A family of universal modules of the ADC/DAC

L502/E502

Programmer manual

Revision 1.1.7
November 2016

L-CARD

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

DAQ SYSTEMS DESIGN, MANUFACTURING & DISTRIBUTION

Author of the manual:
Alexey Borisov

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
Technical support: en@lcard.ru

Table 1: Current document revisions

Revision	Date	Description
1.0.0	27.06.2012	The document first revision
1.0.1	22.11.2012	The description of library application with programs on C# and in LabView has been added, the description of installation for Linux OS has been supplemented as well as the description of functions for cyclic output
1.0.2	20.02.2013	The reference to SDK initial codes has been added. The description of packages installation for Linux has been issued as a separate document
1.0.3	16.02.2015	The sequence of module handling upon synchronous stream output has been corrected. Note on transfer of arrays as output parameters to LabView has been added
1.1.0	02.06.2015	The description has been changed in compliance with modifications implemented into the library in order to support E502 module (implementation of general and specific functions). Brief description of modules differences on the software part has been included. Separate chapters describing the module setting up upon operation through Ethernet and search of modules in the local network have been supplemented.
1.1.1	06.07.2015	The description of possibility of waiting for completion of cyclic signal installation added in 1.1.2 version of the library has been supplemented. The path for updated firmware download having the recommendation on updating has been provided in the section of modules differences when describing ARM-controller availability in E502.
1.1.2	10.07.2015	The description of new algorithm of maximum size of cyclic signal calculation for E502 having firmware ARM 1.0.3 and above has been added
1.1.3	28.07.2015	The description of functions X502_SetExtRefFreqValue() and X502_GetRefFreqValue() has been supplemented
1.1.4	29.06.2016	It is specified that the setting up of the output frequency is available in L502 beginning with version 0.5 of FPGA firmware. The

		recommendation on preliminary set of initial values asynchronously in case of synchronous output to DAC. The description of functions X502_CheckFeature() and X502_OutGetStatusFlags() has been supplemented.
1.1.5	03.08.2016	The description of the library application in Visual Basic 6 has been added
1.1.6	23.08.2016	The reference to updated general programmer low-level description for L502 and E502 has been changed in introduction
1.1.7	16.11.2016	The description of functions X502_CalcAdcFreq(), X502_CalcDinFreq(), X502_CalcOutFreq() has been added

Contents

Chapter 1

What this document is about

This document is mainly intended for the programmers who are going to code for operation with L502 and E502 modules using library provided by L-Card Company.

The issue of library connection to the user's project is under consideration in this document, detailed description of interface functions provided by the library and basic approaches to use these functions are given herein.

The library itself is codded in *C* and all noticed functions and types as well as examples given in this document are provided in *C* language; but all references to other programming languages are only wrappers over the *C* library and all functions, types and parameters keep their meanings for all other programming languages. Therefore this document shall be read by users who coding in other programming languages. Moreover, this documents provides the description of differences and basic principles of libraries application in other supported programming languages. Examples for other languages can be installed together with “L-Card L502/E502 SDK”.

Any issues related to module characteristics and signals connection are not in the scope of this document and the operating principles of the module itself are touched upon in general only. These issues are considered in “User Manual L-502” and “User Manual E-502”, which should be reviewed before reading this document.

The present document does not consider the task of coding own firmware for module signaling processor and module handling without application of library provided by "L-Card". These issues are considered in "Programmer Low-level Description".

Chapter 2

Library installation and connection to the project

To code own software operating with L502 and E502 modules it is required to perform the following:

1. Install driver for modules:
    - To work with modules connected via PCI Express interface it is required to use special driver provided by "L-Card" company.
    - Library libusb-1.0 is used to work with modules connected via USB interface. For Windows the library itself is included in standard library e502api.dll and standard driver WinUSB is used as driver. For Linux OS it is necessary to install library libusb-1.0, no special drivers are required in this case.
    - No special drivers are required to work with modules connected via Ethernet interface (TCP/IP). In order to have possibility to search modules in local network there shall be installed corresponding service (see Chapter Detection of modules in local network).
2. Install required dynamic libraries (.dll for Windows or .so for Linux) to directory present in corresponding variable of environment or to directory with the project. Dynamic library is required when writing programs in any programming language because all references to languages operate through the stated libraries. There are three libraries provided:
    - x502api — has general functions for both modules. Shall be included in any project operating with one of modules excluding the projects written only for L502 before x502api library appearance which can use only l502api
    - l502api — has specific functions for L502 module as well as functions left for compatibility with projects written before x502api appearance.
    - e502api — has specific functions for E502 module.
3. Connect the library to the project.

For Windows OS there is “L-Card L502/E502 SDK” installer provided which automatically installs all required drivers, dynamic libraries into system directory as well as all files required for library connection to the project and examples into specified directory. Further in this Chapter SDK_DIR will be defined as the directory specified during “L-Card L502/E502 SDK” installation.

Installation for Linux OS is described in Section Installation of library and driver for Linux OS.

Connection to the project depends on the applicable language and environment of programming.

2.1 Connection of the library when writing program in C/C++.

The following shall be performed during the connection when writing in C/C++:

1. Header file “l502api.h” and/or “e502api.h” shall be included in the project and add in project the SDK_DIR/include directory to the paths for header files.
2. Linker file of required libraries shall be added to the project for applicable compiler:
   - Microsoft Visual C++ : SDK_DIR/lib/msvc
   - Microsoft Visual C++ 64-bit compiler (detailed information on 64-bit version is given below): SDK_DIR/lib/msvc64
   - Borland C++/Borland C++ Builder: SDK_DIR/lib/borland
   - Borland C++/Borland C++ Builder 64-bit compiler: SDK_DIR/lib/borland64
   - MinGW : SDK_DIR/lib/mingw
   - MinGW 64-bit compiler: SDK_DIR/lib/mingw64

   Examples of programs in *C* are given in SDK_DIR/examples/c. Specific examples for Borland C++ Builder- in SDK_DIR/examples/CppBuilder.

2.2 Library application in the project in Delphi

To write programs in *Delphi* using library for working with L502 and E502 modules it is required to include in program project file SDK_DIR/pas/x502api.pas as well as file SDK_DIR/pas/l502api.pas and/or SDK_DIR/pas/e502api.pas which represent wrapper over libraries in *C*. In project files using types and functions of this document it is required to connect x502api module as well as l502api and/or e502api by means of uses x502api; uses l502api; and uses e502api; accordingly. In this case the same files are used for 64-bit compiler as for 32-bit one (see 64-bit version of library).

It should be noticed that file SDK_DIR/pas/lpcieapi.pas is not used anymore as compared to versions before E502 support. Since this file shall not be applied in project directly, it is not included in new versions of library.

All functions, types and constants of library are represented in *Delphi* as one-to-one excluding the following moments:

- all strings (serial numbers, strings with description of errors codes) are converted by the wrap into string type which is normally used for representation of strings in *Delphi* (it should be kept in mind that this type is a uni-code string in latest environment versions). The exception is structure t_x502_info with information on module where strings is presented by array AnsiChar of fixed lenght.
- all functions operating with arrays are taken as open array parameter (open array parameter), this means that these functions can be transfered as static array as well as dynamic (pre-set of its length using SetLength()). Whereby, since Delphi arrays contains the length, the functions L502_GetSerialList() and E502_UsbGetSerialList() as well as L502_GetDevRecordsList() and E502_UsbGetDevRecordsList() it is not required to transfer the array size separately. However, in functions for working with data (for example,

X502_Recv()) the length is transfered in the same manner as in *C* functions in order to use not the entire array for reception. In such case there is additionally check if the length transfered by separate parameter does not exceed the real length of array. In case of excess the error X502_ERR_INSUFFICIENT_ARRAY_SIZE will be returned.

Example of program in *Delphi* is given in SDK_DIR/examples/Delphi.

2.3 Library application in the project on C#

Special library-wrap lpcieNet.dll is implemented to write programs working with L502 and E502 modules in *C#* language (or any other supporting NetFramework). It applies the above mentioned libraries in *C* where the entire logic of working with device is implemented. The installer allows to install lpcieNet.dll to system cache (GAC), this allows not to copy library together with your project. Nevertheless, unfortunately, Visual Studio does not allow to add links from system cache to project and you will have to make the link/reference itself to local copy (it is not needed to be distributed with project). The library in cache has an advantage in comparison with local one and it is always applied if it has been installed.

To use this library it is enough to add the link to lpcieNet.dll in project and connect the required name spaces in sources:

```
using x502api;
using lpcieapi;
```

The older version of all definitions in l502api name spaces with older version of L502 class is left for compatibility with programs written before including of L502 support .

The following changes have been made as compared to functions of *C* language in *C#* wrap:

- Since *C#* is object-oriented language, the special L502 and E502 classes inherited from general X502 class have been created to control modules.
- All functions taking module handle as the first parameter are implemented by method of classes L502, E502 or X502, in such cases prefix L502_, E502_ or X502_ is not used. For example, hnd.Open(serial) is used instead of L502_Open(hnd, serial). Separate class E502.EthConfig is applied for network configuration of E502 module.
- Functions X502_Create() and X502_Free() called in constructor and destructor of X502 class by separate functions are not implemented. The same is about X502_FreeDevRecordList() and X502.DevRec class as well as about functions E502_EthConfigCreate() and E502_EthConfigFree() and E502.EthConfig class.
- The static method X502.Create(devname) is implemented for creation of required object of device (L502 or E502) due to module name.
- Functions which do not require module instance (do not get the handle as the first parameter) are implemented as static functions of corresponding classes. For example, X502_GetErrorString(err) is implemented as X502.GetErrorString(err).
- Functions of Get/Set type which get module handle and one parameter are implemented as properties (properties). For example, hnd.LChannelCount = value is used instead of

X502_SetLChannelCount(hnd, value). However, you should be attentive because the incorrect set value causes exception X502.Exception.

- Constants are declared within the classes and without the prefix L502_, E502_ or X502_.
- The enumerations are also declared as enumerations within the class and without prefix X502_ENUMERATION. For example, not X502_SYNC_INTERNAL but X502.Sync.INTERNAL.
- Errors codes are given in enumeration ERR in lpcie class because it is planned to use general error codes for further modules as well.
- All functions using strings in C in the form of char * apply strings of String type in wrap.
- Functions L502.GetSerialList(), E502.UsbGetSerialList() as well as functions L502.GetDevRecordsList() and E502.UsbGetDevRecordsList() return the dynamic array of strings/records about the device (not filling the transfered one) created within the function which has already had the length. That's why additional parameter of array size is not required.
- As well as in Delphi when working with data arrays the length is transfered as an additional parameter because less data can be received than in dedicated array.
- Functions taking indexes in *C* get the parameters with specifiers out or ref depending on whether a variable should be initialized before calling function or it is output parameter.

Example of program in *C#* is given in SDK_DIR/examples/cs.

2.4 Library application in the project LabView

You can control L502 and E502 modules from *LabView* using the fact that *LabView* supports the controlled NetFramework libraries. Consequently, you have an access to all functions which the wrap *C#* lpcieNet.dll implements, i.e. all available functions of the library taking into account the differences described in previous section).

As compared to Visual Studio LabView automatically catches up .Net libraries from system cache (GAC) and you can make reference to it and not store the local copy together with program.

In order to work with classes .Net the LabView has special panel Connectivity -> .Net .

You need to use the following blocks:

- Constructor Node - creates an object. Shall be created for each L502 or E502 module with which you will work. During creation LabView will suggest to select library and class (lpcieNet.dll and L502 or E502 from x502api space should be selected). One of the outputs of this block is a reference to an object that is used as an input for the rest blocks to work with the module. The object of logic channel with all settings is created using the constructor. The alternative of module object creation is application of X502.Create method which creates the required object due to module name.
- Close Reference - closes and deletes the object. It must be called for each created object when the operation is completed.

- Invoke Node - function (class method) call. When working with an object a reference and input parameters are sent to the input, and output parameters and an updated reference (which must be used for blocks that will be called after the current one) are sent to the output. The input reference/link determines the methods of which object are used (after the entry link establishment the class name appears on the top string and when clicking the second one its method will be suggested for selection). For functions which do not work with a particular object (GetErrorString, GetSerialList, etc.- these functions are static and marked with [S] at the beginning when being selected), there is no need to provide a reference to the input. However, they still belong to the class that must be selected by clicking the right button on the block and then Select Class/ .Net.

- Property Node- used for installation and getting the properties. Part of parameters (logic table, synchronization mode) is installed through the properties and information on module (in the form of class, each field is also a separate property) can be received. Several properties can be set in one block by expanding it downward. Also, using properties, the user can specify constants from enumerations (which can be clearer than simply giving the numbers to input). In this case, you must select the enumeration class and each value will have its own property.

Peculiarity of arrays transfer as output parameters should be noticed. For effectiveness reasons, the library functions do not allocate data arrays within themselves but use the transferred arrays to store the results. Therefore, such parameters are at the same time both input and output parameters in *LabView* . An array of a size sufficient to store the results (while the data itself does not matter) must be sent to input and the same array will be returned as output parameter already containing the results of the function execution. Examples of such parameters can be parameter of buf function X502_Recv(), parameters adc_data and din_data of function X502_ProcessData() as well as parameter out_buf of function X502_PrepareData().

Examples of programs in *LabView* are given in SDK_DIR/examples/LabView.

2.5 Library application in Visual Basic 6

For operation with modules from the program in *Visual Basic 6* it is required to add to project th modules files x502api.bas, e502api.bas and l502api.bas that can be taken from the example SDK_DIR/examples/vb6/x502_general. Files have declarations of all types, constants and functions. Parameters of functions are the same as of functions in *C* language, excluding the following features:

- since Visual Basic does not used pointers the type Long is used for all handles

- Functions L502_GetSerialList(), E502_UsbGetSerialList() as well as functions L502_GetDevRecordsList() andE502_UsbGetDevRecordsList() accept the dynamic array as parameter and change its size in accordance with number of detected elements. Hence the additional explicit parameters specifying array size to input and number of detected elements is not required.

- Strings in functions are converted into strings of Visual Basic automatically. Strings in structures are represented as bit array. To convert them into String you can use function X502_StrConvert()

- To release resources of one write/record there is additional function X502_FreeDevRecord() calling X502_FreeDevRecordList() for one element in order not to convert the element into array manually

It should be noticed that during debugging from environment *Visual Basic 6* in case of uncompleted program shutdown if at the moment of shutdown there were opened connections with modules these connections can stay opened till the environment reset. Hence, since the number of simultaneous connections is limited the module can be invisible in the list of detected devices or it can not be connected to till the environment reset.

2.6 64-bit library version

In 64-bit Windows version, programs can be executed both compiled by a 32-bit and a 64-bit compiler, therefore, most of programs for Windows exist only in 32-bit version. The 64-bit compiler is generally used for programs that work with large bulk of data since this allows process to have more than 4 GB virtual space.

For 64-bit Windows the "L-Card L502/E502 SDK" installer puts 32-bit version of libraries as well as 64-bit ones into corresponding system directory. In this case, Windows/system32 directory refers to one of these directories, depending on the bit depth of the application itself which refers to the specified path. For 32-bit application, 32-bit libraries are stored in Windows/system32, 64-bit libraries in Windows/Sysnative, and for 64-bit application
64-bit libraries are stored in Windows/system32, and Windows/Sysnative does not exist. At the same time, 32-bit libraries are always stored in Windows/SysWOW64 which always exists irrespectively of the bit depth of the application.

When the application is downloaded, if system libraries are used they are searched for using paths from the PATH environment variable, among which there is Windows/system32. Since this directory refers to different locations depending on the bit depth of the application being launched, the library of the required bit depth is selected from Windows/system32 automatically. If the libraries are distributed with the program, it must be ensured that the bit depth of the assembled application and the libraries in the same directory be the same.

The only difference when writing programs in *C or C++* is the need to attach a lib-file in accordance with the bit depth of the compiler used.

For programs in *Delphi* , you only need to specify for which platform the project will be assembled (win32 or win64), and the assembled program will use the library of that bit depth for which the program was compiled.

Programs in *C#* (or any other using NetFramework) are compiled into machine code when executed. Once created, the program can be executed both in 32-bit version and 64-bit version of the NetFramework virtual machine (in the project it can be specified explicitly for what bit depth of NetFramework the program is intended). Therefore, the same program in 32-bit Windows version will be executed using 32-bit version of the libraries, and in 64-bit version using 64-bit library version. For .Net library, the bit depth is determined by the bit depth of the application that uses the library.

Accordingly, in a *LabView* project using the .Net library the bit depth of the library used is determined by the bit depth of the used *LabView* environment.

2.7 Installation of library and driver for Linux OS

To install driver and library for Linux OS there are two options:

- Use the ready assembled packages provided by "L Card". This is recommended way for distributions for which the assembled packages are provided. List of supported distributions can be found in document "Using external L-Card repositories for Linux distributions"
- Download the initial codes "L-Card L502/E502 SDK" and assemble them (see details in the following section).

For examples of working with L502 and E502 modules in *C* for Linux you can download the archive with SDK sources and see the examples in api/x502api/examples/msvc. Despite the name they can be assembled GCC under Linux OS. For each example there is makefile (with comments) and file CMakeList.txt for those who prefer the assembly using cmake.

Information on connection of external repository, installation of assembled packages as well as on the advantages of this installation method is given in document "Using external L-Card repositories for Linux distributions". The list of the packages themselves used when working with L502 and E502 modules with specified dependences is given here. When connecting external repository the dependences are permitted automatically (except the package lpcie-dkms, this is described below). In case of manual installation of packages without connection of external repository the dependences should be considered when installing the packages (for example, the libraries should be placed as follows: first libx502api1, then libl502api1 and libe502api1, and only after, if required, libx502api1-dev or libx502api1-devel).

The following packages are used to work with modules L502 and E502:

- libx502api1-dev or libx502api1-devel — Package with files for developer: header files and references/links to library of required version. Required when writing you own programs using the described in this document libraries (depends on libx502api1, libl502api1 and libe502api1, due to it the libraries are placed automatically when installing developer files)
- libx502api1, libl502api1 and libe502api1 — Packages with libraries of required version. If you distribute your program, it is enough to include into dependences only the package corresponding to used libraries (with no package containing files for developer), this is performed automatically when creating rpm and deb packages. Package libx502api1 has the library of general functions which is used in libl502api1 and libe502api1that's why the latest depend on the first. Package libe502api1 also depends on the package with library libusb-1.0 for this distribution in order to have it installed automatically and places algorithms udev for provision of access to device E502 connected via USB.
- lpcie-dkms — Package containing sources of driver (core module) to work with modules via interface PCI-Express (L502) using the external modules build system dkms (the details are given below).
- lxfw-update — Utility for updating of FPGA firmware of modules L502 and E502. The package includes the latest version of firmware and scripts l502-fpga-update-all.sh and e502-fpga-update-all.sh for updating of firmware of all detected devices L502 or E502 accordingly.

Since the driver shall be built for certain core version (the core can be updated in one of versions of distribution or even different variants of core can be used) the driver can not be distributed in already built/assembled form. If build is carried out directly when installing the package. In this case it is required to pre-set the package with header files of the current core (usually in packages having names linux-headers or kernel-devel). For some distributions there can be several variants of core (and accordingly several packages), moreover you can use your core. Exactly due to this reason the package is not set by dependences for lpcie-dkms as compared to other dependences. Determine the current version of core by command: uname -r. Make sure that required files are installed by means of checking of availability of files in directory /lib/modules/‘uname -r‘/build (usually it is reference to core header in /usr/src/linux-<version> or /usr/src/kernels/<version>).

If headers of current core are installed, the build of drivers is performed using DKMS during installation of lpcie-dkms package (package dkms included in dependences lpcie-dkms as well as make and gcc required for build). DKMS is quite wide-spread system of build and control of external core modules (it is located in main repository of most distributions Linux, but it is not present in OpenSuse and package dkms covers it through Open Build System). DKMS allows:

- monitor centrally which exterior core modules and which their versions and for which core versions are installed (dkms status)
- always save sources of driver of different versions in centralized place (/usr/src/lpcie-<version>)
- allows automatically re-build driver when shifting to new core
- allows to delete driver any moment or any version of it and all related files (dkms remove -m lpcie -v <version> –all)

Thus, although the package has driver sources and not the built driver, the installation is not much different from installation of other packages except the additional installation of core header files is required and installation of package takes significant time for build.

When installing new core module will be re-build for it automatically or when installing package, or when entering system with new core for the first time.

2.8 SDK initial codes

Initial codes of all constituents of SDK are opened. User has the access for reading to repository of versions control system Mercurial, located on the address https://bitbucket.org/lcard/lpcie_sdk. Detailed information on application of opened repositories of "L Card" initial codes on bitbucket.org you can find in document "Using open L-Card source repositories" at bitbucket.org.

You can also download archive lpcie_sdk_src.zip with all initial codes from attached files of repository project.

Instructions on build is given in source file INSTALL.txt.

Chapter 3

General approach to working with the library

3.1 Differences in handling L502 and E502 modules

3.1.1 Differences in modules capabilities

Functional capabilities of modules L502 and E502 are much similar but there are significant differences.

1. The main difference is related to used interfaces -PCI-Express for L502 and USB or Ethernet for E502. Due to this the procedure of connection with device is a bit different. Moreover, operation through Ethernet interface requires the setting of additional parameters.

2. E502 has additional controller ARM Cortex-M4 used for interfaces logic implementation (USB/Ethernet). This controller has its own firmware which can be updated and where additional capabilities can be implemented (first of all when working through Ethernet). It is recommended always use the latest version of firmware which can be downloaded on https://bitbucket.org/lcard/e502_m4/downloads and update using program L-Card Measurement Studio. Moreover, it is possible, in theory, to create his own firmware by the user for this controller because "L Card" provides the initial codes of this firmware (project of firmware can be found on https://bitbucket.org/lcard/e502_m4), but "L Card" does not provide any manual to this firmware. It is also possible to consider the proposals on order for improvement of firmware ARM Cortex-M4 for required user tasks.

3. There is a limit of data transfer speed for E502. L502 allows to perform simultaneous input and output of all data from analog channels and digital lines at maximum speed but module E502 has limits which depend on used interface:

    - when working through USB the total maximum speed of transfer is about 5 mln samples per second. In other words it is allowed to use, for example, input from ADC and digital lines at 2 MHz and in this case output is only for one channel of DAC/DOUT at 1 MHz or for two at 500 KHz. If there is input only from ADC at 2 MHz, it is possible to use all three channels of output at 1 MHz and
      etc. In such case this limit will be based on the limit of interface speed between ARM Cortex-M4 controller and FPGA an not the USB interface itself.
    - when working through Ethernet (TCP/IP) maximum speed is already limited by speed of transfer via network through protocol TCP. When module operates only to input the speed is limited by 2.5 mln of samples per second. Speed to output and its influence on input speed will be specified further. For output it is recommended, if possible, to use cyclic output mode. The loading of the network itself should be

considered when transfer data via Ethernet, because it can influence greatly on maximum speed of transfer.

4. Due to limit of transfer speed in E502 there is a capability to set the common divider for output frequency which can be specified using X502_SetOutFreqDivider() or X502_SetOutFreq(). The same capability is available in L502 in FPGA firmware version 0.5, thus to use it you may need to update the firmware.

