Библиотека пользовательского интерфейса модуля LTR22

Крейтовая система LTR

Руководство программиста

Ревизия 1.0.4

Апрель 2007г.

Автор руководства:
Акристиний М.В.
m_akristinii@lcard.ru

ЗАО "Л-КАРД"
117105, г. Москва, Варшавское ш., д. 5, корп. 4, стр. 2

тел.: (095) 785-95-25
факс: (095) 785-95-14

Адреса в Интернет:
http://www.lcard.ru/
ftp://ftp.lcard.ru/pub

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

Представители в регионах:
Украина: HOLIT Data Sistems, *http://www.holit.com.ua*, (044) 241-6754
Санкт-Петербург: Autex Spb Ltd., *http://www.autex.spb.ru*, (812) 567-7202
Новосибирск: Сектор-Т, *http://www.sector-t.ru*, (383-2) 396-592
Екатеринбург: Аск, *http://www.ask.ru*, 71-4444
Казань: ООО 'Шатл', *shuttle@kai.ru*, (8432) 38-1600

Крейтовая система LTR

История ревизий настоящего документа.

Ревизия	Дата	Примечания по внесенным изменениям
1.0.0	06.04.2006	Первая доступная для пользователя ревизия
1.0.1	16.10.2006	Добавилось описание работы с синхронизацией, поменялись функции старта сбора данных и останова.
1.0.2	29.11.2006	Изменены калибровки модуля (структура калибровок увеличена)
1.0.3	28.02.2007	Добавлена функция LTR22_ProcessDataTest
1.0.4	23.04.2007	Добавлено описание LTR22_GetErrorString, изменены примеры для подержки ошибки LTR_WARNING_MODULE_IN_USE

На CD-ROM, входящий в комплект поставки, всегда записывается последняя ревизия данного документа. Кроме того, последнюю ревизию Вы сможете найти в разделе *библиотека файлов* на нашем сайте.

L-Card оставляет за собой право обновлять документацию без уведомления пользователей об изменениях.

Содержание :

1. Основные принципы работы с библиотекой пользовательского интерфейса модуля LTR22

Модуль АЦП LTR22 предназначен для создания многоканальных систем сбора данных на производстве и в лаборатории. Основные области применения LTR22: аудио обработка, виброметрия и другие задачи оцифровки сигналов звукового диапазона частот, где принципиальное значение имеет высокое качество и высокая спектральная верность оцифровки переменной составляющей сигнала при среднем качестве оцифровки постоянной составляющей сигнала. Подробно о структуре модуля и всех его возможностях написано в документе Крейтовая система LTR. Руководство пользователя. В данном Руководстве речь пойдет о программировании модуля посредством вызова функций, содержащихся в библиотеке пользовательского интерфейса, а также описание функций AVR.

На плате модуля установлен микроконтроллер **AVR Atmega 8515**, осуществляющий общее управление модулем, контролирующий обмен информацией с крейт-контроллером. Управляющая программа микроконтроллера записывается в его флэш-память при изготовлении модуля. В этой памяти также содержится идентификационная информация (серийный номер, версия управляющей программы микроконтроллера, дата ее создания, имя модуля), а также калибровочные коэффициенты. С точки зрения программного обеспечения связь с микроконтроллером модуля и обмен информацией с ним осуществляются при помощи библиотеки пользовательских функций, для более полного управления модулем конечно же можно применять и функции по работе с AVR.

Запуск считываний данных возможен как от внутренней команды, так и от внешнего синхроимпульса.

Последовательность применения пользовательских функций сходна с использованием аналогичных функций в программном обеспечении других модулей системы LTR. В общих чертах эта последовательность выглядит следующим образом:

- Инициализация интерфейсного канала связи с модулем, установка настроек по умолчанию
- Установка параметров работы (параметры диапазонов, частоты АЦП, измерение нуля, каналов сбора данных, типа синхронизации, фазировки) и загрузка их в память микроконтроллера AVR Atmega 8515
- Получение данных с АЦП
- Закрытие интерфейсного канала связи с модулем.

Функции библиотеки модуля LTR22 разрабатывались, как дополнение к основным функциям крейта LTR, и на практике используются совместно с ними.

2. Описание работы с пользовательской библиотекой API функций

Функции, входящие в состав пользовательской библиотеки модуля LTR22, можно разделить на следующие группы:

- ✓ *Функции инициализации и открытия* – это функции, предназначенные для открытия интерфейса с модулем:
  LTR22_Init()
  LTR22_Open()
- ✓ *Функции конфигурирования* – предназначены для настройки модуля.
  Могут вызываться также и во время сбора данных (кроме функций *LTR22_SwitchMeasureADCZero(), LTR22_SetConfig(), LTR22_GetConfig(), LTR22_GetCalibrovka()*).
  Следует учесть, что при вызове функций LTR22_SetADCRange() и LTR22_SetADCChannel() модуль перестроится сразу, но в буфере сервера также находятся старые данные, и функция LTR22_ProcessData() будет возвращать ошибки.

  LTR22_StartADC() – старт сбора данных
  LTR22_StopADC() – стоп сбора данных

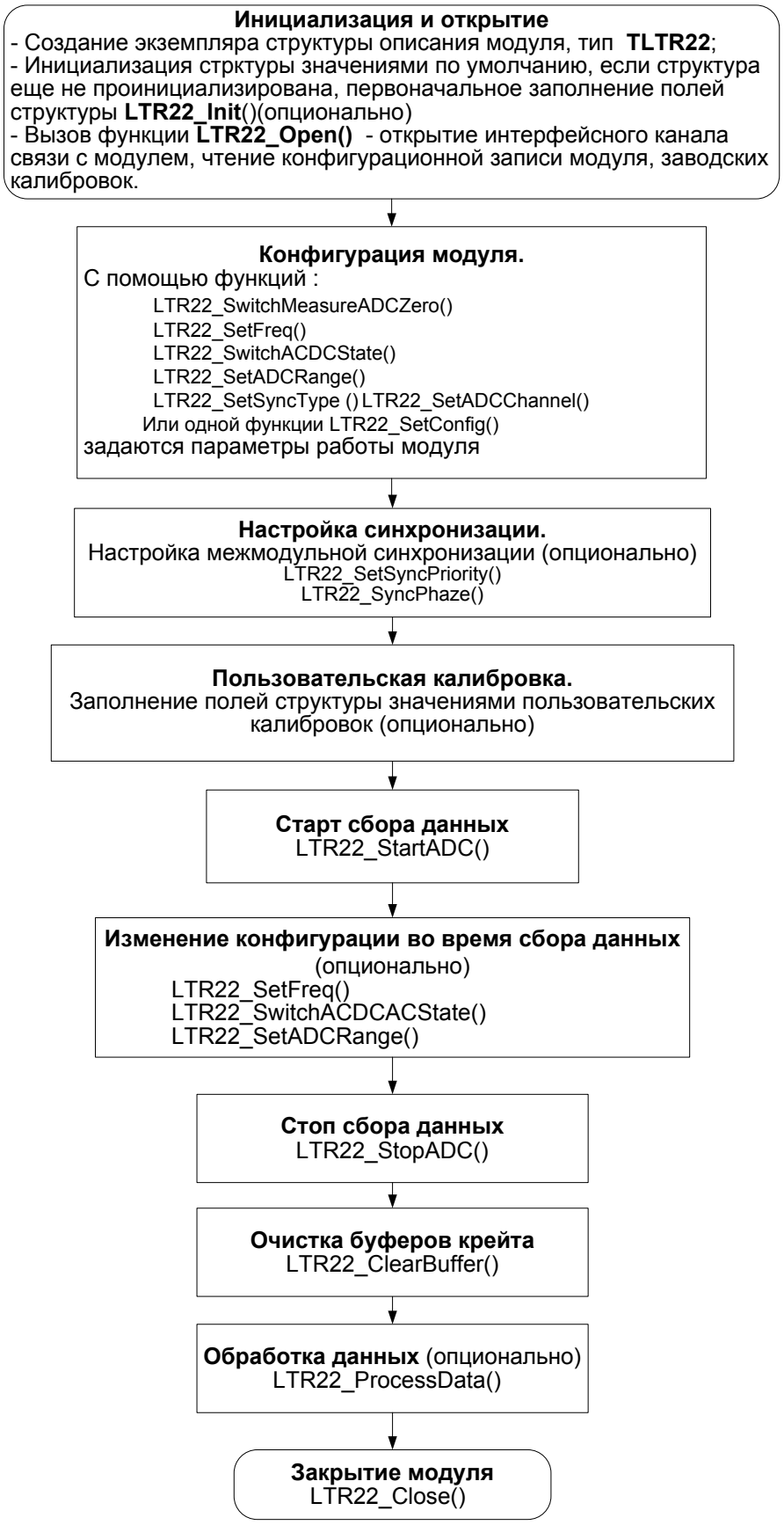
  LTR22_SwitchMeasureADCZero()
  LTR22_SetFreq()
  LTR22_SwitchACDCState()
  LTR22_SetADCRange()
  LTR22_SetADCChannel()

  LTR22_SetConfig() запись всех настроек (замена предыдущих фунций)
  Для считывания текущей конфигурации модуля, также присутствуют функции
  LTR22_GetConfig() – получение состояния модуля
  LTR22_GetCalibrovka() – получение калибровочных фабричных значений
- ✓ *Функции настройки синхронизации*
  LTR22_SetSyncPriority() – режим работы модуля Slave или Master
  LTR22_SyncPhaze() – фазировка модуля
- ✓ *Функции сбора данных* предназначены для сбора и обработки полученных данных :
  LTR22_Recv()
  LTR22_ProcessData()
  LTR22_ProcessDataTest()
- ✓ *Дополнительные функции* :
  EEPROM AVR при использовании модуля не используется, поэтому пользователь может его использовать по собственному назначению.
  LTR22_ReadAVREEPROM() – считывание пользовательских значений EEPROM
  LTR22_WriteAVREEPROM() – запись пользовательских значений EEPROM
  LTR22_IsOpened() – проверка – открыт ли модуль

LTR22_ClearBuffer() – очистка буфферов модуля, обычно используется после приема данных, если данные остались в буффере сервера или крейта.
LTR22_GetModileDescription() – получение описания модуля.
LTR22_GetErrorString() – получение описания кода ошибки.

- ✓ *Функция закрытия*
  LTR22_Close() – закрывает текущий интерфейс с модулем и обнуляет handler модуля, собирая ресурсы, используемые сервером для модуля. Для корректной работы сервера следует обязательно вызывать эту функцию после завершения работы.

