Multichannel data-acquisition systems

User interface library of LTR27 module

Programmer manual

Revision 1.0.3
January 2010

L-CARD

http://en.lcard.ru
en@lcard.ru

DAQ SYSTEMS DESIGN, MANUFACTURING & DISTRIBUTION

Author of the manual:
A.V. Kodorkin

L-Card LLC
117105, Moscow, Varshavskoye shosse, 5, block 4, bld. 2

tel.: +7 (495) 785-95-19
fax: +7 (495) 785-95-14

Internet contacts:
http://en.lcard.ru/

E-Mail:
Sales department: en@lcard.ru
Customer care: en@lcard.ru

Revision history of this document.

Revision	**Date**	**Notes to the updates**
1.0.0	23.04.2006	The first revision available for user
1.0.1	21.11.2006	TLTR27 structure has been changed – module description has been given as well as the module description reading functions have been changed
1.0.2	23.04.2007	the examples for supporting of error LTR_WARNING_MODULE_IN_USE have been changed
1.0.3	14.01.2010	The information about submodules has been corrected.

The latest revision of this document is always being written on the CD-ROM included in the delivery package. Moreover you can find the latest revision in the section of *the files library* on our website.

L-Card reserves the right to update the documentation without notifying the users.

Contents

1. What this document is about. 5
2. General ideology of the user interface for manipulating with module LTR27. 6
3. *Ltr27api.dll* - library for manipulating with module LTR27. 7
   3.1. Use of ltr27api.dll. 7
   3.2. General approach to working with the interface functions of library ltr27api.dll. 7
   3.3. Simple example. 8
4. Description of functions, structures and constants of *ltr27api.dll* library. 9
   4.1. Constants 9
   4.2. Structures 10
      4.2.1. Structure TLTR27 10
      4.2.2. Structure of TINFO_LTR27 12
   4.3. Function 14
      4.3.1. Initialization functions of working with the module. 14
      4.3.2. Functions to manipulate with module ADC 16
      4.3.3. Information type functions 19
      4.3.4. Auxiliary functions 20
5. Annex 21
   5.1. Module interconnect protocol. Command and data formats. 21
      5.1.1. Module interconnect protocol. 21
      5.1.2. Formats of commands and data Format of data words: 21

1. What this document is about.

This document is a Programmer manual. The general ideology of software building for manipulating with module **LTR27** is provided as well as the detailed description of dll-library ***ltr27api.dll***.

Any issues related to the signals connection, parameters and hardware operation principles are not considered in this document. These issues are provided in the document *LTR Crate System. User Manual*. Moreover the current document does not have the description of the libraries for handling of other types of modules and the crate as a whole, they are provided in separate *documents*.

2. General ideology of the user interface for manipulating with module LTR27.

From the user software point of view the module**LTR27** is a 16-channel 16-bit* ADC with configurable sampling frequency. The module can be in one of the two following states: *waiting state* or *data acquisition state*.

In *waiting state* the module receives and processes the commands: reading information about module instance, reading/writing of data acquisition parameters, test command and *data acquisition state command*.

In *data acquisition state* the module performs simultaneous capture of all 16 analog channels (in accordance with configurated parameters) and outputs the values obtained. In addition, on any command the module stops data acquisition and is switched in *waiting state* to process the received command.

Thus, there are three basic steps in manipulating with module **LTR27**:

- Receiving the information about the module instance.
- Setting up the parameters and starting of data acquisition.
  - Reading and post-processing of ADC data.

Note: ADC effective bit depth depends on selected sampling frequency and varies about from 8 bits at frequency of 1 kHz to 16 bits at frequency of 4 Hz. For more details about effective sampling frequency see document *ELECTRICAL MEASURING TRANSDUCERS OF H-27 TYPE*.

3. *Ltr27api.dll* - library for manipulating with module LTR27.

Library ***ltr27api.dll*** presents the set of functions for manipulating with module **LTR27.** The library is written based on api-function calls of basic library of working with ltr-crate***ltrapi.dll*** using programming language **Borland C++** and is come with source texts.

Attention: Strictly speaking, the library functions do not provide "thread safe" operation. So, for the sake of good order, the user shall organize by himself in multi-threaded contexts the correct synchronization of interface function calls in different threads (using, for example, , critical sections, mutexes and, etc.), if required.

3.1. Use of ltr27api.dll.

To call the interface functions of library ***`ltr27api.dll`*** from your application the following is required:

- to create the project in any of the development environment;
- to insert files **ltrapi.dl** and ***ltr27api.dll*** in the project folder or the folder described in ***PATH environment variable***.
- to add information on interface function call of dll-library and used data types to the project. The sequence of actions and applied force may vary in different development environments: ***Borland C++/Borland C++ Builder*** :
  - Attach files ***LTR\LIB\BORLAND\ltr27api.lib, LTR\INCLUDE\ltr27api.h*** to the project. ***Microsoft Visual C++ :***
  - Attach files ***LTR\LIB\MSVC\ltr27api.lib, LTR\INCLUDE\ltr27api.h*** to the project. ***Other development environments:***
  - It is necessary to refer to the corresponding documentation of the development tool.

