L-CARD

# Устройства для мобильных систем

# E-310

## Внешний специализированный модуль на шину **USB 1.1**

*Генератор аналоговых сигналов: 1 канал, от 0 до 10 МГц*
*Частотомер с программной установкой порога*
*Асинхронный ввод с АЦП: 4 канала, 10 бит*
*Конфигурируемые цифровые линии: до 11 входов/выходов*

## Библиотека *Lusbapi 3.4.*

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

## *Руководство программиста*

**Москва. Июнь 2011 г.**
*Ревизия документа A0*

## ООО *«Л КАРД»:*

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

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

## Адреса в Интернет:

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

## E-Mail:

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

## Представители в регионах:

| | | |
|---|---|---|
| Украина: | “ХОЛИТ Дэйта Системс, Лтд” | *www.holit.com.ua* |
| Санкт-Петербург: | ЗАО “АВТЭКС Санкт-Петербург” | *www.autex.spb.ru* |
| Санкт-Петербург: | Компания "Ниеншанц-Автоматика" | *www.nnz-ipc.ru* |
| Новосибирск: | ООО “Сектор Т” | *www.sector-t.ru* |
| Екатеринбург: | ООО “Авеон” | *www.aveon.ru* |
| Казань: | ООО “Шатл” | *www.shuttle.kazan.ru* |

# Оглавление

# 1. Введение

Данное описание предназначено для пользователей, собирающихся разрабатывать свои собственные приложения в операционной среде *Windows'98/2000/XP/Vista/7* для работы с многофункциональным модулем *E-310* от фирмы ООО "Λ Кард". Предварительно настоятельно рекомендуется ознакомиться с *"E-310. Руководство пользователя"*, где можно найти достаточно подробную техническую информацию о модуле, включая описание функциональной схемы, подключение входных сигналов, распиновка внешних разъёмов, характерные неисправности и многое другое.

Для модуля *E-310* фирма ООО "Λ Кард" поставляет **USB** драйвер устройства, готовую динамически подключаемую библиотеку `Lusbapi` с целым рядом законченных примеров. В качестве базового языка при написании библиотеки `Lusbapi` был выбран C++, а конкретнее, старый надёжный **Borland C++ 5.02**. Причём сама библиотека и все примеры поставляются вместе с исходными текстами, снабжёнными достаточно подробными комментариями. Штатная библиотека `Lusbapi` включает в себя множество разнообразных функций помогающих пользователю использовать все заложенные в модуль *E-310* возможности.

Модуль *E-310* разрабатывался с главной целью – обеспечить многофункциональную генерацию аналоговых сигналов. Для этого штатная библиотека `Lusbapi` содержит целый ряд функций, позволяющих организовывать разнообразный вывод сигналов в диапазоне от 0 Гц до 10 МГц с минимальным шагом около 3 Гц. Кроме того модуль позволяет измерять частоту входного сигнала, работать с обычными цифровыми линиями, производить оцифровку аналоговых сигналов с помощью встроенного АЦП и т.д. Мы надеемся, что описываемая ниже библиотека `Lusbapi` упростит и ускорит написание Ваших собственных *Windows*-приложений.

Весь пакет штатного программного обеспечения для работы с модулем *E-310* в среде *Windows'98/2000/XP/Vista/7* находится на прилагаемом к модулю фирменном CD-ROM в директории `\USB\Lusbapi`. **!!!ВНИМАНИЕ!!!** Далее по тексту данного описания все директории указаны относительно неё. Также весь штатный софт можно скачать с нашего сайта *www.lcard.ru* из раздела *"Библиотека файлов"*. Там из подраздела *"ПО для внешних модулей"* следует выбрать самораспаковывающийся архив `lusbapi`***XY***`.exe`, где ***X.Y*** обозначает номер версии программного обеспечения. На момент написания данного руководства последняя библиотека *Lusbapi* имеет версию **3.4**, а содержащий её архив называется *lusbapi34.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 обращать к:

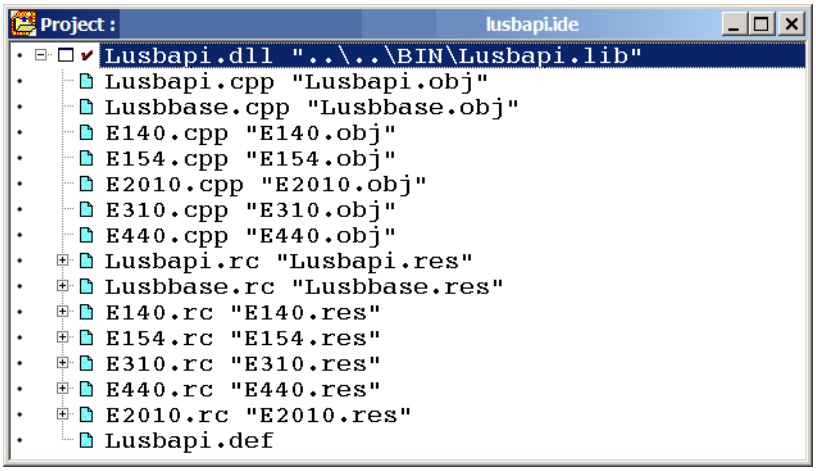
- *"E-310. Руководство пользователя"*;
- *"E-310. Библиотека Lusbapi. История дополнений и изменений"*.

## 2.2. *Подключение модуля E-310 к компьютеру*

Все подробности по процедуре аппаратного подключения модуля *E-310* к компьютеру конечного пользователя и надлежащей установке **USB** драйверов можно найти в *"E-310. Руководство пользователя, § 4 "Инсталляция и настройка"*.

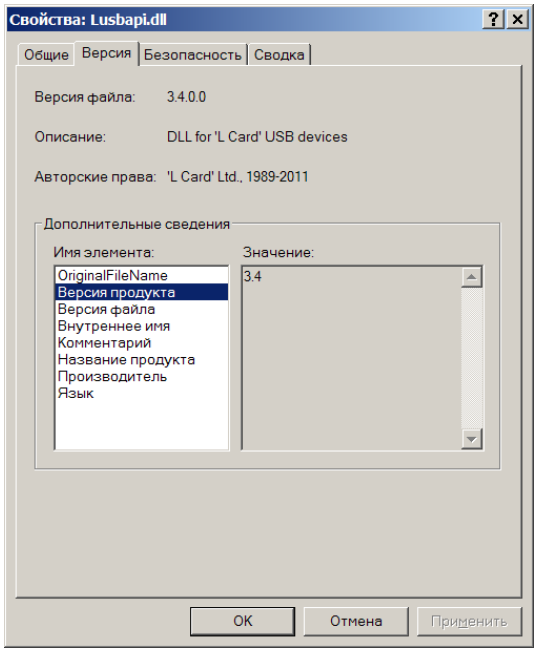
## 2.3. *Библиотека Lusbapi*

Штатная библиотека `Lusbapi` написана с использованием весьма доступного языка программирования **Borland C++ 5.02**. Кроме модуля *E-310* в библиотеке также осуществлена поддержка модулей типа *E-154*, *E14-140*, *E14-440* 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 *E-310*. В дальнейшем, используя этот указатель, можно осуществлять доступ ко всем интерфейсным функциям штатной DLL библиотеки (см. исходные тексты примеров). **!!!Внимание!!!** Все интерфейсные функции, 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 файл библиотеки `Lusbapi.dll` включена информация о текущей версии DLL. Для получения в Вашем приложении сведений о данной версии можно использовать вторую из экспортируемых функций из штатной библиотеки: *GetDllVersion()*. Кроме того, оперативно выявить текущую версию библиотеки можно, используя штатные возможности *Windows*. Например, в '*Windows Explorer*' щелкните правой кнопкой мышки над файлом 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 без труда прочитать текущий номер версии библиотеки. Выглядит это примерно так:



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:

- `\E-310\Examples\Borland C++ 5.02`;
- `\E-310\Examples\Borland C++ Builder 5.0`;
- `\E-310\Examples\Borland Delphi 6.0`;
- `\E-310\Examples\MicroSoft Visual C++ 6.0`.