Рис. 2.1. Типичная последовательность использования модуля LTR22



3. Межмодульная синхронизация

У модулей LTR22 имеется возможность межмодульной синхронизации, но поскольку в модулях установлены сигма – дельта АЦП, со встроенным конвейером и фильтром, то синхронизация проводится сложнее, чем обычно, нельзя просто выключить АЦП и включить его в заданное время. Кроме того АЦП имеет длительное время включения и автокалибровки, последующей за ним. Опрос АЦП в модуле также идет пачками по 2 канала (2 и 4 затем 1 и 3). Чтобы засинхронизировать фазы сбора данных а также фазы пачек АЦП в нескольких модулях применяется процедура фазировки :

1. *Синхроимпульс фазировки генерирует один из модулей LTR22*

   В таком случае у нас есть один ведущий модуль (Master) и несколько ведомых модулей (Slave).

   Slave модули выставляют режим Slave (*LTR22_SetSyncPriority() – с параметром False*), входят в режим ожидания синхроимпульса фазировки на заданное время (*LTR22_SyncPhaze()*),
   Master выставляет режим Master (*LTR22_SetSyncPriority() – с параметром true*), входит в режим ожидания с генерацией синхроимпульса для себя и остальных модулей (*LTR22_SyncPhaze()*).
2. *Синхроимпульс фазировки генерирует внешний источник*

   Все модули работают в режиме Slave.

   Модули выставляют режим Slave (*LTR22_SetSyncPriority() – с параметром False*), входят в режим ожидания синхроимпульса фазировки на заданное время (*LTR22_SyncPhaze()*),

Внимание ! После выполнения процедуры фазировки АЦП обресечиваются, после чего они входят в режим калибровки нуля, время этого режима зависит от выставленной частоты сбора данных (первые 8192 сэмплов с каждого канала – время автокалибровки нуля АЦП, - модуль пропускает, в результате режим ожидания на низкой частоте сбора данных может составить до 3 с).

Внимание ! После фазировки, нельзя менять частоту сбора данных, так как это может привести к нарушению фазировки и потребовать повторения этой процедуры.

Внимание ! Если сигнал синхронизации фазировки не приходит, то модули Slave переходят в бесконечный режим ожидания сигнала синхронизации фазировки, для выхода из него необходимо их обресетить.

Для того, чтобы засинхронизировать старт сбора данных между модулями, применяется межмодульная синхронизация сбора данных, ее процедура похожа на процедуру фазировки :

1. *Синхроимпульс генерирует один из модулей LTR22*

   В таком случае у нас есть один ведущий модуль (Master) и несколько ведомых модулей (Slave).

   Slave модули выставляют режим Slave (*LTR22_SetSyncPriority() – с параметром False*). Модули входят в ожидание внешнего импульса синхронизации сбора данных (*LTR22_StartADC () с параметром WaitSync*), и ожидают прихода данных (что свидетельствует о приходе синхроимпульса) с заданным таймаутом (*LTR22_Recv()*).

   Master выставляет режим Master (*LTR22_SetSyncPriority() – с параметром true*). Модуль начинает сбор данных с параметром ожидания синхронизации (*LTR22_StartADC () - с параметром WaitSync*), во время выполнения этой функции он также генерирует сигнал синхроимпульса для всех Slave модулей и себя, и начинает считывать данные (*LTR22_Recv()*).

2. *Синхроимпульс генерирует внешний источник*

   Все модули работают в режиме Slave.

   Slave модули выставляют режим Slave (*LTR22_SetSyncPriority() – с параметром False*). Модули входят в ожидание внешнего импульса синхронизации сбора данных (*LTR22_StartADC () с параметром WaitSync*), и ожидают прихода данных (что свидетельствует о приходе синхроимпульса) с заданным таймаутом (*LTR22_Recv()*).

Внимание ! После синхронного старта модулей, в АЦП первые 37 сэмплов необходимо откидывать, поскольку в модуле есть цифровой фильтр глубиной 36 сэмплов, и один сэмпл добавляется из – за того, что данные передаются пачками по 2 канала на одну пачку, и может попасть пачка также относящаяся к периоду до синхросигнала.

Функции синхронизации фаз, и синхронизации сбора данных разработаны для наиболее удобного их применения в нескольких потоках (т.е. под каждый модуль – свой поток). Но их можно использовать и в одном потоке.

```
LTR22_SetConfig(module);    // устанавливается конфигурация первого модуля, по заполненной заранее структуре module
LTR22_SetConfig(module2); // устанавливается конфигурация воторго модуля, по заполненной заранее структуре module


LTR22_SetSyncPriority(module,false); // модуль 1 – в режиме Slave
LTR22_SetSyncPriority(module2,true); // модуль 2 – в режиме Master

LTR22_SyncPhaze(module,100); // модуль 1 – переходит в режим бесконечного ожидания
LTR22_SyncPhaze(module2,2000); // модуль 2 – переходит в режим бесконечного ожидания после чего генерирует импульс синхронизации фаз, и выводит первый модуль из состояния ожидания

DWORD SyncResponse=0;
LTR22_Recv(module, & SyncResponse, NULL, 1, 2000); // поскольку не дождались, пока первый модуль завершит синхронизацию фаз, то нужно принять его комманду отклика,

LTR22_StartADC(module,true); // модуль 1 – переходит в режим ожидания синхронизации сбора данных
LTR22_StartADC(module2,true); // модуль 2 – переходит в режим ожидания синхронизации, но полскольку он Master, то после этого он генерирует синхроимпульс


LTR22_Recv(module,(DWORD *)ReadCode_ptr, NULL, NumReads+NumEmptyReads, 3000));
LTR22_Recv(module2,(DWORD *)ReadCode2_ptr,NULL,NumReads+NumEmptyReads,3000));
```

Рекомендуется использовать несколько потоков, так как это гораздо удобнее и правильнее при работе с крейтом. Один поток можно использовать на небольших скоростях сбора данных, или при 1 -2 модулях в крейте.

Подробнее о процедурах синхронизации фаз и синхронизации сбора данных можно посмотреть в документе Крейтовая система LTR. Руководство пользователя.

4. Калибровки модуля

Калибровка модуля записана во внутреннюю память микроконтроллера AVR. Считывается по функции LTR22_GetCalibrovka() и заносится в основную структуру – описатель модуля. У модуля присутствуют 2 типа калибровок – фабричные – для калибровки отсчетов АЦП и пользовательские, например для преобразования отсчетов АЦП в физические величины, например в вольты. Фабричные калибровки прошиваются при изготовлении модуля и хранятся во внутренней памяти AVR модуля. Пользовательские калибровки пользователь задает сам в основной структуре модуля. Калибруется каждый поддиапазон модуля.

Вычисление калибровок :

$$Calib\_Value\_Pset = Value\_Pset * Calib\_Scale + Calib\_Offset$$

где

Calub_Value_Pset – калиброванное значение отсчетов АЦП
Value_Pset – Некалиброванное значение отсчетов АЦП
Calib_Scale – масштабный коэффициент калибровки
Calib_Offset – сдвиговый коэффициент калибровки

Функция LTR22_ProcessData() – автоматически преобразует, некалиброванные значения отсчетов АЦП в калиброванные и еще делает дополнительную калибровку – в физические величины, при указании соответствующих параметров – при вызове функции.

Калибровочные коэффициенты едины для режимов AC и DC.

При смене прошивки модуля через UTS – фабричные калибровочные коэффициенты перепрашиваются теми же значениями что и раньше.

5. Подробное описание библиотеки ltr22api

Для получения возможности вызова интерфейсных функций библиотеки ***ltr22api.dll*** из вашего приложения необходимо следующее:

- создать проект в какой либо из сред разработки;
- поместить в папку проекта или в папку, описанную в переменной окружения **PATH,** файл **ltr22api.dll**.
- добавить в проект информацию о способе вызова интерфейсных функций dll-библиотеки и используемых типах данных. В различных средах разработки последовательность действии и приложенные усилия могут несколько отличаться:

  Borland C++/Borland C++ Builder :
  - подключить к проекту файлы **LTR\LIB\BORLAND\ltr22api.lib** и **LTR\INCLUDE\ltr22api.h**;

  Microsoft Visual C++ :
  - подключить к проекту файлы **LTR\LIB\MICROSOFT\ltr22api.lib** и **LTR\INCLUDE\ltr22api.h**;

  Другие среды разработки :
  - следует обратиться к соответствующей документации на средство разработки.

- создать и добавить в проект файл с исходным текстом будущей программы;
- после этого можно писать свою программу, вызывая соответствующие интерфейсные функции dll-библиотеки.

5.2. Управляющая структура модуля

Поля данной структуры содержат информацию о конфигурации АЦП модуля, режимах работы модуля, диапазонах каналов, описание модуля, и его калибровки, как фабричные, так и пользовательские. Первоначальная конфигурация структуры заполняется с помощью функции **LTR22_Init().** Определение структуры приводится ниже:

```
//**** конфигурация модуля
typedef struct
{
  //**** служебная информация       //
  INT size;                                                    // размер структуры TLTR22 1036
байт
  TLTR Channel;                 // служебная структура для работы с крейтом через сервер

  // настройки модуля
  byte Fdiv_rg;                                          // дивайзер частоты клоков 1..15
  bool Adc384;                                           // дополнительный дивайзер частоты
сэмплов true =3 false =2
  bool AC_DC_State;                                      // состояние true =AC+DC false=AC
  bool MeasureADCZero;                                   // измерение Zero true - включено false -
выключено
```

```
  bool DataReadingProcessed;                // состояние считывания АЦП true-АЦП считывается false - нет
  byte   ADCChannelRange[LTR22_ADC_NUMBERS];// предел имзерений АЦП по каналам 0 - 1В 1 - 0.3В 2 - 0.1В 3 - 0.03В 4 - 10В 5 - 3В

  bool ChannelEnabled[LTR22_ADC_NUMBERS];          // Состояние каналов, включен - true выключен - false

  int FreqDiscretizationIndex;              // частота дискретизации, выставленная сейчас 0..24 - в зависимости от частоты
                                                                  // из массива LTR22_DISK_FREQ_ARRAY

  byte SyncType;              // Тип синхронизации 0 - внутренний старт по сигналу Go
                                          //1 - фазировка модуля
                                          //2 - внешний старт
                                          //3 - резервировано
  bool SyncMaster;            // true - модуль генерит сигнал, false - модуль принимает синхросигнал

  TINFO_LTR22 ModuleInfo;
  ADC_CHANNEL_CALIBRATION ADCCalibration[LTR22_ADC_NUMBERS][LTR22_MAX_DISC_FREQ_NUMBER];

} TLTR22,*PTLTR22;
```

Состав вложенных структур описан в приложениях 2 и 3.

