1 – модуль работает с <b>USB</b> шиной в режиме <i>High-Speed Mode</i> (480 Мбит/с).</li></ul></li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

### 5.4.10. Сброс модуля

<b>Формат:</b>	<b>bool</b>	<i>DSP_RESET(void)</i>	(версия 1.0)
	<b>BOOL</b>	<i>RESET_MODULE(BYTE ResetFlag = INIT_E440)</i>	(с версии 3.0)

#### Назначение:

Данная интерфейсная функция осуществляет сброс модуля. Эта процедура используется при перезагрузке драйвера DSP или для полной остановки работы модуля. Сброс может быть произведён различными способами в зависимости от значения переменной *ResetFlag*, а также можно использовать *константы сброса модуля*.

Значение	Константа	Назначение
0	<b>INIT_E440</b>	Модуль подвергается полной переинициализации. После чего DSP модуля снова готов к загрузке.
1	<b>RESET_E440</b>	Это константа действует только на модуль <i>E14-440</i> с ревизией 'E' и выше. При этом модуль подвергается процедуре полного сброса. После чего DSP находится в состоянии сброса, вторичное питание отключено. Это состояние с пониженным энергопотреблением.
2	<b>INVALID_RESET_TYPE_E440</b>	Неправильный тип сброса модуля.

Необходимо помнить, что после выполнения данной функции работа DSP модуля полностью останавливается. В дальнейшем для приведения модуля в 'чувство' требуется перезагрузить драйвер DSP, используя штатную функцию *LOAD\_MODULE()*.

#### Передаваемые параметры:

- *ResetFlag* – режим сброса модуля.

**Возвращаемое значение:** *TRUE* – функция успешно выполнена;  
*FALSE* – функция выполнена с ошибкой.

### 5.4.11. Передача номер команды в драйвер DSP

<b>Формат:</b>	<b>BOOL</b>	<i>SEND_COMMAND(WORD Command)</i>
----------------	-------------	-----------------------------------

#### Назначение:

Данная интерфейсная функция записывает в predetermined 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 успешного выполнения интерфейсных функций *LOAD\_MODULE()* и *TEST\_MODULE()*.

#### Передаваемые параметры:

- *Command* – номер команды, передаваемый в драйвер DSP.

**Возвращаемое значение:** *TRUE* – функция успешно выполнена;  
*FALSE* – функция выполнена с ошибкой.

### 5.4.12. Получение дескриптора устройства

<b>Формат:</b>	<b>HANDLE</b>	<b><i>GetModuleHandle(void)</i></b>
<b>Назначение:</b>	Данная функция позволяет получить дескриптор ( <i>handle</i> ) используемого модуля <i>E14-440</i> .	
<b>Передаваемые параметры:</b>	нет	
<b>Возвращаемое значение:</b>	В случае успеха – дескриптор модуля <i>E14-440</i> ; в противном случае – <b>INVALID_HANDLE_VALUE</b> .	

### 5.4.13. Получение описания ошибок выполнения функций

<b>Формат:</b>	<b>int</b>	<b><i>GetLastErrorString(LPTSTR lpBuffer, DWORD nSize)</i></b> (версия 1.0)
	<b>BOOL</b>	<b><i>GetLastErrorInfo(LAST_ERROR_INFO_LUSBAPI * const GetLastErrorInfo)</i></b> (с версии 3.0)
<b>Назначение:</b>	Если в процессе работы с библиотекой <i>Lusbapi</i> какая-нибудь интерфейсная функция штатной библиотеки вернула ошибку, то <b>ТОЛЬКО</b> непосредственно после этого с помощью вызова данной интерфейсной функции можно получить краткое толкование произошедшего сбоя. <b>!!!Внимание!!!</b> Данная интерфейсная функция не выполняет классификацию ошибок для интерфейсных функций <i>ReadData()</i> и <i>WriteData()</i> . Т.к. эти функции фактически является слепком со стандартных <i>Windows API</i> функций <i>ReadFile()</i> и <i>WriteFile()</i> , то для выявления ошибок следует пользоваться классификацией ошибок, присущей системе <i>Windows</i> .	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>LastErrorInfo</i> – указатель на структуру типа <i>LAST_ERROR_INFO_LUSBAPI</i>, в которой возвращается краткое описание и номер последней ошибки.</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

## 5.5. 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*.

### 5.5.1. Чтение слова из памяти данных DSP

<b>Формат:</b>	<b>BOOL</b>	<b><i>GET_DM_WORD</i></b> ( <i>WORD Address, SHORT * const Data</i> )
<b>Назначение:</b>	Данная функция считывает значение слова, находящееся по адресу <i>Address</i> в памяти данных DSP модуля.	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>Address</i> – адрес ячейки в памяти данных DSP, значение которой необходимо считать;</li><li>• <i>Data</i> – указатель на переменную, куда функция положит считанное 16<sup>ти</sup> битное слово.</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

### 5.5.2. Чтение слова из памяти программ DSP

<b>Формат:</b>	<b>BOOL</b>	<b><i>GET_PM_WORD</i></b> ( <i>WORD Address, long * const Data</i> )
<b>Назначение:</b>	Данная функция считывает значение слова, находящееся по адресу <i>Address</i> в памяти программ DSP модуля.	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>Address</i> – адрес ячейки в памяти программ DSP, значение которой необходимо считать;</li><li>• <i>Data</i> – указатель на переменную, куда функция положит считанное 24<sup>х</sup> битное слово.</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

### 5.5.3. Запись слова в память данных DSP

<b>Формат:</b>	<b>BOOL</b>	<b><i>PUT_DM_WORD</i></b> ( <i>WORD Address, SHORT Data</i> )
<b>Назначение:</b>	Данная функция записывает значение <i>Data</i> в ячейку с адресом <i>Address</i> в памяти данных DSP модуля.	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>Address</i> – адрес ячейки в памяти данных DSP, куда необходимо записать значение <i>Data</i>;</li><li>• <i>Data</i> – значение записываемого 16<sup>ти</sup> битного слова.</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

#### 5.5.4. Запись слова в память программ DSP

<b>Формат:</b> <b>BOOL</b> <i>PUT_PM_WORD</i> ( <i>WORD Address, long Data</i> )
<b>Назначение:</b> Данная функция записывает значение <i>Data</i> в ячейку с адресом <i>Address</i> в памяти программ DSP модуля.
<b>Передаваемые параметры:</b> <ul style="list-style-type: none"><li>• <i>Address</i> – адрес ячейки в памяти программ DSP, куда записывается значение <i>Data</i>;</li><li>• <i>Data</i> – значение записываемого <math>24^x</math> битного слова.</li></ul>
<b>Возвращаемое значение:</b> <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.

#### 5.5.5. Чтение массива слов из памяти данных DSP

<b>Формат:</b> <b>BOOL</b> <i>GET_DM_ARRAY</i> ( <i>WORD BaseAddress, WORD NPoints, SHORT * const Buffer</i> )
<b>Назначение:</b> Данная функция считывает массив слов длиной <i>NPoints</i> в буфер <i>Buffer</i> , начиная с адреса ячейки <i>BaseAddress</i> в памяти данных DSP модуля. Буфер <i>Buffer</i> надлежащей длины необходимо заранее определить.
<b>Передаваемые параметры:</b> <ul style="list-style-type: none"><li>• <i>BaseAddress</i> – стартовый адрес в памяти данных DSP, начиная с которого производится чтение массива;</li><li>• <i>NPoints</i> – длина считываемого массива;</li><li>• <i>Buffer</i> – указатель на буфер, в который передаются считываемые значения.</li></ul>
<b>Возвращаемое значение:</b> <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.

#### 5.5.6. Чтение массива слов из памяти программ DSP

<b>Формат:</b> <b>BOOL</b> <i>GET_PM_ARRAY</i> ( <i>WORD BaseAddress, WORD NPoints, long * const Buffer</i> )
<b>Назначение:</b> Данная функция считывает массив слов длиной <i>NPoints</i> в буфер <i>Buffer</i> , начиная с адреса ячейки <i>BaseAddress</i> в памяти программ DSP модуля. Буфер <i>Buffer</i> надлежащей длины необходимо заранее определить. При использовании этой функции следует помнить, что одно слово памяти программ DSP является $24^x$ битным.
<b>Передаваемые параметры:</b> <ul style="list-style-type: none"><li>• <i>BaseAddress</i> – стартовый адрес в памяти программ DSP, начиная с которого производится чтение массива;</li><li>• <i>NPoints</i> – число считываемых <math>24^x</math> битных слов;</li><li>• <i>Buffer</i> – указатель на буфер, в который передаются считываемые значения.</li></ul>
<b>Возвращаемое значение:</b> <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.

### 5.5.7. Запись массива слов в память данных DSP

<b>Формат:</b> <code>BOOL PUT_DM_ARRAY(WORD BaseAddress, WORD Npoints, SHORT * const Buffer)</code>
<b>Назначение:</b> Данная функция записывает массив слов длиной <i>NPoints</i> из буфера <i>Buffer</i> в память данных DSP модуля, начиная с адреса <i>BaseAddress</i> .
<b>Передаваемые параметры:</b> <ul style="list-style-type: none"><li>• <i>BaseAddress</i> – стартовый адрес в памяти данных DSP, начиная с которого производится запись массива;</li><li>• <i>NPoints</i> – длина записываемого массива;</li><li>• <i>Buffer</i> – указатель на буфер, из которого идет запись.</li></ul>
<b>Возвращаемое значение:</b> <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.

### 5.5.8. Запись массива слов в память программ DSP

<b>Формат:</b> <code>BOOL PUT_PM_ARRAY(WORD BaseAddress, WORD NPoints, long * const Buffer)</code>
<b>Назначение:</b> Данная функция записывает массив слов длиной <i>NPoints</i> из буфера <i>Buffer</i> в память программ DSP модуля, начиная с адреса <i>BaseAddress</i> . При использовании этой функции следует помнить, что одно слово памяти программ DSP является $24^x$ битным.
<b>Передаваемые параметры:</b> <ul style="list-style-type: none"><li>• <i>BaseAddress</i> – стартовый адрес в памяти программ DSP, начиная с которого производится запись массива;</li><li>• <i>NPoints</i> – число записываемых <math>24^x</math> битных слов;</li><li>• <i>Buffer</i> – указатель на буфер, из которого идет запись.</li></ul>
<b>Возвращаемое значение:</b> <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.