5. In case of cyclic output in L502 the buffer is placed in driver on PC, in E502 the storage of cyclic buffer is implemented inside the memory of module ARM Cortex-M4 controller and this allows to avoid loading of interface and PC during the operation, but it causes the limit of buffer size. In other words, when working through the network the cyclic output does not influence on the interface limit, but the limit of total speed 5 mln samples per second is kept for module. For version of ARM firmware less than 1.0.3 the output buffer of 3 mln samples (total for all channels) is divided into 2 equal parts (one is used for output of signal, another for loading of the following in order to have possibility to change the signal without stop of previous one), that's why one signal is limited in 1.5 mln of samples. Beginning with 1.0.3 buffer can be divided randomly (depending on loading signal size), that's why signal size is limited with value — 3 mln samples minus size of now output signal (0- if there is no generation of previous ongoing during loading). Thus, if capability of changing signal in process (i. e. always after X502_OutCycleSetup() comes X502_OutCycleStop() or X502_StreamsStop() till the next signal loading), it is possible to use all 3 mln samples for one signal. It also should be considered that signal transfer to ARM controller takes time and if it is required to output the signal at the same time with start of collection it is required to wait for signal loading, this can be done, for example, using flag X502_OUT_CYCLE_FLAGS_WAIT_DONE in X502_OutCycleSetup().

6. When working via Ethernet interface function X502_GetSendReadyCount() is not implemented and function X502_GetRecvReadyCount() works only in Windows OS.

7. When reading values of digital inputs for E502 the high lines DI14, DI15, DI16 are united with lines of synchronization DI_SYN2, CONV_IN and START_IN accordingly. In this case since both modules have lines 17 and 18 united with DI_SYN1 and DI_SYN2, value of 18th and 14th lines for E502 are always similar.

8. Setting of pull-ups for digital inputs in E502 differ from L502 (see description of type t_x502_pullups)
9. In E502 another DAC microcircuit chip is used that is also planned to be used in further revisions of L502 module.

3.1.2 General and specific functions for working with module

Due to fact that greater amount of functionality of L502 and E502 modules is similar, most of functions are implemented by general for both modules. All general functions are implemented in library x502api. In this case names of functions and constants starts with X502_, and types with t_x502_. Accordingly, for working with both modules the same type of module handle is used t_x502_hnd.

The main difference during operation depending on interface of connection with module is the procedure of connection establishment. These functions are implemented in certain libraries l502api and e502api for modules L502 and E502 accordingly. When setting connection

all required information about how to work with module via needed interface is saved inside opaque module handle, and user after opening of connection can work with module both using the same general functions from x502api regardless of module type and used interface.

Separate group of specific functions includes functions for setting of Ethernet interface implemented only in e502api and described in section Functions for setting of network parameters of E502 module.

3.1.3 Compatibility of projects developed before the implementation of library x502api

For implementation of complete compatibility with projects operating only with module L502 and ones developed before implementation of support of operation with module E502 (implemented since 1.1.0) and creation of common library, functions from this library of previous versions (1.0.x) are implemented in l502api. In this case these declarations of these functions and corresponding types are placed in separate file “l502api_compat.h” actuated from “l502api.h” to keep compatibility. These types are specified through common types from “x502api.h” and functions actually only call similar general functions from x502api. Accordingly, projects developed for previous version with no consideration of common functions shall be collected correctly in new version of libraries as well. The main difference that shall be considered is that if these projects are distributed with new version l502api, it is required to distribute the library x502api as well because its functions are used by functions of new version l502api.

Since these functions and types are completely similar the majority of generalized functions (except prefixes in names), they are not given in the present document.

3.2 General algorithm for module handling.

This section describes typical sequence of functions call for working with modules L502 and E502. Each step is described in details in the following chapters.

The typical sequence of calls is as follows:

1. When working with modules via PCI-Express or USB interfaces it is possible to get list of serial numbers using functions L502_GetSerialList() and E502_UsbGetSerialList(), accordingly, or get list of records about modules using L502_GetDevRecordsList() and E502_UsbGetDevRecordsList(). When working via Ethernet you can use functions of devices detection in local network described in chapter Detection of modules in local network or if device IP-address is known come to point 2.

2. If system has required module, create module handle using X502_Create().

3. Establish the connection with the module. When using records about devices opening is always performed using X502_OpenByDevRecord(). When using serial number L502_Open() or E502_OpenUsb() are applied for L502 and E502 connected via USB accordingly. To set connection with module via Ethernet using IP-address it is required to apply function E502_OpenByIpAddr().

4. If required, in order to get additional information on device use X502_GetDevInfo() (in particular for checking of BlackFin signaling processor availability).

5. If signaling processor is available (and if you want to work using it), you can download firmware of signaling processor using X502_BfLoadFirmware().
6. Setting of module parameters using set of functions for module setting change (functions names begins with X502_Set)
7. Transfer of specified parameters to module using X502_Configure().
8. Working with module in synchronous and/or asynchronous mode (described in following sub-sections).
9. Closing module X502_Close().
10. Release of module handle by function X502_Free().

3.2.1 Module handling during synchronous input

Typical operation with module during synchronous input consists of the following steps:

1. Permission of required synchronous streams (ADC and/or digital data) using X502_StreamsEnable().
2. Start of synchronous streams X502_StreamsStart().
3. Reading the data received from module using X502_Recv().
4. Processing of read data using X502_ProcessData(), X502_ProcessAdcData() or X502_ProcessDataWithUserExt().
5. If the receiving and processing of the following block is required, proceed to step 3.
6. Stop of synchronous streams using X502_StreamsStop().

3.2.2 Module handling during synchronous stream output

Typical operation with module during synchronous output consists of the following steps:

1. Set-up of initial values for DAC using asynchronous output X502_AsyncOutDac().
2. Permission of required synchronous streams (DAC channels, digital outputs) using X502_StreamsEnable().
3. Start of preliminary loading of data for output using X502_PreloadStart().
4. Preparation of data block for writing using X502_PrepareData().
5. Writing of prepared block to module using X502_Send().
6. If required repeat steps 4. and 5. as many times as needed. In this case the general size of pre-loaded data shall not exceed the buffer size (by default 9 MWords)
7. Start of synchronous stream by calling X502_StreamsStart().
8. Each time when it is required to load additionally new data to buffer, execute steps 4. and 5.

9. When operation is completed stop the synchronous streams using X502_StreamsStop().

3.2.3 Module handling during cyclic output

To set the cyclic signal without additional load the typical sequence is as follows:

1. Set-up of initial values for DAC using asynchronous output X502_AsyncOutDac().
2. Permission of required synchronous streams (DAC channels, digital outputs) using X502_StreamsEnable().
3. Allocation of cyclic buffer of specified size using X502_OutCycleLoadStart().
4. Loading of data with specified size for cyclic output using one or several calls of X502_Send().
5. Make the loaded signal active using X502_OutCycleSetup() with flag X502_OUT_CYCLE_FLAGS_WAIT_DONE.
6. Start the synchronous input-output through X502_StreamsStart().
7. If it is required to connect new signal perform steps 3.-5.
8. When operation is completed stop the synchronous input-output using X502_StreamsStop() or only cyclic output through X502_OutCycleStop().

3.2.4 Module handling during asynchronous input-output

Typical operation during asynchronous input-output consists of calling of one of functions:

- X502_AsyncInDig() - asynchronous input of digital lines values
- X502_AsyncOutDig() - asynchronous output of values to digital lines
- X502_AsyncOutDac() - asynchronous output of values to one of DAC channels
- X502_AsyncGetAdcFrame() - asynchronous reception of one ADC frame

3.3 Creation and release of module handle.

All work with modules L502 and E502 is performed through module handle of t_x502_hnd type. Module handle is opaque pointer to structure that keeps all information on module and status of connection with it. User does not have direct access to fields of structure and all activities with module are executed by means of corresponding functions of library that take module handle as first parameter.

Before trial of establishment of connection with module it is required to create handle by calling function X502_Create() that allocates memory for structure, initialises its fields by default values and returns pointer to it- module handle.

As soon as work with module is completed the memory allocated by function X502_Create() shall be released by calling X502_Free(). After release the handle can already be used.

3.4 Opening of connection with module

3.4.1 Setting the connection with E502 module through Ethernet interface

PCI-Express

To start working with module it is required to set connection with it using function L502_Open(). Modules serial number are used to differentiate them.

You can get list of serial numbers of all found modules L502 using function L502_GetSerialList(). This function receives flat array where found serial numbers will be saved and maximum amount of serial numbers that can be saved in transfered array.

In simplest case it is possible to set maximum value of modules and serial numbers to use statically allocated array:

```
#define MAX_MODULES_CNT 16

char serial_list[MAX_MODULES_CNT][X502_SERIAL_SIZE]; int32_t get_list_res =
L502_GetSerialList(serial_list, MAX_MODULES_CNT, 0, NULL);
if (get_list_res<0) {
      /* Failed to get the list of serial numbers */
} else if (get_list_res==0) {
       /* No module detected */
} else {
       /* get_list_res modules detected */
}
```

In general case for operation with random maximum amount of modules you can use third parameter of function to get number of found modules in the system. In this case zero/null pointer can be sent as array of serial numbers and specify zero size of array. After this you can dynamically allocate array for received amount of serial numbers and again call L502_GetSerialList() for receiving serial numbers of all L502 modules:

```
uint32_t dev_cnt; int32_t res;

/* Getting the number of modules in the system */ res =
L502_GetSerialList(NULL, 0, 0, &dev_cnt);
if (res<0) {
       /* Failed to get the list of serial numbers */
} else if (dev_cnt==0) {
       /* No module detected */
} else {
      /* Allocate flat array for dev_cnt serial numbers of size dev_cnt*L502_SERIAL_SIZE */
      t_x502_serial_list serial_list= (t_x502_serial_list)
                malloc(dev_cnt*X502_SERIAL_SIZE);
      if (serial_list == NULL) {
```

```
            /* Error of memory allocation */
        } else {
            res = L502_GetSerialList(serial_list, dev_cnt, 0,
                                                    NULL);
            if (res>0) {
                /* res serial number received */
            }
            /* Release the dedicated array for serial numbers */ free(serial_list);
        }
}
```

It should be noticed that only on connection can be established with one module simultaneously. If you try to open module that has already connected through another handle (maybe in other program) L502_Open() returns X502_ERR_DEVICE_ACCESS_DENIED. In this case L502_GetSerialList() by default returns list of all serial numbers of module including those already connected with. If it is required to get list of only those devices not connected ye, flag X502_GETDEVS_FLAGS_ONLY_NOT_OPENED can be sent to L502_GetSerialList() .

If zero pointer or empty string is sent as serial number to L502_Open() , there will be trial of opening first module that is successfully connected to. If no module is connected, error received upon trial of latest module opening will be returned. In other words, in case if two modules L502 are available in system, first call of L502_Open() establishes connection with first L502 module, second call- with the second one, and third will return access error X502_ERR_DEVICE_ACCESS_DENIED.

3.4.2 Setting the connection with E502 module through USB interface

When working via USB the algorithm of connection establishment is similar to module L502 opening, the only difference is that to get list of serial numbers function E502_UsbGetSerialList() is used and to open module E502 due to serial number E502_OpenUsb() is used.

3.4.3 Setting the connection with E502 module through Ethernet interface

Set the connection with module via Ethernet is possible due to explicitly specified IP-address of device as well as using functions of devices detection in local network, detailed information is given in chapter Modules detection in local network.

When setting the connection through IP-address it is enough to call function E502_OpenByIpAddr().

It should be noticed that as compared to other interfaces for correct operation via Ethernet there should be required parameters set, chapter Features of operation via Ethernet interface and setting of network parameters has information on it.

3.4.4 Setting the connection with modules using the records about device

One of disadvantages of opening due to serial number function is that at the moment of opening it is required to know device having this number is connected via which interface to select corresponding function for device opening. To avoid this the special type t_x502_devrec is implemented corresponding to record about found device. This type has information on found device (name, serial number, interface, flags with supported capabilities, etc.) and all required information about how to establish connection and work with corresponding device. Accordingly, only functions of receiving of records about the device depend on device and interface and connection is established by general function X502_OpenByDevRecord().

It allows to get all required records at separate stage and save them in general array and later perform opening of connection on required records not determining separately what is the device and via which interface it is connected.

Feature is necessity of user to free the memory allocated at the stage of initialization of record about the device. Cleaning of memory is executed using X502_FreeDevRecordList(). Record about the device is used only inside X502_OpenByDevRecord() or, if required, can be released right after call of this function if list of records is not needed anymore. It also should be considered that before sending pre-initialized record for reception or initialization of new record (for example, for list update) it should be first cleaned to avoid memory leaks.

The following functions for records initialization are available:

- L502_GetDevRecordsList() initializes records corresponding to connected L502 modules via interface PCI-Express. As per parameters it is similar to L502_GetSerialList() considering the necessity of records release.
- E502_UsbGetDevRecordsList() initializes records corresponding to connected L502 modules via interface As per parameters it is similar to E502_UsbGetSerialList() considering the necessity of records release.
- E502_MakeDevRecordByIpAddr() initializes record for setting the connection with E502 module with specified address via Ethernet interface.
- E502_MakeDevRecordByEthSvc() initializes record for setting the connection with E502 module corresponding automatically detected service via Ethernet interface

Example creating records for all found modules connected via interfaces USB and PCI-Express is given below: Moreover, it is supposed that array ip_addr_list has ip_cnt of addresses for which records are also created and definition TCP_CONNECTION_TOUT sets time-out in ms for connection via network interface. Then the choice of required device is provided after that the connection is set with it and all records are cleaned after.

```
uint32_t ip_addr_list[] = { .... } ; /* list of ip-addresses of devices */
uint32_t ip_cnt = ... ; /* size of this list */

uint32_t pci_devcnt = 0;
uint32_t usb_devcnt = 0;
int32_t fnd_devcnt = 0; /* total amount of found records */
t_x502_devrec *devrec_list = NULL; /* list of records about devices */
t_x502_hnd hnd = NULL; /* handle of opened device */

/* getting amount of connected devices via interfaces PCI and USB */
```

```
L502_GetDevRecordsList(NULL, 0, 0, &pci_devcnt);
E502_UsbGetDevRecordsList(NULL, 0, 0, &usb_devcnt);

if ((pci_devcnt+usb_devcnt + ip_cnt) != 0) {
    /* allocating memory for array to save found amount of records */
    devrec_list = malloc((pci_devcnt + usb_devcnt + ip_cnt) *
    sizeof(t_x502_devrec));

    if (devrec_list != NULL) { unsigned i;
        /* getting records about L502 modules but not more than pci_devcnt */
        if (pci_devcnt!=0) { int32_t res =
        L502_GetDevRecordsList(&devrec_list[fnd_devcnt], pci_devcnt, 0, NULL);
            if (res >= 0) {
                fnd_devcnt += res;
            }
        }
        /* adding records about E502modules connected via USB in the end of array */
        if (usb_devcnt!=0) { int32_t res = E502_UsbGetDevRecordsList(&devrec_list[fnd_devcnt],
        usb_devcnt, 0, NULL);
            if (res >= 0) {
                fnd_devcnt += res;
            }
        }

        /* creating records for transfered array of ip-addresses */
        for (i=0; i < ip_cnt; i++) {
            if (E502_MakeDevRecordByIpAddr(&devrec_list[fnd_devcnt],
                                           ip_addr_list[i], 0,
                                           TCP_CONNECTION_TOUT) == X502_ERR_OK) {
                fnd_devcnt++;
            }
        }
    }
}

if (fnd_devcnt != 0) {
   uint32_t dev_ind;

   /* processing list and selection of required device which index is saved (as an
     example) in dev_ind */ .....
```

```
        if ( <Device is selected> ) hnd =
            X502_Create(); if
            (hnd==NULL) {
                /* Error of module handle creation! */
            } else {
                /* setting connection with module due to record */ int32_t err =
                X502_OpenByDevRecord(hnd, &devrec_list[dev_ind]);
                if (err != X502_ERR_OK) {
                    /* error of connection setting */
                    X502_Free(hnd); hnd = NULL;
                }
            }
        }

        /* releasing resources of valid records from list */
        X502_FreeDevRecordList(devrec_list, fnd_devcnt);
}
/* cleaning memory of array itself */ free(devrec_list);


if (hnd != NULL) {
        /* working with module */

        X502_Close(hnd);
        X502_Free(hnd);
}
```

This example has all resources of records released right after selection of required device and setting the connection with it. If needed, it is possible to use another approach, for example, saving records together with created device handles to have possibility to open, close and open again the device any required moment releasing each record only at the moment of operation completion or receiving new list of devices. In this case it should be considered that copying record about device is possible but X502_FreeDevRecordList shall be called only once per each copy of each record.

It should be noticed that all functions of serial numbers receiving and setting connection with modules from previous sections are actually implemented through functions described in this section and are only wraps created to simplify the procedure of search and setting connection with devices.

3.5 Operating modes with signaling processor and without it

Modules L502 and E502 can operate in two modes:

- In standard mode (X502_MODE_FPGA) all data processing is carried out by hardware in FPGA of module and control by module is performed using writing of values in FPGA registers. In this mode all standard functions of data collection are available but user has

no possibility to extend functional capabilities of module. This mode is available for all modifications of L502 and E502.

- In mode of operation with signaling processor (X502_MODE_DSP) the control of data collection is performed by BlackFin signaling processor and all data streams to input and output go through it. Thus, user can implement additional capabilities by means of creation of BalckFin modified firmware (for example, feedback in real time mode). This mode is available only for modifications L-502-P-G, L-502-P-G-D and E-502-P-EUD. You can find information on availability of signaling processor using program by flag X502_DEVFLAGS_BF_PRESENT in flags of records about device or in flags of information on module that can be get after setting the connection with module via function L502_GetDevInfo().

When feeding power module is always in standard mode without use of signaling processor. For operation of signaling processor it is required to pre-load program (firmware). This can be done from file of ldr format using function X502_BfLoadFirmware(). After this module will be automatically switched into mode with signaling processor.

If required, you can intentionally change operational mode using X502_SetMode(). It can be required, for example, if firmware is loaded to BlackFin via interface JTAG. Moreover, it should be considered that during opening of connection with device the change of mode is not executed. In other words, if one program specified mode X502_MODE_DSP, when opening module from another program this mode is kept. Due to it you may need to explicitly switch module to standard mode using X502_SetMode(). That's why if program does not suppose that module could operate in mode X502_MODE_DSP and perform any functions that need not be interrupted, and work with module "from scratch", it is recommended to set explicitly required operational mode right after setting the connection.

All settings of module and operation with synchronous input-output shall be executed after set up of required mode.

Any moment yo can know the current operational mode using X502_GetMode().

3.6 Setting module configuration

Prior to use module, as a rule, it is required to perform setting of its parameters. First all settings are written in structure fields of module handle using functions beginning with X502_Set that will be described in following sub-sections, after this the set parameters are transfered to module using X502_Configure().

3.6.1 Setting ADC channels poll sequence

Modules L502 and E502 are ADC with sequential channels switching. It means that measurement of several channels is performed sequentially by means of switching of input ADC switch board. As in most "L Card" models the sequence of channels poll is specified using control table of ADC logic channels. Table shall contain in total from one up to X502_LTABLE_MAX_CH_CNT logic channels.

Each logic channel specifies following parameters:

- number of physical channel from which measurement is executed. Number of physical channel is specified counting from 0, in other words 0 means first channel, 1- second, etc. Thus, in differential mode channel number can be fro 0 to15 and in measuring mode with common ground-from 0 to 31.
- mode of ADC measurement from t_x502_lch_mode.
- used range of measurement (from t_x502_adc_range).
- averaging factor due to specified logic channel (see section Averaging factor for logic channel).

Set parameters of logic channel with required number is possible using functions X502_SetLChannel() and amount of logic channels in control table- using X502_SetLChannelCount().

For example, it is required to measure first voltage of input X1 relative to common ground for range +/-10V, then measure value of 16th channel in differential mode (between inputs X16 and Y16) with range +/-1V and after measure voltage between Y1 and common ground (17 channel in mode with common ground) with range +/-0.2V (Purpose of signal connector outputs and connection of signals to module is described in "User Manual"). In this case setting of logic table will be as follows:

```
/* setting 3 logic channels */ int32_t err =
X502_SetLChannelCount(hnd, 3);
if (err == X502_ERR_OK) {
    /* first logic channel corresponds to measurement of 1 channel in relation to common
        ground */
    err = X502_SetLChannel(hnd,0,0,X502_LCH_MODE_COMM, X502_ADC_RANGE_10,0);
}
if (err == X502_ERR_OK) {
    /* second logic channel corresponds to measurement of 16 channel in
        differential mode */
    err = X502_SetLChannel(hnd,1,15,X502_LCH_MODE_DIFF, X502_ADC_RANGE_1, 0);
}
if (err == X502_ERR_OK) {
    /* third logic channel- measurement of 17-th channel in relation to common
        ground */
    err = X502_SetLChannel(hnd,2,16,X502_LCH_MODE_COMM, X502_ADC_RANGE_02,
        0);
}

if (err == X502_ERR_OK) {
    /* establishment of other settings */
}
if (err == X502_ERR_OK) {
    /* sending settings to module */ err =
    X502_Configure(hnd,0);
```

```
}

if (err != X502_ERR_OK) {
    /* error occurred during setting of parameters... */
}
```

After completion of measurement with settings corresponding to last logic channel measurement corresponding to fist logic channel (with zero number) again is following. Sequence of measurements corresponding to one pass of logic table is called frame.

If needed, between completion of measurement corresponding to last logic channel of frame and beginning of measurement corresponding to first logic channel of following frame the inter-frame delay can be set.

3.6.2 Setting frequency of synchronous input/output

All frequencies of stream collection and output of data are based on reference synchronization frequency. Internal or external source of frequency can be used as reference frequency. In first case the reference frequency can be 2 MHz or 1.5 MHz. 2 MHz is used by default. It can be changed using function X502_SetRefFreq().

Signal with random frequency up to 1.5 MHz can be used at external reference frequency. In this case it is required to set value of external reference frequency that will be supplied in order that library functions can optimally select settings of data transfer (if they are not set manually) and functions selecting dividers (see below) operate correctly to have required frequencies of input/output. This value can be set using function X502_SetExtRefFreqValue().

ADC collection frequency is obtained due to division of reference frequency by specified coefficient that can be within the range from 1 to X502_ADC_FREQ_DIV_MAX. Moreover, as it was already stated in previous section, between measurement of last logic channel of one frame and beginning of following frame the inter-frame delay can be added. The inter-frame delay is set as amount of periods of synchronization reference frequency.

ADC collection frequency divider and amount of periods of reference frequency for inter-frame delay can be explicitly set by function X502_SetAdcFreqDivider() and X502_SetAdcInterframeDelay() accordingly. Instead of these functions it is possible to use function X502_SetAdcFreq() for convenience that can be transfered with value of ADC collection frequency and frames frequency in Hertz and value of inter-frame delay in order that received frequencies were the closest to specified ones. In this case function will return actually set values of frequencies.