Калибровки модуля – определяются в зависимости от частоты дискретизации, выбранного диапазона и канала.

Допустимые частоты дискретизации :

```
const int LTR22_DISK_FREQ_ARRAY[LTR22_MAX_DISC_FREQ_NUMBER]=
{
      3472,   3720,    4006,   4340,   4734,   5208,   5580,   5787,
      6009,   6510,    7102,   7440,   7812,   8680,   9765,   10416,
      11160, 13020, 15625,   17361, 19531, 26041, 39062, 52083,
      78125
};
```

Согласно этому массиву выбирается и FreqDiscretizationIndex.

5.2. Описание функций библиотеки ltr22api.dll

Функции библиотеки ltr22api.dll используются совместно со штатными функциями ltrapi.dll, дополняя их и облегчая использование модуля LTR22.

Для удобства использования библиотеки в managed языках введены локальные типы данных :

```
typedef BYTE byte;

#ifndef __cplusplus
#define true 1
#define false 0
typedef byte bool;
#endif
```

Внимание, после подачи ресета в модуль и получения ответа модуля на ресет необходимо ждать не менее 110 мс перед записью команд, в это время происходит автокалибровка нуля АЦП модуля, и все отосланные команду будут игнорироваться (это учитывается в LTR22_Open()).

5.2.1. LTR22_Init

int LTR22_Init(TLTR22 *module);

Параметры :

- ***module*** - указатель на структуру типа TLTR22, которая должна существовать в памяти перед вызовом функции.

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция инициализирует поля структуры модуля (module) по умолчанию. Необходима после первоначального создании структуры TLTR22.

Удобна – при создании нескольких структур типа TLTR22 и заполнении их значениями по умолчанию. В работе может не использоваться, поскольку все равно включена в функцию LTR22_Open().

Также инициализируются пользовательские калибровки (module.ADCCalibration[i].UserCalibScale) для идеального случая преобразования получаемых отсчетов АЦП в Вольты.

Обновляются все поля в структуре module.

Состояние структуры :

```
//**** конфигурация модуля
typedef struct
{
//**** служебная информация       //

  INT size = 1036
  TLTR Channel=<Информация крейта о модуле начальные значения>;

  // настройки модуля
  byte Fdiv_rg=4;                          // дивайзер частоты клоков 1..15
  bool Adc384 = false;              // дополнительный дивайзер частоты сэмплов true =3 false =4
  bool AC_DC_State = true;        // состояние true =AC+DC false=AC
  bool MeasureADCZero = true;  // измерение Zero true - включено false - выключено
  bool DataReadingProcessed=false;      // состояние сбора данных, true – сбор идет
  byte ADCChannelRange[LTR22_ADC_NUMBERS]={4,4,4,4};// предел имзерений АЦП по каналам 0 - 1В
1 - 0.3В 2 - 0.1В 3 - 0.03В 4 - 10В 5 - 3В

  bool ChannelEnabled[LTR22_ADC_NUMBERS]={true,true,true,true};// Состояние каналов, включен - true
выключен - false

  byte SyncType=0;// Тип синхронизации 0 - внутренний старт по сигналу Go
```

```
  bool SyncMaster=false; // Режим Slave


  TINFO_LTR22 ModuleInfo=<Пусто>;
  ADC_CHANNEL_CALIBRATION ADCCalibration=<Пользовательские калибровки по умолчанию>;
} TLTR22;
```

5.2.2. LTR22_Open

int LTR22_Open(TLTR22 *module, DWORD saddr, WORD sport, CHAR *csn, WORD cc);

Параметры :

- ***module*** - указатель на структуру типа TLTR22, которая должна существовать в памяти перед вызовом функции.
- ***saddr*** - сетевой адрес сервера A.B.C.D в формате HEX: 0xABCD. Например, **net_addr** для адреса 127.0.0.1 будет выглядеть следующим образом: 0x7F000001. Необходимо помнить, что все компоненты адреса должны иметь значение, не превосходящее 255.
- ***sport*** - сетевой порт сервера.
- ***csn*** – серийный номер крейта (массив char [16])
- ***cc*** – номер слота модуля (нумерация от 1 до 16)

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

если код ошибки LTR_WARNING_MODULE_IN_USE, то с модулем можно работать, и обязательно завершить сессию с сервером LTR22_Close()

Описание :

Функция открывает интерфейсный канал связи с модулем, выполняет необходимые проверки, а также считывает из ППЗУ модуля его фабричные калибровки. После работы функции в соответствующих полях структуры описания модуля будут находиться: версия БИОСа, дата создания БИОСа, имя модуля и серийный номер модуля, описание контроллера модуля, его тактовая частота, фабричные калибровки модуля, его текущее состояние настройки. Выполняется также автоматическая инициализация структуры TLTR22 *module.

Обновляются все поля в структуре module

Состояние структуры :

```
//**** конфигурация модуля
typedef struct
{
//**** служебная информация       //
  INT size = 1036
  TLTR Channel=<Информация крейта о модуле начальные значения>;

  // настройки модуля
```

```
  byte Fdiv_rg=4;                          // дивайзер частоты клоков 1..15
  bool Adc384 = false;                // дополнительный дивайзер частоты сэмплов true =3 false =4
  bool AC_DC_State = true;         // состояние true =AC+DC false=AC
  bool MeasureADCZero = true;  // измерение Zero true - включено false - выключено
  bool DataReadingProcessed=false;      // состояние сбора данных, true – сбор идет
  byte ADCChannelRange[LTR22_ADC_NUMBERS]={4,4,4,4};// предел имзерений АЦП по каналам 0 - 1В
1 - 0.3В 2 - 0.1В 3 - 0.03В 4 - 10В 5 - 3В

  bool ChannelEnabled[LTR22_ADC_NUMBERS]={true,true,true,true};// Состояние каналов, включен - true
выключен - false

  byte SyncType=0;// Тип синхронизации 0 - внутренний старт по сигналу Go

  bool SyncMaster=false; //Режим Slave

  TINFO_LTR22 ModuleInfo=<Фабричное описание>;
  ADC_CHANNEL_CALIBRATION ADCCalibration=<Фабричные калибровки и Пользовательские
калибровки по умолчанию>;

} TLTR22;
```

5.2.3. LTR22_Close

int LTR22_Close(TLTR22 *module);

Параметры :

- ***module*** - указатель на структуру типа TLTR22.

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается «0» (LTR_OK).

Описание :

Выполняет закрытие интерфейсного канала связи с модулем. Эту функцию следует вызывать всегда перед окончанием работы с модулем.

Обновляются поля в структуре module, относящиеся к модулю.

5.2.4. LTR22_IsOpened

int LTR22_IsOpened(TLTR22 *module);

Параметры :

- ***module*** - указатель на структуру типа TLTR22

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция позволяет отслеживать состояние соединения с модулем, если возвращаемый результат отличен от 0, то соединения нет.

Обновляются поля в структуре module, относящиеся к модулю.

5.2.5. LTR22_SwitchMeasureADCZero

int LTR22_SwitchMeasureADCZero(TLTR22 *module, bool SetMeasure);

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***SetMeasure*** - параметр, true – измерение нуля включить, false – измерение нуля выключить.

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет включение – выключение измерения нуля АЦП, при включении этой функции входы АЦП переключаются на землю, что позволяет проводить динамическое измерение нуля, поскольку у сигма – дельта АЦП присутствует большой температурный дрейф во время работы.

Обновляется поле в структуре TLTR22.MeasureADCZero

5.2.6. LTR22_SwitchACDCState

int LTR22_SwitchACDCState(TLTR22 *module, bool ACDCState);

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***ACDCState*** - параметр, true – режим AC+DC (переменная с постоянной составляющей), false – режим AC (только переменная составляющая, постоянная составляющая, за счет интегратора убирается).

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет переключение между режимами AC+DC и AC.

Обновляется поле в структуре TLTR22.AC_DC_State

5.2.7. LTR22_SetADCRange

int LTR22_SetADCRange(TLTR22 *module, byte ADCChannel, byte ADCChannelRange);

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***ADCChannel*** - номер канала АЦП (0..3).
- ***ADCChannelRange*** – значение диапазона АЦП (0..5)

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет задачу диапазона для заданного канала АЦП. Возможные значения диапазонов :

- 0 – ±1 В
- 1 – ±0.3 В
- 2 – ±0.1 В
- 3 – ±0.03 В
- 4 – ±10 В
- 5 – ±3 В

Обновляются поля в структуре TLTR22. ADCChannelRange[ADCChannel]

5.2.8. LTR22_SetADCChannel

int LTR22_SetADCChannel(TLTR22 *module, byte ADCChannel, bool EnableADC);

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***ADCChannel*** - номер канала АЦП (0..3).
- ***EnableADC*** – параметр, true – канал выключен, false – канал включен.

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет включение – выключение каналов (всего 4 канала) во время сбора данных, вызываемого функцией LTR22_StartADC (). В результате получаем поток из значений включенных каналов.

Обновляются поля в структуре TLTR22. ChannelEnabled [ADCChannel]

5.2.9. LTR22_SetFreq

int LTR22_SetFreq(TLTR22 *module, bool adc384, byte Freq_dv);

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- *adc384* - дополнительный дивайзер, true – fdiv_extra=3, false – fdiv_extra=2.
- Freq_dv – основной дивайзер (fdiv_main), принимает значения от 1 до 15 (4 бит)

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция выставления частоты клоков АЦП (Fclkin) и частоту сэмплов АЦП(Flrck).

$$F_{CLKIN} = \frac{20 * 10^6}{fdiv\_main} \quad \text{Гц}$$

Отсюда мы можем получить частоту сэмплов АЦП (на канал)

$$F_{LRCK} = \frac{F_{CLKIN}}{128 \cdot fdiv\_extra} \quad Samples, Гц$$

Следует учитывать, что частота клоков АЦП (Fclk) меняется только в диапазоне от 1.28 – 20.84 МГц. Функция LTR22_SetFreq() также отслеживает, чтобы частота была в допустимых пределах. Если частота превышает допустимые пределы, функция не выставляет новое значение и возвращает код ошибки.

