Multichannel data-acquisition systems

LTR212 Strain-gauge module

Programmer manual

Revision 1.0.4
November 2013

L-CARD

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

DAQ SYSTEMS DESIGN, MANUFACTURING & DISTRIBUTION

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	23.01.2006	The first revision available for user
1.0.1	07.03.2006	Module description structure is revised.
1.0.2	21.11.2006	Changes were made in the format of the module description structure and field names in order to unify the software interface of all modules
1.0.3	01.03.2010	New calibration mode is added. User-specific PROM calibration storage and record functions are added.
1.0.4	01.09.2013	Information on new module modifications is added. Module description structure is revised (field `Type` of `TINFO_LTR212` structure is added). Logic channel format is revised (field *Bridge connection type is added*). New function *LTR212_CreateLChannel2() is introduced*.

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

LLC "L-Card" reserves the right to update the documentation without prior notification of users.

Contents

1 Module LTR-212 essentials.

Strain-gauge module *LTR-212* has three basic operating modes**: 4-link with mean accuracy**, **4-link with high accuracy** and **8-link with high accuracy.** Whereby reference voltage value may be specified (2.5 V or 5 V) and selecting its type: *constant* or *alternating*. When operating on each of mode the digital filtration of collected data is performed. Module allows for program zero and range calibration compensating signal displacement and amplification errors. Calibrating registers ADC AD7730 and calibrating DAC integrated in the microchip are involved in zero calibration process. Calibrating registers ADC are used only for range calibration. Data acquisition and calibration mode control, programming of ADC AD7730, reading of data obtained from ADC registers and buffering in FIFO of crate-controller shall be performed by module-mounted Digital Signal Processor ADSP-2185M. The processor has an individual low level software hereinafter referred to as *BIOS*.

1.1 What's the news?

Generally this paragraph shall contain only essentials of module revision. Refer to *Chapter 6* of "LTR Crate System for detailed information. User Manual"

1.1.1 2013

LLC "L-Card" (based on generic user stories over the course of modules *LC-212* and *LTR-212* release history) in 2013 has released new module modifications: *LTR-212M-1*, *LTR-212M-2* and *LTR-212M-3*.
In the description, the following abbreviated designations are introduced:

- *LTR-212* – previous modification released up to 2013;
- *LTR-212M* – any new modifications *LTR212M-1*, *LTR212M-2* and *LTR212M-3*;
- *LTR-212(M)* – any modification: *LTR212*, *LTR212M-1*, *LTR212M-2* and *LTR212M-3*.

2 General information of Library LR212API.

From a software point of view any interaction with the module shall be performed via library of user-specific functions ***Ltr212api***.

The sequence of application of these functions is similar to using equivalent functions in the software of other modules of the LTR system. In general, this sequence can be described as follows:

- Initialization of the communication interface with the module, setting the default settings;
- BIOS *downloading* (low level Software) into internal memory of Digital Signal Processor ADSP-2185M;
- Parameter setting (data acquisition link coding, determination of data acquisition or calibration modes);
- Acquisition start;
- Acquisition and data processing (frequently this process is cyclic);
- Acquisition stop;
- Closing of the communication interface with the module.

The library of user interface functions contains the following general components: **a module description structure** and **a set of functions for module communication and operation**.

2.1 Module description structure.

The module description structure (type **TLTR212)** is designed for initialization, storage and change of information about module configuration within the whipped out code. The structure members allow for setting of data acquisition mode, identification of calibration factors application, generation of information about involved data acquisition links and related voltage range of input signal. The structure also incorporates substructure that contains information about program filters. Prior to call of (**LTR212_SetADC()**) module configuration function, it is necessary to fill the fields of the module description structure. Otherwise, the module would generate wrong data.

2.2 User-defined library functions.

Functions of the ***Ltr212api*** library are intended for configuration, programming, data acquisition and diagnostics of the module.

2.2.1 Classification of library functions.

Functions integrated in the user-defined library *LTR-212(M)* of the module may be divided in the following groups:

- Initialization and opening functions;
- Configuration functions;
- Data acquisition functions;
- Calibration function;
- Closing function;
- Auxiliary functions.

The **initialization and opening functions** include functions **LTR212_Init()** and **LTR212_Open().** They should be called first. They are intended to fill the module description structure members with values by default, open interface communication link with the module, download *BIOS* into DSP module and perform the required checks. Without calling this functions, the further work with the module is impossible.

Configuration functions – LTR212_CreateLChannel(), LTR212_CreateLChannel2() and **LTR212_SetADC().** Function **LTR212_CreateLChannel()** allows for creation of logic channel table, i.e. bonding of physical channels of the module and corresponding voltage ranges of input signal, and determination of physical channels involved in data acquisition or otherwise. Function

LTR212_CreateLChannel2() which may set type of bridge connection pattern for sensors (for *LTR-212M-1*) may be used as well. Function **LTR212_SetADC()** transmits all configuration parameters recorded in module description structure members, in module DSP which in turn performs programming of on-board ADC AD7730. After the indicated functions are completed, the device is ready for data acquisition.

Data acquisition functions: LTR212_Start(), LTR212_Stop(), LTR212_ProcessData(). are intended for acquisition start and stop, for validation of data received fro the module, and for recalculation from ADC codes to voltage values (V).

Calibration function: LTR212_Calibrate(). Used for module calibration. The function arguments determine calibration mode and channels for which it should be done.

Auxiliary functions: LTR212_GetErrorString() returns error string corresponding to the error code; **LTR212_CalcFS()** returns acquisition frequency; **LTR212_TestEEPROM()** verifies data integrity in the module PROM.

Closing function: LTR212_Close(). It is called upon the module shutdown to close the interface channel. For clean termination of the module it is necessary to call this function.

2.2.2 A typical sequence of program writing.

When programming module one should comply with the diagram specified below.

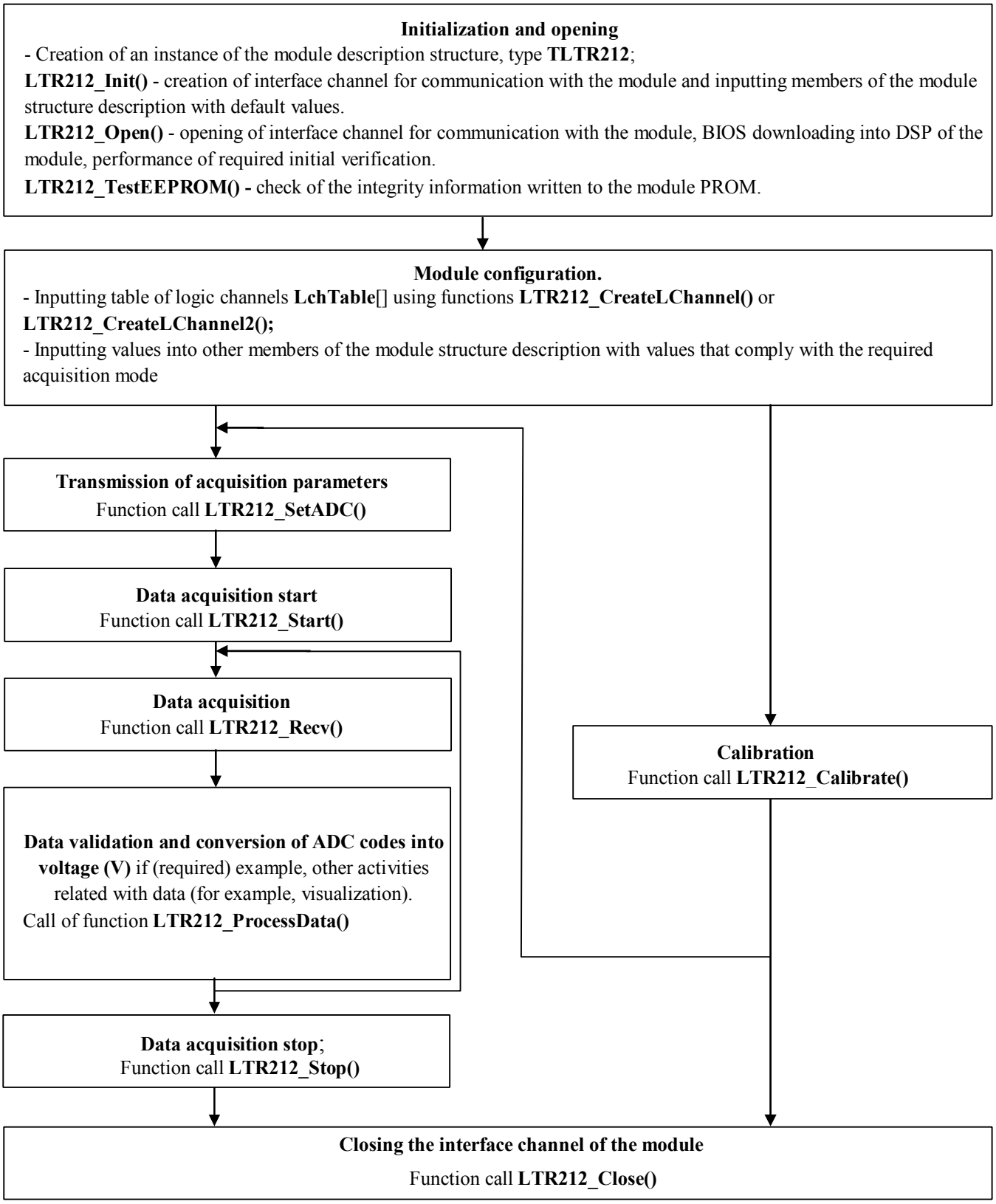


Fig. 1

3 Detailed description of Library LTR212API.

This paragraph shall cover user-specified library ***Ltr212api*** components in detail.