### 5.5.9. Чтение переменной драйвера DSP

<b>Формат:</b>	<b>BOOL</b>	<b>GET_LBIOS_WORD</b> (WORD Address, SHORT * const Data)
<b>Назначение:</b>	Данная функция осуществляет аккуратное считывание 16 <sup>™</sup> битной переменной штатного LBIOS, расположенной по адресу Address в 24 <sup>х</sup> битной памяти программ DSP модуля (см. § 5.3.1. "Переменные драйвера DSP").	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• Address – адрес ячейки переменной LBIOS в памяти программ DSP, значение которой необходимо считать;</li><li>• Data – указатель на переменную, куда функция положит считанное 16<sup>™</sup> битное значение переменной штатного LBIOS.</li></ul>	
<b>Возвращаемое значение:</b>	TRUE – функция успешно выполнена; FALSE – функция выполнена с ошибкой.	

### 5.5.10. Запись переменной драйвера DSP

<b>Формат:</b>	<b>BOOL</b>	<b>PUT_LBIOS_WORD</b> (WORD Address, SHORT Data)
<b>Назначение:</b>	Данная функция осуществляет аккуратную запись 16 <sup>™</sup> битного значения Data в переменную штатного LBIOS, расположенную по адресу Address в 24 <sup>х</sup> битной памяти программ DSP модуля (см. § 5.3.1. "Переменные драйвера DSP").	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• Address – адрес ячейки переменной LBIOS в памяти программ DSP, куда необходимо записать значение Data;</li><li>• Data – значение записываемого 16<sup>™</sup> битного значения переменной штатного LBIOS.</li></ul>	
<b>Возвращаемое значение:</b>	TRUE – функция успешно выполнена; FALSE – функция выполнена с ошибкой.	

## 5.6. Функции для работы с АЦП

Интерфейсные функции штатной библиотеки `Lusbapi` позволяют реализовывать разнообразные алгоритмы работы модуля *E14-440* с АЦП *независимо* от состояния ЦАП. Но основным режимом работы модуля является, конечно же, непрерывный *поточный* сбор данных с АЦП. А вообще модуль, с точки зрения состояния АЦП, может находиться как бы в двух режимах:

1. режим “покоя”;
2. поточный, т.е. *перманентный*, сбор данных с АЦП.

Функция `START_ADC()` позволяет переводить модуль во второе из этих состояний, а `STOP_ADC()` — в первое. Прежде чем запустить модуль на сбор данных с АЦП, необходимо в него передать все требуемые параметры функционирования АЦП: тип синхронизации, частота работы АЦП, длина и базовый адрес FIFO буфера АЦП, управляющую таблицу и т.д. Эту операцию можно осуществить с помощью интерфейсной функции `SET_ADC_PARS()`. После этого, в принципе, можно запускать модуль на сбор данных, выполнив функцию `START_ADC()`. Для извлечения из модуля уже полученных с АЦП данных следует пользоваться функцией `ReadData()`. При этом функция `ReadData()` может выполняться как в *синхронном*, так и в *асинхронном* режимах. 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 директории `\E14-440\Example\`.

### 5.6.1. Корректировка данных АЦП

Схемотехника и использованные компоненты обеспечивают линейность передаточной характеристики аналогового тракта АЦП модуля *E14-440*. Но каждый экземпляр модуля обладает своими, сугубо индивидуальными, значениями смещения нуля и неточности в передаче масштаба отсчёта АЦП. Это происходит вследствие того, что на модуле отсутствуют какие-либо подстроечные резисторы, что в целом позволяет улучшить шумовые характеристики модуля и увеличивает их надежность.

Пользователь может возложить всю работу по корректировке поступающих с модуля данных АЦП либо на своё приложение в РС, либо потребовать, чтобы сам модуль осуществлял всю эту процедуру на уровне драйвера DSP. В последнем случае это приводит к тому, что показания АЦП, поступающие из модуля в РС, имеют уже полностью откорректированный вид. И поэтому на уровне приложения в РС нет никакой необходимости в реализации всей этой нудной задачи по корректировке данных АЦП.

В качестве корректировочных коэффициентов вполне можно использовать как *штатные*, так и свои собственные, т.е. *пользовательские*.

*Пользовательские* корректировочные коэффициенты могут быть использованы, например, для целей компенсации погрешностей целого измерительного тракта какого-нибудь стенда, составной частью которого вполне может служить модуль *E14-440*. При этом вся ответственность за формирование и корректное применение *пользовательских* корректировочных коэффициентов полностью ложится на плечи конечного пользователя.

*Штатные* корректировочные коэффициенты располагаются в полях `Adc.OffsetCalibration[]` и `Adc.ScaleCalibration[]` структуры служебной информации `MODULE_DESCRIPTION_E440`. Вся служебная информация совместно с корректировочными коэффициентами записывается в модуль на этапе при его наладке в **ЗАО “Л-Кард”**. Поля коэффициентов представляют собой массивы типа `double`. Для модуля *E14-440* в каждом из этих массивов имеют смысл только первые `ADC_CALIBR_COEFS_QUANTITY_E440` элементов. Массив `Adc.OffsetCalibration` содержит коэффициенты корректировки смещение нуля, а массив `Adc.ScaleCalibration` – масштаба. Если в *логическом номере канала АЦП* индекс коэффициента усиления равен *i*, то корректировочные коэффициенты этого логического канала могут быть получены следующим образом:

- смещение: `Adc.OffsetCalibration[i];`
- масштаб: `Adc.ScaleCalibration[i].`

Поле *IsCorrectionEnabled* структуры [ADC\\_PARS\\_E440](#) определяет необходимость проведения автоматической корректировки данных АЦП на уровне драйвера DSP. Если же такая корректировка нужна, то в полях *AdcOffsetCoefs* и *AdcScaleCoefs* структуры [ADC\\_PARS\\_E440](#) требуется передать в DSP модуля все корректировочные коэффициенты.

В общем виде при использовании штатных коэффициентов корректировка отсчётов АЦП выполняется по следующей формуле:

$$Y = (X+A)*B,$$

где: X – некорректированные данные АЦП [в отсчётах АЦП],

Y – скорректированные данные АЦП [в отсчётах АЦП],

A – коэффициент смещения нуля [в отсчётах АЦП],

B – коэффициент масштаба [безразмерный].

Например, пусть со второго канала АЦП, настроенного на входной диапазон  $\pm 2.5$  В (индекс коэффициента усиления равен 1 или [ADC\\_INPUT\\_RANGE\\_2500mV\\_E440](#)), получены следующие данные: X1 = 1000, X2 = -1000 и X3 = 0. Тогда коэффициенты и скорректированные данные можно представить так:

- A = `Adc.OffsetCalibration[1]`;
- B = `Adc.ScaleCalibration[1]`;
- Y1 = (A+1000)\*B, Y2 = (A-1000)\*B, Y3 = A\*B.

### 5.6.2. Запуск сбора данных АЦП

<b>Формат:</b>	<b>BOOL</b>	<b>START_ADC(void)</b>
<b>Назначение:</b>	<p>Данная функция запускает модуль <i>E14-440</i> на непрерывный <i>поточковый</i> сбор данных с АЦП. Перед любым запуском сбора данных <b>настоятельно</b> рекомендуется выполнять функцию <a href="#">STOP_ADC()</a>. Перед началом сбора можно установить требуемые параметры работы АЦП, которые передаются в модуль с помощью интерфейсной функции <a href="#">SET_ADC_PARS()</a>. Извлечение из модуля уже собранных с АЦП данных можно осуществлять с помощью интерфейсной функции <a href="#">ReadData()</a>.</p>	
<b>Передаваемые параметры:</b>	нет	
<b>Возвращаемое значение:</b>	<p><i>TRUE</i> – функция успешно выполнена;  <i>FALSE</i> – функция выполнена с ошибкой.</p>	

### 5.6.3. Останов сбора данных АЦП

<b>Формат:</b>	<b>BOOL</b>	<b>STOP_ADC(void)</b>
<b>Назначение:</b>	<p>Данная функция останавливает в модуле <i>E14-440</i> механизм сбора данных с АЦП. Попутно эта функция как бы ‘<i>приводит в чувство</i>’ основную программу микроконтроллера модуля, а также сбрасывает используемый канал передачи данных по <b>USB</b> шине. Поэтому <b>настоятельно</b> рекомендуется применять эту функцию <b>перед</b> каждым запуском сбора данных функцией <a href="#">START_ADC()</a>.</p>	
<b>Передаваемые параметры:</b>	нет	
<b>Возвращаемое значение:</b>	<p><i>TRUE</i> – функция успешно выполнена;  <i>FALSE</i> – функция выполнена с ошибкой.</p>	

### 5.6.4. Установка параметров работы АЦП

**Формат:** **bool** *FILL\_ADC\_PARS(ADC\_PARS\_E440 \*am)* (версия 1.0)  
**BOOL** *SET\_ADC\_PARS(ADC\_PARS\_E440 \*const AdcPars)* (с версии 3.0)

#### Назначение:

Данная функция передает в *E14-440* всю необходимую информацию, которая используется модулем для организации заданного режима сбора данных с АЦП. Вся нужную информацию описываемая интерфейсная функция извлекает из полей передаваемой структуры типа *ADC\_PARS\_E440*. Собственно, использование модулем **именно** этой переданной информации начинается **только** после выполнения интерфейсной функции *START\_ADC()*. Также крайне не рекомендуется вызывать эту функцию собственно в процессе сбора данных. Её следует применять **только** после выполнения функции *STOP\_ADC()*.

Описание структуры *ADC\_PARS\_E440* приведено ранее в § 5.2.2. "Структура *ADC\_PARS\_E440*", а назначение отдельных ее полей описано ниже.