ADC collection frequency (f_acq) is value opposite to time of one conversion corresponding to one logic channel. Frames frequency (f_frame) is value opposite to time from beginning of measurement of first logic channel of one frame till beginning of measurement of first logic channel of following frame. This frequency corresponds to collection frequency for one logic channel.

The diagram representing how the above mentioned frequencies are determined based on the example of collection upon specified three logic channels is given below.

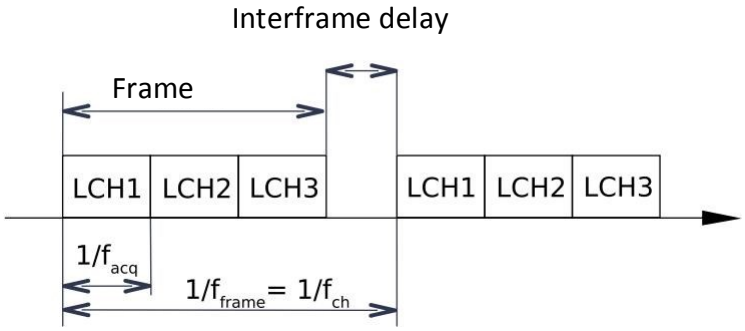

Fig. 3.1: Diagram of ADC collection for three logic channels

If inter-frame delay is not required, it is possible to send zero pointer as the second parameter X502_SetAdcFreq() then always zero inter-frame delay will be used (i. e. measurement of following frame starts right after completion of previous one).

In addition to synchronous input from ADC modules L502 and E502 allow to perform synchronous entry from digital inputs (amount depends on module type, this is described in section Modules capabilities difference). As well as for synchronous ADC data collection the frequency of synchronous digital input is defined as reference frequency divided by coefficient that can be set using X502_SetDinFreqDivider(). It is also possible to call function X502_SetDinFreq() and it will calculate this coefficient to get frequency closest to specified one. For synchronous input of digital lines there is neither logic table (because each time value of all digital inputs is read and sent as one word) nor inter-frame delay- when starting synchronous input all measurements are performed in equal periods of time. In this case frequency for input from digital lines can be different from ADC collection frequency.

Modules L502 and E502 allow to perform synchronous output to two DAC channels in parallel (option) and to digital outputs. In this case maximum output frequency of each channel is two times less than value of reference frequency. For module E502 and L502 having FPGA firmware of version 0.5 or above it is possible to set divider of output frequency (in relation to reference frequency within the range from X502_OUT_FREQ_DIV_MIN to X502_OUT_FREQ_DIV_MAX) or explicitly using function X502_SetOutFreqDivider()or by calling function X502_SetOutFreq() such that it select the divider to have frequency closest to specified one. In this case frequency is set as general for all streams of output.

3.6.3 Averaging factor for logic channel

Actually microcircuit chip of ADC always works at frequency equal to reference synchronization frequency. In case if ADC collection frequency is set less than reference synchronization frequency, per one measurement of logic channel value there is n ADC measurements(n = f_acq/f_ref — relation of specified ADC collection frequency to sampling frequency).

In case when averaging is off the first n-1 measurements are rejected thus increasing the time of signal establishment. If required it is possible to use several last samples (navg) to obtain resulting value. Then resulting value will be the average between navg last measurements, but this accordingly reduces time for signal establishment. Naturally navg is always less or equal to n. Moreover, navg can not exceed maximum value equal to X502_LCH_AVG_SIZE_MAX.

Value of navg is set by last parameter of function X502_SetLChannel(). Value equal to 1 means absence of averaging. Value equal to 0 means that averaging coefficient can be selected at the discretion of library. In the current implementation value 0 is similar to value 1 but it can be changed in further versions.

3.6.4 Setting synchronization modes

Internal frequency of module is used as reference synchronization frequency and start of all synchronous measurements is executed when performing function X502_StreamsStart().

But, if required, it is possible to set the internal source of reference frequency as well as external signal of start of synchronous data collection/output.

For this inputs of digital connector DI_SYN1 and DI_SYN2 can be used (can be used as edge/rise or fall of one of these signals) or synchronization connector can be used for arrangement of synchronous data collection due to principle master-slave.

Selection of external signal for setting reference synchronization frequency is set using X502_SetSyncMode() and start condition using function X502_SetSyncStartMode(). It should be noticed that if external event of start is specified, it is required to call X502_StreamsStart(). so module switches into waiting for this event mode.

Stop of synchronous data collection/output is always performed in program using X502_StreamsStop().

When using synchronization connectors to arrange data collection due to principle master-slave, for master module the internal frequency is source of reference synchronization frequency (mode X502_SYNC_INTERNAL) and each slave module uses reference frequency and/or attribute of collection start from external master, i. e. for each slave module there should be set mode X502_SYNC_EXTERNAL_MASTER.

3.7 Synchronous and asynchronous operating modes.

For modules L502 and E502 the following data to input are available:

- samples from ADC
- values of digital inputs

Module also cna be used for output:

- samples to first DAC channel
- samples to second DAC channel
- values to digital outputs

Thus, there are 2 channels to input and 3 channels to output.

Each of these channels can operate in synchronous mode as well as in asynchronous one. In this case each channel can be set individually, in other words it is possible to perform, for example, asynchronous input of digital lines against the synchronous stream collection from ADC or send to one DAC channel signal in synchronous stream mode while output data to another channel asynchronously. The only exception- it is impossible to perform asynchronous input from ADC while there is synchronous data collection from digital inputs.

3.7.1 Asynchronous operating mode

When supplying power all channels are in asynchronous mode. In asynchronous mode when calling function of asynchronous input/output single-shot input or output of specified information is performed. In this case the delay from function call to the moment of data measurement for input or submission of the specified value at output for out is not defined exactly. The delay between two sequential operations of input/output can not also be defined exactly.

Advantage of asynchronous mode is that it can be applied easily- one call of required function is enough:

- X502_AsyncInDig() - asynchronous input of digital lines values
- X502_AsyncOutDig() - asynchronous output of values to digital lines
- X502_AsyncOutDac() - asynchronous output of values to one of DAC channels

For single-shot data input from ADC function X502_AsyncGetAdcFrame() is used performing input of one ADC data frame. As compared to another functions of asynchronous input-output prior to call this function it is required to perform module setting: it is required to set ADC control table (see Setting ADC channels poll sequence). Measurement of logic channels within the frame is conducted synchronously with specified ADC collection frequency. The input of frames themselves is asynchronous, this means that delay between measurements of frames upon sequential call of X502_AsyncGetAdcFrame() is not defined.

For example, code for execution of single-shot measurement from 7th physical channel in differential mode with range +/-0.5V can be as follows:

```
/* setting 1 logic channel in control table */ int32_t err =
X502_SetLChannelCount(hnd, 1);
if (err == X502_ERR_OK) {
    /* logic channel corresponds to measurement of 7th channel in
        differential mode */
    err = X502_SetLChannel(hnd,0,6,X502_LCH_MODE_DIFF,L502_ADC_RANGE_05,0
        );
}
if (err == X502_ERR_OK) {
    /* sending settings to module */
  err = X502_Configure(hnd,0);
}
if (err == X502_ERR_OK) {
    /* Reading ADC data frame from one of samples */ double val;
    err = X502_AsyncGetAdcFrame(hnd,
            X502_PROC_FLAGS_VOLT, 1000, &val);
    if (err == X502_ERR_OK) {
        /* value val is read correctly */
    }
```

}

3.7.2 Synchronous operating mode

In synchronous mode of data input or output is performed at specified frequency, in other words the time between adjacent measurements or output of adjacent samples is defined. Collection frequencies for each channel are set in relation to common reference synchronization frequency (for details see chapter "Setting frequency of synchronous input/output") and start of synchronous input-output for all channels is executed simultaneously.

To start synchronous mode it is required first permit synchronous mode through required channels using function X502_StreamsEnable() and then actuate synchronous input/output through all permitted channels using X502_StreamsStart().

At synchronous input module performs measurements at specified frequency and transfers data via interface to buffer (for L502 this buffer is in driver and sent by module using BusMaster DMA, and for E502- buffer is allocated by library). Data received by buffer can be read by program using X502_Recv().

Similar for synchronous output module itself as and when required reads data from buffer and submits read samples at specified frequency. Data for driver buffer shall preliminary be written using X502_Send(). In this case if by the moment of output of following sample data did not arrive to driver buffer, the previous value will be submitted.

Only two buffers are allocated in driver or library- one for reception and one for transfer. In other words, values for synchronous input from digital lines and ADC samples are sent by one stream, all data to output are also sent by one stream. Each sample is transfered as 32-bit word containing additional information including attribute which type of data this sample is referred to.

Parsing of received data into ADC samples and values of digital outputs is carried out using X502_ProcessData(). In addition to it this function can also perform conversion of ADC samples into Volts. It should be considered that as compared to some other product of "L Card", application of calibration coefficients is executed by hardware and values already come in form of 24-bit samples with applied coefficients.

In 32-bit word corresponding to ADC sample measurement mode and number of physical channel is transfered additionally. X502_ProcessData() compares these values with those being specified at setting of control table to ensure the correctness of receivable data. In this case X502_ProcessData() waits that it will be used to process all received data.

Data from ADC come in the order as measurements performed, i. e. at first measurements corresponding to all logic channels of first frame, then of the second one, etc. X502_ProcessData() returns converted ADC samples in the same order. In this case X502_ProcessData() can transfer non-integral amount of frames (for example, if synchronous input from digital lines is running, it is difficult to pre-determine how many samples from digital lines and how many samples from ADC are in data block), in this case X502_ProcessData() processes and outputs all samples including samples of non-integral frame and upon its following call checks that ADC samples start with logic channel following the channel last processed before it. Which logic channel is expected to be following for processing can be known using function X502_GetNextExpectedLchNum().

For example, control table of ADC can have 7 logic channels specified. If data block containing 5 ADC samples is received from module (and random amount of values of digital

inputs if synchronous input from digital lines is on) and processed using X502_ProcessData(), X502_ProcessData() will return 5 converted ADC samples corresponding to logic channels with indexes 0, 1, 2, 3, 4. Following logic channel expected for processing- logic channel having index 5, that's why X502_GetNextExpectedLchNum() returns value 5. If the next data block containing 5 following samples is processed, X502_ProcessData() returns samples corresponding to channels having indexes 5, 6, 0, 1, 2. I. e. by calling X502_GetNextExpectedLchNum() it is possible to know to which logic sample will correspond first element of output array when calling X502_ProcessData().

It should be considered that by default buffer in driver or library is rated at amount of samples that will be entered during 4 s of continuous collection. If data are not read in due time using X502_Recv(), overflow of buffer in driver will occur and part of data for which there is no space in buffer will be lost. In case of further appearance of space in buffer, the place in stream where break of continuous data stream has occurred will be filled with word representing message on buffer overflow. If X502_ProcessData() in input array detects this word, function will return error X502_ERR_STREAM_OVERFLOW. In such case as well as in case of another errors of processing appearance all samples that were before error occurrence will be processed and returned within output arrays (which sizes will be updated accordingly).

The same is when forming common stream to output in required format function X502_PrepareData() receiving data from three arrays and saving them into external array is required. If any of sources shall not be used, zero pointer is sent as array. For channels that were not set for synchronous mode using X502_StreamsEnable() input array is not analyzed and data from it are not used.

It should be noticed that if for synchronous input initialization of transfer stream is executed via X502_StreamsStart(), since data start coming only after synchronous input start, the things are different about synchronous output. Since via X502_StreamsStart() the output of synchronous data should already be started, the part of data shall be loaded to module already. Thus, after permission of synchronous output through required channels using X502_StreamsEnable() and before start of synchronous output using X502_StreamsStart() it is required to pre-load part of synchronous stream data. For this you should call function X502_PreloadStart()due to which buffer for transfer will be allocated in driver or library and stream for transfer will be initialized and then write part of synchronous data into driver buffer using X502_PrepareData() and X502_Send(). If this is not done, synchronous output will start only when data are written in module and will not be attached to beginning of synchronous data collection/output.

Moreover, for synchronous output of data to DAC it is recommended to pre-set initial values at DAC using functions of asynchronous output. Otherwise upon beginning of synchronous output there can be small transition process from value to DAC that was before start of synchronous output, before submission of first required values, since DAC has its own filter and limits for speed of signal measurement.

3.7.3 Cyclic output

For module L502 beginning with driver and library version 1.0.4 and for module E502 (beginning with version 1.1.0) the support of cyclic output to DAC and digital outputs is implemented. This mode allows to load signal completely in buffer inside the driver (for L502) or processor Cortex-M4 (for E502) which content will be cyclically output with no need in further swapping.

Data for loading to cyclic buffer are prepared in the same way as for stream output using X502_PrepareData() and written using X502_Send() and can contain combination of data to both DAC channels and to digital outputs. Cyclic output is variant of synchronous output and for its operation it is required to permit required streams to output via X502_StreamsEnable() and synchronous input-output shall be started through X502_StreamsStart(). The same as with common stream output a part of channels can be used for output of cyclic signal, and part- asynchronously. But it is not allowed to use part of output signals in cyclic mode and part in stream mode with swapping (naturally, cyclic mode to output can be used with stream one to input).

For cyclic signal output there is double buffering used- in other words there can be two buffers allocated, while signal is output from one the following signal can be loaded to another buffer. Change of signal is executed upon end of period of previous one. In this case after writing of one signal it is required that signal change has been made before the next one can be loaded, otherwise function X502_OutCycleLoadStart() will return error (this can be used as attribute of that buffer is not ready for loading of new signal). Check of signal change completion can be done upon corresponding versions of software (see description of function X502_OutCycleCheckSetupDone()) and explicitly by function X502_OutCycleCheckSetupDone() or using flag X502_OUT_CYCLE_FLAGS_WAIT_DONE when calling X502_OutCycleSetup() in order that function returns control only after signals change fulfillment.

For loading signal at first function X502_OutCycleLoadStart() is called which specifies cyclic buffer size that will be used for storage of samples of all used output channels. For example, if it is required to use two DAC channels each is supplied with signal of 1000 points, the size shall be specified as 2000. After this the samples are loaded as in stream output with swapping using functions X502_PrepareData() and X502_Send(). In this case there shall be written in total as many samples as it was specified upon call of X502_OutCycleLoadStart(). After loading upon call of X502_OutCycleSetup() switching to loaded buffer is done and in this case depending on current status this causes the following:

- if synchronous input-output is not running (there was no call of X502_StreamsStart()), pre-loading of cyclic signal to module starts, but actual output of signal will begin only upon call of X502_StreamsStart() (or upon external condition of start). This allows to attach beginning of output of first cyclic signal sample to input beginning. In this case it is required to call X502_OutCycleSetup() with flag X502_OUT_CYCLE_FLAGS_WAIT_DONE to ensure that loading of signal will be completed before X502_StreamsStart() (first of all it is up-to-date for E502 where data transfer goes via interface and can take significant time).

- if synchronous input-output is running but no cyclic signal has been output before, cyclic signal output will begin upon X502_OutCycleSetup().

- if synchronous input-output is running and the previous cyclic signal is already being output, after call of X502_OutCycleSetup() flag on necessity to switch signals will appear in driver (or module for E502). After this event and up to reaching the end of previous cyclic buffer the change of output buffers will be executed. Thus, this allows to perform change of signal always at known point. When changing signal the release of old buffer is carried out and only after actual change it will be possible to call next time X502_OutCycleLoadStart() for loading of following signal. When using flag X502_OUT_CYCLE_FLAGS_WAIT_DONE function returns control only at the moment when the change itself is completed (if this capability is supported by software).

Stop of cyclic output can be executed by one of following methods:

- X502_OutCycleStop() stops cyclic output after output of last point at the boarder of cyclic buffer. This means that this function is used in order that cyclic output to be completed exactly at known point and values corresponding to last samples in cyclic signal to be left at outputs. And the function itself does not wait for stop of output if flag X502_OUT_CYCLE_FLAGS_WAIT_DONE is not specified.
- X502_StreamsDisable() specifying all used channels of output causes immediate completion of output and release of all buffers. At outputs there will be values left that were at the moment of call. After this you can complete again and initiate output in cyclic as well as in stream mode with swapping.
- X502_StreamsStop() causes immediate stop of all streams and stop of synchronization reference frequency generation. At output there will be values left that were at the moment of function call.

3.7.4 Buffer size and step for synchronous mode

This section has additional information on that how it is possible to set additional parameters controlling transfer of data stream in synchronous mode between module and PC. These parameters by default are set by library automatically. It is supposed that automatically set parameters should be suitable for most users and this section is not obligatory. But for cases when automatically defined parameters are not suitable user can set them himself. For this the present section provides description on how the buffer size and step is selected by library, what mean these parameters and how they can be set manually.

As it has already been said in previous section reception and transfer of synchronous data is conducted through buffer in driver or library- one buffer for reception, one for transfer.

Allocation of buffer to input is executed via X502_StreamsStart() if at least one source for synchronous input has been permitted. Allocation of buffer to output is carried out via X502_PreloadStart().

In this case buffer size is defined automatically by library depending on specified data transfer frequency. Buffer size is calculated such as it was enough for 4 seconds during synchronous input and for 3 seconds during synchronous output.

The second parameter characterizing transfer is transfer step/spacing. For module L502 this parameter determines step/pitch of interruptions. Data transfer between driver buffer and module is carried out directly by module via DMA. In this case in order that driver can know if data were written to buffer or read from it, during transfer of defined amount of samples module generates the interruption. In other words actually driver "knows" if data are transfered only after the transfer of specified amount of samples called as interruption step/pitch in this section (To be more exact not later than the specified amount of samples will be transfered because driver can read value of counter of transfered data from module and due to another conditions).

Thus the low pitch of interruptions allows driver to know earlier about received or transfered data but causes great loading of system. Library calculates the pitch of interruptions such as interruptions occur at frequency of 64 time per second.

For module E502 this parameter determines used size of request via USB. Moreover during input the data are placed for transfer in PC at their size equal to pitch but in this case the absence of arrival of new data can be put for transfer and less amount of data.

If user is not at any reason satisfied with these values he can set them manually using functions X502_SetStreamBufSize() and X502_SetStreamStep(). These functions shall be called before initialization of transfer streams (before X502_StreamsStart() or X502_PreloadStart()).

In particular, these are the cases when library values can not satisfy:

- User applies his BlackFin firmware and uses channels of synchronous data for transfer of user data that change greatly the data transfer speed. In this case library can not defined transfer frequency correctly because library does not know the speed of user data transfer.

- User changes channels that used in synchronous mode in process (after X502_StreamsStart()) and in this case speeds of transfer via these channels are significantly different. Since the calculation of buffer size is executed during channel initialization, it is performed only via the channels that were permitted at that moment. If, for example, only synchronous collection from ADC at relatively svelte frequency was permitted, buffer will also be allocated as svelte. In this case after data collection start synchronous input from digital lines at frequency 2MHz will be permitted, probably this buffer is appeared to be of insufficient size and most probably it will be overloaded. If both these streams were permitted initially and later synchronous input of digital lines will be prohibited, the initially calculated pitch of interruption will be too big and data from slow ADC stream will be updated at great delays. If channels frequencies are commensurable, switching on/off of one of them does not cause the significant change of parameters. Change of interruption pitch and buffer size at running data collection at the moment is not possible.

3.8 Features of operation via Ethernet interface and setting of network parameters

If USB interface in module E502 always works and dose not require additional configuration, for operation via Ethernet interface it is required to perform setting of interface parameters and permit this interface. It should be noticed that upon permitted Ethernet-interface it is possible to work with module via USB as well as via Ethernet. In this case collection/generation of data can be executed simultaneously only via one of interfaces (the one used for command for start of collection/output). Main parameters for operation via Ethernet are:

- IP-address of device. Written as 4 digits from 0 to 255 (module supports only protocol IPv4) separated by dots (for example, 192.168.0.10). Consists
of sub-network address and address of device inside the network. The latest shall be unique within sub-network.

- Subnet mask. Defines which part of address is related to sub-network address and which to device address. Mask 255.255.255.0 means that first 3 digits (192.168.0) designate sub-network address, the last digit (10) - address of device in sub-network.

- IP-address of gateway. Used only when module E502 and host from which module is controlled are in different sub-networks. Module transfers packages due to gateway address if address of destination is not in the same sub-network as the module. When working in local network it is not used.

- MAC-address of module (6 digits from 0 to 255 that are written in 16-decimal format). Physical address of device that shall be unique within local network. In "L Card" each module has its own factory MAC-address that can not be changed. But, if required, user can set his user MAC address and permit its application instead of factory one. In this case there is always possibility to return factory MAC address inhibiting the user one.

- Name of device instance. Unique name of this instance in form of string (up to 64 English symbols or 32 Russian one). Used for capability of automatic detection of modules in local network(details are in chapter Detection of modules in local network).

For operation, first of all, it is required to set correct IP-parameters (address, mask and, if required, gateway address), details are given in corresponding section FAQ.

IP-parameters of E502 module can be set in 3 ways:

- Set manually (static parameters). User shall take care that address belongs to required sub-network and be unique.

- Obtained automatically from DHCP-server. If automatic obtaining of address is set and DHCP-server is available in local network, module sends request to it and uses the IP-addresses allocated by DHCP-server.

- Automatically obtained local (link-local) address can be used (in compliance with RFC3927). Address is selected randomly within the range from 169.254.1.0 to 169.254.254.255 and it is checked that there is no other device having the same address in network (if there is it is tried to select next address, etc.). This makes possible the connection to device in local network without special configuration. It should be noticed that since the address is valid only within one network two different devices in different networks can have similar linklocal address, this causes that if PC has several active interfaces (and link-local address is used on both or vice versa the common address), the host does not know on which interface to search for required device. I.e. for connection via link-local address PC shall have whether one active network interface or the required interface shall use link-local address and another one shall use static or received via DHCP address.

When switching on automatic reception of address module selects link-local address (and checks its uniqueness) performing search of DHCP-server in network in parallel. If DHCP server is not found, link-local address is used. As soon as DHCP-server is found, it is preferred to use received from it address (in particular upon start in network with DHCP server module can for some time before getting address use link-local). Such algorithm is used in case of automatic reception of address, particularly in Windows OS, and in many Linux distributions (sometimes providing possibility of separate permission of DHCP and link-local address). It should be considered that automatically received address is checked by module for uniqueness in network that's why there is delay of several seconds from module connection to network till address assignment. Automatic reception of address does not require additional settings but in this case device address is unknown from PC side for connection establishment. In order to solve this

problem it is possible to use procedure of devices search in local network described in following section.

Change network settings is possible using program L-Card Measurement Studio.

Moreover, change of network settings of module can be done in software via API of library. For this there is separate type of configuration handle t_e502_eth_config_hnd. You should do following to change configuration:

1. Create configuration handle using E502_EthConfigCreate().
2. Read current device configuration using E502_EthConfigRead()
   (connection with module shall be set)
3. It is possible to get required parameters using functions E502_EthConfigGetXXX() and/or set new values using functions E502_EthConfigSetXXX().
4. After completion of changes you can write the modified configuration to module using E502_EthConfigWrite(). Module saves new configuration in non-volatile memory, inhibits Ethernet-interface after this initializes it again with new parameters. In this case if connection with device is established via Ethernet, it is required to break the connection and restore it for further operation (using new parameters)

To avoid unintentional change of configuration via network the configuration can be protected by simple password. Until the password is set it is required to send empty string as password. Setting new password is carried out similar to any other configuration parameters changes (using E502_EthConfigSetNewPassword()).