- create and add the file containing the source text of the future program to the project; then you can write
- the program calling the corresponding interface functions of dll-library.

3.2. General approach to working with the interface functions of library ltr27api.dll.

The following actions shall be made in order to interact with **LTR27** module:

- Create the instance of the *TLTR27* structure and initialize it by calling the function *LTR27_Init()*.
- Make connection with the required module by calling the function *LTR27_Open()*. - Read the module configuration by calling the function *LTR27_GetConfig()*.
- Read the description of the module and installed submodules by calling function *LTR27_GetModuleDescription()*.
- Set the parameters of data acquisition and transmit them to the module using function *LTR27_SetConfig()*.
- Start ADC data acquisition using the function *LTR27_ADCStart()*.
- Capture ADC data at times by calling function *LTR27_Recv()*.
- Select ADC data of targeted submodule, apply the calibration coefficients and convert ADC code in physical values by calling function *LTR27_ProcessData()*.
- Stop ADC data acquisition by calling function *LTR27_ADCStop()*.
- Close connection with the module by calling the function *LTR27_Close()*.

3.3. Simple example.

```
//-----------------------------------------------------------------------------------
// This example describes the configuration of module LTR27 and collection
// of ADC 1024 samples with their subsequent correcting and displaying
//----------------------------------------------------------------------------------- #include
<stdio.h>
#include "ltr\\include\\ltr27api.h"
#define NSAMPLES (2*LTR27_MEZZANINE_NUMBER*1024) int
main(void)
{
 INT res, size;
 TLTR27 ltr27;
 // initialize the structure fields with values by default
res=LTR27_Init(&ltr27);  if(res==LTR_OK) {
    // make connection with the module in the first slot of crate.
    // for network address, network port of ltr-server and serial number
    // of crate the values by default should be used
    res=LTR27_Open(&ltr27, SADDR_DEFAULT, SPORT_DEFAULT, “”, CC_MODULE1);
if (res==LTR_WARNING_MODULE_IN_USE)
    {
       oem_printf(">> Warning, module Already Opened         \n");
       res = LTR_OK;
    }
    if(res==LTR_OK) {
       // obtain module configuration
res=LTR27_GetConfig(&ltr27);
        if(res==LTR_OK) {
           // read the description of module and submodules
res=LTR27_GetModuleDescription(&ltr27, LTR27_ALL_DESCRIPTION);       if(res==LTR_OK) {
              // select sampling frequency of 100 Hz
ltr27.FrequencyDivisor=9;
             // copy the calibration coefficients              for(int i=0; i<
LTR27_MEZZANINE_NUMBER; i++)                   for(int j=0; j<4; j++)
ltr27.Mezzanine[i].CalibrCoeff[j]= ltr27.ModuleInfo.Mezzanine[i].Calibration[j];
              // transmit data acquisition parameters to the module
              res=LTR27_SetConfig(&ltr27);
             if(res==LTR_OK) {
               // start ADC data acquisition
res=LTR27_ADCStart(&ltr27);
               if(res==LTR_OK) {
                   DWORD buf[NSAMPLES];
// capture ADC data
                  size=LTR27_Recv(&ltr27, buf, NULL, NSAMPLES, 1000);
                   if(size>0) {
                     double data[NSAMPLES];
                      // use calibration and convert in Volts
                     res=LTR27_ProcessData(&ltr27, buf, data, &size, 1, 1);
                     // display the measured voltage
if(res==LTR_OK) {                         int i=0;
while(i<size)
                             for(int j=0; j<2* LTR27_MEZZANINE_NUMBER; j++; i++)
printf(“channel%d %f %s\n”,
                                              j+1, data[i], ltr27.Mezzanine[j/2].Unit);
}
                  }
                  // stop ADC
res=LTR27_ADCStop(&ltr27);
               }
             }
          }
       }
       // close connection   LTR27_Close(&ltr27);
    }
 }
 // output error message
 if(res!=LTR_OK) printf(">> %s\n", LTR_GetErrorString(res)); }
```

4. Description of functions, structures and constants of *ltr27api.dll* library.

The present section has detailed description of constants, structures and interface functions included in ***ltr27api.dll*** *library*.

__Note:__ The recommended sequence of interface function calls see in *General approach to working with the interface functions of dll-library*.