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

- создать файл проектов (например, для **Borland C++ 5.02**, `test.ide`);
- добавить в проект файл библиотеки импорта `\DLL\Lib\Borland\LUSBAPI.LIB`;
- создать и добавить в проект Ваш файл с будущей программой (например, `test.cpp`);
- включить в начало вашего файла заголовочный файл `#include "LUSBAPI.H"`, содержащий описание интерфейса модуля *E-310*;
- в принципе, с помощью функции *GetDllVersion()*, желательно сравнить версию используемой DLL библиотеки с версией текущего программного обеспечения;
- 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 DLL библиотеки `Lusbapi.dll`.

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

1. Динамическая загрузка библиотеки `Lusbapi` на этапе выполнения приложения. Подробности смотри в исходных текстах примера из директории `\E-310\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 *E-310* 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. Микроконтроллер модуля

На модуле E-310 в качестве главного исполнительного устройства используется микроконтроллер (MCU) типа *ARM AT91SAM7S64* от фирмы *Atmel Corporation*. MCU отвечает за корректное функционирование **USB** интерфейса модуля, а также разбирает все пользовательские команды, поступающие из компьютера и задающие различные режимы работы модуля. Особенностью программного софта, заложенного в основу работы MCU, является его двухкомпонентность. Т.е. он состоит как бы из двух частей: основной программы (`Firmware`) и загрузчика (`BootLoader`). Фирменный загрузчик, как впрочем, и основная программа, '*заливается*' в MCU на этапе наладки модуля E-310 в ООО "Λ Кард" и конечный пользователь не имеет возможности его обновления без специального прошивочного кабеля. Но при этом `BootLoader` предоставляет возможность безболезненной перепрошивки `Firmware` модуля по **USB** шине, что является крайне удобным при обновлении версий основной программы. Самую последнюю версию `Firmware` MCU всегда можно скачать с нашего сайта *www.lcard.ru* из раздела *"Библиотека файлов"*. Там из подраздела *"Firmware и BIOS"* следует выбрать архив `e310fw_XY.zip`, где **`X.Y`** означает номер версии основной программы MCU для модуля *E-310*. На момент написания данного руководства этот архив имеет имя *e310fw_10.zip*.

## *2.5. Возможные проблемы при работе с модулем*

Перед началом работы со штатным программным обеспечением модуля *E-310*, 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. Также их можно попробовать скачать из Интернета с сайта производителя.

# 3. Описание библиотеки Lusbapi

В настоящем разделе приведены достаточно подробные описания констант, структур и интерфейсных функций, входящих в состав штатной DLL библиотеки `Lusbapi` для модуля *E-310*.

## *3.1. Общие принципы работы с модулем*

Целью штатной DLL библиотеки `Lusbapi`, поставляемой с модулем *E-310*, 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 *E-310*.

Перед началом работы с библиотекой `Lusbapi` в пользовательской программе должны быть сделать следующие объявление (как минимум):

```
ILE310 *pModule;                    // указатель на интерфейс модуля E-310
MODULE_DESCRIPTION_E310 md;         // структура служебной информации о модуле
```

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

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

После этого, используя уже полученный указатель на интерфейс модуля, следует проинициализировать доступ к соответствующему виртуальному слоту, к которому подключён модуль *E-310*. Для этого предусмотрена интерфейсная функция *OpenLDevice()*. Если нет ошибки при выполнении этой функции, то можно быть уверенным, что устройство типа *E-310* обнаружено в выбранном виртуальном слоте.

На следующем этапе, в принципе, можно прочитать служебную информацию о модуле. Эта функция позволяет получить такую информацию как: название модуля, серийный номер модуля, версии прошивок MCU и т.д. Если функция не вернула ошибку, то это означает, что информация о модуле успешно получена и можно продолжать работу.

В общем-то, предварительный этап работы с модулем *E-310* можно считать успешно завершённым. Теперь можно спокойно управлять всей доступной периферией на модуле с помощью соответствующих интерфейсных функций библиотеки `Lusbapi` и организовывать различные режимы работы модуля.

В качестве примера приведем исходный текст, а вернее сказать '*скелет*', небольшой консольной программы для работы с модулем *E-310*:

```
#include <stdlib.h>
#include <stdio.h>
#include "Lusbapi.h"            // заголовочный файл библиотеки Lusbapi
ILE310 *pModule;                // указатель на интерфейс модуля
MODULE_DESCRIPTION_E310 md;     // структура с информацией о модуле
BYTE UsbSpeed;                  // скорость работы шины USB
char ModuleName[0x10];          // название модуля
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<ILE310 *>(CreateLInstance("e310"));
```

```
   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
   }
   // теперь отобразим полученную скорость работы шины USB
   printf(" USB is in %s\n", UsbSpeed ? "High-Speed Mode
                      (480 Mbit/s)" : "Full-Speed Mode (12 Mbit/s)");
   // 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
   }
   // на всякий случай проверим: этот модуль - 'E-310'?
   if(strcmp(ModuleName, "E-310"))
   {
      printf(" В нулевом виртуальном слоте не 'E-310'\n");
      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
   }
   // отображаем информацию о модуле
   printf("Модуль E-310 (серийный номер %s) полностью готов к\
                                 работе!", md.Module.SerialNumber);
   // далее можно располагать функции для непосредственного
   // управления модулем, например, по сбору данных с АЦП
   . . . . . .
   // завершим работу с модулем
   if(!pModule->ReleaseLInstance())
   {
      printf("Не выполнена функция ReleaseLInstance()!");
      return 1;                  //выйдем из программы с ошибкой
   }

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

## *3.2. Константы*

В исходных текстах приложения при работе с модулем *E-310* настоятельно рекомендуется использовать предопределёнными константами. Это весьма повышает *читаемость* и *понимаемаемость* исходных текстов, а также значительно облегчает сопровождение программ. Все доступные константы расположены в файле `\DLL\Include\Lusbapi.h`.

## 3.3. *Структуры*

В данном разделе приведены основные типы структур, которые применяются в библиотеке `Lusbapi` при работе с модулем *E-310*.

### 3.3.1. Структура MODULE_DESCRIPTION_E310

Структура *MODULE_DESCRIPTION_E310* описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct MODULE_DESCRIPTION_E310
{
    MODULE_INFO_LUSBAPI      Module;        // общая информация о модуле
    INTERFACE_INFO_LUSBAPI   Interface;     // информация об интерфейсе
    MCU_INFO_LUSBAPI<MCU_VERSION_INFO_LUSBAPI> Mcu;   // информация о MCU
    ADC_INFO_LUSBAPI         Adc;           // информация о АЦП
    DIGITAL_IO_INFO_LUSBAPI DigitalIo;      // информация о цифровых линиях
};
```

В данной структуре представлена самая общая служебная информация об используемом экземпляре модуля *E-310*. 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 А*.