In case if password is forgotten, you can set connection via USB and change configuration (including password) by entering module serial number as current password.

3.9Detection of modules in local network

As compared to USB and PCI-Express interfaces for Ethernet-interface there is no standard capability to detect connected devices. But there are several protocols implementing which it is possible to detect devices of specified type in local network. For this capability module E502 supports protocols mDNS (in compliance with RFC6762) and DNS-SD (RFC6763). According to them each device upon connection declares set of services that it supports.

In order to differentiate instances of devices supporting similar type of services each instance has its unique name. This name is et during module configuration. If it is not set, “E502_<serial number>” is used as instance name, but user can set his own name characterizing purpose of certain module for more vivid identification. In addition to name of instance each service can have a set of text parameters describing the instance (for E502 parameters setting name of device type (field devname, value is always equal to E502) and serial number (field serial).

In compliance with this protocol host in local network has capability to find all instances of specified service in network. For detection there shall be corresponding service (or daemon) running monitoring changes of devices availability in network and functions e502apiwork with this service. Linux OS uses daemon Avahi as the implementation of this protocol, this daemon is included in most modern distributions and included into standard setting (or it is required to install corresponding pack manually). Windows OS uses service Bonjour that is not installed on standard basis , but installer is included in “L-Card L502/E502 SDK” and this service will be

installed when selecting corresponding point (it should be noticed that since the service is separate product which can be used with other software, it is not deleted automatically when deleting “L-Card L502/E502 SDK”. If required you should manually delete service through installation and delete of programs in "Control panel").

Application of this API allows automatically detect connected devices in local network mainly similar to other interfaces. But there are following features:

- Detection of module is related to packages sending and receiving responses that can be lost upon certain conditions and require re-repeats. This is executed at protocol level and not visible for user, but you should bear in mind that detection of device can take some time.
- Since switching off is monitored at protocol level there is no notification on physical cable disconnection. Accordingly when switching power off or disconnecting cable module switching off can not be detected during long time.

For search of devices you should call E502_EthSvcBrowseStart() after this use received context of devices search in network for further calls of E502_EthSvcBrowseGetEvent(). Each call returns information maximum about one event and immediately returns control as soon as it occurred. Each new detected module has corresponding event E502_ETH_SVC_EVENT_ADD. In case of parameters change (for example, address) event E502_ETH_SVC_EVENT_CHANGED arrives and upon disappearance (upon condition of above described features) — E502_ETH_SVC_EVENT_REMOVE. For each event handle of network service is returned due to which it is possible to define which module the event corresponds to (clarify name of instance and module serial number) and request IP-address of module. For each event (except case when event is not detected and code E502_ETH_SVC_EVENT_NONE is returned) this handle is required to be released using E502_EthSvcRecordFree() as soon as it becomes not required. In simplest case you can call E502_EthSvcBrowseGetEvent() until appearance of required device is detected or time-out is completed for device search. If needed, you can also use periodic call of E502_EthSvcBrowseGetEvent() for continuous monitoring of devices in network. In any case, when search of devices is finished it is required to call E502_EthSvcBrowseStop().

Setting connection with module via handle of network service is possible to be done manually due to received address through E502_EthSvcRecordResolveIPv4Addr() as well as by creating record about device using E502_MakeDevRecordByEthSvc() for further opening through X502_OpenByDevRecord().

Chapter 4

Constants, types of data and library functions

4.1Constants and enumerations.

4.1.1 Constants and macros.

Constant	Value	Description
X502_LTABLE_MAX_CH_CNT	256	Maximum number of logic channels in table
X502_ADC_RANGE_CNT	6	Amount of ranges for voltage measurement
X502_ADC_COMM_CH_CNT	32	Amount of ADC channels in mode with common ground
X502_ADC_DIFF_CH_CNT	16	Number of ADC channels in differential mode
X502_LCH_AVG_SIZE_MAX	128	Maximum value for hardware averaging for logic channel
X502_ADC_FREQ_DIV_MAX	(1024*1024)	Maximum value of ADC frequency divider
X502_DIN_FREQ_DIV_MAX	(1024*1024)	Maximum value of synchronous digital input frequency divider
X502_OUT_FREQ_DIV_MIN	2	Minimum value of synchronous output frequency divider
X502_OUT_FREQ_DIV_MAX	1024	Maximum value of synchronous output frequency divider
X502_OUT_FREQ_DIV_DEFAULT	2	Value of output frequency divider by default (that is always used in L502 with FPGA firmware version below 0.5)
X502_ADC_INTERFRAME_DELAY_MAX	(0x1FFFFF)	Maximum value of inter-frame delay for ADC
X502_BF_CMD_DEFAULT_TOUT	500	Time-out by default for performance of command to BlackFin
X502_ADC_SCALE_CODE_MAX	6000000	ADC code corresponding to maximum scale value

X502_DAC_SCALE_CODE_MAX	30000	DAC code corresponding to maximum scale value
X502_DEVNAME_SIZE	32	Maximum amount of symbols in the string with device name
X502_SERIAL_SIZE	32	Maximum amount of symbols in the string with serial number
X502_LOCATION_STR_SIZE	64	Maximum amount of symbols in the string with connection description
X502_MAC_ADDR_SIZE	6	Size of MAC-address for Ethernet interface
X502_INSTANCE_NAME_SIZE	64	Size of the string with description of device instance
X502_PASSWORD_SIZE	32	Maximum size of the string with password for settings
X502_EXT_REF_FREQ_MAX	1500000	Maximum possible value of external reference frequency
X502_FLASH_USER_SIZE	0x100000	Size of user area of Flash-memory
X502_BF_REQ_TOUT	500	Standard time-out for performance of request to BlackFin in ms
X502_DAC_RANGE	5.	DAC range in volts
X502_DAC_CH_CNT	2	Amount of DAC channels
X502_DOUT_LINES_CNT	16	Amount of module digital outputs
X502_STREAM_IN_MSG_OVERFLOW	0x01010000	word in stream meaning that overflow occurred
X502_DEVREC_SIGN	0x4C524543	Value of signature field in record about device t_x502_devrec. Attribute of that record is valid (set by functions upon receiving records about devices)

4.1.2 Events of network services search

Type: t_e502_eth_svc_event		
Description: Codes of events appeared during search of network services returned by function E502_EthSvcBrowseGetEvent()		
Constant	Value	Description
E502_ETH_SVC_EVENT_NONE	0	No event occurred

E502_ETH_SVC_EVENT_ADD	1	Appearance of new network service detected
E502_ETH_SVC_EVENT_REMOVE	2	Disappearance of previously available network service detected
E502_ETH_SVC_EVENT_CHANGED	3	Change of parameters of previously detected network service

4.1.3 Library error codes

Type: t_x502_errs		
Description: Library error codes		
Constant	Value	Description
X502_ERR_OK	0	Function is fulfilled with no error
X502_ERR_INVALID_HANDLE	-1	Invalid module handle is sent to function
X502_ERR_MEMORY_ALLOC	-2	Memory allocation error
X502_ERR_ALREADY_OPENED	-3	Try to open already opened device
X502_ERR_DEVICE_NOT_FOUND	-4	Device with specified parameters is not found in the system
X502_ERR_DEVICE_ACCESS_DENIED	-5	Access to device is prohibited (As a rule due to the fact that the device is already opened in other program)
X502_ERR_DEVICE_OPEN	-6	Device opening error
X502_ERR_INVALID_POINTER	-7	Invalid pointer is sent to function
X502_ERR_STREAM_IS_RUNNING	-8	Function can not be fulfilled when data collection stream is running
X502_ERR_RECV	-9	Error of reading data received from synchronous input
X502_ERR_SEND	-10	Error of writing data for synchronous output
X502_ERR_STREAM_OVERFLOW	-11	Overflow of internal buffer for synchronous input stream is occurred
X502_ERR_UNSUP_STREAM_MSG	-12	Unknown message in synchronous input stream

X502_ERR_MUTEX_CREATE	-13	Error of creation of system mutex
X502_ERR_MUTEX_INVALID_HANDLE	-14	Incorrect mutex handle
X502_ERR_MUTEX_LOCK_TOUT	-15	Time of waiting for mutex release is up
X502_ERR_MUTEX_RELEASE	-16	Error of mutex release
X502_ERR_INSUFFICIENT_SYSTEM_ RESOURCES	-17	Not enough system resources
X502_ERR_NOT_IMPLEMENTED	-18	This capability is not implemented yet
X502_ERR_INSUFFICIENT_ARRAY_SIZE	-19	Insufficient array size
X502_ERR_FPGA_REG_READ	-20	Error of reading FPGA register
X502_ERR_FPGA_REG_WRITE	-21	Error of writing FPGA register
X502_ERR_STREAM_IS_NOT_RUNNING	-22	Data collection has been established
X502_ERR_INTERFACE_RELEASE	-23	Error of release of interface
X502_ERR_THREAD_START	-24	Error of stream start
X502_ERR_THREAD_STOP	-25	Error of stream stop
X502_ERR_DEVICE_DISCONNECTED	-26	Device has been disconnected
X502_ERR_IOCTL_INVALID_RESP_SIZE	-27	Invalid size of response for controlling request
X502_ERR_INVALID_DEVICE	-28	Invalid type of device
X502_ERR_INVALID_DEVICE_RECORD	-29	Invalid record about the device
X502_ERR_INVALID_CONFIG_HANDLE	-30	Invalid module configuration handle
X502_ERR_DEVICE_NOT_OPENED	-31	Connection with device is closed or has not been established
X502_ERR_INVALID_OP_FOR_IFACE	-32	This operation is not available for current interface of connection with device
X502_ERR_FPGA_NOT_LOADED	-33	Module FPGA is not loaded
X502_ERR_INVALID_USB_ CONFIGURATION	-34	Invalid configuration of USB- of device
X502_ERR_INVALID_SVC_BROWSE_ HANDLE	-35	Invalid handle of context of device search in network
X502_ERR_INVALID_SVC_RECORD_ HANDLE	-36	Invalid handle of record about service

X502_ERR_DNSSD_NOT_RUNNING	-37	Devices detection in local network program is not running
X502_ERR_DNSSD_COMMUNICATION	-38	Error while addressing to program of devices detection in local network
X502_ERR_SVC_RESOLVE_TIMEOUT	-39	Time-out of request for parameters of auto-detection of device network service is exceeded
X502_ERR_INSTANCE_NAME_ENCODING	-40	Error in encoding of device instance name
X502_ERR_INSTANCE_MISMATCH	-41	Modules instances are mismatched
X502_ERR_NOT_SUP_BY_FIRMWARE	-42	Capability is not supported by current firmware version of device
X502_ERR_NOT_SUP_BY_DRIVER	-43	Capability is not supported by current firmware version of driver of device
X502_ERR_OUT_CYCLE_SETUP_TOUT	-44	Time of waiting for establishment of cyclic signal to output
X502_ERR_UNKNOWN_FEATURE_CODE	-45	Unknown code of supported capability
X502_ERR_INVALID_LTABLE_SIZE	-102	Invalid size of logic table is specified
X502_ERR_INVALID_LCH_NUMBER	-103	Invalid number of logic channel is specified
X502_ERR_INVALID_LCH_RANGE	-104	ADC range value is specified incorrectly
X502_ERR_INVALID_LCH_MODE	-105	Measurement mode for logic channel is set incorrectly
X502_ERR_INVALID_LCH_PHY_NUMBER	-106	Physical channel number is set incorrectly when setting logic one
X502_ERR_INVALID_LCH_AVG_SIZE	-107	Averaging size for logic channel is set incorrectly
X502_ERR_INVALID_ADC_FREQ_DIV	-108	ADC data collection frequency divider is set incorrectly

X502_ERR_INVALID_DIN_FREQ_DIV	-109	Synchronous input frequency divider of digital lines is set incorrectly
X502_ERR_INVALID_MODE	-110	Module X502 operating mode is set incorrectly
X502_ERR_INVALID_DAC_CHANNEL	-111	DAC channel number is incorrect
X502_ERR_INVALID_REF_FREQ	-112	Incorrect code of selection of synchronization reference frequency
X502_ERR_INVALID_INTERFRAME_DELAY	-113	Value of inter-frame delay is set incorrectly
X502_ERR_INVALID_SYNC_MODE	-114	Synchronization mode is set incorrectly
X502_ERR_INVALID_STREAM_CH	-115	Data stream number is set incorrectly
X502_ERR_INVALID_OUT_FREQ_DIV	-116	Synchronous output frequency divider is set incorrectly
X502_ERR_REF_FREQ_NOT_LOCKED	-131	Error of synchronization reference frequency capture
X502_ERR_IOCTL_FAILD	-132	Controlling request to driver is completed with error
X502_ERR_IOCTL_TIMEOUT	-133	Time-out of waiting for completion of controlling request to driver is over
X502_ERR_GET_INFO	-134	Error of receiving information on device from driver
X502_ERR_DIG_IN_NOT_RDY	-135	New word from digital lines has not been read during the period of waiting
X502_ERR_RECV_INSUFFICIENT_WORDS	-136	Not enough words received from module
X502_ERR_DAC_NOT_PRESENT	-137	A try to execute operation requiring DAC availability upon its absence
X502_ERR_SEND_INSUFFICIENT_WORDS	-138	Not enough words sent to module
X502_ERR_NO_CMD_RESPONSE	-139	No response arrived for transfered command
X502_ERR_PROC_INVALID_CH_NUM	-140	Incorrect number of channel in synchronous input stream being processed

X502_ERR_PROC_INVALID_CH_RANGE	-141	Incorrect code of range within synchronous input stream being processed
X502_ERR_FLASH_INVALID_ADDR	-142	Incorrect address is set in Flash-memory
X502_ERR_FLASH_INVALID_SIZE	-143	Incorrect data block size is set when working with Flash-memory
X502_ERR_FLASH_WRITE_TOUT	-144	Time-out of waiting for completion of writing to Flash-memory is over
X502_ERR_FLASH_ERASE_TOUT	-145	Time-out of waiting for completion of Flash-memory block erasing is over
X502_ERR_FLASH_SECTOR_BOUNDARY	-146	Specified area for Flash-memory erasing breaks the block boarder of 4 KB
X502_ERR_SOCKET_OPEN	-147	Failed to open socket for connection
X502_ERR_CONNECTION_TOUT	-148	Time of connection is exceeded
X502_ERR_CONNECTION_CLOSED_BY_DEV	-149	Connection is closed by other device
X502_ERR_SOCKET_SET_BUF_SIZE	-150	Failed to set specified size of socket buffer
X502_ERR_NO_DATA_CONNECTION	-151	Connection for data transfer is not set
X502_ERR_NO_STREAM_END_MSG	-152	Failed to wait for message on stream completion
X502_ERR_CONNECTION_RESET	-153	Connection has been reset by other party
X502_ERR_HOST_UNREACHABLE	-154	Failed to find host with specified address
X502_ERR_TCP_CONNECTION_ERROR	-155	Error of setting TCP-connection
X502_ERR_LDR_FILE_OPEN	-180	Failed to open file of BlackFin firmware
X502_ERR_LDR_FILE_READ	-181	Error of reading from file of BlackFin firmware
X502_ERR_LDR_FILE_FORMAT	-182	Invalid format of BlackFin firmware file

X502_ERR_LDR_FILE_UNSUP_FEATURE	-183	Capability of LDR-file not available during writing BlackFin firmware via HDMA is used
X502_ERR_LDR_FILE_UNSUP_STARTUP_ ADDR	-184	Invalid start address of program in BlackFin firmware
X502_ERR_BF_REQ_TIMEOUT	-185	Time-out of request for reading/writing of memory performance is over BlackFin
X502_ERR_BF_CMD_IN_PROGRESS	-186	Command for BlackFin is still being processed
X502_ERR_BF_CMD_TIMEOUT	-187	Time of controlling command performance by BlackFin is over
X502_ERR_BF_CMD_RETURN_INSUF_DATA	-188	Not enough data are returned as response to command to BlackFin
X502_ERR_BF_LOAD_RDY_TOUT	-189	Time-out of waiting for BlackFin processor readiness for firmware writing is over
X502_ERR_BF_NOT_PRESENT	-190	A try to execute operation for which the signaling processor is required upon absence of signaling processor in module
X502_ERR_BF_INVALID_ADDR	-191	Invalid address of BlackFin memory when writing or reading via HDMA
X502_ERR_BF_INVALID_CMD_DATA_SIZE	-192	Invalid size of data transfered with controlling command in BlackFin

4.1.4 Interface of connection with module

Type: t_x502_iface		
Description: Interface of connection with module		
Constant	Value	Description
X502_IFACE_UNKNOWN	0	Unknown interface
X502_IFACE_USB	1	Device is connected via USB
X502_IFACE_ETH	2	Device is connected through Ethernet via TCP/IP
X502_IFACE_PCI	3	Device is connected via PCI/PCIe

4.1.5 Flags controlling search of present modules

Type: t_x502_getdevs_flags		
Description: Flags controlling search of present modules		
Constant	Value	Description
X502_GETDEVS_FLAGS_ONLY_NOT_OPENED	1	Attribute of that it is required to return serial numbers only of devices that has not been opened yet

4.1.6 Flags to control digital outputs.

Type: t_x502_digout_word_flags		
Description: Flags to control digital outputs. Can be united through logic "OR" with values of digital outputs upon asynchronous output using X502_AsyncOutDig() or transfered inX502_PrepareData() upon synchronous output.		
Constant	Value	Description
X502_DIGOUT_WORD_DIS_H	0x00020000	Inhibit (transfer into third status) of high half of digital outputs
X502_DIGOUT_WORD_DIS_L	0x00010000	Inhibit of low half of digital outputs

4.1.7 Constants for reference frequency selection

Type: t_x502_ref_freq		
Description: Constants for reference frequency selection		
Constant	Value	Description
X502_REF_FREQ_2000KHZ	2000000	Frequency 2MHz
X502_REF_FREQ_1500KHZ	1500000	Frequency 1.5MHz

4.1.8 ADC channel measurement ranges

Type: t_x502_adc_range		
Description: ADC channel measurement ranges		
Constant	Value	Description
X502_ADC_RANGE_10	0	Range +/-10V
X502_ADC_RANGE_5	1	Range +/-5V
X502_ADC_RANGE_2	2	Range +/-2V
X502_ADC_RANGE_1	3	Range +/-1V
X502_ADC_RANGE_05	4	Range +/-0.5V
X502_ADC_RANGE_02	5	Range +/-0.2V

4.1.9 Measurement mode for logic channel

Type: t_x502_lch_mode		
Description: Measurement mode for logic channel		
Constant	Value	Description
X502_LCH_MODE_COMM	0	Voltage measurement in relation to common ground
X502_LCH_MODE_DIFF	1	Differential measurement of voltage
X502_LCH_MODE_ZERO	2	Self zero measurement

4.1.10 Synchronization modes.

Type: t_x502_sync_mode		
Description: Modes of setting of synchronization frequency source and attribute of synchronous input-output start		
Constant	Value	Description
X502_SYNC_INTERNAL	0	Internal signal
X502_SYNC_EXTERNAL_MASTER	1	From external master via inter-module synchronization connector
X502_SYNC_DI_SYN1_RISE	2	Due to edge/rise of signal DI_SYN1
X502_SYNC_DI_SYN1_FALL	3	Due to edge/rise of signal DI_SYN2
X502_SYNC_DI_SYN2_RISE	6	Due to fall of signal DI_SYN1
X502_SYNC_DI_SYN2_FALL	7	Due to fall of signal DI_SYN2

4.1.11 Flags controlling processing of received data

Type: t_x502_proc_flags		
Description: Flags controlling processing of received data		
Constant	Value	Description
X502_PROC_FLAGS_VOLT	0x00000001	Attribute that ADC value shall be converted into volts
X502_PROC_FLAGS_DONT_CHECK_CH	0x00010000	Attribute that coincidence of channels numbers in received data with channels from logic table should not be checked. Can be used at non-standard BlackFin firmware during transfer to PC of not all data.

4.1.12 Flags for designation of synchronous data streams

Type: t_x502_streams		
Description: Flags for designation of synchronous data streams		
Constant	Value	Description
X502_STREAM_ADC	0x01	Data stream from ADC
X502_STREAM_DIN	0x02	Data stream from digital inputs
X502_STREAM_DAC1	0x10	Data stream of first DAC channel
X502_STREAM_DAC2	0x20	Data stream of second DAC channel
X502_STREAM_DOUT	0x40	Data stream to digital outputs
X502_STREAM_ALL_IN	X502_STREAM_ADC | X502_STREAM_DIN	Unification of all flags denoting streams of data to input
X502_STREAM_ALL_OUT	X502_STREAM_DAC1 | X502_STREAM_DAC2 | X502_STREAM_DOUT	Unification of all flags denoting streams of data to output

4.1.13 Constants determining type of transfered sample from PC to module

Type: t_x502_stream_out_wrd_type		
Description: Constants determining type of transfered sample from PC to module		
Constant	Value	Description
X502_STREAM_OUT_WORD_TYPE_DOUT	0x0	Digital output
X502_STREAM_OUT_WORD_TYPE_DAC1	0x40000000	Code for 1 DAC channel
X502_STREAM_OUT_WORD_TYPE_DAC2	0x80000000	Code for 2 DAC channel

4.1.14 L502 module operation mode

Type: t_x502_mode		
Description: L502 module operation mode		
Constant	Value	Description
X502_MODE_FPGA	0	All streams of data are transfered via FPGA bypassing signaling processor BlackFin
X502_MODE_DSP	1	All streams of data are transfered via signaling processor that shall be loaded by firmware to process these streams
X502_MODE_DEBUG	2	Debugging mode

4.1.15 DAC channels numbers.

Type: t_x502_dac_ch		
Description: Numbers of DAC channels for indication in X502_AsyncOutDac()		
Constant	Value	Description
X502_DAC_CH1	0	First DAC channel
X502_DAC_CH2	1	Second DAC channel

4.1.16 Flags used under data output to DAC.

Type: t_x502_dacout_flags		
Description: Flags combination of which is possible to be transfered to v X502_AsyncOutDac() or X502_PrepareData() to determine activities that shall be performed by these functions with transfered value before output them to DAC		
Constant	Value	Description
X502_DAC_FLAGS_VOLT	0x0001	Specifies that value is set in Volts and during output it should be converted in DAC codes. If flag is not specified, it is considered that value is initially in codes
X502_DAC_FLAGS_CALIBR	0x0002	Specifies that calibration coefficients should be applied before output value to DAC.