4.1. Constants

Constant	Value	Description
Constants used in data received analysing.		
LTR27_DATA_CORRECTION	1	In data processing use the correction coefficients.
LTR27_DATA_FORMAT_CODE	0	Present resulting data in ADC codes.
LTR27_DATA_FORMAT_VALUE	2	Present resulting data in physical values.
Constants used in reading of module and submodules description		
FLAG_MODULE_DESCRIPTION	1	Read module description
FLAG_MEZZANINE1_DESCRIPTION	2	Read description of the mezzanine in slot 1
FLAG_MEZZANINE2_DESCRIPTION	4	Read description of the mezzanine in slot 2
FLAG_MEZZANINE3_DESCRIPTION	8	Read description of the mezzanine in slot 3
FLAG_MEZZANINE4_DESCRIPTION	16	Read description of the mezzanine in slot 4
FLAG_MEZZANINE5_DESCRIPTION	32	Read description of the mezzanine in slot 5
FLAG_MEZZANINE6_DESCRIPTION	64	Read description of the mezzanine in slot 6
FLAG_MEZZANINE7_DESCRIPTION	128	Read description of the mezzanine in slot 7
FLAG_MEZZANINE8_DESCRIPTION	256	Read description of the mezzanine in slot 8
FLAG_ALL_MEZZANINE_DESCRIPTION	510	Read description of all mezzanines
FLAG_ALL_DESCRIPTION	511	Read the description of the module and all mezzanines
Errors codes		
LTR_OK	0	Executed without errors.

LTR27_ERROR_SEND_ DATA	-3000	Error in data submission to the module
LTR27_ERROR_RECV_ DATA	-3001	Error in receiving data from the module
LTR27_ERROR_RESET _MODULE	-3002	Module no replay to RESET command.
The rest of constants		
LTR27_MEZZANINE_N UMBER	8	Number of slots to install submodules.

4.2. Structures

4.2.1. Structure TLTR27

Structure TLTR27 – is the basic structure containing all required information about the module configuration and channel state. This structure is used in all library functions of communication with the module.

The definition of the structure is in file `ltr27api.h` and is given below:

```
typedef struct {
//**** service information
  TLTR ltr;
  BYTE subchannel;
  //**** module configurations
BYTE   FrequencyDivisor;
struct TMezzanine {
CHAR Name[16];
CHAR Unit[16];
double ConvCoeff[2];
double CalibrCoeff[4];         }
Mezzanine[MEZZANINE_NUMBER];
} TLTR27;
```

Prior to start to manipulate with the module it is necessary to: create the instance of this structure and initialize the fields with values by default by calling function *LTR27_Init()*.

Field name	**Intended purpose of the field**
`Ltr`	The instance of structure *TLTR* servicing for providing communication channel with the module. The field is updated automatically when library interface functions are called and does not require any attention from the user PC side.

`Subchannel`	The field required to check continuity of received data stream. The field is updated automatically when library interface functions are called and does not require any attention from the user PC side.
`Frequency Divisor`	ADC sampling frequency divisor Range of values - from 0 to 255. $\text{ADC sampling frequency} = \frac{1000\ \text{Hz}}{\text{Sampling frequency divisor} + 1}$ The field serves to set sampling frequency in calling function *LTR27_SetConfig()* and to aligning of ADC code to 16-bit range when calling function *LTR27_ProcessData()*. The field is filled by the user or automatically when calling function *LTR27_GetConfig()*.
`Mezzanine[I]. Name`	Type mezzanine in module slot (i+1) The field is filled by the user or automatically when calling function *LTR27_GetConfig()*.
`Mezzanine[I]. Unit`	Physical values to measure which the mezzanine in module slot (i+1) is provided. The field is filled by the user or automatically when calling function *LTR27_GetConfig()*.
`Mezzanine[i]. ConvCoeff[0]`	Coefficient of scale for reconversion of ADC code of the mezzanine in slot (i+1) in physical values. Used when calling function *LTR27_ProcessData()*. The field is filled by the user or automatically when calling function *LTR27_GetConfig()*.
Field name	**Intended purpose of the field**
`Mezzanine[i]. ConvCoeff[0]`	Coefficient of zero offset for reconversion of ADC code of the mezzanine in slot (i+1) in physical values. Used when calling function *LTR27_ProcessData()*. The field is filled by the user or automatically when calling function *LTR27_GetConfig()*.
`Mezzanine[i]. CalibrCoeff[0]`	Coefficient of scale for correcting of ADC code of the mezzanine first channel in slot (i+1). Used when calling function *LTR27_ProcessData()*. Field is filled by the user.
`Mezzanine[i]. CalibrCoeff[1]`	Coefficient of zero offset for correcting of ADC code of the mezzanine first channel in slot (i+1). Used when calling function *LTR27_ProcessData()*. Field is filled by the user.
`Mezzanine[i]. CalibrCoeff[2]`	Coefficient of scale for correcting of ADC code of the mezzanine second channel in slot (i+1). Used when calling function *LTR27_ProcessData()*. Field is filled by the user.
`Mezzanine[i]. CalibrCoeff[3]`	Coefficient of zero offset for correcting of ADC code of the mezzanine second channel in slot (i+1). Used when calling function *LTR27_ProcessData()*. Field is filled by the user.