### 3.3.2. Структура GENERATOR_PARS_E310

Структура *GENERATOR_PARS_E310* представляет собой довольно обширную группу параметров, задающих режим работы генератора. Данная структура описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct GENERATOR_PARS_E310
{
    BOOL GeneratorEna;       // [in]      текущее состояние работы генератора
    double StartFrequency;   // [in-out]  начальная частота сканирования в кГц
    double FinalFrequency;   // [out]     конечная частота в сканирования кГц
    double FrequencyIncrements; // [in-out]  частота приращения в кГц
    WORD NumberOfIncrements;    // [in-out]  кол-во приращений частоты сканирования
    INCREMENT_INTRERVAL_PARS_E310 IncrementIntervalPars;   // [in-out] структура с
                                                           // параметрами частотного приращения
    double MasterClock;          // [in-out] частота тактирующего сигнала генератора в кГц
    BYTE MasterClockSource;      // [in-out] источник тактирующего сигнала генератора:
                                 // внутренний или внешний
    BYTE CyclicAutoScanType;     // [in-out] тип циклического автосканирования:
                                 // нет циклического сканирования, 'пила' или 'треугольник'
    BYTE IncrementType;      // [in-out] тип инкрементации частоты генератора:
                 // внутренняя (автоматическая) или с помощью управляющей линии "CTRL"
    BYTE CtrlLineType;       // [in-out] тип линии "CTRL" для (управления инкрементацией
                 // частоты)/старта генератора: внутренняя (от MCU) или внешняя
    BYTE InterrupLineType;       // [in-out] тип линии "INTERRUPT" для останова
                                 // работы генератора: внутренняя (от MCU) или внешняя
    BOOL SquareWaveOutputEna;    // [in-out]  разрешение цифрового сигнала
                                 // генератора на выходе "Меандр"
    BOOL SynchroOutEna;          // [in-out]  разрешение синхросигнала генератора
                                 // на выходной линии "SYNCOUT"
   BYTE SynchroOutType;      // [in-out]  тип формирования синхросигнала генератора:
                 // при каждом приращении частоты или только по окончании сканирования
```

```
    ANALOG_OUTPUTS_PARS_E310 AnalogOutputsPars;  // [in-out] параметры работы
                                                 // аналоговых выходов генератора
};
```

Перед началом работы с генератором необходимо заполнить поля данной структуры и передать ее в модуль с помощью интерфейсной функции *SET_GENERATOR_PARS()*. В описании этой функции подробно прокомментированы смысл и назначение всех полей данной структуры. Также при необходимости можно считать из модуля текущие параметры функционирования генератора, используя интерфейсную функцию *GET_GENERATOR_PARS()*.

### 3.3.3. Структура INCREMENT_INTRERVAL_PARS_E310

Структура `INCREMENT_INTRERVAL_PARS_E310` представляет собой группу параметров, используемых для задания параметров частотного приращения сканирования. Данная структура описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct INCREMENT_INTRERVAL_PARS_E310
{
    BYTE BaseIntervalType;   // [in-out]   тип базового интервала приращения, который
                             // может быть кратен: периоду MCLK или периоду
    BYTE MultiplierIndex;    // [in-out]   индекс умножителя для базового
                             // интервала приращения
    double MultiplierValue;  // [out]  величина умножителя для базового интервала
                             // приращения: 1, 5, 100 или 500
    WORD BaseIntervalsNumber;   // [in-out]   кол-во базовых интервалов в
                                // интервале приращения
    double Duration;         // [out] общая длительность приращения в мс
                             // (только для интервала приращений по MCLK, иначе 0)
};
```

Структура `INCREMENT_INTRERVAL_PARS_E310` входит составной частью в структуру *GENERATOR_PARS_E310*.

### 3.3.4. Структура ANALOG_OUTPUTS_PARS_E310

Структура `ANALOG_OUTPUTS_PARS_E310` представляет собой группу параметров выходных аналалоговых сигналов. Данная структура описана в файле `\DLL\Include\Lusbapi.h` и представлена ниже:

```
struct ANALOG_OUTPUTS_PARS_E310
{
    BYTE SignalType;            // [in-out] тип аналогового сигнала на выходах
                                // 10 и 50 Ом: синусоидальный или треугольный
    BYTE GainIndex;             //[in-out]  индекс усиления выходного тракта генератора
    double GaindB;              // [out]   усиление выходного тракта генератора в дБ
    double Output10OhmInV;      // [out]   амплитуда сигнала на выходе 10 Ом в В
    double Output10OhmIndB;     // [out]   амплитуда сигнала на выходе 10 Ом в дБ
    double Output10OhmOffset;   // [in-out]   величина внутреннего смещения
                                // на выходе 10 Ом в В
    BYTE Output10OhmOffsetSource;  // [in-out]   тип смещения на выходе 10 Ом:
                                   // внутреннее или внешнее
    double Output50OhmInV;      // [out]   амплитуда сигнала на выходе 50 Ом в В
    double Output50OhmIndB;     // [out]   амплитуда сигнала на выходе 50 Ом в дБ
};
```

Структура `ANALOG_OUTPUTS_PARS_E310` входит составной частью в структуру *GENERATOR_PARS_E310*.

### 3.3.5. Структура FM_PARS_E310

Структура FM_PARS_E310 представляет собой группу параметров, используемых для задания режимов работы измерителя частоты (FM). Данная структура описана в файле \DLL\Include\LusbapiTypes.h и представлена ниже:

```
struct FM_PARS_E310
{
    BOOL FmEna;                 // [out]   текущее состояние работы измерителя частоты
    BYTE Mode;                  // [in-out]  режим работы измерителя частоты
    BYTE InputDivider;          // [in-out]  управление входным делителем частоты 1/8
    BYTE BaseClockRateDivIndex;  // [in-out]    индекс делителя базовой
                                 // тактовой измерителя частоты
    double Offset;              // [in-out]  смещение порога измерителя частоты в В
    DWORD  ClockRate;           // [out]     рабочая тактовая частота счётчика FM в Гц
    DWORD  BaseClockRate; // [const]  базовая тактовая частота счётчика FM: 25 000 000 Гц
};
```

Перед началом работы с измерителем частоты необходимо заполнить поля данной структуры и передать ее в модуль с помощью интерфейсной функции *SET_FM_PARS()*. В описании этой функции подробно прокомментированы смысл и назначение всех полей данной структуры. Также при необходимости можно считать из модуля текущие параметры функционирования измерителя частоты, используя интерфейсную функцию *GET_FM_PARS()*.

### 3.3.6. Структура FM_SAMPLE_E310

Структура FM_SAMPLE_E310 содержит информацию о частоте измеряемого сигнала. Данная структура описана в файле \DLL\Include\LusbapiTypes.h и представлена ниже:

```
struct FM_SAMPLE_E310
{
    BOOL IsActual;              // [out]  признак действительности полученных данных
    double Frequency;           // [out]  частота измеряемого сигнала в кГц
    double Period;              // [out]  период измеряемого сигнала в мс
    double DutyCycle;           // [out]  скважность измеряемого сигнала в мс
};
```

Структура FM_SAMPLE_E310 используется совместно с интерфейсной функции *FM_SAMPLE()*.