Обновляются поля в структуре TLTR22. Fdiv_rg и TLTR22. Adc384 и TLTR22.FreqDiscretizationIndex (TLTR22.FreqDiscretizationIndex – индекс текущей частоты в массиве LTR22_DISK_FREQ_ARRAY[])

5.2.10. LTR22_SetSyncPriority

int LTR22_SetSyncPriority(TLTR22 *module, bool SyncMaster)

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***SyncMaster*** - определяет, будет ли модуль задавать сигнал – при синхронизации или нет

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Если модуль задан как мастер, то он будет выдавать синхроимпульс при синхронизации фаз, а также при синхронном старте АЦП, между модулями, иначе он будет только принимать синхроимпульс.

Эта функция должна предшествовать функциям фазировки, а также функциям синхронного старта АЦП.

По умолчанию модуль работает в режиме Slave.

Обновляются поля в структуре TLTR22. SyncMaster

5.2.11. LTR22_SyncPhaze

int LTR22_SyncPhaze(TLTR22 *module, DWORD timeout)

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***timeout*** - время ожидания ответа от модуля, о том, что пришел сигнал синхронизации

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Происходит фазировка модуля с другими модулями, если был задан выход внешнего сигнала синхронизации (LTR22_SetSyncPriority), то во время фазировки модуль является мастером для других модулей, и фактически генерирует сигнал синхронизации для себя и остальных.

Функция зависает на заданное время в ожидании ответа от модуля, а модуль ждет синхроимпульс бесконечно, для того, чтобы вывести его из режима ожидания – необходимо его обресетить.

Процедура фазировки заключается в синхронизации фаз сбора данных между модулями, или между модулем и внешним источником синхронизации.

После процедуры фазировки AVR ждет 8192 сэмплов АЦП – поскольку АЦП входит в режим автокалибровки, поэтому ***timeout*** на минимальной частоте сбора данных (3472 сэмпла/с) может составить 2,35 с, максимальное значение определяется пользователем.

В процессе процедуры фазировки меняется конфигурация модуля – поддиапазоны выставляются на максимальные значения, включается режим AC+DC, а также режим измерения нуля, после фазировки они возвращаются в прежнее положение. После фазировки нельзя менять частоту сбора данных, иначе фазировка собъется, и процесс фазировки необходимо будет повторить заново.

Обновляются поля в структуре TLTR22. SyncType

5.2.12. LTR22_SetConfig

int LTR22_SetConfig(TLTR22 *module);

Параметры :

- ***module*** - указатель на структуру типа TLTR22

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет запись настроек модуля из полей модуля в AVR :

Fdiv_rg,
Adc384,
AC_DC_State,
MeasureADCZero,
DataReadingProcessed,
ADCChannelRange[LTR22_ADC_NUMBERS].
ChannelEnabled[LTR22_ADC_NUMBERS];
SyncType;
SyncMaster;

Подразумевается, что поля заполняются пользователем, до вызова функции в ручную.

DataReadingProcessed –всегда переводится в false.

Удобна при считывании настроек модуля из сохраненной конфигурации и записи их в модуль. Главное отличие от других функций, поскольку каждая функция изменения настройки если не включен сбор данных ждет ответа от модуля, следовательно при вызове всех этих функций нужно будет ждать ответа. В отличие от набора функций, функция LTR22_SetConfig перезаписывает регистры напрямую, и отсылает всего четыре комманды в AVR, при этом ожидает ответа не от каждой комманды в отдельности, а сразу от четырех, в результате существенно сокращается время. Не обновляется только тип синхронизации (вообще SyncType – служебное поле, используемое больше в качестве проверки выставленного типа синхронизации в модуле, и используется в основном только на чтение)

Недостаток - пользователю необходимо будет записать поля в структуру модуля в ручную, кроме того не может использоватся во время сбора данных.

Также обновляются поля в структуре module, относящиеся к модулю и информации о модуле из крейта.

5.2.13. LTR22_GetConfig

int LTR22_GetConfig(TLTR22 *module);

Параметры :

- ***module*** - указатель на структуру типа TLTR22

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет подачу запроса модулю, чтобы он выслал статусные регистры, при этом заполняется структура модуля : состояние делителей, состояние измерения нуля, режим постоянно – переменного напряжения.

Обновляются поля в структуре TLTR22.
Fdiv_rg,
Adc384,
AC_DC_State,
MeasureADCZero,
DataReadingProcessed,
ADCChannelRange[LTR22_ADC_NUMBERS].
ChannelEnabled[LTR22_ADC_NUMBERS];
SyncType;
SyncMaster;
Поля заполняются считанными значениями.

Также обновляются поля в структуре module, относящиеся к модулю и информации о модуле из крейта.

5.2.14. LTR22_GetCalibrovka

int LTR22_GetCalibrovka(TLTR22 *module);

Параметры :

- ***module*** - указатель на структуру типа TLTR22

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет подачу запроса модулю на фабричные калибровочные значения. При правильном ответе модуля заполняются массивы:
float FactoryCalibOffset[LTR22_RANGE_NUMBER];
float FactoryCalibScale[LTR22_RANGE_NUMBER];
в структуре TLTR22 *module.

5.2.15. LTR22_GetModuleDescription

int LTR22_GetModileDescription(TLTR22 *module);

Параметры :

- ***module*** - указатель на структуру типа TLTR22

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет подачу запроса модулю на получения описания модуля. При правильном ответе модуля заполняется структура *TDESCRIPTION_LTR22* в структуре TLTR22 *module.

5.2.16. LTR22_StartADC

int LTR22_StartADC(TLTR22 *module, bool Sync)

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***Sync*** - параметр синхронного старта, true – модуль стартует только после получения синхроимпульса (сгенеренного самим модулем, или внешним источником, см. LTR22_SetSyncPriority)

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет старт сбора данных. Во время сбора данных запрещены любые отклики модуля, кроме отсыла самих данных.

Сбор данных может быть как асинхронным – по команде, так и синхронным с ожиданием внешнего сигнала синхронизации.

При синхронном старте первые 37 сэмплов должны отбрасываться, поскольку они относятся к достартовому периоду времени (это связанно с тем, что в АЦП присутствует аппаратный фильтр, и его глубина – 36 значений, и одно значение – добавляется из – за того, что присутствуют фазы сбора данных (сначала 2 и 4 каналы, а потом 1 и 3)).

При асинхронном старте модуль выдает данные со своего входа по приходу команды старт из ПК.

Обновляются поля в структуре TLTR22. DataReadingProcessed

5.2.17. LTR22_StopADC

int LTR22_StopADC(TLTR22 *module)

Параметры :

- ***module*** - указатель на структуру типа TLTR22

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет останов сбора данных. Во время сбора данных запрещены любые отклики модуля, кроме отсыла самих данных. После останова сбора данных рекомендуется запустить команду *LTR22_ClearBuffer()*, которая очистит буфферы сервера и крейта, для того, чтобы избежать ошибок в интерфейсе модуля.

В конце полученных данных присутствует метка об окончании сбора данных – слово `0xf2f38NFE(где N - номер слота в крейте)`, эта метка может автоматически ожидатся и проглатыватся функцией *LTR22_ClearBuffer(true)*.

Обновляются поля в структуре TLTR22. DataReadingProcessed

5.2.18. LTR22_ClearBuffer

int LTR22_ClearBuffer(TLTR22 *module);

Параметры :

- ***module*** - указатель на структуру типа TLTR22

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет очистку буфера крейта и сервера от накопившейся информации после сбора данных. Поскольку используется метка окончания сбора данных высылаемая AVR, то перед использованием этой функции необходимо, чтобы эта метка присутствовала в конце данных, накопившихся в буфере, иначе функция будет ждать ее бесконечно. Метка помещается вызовом функции LTR22_StopADC(TLTR22 *module).

Обновляются поля в структуре module, относящиеся к модулю.

5.2.19. LTR22_Recv

```
int LTR22_Recv(TLTR22 *module, DWORD *data, DWORD *tstamp,
        DWORD size, DWORD timeout);
```

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***data*** – массив DWORD, куда будут складыватся полученные значения
- ***tstamp*** – ссылка на переменную, куда будет записана временная метка.
- ***size*** – размер массива data в DWORD
- ***timeout*** – время, за которое необходимо получить заданное количество данных, в мс.

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается – количество полученных данных, если нет, то возвращаемый результат меньше нуля – содержит код ошибки.

Описание :

Функция осуществляет прием заданного количества данных из модуля. Поскольку сервер буферизирует данные, то могут возникать непредвиденные задержки не более 0.6 с.

Если данные пришли раньше заданного времени, то сервер не ждет до конца заданного периода.

Обновляются поля в структуре module, относящиеся к модулю.

5.2.20. LTR22_ProcessData

```
int LTR22_ProcessData(TLTR22 *module, DWORD *src_data,
                      double *dst_data,  DWORD size,
                      bool calibrMainPset, bool calibrExtraVolts,
                      byte * OverflowFlags);
```

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***src_data*** – массив DWORD, массив исходных данных, только что полученных из модуля, например с помощью комманды *LTR22_Recv*
- ***dst_data*** – массив double, куда будет записыватся результат, каналов по порядку, можно представить массив ***dst_data*** , как ***dst_data[количество включенных каналов][*** size***]***
- ***size*** – размер массива src_data и dst_data в DWORD.
- ***calibrMainPset*** – параметр, определяет фабричную калибровку, true – делать калибровку входящих значений по точкам (после калибровки имеем массив откалиброванных значений точек в идеальных значениях отсчетов АЦП, т.е. 0т==0В, 32767т==MaxVolt, 32768т==MinVolt), false – не делать калибровку.
- ***calibrExtraVolts*** – параметр, определяет пользовательскую калибровку, на выходе – значения в пользовательских параметрах. True – делать калибровку, false – не делать калибровку.
- ***OverflowFlags*** – массив из 4-х значений byte, (вне зависимости от количества включенных каналов) если в процессе обработки данных было обнаружено переполнение, то в OverflowFlags[Номер канала] будет записана 1, иначе будет записан 0.

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет обработку полученных значений,

calibrMainPset=false, calibrExtraVolts=false :

на выходе имеем массив точек, незакалиброванный.

calibrMainPset=true, calibrExtraVolts=false :

на выходе имеем массив точек, закалиброванный фабричными параметрами.

calibrMainPset=true, calibrExtraVolts=true:

на выходе имеем массив точек, закалиброванный фабричными параметрами, а также преобразованный в пользовательский формат.

calibrMainPset=false, calibrExtraVolts=true:

на выходе имеем массив незакалиброванных точек, преобразованный в пользовательский формат.