4.1.17 Numbers of channels for data streams transfer

Type: t_x502_stream_ch		
Description: Numbers of channels for data streams transfer		
Constant	Value	Description
X502_STREAM_CH_IN	0	Common channel to input
X502_STREAM_CH_OUT	1	Common channel to output

4.1.18 Digital lines where pull-up resistors can be connected

Type: t_x502_pullups		
Description: Flags specifying on which digital inputs pull-up resistors shall be switched on. For different modules different sets of flags are available.		
Constant	Value	Description
X502_PULLUPS_DI_H	0x01	High half of digital inputs (only for L502)
X502_PULLUPS_DI_L	0x02	Low half of digital inputs (only for L502)
X502_PULLUPS_DI_SYN1	0x04	Line SYN1

X502_PULLUPS_DI_SYN2	0x08	Line SYN2
X502_PULLDOWN_CONV_IN	0x10	Pull-up to 0 of line of inter-module synchronization CONV_IN (only for E502)
X502_PULLDOWN_START_IN	0x20	Pull-up to 0 of line of inter-module synchronization START_IN (only for E502)

4.1.19 Flags determining availability of options in the module and availability of required parameters

Type: t_x502_dev_flags		
Description: Flags determining availability of options in the module and availability of required parameters		
Constant	Value	Description
X502_DEVFLAGS_DAC_PRESENT	0x00000001	Attribute of bi-channel DAC availability
X502_DEVFLAGS_GAL_PRESENT	0x00000002	Attribute of galvanic isolation availability
X502_DEVFLAGS_BF_PRESENT	0x00000004	Attribute of BlackFin signaling processor availability
X502_DEVFLAGS_IFACE_SUPPORT_USB	0x00000100	Attribute that device supports USB interface
X502_DEVFLAGS_IFACE_SUPPORT_ETH	0x00000200	Attribute that device supports Ethernet
X502_DEVFLAGS_IFACE_SUPPORT_PCI	0x00000400	Attribute that device supports PCI/PCIExpress interface
X502_DEVFLAGS_INDUSTRIAL	0x00008000	Attribute that device is performed in industrial design
X502_DEVFLAGS_FLASH_DATA_VALID	0x00010000	Attribute that Flash-memory has information on module
X502_DEVFLAGS_FLASH_ADC_CALIBR_VALID	0x00020000	Attribute that Flash-memory has valid calibration coefficients of ADC
X502_DEVFLAGS_FLASH_DAC_CALIBR_VALID	0x00040000	Attribute that Flash-memory has valid calibration coefficients of DAC
X502_DEVFLAGS_FPGA_LOADED	0x00800000	Attribute that there is FPGA firmware and it is successfully loaded

X502_DEVFLAGS_DEVREC_OPENED	0x01000000	Attribute that device is already closed (valid only within t_x502_devrec)

4.1.20 Type of device location string content

Type: t_x502_location_type		
Description: This field defines content of field location in structure t_x502_devrec		
Constant	Value	Description
X502_LOCATION_TYPE_NONE	0	Field of device location has no information
X502_LOCATION_TYPE_ADDR	1	Field of device location has string with device address
X502_LOCATION_TYPE_INSTANCE_NAME	2	Field of device location has string with instance name

4.1.21 Flags for cyclic output mode

Type: t_x502_out_cycle_flags		
Description: These flags can be transfered to X502_OutCycleSetup() and X502_OutCycleStop()		
Constant	Value	Description
X502_OUT_CYCLE_FLAGS_FORCE	0x01	Flag specifies that stop or change of signal can occur without waiting for end of previous signal cycle. This allows to perform switching faster (but any way can be set for transfer up to 256 KSamples that shall be transfered) but point of change or stop can be at any place of period
X502_OUT_CYCLE_FLAGS_WAIT_DONE	0x02	Flag specifies that function shall wait for complete loading of signal and establishment of signal to output (for X502_OutCycleSetup()) or completion of cyclic signal generation (for X502_OutCycleStop()). Without it functions only send command to module returning control at once.

		This wait can take significant time depending on signal size (and on size of previous signal in case of change or stop of generation without X502_OUT_CYCLE_FLAGS_FORCE). This check can be done by separate function X502_OutCycleCheckSetupDone(). This flag matters only in cases when function X502_OutCycleCheckSetupDone() is supported otherwise it is ignored.

4.1.22 Codes of module capabilities which can be supported or not depending on module type, firmware versions, etc.

Type: t_x502_features		
Description: Codes of module capabilities which can be supported or not depending on module type, firmware versions, etc.		
Constant	Value	Description
X502_FEATURE_OUT_FREQ_DIV	1	Support of setting of output frequency divider different from X502_OUT_FREQ_DIV_DEFAULT
X502_FEATURE_OUT_STATUS_FLAGS	2	Capability of output status flags reading using X502_OutGetStatusFlags()

4.1.23 Status flags for synchronous output

Type: t_x502_out_status_flags		
Description: Status flags for synchronous output		
Constant	Value	Description

X502_OUT_STATUS_FLAG_BUF_IS_EMPTY	0x01	Flag specifies that at the moment buffer in module for transfer is empty
X502_OUT_STATUS_FLAG_BUF_WAS_EMPTY	0x02	Flag specifies that buffer to output has been emptied since the beginning of synchronous input-output start or since the moment of latest reading of status using X502_OutGetStatusFlags() (depending on what was the latest)

4.2 Data types.

4.2.1 Record about the device

Type: t_x502_devrec		
Description: Structure describing device due to which it can be connected with		
Field	Type	Field description
sign	uint32_t	Attribute of valid structure. If record is valid (corresponds to any device), shall be equal to X502_DEVREC_SIGN)
devname	char [X502_DEVNAME_SIZE]	Device name
serial	char [X502_SERIAL_SIZE]	Serial number
location	char [X502_LOCATION_STR_SIZE]	Description of connection (if any)
flags	uint32_t	Flags from t_x502_dev_flags describing device
iface	uint8_t	Interface through which device is connected
location_type	uint8_t	Specifies what exactly is saved in field location (one value from t_x502_location_type)
res	char [122]	Reserve
internal	t_x502_devrec_inptr *	Opaque pointer to the structures with additional information required for device opening

4.2.2 Range calibration coefficients.

Type: t_x502_cbr_coef		
Description: Structure contains calibration values of zero shift and coefficient of scale for one ADC or DAC range. Resulting value of ADC is calculated as (val-offs)*k, where val - not calibrated value		
Field	Type	Field description
offs	double	zero shift
k	double	scale coefficient

4.2.3 Module calibration coefficients.

Type: t_x502_cbr		
Description: Structure containing all calibration coefficients that are used by module L502/E502		
Field	Type	Field description
adc	t_x502_cbr_coef [X502_ADC_RANGE_CNT]	ADC calibration coefficients
res1	uint32_t [64]	Reserve
dac	t_x502_cbr_coef [X502_DAC_CH_CNT]	DAC calibration coefficients
res2	uint32_t [20]	Reserve

4.2.4 Information on L502/E502 module.

Type: t_x502_info		
Description: Structure containing constant information on L502/E502 module that, as a rule, is not changed after opening		
Field	Type	Field description
name	char [X502_DEVNAME_ SIZE]	Name of device (“L502” or “E502”)
serial	char [X502_SERIAL_ SIZE]	Serial number
devflags	uint32_t	Flags from t_x502_dev_flags describing availability of certain options in module
fpga_ver	uint16_t	FPGA version (high bit- major, low bit- minor)
plda_ver	uint8_t	FPGA version controlling analog part
board_rev	uint8_t	Board revision

mcu_firmware_ver	uint32_t	Version of CortexM4controller firmware. Valid only for E502
factory_mac	uint8_t [X502_MAC_ ADDR_SIZE]	Factory MAC-address- valid only for devices with Ethernet interface
res	uint8_t [110]	Reserve
cbr	t_x502_cbr	Factory calibration coefficients (from Flash-memory)

4.2.5 Network interface configuration handle.

Type: t_e502_eth_config_hnd
Description: Opaque pointer to the structure containing parameters of network interface configuration of E502module. Structure fields are not available for user program directly only through library functions. Configuration handle is created using E502_EthConfigCreate() and is released at the end of work using E502_EthConfigFree(). As a rule, all settings shall not be filled in by user manually, usually first they are read from device using E502_EthConfigRead() and after this part of settings can be changed and save to module through E502_EthConfigWrite()

4.2.6 Handle of context of device search in network

Type: t_e502_eth_svc_browse_hnd
Description: Pointer to opaque structure with information on status of current session of search for devices in network. Created at the beginning of search by calling E502_EthSvcBrowseStart() and deleted using E502_EthSvcBrowseStop()

4.2.7 Network service handle

Type: t_e502_eth_svc_record_hnd
Description: Pointer to opaque structure with information on service in network corresponding to one E502 module. Used during automatic detection of devices in local network. Created when calling E502_EthSvcBrowseGetEvent() and deleted using E502_EthSvcRecordFree()

4.2.8 Internal information on record about the device

Type: t_x502_devrec_inptr
Description: Opaque structure with information sufficient for establishment of connection with it. Depends on device type, interface of connection and not available for user directly, used by library in X502_OpenByDevRecord()

4.2.9 Module handle.

Type: t_x502_hnd
Description: Opaque pointer to the structure containing information on module settings and current connection with it. Structure fields are not available for user program directly only through library functions. Module control functions take module handle as their first parameter. Module handle is created using X502_Create() and is released at the end of work using X502_Free().

4.2.10 List of serial numbers

Type: t_x502_serial_list
Description: Type determines array of serial numbers for amount of modules defined at program operation stage.

4.3Functions

4.3.1 Functions for creation and release of module handle.

4.3.1.1 Creation of module handle.

Format: t_x502_hnd X502_Create (void)
Description: Creation of module handle for further operation with module E502 and L502. In case of successful memory allocation it initializes the handle fields by default values.
Returned value: NULL in case of error, otherwise- module handle

4.3.1.2 Release of module handle.

Format: int32_t X502_Free (t_x502_hnd hnd)
Description: Release of memory dedicated for module handle of using X502_Create(). After this the handle can not be used regardless of returned value!
Parameters: hnd — Device handle
Returned value: Error code

4.3.2 Functions for opening and receiving information on module.

4.3.2.1 Receiving list of L502 modules serial numbers.

Format: int32_t L502_GetSerialList (char serials[][X502_SERIAL_SIZE], uint32_t size, uint32_t flags, uint32_t *devcnt)
Description: Function returns list of numbers of all found L502 modules regardless of if they are opened or not. If you need list of only those modules that are not opened (i.e. only those with which it is possible to set the connection), flag X502_GETDEVS_FLAGS_ONLY_NOT_OPENED can be sent to function.
Parameters: serials — Array of size size*X502_SERIAL_SIZE bytes where serial numbers of found modules will be saved. Can be NULL, if size=0, and devcnt!=NULL if you need only to receive amount of modules in system. size — Defines how many serial numbers can be saved in array serial. Only first size serial numbers will be saved. Can be 0, if serials=NULL flags — Flags from t_x502_getdevs_flags determining function behavior. devcnt — If devcnt!=NULL, total amount of found L502 modules is saved in this variable (can be more than size).
Returned value: If <0 - error code, otherwise amount of saved serial numbers in array serials (always <= size)

4.3.2.2 Opening L502 module as per its serial number.

Format: int32_t L502_Open (t_x502_hnd hnd, const char *serial)
Description: Function sets connection with L502 module due to its serial number. After successful fulfillment of this function user gets unique access to module through module handle. Before closing connection using X502_Close() nobody will be able to set the connection with module (error X502_ERR_DEVICE_ACCESS_DENIED will be returned). If NULL or empty string is sent as serial number, the connection will be set with first found module with which the connection is successfully established. If there is no module in the system, error X502_ERR_DEVICE_NOT_FOUND will be returned. If there are L502 modules in system but connection is failed to be set with any, error received when tried to set connection with latest found module will be returned. After completion of working with device the connection shall be closed using X502_Close().
Parameters: hnd — Device handle. serial — Pointer to string with serial number of module being opened or NULL.
Returned value: Error code.

4.3.2.3 Receiving list of serial numbers of E502 modules connected through USB.

Format: int32_t E502_UsbGetSerialList (char serials[][X502_SERIAL_SIZE], uint32_t size, uint32_t flags, uint32_t *devcnt)
Description: Function returns list of numbers of all found E502 modules regardless of if they are opened or not. Function for present moment is not supporting flag X502_GETDEVS_FLAGS_ONLY_NOT_OPENED.
Parameters: serials — Array of size size*X502_SERIAL_SIZE bytes where serial numbers of found modules will be saved. Can be NULL, if size=0, and devcnt!=NULL if you need only to receive amount of modules in system. size — Defines how many serial numbers can be saved in array serial. Only first size serial numbers will be saved. Can be 0, if serials=NULL flags — Flags from t_x502_getdevs_flags determining function behavior. devcnt — If devcnt!=NULL, total amount of found E502 modules is saved in this variable (can be more than size).
Returned value: If <0 - error code, otherwise amount of saved serial numbers in array serials (always <= size)

4.3.2.4 Opening of E502 module connected through USB as per its serial number.

Format: int32_t E502_OpenUsb (t_x502_hnd hnd, const char *serial)
Description: Function sets connection with module E502 connected via USB interface according to its serial number. After successful fulfillment of this function user gets unique access to module through module handle. Before closing connection using X502_Close() nobody will be able to set the connection with module (error X502_ERR_DEVICE_ACCESS_DENIED will be returned). If NULL or empty string is sent as serial number, the connection will be set with first found module with which the connection is successfully established. If there is no module in the system, error X502_ERR_DEVICE_NOT_FOUND will be returned. If there are E502 modules in system but connection is failed to be set with any, error received when tried to set connection with latest found module will be returned. After completion of working with device the connection shall be closed using X502_Close().
Parameters: hnd — Device handle. serial — Pointer to string with serial number of module being opened or NULL.
Returned value: Error code.

4.3.2.5 Opening of E502 module as per IP-address

Format: int32_t E502_OpenByIpAddr (t_x502_hnd hnd, uint32_t ip_addr, uint32_t flags, uint32_t tout)
Description: Function sets connection with E502 module connected via Ethernet interface, for which specified address IPv4 is established. After completion of working with device the connection shall be closed using X502_Close().
Parameters: hnd — Device handle. ip_addr — IPv4 address of module in form of 32-bit word. For address “a.b.c.d” ip_addr = (a«24)|(b«16)|(c«8)|d. flags — Flags controlling function operation. Reserve, always shall be 0. tout — Time for setting the connection in ms. If connection can not be completed within specified time, function will return error.
Returned value: Error code.

4.3.2.6 Closing connection with module.

Format: int32_t X502_Close (t_x502_hnd hnd)
Description: Function breaks the connection with module E502/L502 if it has been established before (otherwise it does nothing). Module handle is not released. Memory for module handle shall be released by calling X502_Free().
Parameters: hnd — Module handle.
Returned value: Error code.

4.3.2.7 Receiving information on module.

Format: int32_t X502_GetDevInfo (t_x502_hnd hnd, t_x502_info *info)
Description: Receiving information on module L502/E502 with which the connection is set.
Parameters: hnd — Module handle. info — Information on module (see description of type t_x502_info).
Returned value: Error code.

4.3.3 Functions for working with device records

4.3.3.1 Receive list of records corresponding to connected L502 modules.

Format: int32_t L502_GetDevRecordsList (t_x502_devrec *list, uint32_t size, uint32_t flags, uint32_t *devcnt)
Description: Function detects all connected modules L502 and initializes records about each found device and saves them to transfered list (if not zero). Records returned in the list shall be cleaned after using by means of X502_FreeDevRecordList() (also in case of repeated call of L502_GetDevRecordsList() with the same array of records, records received during previous call shall be first cleaned).
Parameters: list — Array for saving records about found devices. Shall contain space for storage of minimum size records. Can be NULL, if size=0, and devcnt!=NULL if you need only to receive amount of modules in system. size — Determines maximum amount of records can be saved in array list. Only first size records will be saved if there are more devices detected. flags — Flags from t_x502_getdevs_flagsdetermining function behavior. devcnt —If handle is not zero/null, total amount of found L502 modules is saved in this variable (can be more than size).
Returned value: If <0 — error code, otherwise amount of saved records about found devices (always <= size). Exactly for this size shall be made in future X502_FreeDevRecordList() to release memory allocated for information which reference for record.

4.3.3.2 Receive list of records corresponding to connected E502 modules.

Format: int32_t E502_UsbGetDevRecordsList (t_x502_devrec *list, uint32_t size, uint32_t flags, uint32_t *devcnt)
Description: Function detects all modules E502 connected via USB interface and initializes records about each found device and saves them to transfered list (if not zero). Records returned in the list shall be cleaned after using by means of X502_FreeDevRecordList() (also in case of repeated call of E502_UsbGetDevRecordsList() with the same array of records, records received during previous call shall be first cleaned).
Parameters: list — Array for saving records about found devices. Shall contain space for storage of minimum size records. Can be NULL, if size=0, and devcnt!=NULL if you need only to receive amount of modules in system. size — Determines maximum amount of records can be saved in array list. Only first size records will be saved if there are more devices detected. flags — Flags from t_x502_getdevs_flagsdetermining function behavior.

devcnt — If handle is not zero/null, total amount of found E502 modules connected via USB interface is saved in this variable (can be more than size).
Returned value: If <0 — error code, otherwise amount of saved records about found devices (always <= size). Exactly for this size shall be made in future X502_FreeDevRecordList() to release memory allocated for information which reference for record.

4.3.3.3 Creation of records about the device with specified IP-address

Format: int32_t E502_MakeDevRecordByIpAddr (t_x502_devrec *devrec, uint32_t ip_addr, uint32_t flags, uint32_t tout)
Description: This function initializes record about device connected via Ethernet interface with specified IPv4 address. This function only creates record but does not check availability of corresponding device. Connection to module is executed similar to other records via X502_OpenByDevRecord().
Parameters: devrec — Pointer to device record that shall be created and filled in with required parameters. ip_addr — IPv4 address of module in form of 32-bit word (the same as parameter ip_addr of function E502_OpenByIpAddr()). flags — Flags. Reserve, always shall be 0. tout — Time for setting the connection in ms. This tome is saved in record and used during further call of X502_OpenByDevRecord(). If connection can not be completed within this time, function X502_OpenByDevRecord() will return error.
Returned value: Error code

4.3.3.4 Installation of TCP-port of controlling connection for record about the device

Format: int32_t E502_EthDevRecordSetCmdPort (t_x502_devrec *devrec, uint16_t cmd_port)
Description: This function allows to change TCP-port of controlling connection of E502 module. It can be required if module E502 and host from which the connection should be established are in different networks and address of E502 module is not available from host network. In this case setting of ports forwarding on router is required and if more than one such E502 module is available, since all connections go with router, these modules can be differentiated only due to TCP-port, if set different ports during forwarding. In this case in addition to controlling connection port it is required to change the port of connection for data transfer by calling E502_EthDevRecordSetDataPort(). This function shall be called for record created before using E502_MakeDevRecordByIpAddr() and before opening connection using

X502_OpenByDevRecord().
Parameters: devrec — Pointer to device record where it is required to change controlling TCP-port. cmd_port — New value of TCP-port for controlling connection
Returned value: Error code

4.3.3.5 Installation of TCP-port of data transfer connection for record about the device

Format: int32_t E502_EthDevRecordSetDataPort (t_x502_devrec *devrec, uint16_t data_port)
Description: Function is similar to E502_EthDevRecordSetCmdPort() but changes TCP=port for connection via which the exchange of input-output streams data is ongoing.
Parameters: devrec — Pointer to device record where it is required to change controlling TCP-port. data_port — New value of TCP-port for data transfer connection
Returned value: Error code

4.3.3.6 Creation of record about the device due to handle of network service

Format: int32_t E502_MakeDevRecordByEthSvc (t_x502_devrec *devrec, t_e502_eth_svc_record_hnd svc, uint32_t flags, uint32_t tout)
Description: This function initializes record about device connected via Ethernet interface corresponding network service indicated by latest transfered network service handle. This handle can be received using function of network services search corresponding to E502 modules in local network. This function only creates record but does not check availability of corresponding device. Connection to module is executed similar to other records via X502_OpenByDevRecord(). All required information from network service handle is saved in record about device, in other words after calling this function, if needed, the handle of network service can be immediately released using E502_EthSvcRecordFree().
Parameters: devrec — Pointer to device record that shall be created and filled in with required parameters. svc — Handle of network service received using E502_EthSvcBrowseGetEvent(). flags — Flags. Reserve, always shall be 0. tout — Time for setting the connection in ms. This tome is saved in record and used during further call of X502_OpenByDevRecord(). If connection can not be completed within this time, function X502_OpenByDevRecord() will return error.
Returned value: Error code

4.3.3.7 Open connection with the module due to record about the device.

Format: int32_t X502_OpenByDevRecord (t_x502_hnd hnd, const t_x502_devrec *devrec)
Description: Function sets the connection with E502 or L502 module due to record about this device. The required actions depend on which device connected via which interface the record is referred to. The records themselves are created by special functions (special for each type of module and interface of connection) and shall not be changed by user manually.
Parameters: hnd — Module handle. devrec — Record about the device containing required information for establishment of connection with it
Returned value: Error code.

4.3.3.8 Release of records about the devices

Format: int32_t X502_FreeDevRecordList (t_x502_devrec *list, uint32_t size)
Description: Function cleans resources allocated during initialization of record about device for information required to open the device. This function shall be called after initialization of record about the device by one of corresponding functions when the record is not already needed. After setting the connection with device via X502_OpenByDevRecord() record is not used in future and can be, if required, immediately released without closing the connection with device. Function can clean several records from array (if one is cleaned, it is allowed to specify 1 as size).
Parameters: list — Array of records about device or pointer to the only record which resources shall be released size — Amount of records in array
Returned value: Error code.

4.3.4 Module setting change functions

4.3.4.1 Transfer of specified settings to the module.

Format: int32_t X502_Configure (t_x502_hnd hnd, uint32_t flags)
Description: Function performs writing of current settings (which have been specified using functions X502_SetXXX) to module. Shall be called prior to data stream start.
Parameters: hnd — Module handle. flags — Flags (reserve - shall be equal to 0).
Returned value: Error code.

4.3.4.2 Logic channel parameters setting up.

Format: int32_t X502_SetLChannel (t_x502_hnd hnd, uint32_t lch, uint32_t phy_ch, uint32_t mode, uint32_t range, uint32_t avg)
Description: Function specifies parameters of set logic channel in ADC logic table.
Parameters: hnd — Module handle. lch — Number of logic channel. (from 0 to X502_LTABLE_MAX_CH_CNT-1) phy_ch — Number of physical ADC channel starting with 0 (0-15 for differential mode, 0-31 for mode with common ground) mode — Mode of ADC channel measurement (value of type t_x502_lch_mode) range — Range of channel measurement (value of type t_x502_adc_range) avg — Averaging factor for channel. Zero value corresponds to coefficient value specified by library. For explicit set of averaging factor it is required to send value from 1 (no averaging) to X502_LCH_AVG_SIZE_MAX. In case if value of averaging exceeds frequency divider, this value will be corrected
Returned value: Error code.