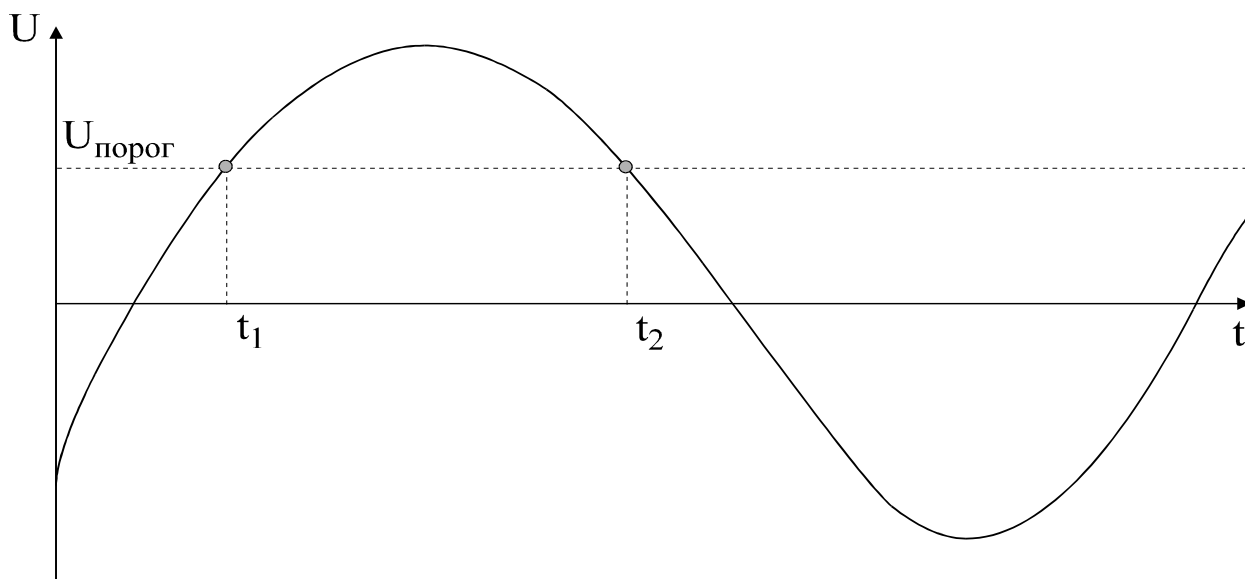
- Поле *AdcPars->IsCorrectionEnabled*. Запись. Данное поле определяет возможность драйвера DSP осуществлять автоматическую корректировку получаемых с АЦП данных. Т.е. из модуля приложение получает уже откорректированные данные с АЦП, если установить поле равным *TRUE*. При использовании корректировки собственно сами корректировочные коэффициенты должны находиться в поле *AdcPars->AdcOffsetCoefs* и *AdcPars->AdcScaleCoefs* (см. ниже).
- Поле *AdcPars->AdcClockSource*. Запись—Чтение. Данное поле задаёт источник *тактовых* импульсов для работы АЦП модуля. Это поле может принимать одно из четырёх значений от 0 до 2, также можно пользоваться *константами источника тактовых импульсов АЦП*. Смысловая нагрузка значений этого поля представлена в таблице ниже:

Значение	Константа	Назначение
0	<i>INT_ADC_CLOCK_E440</i>	<b>Внутренние тактовые импульсы АЦП без трансляции.</b> В этом режиме тактовые импульсы для работы АЦП аппаратно генерируются посредством DSP модуля без их трансляции на внешние разъёмы. При этом линия <i>IN16</i> может без ограничения использоваться как обычная входная цифровая линия.
1	<i>INT_ADC_CLOCK_WITH_TRANS_E440</i>	<b>Внутренние тактовые импульсы АЦП с трансляции.</b> В этом режиме тактовые импульсы для работы АЦП аппаратно генерируются посредством DSP модуля и дополнительно транслируются на линию <i>IN16</i> внешнего цифрового разъёма. При этом линия <i>IN16</i> может использоваться <b>только</b> как выходная линия, транслирующая тактовые импульсы АЦП.
2	<i>EXT_ADC_CLOCK_E440</i>	<b>Внешние тактовые импульсы АЦП.</b> В этом режиме используется внешний источник тактовых импульсов, которые подключается к линии <i>IN16</i> внешнего цифрового разъёма модуля. При этом линия <i>IN16</i> может использоваться <b>только</b> как входная цифровая линия для подачи внешних тактовых импульсов АЦП.
3	<i>INVALID_ADC_CLOCK_E440</i>	Неправильный тип источника тактовых импульсов АЦП.

- Поле `AdcPars->InputMode`. Запись. Значение данного поля может задавать различные виды синхронизации ввода данных с АЦП. Это поле может принимать одно из четырёх значений от 0 до 3, также можно пользоваться *константами синхронизации ввода*. Смысловая нагрузка значений этого поля представлена в таблице ниже:

Значение	Константа	Назначение
0	<code>NO_SYNC_E440</code>	<i>Отсутствие синхронизации ввода.</i> Сбор данных с АЦП модуль начинает непосредственно после выполнения функции <code>START_ADC()</code> .
1	<code>TTL_START_SYNC_E440</code>	<i>Цифровая синхронизация начала ввода.</i> Непосредственно после выполнения функции <code>START_ADC()</code> модуль переходит в состояние ожидания прихода отрицательного перепада ( $\overline{\square}$ ) у TTL-совместимого одиночного импульса на входе <code>TRIG</code> аналогового разъёма <i>DRB-37M</i> . Длительность этого синхроимпульса должна быть не менее 50 нс. Только после этого модуль приступает к сбору данных.
2	<code>TTL_KADR_SYNC_E440</code>	<i>Цифровая покадровая синхронизация ввода.</i> Непосредственно после выполнения функции <code>START_ADC()</code> модуль переходит в состояние ожидания прихода отрицательного перепада ( $\overline{\square}$ ) у TTL-совместимого одиночного импульса на входе <code>TRIG</code> аналогового разъёма <i>DRB-37M</i> . Длительность этого синхроимпульса должна быть не менее 50 нс. После прихода синхроимпульса (см. выше) модуль собирает отсчеты <b>только</b> одного кадра отсчетов. Во время ввода кадра данных вся активность на линии <code>TRIG</code> игнорируется. Только после окончания сбора кадра ожидается приход следующего импульса синхронизации и т.д.
3	<code>ANALOG_SYNC_E440</code>	<i>Аналоговая синхронизация начала ввода.</i> Непосредственно после выполнения функции <code>START_ADC()</code> модуль переходит в состояние ожидания вплоть до момента выполнения условий аналоговой синхронизации. Модуль начинает собирать данные с АЦП <b>только</b> после выполнения заданных соотношений между получаемым значением с заданного аналогового синхроканала и заданным пороговым значением. При этом условия аналоговой синхронизации ввода данных задаются через посредство полей <code>AdcPars-&gt;SynchroAdType</code> , <code>AdcPars-&gt;SynchroAdMode</code> , <code>AdcPars-&gt;SynchroAdChannel</code> и <code>AdcPars-&gt;SynchroAdPorog</code> .
4	<code>INVALID_SYNC_E440</code>	Неправильная синхронизация ввода данных.

- Поля `AdcPars->SynchroAdType`, `AdcPars->SynchroAdMode`, `AdcPars->SynchroAdChannel`, `AdcPars->SynchroAdPorog`. Запись. Эти поля 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** (`AdcPars->SynchroAdType`  $\neq 0$ ) ‘*снизу-вверх*’ (`AdcPars->SynchroAdMode` = 0) при пороговом значении равном `AdcPars->SynchroAdPorog` =  $U_{\text{порог}}$  (в кодах АЦП), то модуль начнет собирать данные только после наступления момента времени  $t_1$ . Тогда, когда уровень входного сигнала на логическом синхроканале `AdcPars->SynchroAdChannel` пересечет пороговую линию в направлении снизу вверх. Аналогично если задан **переход** ‘*сверху-вниз*’ (`AdcPars->SynchroAdMode`  $\neq 0$ ), то старт начало сбора наступит в момент времени  $t_2$ , когда уровень входного сигнала на синхроканале пересечет пороговую линию в направлении сверху вниз. Если же аналоговая синхронизация задается по **уровню** (`AdcPars->SynchroAdType` = 0), то сбор модулем данных начнется в тот момент, когда уровень входного сигнала на синхроканале окажется либо выше (`AdcPars->SynchroAdMode` = 0), либо ниже (`AdcPars->SynchroAdMode`  $\neq 0$ ) пороговой линии  $U_{\text{порог}}$ .

Все вышесказанное относительно аналоговой синхронизации ввода данных можно кратко свести к следующему:

- поле `AdcPars->SynchroAdType` может принимать следующие значения:
  - ✓ `AdcPars->SynchroAdType` = 0 — аналоговая синхронизация по **уровню**,
  - ✓ `AdcPars->SynchroAdType`  $\neq 0$  — аналоговая синхронизация по **переходу**;
- поле `AdcPars->SynchroAdMode` может принимать следующие значения:
  - ✓ при `AdcPars->SynchroAdMode` = 0:
    - аналоговая синхронизация по **уровню** — ‘*выше*’,
    - аналоговая синхронизация по **переходу** — ‘*снизу-вверх*’,
  - ✓ при `AdcPars->SynchroAdMode`  $\neq 0$ :
    - аналоговая синхронизация по **уровню** — ‘*ниже*’,
    - аналоговая синхронизация по **переходу** — ‘*сверху-вниз*’;
- поле `AdcPars->SynchroAdChannel` – логический номер синхроканала АЦП для аналоговой синхронизации;
- поле `AdcPars->SynchroAdPorog` – пороговое значение для аналоговой синхронизации (в кодах АЦП);

- Поле `AdcPars->ChannelsQuantity`. Запись–Чтение. Данное поле задаёт количество активных *логических каналов* в управляющей таблице **ControlTable**. Т.е. модуль при сборе данных с АЦП будет использоваться первые `AdcPars->ChannelsQuantity` элементов массива `AdcPars->ControlTable`. Предельным значением для данного поля является величина [MAX\\_CONTROL\\_TABLE\\_LENGTH\\_E440](#). Если переданное в функцию значение превышает указанное предельное значение, то функция автоматически выполняет необходимую корректировку и по завершении оной в поле `AdcPars->ChannelsQuantity` будет находиться реально установленное количество активных *логических каналов*.
- Поле `AdcPars->ControlTable[]`. Запись. Данное поле является массивом типа *WORD*. Оно задаёт управляющую таблицу **ControlTable**. Т.е. тот массив *логических каналов*, который будет использоваться модулем при работе с АЦП для задания циклической последовательности отсчётов с входных аналоговых каналов.
- Поля `AdcPars->AdcRate` и `AdcPars->InterKadrDelay`. Запись–Чтение. Эти поля при входе в данную функцию должны содержать требуемые временные параметры сбора данных: частота работы АЦП **AdcRate** (обратная величина *межканальной* задержки) и межкадровая задержка **InterKadrDelay**. При этом **AdcRate** задаётся в *кГц*, а **InterKadrDelay** – в *мс*. После выполнения функции `SET_ADC_PARS()` 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 образуют некую сетку частот. Так, в общем виде, частота работы АЦП определяется по следующей формуле:  $AdcRate = F_{clockout} / (2 * (N + 1))$ , где  $F_{clockout}$  – тактовая частота установленного на модуле DSP, равная 48000 кГц, а  $N$  – целое число. Поэтому данная функция просто вычисляет ближайшую к задаваемой дискретную величину **AdcRate**, передает её в модуль в виде целого 16<sup>ти</sup> битного числа  $N$ , а также возвращает её значение в поле `AdcPars->AdcRate`. Все то же самое верно и для межкадровой задержки **InterKadrDelay**, 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 `AdcPars->AdcRate = 0`, то `SET_ADC_PARS()` установит и возвратит минимально возможную величину для данного поля, т.е. 0.366 кГц. Аналогично: если задать `AdcPars->InterKadrDelay = 0`, то данная функция установит и возвратит минимально возможную межкадровую задержку, т.е.  $1/AdcPars->AdcRate$ .
- Поле `AdcPars->KadrRate`. Чтение. В данном поле возвращается частота кадра **KadrRate** в *кГц*. Но это поле не имеет силу при использовании *покадровой синхронизации ввода*, которая может задаваться полем `AdcPars->InputMode`. **KadrRate** рассчитывается исходя из величины `AdcPars->ChannelsQuantity`, а также уже скорректированных `AdcPars->AdcRate` и `AdcPars->InterKadrDelay`. Дополнительно про соотношения между упомянутыми выше величинами `AdcPars->ChannelsQuantity`, `AdcPars->AdcRate`, `AdcPars->InterKadrDelay` и `AdcPars->KadrRate` смотри [§ 3.4. "Формат кадра отсчетов"](#).
- Поле `AdcPars->AdcFifoBaseAddress`. Запись. Данное поле задаёт базовый адрес *FIFO* буфера АЦП в DSP модуля. Для данного модуля он всегда равен 0x0.
- Поле `AdcPars->AdcFifoLength`. Запись–Чтение. Данное поле задаёт 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 автоматически выполняется необходимая корректировка и по завершении оной в поле `dm->AdcFifoLength` будет находиться реально установленное значение длины *FIFO* буфера АЦП. **!!! ВАЖНО !!!** Настоятельно рекомендуется выбирать размер *FIFO* буфера АЦП так,