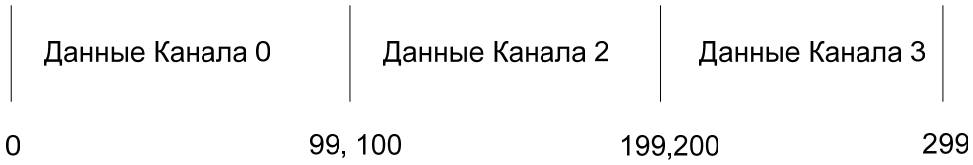
Во время обработки мы подразумеваем, что характеристики модуля линейные, и выходной параметр в таком случае определяется как :

Y=Scale*x+Offset.

Внимание, подразумевается, что размер входного и выходного массива должен быть кратен количеству включенных каналов (функция LTR22_SetADCChannel) в массиве module->ChannelEnabled[LTR22_ADC_NUMBERS], то есть, если включены 0,2,3 каналы, то размер данных должен делится на 3 без остатка.

На выходе данные сортируются по каналам, по порядку, то есть если включены каналы 0,2,3, и получены по 100 результатов на канал, то выходной массив будет иметь вид :

массив ***dst_data :***



Данные о включенных каналах функция берет из массива TLTR22. ChannelEnabled [ADCChannel]

5.2.21. LTR22_ProcessDataTest

```
int LTR22_ProcessData(TLTR22 *module, DWORD *src_data,
                      double *dst_data,  DWORD size,
                      bool calibrMainPset, bool calibrExtraVolts,
                      byte * OverflowFlags, LPCWSTR FilePath);
```

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***src_data*** – массив DWORD, массив исходных данных, только что полученных из модуля, например с помощью комманды *LTR22_Recv*
- ***dst_data*** – массив double, куда будет записыватся результат, каналов по порядку, можно представить массив ***dst_data*** , как ***dst_data[количество включенных каналов][*** size***]***
- ***size*** – размер массива src_data и dst_data в DWORD.
- ***calibrMainPset*** – параметр, определяет фабричную калибровку, true – делать калибровку входящих значений по точкам (после калибровки имеем массив откалиброванных значений точек в идеальных значениях отсчетов АЦП, т.е. 0т==0В, 32767т==MaxVolt, 32768т==MinVolt), false – не делать калибровку.
- ***calibrExtraVolts*** – параметр, определяет пользовательскую калибровку, на выходе – значения в пользовательских параметрах. True – делать калибровку, false – не делать калибровку.
- ***OverflowFlags*** – массив из 4-х значений byte, (вне зависимости от количества включенных каналов) если в процессе обработки данных было обнаружено переполнение, то в OverflowFlags[Номер канала] будет записана 1, иначе будет записан 0.
- ***FilePath*** – путь к файлу, куда будут записываться логи ошибок, при их возникновении

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция аналогичная LTR22_ProcessData(), за исключением того, что при возникновении ошибок они будут логироватся в заданный файл включая дамп всех данных, который используется при возникновении ошибок в интерфейсе модуля при обращении в службу техподдержки.

5.2.22. LTR22_ReadAVREEPROM

```
int LTR22_ReadAVREEPROM(TLTR22 *module, byte *Data,
        DWORD BeginAddress, DWORD size);
```

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***Data*** – массив DWORD, куда будут складыватся полученные значения
- ***BeginAddress*** –начальный адрес, откуда будет производится чтение (0..511)
- ***size*** – количество прочитанных байт, не больше размера массива Data (1..512)

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет чтение заданного количества байт из EEPROM. В модуле LTR22 EEPROM полностью не используется, и целиком отдан в распоряжение пользователя. Размер EEPROM 512 байт.

Обновляются поля в структуре module, относящиеся к модулю.

5.2.23. LTR22_WriteAVREEPROM

```
int LTR22_WriteAVREEPROM(TLTR22 *module, byte *Data,
        DWORD BeginAddress, DWORD size);
```

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***Data*** – массив DWORD, откуда будут братся значения.
- ***BeginAddress*** –начальный адрес, куда будут записыватся значения (0..511)
- ***size*** – количество записываемых байт, не больше размера массива Data (1..512)

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет запись заданного количества байт в EEPROM. В модуле LTR22 EEPROM полностью не используется, и целиком отдан в распоряжение пользователя. Размер EEPROM 512 байт.

Обновляются поля в структуре module, относящиеся к модулю.

5.2.24. LTR22_WriteBroaching

```
int LTR22_WriteBroaching(TLTR22 *module, byte *Data, DWORD size,
        bool WriteCalibrovkaAndDescription, bool ProgrammAVR);
```

Параметры :

- ***module*** - указатель на структуру типа TLTR22
- ***Data*** – массив DWORD, откуда будут братся значения.
- ***size*** – количество записываемых байт, не меньше 8192 (размер массива Data)
- **WriteCalibrovkaAndDescription** – параметр, всегда false, заводские параметры не перезаписываются.
- **ProgrammAVR** – параметр, задает, программировать, или нет AVR, чтобы залить новую прошивку в AVR– true

Возвращаемое значение :

код ошибки, тип int, если функция выполнилась успешно, то возвращается - 0 (LTR_OK).

Описание :

Функция осуществляет перезапись прошивки, сохраняя прежние описание модуля и фабричные калибровки.

Обновляются поля в структуре module, относящиеся к модулю.

5.2.25. LTR22_GetErrorString

LPCSTR LTR22_GetErrorString(int ErrorCode);

Параметры :

- ***ErrorCode*** - код ошибки модуля LTR22

Возвращаемое значение :

Указатель на константную строку, содержащую код ошибки.

Описание :

Функция возвращает строку, содержащую описание ошибки, соответствующее коду ошибки.

Приложения

Приложение 1. Протокол обмена данными с модулем

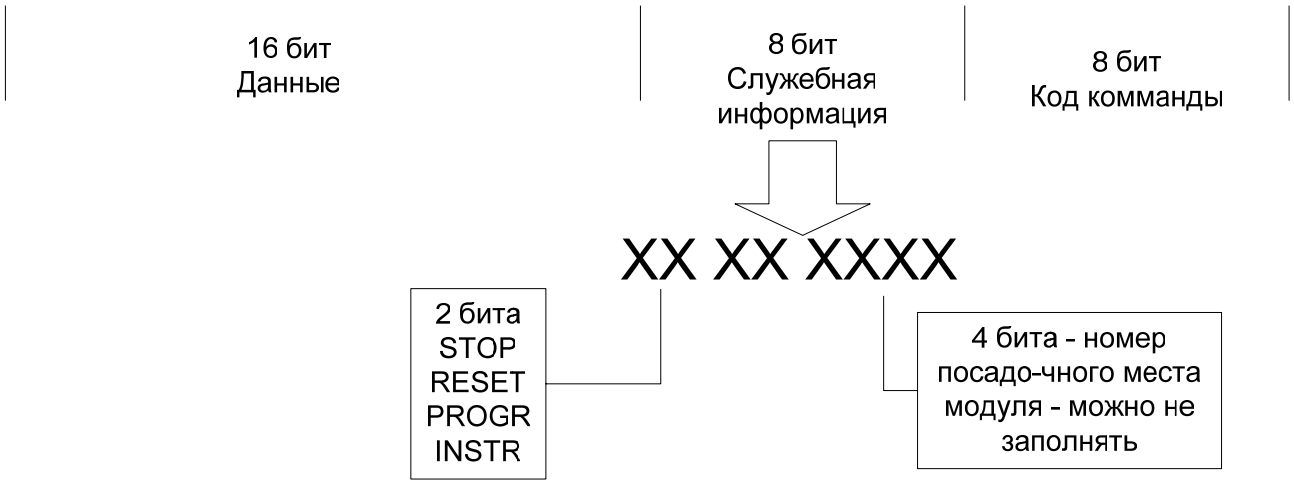
Примечание: Сведения, содержащиеся в данном разделе, являются справочной информацией и не требуются для работы со штатной пользовательской библиотекой.

Протокол обмена данными с модулем основан на использовании формата 4-байтных пакетов команд или данных. Подробно этот формат описан в книге Крейтовая система LTR. Руководство пользователя. Гл. 4.3. Здесь остановимся только на информации, имеющей значение применительно к модулю LTR22.

Все команды от крейт-контроллера к модулю и подтверждения этих команд представляют собой 4-байтные **командные слова**. Данные, считанные с А и передаваемые из модуля в крейт-контроллер, представляют собой 4-байтные **слова данных**. Помимо данных с АЦП модуль шлет еще и командные слова. Следует отметить, что указанный протокол является общим для всех модулей данной крейтовой системы.

Поскольку в модуле присутствуют только АЦП в модуль можно отсылать только команды.

Рис. 1.1п. Формат **командного слова** отсылаемого в модуль (32 бита).



Команды PROGR STOP RESET – общие для всей крейтовой системы. Их описание – в руководстве пользователя. С кодом команды INSTR отсылается команда в AVR, где:

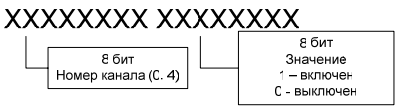
Код команды – номер команды в AVR.

Данные – данные по команде в AVR.

Таблица 1.1п. Список команд отсылаемых в AVR.