To call the library ***Ltr212api*** interface functions using your application follow the procedure mentioned below:

- to create the project in any of the development environment;
- insert a file ltr212api.dll into 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** :
  - link files **LTR\LIB\BORLAND\ltr212api.lib** and **LTR\INCLUDE\ltr212api.h with the project**; **Microsoft Visual C++ :**
  - link files **LTR\LIB\MSVC\ltr212api.lib** and **LTR\INCLUDE\ltr212api.h with the project**;

  Other development environments:
  - it is necessary to refer to the corresponding documentation of the development tool.
- create and link file with the original text of the future program with the project;
- after that you can write your program by calling the corresponding interface functions of the dll-library.

3.1 Module description structure.

Fields of the structure contain information on involved module channels, acquisition or calibration modes, use of calibration factors, program filters connection and other data. Structure definition is shown below:

```
typedef struct
{
 int size;                // Structure size
 TLTR Channel;            // Interface channel for communication with the module
 INT AcqMode;             // Acquisition mode
 INT UseClb;              // Calibration factor application flag
 INT UseFabricClb;        // Factory calibration factor application flag
 INT LChQnt;              // Quantity of logic channels
 INT LChTbl[8];           // Logical Channel Table
 INT REF;                 // Reference voltage selection flag (0->2.5V, 1->5V)
 INT AC;                  // Alternating reference voltage selection flag
 double Fs;               // Reporting frequency

 struct
{
 INT IIR;                   // Infinite-impulse response filter application flag
 INT FIR;                   // Feedback filter application flag
 BYTE Decimation;           // Feedback filter decimation gain
 BYTE TAP;                  // Tap procedure (quantity) of feedback filter
 CHAR IIR_Name[512+1];      // Full path of infinite-impulse response filter
 CHAR FIR_Name[512+1];      // Full path of feedback filter
 } filter;                  // Structure that stores program filter data

 TINFO_LTR212    ModuleInfo // Module identification information
 WORD CRC_PM;                    // for internal use
```

```
 WORD CRC_Flash_Eval;              // for internal use
 WORD CRC_Flash_Read;              // for internal use
} TLTR212, *PTLTR212;              // module description structure
```

Here's the description of the structure members.

Table 1

Field		Type	Description
size		**INT**	memory capacity, allocated for the structure
Channel		**TLTR**	the structure, which is a description of the interface channel for communication with the crate controller;
AcqMode		**INT**	Determines acquisition mode. "0" - 4-channel, mean accuracy, "1" - 4-channel high accuracy, "2" - 8-channel high accuracy.
UseClb		**INT**	Determines if calibration factors stored in calibration area of the module PROM should be used in acquisition. "0" - calibration factors are not used. "1" - calibration factors are used.
UseFabricClb		**INT**	If this flag is set then upon invocation of configuration **LTR212_SetADC()** download factory calibration factors corresponding to each involved channel range shall be automatically downloaded into ADC registers AD7730
LChQnt		**INT**	The number of used logic channels. Refer to the Chapter 3.2 for clarification of the notion "*logic channel*.
LChTbl[8]		**INT**	Logical Channel Table Refer to the Chapter 3.2 for clarification of the notion "*Logical Channel Table*".
filter			Structure of program filter parameters
fields	IIR	**INT**	Determines if the infinite-impulse response filter is involved "0" – filter OFF, "1" – filter on
	FIR	**INT**	Determines if the feedback filter is involved "0" – filter OFF, "1" – filter on
	Decimation	**BYTE**	Decimation gain used with the applied feedback filter. This factor is *hard-linked* with the specific filter.
	TAP	**BYTE**	Tap procedure (quantity) of applied program filter
	IIR_Name[51 2+1]	**CHAR**	The string which is a full name of the file with factors of infinite-impulse response filter

	FIR_Name[5 12+1]	**CHAR**	The string which is a full name of the file with factors of the feedback filter
ModuleInfo		**TINFO_LTR212**	Module identification information. This type is the structure which description is provided below in this chapter.
REF		**INT**	Determines the value of reference voltage: "0" - 2.5 V, "1" - 5.0 V.
AC		**INT**	Alternating reference voltage application flag. "1" – alternating reference voltage is used, "0" – dc reference voltage is used
Fs		**INT**	Reporting frequency (Hz). This field shall be automatically filled after invocation of the function **LTR212_SetADC()**

To set the module operating mode the user must manually determine the following fields of the structure: **AcqMode, UseClb, UseFabricClb, LChQnt, LChTbl, filter.FIR, filter.IIR, filter.IIR_Name, filter.FIR_Name, REF, AC**. Other fields shall be filled automatically upon invocation of different functions. Examples of configuration setting are set out at the end of this manual.

3.1.1 Module identification information

The structure contains all necessary operating information about applied module after execution of the function **LTR212_Open()**:

```
typedef struct
{
 CHAR Name[15];                  // module name string
 BYTE Type;                      // module modification type: LTR-212 (LTR-212M-3),
                                 // LTR-212M-1 or LTR-212M-2
 CHAR Serial[24];                // string with serial number of the module
 CHAR BiosVersion[8];  // string with the BIOS version
 CHAR BiosDate[16];    // string with date of BIOS creation
} TINFO_LTR212, *PTINFO_LTR212;
```

The field `Type` contains information of the current type of used module modification:

Table 2

Module modification	Type	Mnemonics
LTR212 or *LTR212M-3*	0	LTR212_OLD
LTR212M-1	1	LTR212_M_1
LTR212M-2	2	LTR212_M_2

3.1.2 User-specific calibration data storage structure

```
typedef struct
{
 DWORD Offset[MAX_212_CH];       // Offset factors for 8 channels
```

```
 DWORD Scale[MAX_212_CH];          // Scale factors for 8 channels
 BYTE   DAC_Value[MAX_212_CH];    // Codec gaging DAC value    }
TLTR212_Usr_Clb;
```

Calibration information in this structure is stored in internal codec format AD7730.
Direct modification is not advisable without insight into coded operation features.

3.2 Logical Channel Table

One of the module structure description fields – **Logical Channel Table** – is an array with information of physical channels of the module, corresponding acquisition ranges and bridge circuits, and defines channel polling sequence.

Logic channel is a 32x bit word which format is set out below:

Table 3

Bits	31÷28	27÷20	19÷16	15÷4	3÷0
Intended purpose	Type of bridge connection	*Reserved*	Number of physical channel	*Reserved*	Input voltage range

Type of bridge connection sets bridge circuit for connection of sensors to the module.

Table 4

Type of bridge connection	**Mnemonics**	**Value**
0	LTR212_FULL_OR_HALF_BRIDGE	Full- and half-bridge circuit. For modules *LTR-212(M).*
1	LTR212_QUARTER_BRIDGE_ WITH_200_Ohm	**Quarter-bridge** circuit with an internal resistor 200 Ohm. Only for modules *LTR-212M-1*.
2	LTR212_QUARTER_BRIDGE_ WITH_350_Ohm	**Quarter-bridge** circuit with an internal resistor 350 Ohm. Only for modules *LTR-212M-1*.
3	LTR212_QUARTER_BRIDGE_ WITH_CUSTOM_Ohm	**Quarter-bridge** circuit with an external resistor 180÷1000 Ohm on submodule LTR212H. Only for modules *LTR-212M-1*.
4	LTR212_UNBALANCED_QUARTER_ BRIDGE_WITH_200_Ohm	Similar to connection 1 only upon rated disbalance of **quarter-bridges**. Only for modules *LTR-212M-1*.

5	LTR212_UNBALANCED_QUARTER_BRIDGE_WITH_350_Ohm	Similar to connection 2 only upon rated disbalance of **quarter-bridges**. Only for modules *LTR-212M-1*.
6	LTR212_UNBALANCED_QUARTER_BRIDGE_WITH_CUSTOM_Ohm	Similar to connection 3 only upon rated disbalance of **quarter-bridges**. Only for modules *LTR-212M-1*.

Crucially that the quarter-bridge circuit for sensor connection may be used only on module *LTR-212M-1* and only in initial 4[x] **physical channels**. Whereby **all logic channels** where quarter-bridges are used shall involve similar type of bridge connections. In this regard connection types 1 and 4 may be considered compatible, as well as types 2 and 5 or 3 and 6.

A physical channel is a hardware data input channel used to connect an electrical signal to the *LTR-212(M)* module. The module has got 8 physical channels. Complete information on these channels is indicated in book "LTR Crate System. User manual".