чтобы полное время его заполнения было не менее 15÷30 мс. Передача данных из *FIFO* буфера АЦП в РС производится только порциями по  $AdcPars \rightarrow AdcFifoLength/2$  отсчётов (по мере их готовности).

- Поля  $AdcPars \rightarrow AdcOffsetCoefs[]$  и  $AdcPars \rightarrow AdcScaleCoefs[]$ . Запись. Данные поля являются массивами типа *double*. Они должны содержать коэффициенты необходимые при проведении модулем процедуры автоматической корректировки получаемых с АЦП данных. Разрешение применять модулем такую процедуру корректировки данных задаётся с помощью поля  $AdcPars \rightarrow CorrectionEnabled$ . Подробнее смотри § 5.6.1. "Корректировка данных АЦП".

**Передаваемые параметры:**

- *AdcPars* – адрес структуры типа *ADC\_PARS\_E440* с требуемыми параметрами функционирования АЦП.

**Возвращаемое значение:** *TRUE* – функция успешно выполнена;  
*FALSE* – функция выполнена с ошибкой.

### 5.6.5. Получение текущих параметров работы АЦП

<b>Формат:</b>	<b>bool</b>	<i>GET_CUR_ADC_PARS(ADC_PARS_E440 *am)</i>	(версия 1.0)
	<b>BOOL</b>	<i>GET_ADC_PARS(ADC_PARS_E440 * const AdcPars)</i>	(с версии 3.0)

**Назначение:**

Данная функция считывает из модуля *E14-440* всю текущую информацию, которая используется при сборе данных с АЦП.

**Передаваемые параметры:**

- *AdcPars* – адрес структуры типа *ADC\_PARS\_E440* с полученными из модуля текущими параметрами функционирования АЦП.

**Возвращаемое значение:** *TRUE* – функция успешно выполнена;  
*FALSE* – функция выполнена с ошибкой.



### 5.6.6. Получение массива данных с АЦП

**Формат:** `bool` *ReadData*(`SHORT *Buffer`, `DWORD *NumberOfWordsToRead`,  
`LPDWORD NumberOfBytesRead`,  
`LPOVERLAPPED Overlapped`) (версия 1.0)  
**BOOL** *ReadData*(`IO_REQUEST_LUSBAPI * const ReadRequest`) (с версии 3.0)

#### Назначение:

Данная функция предназначена для извлечения из модуля *E14-440* очередной порции собранных данных АЦП. Эта функция должна использоваться совместно с функциями *START\_ADC()* и *STOP\_ADC()*.

Поля передаваемой структуры типа *IO\_REQUEST\_LUSBAPI* определяют параметры и требуемый режим получения данных с модуля *E14-440*. Назначения полей этой структуры приведены в таблице ниже:

Название поля	Описание
Buffer	<i>Буфер данных.</i> Чтение. <b>Buffer</b> предназначен для хранения получаемых с модуля данных АЦП. Перед его использованием в функции приложение само должно позаботиться о выделении достаточного кол-ва памяти под этот буфер. Полученные данные в буфере будут располагаться по-кадрово: 1 <sup>ый</sup> кадр, 2 <sup>ой</sup> кадр и т.д. Причём положение отсчётов в кадрах будет совпадать с порядком размещения соответствующих <i>логических каналов</i> в управляющей таблице <b>ControlTable</b> .
NumberOfWordsToPass	<i>Кол-во передаваемых данных.</i> Запись–Чтение. Данный параметр задаёт то кол-во отсчётов АЦП, которое данная функция просто обязана стребовать с модуля. Величина параметра <b>NumberOfWordsToPass</b> должна находиться в диапазоне от 32 до (1024*1024), а также быть кратной 32. В противном случае данная функция сама подкорректирует величину этого поля, и по возвращении из функции в нём будет находиться <i>реально</i> использованное значение кол-ва затребованных данных.
NumberOfWordsPassed	<i>Кол-во переданных данных.</i> Чтение. В данном параметре возвращается то кол-во отсчётов АЦП, которое данная функция реально получила из модуля. Для <i>асинхронного</i> режима работы данной функции (см. ниже поле <b>Overlapped</b> ) в этом параметре вполне может вернуться число 0, что <i>не является</i> ошибкой, учитывая специфику данного режима.

Overlapped	<p><i>Структура Overlapped.</i> Данное поле определяет, в каком именно режиме будет выполняться данная функция: <i>синхронном</i> или <i>асинхронном</i>:</p> <ul style="list-style-type: none"> <li>• <b>Overlapped = NULL.</b> В этом случае от функции требуется <i>синхронный</i> режим её выполнения. При этом функция честно пытается получить из модуля все затребованные данные, причём в течение всего этого времени функция не возвращает управление вызвавшему её приложению. Если в течение времени <b>TimeOut</b> <i>мс</i> (см. ниже) все требуемые данные из модуля не получены, то функция завершается и возвращает ошибку.</li> <li>• <b>Overlapped != NULL.</b> В этом случае от функции требуется <i>асинхронный</i> режим её выполнения. Подразумевается, что этому полю приложение уже присвоило указатель на заранее подготовленную структуру типа <b>OVERLAPPED</b>. В этом режиме данная функция выставляет системе, т.е. <i>Windows</i>, <i>асинхронный</i> запрос на получение требуемого кол-ва данных из модуля и сразу возвращает управление приложению. Т.е. происходит как бы полное переключивание задачи по сбору данных на <i>ядро</i> системы. Так как <i>асинхронный</i> запрос выполняется уже на уровне <i>ядра</i>, то пока оно его обрабатывает, приложение вполне может заниматься своими собственными задачами. Окончание же текущего <i>асинхронного</i> запроса приложение можно отслеживать с помощью стандартных <i>Windows API</i> функций, таких как: <i>WaitForSingleObject()</i>, <i>GetOverlappedResult()</i> или <i>HasOverlappedIoCompleted()</i>. Данные функции используют событие <i>Event</i>, которые предварительно уже должно было быть определено приложением в соответствующем поле структуры <b>Overlapped</b>. Событие <i>Event</i> активируется системой по окончании сбора <i>всех</i> затребованных данных, завершая тем самым текущий <i>асинхронный</i> запрос. В ряде случаев бывает просто необходимо прервать выполняющийся <i>асинхронный</i> запрос. Для этой цели и существует штатная <i>Windows API</i> функция <i>CancelIo()</i>. К сожалению, эта функция присутствует только в <i>Windows NT</i> подобных системах.</li> </ul>
TimeOut	<p>Время ожидания сбора данных. Это поле предназначено для использования только в <i>синхронном</i> режиме. Задаёт максимальное время ожидания выполнения <i>синхронного</i> запроса на сбор данных в <i>мс</i>. Если по истечении этого времени <i>все</i> затребованные данные недополучены, функция завершается и возвращает ошибку.</p>

В DSP модуле **E14-440** организован *FIFO* буфер АЦП с регулируемым размером от **64** до **12288** отсчётов. Такой буфер необходим уверенного сбора данных на предельных частотах работы АЦП. Так при частотах сбора порядка **400** кГц переполнение полного буфера произойдёт только через **31** мс, что является вполне достаточным периодом времени даже для такой ‘задумчивой’ системы как *Windows*.

Теперь следует упомянуть о некоторой специфике, присущей режимам данной функции:

1. *Синхронный* режим. Данный режим рекомендуется применять при организации однократного сбора данных, в котором количество отсчётов не превышает  $1024 \times 1024 = 1$  МСлова. В этом режиме функция **ReadData()** должна вызываться ТОЛЬКО после успешного исполнения **START\_ADC()**, которой, для порядку, может предшествовать функция **STOP\_ADC()**. Следует с большой осторожностью использовать данный режим при достаточно медленных частотах сбора и большом количестве запрашиваемых данных. Иначе данная функция может надолго ‘уйти’ в сбор данных и, следовательно, весьма длительное время не возвращать управление приложению. Пример корректного использования функций библиотеки `Lusbapi` в *синхронном* режиме в виде консольного приложения можно найти на нашем CD-ROM в директории `\E14-440\Examples\Borland C++ 5.02\ReadDataSynchro`.
2. *Асинхронный* режим. Этот режим функционально намного гибче, чем *синхронный* режим и его рекомендуется использовать при организации непрерывного *потокowego* сбора данных, когда количество вводимых отсчётов превышает  $1024 \times 1024 = 1$  МСлов. Данный режим позволяет организовывать в системе *Windows* очередь *асинхронных* запросов. Так можно сформировать очередь предварительных запросов даже непосредственно перед запуском сбора данных, но после функции **STOP\_ADC()**. Такое использование очереди запросов позволяет резко повысить надёжность сбора данных. Операционная система *Windows* не является, что называется, средой реального времени. Поэтому, работая в ней, как это обычно бывает, всего лишь на *пользовательском* уровне, а не на уровне *ядра*, никогда нельзя быть полностью уверенным в том, что система в самый нужный момент не отвлечётся на свои собственные нужды на более или менее продолжительный промежуток времени. Например, если для частоты сбора 400 кГц после **START\_ADC()**, но перед началом выполнения функции получения данных **ReadData()** система ‘задумалась’ более чем на *31 мс* (что является вполне рядовым событием), то сбой в принимаемых данных практически обеспечен. Однако если непосредственно перед **START\_ADC()** выставить с помощью **ReadData()** несколько (можно и один) предварительных запросов, которые будут отрабатываться уже на уровне *ядра* системы, то сбоев не будет. Это происходит потому, что время отклика на отработку какого-нибудь события (в нашем случае это запрос) на уровне *ядра* существенно меньше, чем на *пользовательском* уровне. Т.о. получается, что после выполнения функции **START\_ADC()** у нас уже есть готовые к обслуживанию запросы на уровне *ядра* системы. Практически никаких задержек. А теперь, пока система выполняет наши предварительные запросы, можно не торопясь, по мере необходимости, выставлять один или несколько следующих запросов. Важно понимать, что для каждого выставленного в очередь или уже выполняющегося запроса у приложения должен существовать свой экземпляр структуры типа **IO\_REQUEST\_LUSBAPI** со своим индивидуальным событием *Event*.