### 3.3.7. Структура ADC_PARS_E310

Структура ADC_PARS_E310 представляет собой группу параметров, задающих режим работы АЦП. Данная структура описана в файле \DLL\Include\LusbapiTypes.h и представлена ниже:

```
struct ADC_PARS_E310
{
    BYTE AdcStartSource;   // [in-out]  источник сигнала запуска АЦП:
                           // внутренний или внешний
    BYTE ChannelsMask;     // [in-out]  битовая маска активных каналов (младшие 4 бита)
    double InputRange;     // [out]     входной диапазон АЦП в В
};
```

Перед началом работы с генератором необходимо заполнить поля данной структуры и передать ее в модуль с помощью интерфейсной функции *SET_ADC_PARS()*. В описании этой функции подробно прокомментированы смысл и назначение всех полей данной структуры. Также при необходимости можно считать из модуля текущие параметры функционирования генератора, используя интерфейсную функцию *GET_ADC_PARS()*.

### 3.3.8. Структура ADC_DATA_E310

Структура ADC_DATA_E310 содержит информацию об отсчетах АЦП. Данная структура описана в файле \DLL\Include\LusbapiTypes.h и представлена ниже:

```
struct ADC_DATA_E310
{
    SHORT DataInCode[ADC_CHANNEL_QUANTITY_E310];  // [out] массив данных
                                                  // с активных каналов АЦП в кодах
    Double DataInV[ADC_CHANNEL_QUANTITY_E310];    // [out] массив данных
                                                  // с активных каналов АЦП в В
};
```

Структура FM_SAMPLE_E310 используется совместно с интерфейсной функции *GET_ADC_DATA()*.

### 3.3.9. Структура USER_FLASH_E310

Структура *USER_FLASH_E310* описана в файле \DLL\Include\Lusbapi.h и представлена ниже:

```
struct USER_FLASH_E310
{
    BYTE Buffer[USER_FLASH_SIZE_E310];        // размер данной структуры в байтах
};
```

Данная структура предназначена для хранения или считывания пользовательской информации. Для работы с ней выделена область размером *USER_FLASH_SIZE_E310* байт в ППЗУ микроконтроллера. Используется в функциях *READ_FLASH_ARRAY()* и *WRITE_FLASH_ARRAY()*.