Input voltage range codes are specified below:

Table 5

Input range code	Range
0	-10 mV/+10mV
1	-20 mV/+20mV
2	-40 mV/+40mV
3	-80 mV/+80mV
4	0mV/+10mV
5	0 mV/+20mV
6	0 mV/+40mV
7	0 mV/+80mV

A logic channel table is an array, each element of which is a logic channel. All components of this table define those physical channels which shall transmit data after start and corresponding input signal voltage range. Filling of this array (logic channel table) is performed by the user. To create the logic channel the function **LTR212_CreateLChannel()** shall be called indicating in parameters number of the physical channel connected with this logic channel and the code of related range. The function returns the generated value of the logic channel, which should be written to the table manually. It is not necessary to arrange **sequentially**, in ascending order of physical channel numbers, the elements corresponding to different physical channels of the module in the logic channels table. Elements with more significant indices shall correspond to physical channels with more significant numbers. If this stipulation is not met the further function **LRT212_SetADC()**call shall fail. Upon operation of the module data shall be acquired only from channels included in the table concerned. Data received from other channels shall be ignored. The **LChQnt** module description structure field must contain the number of logic channels involved (it is also the number of logic channel table elements). In four-channel operating mode (**AcqMode**=0 and

AcqMode=1) maximum number of logic channels is equal to 4. In eight-channel mode (**AcqMode**=2) is equal to 8.

Upon operation of the module data acquired from channels shall be obtained in a sequence determined by the logic channel table.

An example of correct configuration of the logic channel table in 8-channel mode:

LChQnt=6;

LChTbl[]:

Element index (logic channel No.)	Physical channel	Range	Logic channel value (Hex)
0	1	0 (-10 mV/+10mV)	0x00010000
1	2	0 (-10 mV/+10mV)	0x00020000
2	4	3 (-80 mV/+80mV)	0x00040003
3	5	6 (0 mV…+40mV)	0x00050006
4	7	4 (0 mV…+10mV)	0x00070004
5	8	3 (-80 mV/+80mV)	0x00080003

i.e. the array **LChTbl** shall be as follows:

0x00010000
0x00020000
0x00040003
0x00050006
0x00070004
0x00080003

It is apparent that physical channels 3 and 6 are not included in the table. Consequently, no data from these channels shall be displayed in the array received from the module during acquisition. These channels may be considered shut down. Within the given context physical channel polling and data location in the input array upon acquisition are specified below; figures indicate the number of physical channel whereof each element of the input array (data package) is received.

1 – 2 – 4 – 5 – 7 – 8 – 1 –2 – 4 – 5 – 7 – 8 – 1 – 2 – 4 – 5 …….

Detailed information of input array and data location in it shall be specified below in Chapter 4.1.

3.3 Description of the Ltr212api library function.

Format: INT LTR212_Init(PTLTR212 hnd)

Parameter: hnd – pointer to the module description structure (type **PTLTR212)**

Description: Initializes the interface channel for communication with the module and fills in the fields of the module description structure with default values. Below are the values by which this function initiates the fields of the specified structure:

```
hnd->size=sizeof(TLTR212);              // Structure size
hnd->AcqMode=1;                         // 4-channel mode of high accuracy acquisition mode
hnd->UseClb=0;                          // Editorial factors are not used
hnd->UseFabricClb=0;                    // Factory calibration factor direct use mode
                                        // is not operated
hnd->LChQnt=4;                          // Quantity of logic channels is equal to 4
                                        // Initialization of the logic channel table Four channels are
set by default
                                        // with input signal voltage range –80 mV/+80 mV
hnd->LChTbl[0]=LTR212_CreateLChannel(1, 3);
hnd->LChTbl[1]=LTR212_CreateLChannel(2, 3);
hnd->LChTbl[2]=LTR212_CreateLChannel(3, 3);
hnd->LChTbl[3]=LTR212_CreateLChannel(4, 3);
hnd->filter.IIR=0;                      // Program filters OFF
hnd->filter.FIR=0;
hnd->filter.Decimation=0;               // switch off filter decimation
hnd->filter.TAP=0;                      // Filter order - 0
hnd->Fs=150.15;                         // Acquisition frequency for the mode
// Prior to invocation of function LTR212_Open() the following fields shall be blank
strcpy((CHAR *)hnd->ModuleInfo.Serial, "\0");          // Serial number by default
strcpy((CHAR *)hnd->ModuleInfo.BiosVersion, "\0");     // BIOS version by default
strcpy((CHAR *)hnd->ModuleInfo.BiosDate, "\0");        // BIOS date by default
strcpy((CHAR *)hnd->ModuleInfo.Name, "\0");            // Module name by default
```

Returned value: error code, type **int.** If "0" – bug-free function

Format: INT LTR212_IsOpened(PTLTR212 hnd)

Parameter: **hnd** – pointer to the module description structure, type TLTR212

Description: The function enables monitoring of the module connection status: if the returned result varies from 0, there is no connection.

Returned value: If "0" – interface channel for communication with module is created and opened. If the value is nonzero, the channel is not created.

Format: INT LTR212_Open(PTLTR212 hnd, DWORD net_addr, WORD net_port, CHAR *crate_sn, INT slot_num, CHAR *biosname)

Parameters:

- **hnd** – pointer to the module description structure (type **PTLTR212)**
- **net_addr** – network server address A.B.C.D in HEX format: 0xABCD. For example, **net_addr** for address 127.0.0.1 shall be as follows: 0x7F000001. One should bear in mind that all address

components shall have a value not exceeding 255. - **net_port** – network server port - **crate_sn** – serial number of crate.

- **slot_num** – number of slot where the module is located (numeration starts from ***figure one***!)
- ***biosname** – full path to file containing a program for DSP (*BIOS*). BIOS *file* supplied by the company is named **ltr212.bio**. Full path to this file in the user's file system shall be indicated in this function. One should bare in mind that according to the language syntax "S", Symbol '\' shall be replaced to "\\" in strings.

Description: The function opens the interface channel for communication with the module, downloads *BIOS* program into the module DSP, performs necessary checks, and reads its identification record from the PROM of the module. After function operation the following information shall be indicated in corresponding fields of the module description structure: BIOS *number*, date of BIOS issue, module name, type and serial number.

Returned value: error code, type **int.** If "0" – bug-free function. If the return value is –10, then this is a warning that the interface channel is already open. Nevertheless, the function succeeded and you can continue work. However the further module operation can be incorrect. It is recommended to understand reasons of the warning and close previously opened channel.

Format: INT LTR212_CreateLChannel(INT PhysChannel, INT Scale)

Parameters:

- **PhysChannel** – number of physical channel corresponding to created logic channel. Numeration starts from figure one;
- **Scale** – input signal voltage range for this physical channel.

Description: The function creates a logic channel, i.e. generates a 32-bit word that should be written to the corresponding cell in the logic channel table. Refer to Chapter 3.2. of this Manual to get more information on the word format.

Returned value: Generated value of the logic channel, type **int.**

Format: INT LTR212_CreateLChannel2(INT PhysChannel, INT Scale, INT BridgeType)

Parameters:

- **PhysChannel** – number of physical channel corresponding to created logic channel. Numeration starts from figure one;
- **Scale** – input signal voltage range for this physical channel; - **BridgeType** – type of bridge connection.

Description: The function creates a logic channel, i.e. generates a 32-bit word that should be written to the corresponding cell in the logic channel table. Refer to Chapter 3.2. of this Manual to get more information on the word format.

Returned value: Generated value of the logic channel, type **int.**

Format: INT LTR212_ SetADC(PTLTR212 hnd)

Parameter: hnd – a pointer to description structure of the module **PTLTR212**

Description: Channel data and acquisition mode data transmission function. Information of used physical channels, ranges, filtration mode is downloaded into DSP memory storage and ADC registers AD7730 installed on the module board. If calibration factors are used the function shall download them into calibration ADC registers AD7730 corresponding to required channels. When using program filters their factors shall be downloaded to internal memory of the module DSP. Prior to use of this function all fields of the module

description structure shall be filled with required values. Acquisition start should be initiated *only after execution of this function*.

Returned value: error code, type **int.** If "0" – bug-free function

Format: INT LTR212_Start(PTLTR212 hnd)

Parameter: hnd – a pointer to description structure of the module **PTLTR212**

Description: Executes acquisition start. Prior to the function call it is necessary that the function **LTR212_SetADC() is executed.**

Returned value: error code, type **int.** If "0" – bug-free function

Format: INT LTR212_Stop(PTLTR212 hnd)

Parameter: hnd – a pointer to description structure of the module **PTLTR212**

Description: Executes acquisition stop.

Returned value: error code, type **int.** If "0" – bug-free function

Format: INT LTR212_Recv(PTLTR212 hnd, DWORD *data, DWORD *tmark, DWORD size, DWORD timeout)

Parameter: hnd – a pointer to the structure of the module description **PTLTR212;**

***data –** a pointer to the array with input data;

***tmark –** a pointer to the array with the second mark and a mark START;

size – quantity of words in requested data array;

timeout – function timeout (millisecond) for requested number of words

Description: Executes acquisition of the words array from the module with the size **size**. Words received at the function output contain in an array, addressed by the pointer **data**. The pointer **tmark** addresses the array containing labels (Second and START), if any. **If the elements of this array are not used in the program, then, as the tmark parameter value, NULL may be used**. The parameter **timeout** defines time in milliseconds within which the function shall wait for the number of words to be retrieved. If the required amount is received before the timeout expires, the function terminates immediately. If the required amount of words has not been received after the timeout, the function shall still be terminated. The return value of a function is the amount of words received from the module. If the return value is negative, then this indicates an error. In this case, you should identify the error with the function **LTR212_GetErrorString().**

*Note: The description of this function corresponds to the description of the crate-controller's ltrap.dll library function **LTR_Recv()**.*

Returned value: If the value is positive or equal to 0, then it corresponds to the number of words received from the module. If it is negative, then it is an error code.