4.3.4.3 Setting up of logic channels number.

Format: int32_t X502_SetLChannelCount (t_x502_hnd hnd, uint32_t lch_cnt)
Description: Function specifies amount of logic channels in ADC logic table.
Parameters: hnd — Module handle lch_cnt — Amount of logic channels (from 1 to X502_LTABLE_MAX_CH_CNT)
Returned value: Error code

4.3.4.4 Receiving of logic channels number.

Format: int32_t X502_GetLChannelCount (t_x502_hnd hnd, uint32_t *lch_cnt)
Description: Function returns specified earlier using X502_SetLChannelCount() amount of logic channels in ADC controlling table.
Parameters: hnd — Module handle lch_cnt — Amount of logic channels
Returned value: Error code

4.3.4.5 Setting of collection frequency divider for ADC.

Format: int32_t X502_SetAdcFreqDivider (t_x502_hnd hnd, uint32_t adc_freq_div)
Description: ADC collection frequency is the result of division of reference frequency of synchronization (external and internal) by divider specified by this function. Alternative of this function is X502_SetAdcFreq() calculating this divider based on transfered required ADC collection frequency.
Parameters: hnd — Module handle. adc_freq_div — Divider of ADC frequency (from 1 to X502_ADC_FREQ_DIV_MAX).
Returned value: Error code.

4.3.4.6 Setting value of inter-frame delay for ADC.

Format: int32_t X502_SetAdcInterframeDelay (t_x502_hnd hnd, uint32_t delay)
Description: Function sets inter-frame delay for ADC, in other words amount of periods of reference frequency of synchronization that will be ignored after measurement of last channel of logic table till measurement corresponding to first logic channel of following frame. Alternative can be function X502_SetAdcFreq() calculating value of inter-frame delay due to specified parameters of collection frequency and frames spacing frequency (collection frequency to logic channel).
Parameters: hnd — Module handle. delay — Value of inter-frame delay (from 0 to X502_ADC_INTERFRAME_DELAY_MAX)
Returned value: Error code.

4.3.4.7 Setting divider of frequency of synchronous input from digital lines.

Format: int32_t X502_SetDinFreqDivider (t_x502_hnd hnd, uint32_t din_freq_div)
Description: Frequency of synchronous input of data from digital inputs is the result of division of reference synchronization frequency by divider specifying by this function. Alternative of this function is X502_SetDinFreq() calculating this divider based on transfered required frequency of synchronous input from digital lines.
Parameters: hnd — Module handle. din_freq_div — Divider of synchronous input frequency from digital lines (from 1 to X502_DIN_FREQ_DIV_MAX).
Returned value: Error code.

4.3.4.8 Setting frequency divider of synchronous output.

Format: int32_t X502_SetOutFreqDivider (t_x502_hnd hnd, uint32_t out_freq_div)
Description: Frequency of synchronous output of data is the result of division of reference synchronization frequency by divider specifying by this function. Common output frequency is used for each DAC channel and for digital lines (output is performed in parallel). Output frequency can not be more than half of reference frequency. Alternative of this function is X502_SetOutFreq() calculating this divider based on transfered required synchronous output frequency. For module L502in order to have possibility to set divider different from X502_OUT_FREQ_DIV_DEFAULT it is required to update FPGA firmware up to version 0.5 or above. For E502 module this possibility is always supported. You can check if this possibility is available using function X502_CheckFeature().
Parameters: hnd — Module handle. out_freq_div — Divider of synchronous output frequency (from X502_OUT_FREQ_DIV_MIN to X502_OUT_FREQ_DIV_MAX).
Returned value: Error code.

4.3.4.9 Setting ADC collection frequency.

Format: int32_t X502_SetAdcFreq (t_x502_hnd hnd, double *f_acq, double *f_frame)
Description: Function selects divider of ADC frequency such that received frequency of collection was the closest to specified in parameter f_acq. Function returns in this parameter actual frequency that has been specified. Moreover, function can select value of inter-frame delay such that frames sequence frequency (collection frequency to logic channel) was closer to specified value. For

this it is required to send the needed value in variable f_frame (upon completion it will have value of specified frequency). If zero pointer is sent as f_frame, zero inter-frame delay will be set. If it is required to change value of reference frequency, this function shall be called after X502_SetSyncMode() and X502_SetRefFreq() / X502_SetExtRefFreqValue() otherwise the received dividers will be providing incorrect frequency value. If frames frequency is set, function shall be called after setting the required amount of logic channels on control table using X502_SetLChannelCount(). When using external reference frequency of synchronization this function will provide correct result only if this external frequency corresponds to value specified using X502_SetRefFreq().
Parameters: hnd — Module handle. f_acq — At input receives the required value of ADC collection frequency in Hertz. At output it returns actual set value of frequency. f_frame — At input it receives the required value of frames collection frequency (collection frequency to logic channel) of ADC in Hertz. At output it returns actually set value. If zero pointer is sent, it specifies maximum frequency of frames collection (zero inter-frame delay).
Returned value: Error code.

4.3.4.10 Setting frequency of synchronous input from digital inputs.

Format: int32_t X502_SetDinFreq (t_x502_hnd hnd, double *f_din)
Description: Function selects divider of frequency of values input from digital inputs such that received frequency of input was the closest to specified one. Function returns in this parameter actual frequency that has been specified. If it is required to change value of reference synchronization frequency, this function shall be called after X502_SetSyncMode() and X502_SetRefFreq()/ X502_SetExtRefFreqValue() otherwise the received divider will be providing incorrect frequency value. When using external reference frequency of synchronization this function will provide correct result only if this external frequency corresponds to value specified using X502_SetRefFreq().
Parameters: hnd — Module handle. f_din — At input it receives required value of frequency of input from digital inputs in Hertz. At output it returns actually set frequency value.
Returned value: Error code.

4.3.4.11 Setting synchronous output frequency.

Format: int32_t X502_SetOutFreq (t_x502_hnd hnd, double *f_dout)
Description: Function selects divider of synchronous output frequency such that received frequency was the closest to specified one. Function returns in this parameter actual frequency that has been specified. If it is required to change value of reference synchronization frequency, this function shall be called after X502_SetSyncMode() and X502_SetRefFreq() / X502_SetExtRefFreqValue() otherwise the received divider will be providing incorrect frequency value. When using external reference frequency of synchronization this function will provide correct result only if this external frequency corresponds to value specified using X502_SetRefFreq(). For module L502in order to have possibility to set frequency different from reference frequency divided by X502_OUT_FREQ_DIV_DEFAULT it is required to update FPGA firmware up to version 0.5 or above. For E502 module this possibility is always supported. You can check if this possibility is available using function X502_CheckFeature().
Parameters: hnd — Module handle. f_dout — At input gets required value of synchronous output frequency in Hertz. At output it returns actually set frequency value.
Returned value: Error code.

4.3.4.12 Receive current values of ADC collection frequency

Format: int32_t X502_GetAdcFreq (t_x502_hnd hnd, double *f_acq, double *f_frame)
Description: Function returns current specified for module values of collection frequency and ADC frames frequency(frequency for logic channel in Hertz which have been set before using X502_SetAdcFreq() or using functions X502_SetAdcFreqDivider() / X502_SetAdcInterframeDelay().
Parameters: hnd — Module handle. f_acq — If not NULL, current value of ADC collection frequency is returned at output. f_frame — If not NULL,current value of ADC frames frequency is returned at output.

4.3.4.13 Setting up of internal reference synchronization frequency.

Format: int32_t X502_SetRefFreq (t_x502_hnd hnd, uint32_t freq)
Description: Function sets value of internal reference synchronization frequency of synchronous input/output by means of dividing to specified divider.

This function at internal reference frequency selects one of two available frequencies of 2 MHz or 1.5 MHz (2 MHz is default value) to specify which constants from t_x502_ref_freq have been implemented. When using external reference frequency you should use X502_SetExtRefFreqValue(). For module E502 output to DAC at reference frequency of 1.5 MHz operates only for version of firmware PLDA 1 and above.
Parameters: hnd — Module handle. freq — Value from t_x502_ref_freq specifying selected reference frequency.
Returned value: Error code.

4.3.4.14 Setting up of external reference synchronization frequency.

Format: int32_t X502_SetExtRefFreqValue (t_x502_hnd hnd, double freq)
Description: In case of external reference frequency (calling X502_SetSyncMode() with value different from X502_SYNC_INTERNAL) this function allows to set frequency of external reference frequency that can be any but not more than 1.5 MHz. This function does not influence on settings of the module itself, but set of correct value allows to specify the required frequency of data collection by functions X502_SetAdcFreq(), X502_SetDinFreq() and X502_SetOutFreq() and calculate correctly default values for buffer size and step for data transfer between module and PC. This function is available in library of version 1.1.4 or later.
Parameters: hnd — Module handle. freq — Value of external reference frequency in Hz.
Returned value: Error code.

4.3.4.15 Receiving reference synchronization frequency value.

Format: int32_t X502_GetRefFreqValue (t_x502_hnd hnd, double *freq)
Description: This function returns current value of synchronization reference frequency used by library in functions X502_SetAdcFreq(), X502_SetDinFreq() and X502_SetOutFreq() as well as during calculation of parameters of data transfer between module and PC. At internal reference frequency value specified by X502_SetRefFreq() (1.5 or 2 MHz) is used, at external— frequency set by fucntionX502_SetExtRefFreqValue(). This function is available in library of version 1.1.4 or later.
Parameters: hnd — Module handle. freq — Value of external reference frequency in Hz.
Returned value: Error code.

4.3.4.16 Setting up of synchronization frequency start mode.

Format: int32_t X502_SetSyncMode (t_x502_hnd hnd, uint32_t sync_mode)
Description: Function sets the source of synchronization reference frequency generation-module itself or it will use external signal. In mode X502_SYNC_INTERNAL module itself will generate synchronization frequency at frequency specified by X502_SetRefFreq(). In this case start of generation will be conducted upon calling X502_StreamsStart() or according to condition specified in X502_SetSyncStartMode(), and stop according to X502_StreamsStop(). In the rest modes the collection is carried according to external synchronization signal.
Parameters: hnd — Module handle. sync_mode — Value from t_x502_sync_mode determining the source of synchronization frequency.
Returned value: Error code.

4.3.4.17 Setting up of synchronization frequency start mode.

Format: int32_t X502_SetSyncStartMode (t_x502_hnd hnd, uint32_t sync_start_mode)
Description: Function sets condition of synchronous data input/output start. If using X502_SetSyncMode() synchronization mode X502_SYNC_INTERNAL is set, module starts generating synchronization frequency according to condition specified by function, otherwise according to specified condition module starts using externally set synchronization frequency (i. e. synchronization signal will be bypassed at specified input until the condition is executed). Modes of synchronization start condition specifying have the same values as modes of frequency setting (see type t_x502_sync_mode). In case of X502_SYNC_INTERNAL start is performed when executing function X502_StreamsStart(), otherwise- after execution of X502_StreamsStart() module starts waiting for condition specified by this function. In other words, when setting external sources of synchronization anyway it is required to call X502_StreamsStart().
Parameters: hnd — Module handle. sync_start_mode — Value from t_x502_sync_mode determining condition of synchronization frequency start.
Returned value: Error code.

4.3.4.18 Set up the module operational mode.

Format: int32_t X502_SetMode (t_x502_hnd hnd, uint32_t mode)
Description:

Function specifies module operation mode that determines whether data streams will be processed by FPGA or BlackFin signaling processor. When power is supplied FPGA is always controlling module. After loading of firmware using X502_BfLoadFirmware() module switches into signaling processor control mode. This function can be used for manual set of mode, for example, to return into FPGA control mode or to switch into signaling processor control mode if firmware has already been downloaded (for example, via JTAG interface during debugging).
Parameters: hnd — Module handle. mode — Module operation mode from t_x502_mode.
Returned value: Error code.

4.3.4.19 Receiving current module operational mode.

Format: int32_t X502_GetMode (t_x502_hnd hnd, uint32_t *mode)
Description: Function returns current module operating mode.
Parameters: hnd — module handle. mode — This parameter returns current module operating mode (from t_x502_mode).
Returned value: Error code.

4.3.4.20 Set up coefficients for ADC values calibration.

Format: int32_t X502_SetAdcCoef (t_x502_hnd hnd, uint32_t range, double k, double offs)
Description: Function writes coefficients for ADC values calibration into FPGA. When opening module library reads calibration coefficients from protected area of module Flash-memory and writes them into FPGA for calibration performance in process. Resulting ADC value is calculated according to formula (val+offs)*k, where val — not calibrated value. This function allows to change used coefficients before data collection start. In this case only current coefficients are changed and factory calibration coefficients from Flash-memory keep their values and during following opening will be restored.
Parameters: hnd — Module handle. range — ADC range (from t_x502_adc_range). k — Set value of scale coefficient. offs — Set value of zero shift.
Returned value: Error code.

4.3.4.21 Receiving current calibration coefficients of ADC.

Format: int32_t X502_GetAdcCoef (t_x502_hnd hnd, uint32_t range, double *k, double *offs)
Description: Function returns current calibration coefficients for specified ADC measurement range. These coefficients can be different from factory values saved in Flash-memory of module, for example, if user used X502_SetAdcCoef() to set his coefficients.
Parameters: hnd — Module handle. range — ADC range (from t_x502_adc_range). k — This variable returns current scale range. offs — This variable returns current zero shift.
Returned value: Error code.

4.3.4.22 Set up coefficients for DAC values calibration.

Format: int32_t X502_SetDacCoef (t_x502_hnd hnd, uint32_t ch, double k, double offs)
Description: Function specifies calibration coefficients for specified ADC channel that will be used by functions x502api for calibration of entered values of DAC if flag X502_DAC_FLAGS_CALIBR is given. Calibrated value of DAC in codes is as (val+offs)*k, where val — not calibrated value (in codes). When opening module library reads calibration coefficients from protected area of module Flash-memory and use them. This function is required only if user wants to apply his own coefficients. In this case it does not change values in Flash-memory, i. e. in case of following module opening coefficients will be restored from Flash-memory again.
Parameters: hnd — Module handle. ch — DAC channel (from t_x502_dac_ch). k — Set value of scale coefficient. offs — Set value of zero shift.
Returned value: Error code.

4.3.4.23 Receiving current calibration coefficients of DAC.

Format: int32_t X502_GetDacCoef (t_x502_hnd hnd, uint32_t ch, double *k, double *offs)
Description: Function returns current calibration coefficients for specified DAC channel. These coefficients can be different from factory values saved in Flash-memory of module,

for example, if user used X502_SetAdcCoef() to set his coefficients.
Parameters: hnd — Module handle. ch — DAC channel (fromt_x502_dac_ch). k — This variable returns current scale coefficient. offs — This variable returns current zero shift.
Returned value: Error code.

4.3.4.24 Calculation of ADC collection frequency

Format: int32_t X502_CalcAdcFreq (double ref_freq, uint32_t lch_cnt, double *f_acq, double *f_frame, uint32_t *result_freq_div, uint32_t *result_frame_delay)
Description: Based on specified parameters function selects ADC frequency divider and value of inter-frame delay such that obtained frequency will be close to specified ones and returns received values of frequencies. As compared to X502_SetAdcFreq() this function is meant for getting of corrected frequency without application of module handle and only calculates resulting parameters not changing settings.
Parameters: ref_freq — Value of reference frequency in Hz (external or internal)) lch_cnt — Amount of logic channels that will be used. It is required for calculation of inter-frame delay. If null pointer is sent as f_frame, it can be equal to zero. f_acq — At input receives the required value of ADC collection frequency in Hertz. At output it returns the calculated value of frequency that can be established. f_frame — At input it receives the required value of frames collection frequency (collection frequency to logic channel) of ADC in Hertz. At input it returns the calculated value. If null pointer is sent, the delay will not be calculated. If value less or equal to zero is sent, maximum frames frequency will be calculated (with zero inter-frame delay). result_freq_div — This parameter returns the calculated value of ADC frequency divider. Null pointer can be sent if this value is not explicitly required to be known. result_frame_delay — This parameter returns the calculated value of inter-frame delay. Null pointer can be sent if this value is not explicitly required to be known.
Returned value: Error code.

4.3.4.25 Calculation of synchronous entry frequency from digital inputs.

Format: int32_t X502_CalcDinFreq (double ref_freq, double *f_din, uint32_t *result_freq_div)
Description: Based on specified parameters function selects divider of frequency of values entry from digital inputs such that received input frequency is closer to specified and returns the received frequency value.

As compared to X502_SetDinFreq() this function is meant for receiving of corrected frequency without use of module handle and only calculates the resulting parameters not changing the settings.
Parameters: ref_freq — Value reference frequency in Hz (external or internal) f_din — At input it receives the required value of entry frequency from digital inputs in Hertz. At output it returns the calculated value of frequency that can be established. result_freq_div — This parameter returns the calculated value of frequency divider of synchronous input of digital lines. Null pointer can be sent if this value is not explicitly required to be known.
Returned value: Error code.

4.3.4.26 Calculation of synchronous output frequency.

Format: int32_t X502_CalcOutFreq (double ref_freq, double *f_dout, uint32_t *result_freq_div)
Description: Based on specified parameters function selects divider of frequency of synchronous output such that received frequency is closer to specified one and returns the received frequency value. As compared to X502_SetOutFreq() this function is meant for receiving of corrected frequency without use of module handle and only calculates the resulting parameters not changing the settings. Function suggests that module supports change of input frequency (see requirements to module for this in function description X502_SetOutFreq()).
Parameters: ref_freq — Value reference frequency in Hz (external or internal) f_dout — At input it receives the required value of input frequency in Hertz. At output it returns the calculated value of frequency that can be established. result_freq_div — This parameter returns the calculated value of synchronous output frequency divider. Null pointer can be sent if this value is not explicitly required to be known.
Returned value: Error code.

4.3.5 Functions of asynchronous input-output

4.3.5.1 Asynchronous data output to DAC channel.

Format: int32_t X502_AsyncOutDac (t_x502_hnd hnd, uint32_t ch, double data, uint32_t flags)
Description: Function sends the specified value to specified DAC channel. Value can be specified in codes as well as in Volts, and calibration coefficients can be applied to it (defined by flags). Function can be called whether when synchronous collection is not running or when data collection is running if synchronous output via this DAC channel is not permitted.
Parameters: hnd — Module handle. ch — Number of DAC channel (from t_x502_dac_ch). data — Output data to DAC (in codes or volts) flags — Flags from t_x502_dacout_flags.
Returned value: Error code.

4.3.5.2 Asynchronous data output to digital outputs.

Format: int32_t X502_AsyncOutDig (t_x502_hnd hnd, uint32_t val, uint32_t msk)
Description: Function sends the specified value to digital outputs of module. Format of value is the same as X502_PrepareData() - output value is specified in low 16 bits, and in high bits- flags (using which one of parts can be transfered to third status). Function can be called whether when synchronous collection is not running or when data collection is running if synchronous output via digital lines is not permitted. You can use mask to send only to part of outputs left the rest unchanged, but it should be considered that after opening of connection with module it is first required to make output to all lines after that you can use mask in further calls.
Parameters: hnd — Module handle. val — Low-order half- output value, high-order- flags from t_x502_digout_word_flags. msk — Mask- bits specified in mask will not be changed from the previous output status (relates to high part val as well)
Returned value: Error code.

4.3.5.3 Asynchronous entry of values from digital inputs.

Format: int32_t X502_AsyncInDig (t_x502_hnd hnd, uint32_t *din)
Description: Function reads current value of digital inputs. In this case synchronous collection of digital inputs shall not be running (the stream X502_STREAM_DIN) is not permitted. Since module E502/L502 does not support asynchronous input by hardware, if at the moment of calling of this function synchronous input/output is not running using X502_StreamsStart(), this function starts synchronous collection for a period of performance and stops it as soon as one new value of digital inputs is received.
Parameters: hnd — Module handle. din — Upon successful fulfillment this variable returns current status of digital inputs. Low 18 bits are valid, high 14- reserve. Part of bits in this case are joined with synchronization lines and join depends on module type. Detailed information is given in section with differences in modules E502 and L502. Reserve bits can be used in further versions they should not be considered to be equal to zero all the time!
Returned value: Error code.

4.3.5.4 Asynchronous input of one ADC frame.

Format: int32_t X502_AsyncGetAdcFrame (t_x502_hnd hnd, uint32_t flags, uint32_t tout, double *data)
Description: Functions performs single-shot input of frame in compliance with pre-specified logic table. ADC collection frequency corresponds to frequency specified using X502_SetAdcFreq(). Frames spacing frequency does not matter. The frame itself is entered synchronously but upon sequential call of X502_AsyncGetAdcFrame() to measure several frames the delay between these frames is not defined. Function also performs processing of data received from ADC the same as X502_ProcessAdcData() and receives set of flags similar to X502_ProcessAdcData(). For operation of this function synchronous ADC and digital lines input shall not be permitted. Since asynchronous input is not available in board, this function in case of not running stream starts it inside itself, receives one data frame and after this stops synchronous collection.
Parameters: hnd — Module handle. flags — flags from t_x502_proc_flags tout — Time-out for function fulfillment in ms data — Array where in case of success ADC frame samples will be returned to. Shall have size sufficient for storage of samples of double type in the amount equal to number of established logic channels in ADC control table.
Returned value: Error code.

4.3.6 Functions for working with synchronous stream input-output

4.3.6.1 Permission of synchronous streams for input/output.

Format: int32_t X502_StreamsEnable (t_x502_hnd hnd, uint32_t streams)
Description: Function permits reception/transfer for specified streams. Stream that are not specified save their permitted or prohibited status. Can be called before X502_Configure() as well as after it. Permitted streams are specified as a rule before calling X502_StreamsStart(). If required, in some cases it is possible to change the structure of permitted streams during running data collection, but if these streams are of great difference in frequency, the buffer values calculated by library as well as interrupt steps can be not suitable for changed values (see Buffer size and step for synchronous mode)
Parameters: hnd — Module handle. streams — Set of flags t_x502_streams specifying which streams shall be permitted.
Returned value: Error code.

4.3.6.2 Inhibit of synchronous streams for input/output.

Format: int32_t X502_StreamsDisable (t_x502_hnd hnd, uint32_t streams)
Description: Function inhibits transfer of synchronous data for specified streams. Streams that are not specified keep their permitted or prohibited status. Function opposite by implication to X502_StreamsEnable().
Parameters: hnd — Module handle. streams — Set of flags t_x502_streams specifying which streams shall be inhibited.
Returned value: Error code.

4.3.6.3 Receive value, which synchronous streams are permitted.

Format: int32_t X502_GetEnabledStreams (t_x502_hnd hnd, uint32_t *streams)
Description: Function allows to get set of flags specifying which synchronous streams are permitted now.
Parameters: hnd — Module handle. streams — Set of flags t_x502_streams specifying which streams are permitted now.
Returned value: Error code.

4.3.6.4 Start up of synchronous input/output streams.

Format: int32_t X502_StreamsStart (t_x502_hnd hnd)
Description: Function of synchronous data streams start. All synchronous streams are timed from common reference frequency. If internal start of synchronization has been set, streams synchronization will start upon performance of this function, otherwise under this function module switches to mode of waiting for external attribute of initial synchronization. Moreover, function performs initialization of DMA channel for data input from board if ADC stream or synchronous input of digital lines has been permitted and initialization of DMA channel for output if there was at least one stream for output permitted but function X502_PreloadStart() has not been called (but in this case the beginning of output disagrees with beginning of input).
Parameters: hnd — Module handle.
Returned value: Error code.