№ ком -ды	Описание	Данные отсылаемые	Принимаемые данные
1	Старт – стоп измерения нуля Во время сбора данных менять	1 – стоп 0 – старт	

	нельзя		
2	Управление отсечкой постоянной составляющей	0 – режим AC 1 – режим AC+DC	
3	Установка диапазонов каналов. Допустимые значения диапазона : 0 – ±1 В 1 – ±0.3 В 2 – ±0.1 В 3 – ±0.03 В 4 – ±10 В 5 – ±3 В	XXXXXXXX XXXXXXXX 8 бит Номер канала (C..4) 8 бит Номер диапазона (C..5) Старшие 8 бит – номер канала. Младшие 8 бит – номер диапазона.	
4	Установка основного делителя тактовой частоты АЦП. $F_{CLKIN} = \frac{20 * 10^6}{fdiv\_main}$ Гц	fdiv_main=0..15	
5	Установка дополнительного делителя частоты сэмплов $F_{LRCK} = \frac{F_{CLKIN}}{128 \cdot (fdiv\_extr\_bool + 2)}$ *Samples*, *Гц*	fdiv_extra_bool=0..1	
6	Считать внутренние регистры состояния AVR		Команды AVR: 31 32 33 34
7	Считать калибровочные значения. Получаем массив 8-ми битных значений, затем его преобразуем в массив float[48].		Команды AVR: 7 192 команды. В поле данных: 8 младших бит – значения калибровки 8 старших бит – номер в массиве(0..191)
8	Считать описание модуля		Команды AVR: 8 128 команд. В поле данных: 8 младших бит – значения описания 8 старших бит – номер в массиве(0..127)
9	Записать начальный адрес записи или считывания в EEPROM	Адрес, 16 бит (0..512)	

10	Запись байта в EEPROM, по адресу, отосланному в команде 9, при этом адрес в AVR инкрементируется на 1	Данные, 8 бит	
11	Считывание данных из EEPROM, начальный адрес считывания задается командой 9	Количество данных, для считывания	Команды AVR: 11 По количеству считываемых значений, В поле данных: 8 младших бит – значение в ячейке памяти, старшие – младшие 8 бит порядкового номера в передаваемом массиве
12	Установка разрешения канала АЦП на отсыл во время сбора данных	 Старшие 8 бит – номер канала. Младшие 8 бит – параметр	
14	Запись регистра состояния АЦП	Смотри команду 31	
15	Запись регистра частоты АЦП	Смотри команду 32	
16	Запись регистра диапазонов АЦП	Смотри команду 33	
17	Запись регистра каналов АЦП и синхронизации	Смотри команду 34	
18	Фазировка модуля с другими модулями LTR22, или с внешними источниками синхросигналов.	При запуске фазировки, AVR входит в состояние бесконечного ожидания синхросигнала, вне зависимости от того, генерирует синхроимпульс, или нет. По приходу синхроимпульса, AVR ждет время инициализации АЦП, после этого высылает команду подтверждение прихода синхроимпульса. – повторяет во второй раз команду отклика начала выполнения текущей команды, но с кодом 27, а не 26	
19	Выбор Master – Slave при	1 – Master	

	фазировке и синхронном запуске АЦП	0 – Slave	
40	Старт сбора данных	0 – Асинхронный старт 1 – Синхронный старт	Отклик – не высылается, при синхронном старте откликом служат данные
41	Стоп сбора данных		После данных высылается команда подтверждение 0xf2f38NFE, где N – номер слота в крейте
63	Команда отклика модуля Для проверки правильности соединения		Слово 0xF2F38NF1, где N – номер слота в крейте

Примечание : Если не запущен сбор данных, то после каждой отосланной команды в модуль, модуль отсылает команду подтверждения начала исполнения команды (код команды 26).

В ответ на некоторые команды модуль, при выключенном режиме сбора данных отсылает не только команду подтверждения, но и данные.

Рис. 1.2п. Формат **командного слова** принимаемого из модуля (32 бита).

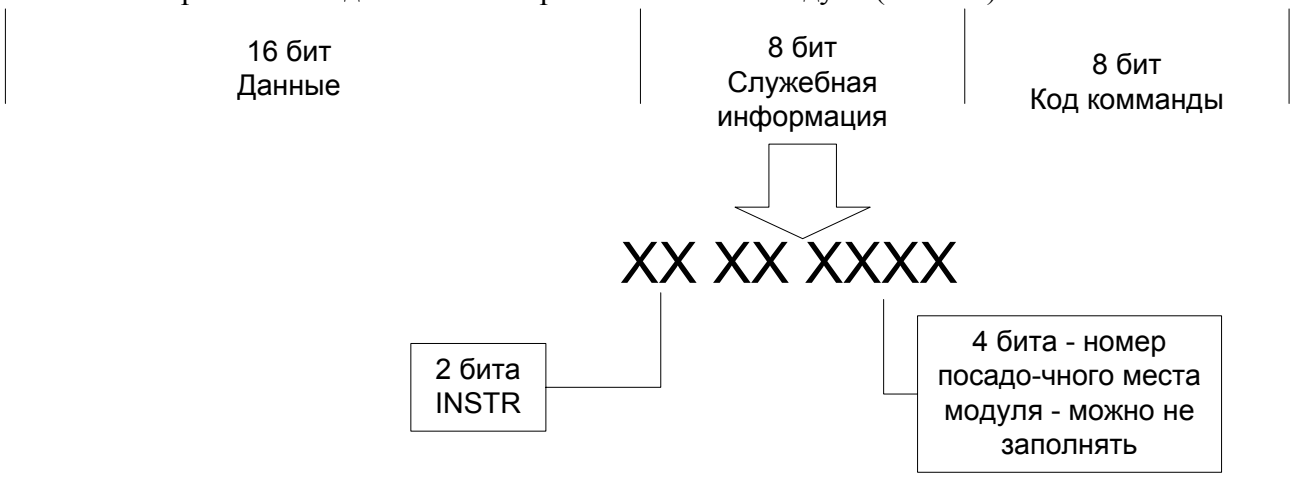
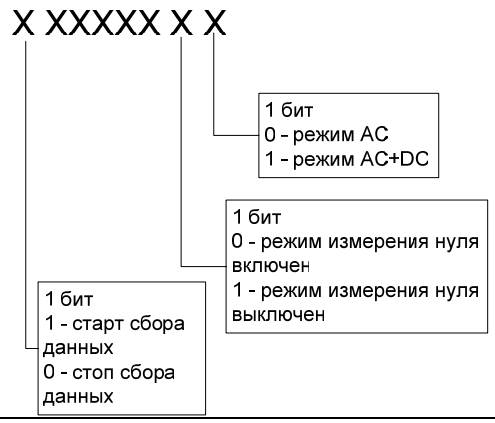
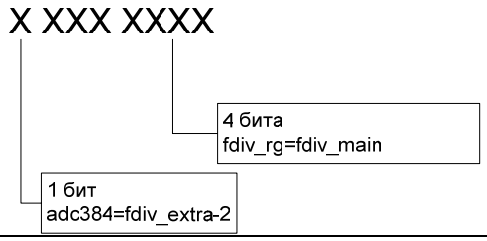
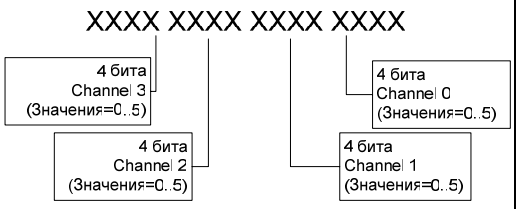
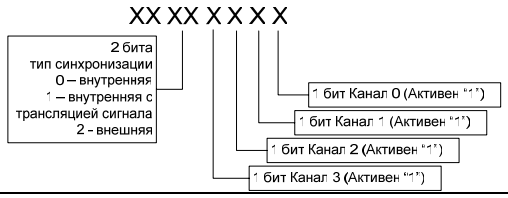
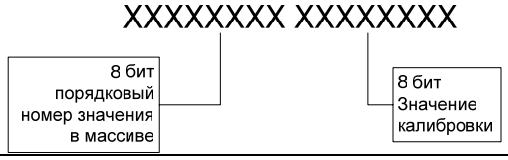
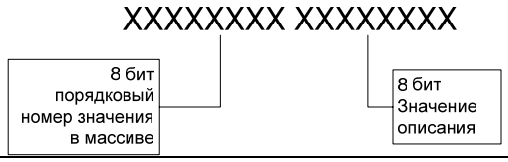


Таблица 1.2п. Список команд принимаемых из AVR.

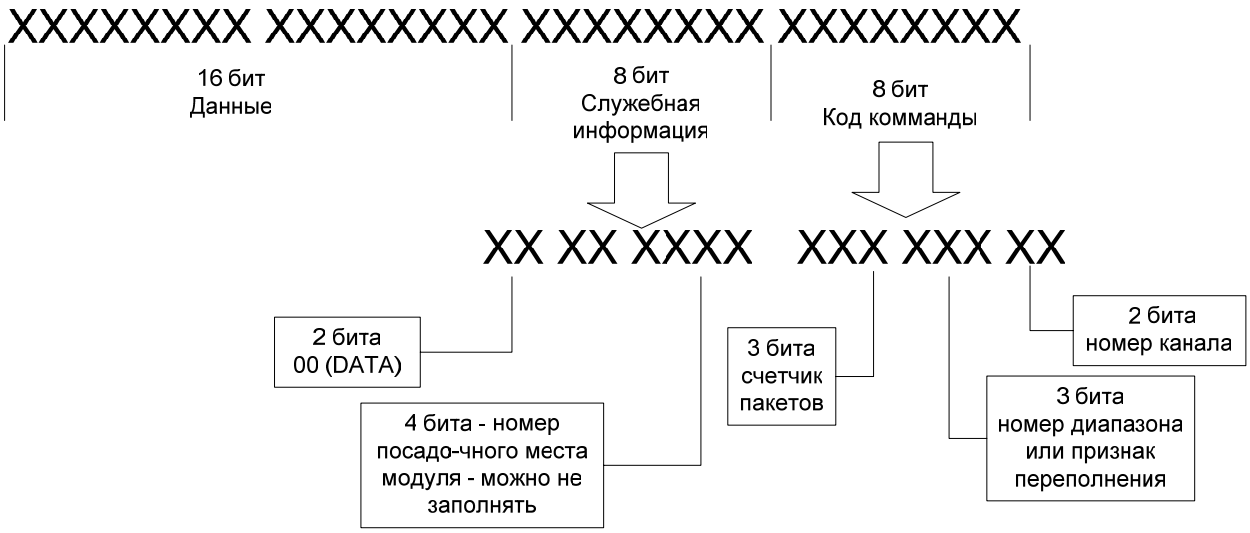
№ ком-ды	Описание	Данные принимаемые	Команда запрос
26	Отклик начала выполнения команды	Номер выполняемой команды	
27	Отклик команды фазировки о том, что фазировка закончена, необходима, поскольку	Номер выполняемой команды	

	фазировка при бесконечном ожидании синхроимпульса может затянутся		
31	Регистр управления АЦП		6
32	Регистр управления частотой АЦП		6
33	Регистр управления диапазонами каналов 0 – ±1 В 1 – ±0.3 В 2 – ±0.1 В 3 – ±0.03 В 4 – ±10 В 5 – ±3 В		6
34	Регистр управления передачей каналов и управления синхронизацией		6
7	Данные калибровки		7
8	Данные описания		8

11	Считывание байта из EEPROM	XXXXXXXX XXXXXXXX 8 бит порядковый номер значения в передаваемом массиве 8 бит Значение ячейки	11
241	Команда отклика	0xF2F3	63
254	Команда – подтверждение окончания сбора данных	0xF2F3	0

Примечание: Команды из AVR отсылаются только, если не запущен сбор данных.