Format: INT LTR212_ProcessData(PTLTR212 hnd, DWORD *src, double *dest, DWORD *size, BOOL volt)

Parameters:

- **hnd –** a pointer to description structure of the module **PTLTR212**
- ***src** – a pointer to the array of data received from the module via previously called function **LTRRcv()**.
- ***dest** – a pointer to the array whereto input data generated by the concerned function shall be recorded.

- **volt –** flag of conversion of received ADC code values into voltage value (V).
- ***size** – a pointer to the data array size variable

Description: The function allows for check of data received from the module and their transformation from the internal format of LTR system directly into code received from ADC. The function also is capable of code conversion into voltage (V) in accordance with ranges selected for applied channels. **src[]** – data array acquired from the module via previously called function **LTR_Recv()**. The function compares channel sequence defined upon generation of logic channel table with sequence required directly from module. In case of non-conformity the function shall suppress frame where the conformity was detected and it shall be replaced with the next one in the output array **dest[]**. "Error" frame shall be deleted in whatever point the required channel sequence has failed. Nonetheless **the function returns the error** despite that the array is adjusted and in fact further operation is possible. The function traces state of sent data package counter. In case of its failure the function shall not interrupt operation, but it shall return an error, and the output array is generated by rules specified above. As a parameter **src** array of any length may be added to this function and any number of points may be processed (determined by **size** parameter), but one should keep in mind that a number of processed points shall be in multiples of quantity of used channels. Otherwise the function shall terminate with an error.

dest[] – an array whereto input data generated by the concerned function shall be recorded. Herewith the function deletes all operating fields from the input array components. If volt parameter is equal to "0" **the** output array is a sequence of codes received from the module ADC frame by frame, i.e. in order in compliance with the logic channel table. If **volt** parameter is null then the output array shall contain volt values calculated per channel ranges set out in logic channel table. It should be noted that in both cases the output array **dest[]** is type **double**.

size – at the function inlet this value corresponds to quantity of the input array components **src[]**. If upon data validation check the concerned function has not detected any failure in sample sequence, then after the function completion the **size** parameter value shall be reduced by half (as each sample in initial array was determined by two adjacent components). If error frames were detected and deleted then the **size** value shall be less, as quantity of the output array components shall reduce through removal of error frames. In this case the **size** parameter shall match half size of the input array Refer to Chapter 4.1 for more detailed information

Returned value: error code, type **int.** If "0" – bug-free function

Format: INT LTR212_Calibrate(PTLTR212 hnd, BYTE *Lchannel_Mask, INT mode, INT reset)

Parameters:

- **hnd –** a pointer to the structure of the module description **PTLTR212;**
- ***LChannel_Mask** – a pointer to the logic channel mask subject to calibration It is a bite where numbers of on-bits coincide with numbers of logic channels subject to calibration. For example, **LChannel_Mask**=0x93 (10010011 in binary system) indicates that logic channels 0, 1, 4, 7 are subject to calibration. These channels are calibrated simultaneously. In case of successful calibration of all channels the mask value returned from function shall not change. This parameter may vary only in case of zero output calibration failure if the integrated DAC value does not match. Then mask bits conforming to uncalibrated channels remain equal to one. Bits conforming to calibrated channels are equal to 0. Hereby in case of error function completion it can be observed which channels were not calibrated. It should be noted that at the first calibration error the function terminates, and calibration of other channels is failed. In case of successful calibration of all channels the mask value shall not change.
- **mode** – calibration mode;
- **reset** – reset flag for all ADC AD7730 prior to calibration. "1" - all ADC are reset, "0" - are not reset.

Description: Performs calibration of specified logic channels followed by calibration factor record to the module PROM. Refer to Chapter 5 "*Module calibration characteristics*" for detailed information about calibration modes. Prior to calibration logic channel mask subject to calibration should be determined.

Returned value: error code, type **int.** If "0" – bug-free function

Format: LPCSTR LTR212_GetErrorString(INT Error_Code)

Parameters:

- **Error_Code** – error code;

Description: Returns a string describing the error corresponding to the code **Error_Code**

Returned value: string corresponding to this error code

INT LTR212_CalcFS(PTLTR212 hnd, double *fsBase, double *fs)

Parameters:

- **hnd –** a pointer to the structure of the module description **PTLTR212;**
- ***fsBase –** a pointer to the ADC discreteness frequency vale upon acquisition**;**
- ***fs** – a pointer to the data output frequency value

Description: Returns data output frequency. When using 4-channel mean accuracy mode without program filters and 4-channel high accuracy mode this frequency matches ADC discreteness frequency. When using feedback filter **fsBase** – ADC discreteness frequency and **fs** is less than **fsBase** by a factor equal to decimation factor **(fs=fsBase/hnd->filter.Decimation**). In eight-channel mode **fsBase** – ADC discreteness frequency in the hardware digital filter tapping, **fsBase**=150.15 Hz, **fs** – data output frequency equal to approximately 3.4 Hz.

Returned value: ----------

INT LTR212_CalcTimeout(PTLTR212 hnd, DWORD n)

Parameters:

- **hnd –** a pointer to the structure of the module description **PTLTR212;**
- **n** – number of points per channel.

Description: Returns maximum time in milliseconds during which the function **LTR212_Recv()** shall wait for data if *a number of points per channel is equal* to **n**. The function calculates timeout via the mode set in the module description structure.

Returned value: maximum time in milliseconds during which the function **LTR212_Recv()** shall wait for data.

Format: INT LTR212_TestEEPROM(PTLTR212 hnd)

Parameters:

- **hnd –** a pointer to description structure of the module **PTLTR212**

Description: The function controls integrity of data recorded in the module PROM.

Returned value: error code, type **int.** If "0" – bug-free function

Format: INT LTR212_ReadUSR_Clb (PTLTR212 hnd, TLTR212_Usr_Clb *CALLIBR)

Parameters:

- **hnd –** a pointer to the structure of the module description **PTLTR212;**
- **CALLIBR –** a pointer to the structure for user-defined calibration saving

Description: Function reading user-defined calibrations from the module PROM.
Returned value: error code, type **int.** If "0" – bug-free function

Format: INT LTR212_WriteUSR_Clb(PTLTR212 hnd, TLTR212_Usr_Clb *CALLIBR)
Parameters: - **hnd –** a pointer to the structure of the module description **PTLTR212;** - **CALLIBR –** a pointer to the structure containing user-defined calibrations
Description: Function recording user-defined calibrations to the module PROM.
Returned value: error code, type **int.** If "0" – bug-free function

Attention!!! Do not confuse data types: module description structure (type TLTR212) and a ***POINTER*** module description structure (type PTLTR212).

4 Data acquisition process characteristics.

4.1 Data acquisition cycle formation

Acquisition start and stop are executed by functions **LTR212_Start()** and **LTR212_Stop().** Data array reading received from the module is executed by the function **LTR_212Recv()**. In this function the parameter **hnd** – a pointer to the module description structure ***data** – a pointer to the array whereto the module data shall be recorded. **size** – number of data words requested from the module. Parameter ***tmark** is a pointer to the array of synchromarks. If no sync marks are applied it may be put as NULL. The **timeout** parameter specifies the maximum waiting time for the requested number of data words in milliseconds. The function returns the number of words received. If the value returned by the function is negative, this indicates that the function completed with an error and returned its code. In this case, you need to call the **LTR212_GetErrorString()** function to define the error by using its code as a parameter.

*One should bear in mind that whereas this module contains 24-bit ADC then 2 data words are required for transmission of each sample; one word contains the most significant bite of 24-bit sample, and another word - the average and least-significant bytes. Therefore double quantity of received words shall be set in the function **LTR212_Recv()** regarding **to** quantity of collection points from which information is requested.*

In the resulting array, the data is arranged frame by frame. **Frame** is a sequence of data words received upon execution of a single polling cycle for all channels used. Data is arranged in the frame in the order of polling of physical channels, i.e. in ascending order. Each input array frame contains the number of components equal to **twice** the number of involved channels. Frame content shall be determined by the logic channel table as it indicates which channels are on, and which channels are off.

After receiving data array from the module by the function **LTR212_Rcv()** delete all operating fields from its components, validate received data and convert ADC code values to signal voltage values in volts, if required, per input voltage ranges for each channel. For this purpose the following function is used

LTR212_ProcessData(*hnd, *src, *dest, *size, volt). Its parameter ***src** is a pointer to initial data array previously received by the function **LTR_Recv(); *dest** – a pointer to the output array which may include either ADC codes or voltage values in volts. It shall be defined by parameter **volt**. If **volt**=0 then the output array shall contain ADC codes; if **volt**=1 then it shall contain values converted into voltage in volts. Anyway this array shall be type **double**. In output array **dest[]** data is also arranged frame by frame, but in case of successful completion of function the number of components shall be equal to the number of collection points, i.e. twice as small as the number of components of the input array **src[].** . Whereby in the invocation of function **LTR212_ProcessData()** the parameter **size** should be set twice the number of collection points, i.e. equal to the number of the input array components **src[]**. The function shall change value of this parameter, and at the exit the parameter **size** shall be equal to half of initial value. As a rule, acquisition process is cyclic (refer to Fig. 1.). Acquisition cycle should be arranged in a separate thread to ensure its effective performance. The example of data acquisition and processing via function **LTR212_Recv()** and **LTR212_ProcessData()**.