### 3.3.10. 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
    DWORD ErrorNumber;             // номер последней ошибки библиотеки Lusbapi
};
```

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

## 3.4. *Функции общего характера*

### 3.4.1. Получение версии библиотеки

**Формат:** **DWORD** ***GetDllVersion***(*void*)

**Назначение:**

Данная функция является одной из *двух* экспортируемых из штатной библиотеки `Lusbapi` функцией. Она возвращает текущую версию используемой библиотеки. Формат номера версии следующий:

| Битовое поле | Назначение |
|---|---|
| <31..16> | Старшее слово версии библиотеки |
| <15..0> | Младшее слово версии библиотеки |

Рекомендованную последовательность вызовов интерфейсных функций см. *§ 3.1. "Общие принципы работы с модулем"*.

Передаваемые параметры: нет.

**Возвращаемое значение:** номер версии библиотеки `Lusbapi`.

### 3.4.2. Получение указателя на интерфейс модуля

**Формат:** **LPVOID** ***CreateLInstance***(*PCHAR const DeviceName*)

**Назначение:**

Данная функция должна обязательно вызываться в начале каждой пользовательской программы, работающей с модулями *E-310*. Она является одной из *двух* экспортируемых из штатной библиотеки `Lusbapi` функцией и возвращает указатель на интерфейс для устройства с названием *DeviceName*. Все последующие интерфейсные функции штатной библиотеки вызываются именно через этот возвращаемый указатель.

Рекомендованную последовательность вызовов интерфейсных функций см. *§ 3.1. "Общие принципы работы с модулем"*

**Передаваемые параметры:**

- *DeviceName* – строка с названием устройства (для данного модуля это – “E310”) .

**Возвращаемое значение:** В случае успеха — указатель на интерфейс, иначе — **NULL**.

### 3.4.3. Завершение работы с интерфейсом модуля

| **Формат:** | **BOOL** | ***ReleaseLInstance***(*void*) |
|---|---|---|

**Назначение:**

Данная интерфейсная функция реализует корректное высвобождение интерфейса модуля *E-310*, проинициализированного с помощью функции *CreateLInstance()*. Используется для аккуратного завершения сеанса работы с модулем (если предварительно удачно выполнилась функция *CreateLInstance()*). **!!!Внимание!!!** Данная функция должна обязательно вызываться в приложении перед его завершением во избежания утечки ресурсов *Windows*.

Рекомендованную последовательность вызовов интерфейсных функций см. *§ 3.1. "Общие принципы работы с модулем"*.

**Передаваемые параметры:** нет.

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

### 3.4.4. Инициализация доступа к модулю

| **Формат:** | **BOOL** | ***OpenLDevice***(*WORD VirtualSlot*) |
|---|---|---|

**Назначение:**

С программной точки зрения, не вдаваясь в излишние тонкости, подсоединенный к компьютеру модуль *E-310* 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 *E-310* находится в заданном виртуальном слоте. Если функция ***OpenLDevice()*** успешно выполнилась для заданного виртуального слота, то можно переходить непосредственно к загрузке модуля и его последующему управлению с помощью соответствующих интерфейсных функций библиотеки `Lusbapi`.

Рекомендованную последовательность вызовов интерфейсных функций см. *§ 3.1. "Общие принципы работы с модулем"*.

**Передаваемые параметры:**

- *VirtualSlot* – номер виртуального слота, к которому, как предполагается, подключен модуль *E-310*.

**Возвращаемое значение:**
*TRUE* – модуль *E-310* находится в выбранном виртуальном слоте и можно переходить непосредственно к загрузке модуля;
*FALSE* – устройства типа модуль *E-310* в выбранном виртуальном слоте нет. Следует попробовать другой номер виртуального слота.

### 3.4.5. Завершение доступа к модулю

| **Формат:** | **BOOL** | ***CloseLDevice***(*void*) |
|---|---|---|

**Назначение:**

Данная интерфейсная функция прерывает всякое взаимодействие с *текущим* виртуальным слотом, к которому подключён модуль. При этом данный виртуальный слот аккуратно закрывается и выполняется освобождение связанных с ним ресурсов *Windows*. После успешного выполнения данной функции всякий доступ к модулю *E-310* становится невозможным. Для возобновления нормального доступа к устройству следует вновь воспользоваться интерфейсной функцией *OpenLDevice()*. Таким образом, эта функция, по своей сути, противоположна интерфейсной функции *OpenLDevice()*. Фактически данная функция используется в таких интерфейсных функциях как *OpenLDevice()* и *ReleaseLInstance()*.

**Передаваемые параметры:** нет.

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

### 3.4.6. Получение названия модуля

| **Формат:** | **BOOL** | ***GetModuleName***(*PCHAR const ModuleName*) |
|---|---|---|

**Назначение:**

Данная вспомогательная интерфейсная функция позволяет получить название подключенного к слоту модуля. Массив под название модуля *ModuleName* (не менее 6 символов плюс признак конца строки '\0', т.е. нулевой байт) должен быть заранее определен.

Рекомендованную последовательность вызовов интерфейсных функций см. *§ 3.1. "Общие принципы работы с модулем"*.

**Передаваемые параметры:**

- *ModuleName* – возвращается строка, не менее 6 символов, с названием модуля (в нашем случае это должна быть строка "`E-310`").

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

### 3.4.7. Получение скорости работы модуля

| **Формат:** | **BOOL** | ***GetUsbSpeed***(*BYTE * const UsbSpeed*) |
|---|---|---|

**Назначение:**

Данная функция позволяет определить, на какой скорости шина **USB** работает с модулем.

**Передаваемые параметры:**

- *UsbSpeed* – возвращаемое значение этой переменной может принимать следующее:
  - ✓ 0 – модуль работает с **USB** шиной в режиме *Full-Speed Mode* (12 Мбит/с)
  - ✓ 1 – модуль работает с **USB** шиной в режиме *High-Speed Mode* (480 Мбит/с).

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

### 3.4.8. Получение дескриптора модуля

| **Формат:** | **HANDLE** | ***GetModuleHandle***(*void*) |
|---|---|---|

**Назначение:**
Данная функция позволяет получить дескриптор (handle) используемого модуля *E-310*.

**Передаваемые параметры:** нет

**Возвращаемое значение:** В случае успеха – дескриптор модуля *E-310*;
в противном случае – **INVALID_HANDLE_VALUE**.

### 3.4.9. Получение описания ошибок выполнения функций

| **Формат:** | **BOOL** | ***GetLastErrorInfo***(*LAST_ERROR_INFO_LUSBAPI * const LastErrorInfo*) |
|---|---|---|

**Назначение:**
Если в процессе работы с библиотекой `Lusbapi` какая-нибудь интерфейсная функция штатной библиотеки вернула ошибку, то только непосредственно после этого с помощью вызова данной интерфейсной функции можно получить краткое толкование произошедшего сбоя. Для некоторых функций, например *ReadData()*, при выявлении причины ошибки может потребоваться дополнительный вызов стандартной *Windows API* функции ***GetLastError()*** для классификации ошибок самой *Windows*.

**Передаваемые параметры:**
- *LastErrorInfo* – указатель на структуру типа *LAST_ERROR_INFO_LUSBAPI*, в которой возвращается краткое описание и номер последней ошибки.

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

## 3.5. *Функции для работы с генератором*

Модуль *E-310* предназначен для многофункциональной генерации сигналов аналогового (синусоидальной или треугольной формы) и цифрового типа с возможностью автоматического сканирования (перестраивания) по частоте. При использовании внутреннего тактового сигнала с генератора можно получать сигнал в диапазоне от 0 Гц до 10 МГц с минимальным шагом около 3 Гц. Кроме того, модуль предоставляет достаточно широкие возможности синхронизации старта и приращения частоты сканирования: по программным событиям (например, по таймеру), по внешнему сигналу старта, по внешнему сигналу приращения, немедленное прерывание сканирования по программному событию или по внешнему сигналу. Также возможна конфигурация выводов сигналов старта, прерывания и приращения частоты сканирования на внешний разъём для синхронизации работы нескольких модулей. Синхронизация может осуществляться от одного модуля являющегося "*ведущим*" на несколько "*ведомых*". Каждый модуль может быть "*ведомыми*" по отношению к другому модулю или независимому сигналу синхронизации, при входы синхронизации TTL 5 В совместимы.

На модуле доступны два аналоговых выхода и один цифровой выход:

- Выход на согласованную нагрузку 50 Ом с программно устанавливаемой амплитудой сигнала от +4 дБ до -30 дБ;
- Выход на несогласованную нагрузку 600 Ом (минимум 10 Ом) с программно устанавливаемой амплитудой сигнала от +10 дБ до -24 дБ;
- Выход прямоугольного цифрового сигнала "Меандр" (совместимый с TTL +5 В) синхронного с аналоговым сигналом (переключение уровней происходит в моменты перехода синусоиды "*через ноль*", фронт не более 10 нс, при токе нагрузки до 20 мА).

### 3.5.1. Запуск генератора

| **Формат:** | **BOOL** | ***START_GENERATOR***(*void*) |
|---|---|---|
| **Назначение:**<br>Данная функция запускает генератор модуля *E-310*. Перед любым запуском генератора настоятельно рекомендуется выполнять функцию *STOP_GENERATOR()*, после чего можно установить требуемые параметры работы генератора с помощью интерфейсной функции *SET_GENERATOR_PARS()*. | | |
| **Передаваемые параметры:** нет | | |
| **Возвращаемое значение:** | *TRUE* – функция успешно выполнена;<br>*FALSE* – функция выполнена с ошибкой. | |

### 3.5.2. Останов генератора

| **Формат:** | **BOOL** | ***STOP_GENERATOR***(*void*) |
|---|---|---|
| **Назначение:**<br>Данная функция останавливает работу генератора модуля *E-310*. Настоятельно рекомендуется применять эту функцию перед каждым запуском генератора с помощью функции *START_GENERATOR()*. | | |
| **Передаваемые параметры:** нет | | |
| **Возвращаемое значение:** | *TRUE* – функция успешно выполнена;<br>*FALSE* – функция выполнена с ошибкой. | |

### 3.5.3. Установка параметров работы генератора

**Формат:** **BOOL** ***SET_GENERATOR_PARS***(*GENERATOR_PARS_E310 * const GenPars*)

**Назначение:**

Данная функция передает в *E-310* всю необходимую информацию, которая используется модулем для организации работы генератора. Всю нужную информацию описываемая интерфейсная функция извлекает из полей передаваемой структуры типа *GENERATOR_PARS_E310*. Собственно, использование модулем именно этой переданной информации начинается только после выполнения интерфейсной функции *START_GENERATOR()*. Данную функцию не слдеует вызывать процессе работы генератора, а только после выполнения функции *STOP_GENERATOR()*.

Формат структуры *GENERATOR_PARS_E310* приведен ранее в *§ 3.3.2. "Структура GENERATOR_PARS_E310"*, а смысл и назначение отдельных её полей достаточно подробно описано ниже.