При сборе данных модулем передаются только **слова данных** (32 бита)**:**



Счетчик пакетов – предназначен для контроля последовательности принимаемых данных.
Номер диапазона – номер диапазона для канала
Номер канала – номер канала АЦП

При останове сбора данных – высылается слово подтверждение окончания сбора данных, присутствующее в конце полученных данных - `0xf2f38NFE (где N – номер слота в крейте)` таким образом можно узнать о конце потока данных из модуля после окончания сбора данных.

После отсыла команды RESET в модуль необходимо ждать не менее 110 мс, пока произойдет автоматическая калибровка нуля АЦП и первоначальная инициализация фильтра.

Приложение 2. Файл «ltr22api.h»

```
#include "ltr\\include\\ltrapitypes.h"
#include "ltr\\include\\ltrapi.h"

#ifndef __LTR22API__
#define __LTR22API__

#ifdef LTR22API_EXPORTS
  #define LTR22API_DllExport(type)   __declspec(dllexport) type APIENTRY
#else
  #define LTR22API_DllExport(type)   __declspec(dllimport) type APIENTRY
#endif

#define LTR22_MODULE_CODE 0x1616 //
#define LTR22_ADC_NUMBERS 4
#define LTR22_ADC_CHANNELS LTR22_ADC_NUMBERS
#define LTR22_RANGE_NUMBER 6
#define LTR22_RANGE_OVERFLOW 7

#define LTR22_MAX_DISC_FREQ_NUMBER 25    // количество выставляемых частот
lbcrhtnbpfwbb

typedef BYTE byte;

#ifndef __cplusplus
        #define true 1
        #define false 0
        typedef byte bool;
#endif


typedef struct
{
        TDESCRIPTION_MODULE Description;      // описание модуля
        TDESCRIPTION_CPU CPU;                         // описание AVR

} TINFO_LTR22, *PTINFO_LTR22;


typedef struct
{
        float FactoryCalibOffset[LTR22_RANGE_NUMBER];
        float FactoryCalibScale[LTR22_RANGE_NUMBER];

        float UserCalibOffset[LTR22_RANGE_NUMBER];
        float UserCalibScale[LTR22_RANGE_NUMBER];

} ADC_CHANNEL_CALIBRATION;


//**** конфигурация модуля
typedef struct
```

```
{
  //**** служебная информация       //
  INT size;                                                    // размер структуры TLTR22 1036
байт
  TLTR Channel;                 // служебная структура для работы с крейтом через сервер

  // настройки модуля
  byte Fdiv_rg;                                          // дивайзер частоты клоков 1..15
  bool Adc384;                                           // дополнительный дивайзер частоты
сэмплов true =3 false =2
  bool AC_DC_State;                                      // состояние true =AC+DC false=AC
  bool MeasureADCZero;                                   // измерение Zero true - включено false -
выключено
  bool DataReadingProcessed;                 // состояние считывания АЦП true-АЦП
считывается false - нет
  byte   ADCChannelRange[LTR22_ADC_NUMBERS];// предел имзерений АЦП по каналам 0 -
1В 1 - 0.3В 2 - 0.1В 3 - 0.03В 4 - 10В 5 - 3В

  bool ChannelEnabled[LTR22_ADC_NUMBERS];          // Состояние каналов, включен - true
выключен - false

  int FreqDiscretizationIndex;                 // частота дискретизации, выставленная сейчас
0..24 - в зависимости от частоты
                                                                  // из массива
LTR22_DISK_FREQ_ARRAY

  byte SyncType;                // Тип синхронизации 0 - внутренний старт по сигналу Go
                                //1 - фазировка модуля
                                //2 - внешний старт
                                //3 - резервировано
  bool SyncMaster;              // true - модуль генерит сигнал, false - модуль принимает
синхросигнал

  TINFO_LTR22 ModuleInfo;
  ADC_CHANNEL_CALIBRATION
ADCCalibration[LTR22_ADC_NUMBERS][LTR22_MAX_DISC_FREQ_NUMBER];

} TLTR22,*PTLTR22;

// возможные варианты ошибок
// константы
#define LTR22_ERROR_SEND_DATA                                  (-
6000)
#define LTR22_ERROR_RECV_DATA                                  (-
6001)
#define LTR22_ERROR_NOT_LTR22                                  (-
6002)
#define LTR22_ERROR_OVERFLOW                                   (-
6003)
#define LTR22_ERROR_CANNOT_DO_WHILE_ADC_RUNNING                (-6004)
#define LTR22_ERROR_MODULE_INTERFACE                           (-6005)
#define LTR22_ERROR_INVALID_FREQ_DIV                           (-6006)
#define LTR22_ERROR_INVALID_TEST_HARD_INTERFACE                (-6007)
```

```
#define LTR22_ERROR_INVALID_DATA_RANGE_FOR_THIS_CHANNEL          (-6008)
#define LTR22_ERROR_INVALID_DATA_COUNTER                         (-6009)
#define LTR22_ERROR_PRERARE_TO_WRITE                             (-6010)
#define LTR22_ERROR_WRITE_AVR_MEMORY                             (-6011)
#define LTR22_ERROR_READ_AVR_MEMORY                                  (-
6012)
#define LTR22_ERROR_PARAMETERS                                       (-
6013)


/*
-----------------------------------------------------------------------------
 декларирование функций
*/
#ifdef __cplusplus
extern "C" {  // only need to export C interface if
            // used by C++ source code
#endif

LTR22API_DllExport(INT) LTR22_Init(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_Close(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_Open(TLTR22 *module, DWORD saddr, WORD sport, CHAR
*csn, WORD cc);
LTR22API_DllExport(INT) LTR22_IsOpened(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_GetConfig(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_SetConfig(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_ClearBuffer(TLTR22 *module, bool wait_response);
LTR22API_DllExport(INT) LTR22_StartADC(TLTR22 *module, bool WaitSync);
LTR22API_DllExport(INT) LTR22_StopADC(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_SetSyncPriority(TLTR22 *module, bool SyncMaster);
LTR22API_DllExport(INT) LTR22_SyncPhaze(TLTR22 *module, DWORD timeout);
LTR22API_DllExport(INT) LTR22_SwitchMeasureADCZero(TLTR22 *module, bool SetMeasure);
LTR22API_DllExport(INT) LTR22_SetFreq(TLTR22 *module, bool adc384, byte Freq_dv);
LTR22API_DllExport(INT) LTR22_SwitchACDCState(TLTR22 *module, bool ACDCState);
LTR22API_DllExport(INT) LTR22_SetADCRange(TLTR22 *module, byte ADCChannel, byte
ADCChannelRange);
LTR22API_DllExport(INT) LTR22_SetADCChannel(TLTR22 *module, byte ADCChannel, bool
EnableADC);
LTR22API_DllExport(INT) LTR22_GetCalibrovka(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_Recv(TLTR22 *module, DWORD *data, DWORD *tstamp,
DWORD size, DWORD timeout);
LTR22API_DllExport(INT) LTR22_GetModuleDescription(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_ProcessData(TLTR22 *module, DWORD *src_data, double
*dst_data,
                                                          DWORD size, bool
calibrMainPset, bool calibrExtraVolts, byte * OverflowFlags);
LTR22API_DllExport(INT) LTR22_ReadAVREEPROM(TLTR22 *module, byte *Data, DWORD
BeginAddress, DWORD size);
LTR22API_DllExport(INT) LTR22_WriteAVREEPROM(TLTR22 *module, byte *Data, DWORD
BeginAddress, DWORD size);

/*
тестовые функции
```

```
*/
LTR22API_DllExport(INT) LTR22_TestHardwareInterface(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_GetADCData(TLTR22 *module, double * Data, DWORD Size,
DWORD time,
                                                                    bool calibrMainPset, bool
calibrExtraVolts);
LTR22API_DllExport(INT) LTR22_ReopenModule(TLTR22 *module);
LTR22API_DllExport(INT) LTR22_ReadAVRBroaching(TLTR22 *module, byte *Data, DWORD
size, DWORD BeginPage, DWORD PageNumbers);


LTR22API_DllExport(INT) LTR22_WriteBroaching(TLTR22 *module, byte *Data, DWORD size,
                                                                          bool
WriteCalibrovkaAndDescription, bool ProgrammAVR);

LTR22API_DllExport(LPCSTR) APIENTRY LTR22_GetErrorString(int ErrorCode);
#ifdef __cplusplus
}
#endif
#endif
```