For example, we wish to receive data from two collection points per each channel. Let the variable **hltr212** be an instance of the module description structure type **TLTR212**. In this example we shall use the same logic channel table as we used in Chapter 3.2, but the input signal range for each channel shall be set +/- 80 mV. Then Logical Channel Table shall be as follows:

hltr212.LChTbl:

0x00010003
0x00020003
0x00040003
0x00050003
0x00070003
0x00080003

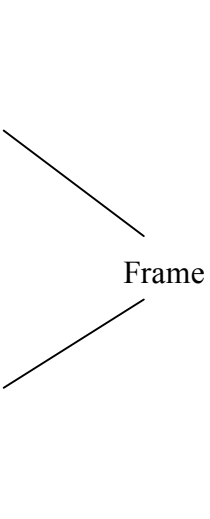
We have the following physical channels involved: 1, 2, 4, 5, 7 and 8. To receive data array from two points of each channel after module configuration and acquisition start we shall use function **LTR212_Recv().** As the number of involved channels is equal to 6 and we wish to receive data from collection points for each channel, then we shall receive information from 6*2=12 points. As referenced above, parameter **size** shall contain double number of collection points, or **size**=12*2=24. Parameter **timeout** is set equal to 0.5s=500ms. **timeout**= 500. **src[]**- the array whereto data words are recorded from the module. After initialization, opening, configuration of module and acquisition start we shall apply the function with parameters indicated above: **LTR212_Recv(&hltr212, src, NULL, 24, 500).** On return form function, the array of "raw" data **src[]** shall be as follows, for example:

src[]:

0x007F0800	
0xFF220810	
0x00840821	
0x03790831	
0x008C0843	
0x0E560853	
0x00820864	
0x03010874	
0x008A0886	
0x05140896	
0x 008E08A7	
0x 0D1508B7	Frame
0x 007F08C0	
0x FF2708D0	
0x 008408E1	
0x 035508F1	
0x 008C0803	
0x 0E5D0813	
0x 00820824	
0x 02DB0834	
0x 008A0846	
0x 05110856	
0x 008E0867	
0x D200877	

As reflected by example, samples in frames are located cyclically in ascending order of physical channels. This is the "raw data" received directly from crate controller. Number of physical channel is presented by four least-significant bits in each data package. In this channel counter the numeration starts from null. Each sample matches two array components. The first one contains the most-significant bite of received sample, the second one - mean and the least-significant bites (refer to Annex 2 for more detailed information about data package format). Then validate data, delete operating fields and, for our example, convert voltage values of input signal. For this purpose call function **LTR212_ProcessData().** Parameter **src[]** – data array previously received by the function **LTR212_Recv()**. Parameter **size** is equal to the number of its components, **size**=24. **dest[]** – output array, containing voltage values in volts for input signals in our case. Parameter **volt**=1 as we wish to recompute ADC code data in the input signal voltage value. Invocation of function **LTR212_ProcessData()** shall be as follows: **LTR212_ProcessData(&hltr212, src, dest, &size, 1)**, where **size**=24. The function, after validation, deletion of operating fields and combination of all bites of each sample, acquired ADC code values and converted these values to volts. Conversion to voltage values on the assumption that input signal range for all channels is +/- 80 mV, as defined above in logic channel table. Output array **dest[]** after operation of the function **LTR212_ProcessData()** shall be as follows: **dest[]:**

-2.117156982421875 E-6
2.508478164672852 E-3
7.534999847412110 E-3
1.257333755493164 E-3
6.262397766113282 E-3
8.781938552856446 E-3
-2.069473266701562 E-6
2.508134841918946 E-3
7.535066704614258 E-3
1.256971359252930 E-3
6.262369155883789 E-3
8.782043457031249 E-3



As reflected by this example, after operation of the function **LTR212_ProcessData()** output array contains twice less components than the initial one. Parameter **size** is currently also equal to 12. Now this data may be used, for example, for visualization. It should always be remembered that the frame, i.e. data sequence in the output array corresponds to physical channel sequence specified in the logic channel table. Number of channel corresponding to this component of the output array shall also be defined by the logic channel table. It must be always used during software coding. Summarizing the above, we site a small fragment of the program code demonstrating data acquisition from the module based on conditions of concerned example.

```
DWORD src[24];
DWORD dest[12];
DWORD size;
TLTR212 hltr212;

/*****************************************************
Data initialization, opening, configuration and acquisition start
.....
/*****************************************************/
// Data acquisition:
```

```
// If required, cycle conditions
 size=24;
 LTR212_Recv(&hltr212, src, NULL, size, 500);  LTR212_ProcessData(src,
dest, &size, 1);
 ……..
 /*****************************************************
 Acquisition cycle stop, data acquisition stop
 *****************************************************/
```

Except for abovementioned actions, the function validates data words path controlling special counters (expedited word counter and channel counter) containing in each word. The function compares channel sequence defined upon generation of logic channel table with sequence required directly from module. In case of non-conformity the function shall suppress frame where the conformity was detected and it shall be replaced with the next one in the output array **dest[]**. "Error" frame shall be deleted in whatever point the required channel sequence has failed. Nonetheless **the function returns the error** despite that the array is adjusted and in fact further operation is possible. The function traces state of sent data package counter. In case of failure it continues further array processing, but it shall be executed with an error. Irrespective of error in the counter of expedited data packages the channel counter shall be checked such as it was described above, and error frames shall be deleted, if any. Refer to Annex 2 for more detailed information about these counters. If "error" frames were detected then at the function exit the parameter **size** shall be not equal to initial value, it shall be less than half of it, as error frames shall be deleted from the output array dest[].

4.2 Data acquisition mode characteristics.

The module has three acquisition modes, two of which are **four-channel** and one is **eight-channel**.

In **four-channel** modes all four ADC AD7730, installed on the module board operate in a synchronous manner and channels fail to switch. Acquisition is continuous. In **eight-channel** mode channels switch in each ADC AD7730. In eight-channel mode the acquisition circuit shall be as follows: the first data acquisition channel is activated in each of four ADC AD7730 installed on the module (physical channels 1, 2, 3, 4). All ADC get started in a synchronous manner for a single acquisition. Received readings are accepted from registers of all ADC and are sent to the crate controller and PC. Then the second channel is activated in each ADC (physical channels 5, 6, 7, 8) and the same actions occur. Afterwards the first group of channels switch on again and the process repeats cyclically. In this mode upon filling of hardware filters the data acquisition is executed on a frequency of set filtration mode (150.1 Hz), but data output frequency in PC shall be significantly less (about 3.4 Hz), as ADC reprogramming, acquisition by another channel group and data transfer to the crate controller and PC requires time upon channel switching. To determine reading output frequency both in four- and eight-channel modes the function **LTR212_CalcFS() is used.**

One should bare in mind that data output frequency in eight-channel mode is not accurate, as the acquisition process is not synchronous! In particular, the frequency depends on number of involved channels. In this regard 8-channel mode should not be used in applications requiring for frequency accuracy and time intervals.

In **four-** and **eight-channel** high-accuracy modes the alternating reference voltage may be used. In this case bridges are supplied with ac voltage with compensation of thermal electromotive force effect occurring upon connection of measuring bridges to the module. The frequency of switching the supply voltage of the bridges is equal to half the frequency of data acquisition. One should bear in mind that **alternating reference voltage is not applied in four-channel mean-accuracy mode.**

5 Module calibration.

5.1 Calibration modes.

Each of 4[x] ADCs AD7730 installed on the module board has integrated calibration functions, special calibration registers (offset and gain), and integrated digital-to-analog converter (DAC). These facilities allow for internal and external calibration of the module channel.

From the software's point of view, calibration is a calibration factor determination and placing to special area of the module PROM hereinafter referred to as the calibration area. If the user wishes to apply calibration factors during acquisition they shall be downloaded from this area to calibration registers of required ADC channels AD7730.

Internal **calibration shall mean** factory calibration factors input (submitted by LLC "L-Card") to the calibration area of the module PROM. Factory calibration factors are stored in another area of PROM which is used only for storage, and data stored there can not be rerecorded.

External **calibration shall** mean determination of calibration factors on the assumption of user-defined measurement conditions and recording of these factors to the calibration area of PROM**.**

Null calibration means calculation (if required) and recording of calibration offset factors and integrated DAC code to the calibration area of PROM.

When acquiring data using offset factors they shall be rerecorded from the calibration area of PROM to **calibration offset registers** of required channels, and to the DAC register. It allows for adjustment of null offset of the input signal. **Range calibration** means calculation (if required) and recording calibration gain factors to the calibration area of PROM. When acquiring data using gain factors they shall be rerecorded from the calibration area of PROM to **calibration gain registers of** required channels. It enables to adjust conformance of actual full range of measured channel with the range specified for the channel. **Full-scale calibration** is both the null calibration and the range calibration. Whereby required calibration factors are input both in PROM cells conforming to offset factors and cells conforming to gain factors**.**

Prior to calibration it is necessary that the acquisition mode is set where determinable calibration factors shall be applied. I.e. the module description structure fields shall be filled prior to calibration. Calibration shall be executed via invocation of the function **LTR212_Calibrate().** Refer to *Chapter 3.3* of the Manual for the function description.