Values accepting with fields Mezzanine[i].Name, Mezzanine[i].Unit, Mezzanine[i]. ConvCoeff[0..1] after call function *LTR27_GetConfig()*.

Type of mezzanine	Name	Unit	ConvCoeff[0]	ConvCoeff[1]
H27_U01	“U01”	“B”	2.0/0x8000	-1.0
H27_U10	“U10”	“B”	20.0/0x8000	-10.0
H27_U20	“U20”	“B”	20.0/0x8000	0.0
H27_I5	“I5”	“mA”	5.0/0x8000	0.0
H27_I10	“I10”	“mA”	20.0/0x8000	-10.0
H27_I20	“I20”	“mA”	20.0/0x8000	0.0
H27_R100	“R100”	“Ohm”	100.0/0x8000	0.0
H27_R250	“R250”	“Ohm”	250.0/0x8000	0.0
H27_T	“T”	“mV”	100.0/0x8000	-25.0
Module is not identified	“EMPTY” or “UDEF”	“”	100.0/0x8000	0.0

ADC data correction is performed by the formula:

$$y = a*x + b$$

y – ADC corrected code

x – ADC uncorrected code

a – coefficient of scale

b – coefficient of zero offset

Note: Prior to correct ADC data it is necessary to perform ADC code aligning to 16-bit range that is performed automatically in function *LTR27_ProcessData()*.

ADC code conversion in physical values is performed by the formula:

$$y = a*x + b$$

y – value in physical values
x – ADC code (corrected or uncorrected) **a** – coefficient of scale
b – coefficient of zero offset

4.2.2. Structure of TINFO_LTR27

Structure TINDO_LTR27 – contains description of the module and installed submodules. Filling of the structure fields is performed when calling function *LTR27_GetModuleDescription()*.

Definition of structure TINFO_LTR27 and accompanied to it structures TDESCRIPTION_MODULE, TDESCRIPTION_CPU, TDESCRIPTION_MEZZANINE are presented in files `ltr27api.h` and `ltrapitypes.h` and are given below:

```
 typedef struct _DESCRIPTION_MODULE_
{
  BYTE   CompanyName[16];
  BYTE   DeviceName[16];
  BYTE   SerialNumber[16];
  BYTE   Revision;
  BYTE   Comment[COMMENT_LENGTH];
} DESCRIPTION_MODULE;
```

Field name	Intended purpose and permissible values of the field
CompanyName	Symbols string containing the name of module manufacturer.
DeviceName	Symbols string containing the name of the module.
SerialNumber	Symbols string containing the module serial number.
Revision	Symbol denoting the device revision.
Comment	Symbols string containing the additional information about the module.

```
typedef struct _DESCRIPTION_CPU_
{
  BYTE   Active;
BYTE   Name[16];
double ClockRate;
  DWORD  FirmwareVersion;
  BYTE   Comment[COMMENT_LENGTH];
} DESCRIPTION_CPU;
```

Field name	Intended purpose and permissible values of the field
Active	Structure fields validity flag. The values different from 0 inform that the rest fields of the structure have been filled correctly.
Name	Symbols string containing the name of control controller mounted on the module.
ClockRate	Control controller operation frequency.
FirmwareVersion	32 control controller x-bit word containing the software version downloaded in the controller.

Bit field	Purpose
31..24	High-order byte of software version.
23..16	Low-order byte of software version.
15..8	High-order byte of software build number.
7..0	Low-order byte of software build number

Comment	String of symbols containing the additional information about software

```
typedef struct _DESCRIPTION_MEZZANINE_
{
  BYTE   Active;
  BYTE   Name[16];
  BYTE   SerialNumber[16];
BYTE   Revision;   double Calibration[4];
  BYTE   Comment[COMMENT_LENGTH];
} DESCRIPTION_MEZZANINE;
```