4.3.6.5 Stop of synchronous input/output streams.

Format: int32_t X502_StreamsStop (t_x502_hnd hnd)
Description: Function of synchronous streams of data input/output stop. After fulfillment of this function module completes generation of reference synchronization frequency (or use external synchronization frequency) and stops synchronous data transfer.
Parameters: hnd — Module handle.
Returned value: Error code.

4.3.6.6 Checking if synchronous input/output has been started up.

Format: int32_t X502_IsRunning (t_x502_hnd hnd)
Description: Function checks if the synchronous input output is strarted using X502_StreamsStart() or any internal logic in BlackFin firmware. If data acquisition is not running, the function returns the error X502_ERR_STREAM_IS_NOT_RUNNING, if running, the null error code
Parameters: hnd — Module handle.
Returned value: Error code.

4.3.6.7 Reading ADC data and digital inputs from module

Format: int32_t X502_Recv (t_x502_hnd hnd, uint32_t *buf, uint32_t size, uint32_t tout)

Description: Function reads data from module that have been received in intermediate buffer in driver or library. Function receives samples in special index format containing information on data (values of digital inputs or ADC samples) and additional information for ADC (channel number, mode). For parsing of received samples function X502_ProcessData() is used. If buffer now has less samples than it was requested, function will wait till the specified amount of data arrives or till the moment of time-out completion. In the latter case function returns as many samples as were in buffer upon time-out completion. Amount of ready for reading samples in driver buffer can be, if required, known using function X502_GetRecvReadyCount(). Before calling X502_Recv() synchronous stream of data collection shall be already started usingX502_StreamsStart().
Parameters: hnd — Module handle. buf — Buffer where samples are saved. size — Number of readable samples (32-bit words). tout — Time-out for data reception in ms.
Returned value: If < 0 - error code. If >= 0 - number of read words.

4.3.6.8 Transfer of DAC stream data and digital outputs to module.

Format: int32_t X502_Send (t_x502_hnd hnd, const uint32_t *buf, uint32_t size, uint32_t tout)
Description: Function writes data for transfer to intermediate buffer after this the data will be sent to module. Data shall be in special format that defines what are the data (digital outputs, DAC1 channel or DAC2 channel). To prepare data in required format you can using X502_PrepareData(). If intermediate buffer for transfer is full, function will wait till the moment it is freed or until the time-out is completed. Amount of free space in buffer can be, if required, known using function X502_GetSendReadyCount(). The return means that data are written into intermediate buffer and not that they already came to module and output. Before calling this function there shall be data pre-loading for output started using X502_PreloadStart().
Parameters: hnd — Module handle. buf — Buffer with words required to be transferred to module size — Number of transferrable samples (32-bit words). tout — Time-out for data transfer (to driver buffer) in ms.
Returned value: If < 0 - error code. If >= 0 - number of written words.

4.3.6.9 Processing of ADC samples received from module

Format: int32_t X502_ProcessAdcData (t_x502_hnd hnd, const uint32_t *src, double *dest, uint32_t *size, uint32_t flags)
Description: Function performs processing of ADC samples read using X502_Recv(). Function checks the service information from input array and converts samples into codes or volts (if flag X502_PROC_FLAGS_VOLT) is specified. Function is used when synchronous input from digital lines is not started and ADC samples are the only data coming from module (if there are other data in received stream- they will be rejected). If synchronous input from digital lines is started, you should use X502_ProcessData() that allocates data from digital lines into separate array.
Parameters: hnd — Module handle. src — Input array of samples received using X502_Recv(). dest — Array where converted data from ADC will be saved. size — At input- amount of words in array scr, at output- amount of saved converted values in array dest flags — Set of flags from t_x502_proc_flags
Returned value: Error code.

4.3.6.10 Processing of data received from the module.

Format: int32_t X502_ProcessData (t_x502_hnd hnd, const uint32_t *src, uint32_t size, uint32_t flags, double *adc_data, uint32_t *adc_data_size, uint32_t *din_data, uint32_t *din_data_size)
Description: Function performs processing of data read using X502_Recv(). Function checks service information from input array, divides data into two arrays- data from ADC converted into double type ans data from synchronous digital input. Data from ADC also can be converted into volts. In this case ADC data come from module already calibrated using calibration coefficients because calibration is conducted by hardware. If ADC data are not converted into Volts and factory calibration coefficients has not been changed, the returned value equal to X502_ADC_SCALE_CODE_MAX corresponds to voltage equal to maximum for used range. Besides function examines messages transfered in data stream (for example, message on buffer overflow).
Parameters: hnd — Module handle. src — Input array of samples received using X502_Recv(). size — Amount of samples (32-bit words) in array src. flags — Set of flags from t_x502_proc_flags controlling function behavior. There can be specified several flags through logic "OR". adc_data — Array where data from ADC converted in compliance with specified flags will be saved. Can be NULL if data from ADC shall not be saved (then

adc_data_size shall be NULL or variable transfers size 0).
adc_data_size — At input this parameter transfers buffer size adc_data. If data from ADC in input array will be more than adc_data_size, only first adc_data_size of samples are saved in adc_data. At output upon successful completion of function this variable has amount of saved ADC samples.
Pointer can be equal to NULL if adc_data = NULL
din_data — Array where reports from synchronous digital input will be saved. Each word corresponds to status of all digital inputs in format described in function X502_AsyncInDig().
din_data_size — Similar to parameter adc_data_size this parameter transfers buffer size din_data in samples, and at output amount of actually saved samples from digital lines is saved. Can be NULL if din_data = NULL.

Returned value: Error code.

4.3.6.11 Processing of data received from module with user data.

Format: int32_t X502_ProcessDataWithUserExt (t_x502_hnd hnd, const uint32_t *src, uint32_t size, uint32_t flags, double *adc_data, uint32_t *adc_data_size, uint32_t *din_data, uint32_t *din_data_size, uint32_t *usr_data, uint32_t *usr_data_size)

Description:
Function is similar to X502_ProcessData() but allows allocate user data from stream. User data are considered to be all samples that are not ADC data, data of digital input or messages. User data are placed without changing into array usr_data (if it is not equal to zero). This function is meant mainly for programmers who will use modified firmware of BlackFin signaling processor.

Parameters:
hnd — Module handle.
src — Input array of samples received using X502_Recv().
size — Number of samples (32-bit words) in array src. flags — Set of flags from t_x502_proc_flags.
adc_data — Array where data from ADC are saved (see X502_ProcessData()).
adc_data_size — see X502_ProcessData()
din_data — Array where reports from synchronous digital input are saved. See X502_ProcessData().
din_data_size — see X502_ProcessData().
usr_data — Array where user data are saved without changing their format.
usr_data_size — This parameter transfers the size of buffer usr_data and the number of actually saved user data samples is saved at output. Can be NULL only if usr_data = NULL.

Returned value: Error code.

4.3.6.12 Data preparation for output to module.

Format: int32_t X502_PrepareData (t_x502_hnd hnd, const double *dac1, const double *dac2, const uint32_t *digout, uint32_t size, int32_t flags, uint32_t *out_buf)

Description: Function receives data from three arrays- data for digital outputs, samples of first and second DAC channels. Null/zero pointer can be transfered as array if data from this source are not required. All used arrays shall be of equal size and function mixes them uniformly into common stream converting into format required for module. Output array shall contain n*size samples, where n- amount of used input arrays (from 1 to 3). Values of digital outputs are 32-bit words, low 16-bit of which determine values of outputs and high- flags from t_x502_digout_word_flagsthat can be used, in particular, for transfer of one (or both) of halves of outputs into third status. Codes as well as Volts can be used as DAC values depending on transfered flags and calibration coefficients can be applied to output values. If DAC codes are used including calibration, code X502_DAC_SCALE_CODE_MAX determines code corresponding to +5V.
Parameters: hnd — Module handle. dac1 — Input array of samples of DAC first channel or NULL, if not used. dac2 — Input array of samples of second DAC channel or NULL if not used. digout — Input array with values of digital outputs or NULL if not used. size — Size of each of used input arrays. flags — Flags controlling operation of function of t_x502_dacout_flags. out_buf — Output array where formed samples will be saved. Shall be of size n*size (n - amount of used input arrays).
Returned value: Error code.

4.3.6.13 Receive number of samples in buffer of stream to input.

Format: int32_t X502_GetRecvReadyCount (t_x502_hnd hnd, uint32_t *rdy_cnt)
Description: Function returns number of samples received from module to internal buffer and ready for reading using X502_Recv() In other words if in X502_Recv() transfer value returned by function, X502_Recv() will return this amount of data without wait(because they will be in buffer already). When working through Ethernet this function returns correct value only for Windows OS.
Parameters: hnd — Module handle. rdy_cnt — Number of samples ready for reception.
Returned value: Error code.

4.3.6.14 Receive size of free space in buffer of stream to output.

Format: int32_t X502_GetSendReadyCount (t_x502_hnd hnd, uint32_t *rdy_cnt)
Description:

Function returns amount of samples corresponding to free space in buffer for transfer to module. This amount of samples X502_Send() without waiting. This function is not implemented when working via Ethernet interface (TCP).
Parameters: hnd — Module handle. rdy_cnt — Number of words corresponds to free space in buffer for transfer.
Returned value: Error code.

4.3.6.15 Receive number of following expected logic channel of ADC for processing.

Format: int32_t X502_GetNextExpectedLchNum (t_x502_hnd hnd, uint32_t *lch)
Description: Function returns number of logic channel of ADC that shall be processed first upon following call of X502_ProcessData()/ X502_ProcessAdcData() in case if data stream is continuous. Actually this is number of logic channel following the logic channel of last processed before this ADC sample. It can be used when processing blocks of data not whole multiple amount of frames. If before X502_ProcessData() you call this function. it will return number of logic channel corresponding to first sample of ADC processed by following call of X502_ProcessData(). For example, if there are 7 logic channels specified and 7-fold amount of samples is transfered to X502_ProcessData() , the following call of X502_GetNextExpectedLchNum() will return channel number equal to 0 (because whole number of frames is processed and beginning of frame is waited for). If in X502_ProcessData() you transfer array with 7*n + 5 ADC sample, the following expected channel will be logic channel with number 5 (channels 0,1,2,3,4 are processed from incomplete frame).
Parameters: hnd — Module handle. lch — Number of logic channel (begins with zero).
Returned value: Error code.

4.3.6.16 Beginning of preparation for synchronous data output.

Format: int32_t X502_PreloadStart (t_x502_hnd hnd)
Description: Function shall be called before starting pre-loading of stream synchronous data to output. To start output of synchronous data simultaneously with starting synchronous input by the moment of data collection start a part of data shall be already loaded to module before calling X502_StreamsStart(). This function initialises the channel of exchange for data transfer to output. After calling this function it will be possible to load a part of data to output using X502_Send().
Parameters:

hnd — Module handle.
Returned value: Error code.

4.3.6.17 Beginning of cyclic signal loading to output

Format: int32_t X502_OutCycleLoadStart (t_x502_hnd hnd, uint32_t size)
Description: Upon calling this function in driver (for L502) or in module controller memory (for E502) space for cyclic buffer to output is allocated. Should be called before loading of cyclic data using X502_Send(). For successful fulfillment there should be empty buffer (double buffering is used)- in other words function can not be called right after the previous X502_OutCycleSetup(). Besides, stream output shall not be used. For L502 the maximum buffer size is determined only by size that is allowed to be allocated by system at driver level. For E502 the size is limited by memory of in-built controller. See the details in description of differences of E502 and L502.
Parameters: hnd — Module handle. size — Number of samples in output cyclic signal in total for all used output channels.
Returned value: Error code.

4.3.6.18 Setting up of pre-loaded cyclic signal to output

Format: int32_t X502_OutCycleSetup (t_x502_hnd hnd, uint32_t flags)
Description: Upon calling this function the cyclic buffer loaded earlier becomes active. If synchronous input-output is running (through X502_StreamsStart()), due to this function signal is sent to output, otherwise output will start upon actuating of synchronous input-output. If the cyclic signal has been already sent before, the shift to new one will be executed in the end of cycle of previous signal if following flag is pointed X502_OUT_CYCLE_FLAGS_FORCE. If flag X502_OUT_CYCLE_FLAGS_WAIT_DONE is not pointed, function only gives the command for signal establishment with no wait for direct change of signal or signal loading. In particular for simultaneous start of synchronous input and output it is required to perform loading of first cyclic signal with this flag to ensure that signal is completely loaded by the moment of synchronous input-output through X502_StreamsStart(). This function shall be called only after calling X502_OutCycleLoadStart() and loading of specified amount of samples to buffer!
Parameters: hnd — Module handle. flags — Flags from t_x502_out_cycle_flags.
Returned value: Error code.

4.3.6.19 Stop of cyclic signal output

Format: int32_t X502_OutCycleStop (t_x502_hnd hnd, uint32_t flags)
Description: upon call of this function the output of pre-established cyclic signal is stopped using X502_OutCycleSetup(). The stop is carried out after output of last sample in period that allows to know which values are left at outputs. Upon call of X502_StreamsStop() (or upon inhibit of all streams to output via X502_StreamsDisable()) stop of all streams is conducted at once and exact point of stop is not known. In such case it should considered that function itself only sends request for stop by default and the actual stop will be performed later. If you call X502_StreamsStop() before completion of stop, the last sample will be unknown, i.e. it is required to wait for stop completion for this flag X502_OUT_CYCLE_FLAGS_WAIT_DONE can be used.
Parameters: hnd — Module handle. flags — Flags from t_x502_out_cycle_flags.
Returned value: Error code.

4.3.6.20 Checking if setting up or stop of cyclic signal has been completed.

Format: int32_t X502_OutCycleCheckSetupDone (t_x502_hnd hnd, uint32_t *done)
Description: Function checks if establishment of cyclic signal after call of X502_OutCycleSetup() is completed or stop of cyclic signal generation is completed after call of X502_OutCycleStop(). As per its purpose it is the same as flag X502_OUT_CYCLE_FLAGS_WAIT_DONE in above described functions but allows to perform waiting manually (with checking of other conditions). Function is available in library starting with version 1.1.2 in this case for function operation it is required to have version of ARM firmware not lower than 1.0.2 for E502 module or driver version not lower than 1.0.9 for L502. As compared to flag if these conditions are not fulfilled, function returns error X502_ERR_NOT_SUP_BY_FIRMWARE or X502_ERR_NOT_SUP_BY_DRIVER. Waiting for the completion can be required when calling X502_OutCycleSetup() during loading of first signal before calling X502_StreamsStart() if it is needed to have the output of first samples to DAC was synchronized with moment of output start, because in other case the signal loading to module can not be completed by the moment of start and signal output begins with delay (relevant to E502). In case of further calls of X502_OutCycleSetup() to change already set signal the installation is considered to be completed after the completion of signal loading and direct change of outcoming signal. This check can be used to know that the signal change has been completed and you can download the next cyclic signal. When calling X502_OutCycleStop() waiting for completion can be used before calling X502_StreamsStop() to be sure (if required) that generation has been completed exactly at the las point of loaded cyclic signal.
Parameters:

hnd — Module handle. done — 0 if there is uncompleted request for installation or cyclic signal stop, 1- in opposite case (including the case when the output of cyclic signal is not running at all)
Returned value: Error code.

4.3.6.21 Reading of output status flags

Format: int32_t X502_OutGetStatusFlags (t_x502_hnd hnd, uint32_t *status)
Description: Function reads the value of synchronous output status flags from status register. In particular per flag X502_OUT_STATUS_FLAG_BUF_WAS_EMPTY you can check if there was no buffer underrun since the moment of synchronous output start to be sure that there was no signal break due to not additionally loaded data in proper time.
Parameters: hnd — Module handle. status — Flag of status — set of bits from t_x502_out_status_flags joined through the logic "IF".
Returned value: Error code.

4.3.6.22 Setting up of buffer size for synchronous input or output.

Format: int32_t X502_SetStreamBufSize (t_x502_hnd hnd, uint32_t ch, uint32_t size)
Description: Function sets the buffer size which is used for temporary storage of data for reception or transfer. It is meant for cases when user is not satisfied the default value calculated by library.
Parameters: hnd — Module handle. ch — Specifies if the buffer size is set for input or output (value from t_x502_stream_ch). size — Buffer size in 32-bit sample
Returned value: Error code.

4.3.6.23 Setting up of step under transfer of stream to input or output.

Format: int32_t X502_SetStreamStep (t_x502_hnd hnd, uint32_t dma_ch, uint32_t step)
Description: Function sets the data transfer step(pace) (step of interruptions generation for PCI-Express or request size for USB) when transfer synchronous data stream for input or output. This function is meant for users who are not satisfied with value automatically calculate by library.
Parameters: hnd — Module handle.

dma_ch — Specifies if step for data transfer is set for input or output. (value from t_x502_stream_ch). step — Step of interruption in 32-bit samples
Returned value: Error code.

4.3.7 Functions for setting of network parameters of E502 module

4.3.7.1 Receiving current IP-address of device.

Format: int32_t E502_GetIpAddr (t_x502_hnd hnd, uint32_t *ip_addr)
Description: Function returns IP-address of device through which the connection has been established. In other words, the connection with device should already been established and it should be done via Ethernet interface.
Parameters: hnd — Device handle ip_addr — Current IPv4 address of module in form of 32-bit word (the same as parameter ip_addr of function E502_OpenByIpAddr()).
Returned value: Error code

4.3.7.2 Creation of handle of network interface configuration.

Format: t_e502_eth_config_hnd E502_EthConfigCreate (void)
Description: Creation of handle of network interface configuration. In case of successful memory allocation it initializes the handle fields by default values.
Returned value: NULL in case of error, otherwise- module handle

4.3.7.3 Release of network interface configuration handle.

Format: int32_t E502_EthConfigFree (t_e502_eth_config_hnd cfg)
Description: Release of memory dedicated for configuration handle of network interface using E502_EthConfigCreate(). After this the handle can not be used regardless of returned value!
Parameters: cfg — Configuration handle of network interface
Returned value: Error code

4.3.7.4 Reading current network configuration of interface

Format: int32_t E502_EthConfigRead (t_x502_hnd hnd, t_e502_eth_config_hnd cfg)
Description: Function reads the current parameters of interface and saves them to the structure which is pointed out by network interface configuration handle created using E502_EthConfigCreate() . Connection with device in this case shall be established but it can be established via any supported interface.

Parameters: hnd — Handle of device from which the configuration shall be read cfg — Network interface configuration handle
Returned value: Error code

4.3.7.5 Writing network configuration of interface

Format: int32_t E502_EthConfigWrite (t_x502_hnd hnd, t_e502_eth_config_hnd cfg, const char *passwd)
Description: Function sends to module network interface configuration that module shall save in its non-volatile memory. In case of successful fulfillment of this function module switches off Ethernet interface, set it up to new parameters and initialises it again, that's why if connection with device is established via network, further operation with device will be impossible- it is required to close the connection with device and establish it again. To change configuration it is required to send the password for configuration (empty string if password has not been installed). When working via USB interface the current serial number of device can be send as password (in case if the specified password has been forgotten).
Parameters: hnd — Handle of device from which the configuration shall be read cfg — Network interface configuration handle passwd — String having password for configuration change
Returned value: Error code

4.3.7.6 Copying the content of interface network configuration

Format: int32_t E502_EthConfigCopy (t_e502_eth_config_hnd src_cfg, t_e502_eth_config_hnd dst_cfg)
Description: Functions executes copying of all parameters of one created configuration to another configuration creating the complete copy.
Parameters: src_cfg — Handle of initial network configuration of interface which content shall be copied. dst_cfg — Handle of network configuration of interface where the content of initial configuration shall be copied to
Returned value: Error code

4.3.7.7 Determining if Ethernet interface is permitted.

Format: int32_t E502_EthConfigGetEnabled (t_e502_eth_config_hnd cfg, uint32_t *en)
Description: Function returns whether Ethernet interface is permitted in specified configuration.

If interface is not permitted, Ethernet controller is completely switched off.
Parameters: cfg — Configuration handle of network interface en — If interface is permitted this variable returns 1, otherwise — 0
Returned value: Error code

4.3.7.8 Ethernet interface permission.

Format: int32_t E502_EthConfigSetEnabled (t_e502_eth_config_hnd cfg, uint32_t en)
Description: Function specifies if operation via Ethernet interface is permitted. If interface is not permitted, Ethernet controller is completely switched off.
Parameters: cfg — Network interface configuration handle en — 0 means inhibit of Ethernet interface, 1 — permission
Returned value: Error code

4.3.7.9 Determining if automatic receiving of IP parameters is permitted.

Format: int32_t E502_EthConfigGetAutoIPEnabled (t_e502_eth_config_hnd cfg, uint32_t *en)
Description: Function returns whether automatic receiving of IP (IP-address, sub-network mask, gateway address) parameters is permitted using DHCP/linklocal or static specified parameters are used.
Parameters: cfg — Configuration handle of network interface en — If automatic receiving of parameters is permitted, 1 is returned, otherwise — 0
Returned value: Error code

4.3.7.10 Automatic receiving of IP parameters permission.

Format: int32_t E502_EthConfigSetAutoIPEnabled (t_e502_eth_config_hnd cfg, uint32_t en)
Description: Function specifies whether automatic receiving of IP (IP-address, sub-network mask, gateway address) parameters is permitted using DHCP/linklocal or static specified parameters are used.
Parameters: cfg — Configuration handle of network interface en — If automatic receiving of parameters is permitted, 1 is returned, otherwise — 0
Returned value: Error code

4.3.7.11 Determining if user MAC-address is permitted

Format: int32_t E502_EthConfigGetUserMACEnabled (t_e502_eth_config_hnd cfg, uint32_t *en)
Description: Function returns whether MAC-address specified by user or factory MAC-address is used.
Parameters: cfg — Network interface configuration handle. en — If user MAC-address is permitted, 1 is returned, otherwise (if factory address is used) — 0
Returned value: Error code

4.3.7.12 Determining if user MAC-address is permitted

Format: int32_t E502_EthConfigSetUserMACEnabled (t_e502_eth_config_hnd cfg, uint32_t en)
Description: Function returns whether MAC-address specified by user or factory MAC-address is used.
Parameters: cfg — Network interface configuration handle. en — If user MAC-address is permitted, 1 is returned, otherwise (if factory address is used) — 0
Returned value: Error code

4.3.7.13 Receiving the specified static IP-address

Format: int32_t E502_EthConfigGetIPv4Addr (t_e502_eth_config_hnd cfg, uint32_t *ip_addr)
Description: Function returns static IP-address specified in configuration parameters and used by device if automatic IP-parameters receiving is prohibited.
Parameters: cfg — Network interface configuration handle. ip_addr — Specified IP-address in form of 32-bit word (the same as parameter ip_addr of function E502_OpenByIpAddr()).
Returned value: Error code

4.3.7.14 Setting up of static IP-address

Format: int32_t E502_EthConfigSetIPv4Addr (t_e502_eth_config_hnd cfg, uint32_t ip_addr)
Description: Function specifies in configuration specified static IP-address and which will be used by device if automatic IP-parameters receiving is prohibited.