In case of internal calibration this function reads factors out of the factory factors storage area to the calibration area of the module PROM; in case of external calibration factors are calculated and recorded to the calibration area of PROM. After calibration upon setting of the module description structure field **UseCalibration**=1 and further invocation of function **LTR212_SetADC()** calibration factors shall be read out of this PROM area and recorded to **calibration registers** of corresponding ADC channels Parameter **mode** of the function **LTR212_Calibrate()** determines calibration mode. Upon execution of internal calibration the user may not address any signal to the module inputs as in such case the calibration process shall be just a rerecording of factors from one PROM area to another one. In case of **external null calibration** null voltage shall be supplied to the module inputs. In case of **external range calibration** the module channel inputs shall be fed with voltage equal to maximum positive voltage for this range. In case of full-range external calibration start with null calibration, followed with the range calibration. Description of the module calibration modes is provided below.

Table 6

mode	Mode	Description
0	**Internal** null **calibration**	Records factory calibration offset factors to corresponding cells of the PROM calibration areas.
1	**Internal** range **calibration**	Records factory calibration gain factors to corresponding cells of the PROM calibration areas.
2	**Full-scale internal** calibration	Records factory calibration offset and gain factors to corresponding cells of calibration areas of the module PROM.
3	**External** null **calibration**	Calculates calibration offset factors and records them to corresponding cells of calibration areas of the module PROM. During calibration the module inputs should be fed with dc voltage conforming to the null signal
4	External range **calibration**	Calculates calibration gain factors and records them to corresponding cells of calibration areas of the module PROM. During calibration the module inputs should be fed with dc voltage conforming to maximum positive voltage of the input signal voltage range.
5	**Internal range + external null**	Records factory calibration gain factors to corresponding cells of calibration areas of the module PROM, then calculates and records calibration offset factors to PROM. During calibration the module inputs should be fed with dc voltage conforming to the null signal.
6	**External** calibration **range** (the second stage in case of the full-scale external calibration)	Applicable ONLY in case of full-scale external calibration AFTER execution of external null calibration. During calibration the module inputs should be fed with dc voltage conforming to maximum positive voltage of the signal.
7	**External** null **calibration** with reservation of scaling factors	Analogue of the "External null calibration", however, upon execution, scaling factors previously determined during the full-scale external calibration shall be saved.

Parameter **reset** of the function **LTR212_Calibrate()** allows for indication whether calibration registers of ADC AD7730 should be cleared prior to calibration. For example, if null calibration is subject to adjustment without modification to gain factors, then required channels shall be calibrated without

preclean of registers. To clear registers prior to calibration set the parameter **reset**=1. If not applicable then **reset**=0.

Software also assumes possibility of automatic application of factory calibration factors. If the flag **UseFabricClb**=1 is set for each invocation of function **LTR212_SetADC()** upon modification of channel range the factory calibration factors, both offset and gain, corresponding to this range, shall be automatically downloaded to PROM area.

Maximum offset voltage values which may be compensated via external calibration are specified in the table below.

Table 7

Compensated offset in ranges:	
± 10mV	±78.7mV *
± 19mV	±78.7mV*
± 40mV	±79.5mV*
± 80mV	±81.5mV*
10mV,	
19mV,	
40mV,	
80mV,	

* If reference voltage 2.5 V is applied then specified values shall be twice as less.

5.2 Different calibration modes usage pattern.

Calibration of the module *LTR-212(M)* has a number of features which should be considered during coding:

- **Factory calibration** factors:
  - for *LTR212* and *LTR212M-3* these factors are used **ONLY** at reference voltage 5.0 V. AT reference voltage 2.5 V functions of internal calibration shall be executed with an error!!! Upon operation at reference voltage 2.5 V apply external calibration.
  - for *LTR212M-1* and *LTR212M-2* these factors are applied both at reference voltage 5.0 V and at 2.5 V.
- In case of **internal calibration** in **four-channel** mean **accuracy** mode ONLY factory gain factors may be used. That means in this mode only internal range calibration may be executed. At attempt of internal null calibration or full-scale internal calibration the function **LTR212_Calibrate()** shall be executed with an error. External calibration may be used without limitation, just as "Internal range calibration + external null calibration" mode.
- Software provides for the **factory calibration factors direct application mode**. It is switched on via setting of the module description structure field **UseFabricClb** as one (**UseFabricClb**=1). If this mode is switched on then factory calibration offset and gain factors shall be downloaded directly from PROM factory calibration factor storage area to calibration registers of ADC AD7730. Essential factors shall be selected **automatically** according to selected channels and corresponding input signal ranges. Whereby PROM calibration area shall not be modified. This mode enables automatic downloading of factory calibration factors upon switching to different measurement ranges. It also enables factory calibration application without deletion of previously set calibration factors placed in PROM calibration area. Thus, the user may easily switch from his/her own calibration factors to the factory-set factors and vice versa. This mode is not applicable in four-channel mean-accuracy mode. This mode is also not applicable for *LTR212* and*LTR212M-3* at reference voltage 2.5 V. As for *LTR212M-1* and *LTR212M-2* this mode is applicable at both reference voltage values: 2.5 V and 5.0 V.

The Table clarifies the above-said for *LTR212* and *LTR212M-3*.

Table 8

Internal **calibration** mode	**Internal** range calibration	**Internal** range calibration	**Full-scale internal** calibration	**Internal** range **calibration** + **external** null **calibration**	Factory calibration factor direct **application** mode
Four-channel mean-**accuracy** mode	-	+	-	+	-
Four-channel high-**accuracy** mode	+	+	+	+	+
Eight-channel high-**accuracy** mode	+	+	+	+	+
Any mode at **reference** voltage 2.5 V	-	-	-	-	-

It should be noted again that **external calibration may be used without any limitation.**

- In case of full-scale external calibration first of all perform external null calibration (invocation of function **LTR212_Calibrate()** with parameter **mode**=3), then perform external range calibration but as the second stage of the full-scale external calibration (invocation of function **LTR212_Calibrate()** with parameter **mode**=6. But not **mode**=4!!!).

Attention!!! Be careful when switching on signals during calibration! If any channel subject to calibration is "up in the air" or its signal is outside of the input signal voltage range, then calibration shall be executed with an error. In case of calibration function failure always analyze logic channel mask value. If it has changed then values for integrated DAC failed to match. It may indicate whether at the out-of-range signal address compensated via DAC, or at input signals "up in the air".

6 Digital filtration.

6.1 Digital filters used in the module LTR212.

All data acquired by the module as subject to digital filtration. Each of ADC AD7730, installed on the module board has two integrated hardware digital filters: first stage filter and second stage filter. First stage filter is a low-pass filter type $sinc^3$ designed for elimination of quantisation noise occurred in modulator. Second stage filter is a digital feedback filter with 22 taps that process signal at the first stage filter outlet**.**

First stage hardware digital filter is involved in all three operating modes of the module LTR212. **Second stage hardware digital filter is applicable only in 4-channel high-accuracy and 8-channel high-accuracy modes. In 4-channel mean-accuracy mode this filter shall be SWITCHED OFF!**

However this mode enables application of software digital filters developed by "L-CARD". Refer to AD7730 Data Sheet (www.analog.com/en/prod/0,2877,AD7730,00.html) for more detailed information about hardware 1 and 2 stage filters. More detailed information about software digital filters shall be provided in next chapter.

AFC of the module operation using software and hardware filters is specified in the document "Crate system LTR. User manual", Annex A.2.2.1.

Recall that if the software feedback filter is switched on prior to acquisition of the first filtration reading the module DSP memory buffer, equal to the filter order, shall be filled with samples. For example, if the filter order is 225, then after ADC start the filter buffer shall be filled at first (225 words for each channel, and only then the module shall generate data.

If the software infinite-impulse response filter is ON, then initial 30 readings of each channel shall be used to fill in delay line and to provide filter stabilization. The module shall not address this data to the crate controller and PC.

6.2 Software filters application

"L-CARD" has developed a set of software digital filters realized at the module DSP level. They allow for digital data filtration at the maximum acquisition frequency (7680 Hz) which may not be performed using integrated hardware 2 stage filters. It should be bared in mind that ADC operates in 16-bit mode in this software **version if software filters are switched on!** However 16-bit data is addressed to the crate controller in two packages as 24-bit data. Due to this fact the function **LTR212_ProcessData()** is applied as well as in other operating modes. The software filter set includes 2 stage infinite-impulse response filter aligning initial AFC in specified frequency bandwidth with an accuracy of 0.02dB and 5 feedback filters of different order with various frequency values of initial delay line. Existing software filters shall operate **only in 4-channel mean-accuracy mode** at the acquisition frequency 7680 Hz. Filter factors are stored in files that are read out by the function **LTR212_SetADC()** and then are addressed to the module DSP. Full names of files with software filter factors shall be previously indicated in the module description structure components **filter.IIR_Name** and **filter.FIR_Name**. The rest fields of the substructure **filter** shall be automatically filled with values read out of the file.

The infinite-impulse response filter file contains only factors. As for the feedback filter the initial file lines shall contain data of the filter order and decimation factor in the following format:

```
FS=XXXX HZ
DECIMATION= XX
```

For example,

```
FS= 7680 HZ
DECIMATION= 24
0001
0001
…….  Factors
…….
```

Attention! The feedback filter order shall not exceed 255!