- Поле GenPars->GeneratorEna. Чтение. В данном поле возвращается текущее состояние работы генератора.
- Поле GenPars->StartFrequency. Запись–Чтение. Данное поле передаёт в функцию требуемое значение начальной частоты генератора в *кГц*. После выполнения функции в этом поле возвращается реально установленное значение начальной частоты.
- Поле GenPars->FinalFrequency. Чтение. После выполнения функции в этом поле возвращается конечная частота генератора в *кГц*.
- Поле GenPars->FrequencyIncrements. Запись–Чтение. Данное поле передаёт в функцию требуемое значение частоты приращения в *кГц*. После выполнения функции в этом поле возвращается реально установленное значение частоты приращения.
- Поле GenPars->NumberOfIncrements. Запись–Чтение. Данное поле передаёт в функцию кол-во приращений частоты сканирования. Данное значение может находиться в диапазоне от 0 до 4095.
- Поле GenPars->IncrementIntervalPars. Запись–Чтение. Это поле является вложенной структурой типа *INCREMENT_INTRERVAL_PARS_E310*, в которой сгруппированы все параметры, относящиеся к частотному приращению:
  - Поле AdcPars->IncrementIntervalPars.BaseIntervalType. Запись–Чтение. Данное поле определяет тип базового интервала приращения. Базовый интервал может быть кратен либо периоду тактовой частоты генератора, либо периоду выводимого сигнала.
  - Поле AdcPars->IncrementIntervalPars.MultiplierIndex. Запись–Чтение. Данное поле задает индекс умножителя для базового интервала приращения. Это поле может принимать следующие значения: 0, 1, 2 или 3.
  - Поле AdcPars->IncrementIntervalPars.MultiplierValue. Чтение. В данном поле возвращается значение умножителя для базового интервала приращения. В этом поле может возвращаться только следующие значения: 1, 5, 100 или 500.
  - Поле AdcPars->IncrementIntervalPars.BaseIntervalsNumber. Запись–Чтение. Данное поле передаёт в функцию кол-во базовых интервалов в интервале приращения. Это поле может принимать значения в диапазоне от 2 до 2047.
  - Поле AdcPars->IncrementIntervalPars.Duration. Чтение. В данном поле возвращается общая длительность приращения в *мс*. Это действительно только для интервала частотного приращения по периоду тактовой частоты генератора, иначе 0.
- Поле GenPars->MasterClock. Запись–Чтение. Данное поле задает величину частоты в *кГц* для внешнего тактового сигнала генератора. Данное значение может находиться в диапазоне от 0.001 до 50000.

- Поле GenPars->MasterClockSource. Запись–Чтение. Данное поле задаёт источник тактирующего сигнала генератора. Тактирующий сигнал может быть либо внутренним, либо внешним.
- Поле GenPars->CyclicAutoScanType. Запись–Чтение. Данное поле задаёт тип циклического автосканирования выходного сигнала. Пока реализованы следующие возможности: нет циклического сканирования, '*пила*' или '*треугольник*'.
- Поле GenPars->IncrementType. Запись–Чтение. Данное поле задаёт тип приращения частоты генератора. Приращение частоты может быть либо автоматическое (внутреннее), либо под управлением линии "*CTRL*".
- Поле GenPars->CtrlLineType. Запись–Чтение. Данное поле задаёт тип управляющей линии "*CTRL*", которая используется для запуска генератора, либо для инкрементации частоты. Управление этой линией может быть либо внутренним (от микроконтроллера), либо внешним.
- Поле GenPars->InterrupLineType. Запись–Чтение. Данное поле задаёт тип управляющей линии " *INTERRUPT*", которая используется для останова (прерывания) работы генератора. Управление этой линией может быть либо внутренним (от микроконтроллера), либо внешним.
- Поле GenPars->SquareWaveOutputEna. Запись–Чтение. Данное поле разрешает/запрещает сигнал на выходной цифровой линии "*Меандр*".
- Поле GenPars->SynchroOutEna. Запись–Чтение. Данное поле разрешает/запрещает синхросигнал генератора на выходной линии "*SYNCOUT*".
- Поле GenPars->SynchroOutType. Запись–Чтение. Данное поле задаёт тип формирования синхросигнала генератора. Синхросигнал может формироваться либо при каждом приращении частоты, либо только по окончании процесса сканирования.
- Поле GenPars->IncrementIntervalPars. Запись–Чтение. Это поле является вложенной структурой типа *INCREMENT_INTRERVAL_PARS_E310*, в которой сгруппированы все параметры, относящиеся к частотному приращению:
- Поле GenPars->AnalogOutputsPars. Запись–Чтение. Это поле является вложенной структурой типа *ANALOG_OUTPUTS_PARS_E310*, в которой сгруппированы все параметры аналоговых выходов:
  - Поле AdcPars->AnalogOutputsPars.SignalType. Запись–Чтение. Данное поле определяет тип аналогового сигнала на выходах 10 Ом и 50 Ом. Этот сигнал может быть либо синусоидальный, либо треугольный.
  - Поле AdcPars->AnalogOutputsPars.GainIndex. Запись–Чтение. Данное поле задает индекс усиления выходного тракта генератора. Данное поле может принимать значения в диапазоне от 0 до 13.
  - Поле AdcPars->AnalogOutputsPars.GaindB. Чтение. В данном поле возвращается значение усиление выходного тракта генератора в *дБ*.
  - Поле AdcPars->AnalogOutputsPars.Output10OhmInV. Чтение. В данном поле возвращается значение амплитуды сигнала на выходе 10 Ом в *В*.
  - Поле AdcPars->AnalogOutputsPars.Output10OhmIndB. Чтение. В данном поле возвращается значение амплитуды сигнала на выходе 10 Ом в *дБ*.
  - Поле AdcPars->AnalogOutputsPars.Output10OhmOffset. Запись–Чтение. Данное поле задает величину внутреннего смещения в *В* на выходе 10 Ом. Данное поле может принимать значения в диапазоне от -4 до 4. После выполнения функции в этом поле возвращается реально установленное значение внутреннего смещения.
  - Поле AdcPars->AnalogOutputsPars.Output10OhmOffsetSource. Запись–Чтение. Данное поле задает тип смещения на выходе 10 Ом. Смещение может быть либо внутренним, либо внешним.

- Поле AdcPars->AnalogOutputsPars.Output50OhmInV. Чтение. В данном поле возвращается значение амплитуды сигнала на выходе 50 Ом в *В*.
- Поле AdcPars->AnalogOutputsPars.Output50OhmIndB. Чтение. В данном поле возвращается значение амплитуды сигнала на выходе 50 Ом в *дБ*.

**Передаваемые параметры:**

- *GenPars* – адрес структуры типа *GENERATOR_PARS_E310* с требуемыми параметрами функционирования генератора.

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

### 3.5.4. Получение текущих параметров работы генератора

**Формат:** **BOOL** ***GET_GENERATOR_PARS***(*GENERATOR_PARS_E310 * const GenPars*)

**Назначение:**

Данная функция считывает из модуля *E-310* всю текущую информацию, которая используется при работе с генератором.

**Передаваемые параметры:**

- *GenPars* – адрес структуры типа *GENERATOR_PARS_E310* с полученными из модуля текущими параметрами функционирования генератора.

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

## 3.6. *Функции для работы с частотомером*

Частотомер модуля *E-310* предназначен для измерения частоты входного сигнала в диапазоне от 0 Гц до 96 МГц. Допустимое напряжение входного сигнала ±5 В. Пользователь может программным образом устанавливать входной порог частотомера, а также делить частоту входного сигнала на 8.

### 3.6.1. Запуск частотомера

| **Формат:** | **BOOL** | ***START_FM***(*void*) |
|---|---|---|
| **Назначение:**<br>Данная функция запускает частотомер модуля *E-310*. Перед любым запуском частотомера настоятельно рекомендуется выполнять функцию *STOP_FM()*, после чего можно установить требуемые параметры работы частотомера с помощью интерфейсной функции *SET_FM_PARS()*. | | |
| **Передаваемые параметры:** нет | | |
| **Возвращаемое значение:** | *TRUE* – функция успешно выполнена;<br>*FALSE* – функция выполнена с ошибкой. | |

### 3.6.2. Останов частотомера

| **Формат:** | **BOOL** | ***STOP_FM***(*void*) |
|---|---|---|
| **Назначение:**<br>Данная функция останавливает работу частотомера модуля *E-310*. Настоятельно рекомендуется применять эту функцию перед каждым запуском частотомера с помощью функции *START_FM()*. | | |
| **Передаваемые параметры:** нет | | |
| **Возвращаемое значение:** | *TRUE* – функция успешно выполнена;<br>*FALSE* – функция выполнена с ошибкой. | |

### 3.6.3. Установка параметров работы частотомера

| **Формат:** | **BOOL** | ***SET_FM_PARS***(*FM_PARS_E310 * const FmPars*) |
|---|---|---|

**Назначение:**

Данная функция передает в *E-310* всю необходимую информацию, которая используется модулем для организации работы частотомера. Всю нужную информацию описываемая интерфейсная функция извлекает из полей передаваемой структуры типа *FM_PARS_E310*. Собственно, использование модулем именно этой переданной информации начинается только после выполнения интерфейсной функции *START_FM()*. Данную функцию не следует вызывать процессе работы частотомера, а только после выполнения функции *STOP_FM()*.

Формат структуры *FM_PARS_E310* приведен ранее в *§ 3.3.5. "Структура FM_PARS_E310"*, а смысл и назначение отдельных её полей достаточно подробно описано ниже.

- Поле FmPars->FmEna. Чтение. В данном поле возвращается текущее состояние работы частотомера.
- Поле FmPars->Mode. Запись–Чтение. Данное поле передаёт в функцию режим работы частотомера. На сегодняшний момент реализован только метод измерение периода им-

пульсов.

- Поле FmPars->InputDivider. Запись–Чтение. Данное поле позволяет управлть делителем частоты 1/8 для входного сигнала.
- Поле FmPars->BaseClockRateDivIndex. Запись–Чтение. Данное поле задаёт индекс делителя базовой тактовой частоты частотомера. Данное индекс может принимать значения в диапазоне от 1 до 6.
- Поле FmPars->ClockRate. Чтение. В данном поле возвращается значение рабочей тактовой частоты счётчика частотомера в *Гц*.
- Поле FmPars->BaseClockRate. Чтение. В данном поле возвращается значение базовой тактовой частоты счётчика частотомера, равной 25 000 000 *Гц*.
- Поле FmPars->Offset. Запись–Чтение. Данное поле задает величину смещения порога частотомера в *В*. Данное поле может принимать значения в диапазоне от -4 до 4. После выполнения функции в этом поле возвращается реально установленное значение смещения порога частотомера.

**Передаваемые параметры:**

- *FmPars* – адрес структуры типа *FM_PARS_E310* с требуемыми параметрами функционирования частотомера.

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

### 3.6.4. Получение текущих параметров работы частотомера

**Формат:** **BOOL** ***GET_FM_PARS***(*FM_PARS_E310 * const FmPars*)

**Назначение:**

Данная функция считывает из модуля *E-310* всю текущую информацию, которая используется при работе с частотомером.

**Передаваемые параметры:**

- *FmPars* – адрес структуры типа *FM_PARS_E310* с полученными из модуля текущими параметрами функционирования частотомера.

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