Parameters: cfg — Network interface configuration handle. ip_addr — Specified IP-address in form of 32-bit word (the same as parameter ip_addr of function E502_OpenByIpAddr()).
Returned value: Error code

4.3.7.15 Receiving specified static sub-network mask

Format: int32_t E502_EthConfigGetIPv4Mask (t_e502_eth_config_hnd cfg, uint32_t *mask)
Description: Function returns value of sub-network mask specified in configuration which is used by device if automatic receiving of IP-parameters is prohibited.
Parameters: cfg — Network interface configuration handle. mask — Sub-network mask in form of 32-bit word (the same as parameter ip_addr of function E502_OpenByIpAddr()).
Returned value: Error code

4.3.7.16 Setting up of static sub-network mask

Format: int32_t E502_EthConfigSetIPv4Mask (t_e502_eth_config_hnd cfg, uint32_t mask)
Description: Function specifies value of sub-network mask in configuration which will be used by device if automatic receiving of IP-parameters is prohibited.
Parameters: cfg — Network interface configuration handle. mask — Specified value of sub-network mask in form of 32-bit word (the same as parameter ip_addr of function E502_OpenByIpAddr()).
Returned value: Error code

4.3.7.17 Receiving specified static address of gateway

Format: int32_t E502_EthConfigGetIPv4Gate (t_e502_eth_config_hnd cfg, uint32_t *gate)
Description: Function returns value of gateway address specified in configuration which is used by device if automatic receiving of IP-parameters is prohibited.
Parameters: cfg — Network interface configuration handle. gate — Gateway address in form of 32-bit word (the same as parameter ip_addr of function E502_OpenByIpAddr()).
Returned value: Error code

4.3.7.18 Setting up of static address of gateway

Format: int32_t E502_EthConfigSetIPv4Gate (t_e502_eth_config_hnd cfg, uint32_t gate)
Description: Function specifies value of gateway address in configuration which will be used by device if automatic receiving of IP-parameters is prohibited.
Parameters: cfg — Network interface configuration handle. gate — Specified value of gateway address in form of 32-bit word (the same as parameter ip_addr of function E502_OpenByIpAddr()).
Returned value: Error code

4.3.7.19 Receiving specified user MAC-address

Format: int32_t E502_EthConfigGetUserMac (t_e502_eth_config_hnd cfg, uint8_t *mac)
Description: Function returns the value of user MAC-address specified in configuration which is used by device at its explicit permission (see E502_EthConfigSetUserMACEnabled()).
Parameters: cfg — Network interface configuration handle. mac — User MAC-address of device v the array of X502_MAC_ADDR_SIZE bytes in the order of address writing
Returned value: Error code

4.3.7.20 Setting up of user MAC-address

Format: int32_t E502_EthConfigSetUserMac (t_e502_eth_config_hnd cfg, const uint8_t *mac)
Description: Function specifies value of user MAC-address specified in configuration which will be used by device at its explicit permission (see E502_EthConfigSetUserMACEnabled()).
Parameters: cfg — Network interface configuration handle. mac — Specified value of user MAC-address of device in form of array of X502_MAC_ADDR_SIZE bytes in the order of address writing
Returned value: Error code

4.3.7.21 Receiving factory MAC-address of device

Format: int32_t E502_EthConfigGetFactoryMac (t_e502_eth_config_hnd cfg, uint8_t *mac)
Description: Function returns value of factory MAC-address of device to which configuration

transfered by first parameter correspond. Factory MAC-address used by device by default is written by manufacturer (in "L Card") during device production together with its serial number and can not be changed by user. If user needs to change MAC-address of device, he should specify user MAC-address using E502_EthConfigGetUserMac() and permit its application throughE502_EthConfigSetUserMACEnabled(). In such case there is possibility to return the factory MAC-address.
Parameters: cfg — Network interface configuration handle. mac — Factory MAC-address of device in form of array of X502_MAC_ADDR_SIZE bytes in the order of address writing
Returned value: Error code

4.3.7.22 Receiving specified name of device instance

Format: int32_t E502_EthConfigGetInstanceName (t_e502_eth_config_hnd cfg, char *name)
Description: Function returns device instance name specified by user. This name can be used for detection of device in network. If not specified, the name formed by device name and its serial number is used. This name shall be unique within network.
Parameters: cfg — Network interface configuration handle. name — Ending by null string with specified device instance name in format UTF-8. Array shall be calculated as X502_INSTANCE_NAME_SIZE bytes of data.
Returned value: Error code.

4.3.7.23 Setting up device instance name

Format: int32_t E502_EthConfigSetInstanceName (t_e502_eth_config_hnd cfg, const char *name)
Description: Function specifies name of device instance which can be used for detection of device in local network.
Parameters: cfg — Network interface configuration handle. name — Ending by null string with specified device instance name in format UTF-8. Maximum array size (including terminating zero/null) is X502_INSTANCE_NAME_SIZE bytes of data.
Returned value: Error code.

4.3.7.24 Setting up new password for configuration change

Format: int32_t E502_EthConfigSetNewPassword (t_e502_eth_config_hnd cfg, const char *new_passwd)

Description:

Function specifies new value of password which shall be used to change configuration via E502_EthConfigWrite().

In such case when saving configuration with specified new password it is required to transfer previously specified password for successful change of configuration in E502_EthConfigWrite() . If function is completed successfully, for further change of configuration in E502_EthConfigWrite() you will need to transfer newly specified password.

Parameters:

cfg — Network interface configuration handle.

new_passwd — Ending by zero/null string containing new password to change network interface configuration. Maximum array size (including terminating zero/null) is X502_PASSWORD_SIZE bytes of data.

Returned value: Error code.

4.3.8 Functions for search of modules in local network

4.3.8.1 Beginning of modules search session in local network

Format: int32_t E502_EthSvcBrowseStart (t_e502_eth_svc_browse_hnd *pcontext, uint32_t flags)
Description: When calling this function the process of search of services corresponding to E502 modules in local network is initiated and context of current search session is created. This context is used for further calls E502_EthSvcBrowseGetEvent(). After completion of search function E502_EthSvcBrowseStop() shall be called. To start the session it is required to start service (daemon) of detection- Bonjour for OS Windows and Avahi for OS Linux are supported.
Parameters: pcontext — Pointer where upon successful completion the context of devices search session is saved. flags — Flags (reserve). Shall always be transfered 0.
Returned value: Error code.

4.3.8.2 Receiving information on change in modules availability in local network

Format: int32_t E502_EthSvcBrowseGetEvent (t_e502_eth_svc_browse_hnd context, t_e502_eth_svc_record_hnd *svc, uint32_t *event, uint32_t *flags, uint32_t tout)
Description: This function allows to get the list of available modules (network services) in local network as well as to monitor their status change in future. Function waits for first change of status and returns information on it. Information consists of event (appearance of network service, disappearance, change of parameters) and network service handle to which the event corresponds. After beginning of search using E502_EthSvcBrowseStart() context does not have information on network services availability. If there are E502modules connected in local network, the information on them will be returned in following E502_EthSvcBrowseGetEvent() with event E502_ETH_SVC_EVENT_ADD, each call per one device. If within specified time-out there was no change, function will be completed successfully upon time-out completion and return event E502_ETH_SVC_EVENT_NONE. If necessary, you can continue calling this function for continuous monitoring of modules connection status.
Parameters: context — Handle of search context created during calling E502_EthSvcBrowseStart(). svc — If returned event is not equal to E502_ETH_SVC_EVENT_NONE, created handle of network service corresponding to specified event will be saved in this variable. This handle shall always be deleted manually using E502_EthSvcRecordFree(). event — Event code is saved to this variable (one of t_e502_eth_svc_event). If within

specified time there was no event, code E502_ETH_SVC_EVENT_NONE is returned. flags — Additional codes of flags are saved in this variable (reserve). The null pointer can be transfered if flags value is no longer needed. tout — Time-out (in ms) for time of waiting for event
Returned value: Error code.

4.3.8.3 Stop of modules search session in local network

Format: int32_t E502_EthSvcBrowseStop (t_e502_eth_svc_browse_hnd context)
Description: When calling this function the process of network services corresponding to specified context search is stopped. All resources allocated at the stageE502_EthSvcBrowseStart() are released. Since this moment the context is invalid. Calling of E502_EthSvcBrowseStart() shall always be followed by E502_EthSvcBrowseStop() for correct resources release.
Parameters: context — Handle of search context created during calling E502_EthSvcBrowseStart().
Returned value: Error code.

4.3.8.4 Release of network service handle

Format: int32_t E502_EthSvcRecordFree (t_e502_eth_svc_record_hnd svc)
Description: Release of memory allocated for network service handle when calling E502_EthSvcBrowseGetEvent().
Parameters: svc — Network service handle
Returned value: Error code

4.3.8.5 Receive instance name due to service handle

Format: int32_t E502_EthSvcRecordGetInstanceName (t_e502_eth_svc_record_hnd svc, char *name)
Description: Function returns name of service instance. This name corresponds to name specified in network settings of module corresponding to indicated service using E502_EthConfigSetInstanceName(). It should be noticed that this name, as compared to other strings, is represented in encoding UTF-8 which is similar to common ASCII string only for symbols of English alphabet. Function does not execute requests to module itself.
Parameters: svc — Network service handle name — Array for X502_INSTANCE_NAME_SIZE bytes where instance name will be saved
Returned value: Error code

4.3.8.6 Receive serial number of module due to network service handle

Format: int32_t E502_EthSvcRecordGetDevSerial (t_e502_eth_svc_record_hnd svc, char *serial)
Description: Function returns serial number of E502 module corresponding to network service pointed by transfered handle. Function does not execute requests to module itself.
Parameters: svc — Network service handle serial — Array for X502_SERIAL_SIZE bytes where serial number will be saved
Returned value: Error code

4.3.8.7 Receive IP address of network service

Format: int32_t E502_EthSvcRecordResolveIPv4Addr (t_e502_eth_svc_record_hnd svc, uint32_t *addr, uint32_t tout)
Description: Function receives IP-address of E502 module corresponding to network service pointed by transfered handle. Function, if required, can perform requests to module itself to get this address if there is no information on address in cache.
Parameters: svc — Network service handle addr — IP-address of module in form of 32-bit word (the same as parameter ip_addr of function E502_OpenByIpAddr()) tout — Time of waiting for response from module if it is required to make request to establish the address.
Returned value: Error code

4.3.8.8 Checking if both handles indicates one service instance

Format: int32_t E502_EthSvcRecordIsSameInstance (t_e502_eth_svc_record_hnd svc1, t_e502_eth_svc_record_hnd svc2)
Description: Function checks if both services handles point out the same instance. If application saves the list of services handles in case of their detection, this function can be used, for example, during events E502_ETH_SVC_EVENT_REMOVE or E502_ETH_SVC_EVENT_CHANGED to understand which record of saved list corresponds to event (i.e. function E502_EthSvcBrowseGetEvent() will return new handle but pointing out the same instance as during event E502_ETH_SVC_EVENT_ADD)
Parameters: svc1 — First handle of network service for comparison svc2 — Second handle of network service for comparison
Returned value: Error code. Returns X502_ERR_OK if both handles point out the same instance.

4.3.9 Functions for working with signaling processor

4.3.9.1 Loading of BlackFin signaling processor firmware.

Format: int32_t X502_BfLoadFirmware (t_x502_hnd hnd, const char *filename)
Description: Function downloads firmware of signaling processor from specified file to processor and starts it, check correctness of loading by means of getting firmware version (through special command). Firmware shall be in binary format LDR.
Parameters: hnd — Module handle. filename — Name of file with loading firmware.
Returned value: Error code.

4.3.9.2 Checking if BlackFIn firmware is loaded.

Format: int32_t X502_BfCheckFirmwareIsLoaded (t_x502_hnd hnd, uint32_t *version)
Description: Function sends commands to BlackFin processor in order to get firmware version and its status. Successful fulfillment of commands indicates that valid firmware is loaded to BlackFin. Moreover, firmware gets information on module (availability of options, FPGA version, etc.) for internal use. In case of success module shifts to DSP mode. This function can serve for checking if firmware has been loaded before (in order not to download it again) or to check if it is loaded through JTAG-interface.
Parameters: hnd — Module handle. version — If handle is not null, BlackFin firmware version is returned in this variable in case of successful checking.
Returned value: Error code.

4.3.9.3 Reading data block from signaling processor memory.

Format: int32_t X502_BfMemRead (t_x502_hnd hnd, uint32_t addr, uint32_t *regs, uint32_t size)
Description: Function reads data block directly from processor memory. Data can be read from internal memory (L1) as well as from external SDRAM. To perform this function its firmware shall be downloaded in BlackFin. Function is meant, mainly, for users writing their own programs for signaling processor.
Parameters: hnd — Module handle.

addr — Address of memory starting with which data block will be read. regs — Array where read memory content will be saved. size — Number of readable 32-bit words.
Returned value: Error code.

4.3.9.4 Writing data block to signaling processor memory.

Format: int32_t X502_BfMemWrite (t_x502_hnd hnd, uint32_t addr, const uint32_t *regs, uint32_t size)
Description: Function writes data block directly to BlackFin processor memory. Data block shall always be 8-fold 32-bit words (32 bytes). Writing can be executed to internal memory (L1) as well as to external SDRAM. To perform this function its firmware shall be downloaded in BlackFin. Function is meant, mainly, for users writing their own programs for signaling processor. You should be careful because writing to data area used by program can cause its malfunction.
Parameters: hnd — Module handle. addr — Address of memory starting with which data block will be written. regs — Array with data for writing to signaling processor. size — Number of data to be written in 32-bit words (shall be 8-fold).
Returned value: Error code.

4.3.9.5 Transfer of controlling command to signaling processor.

Format: int32_t X502_BfExecCmd (t_x502_hnd hnd, uint16_t cmd_code, uint32_t par, const uint32_t *snd_data, uint32_t snd_size, uint32_t *rcv_data, uint32_t rcv_size, uint32_t tout, uint32_t *recvd_size)
Description: Function is meant for transfer of user controlling commands to processor for users writing their BlackFin firmware. Control of signaling processor operation in standard way is executed through controlling commands written in special area of signaling processor memory. Signaling processor processes command and upon completion writes the result in the same area. Commands are divided into standard used by x502api library and implemented in standard firmware of signaling processor and user ones which user can specify as he wants. User commands begin with code X502_BF_CMD_CODE_USER (0x8000).
Parameters: hnd — Module handle. cmd_code — Command code- determines which command is being performed. par — Parameter transfered with command (value depends on command code). snd_data — Optional data transfered together with command. If data not transfered, null pointer shall be transfered as well as snd_size = 0.

snd_size — Number of 32-bit words transfered to snd_data

rcv_data — Array where data returned by processor upon command completion will be transfered. If data shall not be transfered, null pointer shall be transfered as well as rcv_size = 0.

rcv_size — Number of 32-bit words expected to be returned by processor upon command completion. Array rcv_data shall be calculated for specified number of words.

tout — Time-out during which it is expected when processor completes the command. Function returns the control whether upon completion of command or upon time-out.

recvd_size — If it is not null pointer, number of 32-bit words actually returned by processor after command execution will be saved in this variable (processor has the right to return less data than has been requested in rcv_size).

Returned value:

Error code. If processor executed command with null code of completion, this code will be returned by function.

4.3.10 Functions for working with Flash-memory of module

4.3.10.1 Reading data block from Flash-memory.

Format: int32_t X502_FlashRead (t_x502_hnd hnd, uint32_t addr, uint8_t *data, uint32_t size)
Description: Function reads data array from Flash-memory of module to array transfered by user. There is no special permission required for reading, it is always available.
Parameters: hnd — Module handle. addr — Address of block beginning. data — Array where read data will be saved (shall be not less than size bytes). size — Amount of bytes for reading.
Returned value: Error code.

4.3.10.2 Writing data block to Flash-memory of module.

Format: int32_t X502_FlashWrite (t_x502_hnd hnd, uint32_t addr, const uint8_t *data, uint32_t size)
Description: Function writes data array transfered to module Flash-memory. This area shall be preliminary erased using X502_FlashErase() and before starting changes the function X502_FlashWriteEnable() shall be called to permit any change of Flash-memory content. Only first X502_FLASH_USER_SIZE bytes of Flash-memory are available for writing by user.
Parameters: hnd — Module handle. addr — Address of block beginning. data — Array of recordable data (shall be not less than size bytes). size — Number of bytes for writing.
Returned value: Error code.

4.3.10.3 Erasing of block in Flash-memory.

Format: int32_t X502_FlashErase (t_x502_hnd hnd, uint32_t addr, uint32_t size)
Description: Function deletes the block in Flash-memory of module (all cells will be read as 0xFF). Address and size shall be 4096-fold bytes! Before calling this function writing to user area shall be permitted using X502_FlashWriteEnable().
Parameters: hnd — Module handle. addr — Address of block beginning (shall be 4K-fold).

size — Number of bytes to delete (4K-fold).
Returned value: Error code.

4.3.10.4 Permission of writing to user domain of Flash-memory.

Format: int32_t X502_FlashWriteEnable (t_x502_hnd hnd)
Description: Function permits writing to user area of flash-memory (first X502_FLASH_USER_SIZE bytes). Shall be called before using X502_FlashErase() and X502_FlashWrite() to change content of user area of memory. After completion of changes X502_FlashWriteDisable() should be called.
Parameters: hnd — Module handle.
Returned value: Error code.

4.3.10.5 Inhibit of writing to user domain of Flash-memory.

Format: int32_t X502_FlashWriteDisable (t_x502_hnd hnd)
Description: Function inhibits writing to user area of module flash-memory (first X502_FLASH_USER_SIZE bytes). Shall be called when required data in user area have been changed using X502_FlashErase() and X502_FlashWrite() to protect user area from occasional change in future.
Parameters: hnd — Module handle.
Returned value: Error code.

4.3.11 Additional supplementary functions.

4.3.11.1 Receive version of L502 module driver.

Format: int32_t L502_GetDriverVersion (t_x502_hnd hnd, uint32_t *ver)
Description: Function returns driver version specified for indicated opened device. Version is returned in form of 32-bit digit. String representation of returned version-four digits, the high corresponds to high byte, the low- low byte. High byte- major version, second byte- minor, third- revision, fourth- number of build (not used- always 0). This the version represented in Windows Device Manager or using modinfo in Linux. This function is available only for devices with PCI/PCIExpress (L502) interface.
Parameters: hnd — Module handle. ver — 32-bit digit representing driver version
Returned value: Error code.

4.3.11.2 Switching E502 module to loader mode

Format: int32_t E502_SwitchToBootloader (t_x502_hnd hnd)
Description: Function transfer the device into loader mode to have possibility of updating of Cortex-M4 controller firmware of E502 module using utility lboot. Depending on used current interface for connection with module the module is transfered in mode of firmware loading via USB interface (if connection was via USB) or via TFTP (if connection was via Ethernet interface). In case of successful calling of this function further operation with current connection is impossible, in other words the only permissible following call is X502_Close(). When transfered into loader it stays in loader mode for 30 s after this if no request for reflashing delivered the loader returns the control to standard firmware. While module is in loader mode it can not be connected to using functions of this library.
Parameters: hnd — Device handle.
Returned value: Error code.

4.3.11.3 Reload of FPGA firmware

Format: int32_t E502_ReloadFPGA (t_x502_hnd hnd)
Description: Upon this command controller Cortex-M4 of E502 module resets FPGA and FPGA firmware loading from internal Flash-memory. This service function used mainly for FPGA firmware update.

Parameters: hnd — Device handle.
Returned value: Error code.

4.3.11.4 Transfer of controlling command to Cortex-M4 controller.

Format: int32_t E502_CortexExecCmd (t_x502_hnd hnd, uint32_t cmd_code, uint32_t par, const uint8_t *snd_data, uint32_t snd_size, uint8_t *rcv_data, uint32_t rcv_size, uint32_t tout, uint32_t *recvd_size)
Description: Function is meant for transfer of user controlling commands to controller in case of modified Cortex-M4 firmware.
Parameters: hnd — Module handle. cmd_code — Command code- determines which command is being performed. par — Parameter transfered with command (value depends on command code). snd_data — Optional data transfered together with command. If data not transfered, null pointer shall be transfered as well as snd_size = 0. snd_size — Number of bytes transfered to snd_data rcv_data — Array where data returned by the processor upon command completion will be sent If data shall not be transfered, null pointer shall be transfered as well as rcv_size = 0. rcv_size — Number of bytes expected to be returned by controller upon command completion. Array rcv_data shall be calculated for specified number of words. tout — Time-out during which it is expected when controller completes the command. Function returns the control whether upon completion of command or upon time-out. recvd_size — If it is not null pointer, number of bytes actually returned by controller after command execution will be saved in this variable (controller has the right to return less data than has been requested in rcv_size). If pointer is null, the return less data is considered as error.
Returned value: Error code.

4.3.11.5 Receive library version.

Format: uint32_t X502_GetLibraryVersion (void)
Description: Function returns library version x502api. Version is returned in form of 32-bit digit. String representation of returned version-four digits, the high corresponds to high byte, the low- low byte. High byte- major version, second byte- minor, third- revision, fourth- number of build (not used- always 0)
Returned value: 32-bit digit representing library version

4.3.11.6 Receiving error string.

Format: const char* X502_GetErrorString (int32_t err)
Description: Function returns string corresponding to transfered error code. At the moment Russian version of string is always returned (maybe in future it will be possible to change the language by global function). It should be considered that in Windows OS the string is returned in standard for Windows encoding CP1251 while in Linux UTF-8 encoding is used.
Parameters: err — Error code requiring to return the string.
Returned value: Pointer to string corresponding to error code

4.3.11.7 Light-emitting diode blinking.

Format: int32_t X502_LedBlink (t_x502_hnd hnd)
Description: When calling this function if synchronous input/output is selected short-term fading of LED red light occurs on L502 module front panel and LED1 of E502 module. Can be used for visual identification of module after its opening. When synchronous input/output is running the LED is alway green and this function does not influence on its status.
Parameters: hnd — Module handle.
Returned value: Error code.

4.3.11.8 Installation of pull-up resistors on input lines.

Format: int32_t X502_SetDigInPullup (t_x502_hnd hnd, uint32_t pullups)
Description: Function can be used to switch on the pull-up resistors on digital inputs. For different modules the pull-up resistors are implemented on different inputs. For all modules they can be switched on SYN1 and SYN2 lines. For L502 it is possible to set whether the pull-ups are on or off on low-order or high-order half of digital lines. For E502 it is possible to switch on the resistors pulling up to zero at inputs of inter-module synchronization. For not stated lines the pull-up resistors will be switched off if they have been switched on before. When power is supplied all pull-up resistors are off.
Parameters: hnd — Module handle. pullups — Flags (from t_x502_pullups) determining which lines have connected pull-up resistors.
Returned value: Error code.

4.3.11.9 Checking if the module supports the specified feature

Format: int32_t X502_CheckFeature (t_x502_hnd hnd, uint32_t feature)
Description: Function is used to check if the certain feature from t_x502_features is supported for the present module with current firmwares. If feature is supported, the code X502_ERR_OK will be returned. This function is available in library of version 1.1.6 or later.
Parameters: hnd — Module handle. feature — Value from t_x502_features determining which feature should be checked.
Returned value: If feature is supported, X502_ERR_OK is returned otherwise — error code