The function **LTR212_SetADC()** "understands" only this format of the feedback filter file. The user may individually calculate digital filter and record its factors to the file. The function **LTR212_SetADC()** shall read the factors and download them to the internal memory of the module DSP. The filter may be developed, for example, in the QED environment. Received factors should be multiplied by 32768 and the result shall be rounded to integral two-bite value. Then the text file of the abovementioned format shall be generated whereto the factors should be put in. In this particular kind they may be applied by the software digital filters.

7 PROM check.

User interface library contains useful data integrity check function in the module PROM **LTR212_TestEEPROM().** If the function is not failed that shall mean that data checksum in the module PROM is correct. If the function returns value "–2031" (Incorrect PROM data checksum) then PROM data is damaged. In spite of the fact, the module operation may be continued. However consider that data in some PROM cells may be incorrect. This function may be applicable, for example, each time as the module is opened (after execution of the function **LTR212_Open()**) to control integrity of calibration factors and identification record.

Annex 1. Examples of code writing.

P1.1 Configuration examples.

Prior to set configuration it is necessary to initialize and open interface channel for communication with the module. It is done by calling the following functions: **LTR212_Init()** and **LTR212_Open().** Then you must fill in the fields of the module description structure with the required values. The following are examples of setting the module configuration (defining the fields of the module description structure):

1. **Four-channel mean-accuracy mode**, acquisition frequency 7600 Hz, software filters are ON. The feedback filter with the initial frequency of delay line 345 Hz is selected. All 4 channels are involved. Input signal voltage range for all channels is 80 mV. Calibration factors are applied. Constant reference voltage 5 V.

```
TLTR212 conf_1;  // Declare structure type TLTR212
...
conf_1.AcqMode=0;  // 4-channel mean-accuracy mode
conf_1.UseClb=1;  // Apply calibration factors
conf_1.UseFabricClb=0;  // Do not apply factory calibration factors
conf_1.LChQnt=4;      // Quantity of logic channels is equal to 4

// Fill the Logical Channel Table in. For each channel set  // input signal range 80
mV.                                ±

for(i=0; i< conf_1.LChQnt; i++)

conf_1.LChTbl[i]=LTR212_CreateLChannel(i,3);
// Set the parameters of software filters
conf_1.filter.IIR=1;  // Infinite-impulse response filter ON
conf_1.filter.FIR=1;  // Feedback filter ON
conf_1.filter.IIRName="C://Filter// D212_IIR.flt";  // Filter files
conf_1.filter.FIRName="C://Filter// D212_345.flt";
// Set reference voltage polarity and value
conf_2.AC=0;     // DC reference voltage
conf_1.REF=1;  // Reference voltage 5 V.
```

2. **4-channel high-accuracy mode**, acquisition frequency 150.1 Hz, software filters OFF. Hardware filters in ADC AD7730 ON. 2 logic channels corresponding to physical channels 1 and 4 are involved. Ranges 10mV and +40mV correspondingly. Calibration factors are applied. Constant reference voltage 2.5 V.

```
TLTR212 conf_2;  // Declare structure type TLTR212
...
conf_2.AcqMode=1;  // 4-channel high-accuracy mode
conf_2.UseClb=1;  // Apply calibration factors
conf_1.UseFabricClb=0;  // Do not apply factory calibration factors
```

```
conf_2.LChQnt=2;      // Quantity of logic channels is equal to 2

// Fill the Logic channel table in.
conf_2.LChTbl[0]=LTR212_CreateLChannel(1,0);  // physical channel 1, range ±10mV
conf_2.LChTbl[1]=LTR212_CreateLChannel(4,6);  // physical channel 4, range +40mV
// Set the parameters of software filters
conf_2.filter.IIR=0;       // Infinite-impulse response filter OFF
conf_2.filter.FIR=0;       // Feedback filter OFF

// Reference voltage
conf_2.AC=0;    // DC reference voltage
conf_2.REF=0;   // Reference voltage 2,5 V.
```

3. **4-channel high-accuracy mode**, acquisition frequency 150.1 Hz, software filters OFF. 3 logic channels corresponding to physical channels 1, 3, and 4 are involved. Channel range 10 mV, +20 mV and +80 mV correspondingly. Factory calibration factors are applied. Alternating reference voltage 5 V.

```
TLTR212 conf_3;       // Declare structure type TLTR212

conf_3.Mode=1;        // 4-channel high-accuracy mode
conf_3.UseClb=0;      // Do not apply calibration factors
conf_3.UseFabricClb=1;    // Apply factory calibration factors
conf_3.LChQnt=3;      // Quantity of logic channels is equal to 3

// Fill the Logic channel table in.
conf_3.LChTbl[0]=LTR212_CreateLChannel(1,0);  // physical channel 1, range ±10mV
conf_3.LChTbl[1]=LTR212_CreateLChannel(3,5);  // physical channel 3, range +20mV
conf_3.LChTbl[2]=LTR212_CreateLChannel(4,7);  // physical channel 4, range +80mV
// Set the parameters of software filters
conf_3.filter.IIR=0;       // Infinite-impulse response filter OFF
conf_3.filter.FIR=0;       // Feedback filter OFF

// Reference voltage
conf_3.AC=1;    // Alternating reference voltage
conf_3.codec.REF=1;       // Reference voltage 5 V.
```

4. **8-channel high-accuracy mode**, approximate acquisition frequency 3.4 Hz, software filters OFF. Interruption mode OFF. 6 logic channels corresponding to physical channels: 1, 2, 3, 4, 5, 8 are involved. Range for all channels 20 mV. Calibration factors are applied. Constant reference voltage 5 V.

```
TLTR212 conf_4;       // Declare instance of the  module description structure

conf_4.size=Sizeof(PTLTR212);        // Structure size
conf_4.Mode=1;                // 8-channel high-accuracy mode
```

```
conf_4.UseClb=1;          // Apply calibration factors
conf_4.UseFabricClb=0;    // Do not apply factory calibration factors
conf_4.LChQnt=6;          // Quantity of virtual channels is equal to 6

// Fill the Logic channel table in.
conf_4.LChTbl[0]=LTR212_CreateLChannel(1,1);  // physical channel 1, range ±20mV
conf_4.LChTbl[1]=LTR212_CreateLChannel(2,1);  // physical channel 2, range ±20mV
conf_4.LChTbl[2]=LTR212_CreateLChannel(3,1);  // physical channel 3, range ±20mV
conf_4.LChTbl[3]=LTR212_CreateLChannel(4,1);  // physical channel 4, range ±20mV
conf_4.LChTbl[4]=LTR212_CreateLChannel(5,1);  // physical channel 5, range ±20mV
conf_4.LChTbl[5]=LTR212_CreateLChannel(8,1);  // physical channel 8, range ±20mV
// Set the parameters of software filters
conf_4.filter.IIR=0;      // Infinite-impulse response filter OFF
conf_4.filter.FIR=0;      // Feedback filter OFF

// Reference voltage
conf_4.codec.AC=0;              // DC reference voltage
conf_4.codec.REF=1;             // Reference voltage 5 V.
```

P1.2. An example of an application.