Field name	Intended purpose and permissible values of the field
Active	Structure fields validity flag. The values different from 0 inform that the rest fields of the structure have been filled correctly.
Name	Symbols string containing the name of the mezanine.
SerialNumber	Symbols string containing the mezanine serial number.
Revision	Symbol denoting the mezzanine revision.
Calibration	Mezzanine calibration coefficients. **Coefficient index** | **Intended purpose** 0 | Scale coefficient of the mezzanine first channel 1 | Coefficient of zero offset of the mezzanine first channel 2 | Scale coefficient of the mezzanine second channel 3 | Coefficient of zero offset of the mezzanine second channel
Comment	Symbols string containing the additional information about the mezzanine.

```
typedef struct
{
  TDESCRIPTION_MODULE    Module;
  TDESCRIPTION_CPU       Cpu;
  TDESCRIPTION_MEZZANINE Mezzanine[MEZZANINE_NUMBER];
} TDESCRIPTION_LTR27;
```

Field name	Intended purpose and permissible values of the field
Module	Description of the module instance: name, serial number, revision.
Cpu	Description of control controller: name of the controller, software version.
Mezzanine	Description of mezzanines mounted in the module: name, serial number, revision, calibration coefficients.

4.3. Function

All interface functions of library ***ltr27api.dll***, except function *LTR27_GetErrorString()*, as the first parameter accept the pointer to instance of structure *TLTR27*.

Moreover all interface functions have the same type of returned value- INT. The returned value informs on the result of function execution. Negative values indicate the occurrence of *error*. Zero value corresponds to successful termination of the function excepting the functions *LTR27_Recv()*. Positive values are defined for function *LTR27_Recv()* only and define the number of received data.

4.3.1. Initialization functions of working with the module.

Functions of these sub-group carry out the actions on establishment and break of the connection with the module.

4.3.1.1. Initialization of the structure fields

Format: `INT LTR27_Init(TLTR27 *module)`

Purpose: Initialization of the structure fields by the default values.

This function shall be called once for each created instance of *TLTR27* structure before the rest functions of the library are called.

Transmitted parameters:

- module – pointer `to instance of` *TLTR27* `structure.`

Returned value:

- *Error code*.

4.3.1.2. Establishing the connection with the module

Format:
```
INT LTR27_Open(TLTR27 *module,
                 DWORD saddr, WORD sport, CHAR *csn,   WORD cc)
```

Purpose: Establish the connection with the module.

This function shall be called before starting communication with the module. Selection of module is performed in compliance with functions transferred to functions. If the transmitted instance of the structure indicates open connection to the module, then this connection will be automatically closed and an attempt will be made to reconnect.

Transmitted parameters:

- `module` – Pointer to instance of structure TLTR27.
- `saddr` – Network address of *LTR-server* - is a packaged in 32-bit unsigned integer (bigendian) ip-address of the computer on which the *LTR-server is started*.

  Example: If ip-address of the computer is “a.b.c.d”, the field shall have value (a<<24)|(b<<16)|(c<<8)|(d<<0).

  Note: If *LTR-server* is started on the same computer as the user program, the constant *SADDR_DEFAULT* can be used as network address.
- `sport` – Network port of LTR-server – is a packaged in 16-bit unsigned integer (bigendian) number of port to which *LTR-server* is configurated. By default *LTR-server* listens *SPORT_DEFAULT*.
- `csn` - Serial number of LTR-crate - the string with length of up to *SERIAL_NUMBER_SIZE* symbols.

  If the serial number length is less than *SERIAL_NUMBER_SIZE*, the line shall end by zero.

  Note: If spare line is specified as the serial number, an attempt to set the connection with the first detected LTR-crate will be performed.
- `cc` - Module logical number – 16-bit unsigned integer identifying the targeted module LTR27.

  Permissible values: *CC_MODULE1* – module located in crate first slot, *CC_MODULE2* – module located in crate second slot,.., *CC_MODULE16* – module located in crate 16th slot.

Returned value:

- *Error code*.

4.3.1.3. Loss of connection with the module

Format: `INT LTR27_Close(TLTR27 *module)`
Purpose: Break the connection with the module. This function shall be called after completion of data exchange with the module for correct closure of the connection and release of system resources reserved upon the connection opening.
Transmitted parameters: • module – pointer to instance of structure *TLTR27*.
Returned value: • *Error code*.

4.3.1.4. The status of connection with the module

Format: `INT LTR27_IsOpened(TLTR27 *module)`
Purpose: Define the status of connection with the module. The function enables defining of the module connection status:
Transmitted parameters: • module – pointer to instance of structure *TLTR27*.
Returned value: • *Error code*.

4.3.2. Functions to manipulate with module ADC

This subgroup functions perform operations required in manipulating with module ADC

4.3.2.1. Reading of module ADC settings

Format: `INT LTR27_GetConfig(TLTR27 *module)`
Purpose: Reading of current module ADC settings The function allows to read the current module ADC settings.
Transmitted parameters: • module – pointer to instance of structure *TLTR27*. In the event of the function successful completing the information on the ADC frequency divisor, the type of mezzanines installed, units of physical quantities and conversion coefficients from the ADC codes to physical quantities for each mezzanine will be updated in the transmitted module structure. Thus, the following fields of the structure will be updated: - `module->FrequencyDivisor` - `module->Mezzanine[0..7].Name` - `module->Mezzanine[0..7].Unit` - `module->Mezzanine[0..7].ConvCoeff[0..1]`
Returned value: • *Error code*.