### 3.6.5. Получение отсчета измерения частоты

**Формат:** **BOOL** ***FM_SAMPLE***(*FM_SAMPLE_E310 * const FmSample*)

**Назначение:**

Данная функция пытается получить отсчет измерения частоты входного сигнала. Функцию ***FM_SAMPLE()*** следует вызывать только после выполнения интерфейсной функции *START_FM()*.

Формат структуры *FM_SAMPLE_E310* приведен ранее в *§ 3.3.6. "Структура FM_SAMPLE_E310"*, а смысл и назначение отдельных её полей достаточно подробно описано ниже.

- Поле FmSample->IsActual. Чтение. В данном поле возвращается статус отсчета измерения частотомера.
- Поле FmSample->Frequency. Чтение. Данное поле возвращает значение частоты измеряемого сигнала в *кГц*.
- Поле FmSample->Period. Чтение. Данное поле возвращает значение периода измеряемого сигнала в *мс*.
- Поле FmSample->DutyCycle. Чтение. Данное поле возвращает значение скважности измеряемого сигнала в *мс*.

**Передаваемые параметры:**

- *FmSample* – адрес структуры типа *FM_SAMPLE_E310* с полученным отсчетом измерения частоты входного сигнала.

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

## 3.7. Функции для работы с АЦП

Модуль *E-310* содержит 4х канальный 10ти битный АЦП. Данные с АЦП пользователь может получать только в однократном, и потому достаточно медленном, режиме.

### 3.7.1. Установка параметров работы АЦП

| **Формат:** | **BOOL** | ***SET_ADC_PARS***(*ADC_PARS_E310 * const AdcPars*) |
|---|---|---|

**Назначение:**

Данная функция передает в модуль *E-310* всю необходимую информацию, которая необходима для работы АЦП. Эта информация извлекается из полей передаваемой структуры типа *ADC_PARS_E310*.

Формат структуры *ADC_PARS_E310* приведен ранее в *§ 3.3.2. "Структура ADC_PARS_E310"*, а смысл и назначение отдельных её полей достаточно подробно описано ниже.

- Поле AdcPars->AdcStartSource. Запись–Чтение. Данное поле позволяет задавать источник сигнала, используемого для запуска АЦП. Этот сигнал может быть либо внутренним, либо внешним.
- Поле AdcPars->ChannelsMask. Запись–Чтение. Данное поле содержит битовую маску для задания активных каналов АЦП. В качестве битовой маски можно использовать младшие 4 бита.
- Поле AdcPars->InputRange. Чтение. Данное поле возвращает значение входного диапазона АЦП в *В*.

**Передаваемые параметры:**

- *AdcPars* – адрес структуры типа *ADC_PARS_E310* с требуемыми параметрами работы АЦП.

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

### 3.7.2. Получение текущих параметров работы АЦП

| **Формат:** | **BOOL** | ***GET_ADC_PARS***(*ADC_PARS_E310 * const AdcPars*) |
|---|---|---|

**Назначение:**

Данная функция считывает из модуля *E-310* всю текущую информацию, которая используется при работе АЦП.

**Передаваемые параметры:**

- *AdcPars* – адрес структуры типа *ADC_PARS_E310* с полученными из модуля текущими параметрами функционирования АЦП.

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

### 3.7.3. Получение данных АЦП

| **Формат:** | **BOOL** | ***GET_ADC_DATA***(*ADC_DATA_E310 * const AdcData*) |
|---|---|---|

**Назначение:**

Данная функция предназначена для получения очередных отсчётов с АЦП модуля. Эту функцию следует выполнять после вызова *SET_ADC_PARS()*.

Формат структуры *ADC_DATA_E310* приведен ранее в *§ 3.3.8. "Структура ADC_DATA_E310"*, а смысл и назначение отдельных её полей достаточно подробно описано ниже.

- Поле AdcData->DataInCode. Чтение. Данное поле является массивом из 4[x] элементов типа `SHORT`. В данном поле возвращаются значения отсчётов с активных каналов, выраженное в кодах АЦП. Если канал не активен, то соответствующий элемент массива будет равен -1.
- Поле AdcData ->DataInV. Чтение. Данное поле является массивом из 4[x] элементов типа `double`. В данном поле возвращаются значения отсчётов с активных каналов, выраженное в *В*. Если канал не активен, то соответствующий элемент массива будет равен -1.

**Передаваемые параметры:**

- *AdcData* – структура типа *ADC_DATA_E310* с отсчётами АЦП.

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

## 3.8. Функции для работы с цифровыми линиями

Модуль *E-310* оборудован внешними конфигурируемыми цифровыми линиями. Кол-во доступных цифровых линий можно программным образом менять: 8мь или 11ть. Каждую их цифровых линий можно индивидуальным образом настроить либо на вход, либо на выход.

Аппаратура модуля *E-310* 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 *потоковой* работы с ними.

### 3.8.1. Конфигурирование цифровых линий

| **Формат:** **BOOL** ***CONFIG_TTL_LINES***(*WORD Pattern, BOOL AddTtlLinesEna = FALSE*) |
|---|
| **Назначение:**<br>Данная интерфейсная функция позволяет осуществлять конфигурацию цифровых линий. При этом общее кол-во цифровых линий задается параметром *AddTtlLinesEna*:<br>• если *AddTtlLinesEna* = `FALSE`, то кол-во цифровых линий равно 8;<br>• если *AddTtlLinesEna* = `TRUE`, то кол-во цифровых линий равно 11 (!!!ВАЖНО!!! при этом исключается возможность полноценной работы с генератором и частотомером).<br>Кроме того, каждую из цифровых линий можно индивидуальным образом настроить на вход или выход, что определяется **`битовой`** маской параметра *Pattern*. |
| **Передаваемые параметры:**<br>• *Pattern* – битовая маска вход/выход.<br>• *AddTtlLinesEna* – параметр задающий кол-во цифровых линий. |
| **Возвращаемое значение:** *TRUE* – функция успешно выполнена;<br>*FALSE* – функция выполнена с ошибкой. |

### 3.8.2. Чтение внешних цифровых линий

| **Формат:** **BOOL** ***TTL_IN***(*WORD * const TtlIn*) |
|---|
| **Назначение:**<br>Данная интерфейсная функция осуществляет однократное чтение состояний *всех* активных цифровых входных линий.<br>Перед началом работы цифровые входные линии рекомендуется отконфигурить с помощью интерфейсной функции *CONFIG_TTL_LINES()*. |
| **Передаваемые параметры:**<br>• *TtlIn* – переменная, содержащая побитовое состояние активных входных цифровых линий. |
| **Возвращаемое значение:** *TRUE* – функция успешно выполнена;<br>*FALSE* – функция выполнена с ошибкой. |

### 3.8.3. Вывод на внешние цифровые линии

| **Формат:** | **BOOL** | ***TTL_OUT***(*WORD * const TtlOut*) |
|---|---|---|

**Назначение:**

Данная интерфейсная функция осуществляет однократную установку требуемых состояний сразу на *всех* активных цифровых выходных линиях. При этом в параметре *TtlOut* передается **побитовая** информация о требуемых состояниях для всех цифровых выходных линий. Так 1ый бит параметра *TtlOut* задаёт состояние выходной линии DO1, 2ой – DO2 и т.д.

Перед началом работы цифровые выходные линии рекомендуется отконфигурить с помощью интерфейсной функции *CONFIG_TTL_LINES()*.

**Передаваемые параметры:**

- *TtlOut* – переменная, содержащая побитовое состояние устанавливаемых выходных цифровых линий.

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

## 3.9. Функции для работы с пользовательским ППЗУ

На модуле *E-310* часть памяти микроконтроллера отведена под пользовательское ППЗУ. Размер этой области составляет *USER_FLASH_SIZE_E310* байт. Всю эту область пользователь может смело использовать в своих сугубо частнособственнических интересах.

### 3.9.1. Разрешение записи в ППЗУ

| **Формат:** | **BOOL** | ***ENABLE_FLASH_WRITE***(*BOOL IsUserFlashWriteEnabled*) |
|---|---|---|

**Назначение:**

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

**Передаваемые параметры:**

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

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

### 3.9.2. Запись данных в ППЗУ

| **Формат:** | **BOOL** | ***WRITE_FLASH_ARRAY***(*USER_FLASH_E310 * const UserFlash*) |
|---|---|---|

**Назначение:**

Данная интерфейсная функция выполняет запись массива байт размером *USER_FLASH_SIZE_E310* в ППЗУ. Т.о. перезаписывается сразу *вся* доступная область пользовательского ППЗУ. Перед началом процедуры записи в пользовательское ППЗУ необходимо разрешить эту операцию с помощью интерфейсной функции *ENABLE_FLASH_WRITE()*. После окончания процедуры записи всей требуемой информации необходимо запретить режим записи с помощью той же функции *ENABLE_FLASH_WRITE()*.

**Передаваемые параметры:**

- *UserFlash* – фактически это байтовый массив, который должен быть записан в пользовательское ППЗУ.

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

### 3.9.3. Чтение данных из ППЗУ

| **Формат:** **BOOL** ***READ_FLASH_ARRAY***(*USER_FLASH_E310 * const UserFlash*) |
|---|
| **Назначение:**<br>Данная интерфейсная функция вычитывает содержимое сразу всей области пользовательского ППЗУ. |
| **Передаваемые параметры:**<br>• *UserFlash* – в этом байтовом массиве возвращается образ всего пользовательского ППЗУ. |
| **Возвращаемое значение:** *TRUE* – функция успешно выполнена;<br>*FALSE* – функция выполнена с ошибкой. |

## *3.10. Функции для работы со служебной информацией*

Служебная информация содержит самые общие данные об используемом модуле *E-310*: название модуля, его серийный номер и ревизию, версии используемых прошивки микроконтроллера, тактовые частоты работы и т.д.

### 3.10.1. Чтение служебной информации

| | | |
|---|---|---|
| **Формат:** | **BOOL** | ***GET_MODULE_DESCRIPTION***(<br>*MODULE_DESCRIPTION_E310 * const ModuleDescription*) |
| **Назначение:**<br>Данная интерфейсная функция осуществляет чтение всей служебной информации в структуру типа *MODULE_DESCRIPTION_E310*. | | |
| **Передаваемые параметры:**<br>• *ModuleDescription* – указатель на структуру типа *MODULE_DESCRIPTION_E310*. | | |
| **Возвращаемое значение:** | *TRUE* – функция успешно выполнена;<br>*FALSE* – функция выполнена с ошибкой. | |

### 3.10.2. Запись служебной информации

| | | |
|---|---|---|
| **Формат:** | **BOOL** | ***SAVE_MODULE_DESCRIPTION***(<br>*MODULE_DESCRIPTION_E310 * const ModuleDescription*) |
| **Назначение:**<br>Данная интерфейсная функция позволяет сохранять в модуле всю служебную информацию из структуры типа *MODULE_DESCRIPTION_E310*. **!!!Внимание!!!** Применять данную функцию нужно только в случае крайней необходимости. Например, когда по тем или иным обстоятельствам испортилось содержимое служебной информации. | | |
| **Передаваемые параметры:**<br>• *ModuleDescription* – указатель на структуру типа *MODULE_DESCRIPTION_E310*, из которой служебная информация переносится в модуль. | | |
| **Возвращаемое значение:** | *TRUE* – функция успешно выполнена;<br>*FALSE* – функция выполнена с ошибкой. | |

# Приложение А. Вспомогательные константы и типы

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

## А.1. Константы

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

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

## А.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
};
```