**Передаваемые параметры:**

- *ReadRequest* – структура типа **IO\_REQUEST\_LUSBAPI** с параметрами извлечения готовых данных АЦП из модуля *E14-440*.

**Возвращаемое значение:** *TRUE* – функция успешно выполнена;  
*FALSE* – функция выполнена с ошибкой.

### 5.6.7. Ввод кадра отсчетов с АЦП

<b>Формат:</b>	<b>BOOL</b>	<i>ADC_KADR(SHORT * const Data)</i>	(с версии 2.0)
<b>Назначение:</b>	<p>С версии 2.0 в библиотеке Lusbari появилась дополнительная интерфейсная функция, которая позволяет осуществлять ввод кадра отсчетов с АЦП модуля. Эта функция удобна для осуществления достаточно <i>медленного</i>, порядка нескольких десятков Гц, 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 <i>SET_ADC_PARS()</i>. При этом информация о синхронизации ввода данных никак не используется, т.е. просто игнорируется. Массив <i>Data</i> необходимой длины для получаемых с модуля данных следует заранее определить. Более подробно смотри исходные тексты примера из директории <code>\E14-440\Example\Borland C++ 5.02\AdcKadr</code>.</p>		
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>Data</i> – указатель на массив, в который складываются полученный кадр отсчетов с АЦП.</li></ul>		
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.		

### 5.6.8. Однократный ввод с АЦП

<b>Формат:</b>	<b>BOOL</b>	<i>ADC_SAMPLE(SHORT * const AdcData, WORD AdcChannel)</i>	
<b>Назначение:</b>	<p>Данная функция устанавливает заданный логический канал и осуществляет его однократное аналого-цифровое преобразование. Эта функция удобна для осуществления достаточно <i>медленного</i>, порядка нескольких десятков Гц, асинхронного ввода данных с задаваемого логического канала АЦП (см. § 3.2.3. "Логический номер канала АЦП"). Предварительно, с помощью штатной интерфейсной функции <i>SET_ADC_PARS()</i>, можно разрешить корректировку данных с заданного канала АЦП. Более подробно смотри исходные тексты примера из директории <code>\E14-440\Example\Borland C++ 5.02\AdcSample</code>.</p>		
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>AdcSample</i> – результат преобразования по заданному логическому каналу АЦП <i>AdcChannel</i>;</li><li>• <i>AdcChannel</i> – требуемый логический номер канала АЦП.</li></ul>		
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.		

## 5.7. Функции для работы с ЦАП

Интерфейсные функции штатной библиотеки `Lusbapi` позволяют реализовывать разнообразные алгоритмы работы модуля *E14-440* 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, т.е. *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 функционирования: частота работы ЦАП, длина и базовый адрес *FIFO* буфера ЦАП. Эту операцию можно выполнить с помощью интерфейсной функции `SET_DAC_PARS()`. Непосредственно после выполнения функции `START_DAC()` модуль приступает к выводу данных на ЦАП. Причём данные для этого берутся из *FIFO* буфера ЦАП, который расположен в памяти данных DSP модуля (см. § 4.2. “Общая структура драйвера DSP”). Поэтому **очень важно**, перед запуском ЦАП, проинициализировать этот буфер требуемыми данными, например, с помощью интерфейсной функции `PUT_DM_ARRAY()`. О формате данных передаваемых в *FIFO* буфер ЦАП смотри § 3.2.2. “Формат слова данных для ЦАП”. Для ‘подкачки’ же из приложения в *FIFO* буфер ЦАП *новых* данных уже в процессе потокового вывода, 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 директории `\E14-440\Example`.

### 5.7.1. Корректировка данных ЦАП

Схемотехника и использованные компоненты обеспечивают линейность передаточной характеристики ЦАП тракта модуля *E14-440*. Однако на *сегодняшний* день модуль не умеет производить автоматическую корректировку выводимых на ЦАП данных. Это приводит к тому, что выходные показания ЦАП могут иметь некоторое смещение нуля и неточность в передаче масштаба. Поэтому на уровне приложения необходима реализация всей этой нудной задачи по корректировке данных ЦАП. Для этих целей предназначены соответствующие *штатные* калибровочные коэффициенты, хранящиеся в служебной информации модуля. Служебная информация совместно с нужными коэффициентами записывается в модуль на этапе при его наладке в **ЗАО “Л-Кард”**. Благодаря этому на модуле отсутствуют подстроечные резисторы, что улучшает шумовые характеристики модуля и увеличивает их надежность.

*Штатные* коэффициенты располагаются в полях `Dac.OffsetCalibration[]` и `Dac.ScaleCalibration[]` структуры служебной информации `MODULE_DESCRIPTION_E440`. Эти поля представляют собой массивы типа `double`. Для модуля *E14-440* в каждом из этих массивов используются только первые `DAC_CALIBR_COEFS_QUANTITY_E440` элементов. Массив `Dac.OffsetCalibration` содержит коэффициенты для корректировки смещение нуля, а массив `Dac.ScaleCalibration` – для корректировки масштаба.

В общем виде при использовании *штатных* коэффициентов корректировка данных ЦАП производится по следующей формуле:

$$Y = (X+A)*B,$$

где: X – некорректированные данные ЦАП [в кодах ЦАП],

Y – скорректированные данные ЦАП [в кодах ЦАП],

A – коэффициент смещения нуля [в кодах ЦАП],

B – коэффициент масштаба [безразмерный].

Например, на втором канале ЦАП необходимо выставить напряжения, соответствующие следующим кодам ЦАП: X1 = 1000, X2 = -1000, X3 = 0. Тогда коэффициенты коррекции и данные для второго канала ЦАП можно представить так: A = `Dac.OffsetCalibration[1]`, B = `Dac.ScaleCalibration[1]`, Y1=(A+1000)\*B, Y2=(A-1000)\*B, Y3=A\*B.

### 5.7.2. Запуск вывода данных на ЦАП

<b>Формат:</b>	<b>BOOL</b>	<b><i>START_DAC(void)</i></b>
<b>Назначение:</b>	<p>Данная функция запускает модуль <i>E14-440</i> на непрерывный <i>поточковый</i> вывод данных на ЦАП. Перед любым запуском вывода данных настоятельно рекомендуется выполнять функцию <i>STOP_DAC()</i>. После выполнения функции <i>START_DAC()</i> данные из <i>FIFO</i> буфера ЦАП начинают последовательно, с заданной частотой, выводиться на ЦАП. Параметры работы ЦАП должны быть предварительно переданы в модуль с помощью интерфейсной функции <i>SET_DAC_PARS()</i>. Также предварительно перед запуском ЦАП следует проинициализировать <i>FIFO</i> буфер ЦАП необходимыми начальными значениями с помощью, например, интерфейсной функции <i>PUT_DM_ARRAY()</i>. Подробности о начальной инициализации <i>FIFO</i> буфера ЦАП и формате данных для ЦАП см. в прилагаемых к модулю примерах, а также см. § 3.2.2. "Формат слова данных для ЦАП". 'Подкачку' же в модуль <i>новых</i> данных (уже в ходе работы ЦАП) для <i>FIFO</i> буфера ЦАП можно осуществлять с помощью интерфейсной функции <i>WriteData()</i>.</p>	
<b>Передаваемые параметры:</b>	нет	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

### 5.7.3. Останов вывода данных на ЦАП

<b>Формат:</b>	<b>BOOL</b>	<b><i>STOP_DAC(void)</i></b>
<b>Назначение:</b>	<p>Данная функция останавливает в модуле <i>E14-440</i> механизм вывода данных на ЦАП. Попутно эта функция 'приводит в чувство' основную программу микроконтроллера модуля, а также сбрасывает используемый канал передачи данных по <b>USB</b> шине. Поэтому весьма <b>настоятельно</b> рекомендуется применять эту функцию <b>перед</b> каждым запуском вывода данных функцией <i>START_DAC()</i>.</p>	
<b>Передаваемые параметры:</b>	нет	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