4.3.2.2. Writing of module ADC settings

Format: `INT LTR27_SetConfig(TLTR27 *module)`
Purpose: Writing of module ADC settings. This function allows to set module ADC configurations in accordance with which the data acquisition will be performed.
Transmitted parameters: • module – pointer to instance of structure *TLTR27*. In the event of the function successful performing the information on the ADC sampling frequency divisor will be transmitted to the module. Thus, the values of the following structure fields will be transmitted: `- module->FrequencyDivisor`
Returned value: • *Error code*.

4.3.2.3. ADC start-up

Format: `INT LTR27_ADCStart(TLTR27 *module)`
Purpose: Module ADC start-up. This function allows to switch the module from *waiting state* into *data acquisition state*.
Transmitted parameters: • module – pointer to instance of structure *TLTR27*.
Returned value: • *Error code*.

4.3.2.4. ADC stopping operations

Format: `INT LTR27_ADCStop(TLTR27 *module)`
Purpose: To stop the module ADC. This function allows to switch the module from *data acquisition state* into *waiting state*.
Transmitted parameters: • module – pointer to instance of structure *TLTR27*.
Returned value: • *Error code*.

4.3.2.5. Receiving of data from the module

Format: `INT LTR27_Recv(TLTR27 *module, DWORD *des_data, DWORD *tmark, DWORD size, DWORD timeout)`

Purpose: Receiving of data from the module.

This function receives and monitors the data parity. It is used in combination with function *LTR27_ProcessData()* performing post-processing of received data (aligning to range of values of 16-bit ADC, using the calibrations, conversing of ADC code in physical quantities)

Transmitted parameters:

- module – pointer to instance of structure *TLTR27*.
- des_data – pointer to array of DWORD[size] type where the received data from the module will be inserted. Each element of the output array contains uncalibrated data of one of the ADC channels and several service information fields. Regardless of the ADC operation parameters the order of the data is always fixed:

  - data of 1-st channel of mezzanine located in 1-st module slot;
  - data of 2-nd channel of mezzanine located in 1-st module slot;
  - and etc.
  - data of 2-nd channel of mezzanine located in 8-th module slot;

  So, each 16th element of the array contains samples of the same ADC channel. To convert an array of received data to an ADC sample array or an array of physical quantities and to use the calibrations, the function *LTR27_ProcessData ()* is used.

- tmark – pointer to array of DWORD[size]`type` where the *marks of time* corresponding to received data will be inserted. Thus, the element of array tmark[i] containing the mark of time correposponds to each element of array data[i]. If there is no need in time tags, NULL can be sent as the parameter.
- size – number of samples which should be received from the module.
- timeout – interval of time in milliseconds during which the receiving of requested samples quantity is expected. If data from module are not received within the specified period, the exit a function will occur.

Returned value:

- Values less than zero should be considered as *error codes*. The values greater than zero or equal to zero should be considered as quantity of words actually received from the module within the allowed time.

4.3.2.6. Module data processing

Format: `INT LTR27_ProcessData(TLTR27 *module, DWORD *src_data, double *dst_data, DWORD *size, BOOL calibr, BOOL value)`

Purpose: Received data processing.

The function performs the received data processing using *LTR27_Recv()*:

- aligning of ADC code to 16-bit range
- ADC data calibration
- conversion of ADC code in physical quantities

Transmitted parameters:

- module – pointer to instance of structure *TLTR27*.
- src_data – pointer to array of DWORD[size] type containing the data received using the function *LTR27_Recv()* and are to be processed
- dst_data – pointer to array of double[size] type where the output data will be transmitted in. The sequencing of data corresponds to the sequencing of data in input buffer src_data.
- size – at output, defines the number of samples contained in array src_data,

  at output, defines the number of data processed and transmitted in array dst_data
- calibr – the flag selecting the calibration coefficient usage mode.

Value	Description
0	The calibration will not be applied to the output data.
1	The calibration will be applied to the output data. The corresponding fields of structure *TLTR27* will be used as calibration coefficients.

- value – flags selecting output data format.

Value	Description
0	Output data will be presented as ADC samples aligned to 16-bit range. Thus, in output array the data in format with floating point accepting the value within the range from –32768.0 to 32768.0 will be inserted.
1	The output data will be conversed in physical quantities. The corresponding fields of structure *TLTR27* will be used as conversion coefficients.

Returned value:

- *Error code*.