Приложение 3. Файл «ltrapi.h»

```
//---------------------------------------------------------------------------
// common part
//---------------------------------------------------------------------------
#ifndef __ltrapi__
#define __ltrapi__
#include <windows.h>
#include "ltr\\include\\ltrapidefine.h"
#include "ltr\\include\\ltrapitypes.h"

#ifdef LTRAPIWIN_EXPORTS
  #define LTRAPIWIN_DllExport(type)   __declspec(dllexport)  type APIENTRY
#else
  #define LTRAPIWIN_DllExport(type)   __declspec(dllimport) type APIENTRY
#endif


// коды ошибок
#define LTR_OK                  ( 0l)    /*Выполнено без ошибок.*/
#define LTR_ERROR_UNKNOWN       (-1l)    /*Неизвестная ошибка.*/
#define LTR_ERROR_PARAMETRS     (-2l)    /*Ошибка входных параметров.*/
#define LTR_ERROR_MEMORY_ALLOC  (-3l)    /*Ошибка динамического выделения
памяти.*/
#define LTR_ERROR_OPEN_CHANNEL  (-4l)    /*Ошибка открытия канала обмена с
сервером.*/
#define LTR_ERROR_OPEN_SOCKET   (-5l)    /*Ошибка открытия сокета.*/
#define LTR_ERROR_CHANNEL_CLOSED (-6l)   /*Ошибка. Канал обмена с сервером не
создан.*/
#define LTR_ERROR_SEND          (-7l)    /*Ошибка отправления данных.*/
#define LTR_ERROR_RECV          (-8l)    /*Ошибка приема данных.*/
#define LTR_ERROR_EXECUTE       (-9l)    /*Ошибка обмена с крейт-контроллером.*/
//---------------------------------------------------------------------------
#pragma pack(4)                 //
typedef struct {                //
 DWORD saddr;                   // сетевой адрес сервера
 WORD  sport;                   // сетевой порт сервера
 CHAR  csn[16];                 // серийный номер крейта
 WORD  cc;                      // номер канала крейта
 DWORD flags;                   // флаги состояния канала
 DWORD tmark;                   // последняя принятая метка времени
 //                             //
 LPVOID internal;               // указатель на канал
} TLTR;                         //
#pragma pack()                  //
//---------------------------------------------------------------------------

#ifdef __cplusplus
extern "C" {
#endif
LTRAPIWIN_DllExport(INT) LTR_Init(TLTR *ltr);
LTRAPIWIN_DllExport(INT) LTR_Open(TLTR *ltr);
LTRAPIWIN_DllExport(INT)  LTR_Close(TLTR *ltr);
```

```
LTRAPIWIN_DllExport(INT)  LTR_IsOpened(TLTR *ltr);
LTRAPIWIN_DllExport(INT)  LTR_Recv(TLTR *ltr, DWORD *data, DWORD *tmark, DWORD size,
DWORD timeout);
LTRAPIWIN_DllExport(INT)  LTR_Send(TLTR *ltr, DWORD *data, DWORD size, DWORD
timeout);
//
LTRAPIWIN_DllExport(INT)  LTR_GetCrates(TLTR *ltr, BYTE *csn);
LTRAPIWIN_DllExport(INT)  LTR_GetCrateModules(TLTR *ltr, WORD *mid);
//
LTRAPIWIN_DllExport(INT)  LTR_SetServerProcessPriority(TLTR *ltr, DWORD Priority);
//
LTRAPIWIN_DllExport(LPCSTR)  LTR_GetErrorString(INT error);
#ifdef __cplusplus
}
#endif

#endif
```

Приложение 4. Файл «ltraptypes.h»

```
#ifndef __ltrapitypes__
#define __ltrapitypes__
#include <windows.h>
#ifdef __cplusplus
extern "C" {
#endif
//
#ifndef COMMENT_LENGTH
#define COMMENT_LENGTH 256
#endif
#ifndef ADC_CALIBRATION_NUMBER
#define ADC_CALIBRATION_NUMBER 256
#endif
#ifndef DAC_CALIBRATION_NUMBER
#define DAC_CALIBRATION_NUMBER 256
#endif
//
#pragma pack(4)
// описание модуля
typedef struct _DESCRIPTION_MODULE_                    //
{                                                      //
  BYTE   CompanyName[16];                              //
  BYTE   DeviceName[16];                               // название изделия
  BYTE   SerialNumber[16];                             // серийный номер изделия
  BYTE   Revision;                                     // ревизия изделия
  BYTE   Comment[COMMENT_LENGTH];                      //
} TDESCRIPTION_MODULE;                                 //
// описание процессора и програмного обеспечения
typedef struct _DESCRIPTION_CPU_                       //
{                                                      //
  BYTE   Active;                                       // флаг достоверности остальных полей структуры
  BYTE   Name[16];                                     // название
  double ClockRate;                                    //
  DWORD  FirmwareVersion;                              //
  BYTE   Comment[COMMENT_LENGTH];                      //
} TDESCRIPTION_CPU;                                    //
// описание плис
typedef struct _DESCRIPTION_FPGA_                      //
{                                                      //
  BYTE   Active;                                       // флаг достоверности остальных полей структуры
  BYTE   Name[16];                                     // название
  double ClockRate;                                    //
  DWORD  FirmwareVersion;                              //
  BYTE   Comment[COMMENT_LENGTH];                      //
} TDESCRIPTION_FPGA;                                   //
// описание ацп
typedef struct _DESCRIPTION_ADC_                       //
{                                                      //
  BYTE   Active;                                       // флаг достоверности остальных полей структуры
  BYTE   Name[16];                                     // название
  double Calibration[ADC_CALIBRATION_NUMBER];          // корректировочные коэффициенты
  BYTE   Comment[COMMENT_LENGTH];                      //
} TDESCRIPTION_ADC;                                    //
// описание цап
typedef struct _DESCRIPTION_DAC_                       //
{                                                      //
  BYTE   Active;                                       // флаг достоверности остальных полей структуры
```

```
  BYTE   Name[16];                                  // название
  double Calibration[DAC_CALIBRATION_NUMBER];       // корректировочные коэффициенты
  BYTE   Comment[COMMENT_LENGTH];                   //
} TDESCRIPTION_DAC;                                 //
// описание h-мезанинов
typedef struct _DESCRIPTION_MEZZANINE_              //
{                                                   //
  BYTE   Active;                                    // флаг достоверности остальных полей структуры
  BYTE   Name[16];                                  // название изделия
  BYTE   SerialNumber[16];                          // серийный номер изделия
  BYTE   Revision;                                  // ревизия изделия
  double Calibration[4];                            // корректировочные коэффициенты
  BYTE   Comment[COMMENT_LENGTH];                   // комментарий
} TDESCRIPTION_MEZZANINE;                           //
// описание цифрового вв
typedef struct _DESCRIPTION_DIGITAL_IO_             //
{                                                   //
  BYTE   Active;                                    // флаг достоверности остальных полей структуры
  BYTE   Name[16];                                  // название ???????
  WORD   InChannels;                                // число каналов
  WORD   OutChannels;                               // число каналов
  BYTE   Comment[COMMENT_LENGTH];                   //
} TDESCRIPTION_DIGITAL_IO;                          //
// описание интерфейсных модулей
typedef struct _DESCRIPTION_INTERFACE_              //
{                                                   //
  BYTE   Active;                                    // флаг достоверности остальных полей структуры
  BYTE   Name[16];                                  // название
  BYTE   Comment[COMMENT_LENGTH];                   //
} TDESCRIPTION_INTERFACE;                           //
#pragma pack()
//
#ifdef __cplusplus
}
#endif
#endif
```

Приложение 5. Пример программы

```
/*
	Тестовая программа для LTR22
	Выполняются элементарные действия над модулем,
	(модуль открывается, конфигурируется,
	проверяется старт - стоп сбора данных,
	проверяется запись в EEPROM модуля)
*/

#include "stdafx.h"
#include <iostream>
#include "ltr\\include\\ltr22api.h"

using namespace std;

int TestADCStartStop(TLTR22 * module)
{
	DWORD rbuf[100];
	double rbuf2[100];
	int res=0;

	if ((res=LTR22_StartADC(module,false))==LTR_OK) // старт сбора данных
	{
		if (LTR22_Recv(module, rbuf, NULL, 100, 500)==100) // получение данных
		{
			if ((res=LTR22_StopADC(module))==LTR_OK) // выключение сбора данных
			{
				if ((res=LTR22_ClearBuffer(module,true))==LTR_OK) // очистка буфферов

				{
					if ((res=LTR22_ProcessData(module, rbuf, rbuf2, 100, false, false,
NULL))!=LTR_OK)// преобразование данных в обычный формат
						cout<<"Error Process Data"<<res<<endl;
				}else cout<<"Error Clear buffers"<<res<<endl;
			}else cout<<"Error stop ADC"<<res<<endl;
		}else cout<<"Error Recieve ADC data"<<res<<endl;
	}else cout<<"Error start ADC"<<res<<endl;

	return res;
}


int _tmain(int argc, _TCHAR* argv[])
{
	TLTR22* module = new TLTR22();		// создаем структуру в памяти

	int res=0;

	memset(module,0,sizeof(TLTR22));

	res = LTR22_Close(module);

	res=LTR22_Open(module, module->Channel.saddr, module->Channel.sport, module->Channel.csn,
8)

	if (res==LTR_WARNING_MODULE_IN_USE)
	{
```

```
                cout<<"Warning Module already opened anywhere"<<endl;
                res = LTR_OK;
        }

        if (res==LTR_OK) // открываем модуль
        {
                if ((res=LTR22_IsOpened(module))!=LTR_OK) cout<<res<<endl;
                else cout<<"Module Opened Successfull"<<endl;

                res = LTR22_SetFreq(module,false, 1);

                // установили рабочие диапазоны модуля
                if ((res=LTR22_SetADCRange(module, 0, 3))!=LTR_OK) cout<<"Error set ADC
Range"<<res<<endl;
                if ((res=LTR22_SetADCRange(module, 1, 2))!=LTR_OK) cout<<"Error set ADC
Range"<<res<<endl;
                if ((res=LTR22_SetADCRange(module, 2, 1))!=LTR_OK) cout<<"Error set ADC
Range"<<res<<endl;
                if ((res=LTR22_SetADCRange(module, 3, 0))!=LTR_OK) cout<<"Error set ADC
Range"<<res<<endl;

                // считываем значения из модуля, и смотрим, что все считалось правильно
                if ((res=LTR22_GetConfig(module))!=LTR_OK) cout<<"Error get Module
Config"<<res<<endl;

                // тестирование старта - стопа обработки данных
                if ((res=TestADCStartStop(module))!=LTR_OK) cout<<"Error
TestStartStopTest"<<res<<endl;
                else cout<<"Test ADC StartStop Successfull"<<endl;

                // получаем информацию о модуле
                if ((res=LTR22_GetModuleDescription(module))!=LTR_OK) cout<<"Error get module
Description"<<res<<endl;

                // получаем калибровки модуля
                if ((res=LTR22_GetCalibrovka(module))!=LTR_OK) cout<<"Error get module
Calibration"<<res<<endl;


                byte EEPROMData[512];
                for (int i=0;i<sizeof(EEPROMData);i++)
                        EEPROMData[i]=i;

                // перезаписываем EEPROM
                if ((res=LTR22_WriteAVREEPROM(module, EEPROMData,0,512))!=LTR_OK) cout<<"Error
Write AVR EEPROM"<<res<<endl;

                memset(EEPROMData,0,sizeof(EEPROMData));

                // считываем EEPROM
                if ((res=LTR22_ReadAVREEPROM(module, EEPROMData,0,512))!=LTR_OK) cout<<"Error
Read AVR EEPROM"<<res<<endl;

                // закрываем интерфейс с модулем
                if ((res=LTR22_Close(module))!=LTR_OK) cout<<res<<endl;
        }else cout<<"Open Module failed"<<endl;

        if (res==LTR_OK)cout<<"All Right -)"<<endl;
        cout<<"Enter any value and press enter"<<endl;
```

```
        cin>>res;
        return 0;
}
```