#### 5.7.4. Установка параметров работы ЦАП

<b>Формат:</b>	<b>bool</b> <i>FILL_DAC_PARS</i> ( <i>DAC_PARS_E440 *dm</i> ) (версия 1.0) <b>BOOL</b> <i>SET_DAC_PARS</i> ( <i>DAC_PARS_E440 * const DacPars</i> ) (с версии 3.0)
<b>Назначение:</b>	<p>Данная функция передает в <i>E14-440</i> всю необходимую информацию, которая используется модулем для организации вывода данных на ЦАП. Всю нужную информацию описываемая интерфейсная функция извлекает из полей передаваемой структуры типа <i>DAC_PARS_E440</i>. Собственно использование модулем <b>именно</b> этой переданной информацией начинается <b>только</b> после выполнения интерфейсной функции <i>START_DAC()</i>. Также крайне не рекомендуется вызывать эту функцию собственно в процессе сбора данных. Её следует применять <b>только</b> после выполнения функции <i>STOP_DAC()</i>.</p> <p>Описание структуры <i>DAC_PARS_E440</i> приведено ранее в § 5.2.3. "Структура <i>DAC_PARS_E440</i>", а назначение отдельных ее полей описано ниже.</p> <ul style="list-style-type: none"><li>• Поле <i>DacPars-&gt;DacRate</i>. Запись–Чтение. При входе в функцию данное поле содержит требуемую частоту работы ЦАП <b>DacRate</b> в кГц. После выполнения данной интерфейсной функции в этом поле возвращается <b>реально установленное</b> значение частоты вывода данных на ЦАП, <b>максимально близкое</b> к изначально задаваемой величине. Это происходит вследствие того, что реальные значения <b>DacRate</b> не являются непрерывной величиной, а образуют некую сетку частот. Так, в общем виде, частота работы ЦАП определяется по следующей формуле: <math>DacRate = F_{clockout} / (12 * (N + 1))</math>, где <math>F_{clockout}</math> – тактовая частота DSP равная 48000 кГц, N – целое 16<sup>ти</sup> битное число. Поэтому данная функция просто вычисляет ближайшую к задаваемой дискретную величину <b>DacRate</b>, передает ее в модуль (в виде числа N), а также возвращает её значение в поле <i>DacPars-&gt;DacRate</i>. Минимальное значение <b>DacRate</b> равно 0.061 кГц, максимальное – 125 кГц.</li><li>• Поле <i>DacPars-&gt;DacFifoBaseAddress</i>. Запись. Данное поле задаёт базовый адрес <i>FIFO</i> буфера ЦАП в DSP модуля. Для данного модуля он всегда равен 0x3000.</li><li>• Поле <i>DacPars-&gt;DacFifoLength</i>. Запись–Чтение. Данное поле задаёт длину <i>FIFO</i> буфера ЦАП в DSP модуля. Для данного модуля эта величина может находиться в диапазоне от 0x40 (64) до 0xFC0 (4032), а также быть обязательно кратной 0x40(64). Если переданное в функцию значение не удовлетворяет указанным требованиям, то автоматически выполняется необходимая корректировка и по завершении данной функции в поле <i>DacPars-&gt;DacFifoBaseAddress</i> будет находиться <b>реально установленное</b> значение длины <i>FIFO</i> буфера ЦАП. Передача ('подкачка') новых данных из PC в <i>FIFO</i> буфер ЦАП модуля производится порциями по <i>DacPars-&gt;DacFifoBaseAddress/2</i> отсчетов (по мере их необходимости).</li></ul>
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>DacPars</i> – адрес структуры типа <i>DAC_PARS_E440</i> с параметрами функционирования ЦАП.</li></ul>
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.

### 5.7.5. Получение текущих параметров работы ЦАП

<b>Формат:</b>	<b>bool</b> <i>GET_CUR_DAC_PARS(DAC_PARS_E440 *dm)</i> (версия 1.0)
	<b>BOOL</b> <i>GET_DAC_PARS(DAC_PARS_E440 * const DacPars)</i> (с версии 3.0)
<b>Назначение:</b>	Данная функция считывает из модуля всю текущую информацию, которая используется в процессе потоковой выдачи данных на ЦАП.
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"> <li><i>DacPars</i> – адрес структуры <a href="#">DAC_PARS_E440</a> с полученными из модуля текущими параметрами функционирования ЦАП.</li> </ul>
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.

### 5.7.6. Передача массива данных в ЦАП

<b>Формат:</b>	<b>bool</b> <i>WriteData(WORD *Buffer, DWORD *NumberOfWordsToWrite, LPDWORD NumberOfBytesWritten, LPOVERLAPPED Overlapped)</i> (версия 1.0)
	<b>BOOL</b> <i>WriteData(IO_REQUEST_LUSBAPI * const WriteRequest)</i> (с версии 3.0)
<b>Назначение:</b>	<p>Данная функция предназначена для передачи очередной порции <i>новых</i> данных в <i>FIFO</i> буфер ЦАП модуля для их последующего вывода на ЦАП. Эта функция должна использоваться совместно с функциями <a href="#">START_DAC()</a> и <a href="#">STOP_DAC()</a>.</p> <p>Поля передаваемой структуры типа <a href="#">IO_REQUEST_LUSBAPI</a> определяют параметры и требуемый режим передачи данных в модуль <i>E14-440</i>. Назначения полей этой структуры приведены в таблице ниже:</p>
<b>Название поля</b>	<b>Описание</b>
Buffer	<i>Буфер данных.</i> Запись. <b>Buffer</b> предназначен для хранения очередной порции данных, которые необходимо будет переслать в <i>FIFO</i> буфер ЦАП модуля. Перед использованием <b>Buffer</b> в функции приложение само должно позаботиться о выделении достаточного кол-ва памяти под этот буфер. Также необходимо сформировать и расположить в <b>Buffer</b> сами данные для ЦАП. Формат данных для ЦАП см <a href="#">§ 3.2.2. "Формат слова данных для ЦАП"</a> .
NumberOfWordsToPass	<i>Кол-во передаваемых данных.</i> Запись–Чтение. Данный параметр задаёт то кол-во отсчётов ЦАП, которое данная функция просто обязана передать в модуль. Величина параметра <b>NumberOfWordsToPass</b> должна находиться в диапазоне от 32 до (1024*1024), а также быть кратной 32. В противном случае данная функция сама подкорректирует величину этого поля, и по возвращении из функции в нём будет находиться <i>реально</i> использованное значение кол-ва передаваемых данных.



<p>NumberOfWordsPassed</p>	<p><i>Кол-во переданных данных.</i> Чтение. В данном параметре возвращается то кол-во отсчётов ЦАП, которое данная функция реально передала в модуль. Для <i>асинхронного</i> режима работы данной функции (см. ниже поле <b>Overlapped</b>) в этом параметре вполне может вернуть число 0, что <i>не является</i> ошибкой, учитывая специфику данного режима.</p>
<p>Overlapped</p>	<p><i>Структура Overlapped.</i> Данное поле определяет, в каком именно режиме будет выполняться данная функция: <i>синхронном</i> или <i>асинхронном</i>:</p> <ul style="list-style-type: none"> <li>• <b>Overlapped = NULL.</b> В этом случае от функции требуется <i>синхронный</i> режим её выполнения. При этом функция честно пытается переслать в модуль все требуемые данные, причём в течение всего этого времени функция не возвращает управление вызвавшему её приложению. Если в течение времени <b>TimeOut</b> <i>мс</i> (см. ниже) все требуемые данные в модуль не переданы, то функция завершается и возвращает ошибку.</li> <li>• <b>Overlapped != NULL.</b> В этом случае от функции требуется <i>асинхронный</i> режим её выполнения. Подразумевается, что этому полю приложение уже присвоила указатель на заранее подготовленную структуру типа <i>OVERLAPPED</i>. В этом режиме данная функция выставляет системе, т.е. <i>Windows</i>, <i>асинхронный</i> запрос на передачу требуемого кол-ва данных в модуль и сразу возвращает управление приложению. Т.е. происходит как бы полное переключивание задачи по передаче данных на <i>ядро</i> системы. Так как <i>асинхронный</i> запрос выполняется уже на уровне <i>ядра</i>, то пока оно его обрабатывает, приложение вполне может занимать своими собственными делами. Окончание же текущего <i>асинхронного</i> запроса приложение можно отслеживать с помощью стандартных <i>Windows API</i> функций, таких как: <i>WaitForSingleObject()</i>, <i>GetOverlappedResult()</i> или <i>HasOverlappedIoCompleted()</i>. Данные функции используют событие <i>Event</i>, которые предварительно уже должно было быть определено приложением в соответствующем поле структуры <b>Overlapped</b>. Событие <i>Event</i> активируется системой по окончании передачи <i>всех</i> затребованных данных, завершая тем самым текущий <i>асинхронный</i> запрос. В ряде случаев бывает просто необходимо прервать выполняющийся <i>асинхронный</i> запрос. Для этой цели и существует штатная <i>Windows API</i> функция <i>CancelIo()</i>. К сожалению, существует эта функция только в <i>Windows NT</i> подобных системах.</li> </ul>
<p>TimeOut</p>	<p><i>Время ожидания передачи данных.</i> Это поле предназначено для использования только в <i>синхронном</i> режиме. Задаёт максимальное время ожидания выполнения <i>синхронного</i> запроса на сбор данных в <i>мс</i>. Если по истечении этого времени <i>все</i> затребованные данные недопереданы, функция завершается и возвращает ошибку.</p>

В DSP модуле *E14-440* организован программный *FIFO* буфер ЦАП с регулируемым размером от 64 до 4032 отсчётов. Такой буфер необходим уверенного вывода данных на предельных частотах работы ЦАП. Так при частотах вывода порядка 125 кГц переполнение полного буфера произойдёт только через 32 мс, что является вполне достаточным периодом времени даже для такой ‘задумчивой’ системы как *Windows*.

Теперь следует упомянуть о некоторой специфике присущей режимам данной функции:

1. *Синхронный* режим. Данный режим рекомендуется применять при организации однократного вывода данных на ЦАП, в котором количество отсчётов не превышает  $1024 * 1024 = 1$  МСлова. В этом режиме функция *WriteData()* должна вызываться только после успешного исполнения *START\_DAC()*, которой, для порядку, может предшествовать функция *STOP\_DAC()*. Следует с великой аккуратностью использовать данный режим при достаточно медленных частотах вывода и большом количестве передаваемых данных. Иначе данная функция может надолго ‘уйти’ в вывод данных и, следовательно, весьма длительное время не возвращать управление приложению.
2. *Асинхронный* режим. Этот режим функционально намного гибче, чем *синхронный* режим и его рекомендуется использовать при организации непрерывного *потокowego* вывода данных на ЦАП, когда количество выводимых отсчётов превышает  $1024 * 1024 = 1$  МСлов. Данный режим позволяет организовывать в системе *Windows* очередь *асинхронных* запросов. Так можно сформировать очередь предварительных запросов даже непосредственно перед запуском вывода данных, но после функции *STOP\_DAC()*. Такое использование очереди запросов позволяет резко повысить надёжность вывода данных. Операционная система *Windows* не является, что называется, средой реального времени. Поэтому, работая в ней, как это обычно бывает, всего лишь на *пользовательском* уровне, а не на уровне *ядра*, никогда нельзя быть полностью уверенным в том, что система в самый нужный момент не отвлечётся на свои собственные нужды на более или менее продолжительный промежуток времени. Например, если для частоты работы ЦАП равной 125 кГц после *START\_DAC()*, но перед началом выполнения функции *WriteData()* система ‘задумалась’ более чем на 32 мс (что является вполне рядовым событием), то сбой в передаваемых данных практически обеспечен. Однако если непосредственно перед *START\_DAC()* выставить с помощью *WriteData()* несколько (можно и один) предварительных запросов, которые будут отрабатываться уже на уровне *ядра* системы, то сбоев не будет. Это происходит потому, что время отклика на отработку какого-нибудь события (в нашем случае это запрос) на уровне *ядра* существенно меньше, чем на *пользовательском* уровне. Т.о. получается, что после выполнения функции *START\_DAC()* у нас уже есть готовые к обслуживанию запросы на уровне *ядра* системы. Практически никаких задержек. А теперь пока система выполняет наши предварительные запросы, можно не торопясь, по мере надобности, выставлять один или несколько следующих запросов. Важно понимать, что для каждого выставленного в очередь или уже выполняющегося запроса у приложения должен существовать свой экземпляр структуры типа *IO\_REQUEST\_LUSBAPI* со своим индивидуальным событием *Event*.