4.3.3. Information type functions

This subgroup functions allow to obtain information about the module.

4.3.3.1. Reading of module and submodules description.

Format: `INT LTR27_GetDescription(TLTR27 *module, WORD flags)`

Purpose: Reading of module and submodules description

The function allows to obtain the description of the module and/or installed mezzanines.

Fields TLTR27->ModuleInfo are filled

Transmitted parameters: • module – pointer to instance of *TLTR27* structure. • flags – flags indicating the fields of structure ModuleInfo to be filled:
Returned value: • *Error code*

4.3.4. Auxiliary functions

4.3.4.1. Interface with the module testing

Format: `INT LTR27_Echo(TLTR27 *module)`
Purpose: Interface with the module testing The test function allows to check functionality of communication channel with the module. In performing of this function the package of "empty" commands is transmitted to the module which it shall response on. In the event of correct response of the module the interface with the module is proper.
Transmitted parameters: • module – pointer to instance of structure *TLTR27*.
Returned value: • pointer to the constant string containing the message on error.

4.3.4.2. Text message on the error

Format: `LPCSTR LTR27_GetErrorString(INT error)`
Purpose: Receive message on error in text. Function returns the string containing message on error corresponding to error code transferred to function.
Transmitted parameters: • error – error code.
Returned value: • pointer to the constant string containing the message on error.

5. Annex

5.1. Module interconnect protocol. Command and data formats.

This section describes the information about low-level protocol on communication with the module **LTR27** and about the format of data involving in this communication. All features of the protocol and data format are taken into account when writing the ***ltr27api.dll*** library and do not require the understanding of the programmer using the functions of this library. The section is intended for general acquaintance, as well as for the users who intend to implement the module interconnect protocol in their software.

5.1.1. Module interconnect protocol.

On energizing and exiting the reset condition (for details see *LTR Crate System. User Manual*) the module is switched in *waiting state* in which it can receive and process host-computer commands. On ADC start command arrival the module is switched in *data acquisition mode*. Being in this state, the module performs parallel digitization of all 16 analog channels (in accordance with the specified sampling frequency divisor) and output the accumulated values to the host computer. Any command coming at this time stops the ADC data acquisition, stops transmission of the accumulated values to the host computer, and switches the module to *the waiting mode*(command loss does not occur during such switching and immediately starts to be processed after switching).

All commands and data received and transmitted by the module contain a parity bit which serves as a sign of reliability of the received information.

*Waiting mode***.** All commands are the same size equal to one 32-bit word. Bit fields of the command contain the code of operation and data required for its performing. The module shall response on each received command (regardless of its validity) by sending one 32-bit word as follows:

- data in the event when the operation requires some kind of reading
- positive confirmation in the event when the operation is successfully completed
- negative confirmation in the event when: the command implementation failure, parity error in receiving the command or receiving the unsupported command

*Data acquisition mode***.** ADC data are transmitted to host-computer in frames by 16 of 32-bit words containing the samples of all 16 channels regardless of availability on the mezzanines board. Data of 1-st channel of mezzanine located in 1-st module slot are transmitted first; - data of 2-nd channel of mezzanine located in 8-th module slot are transmitted last.

To speed up exchange with host-computer the module *in waiting mode* is capable to buffer up to 128 commands. The processing of buffered commands is performed in their enqueuing order.

So, the host-computer is capable to sent commands to module in small blocks and to wait receipt of responses for the whole block.

5.1.2. Formats of commands and data **Format**

of data words:

`DDDDDDDD DDDDDDDD 0000MMMM 11P0SSSS`

Bit is left-to-right in order of descending of bit number.

`S` – subchannel number

`P` – parity bit

`M` – module number in crate

`D` – subchannel data

Note:

Subchannel data contain the code with varied number of significant bits depending on selected sampling frequency. However, the calibration coefficient contained in eeprom of mezzanines have been calculated for ADC having 16effective bits of data. So, prior to use calibration it is necessary to align code ADC to 16- bit range using the following formula:

$$\text{16 - bit ADC code} = \frac{32767 * (\text{ADC code received from the module})}{250 * (\text{Sampling frequency divisor} + 1)}$$

<u>Format of command words and positive confirmation words</u>:

`DDDDDDDD DDDDDDDD 1000MMMM 11PCCCCC`

Bit is left-to-right in order of descending of bit number.

`C` – command code

`P` – parity bit

`M` – module number in crate

`D` – data depending on command code

<u>Negative confirmation</u>:

`11111111 11111111 1000MMMM 11P01000`

Bit is left-to-right in order of descending of bit number.