Simple console application written in the Microsoft Visual C++ 2005 environment.

```
#pragma hdrstop

#include "ltr\\include\\ltr212api.h"
/* other header files */
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <conio.h>

#define ACQ_BLOCKS_QNT (2)
#define TOTAL_BLOCKS_QNT (10)
#define POINTS_PER_CHANNEL (256)
#define CHANNELS_QNT (4)

TLTR212 hltr212;       // The instance of the module description structure
HANDLE AcqThreadHnd;   // Acquisition thread
CHAR ErrorString[255]; // The string for outputting the error description
CHAR MsgString[255]; // The line for outputting messages to the console

double voltage[ACQ_BLOCKS_QNT][CHANNELS_QNT*POINTS_PER_CHANNEL]; static
volatile int RunFlag = 0; // Acquisition run flag static volatile int
BlockReady[ACQ_BLOCKS_QNT];

static DWORD WINAPI AcqThread (LPVOID param);

static DWORD WINAPI AcquireThread(LPVOID param)
```

```
{  int i;  int
err=0;  INT
DataCntr=0;
 DWORD data[2*POINTS_PER_CHANNEL*CHANNELS_QNT];
 DWORD to; // Timeout
INT AcqBlockCntr=0;
int BlockCntr=0;
 DWORD size; // Data portion size
 DWORD ExitCode;


    /* The function started as a data acquisition thread. */
for(i=0;i<ACQ_BLOCKS_QNT;i++)
    BlockReady[i]=0;

err=LTR212_Start(&hltr212);
if(err)  {
   strcpy(ErrorString, (char *) LTR212_GetErrorString(err));
   CharToOem(ErrorString,ErrorString);
   printf("%s",ErrorString);
   LTR212_Stop(&hltr212);
      RunFlag=0;

}
  size=2*POINTS_PER_CHANNEL*CHANNELS_QNT; // Data portion size
to=LTR212_CalcTimeOut(&hltr212, POINTS_PER_CHANNEL);
   while(RunFlag) // while acquisition run
flag...
  {
   size=2*POINTS_PER_CHANNEL*CHANNELS_QNT;

   DataCntr=LTR212_Recv(&hltr212, data, NULL, size, to);
if(DataCntr!=size)
    {
     RunFlag=0;
     if(DataCntr>=0)
    {
       sprintf(MsgString,"%s", "Error! Quantity of received samples is less
than requested one!\n");
       CharToOem(MsgString,MsgString);
       printf("%s\n",MsgString);
    }
     else
{
        err=DataCntr;
        strcpy(ErrorString, (char *) LTR212_GetErrorString(err));
      CharToOem(ErrorString,ErrorString);
printf("%s",ErrorString);
       }
break;
     } // к if(DataCntr!=size)

     err=LTR212_ProcessData(&hltr212, data, voltage[BlockCntr], &size,
1);      if(err)       {
        strcpy(ErrorString, (char *) LTR212_GetErrorString(err));
      CharToOem(ErrorString,ErrorString);
printf("%s",ErrorString);
```

```
        RunFlag=0;
break;
       }

     AcqBlockCntr++;
       sprintf(MsgString,"%-th data portion received",
AcqBlockCntr);
      CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);

  /* Block interleaving is used for synchronization of file record in the
main thread */

      BlockReady[BlockCntr]=1;
      BlockCntr++;

if(BlockCntr>=ACQ_BLOCKS_QNT)
        BlockCntr=0;

  } // to while(RunFlag)

  RunFlag=0;
  err=LTR212_Stop(&hltr212);
  if(err)
{
    strcpy(ErrorString, (char *) LTR212_GetErrorString(err));
   CharToOem(ErrorString,ErrorString);
   printf("%s",ErrorString);
  }

  ExitThread(0);
return 0;
}
void main()
{

DWORD AcqThreadId;
DWORD ThreadSatus;

INT TotalBlockCntr; // Counter of blocks recorded to the file
INT err;            // Variable for error code storage
INT i;              // applied for ineraction counters in cycles DWORD
data[2*5000]; /* array whereto module data is recorded*/ FILE
*DataFile;
const char FileName[] = "ltr212_data.bin";
 int
BlockCntr=0;
int
BlockNumber;

/* We initialize the communication channel for the module. Fields of the
module description structure are filled with default values. */

sprintf(MsgString,"%s", "Initialization of the module description
structure\n");
```

```
CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);

 err=LTR212_Init(&hltr212);  if(err)    {
   strcpy(ErrorString, (char *) LTR212_GetErrorString(err));
   CharToOem(ErrorString,ErrorString);
   printf("%s",ErrorString);
   LTR212_Close(&hltr212);
   Sleep(3000);
   return;

}
// Open the communication channel with the module. Network address and port
number by default
// Serial number of the first detected crate;
// Slot number - 0;
 err=LTR212_Open(&hltr212, SADDR_DEFAULT, SPORT_DEFAULT, "",
CC_MODULE1,
"ltr212.bio")
;  if(err)
{
   if(err==LTR_WARNING_MODULE_IN_USE)
    {
         strcpy(ErrorString, (char *) LTR212_GetErrorString(err));
       CharToOem(ErrorString,ErrorString);
printf("%s",ErrorString);
       }
   else
   {
    strcpy(ErrorString, (char *) LTR212_GetErrorString(err));
    CharToOem(ErrorString,ErrorString);
printf("%s",ErrorString);
    Sleep(3000);
    return;
   }

}
/* Fill in the fields for information. Here is done only for demonstration
*/
sprintf(MsgString,"Module name: %s", hltr212.ModuleInfo.Name);
CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);

sprintf(MsgString,"BIOS version: %s", hltr212.ModuleInfo.BiosVersion);
CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);

sprintf(MsgString,"BIOS date of issue: %s", hltr212.ModuleInfo.BiosDate);
CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);

sprintf(MsgString,"Module serial number: %s", hltr212.ModuleInfo.Serial);
CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);
```

```
// Fill the fields of the module description structure with the required
values

hltr212.size=sizeof(TLTR212);  // Structure size
hltr212.AcqMode=1;             // 4-channel high-accuracy mode
hltr212.UseClb=0;     // Do not apply user-defined calibration factors
hltr212.UseFabricClb=1;        // Apply factory calibration factors.
hltr212.LChQnt=4;              // Quantity of logic channels is equal to 4
 hltr212.REF=1;      // Reference voltage 5 V.
hltr212.AC=0;      //  DC reference voltage

for(i=0; i<hltr212.LChQnt; i++)
  hltr212.LChTbl[i]=LTR212_CreateLChannel(i+1,3);

// Address ADC parameters to the module control program
 err=LTR212_SetADC(&hltr212); if(err)  if(err)    {
   strcpy(ErrorString, (char *) LTR212_GetErrorString(err));
   CharToOem(ErrorString,ErrorString);
   printf("%s",ErrorString);
   LTR212_Close;
   Sleep(3000);
   return;
   }

 DataFile = fopen(FileName, "wb");
 if(DataFile==NULL)
  {
   strcpy(MsgString,"Could not create file!");
   CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);
   LTR212_Close(&hltr212);
   Sleep(3000);
return;
  }
 RunFlag = 1;

// Create acquisition thread
AcqThreadHnd=CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)AcquireThread,
NULL, CREATE_SUSPENDED, (LPDWORD) &AcqThreadId);

if(AcqThreadHnd==NULL)
{
   strcpy(MsgString,"Could not create acquisition thread!");
   CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);
   LTR212_Close(&hltr212);
   Sleep(3000);
return;
}

// Resume thread
ResumeThread(AcqThreadHnd);

 strcpy(MsgString,"Acquisition thread is resumed!\n");
```

```
 CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);

TotalBlockCntr  = 0;

while(RunFlag)
{
 if(BlockReady[BlockCntr]) // If regular block is read
 {
   fwrite(voltage[BlockCntr], sizeof voltage[0][0],
   CHANNELS_QNT*POINTS_PER_CHANNEL, DataFile);

   TotalBlockCntr++; // Block counter
    sprintf(MsgString,"%d-я %s\n",TotalBlockCntr, "data portion recorded
to the file");
   CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);

   BlockReady[BlockCntr]=0;
   BlockCntr++;

   if(BlockCntr>=2)
    BlockCntr=0;
    if (TotalBlockCntr >= TOTAL_BLOCKS_QNT)
    {
     strcpy(MsgString,">> Acquisition is successfully executed.\n");
     CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);       RunFlag = 0;
     }
    } // to if(BlockReady[BlockCntr]...)
  }    // to while(RunFlag)

//Monitor thread stop
  WaitForSingleObject(AcqThreadHnd, INFINITE);
  GetExitCodeThread(AcqThreadHnd, &ThreadSatus);

  CloseHandle(AcqThreadHnd);

   strcpy(MsgString,"Acquisition thread deleted.\n");
   CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);

   fclose(DataFile);
   strcpy(MsgString,"Data record file closed.\n");
   CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);

   strcpy(MsgString,">> Press any key for exit\n");
   CharToOem(MsgString,MsgString);
printf("%s\n",MsgString);
while(!kbhit())
        continue;
   return;
}
```

Annex 2. Protocol of data exchange with the module.

The data exchange protocol with the module is based on the use of the format of 4-byte instruction packets or data. This format is described in details in the book "LTR Crate System. User manual". Ch. 4.3. Here we turn our attention to information with a meaning applicable to the module LTR212.

All commands from the crate controller to the module and the acknowledgment of these commands are 4-byte **command words**. The data collected by the module ADC and transmitted from the module to the crate controller are 4-byte **data words**. It should be noted that this protocol is common for all modules of this crate system.

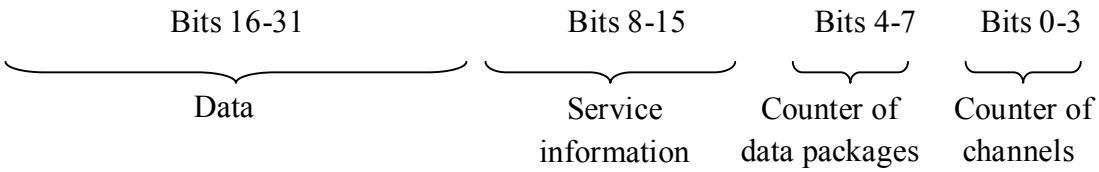
The **command word** format applicable to the module LTR212 is as follows:

- **The command code** is a number that specifies the *BIOS* procedure to be performed by the processor of the module.
- **Service information** - is an information about the slot number, tag-bit of the command word, time tag, second tag. This byte carries the level information of the crate controller. In addition to the command tag-bit or data bit, these bits are not transmitted to and from the module.
- **The command context** is the values passed along with the command and used by the processor when executing it. For example, calibration parameters, ADC register content during programming, etc.

The commands in the described format are also transmitted in the opposite direction: from module to crate controller. In this case, they represent either confirmations of the execution of commands, or contain in the context fields the values that were required from the module when programming it (but not when collecting data!!!). For example, content of the read out ADC register AD7730.

Data words are used in this module only to transfer data from the module to the crate controller and the PC during data collection. When programming the module, data words **are not used**.

The format of the **data word** is as follows:



- **Data** - ADC codes transferred from the module to the crate controller and PC.
- **Service information** - is an information about the slot number, tag-bit of the command word, time tag, second tag. This byte carries the level information of the crate controller. In addition to the command tag-bit or data bit, these bits are not transmitted to and from the module.
- **Data package counter** – 4-bit counter for data packages transmitted by the module to crate-controller. Whenever each new package is sent, the processor increments the value of this counter, which is then be used to verify the consistency of the packages from the module.
- **Channel counter** – this field shall be filled in by the processor with the physical channel number whereof the addressed data was acquired, for each data package. Applicable for verification of packages consistency from the module. Channel numeration starts from null.