## А.3. Структура MCU_VERSION_INFO_LUSBAPI

Вспомогательная структура *MCU_VERSION_INFO_LUSBAPI* состоит из двух структур *VERSION_INFO_LUSBAPI* и содержит информацию о программном обеспечении исполнительного устройства, которая включает в себя информацию о прошивках как основной программы (*Firmware*), так и загрузчика (*Bootloader*). Данная структура описывается следующим образом:

```
struct MCU_VERSION_INFO_LUSBAPI
{
    VERSION_INFO_LUSBAPI FwVersion;     // версия основной программы (Firmware)
    VERSION_INFO_LUSBAPI BlVersion;     // версия загрузчика (BootLoader)
};
```

## А.4. 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-изготовите-
                                                  // ля изделия
    BYTE DeviceName[NAME_LINE_LENGTH_LUSBAPI];    // название изделия
    BYTE SerialNumber[16];                        // серийный номер изделия
    BYTE Revision;                                // ревизия изделия
    BYTE Modification;                           // исполнение (вариант) модуля;
    BYTE Comment[COMMENT_LINE_LENGTH_LUSBAPI];    // строка комментария
};
```

## A.5. 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
};
```

## A.6. 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 Firmware и BootLoader
    BYTE   Comment[COMMENT_LINE_LENGTH_LUSBAPI];    // строка комментария
};
```

## A.7. Структура PLD_INFO_LUSBAPI

Вспомогательная структура *PLD_INFO_LUSBAPI* содержит самую общую информацию об используемом исполнительном устройстве типа программируемая логическая интегральная схема (ПЛИС). Данная структура описывается следующим образом:

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

## A.8. 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
};
```

## A.9. 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
};
```

## A.10. 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
};
```