`P` – parity bit

`M` – module number in crate

<u>Parity</u>:

Prior to calculate the parity bit the mask is set for command word or data word - `11111111 11111111 00000000 11011111.` The parity bit is calculated as the sum of all bits on module 2:

```
P=COMMAND_DATA_WORD&0xFFFF00DF;
P^=(P>>16);
P^=(P>>8);
P^=(P>>4);
P^=(P>>2);
P^=(P>>1);
P&=1;
```

Data(to the module, from the module)	Command code	Description
→ `XXXXXXXX XXXXXXXX` ← `XXXXXXXX XXXXXXXX`	`00000`	**Echo** Empty command allows to check functionality of interface and module control controller. `X` – has no value and is not changed with the module during the command performing.
→ `XXXXXXXT XXXXXXXX` ← `XXXXXXXT XXXXXXXX`	`00001`	**SetFlags** The command allows to set the module operation control flags. `T` – test mode flag. If the flag is set then in switching in *data acquisition mode* instead of ADC actual data will be transmitted value of internal incremented counter. `X` – has no value and is not changed with the module during the command performing.

→ XXXXXXXX XXXXXXXX ← XXXXXXXX XXXXXXXX	00010	**StopADC** The command switches the module in *waiting mode*. **X** – has no value and is not changed with the module during the command performing.
→ XXXXXXXX XXXXXXXX ← XXXXXXXX XXXXXXXX	00011	**StartADC** The command switches the module in *data acquisition mode*. **X** – has no value and is not changed with the module during the command performing.
→ XXXXXSSS XXXXXXXF ← XXXXXSSS XXXXXXXF	00111	**Mezzonine EEPROM Write Enable/Disable** The command of enabling/disabling of writing in mezzanine eeprom. **S** – selection of mezzanine. 0 – mezzanine in 1-st module slot. 1 – mezzanine in 2-nd module slot. …. 7 – – mezzanine in 8-th module slot. **F** – flag enabling writing in mezzanine eeprom. 0 – disable writing 1 – enable writing **X** – has no value and is not changed with the module during the command performing.
→ AAAAAAAA XXXXXXXX ← AAAAAAAA DDDDDDDD	010SS	**Read AVR memory** Command to read byte from controller memory. **S** – selection of block from which the reading will be performed. 0 –internal RAM AVR storing the local variables. Block size - 256 bytes. 1 – reserve. 2 – reserve. 3 – internal block of AVR flash-memory storing the module descriptor. Block size - 256 bytes. **A** – byte address in block **D** – referenced byte **X** – has no value and is not changed with the module during the command performing.

→ AAAAAAAA DDDDDDDD ← AAAAAAAA DDDDDDDD	011SS	**Write AVR memory** Command to write byte in controller memory. **S** – selection of block in which the writing will be performed. 0 – block of internal RAM of AVR memory storing the local variables. Block size - 256 bytes. 1 – reserve. 2 – reserve. 3 – reserve. **A** – byte address in block **D** – referenced byte **X** – has no value and is not changed with the module during the command performing.
→ AAAAAAAA XXXXXXXX ← AAAAAAAA DDDDDDDD	10SSS	**Read Mezzanine EEPROM** Command to read byte from mezzanine eeprom. **S** – selection of mezzanine. 0 – mezzanine in 1-st module slot. 1 – mezzanine in 2-nd module slot. …. 7 – – mezzanine in 8-th module slot. **A** – address of byte for reading **D** – referenced byte **X** – has no value and is not changed with the module during the command performing.
→ AAAAAAAA DDDDDDDD ← AAAAAAAA DDDDDDDD	11SSS	**Write Mezzanine EEPROM** Command to write byte in mezzanine eeprom. **S** – selection of mezzanine. 0 – mezzanine in 1-st module slot. 1 – mezzanine in 2-nd module slot. …. 7 – – mezzanine in 8-th module slot. **A** – address of byte for writing **D** – referenced byte **X** – has no value and is not changed with the module during the command performing.

<u>Module control block address space</u>:

Block number controller memory	Description
0	Internal controller RAM storing the local variables.

Address	Size	Access	Description
0	1	reading/ writing	*ADC sampling frequency divisor* Range values - 0...255 Sampling frequency = 1000 Hz/(Divisor +1)
1	255	reading/ writing	*Reserve*

Block number controller memory	Description
1	Reserve
2	Reserve
3	Controller internal ROM storing the description of the module

Address	Size	Access	Description
0	128	reading	*Reserve*
128	16	reading	*Name of manufacturer* (L-CARD)
144	16	reading	*Name of the device* (LTR27)
160	16	reading	*Serial number*
176	16	reading	*Type of control controller* (ATMega8515)
192	4	reading	*Controller operation bit-timing frequency*
196	4	reading	*Software version*
200	1	reading	*Module revision*
201	53	reading	*Comments*
254	2	reading	*Block checksum*

<u>Mezzanines address space</u>:

See description of modules H-27.