**Передаваемые параметры:**

- *WriteRequest* – структура типа *IO\_REQUEST\_LUSBAPI* с параметрами вывода данных на ЦАП модуля *E14-440*.

**Возвращаемое значение:** *TRUE* – функция успешно выполнена;  
*FALSE* – функция выполнена с ошибкой.

### 5.7.7. Однократный вывод на ЦАП

<b>Формат:</b>	<b>bool</b>	<i>DAC_SAMPLE</i> ( <i>WORD DacData</i> , <i>WORD DacChannel</i> )	(версия 1.0)
	<b>BOOL</b>	<i>DAC_SAMPLE</i> ( <i>SHORT * const DacData</i> , <i>WORD DacChannel</i> )	(с версии 3.0)
<b>Назначение:</b>	<p>Данная функция позволяет однократно устанавливать на задаваемом канале ЦАП <i>DacChannel</i> напряжение в соответствии со значением <i>DacData</i> (в кодах ЦАП). Ожидается, что величина <i>DacData</i> должна находиться в диапазоне от <b>-2048</b> до <b>2047</b>, иначе функция автоматически ограничит значение <i>DacData</i> указанными пределами. Функция <i>DAC_SAMPLE()</i> выполняется достаточно <i>медленно</i> и, используя её, можно достичь частоты вывода данных на ЦАП порядка нескольких десятков <i>Гц</i>. О соответствии кода ЦАП величине устанавливаемого на выходе модуля аналогового напряжения см. <a href="#">§ 3.2.2. "Формат слова данных для ЦАП"</a>.</p>		
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>DacChannel</i> – требуемый номер канала ЦАП (0 или 1).</li><li>• <i>DacData</i> – устанавливаемое значение напряжения в кодах ЦАП (от <b>-2048</b> до <b>2047</b>).</li></ul>		
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.		

## 5.8. Функции для работы с внешними цифровыми линиями

Все доступные цифровые линии модуля *E14-440* располагаются на внешнем разъёме *DRB-37F*. Дополнительную информацию об этом разъёме можно найти в "*E14-440. Руководство пользователя*", § 3.3.2. "*Внешний разъём для подключения цифровых сигналов*".

Аппаратура модуля *E14-440* и, соответственно, библиотека `Lusbapi` позволяет работать с цифровыми линиями **ТОЛЬКО** асинхронным (однократным) образом. Т.о. работа с цифровыми линиями получается сравнительно медленной операцией, т.к. на модуле не предусмотрена аппаратная поддержка *поточковой* работы с ними.

### 5.8.1. Разрешение выходных цифровых линий

<b>Формат:</b>	<b>BOOL</b>	<i>ENABLE_TTL_OUT(BOOL EnableTtlOut)</i>
<b>Назначение:</b>	Данная интерфейсная функция позволяет осуществлять управление разрешением <i>всех</i> выходных линий внешнего цифрового разъёма <i>DRB-37F</i> . Т.о. существует возможность перевода их в третье, <i>высокоимпеданное</i> , состояние и обратно. Непосредственно после подачи на модуль питания выходные цифровые линии находятся в третьем состоянии.	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>EnableTtlOut</i> – флажок, управляющий состоянием разрешения всех цифровых выходных линий.</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

### 5.8.2. Чтение внешних цифровых линий

<b>Формат:</b>	<b>BOOL</b>	<i>TTL_IN(WORD * const TtlIn)</i>
<b>Назначение:</b>	Данная интерфейсная функция осуществляет однократное асинхронное чтение состояний всех $16^{\text{th}}$ входных цифровых линий на внешнем разъёме <i>DRB-37F</i> модуля <i>E14-440</i> . Функция <i>TTL_IN()</i> выполняется достаточно <i>медленно</i> и, используя её, можно достичь частоты ввода данных с цифровых линий порядка нескольких десятков <i>Гц</i> .	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>TtlIn</i> – переменная, в которой возвращается побитовое состояние входных цифровых линий модуля.</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

### 5.8.3. Вывод на внешние цифровые линии

<b>Формат:</b>	<b>BOOL</b>	<b><i>TTL_OUT</i>(WORD <i>TtlOut</i>)</b>
<b>Назначение:</b>	<p>Данная интерфейсная функция осуществляет установку <i>всех</i> 16<sup>ти</sup> выходных цифровых линий на внешнем разъёме <i>DRB-37F</i> модуля <i>E14-440</i> в соответствии с битами передаваемого параметра <i>TtlOut</i>. Работа с цифровыми выходами предварительно <b>должна быть разрешена</b> с помощью интерфейсной функции <a href="#">ENABLE_TTL_OUT()</a>. Функция <i>TTL_OUT()</i> выполняется достаточно <i>медленно</i> и, используя её, можно достичь частоты вывода данных на цифровые линии порядка нескольких десятков Гц.</p>	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>TtlOut</i> – переменная, содержащая побитовое состояние устанавливаемых выходных цифровых линий модуля.</li></ul>	
<b>Возвращаемое значение:</b>	<p><i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.</p>	

## 5.9. Функции для работы с пользовательским ППЗУ

На модуле *E14-440* 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 отсчетов АЦП и ЦАП и т.д. А оставшиеся 32 ячейки предназначены для целей пользователя.

### 5.9.1. Разрешение/запрещение записи в ППЗУ

<b>Формат:</b>	<b>BOOL</b>	<b><i>ENABLE_FLASH_WRITE</i></b> ( <i>BOOL EnableFlashWrite</i> )
<b>Назначение:</b>	Данная интерфейсная функция осуществляет управление режимом записи данных в пользовательское ППЗУ. Если запись разрешена, то с помощью штатной интерфейсной функции <b><i>WRITE_FLASH_WORD()</i></b> можно реализовать саму процедуру записи данных. Следует помнить, что после завершения всех требуемых операций записи информации в ППЗУ, настоятельно рекомендуется запретить с помощью данной интерфейсной функции режим записи данных в пользовательское ППЗУ.	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>EnableFlashWrite</i> – переменная может принимать следующие значения:<ul style="list-style-type: none"><li>✓ если <i>TRUE</i>, то режим записи в ППЗУ разрешен,</li><li>✓ если <i>FALSE</i>, то режим записи в ППЗУ запрещен.</li></ul></li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

### 5.9.2. Запись слова в ППЗУ

<b>Формат:</b>	<b>BOOL</b>	<b><i>WRITE_FLASH_WORD</i></b> ( <i>WORD FlashAddress, SHORT FlashWord</i> )
<b>Назначение:</b>	Данная интерфейсная функция выполняет запись 16 <sup>ти</sup> битного слова <i>FlashWord</i> в ячейку пользовательского ППЗУ с номером <i>FlashAddress</i> . Перед началом записи в ППЗУ необходимо разрешить эту операцию с помощью интерфейсной функции <b><i>ENABLE_FLASH_WRITE()</i></b> . После окончания цикла записи всей требуемой информации настоятельно рекомендуется запретить режим записи в пользовательское ППЗУ с помощью той же функции <b><i>ENABLE_FLASH_WRITE()</i></b> . Т.к. в первых 32 ячейках ППЗУ находится служебная информация, которая используется библиотекой <i>Lusbapi</i> в процессе работы с модулем, то для пользователя доступны адреса ячеек только с 32 по 63.	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>FlashAddress</i> – номер ячейки ППЗУ, куда будет записано слово <i>FlashWord</i>;</li><li>• <i>FlashWord</i> – слово, значение которого должно быть записано в ППЗУ.</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

### 5.9.3. Чтение слова из ППЗУ

<b>Формат:</b>	<b>BOOL</b>	<b><i>READ_FLASH_WORD</i></b> ( <i>WORD FlashAddress</i> , <i>SHORT * const FlashWord</i> )
<b>Назначение:</b>	Данная интерфейсная функция возвращает значения слова, находящегося в ячейке пользовательского ППЗУ с номером <i>FlashAddress</i> .	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>FlashAddress</i> – номер ячейки ППЗУ, откуда должно быть считано слово;</li><li>• <i>FlashWord</i> – считанное значение.</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

## 5.10. Функции для работы со служебной информацией

Служебная информация содержит самые общие данные об используемом модуле *E14-440*: название модуля, его серийный номер и ревизию, корректировочные коэффициенты для АЦП и ЦАП, версии используемых прошивок MCU и DSP, тактовые частоты работы исполнительных устройств (MCU, DSP) и многое другое. Некоторые данные из этой служебной информации необходимы функциям штатной библиотеки `Lusbapi` для своей корректной работы.

### 5.10.1. Чтение служебной информации

<b>Формат:</b>	<b>BOOL</b> <i>GET_MODULE_DESCR</i> ( <i>MODULE_DESCR_E440 *md</i> ) (версия 1.0) <b>BOOL</b> <i>GET_MODULE_DESCRIPTION</i> ( <i>MODULE_DESCRIPTION_E440 * const ModuleDescription</i> ) (с версии 3.0)
<b>Назначение:</b>	Данная интерфейсная функция осуществляет чтение всей служебной информации о модуле <i>E14-440</i> в структуру типа <i>MODULE_DESCRIPTION_E440</i> (см. § 5.2.1 "Структура <i>MODULE_DESCRIPTION_E440</i> "). Эта информация требуется при работе с некоторыми интерфейсными функциями библиотеки <code>Lusbapi</code> . Поэтому данную функцию, во избежания непредсказуемого поведения приложений, следует <b>обязательно</b> вызывать непосредственно после загрузки в модуль драйвера DSP и проверки его работоспособности (см. § 4.1. "Общий подход к работе с интерфейсными функциями")
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>ModuleDescription</i> – указатель на структуру типа <i>MODULE_DESCRIPTION_E440</i>, в которую заносится вся служебная информация модуля.</li></ul>
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.

### 5.10.2. Запись служебной информации

<b>Формат:</b>	<b>bool</b> <i>SAVE_MODULE_DESCR</i> ( <i>MODULE_DESCR_E440 *md</i> ) (версия 1.0) <b>BOOL</b> <i>SAVE_MODULE_DESCRIPTION</i> ( <i>MODULE_DESCRIPTION_E440 * const ModuleDescription</i> ) (с версии 3.0)
<b>Назначение:</b>	Данная интерфейсная функция позволяет сохранять в модуле всю служебную информацию из структуры типа <i>MODULE_DESCRIPTION_E440</i> . <b>!!!Внимание!!!</b> Применять данную функцию нужно только в случае крайней необходимости. Например, когда по тем или иным обстоятельствам испортилось содержимое служебной информации.
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>ModuleDescription</i> – указатель на структуру типа <i>MODULE_DESCRIPTION_E440</i>, из которой служебная информация переносится в модуль.</li></ul>
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.



## 5.11. Функции для работы с загрузочным ППЗУ

Начиная с ревизии **F** на модуле *E14-440* устанавливается *загрузочное* ППЗУ объёмом кБайт. По замыслу оно предназначено для хранения образа управляющей программы DSP. Однако пользователь по своему усмотрению может хранить в нём любую нужную ему информацию. Для работы с *загрузочным* ППЗУ доступны функции по сбросу всего ППЗУ (*reset*), а также записи и чтения заданного массива данных.

### 5.11.1. Загрузка DSP модуля

Наличие *загрузочного* ППЗУ позволяет организовать загрузку DSP либо по программному запросу (смотри описание функции [LOAD\\_MODULE\(\)](#)), либо автоматически после подачи питания на модуль.

Для этого необходимо, чтобы в первых 8<sup>МБ</sup> байтах загрузочного ППЗУ располагалась информация, структурированная особым образом. Эту структуру можно определить так:

```
struct BOOT_FLASH_HEADER
{
    BYTE KeyBytes[0x4];
    WORD PmWords;
    WORD DmWords;
};
```

### 5.11.2. Сброс загрузочного ППЗУ

<b>Формат:</b>	<b>BOOL</b>	<b><i>ERASE_BOOT_FLASH</i>(void)</b>
<b>Назначение:</b>	Данная интерфейсная функция осуществляет полный сброс ( <i>reset</i> ) загрузочного ППЗУ.	
<b>• Передаваемые параметры:</b>	нет.	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

### 5.11.3. Запись загрузочного ППЗУ

<b>Формат:</b>	<b>BOOL</b>	<b><i>PUT_ARRAY_BOOT_FLASH</i>(DWORD BaseAddress, DWORD NBytes, BYTE *Data)</b>
<b>Назначение:</b>	Данная интерфейсная функция осуществляет	
<b>Передаваемые параметры:</b>	<ul style="list-style-type: none"><li>• <i>BaseAddress</i> – номер ячейки ППЗУ, куда будет записано слово <i>FlashWord</i>;</li><li>• <i>NBytes</i> – ;</li><li>• <i>Data</i> – массив передаваемых данных.</li></ul>	
<b>Возвращаемое значение:</b>	<i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.	

#### 5.11.4. Чтение загрузочного ППЗУ

<b>Формат:</b> <b>BOOL</b> <b>GET_ARRAY_BOOT_FLASH</b> ( <i>DWORD BaseAddress,</i> <i>DWORD NBytes, BYTE *Data</i> )
<b>Назначение:</b> Данная интерфейсная функция осуществляет
<b>Передаваемые параметры:</b> <ul style="list-style-type: none"><li>• <i>BaseAddress</i> – номер ячейки ППЗУ, куда будет записано слово <i>FlashWord</i>;</li><li>• <i>NBytes</i> – ;</li><li>• <i>Data</i> – буфер под массив получаемых данных.</li></ul>
<b>Возвращаемое значение:</b> <i>TRUE</i> – функция успешно выполнена; <i>FALSE</i> – функция выполнена с ошибкой.

## Приложение А. СТРУКТУРА ПАМЯТИ ДРАЙВЕРА DSP

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:

Память программ	адрес	Память данных	адрес
Код фирменного драйвера LBIOS	0x3FFF	32 управляющих регистра DSP	0x3FFF
			0x3FE0
		FIFO буфер ЦАП	0x3FDF
			0x3000
	0x0400	FIFO буфер АЦП	0x2FFF
Переменные LBIOS	0x03FF		
	0x0030		
Таблица прерываний	0x002F		
	0x0000		0x0000

Карта распределения памяти для ADSP-2185

## Приложение В. ВСПОМОГАТЕЛЬНЫЕ КОНСТАНТЫ И ТИПЫ

Вспомогательные константы и типы данных описаны в заголовочном файле `\DLL\Include\LusbapiTypes.h` и рассмотрены в нижеследующих разделах.

### В.1. КОНСТАНТЫ

Вспомогательные константы, определённые в библиотеке `Lusbapi`, приведены в следующей таблице:

Название	Значение	Смысл
<code>NAME_LINE_LENGTH_LUSBAPI</code>	25	Длина строки с названием чего-либо. Например, название производителя или изделия, имя автора и т.д.
<code>COMMENT_LINE_LENGTH_LUSBAPI</code>	256	Длина строки с комментарием в какой-либо вспомогательной структуре.
<code>ADC_CALIBR_COEFS_QUANTITY_LUSBAPI</code>	128	Максимально возможное число корректировочных коэффициентов АЦП.
<code>DAC_CALIBR_COEFS_QUANTITY_LUSBAPI</code>	128	Максимально возможное число корректировочных коэффициентов ЦАП.

### В.2. СТРУКТУРА VERSION INFO LUSBAPI

Вспомогательная структура `VERSION_INFO_LUSBAPI` содержит более или менее подробную информацию о программном обеспечении, работающем в каком-либо исполнительном устройстве: MCU, DSP, PLD и т.д. Данная структура описывается следующим образом:

```
struct VERSION_INFO_LUSBAPI
{
    BYTE Version[10];           // версия ПО для исполнительного устройства
    BYTE Date[14];             // дата сборки ПО
    BYTE Manufacturer[NAME_LINE_LENGTH_LUSBAPI]; // производитель ПО
    BYTE Author[NAME_LINE_LENGTH_LUSBAPI];       // автор ПО
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI];  // строка комментария
};
```

### В.3. СТРУКТУРА MODULE INFO LUSBAPI

Данная вспомогательная структура `MODULE_INFO_LUSBAPI` содержит самую общую информацию о модуле: название фирмы-изготовителя изделия, название изделия, серийный номер изделия, ревизия изделия и строка комментария. Эта структура описывается следующим образом:

```
struct MODULE_INFO_LUSBAPI
{
    BYTE CompanyName[NAME_LINE_LENGTH_LUSBAPI]; // название фирмы-изготовителя
                                                    // изделия
    BYTE DeviceName[NAME_LINE_LENGTH_LUSBAPI];  // название изделия
    BYTE SerialNumber[16];                      // серийный номер изделия
    BYTE Revision;                              // ревизия изделия
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI];  // строка комментария
};
```

## B.4. СТРУКТУРА INTERFACE\_INFO\_LUSBAPI

Вспомогательная структура *INTERFACE\_INFO\_LUSBAPI* содержит самую общую информацию об используемом интерфейсе для доступа к модулю. Данная структура описывается следующим образом:

```
struct INTERFACE_INFO_LUSBAPI
{
    BOOL Active; // флаг достоверности остальных полей структуры
    BYTE Name[NAME_LINE_LENGTH_LUSBAPI]; // название интерфейса
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI]; // строка комментария
};
```

## B.5. СТРУКТУРА MCU\_INFO\_LUSBAPI

Вспомогательная структура *MCU\_INFO\_LUSBAPI* содержит самую общую информацию об используемом исполнительном устройстве типа микроконтроллер (MCU). Данная структура описывается следующим образом:

```
template <class VersionType>
struct MCU_INFO_LUSBAPI
{
    BOOL Active; // флаг достоверности остальных полей структуры
    BYTE Name[NAME_LINE_LENGTH_LUSBAPI]; // название MCU
    double ClockRate; // тактовая частота работы MCU в кГц
    VersionType Version; // информация о программе MCU
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI]; // строка комментария
};
```

## B.6. СТРУКТУРА DSP\_INFO\_LUSBAPI

Вспомогательная структура *DSP\_INFO\_LUSBAPI* содержит информацию об используемом исполнительном устройстве типа DSP. Данная структура описывается следующим образом:

```
struct DSP_INFO_LUSBAPI
{
    BOOL Active; // флаг достоверности остальных полей структуры
    BYTE Name[NAME_LINE_LENGTH_LUSBAPI]; // название DSP
    double ClockRate; // тактовая частота работы DSP в кГц
    VERSION_INFO_LUSBAPI Version; // информация о драйвере DSP
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI]; // строка комментария
};
```

## B.7. СТРУКТУРА ADC\_INFO\_LUSBAPI

Вспомогательная структура *ADC\_INFO\_LUSBAPI* содержит самую общую информацию об используемом устройстве типа АЦП. Данная структура описывается следующим образом:

```
struct ADC_INFO_LUSBAPI
{
    BOOL Active; // флаг достоверности остальных полей структуры
    BYTE Name[NAME_LINE_LENGTH_LUSBAPI]; // название АЦП
    double OffsetCalibration[ADC_CALIBR_COEFS_QUANTITY_LUSBAPI];
    // корректировочные коэффициенты смещения нуля АЦП
    double ScaleCalibration[ADC_CALIBR_COEFS_QUANTITY_LUSBAPI];
    // корректировочные коэффициенты масштаба АЦП
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI]; // строка комментария
};
```

## B.8. СТРУКТУРА DAC\_INFO\_LUSBAPI

Вспомогательная структура *DAC\_INFO\_LUSBAPI* содержит самую общую информацию об используемом устройстве типа ЦАП. Данная структура описывается следующим образом:

```
struct DAC_INFO_LUSBAPI
{
    BOOL    Active;                // флаг достоверности остальных полей структуры
    BYTE    Name[NAME_LINE_LENGTH_LUSBAPI];    // название ЦАП
    double  OffsetCalibration[DAC_CALIBR_COEFS_QUANTITY_LUSBAPI];
    // корректировочные коэффициенты смещения нуля ЦАП
    double  ScaleCalibration[DAC_CALIBR_COEFS_QUANTITY_LUSBAPI];
    // корректировочные коэффициенты масштаба ЦАП
    BYTE    Comment[COMMENT_LINE_LENGTH_LUSBAPI];    // строка комментария
};
```

## B.9. СТРУКТУРА DIGITAL\_IO\_INFO\_LUSBAPI

Вспомогательная структура *DIGITAL\_IO\_INFO\_LUSBAPI* содержит самую общую информацию об используемых устройствах цифрового ввода-вывода. Данная структура описывается следующим образом:

```
struct DIGITAL_IO_INFO_LUSBAPI
{
    BOOL    Active;                // флаг достоверности остальных полей структуры
    BYTE    Name[NAME_LINE_LENGTH_LUSBAPI];    // название цифровой микросхемы
    WORD    InLinesQuantity;        // кол-во входных линий
    WORD    OutLinesQuantity;       // кол-во выходных линий
    BYTE    Comment[COMMENT_LINE_LENGTH_LUSBAPI];    // строка комментария
};
```