Diverse Embedded Control Cross

Developer's Guide

LOXTechnologies

DEX - Diverse Embedded Control Cross

Abstract

DEX enables convenient design of user-specific graphic interfaces to simulated and real-world embedded modules with integrated system development and evaluation functionality. The modules come in a form of dynamically linkable libraries (*.DLL) where exported global variables are accessed by DEX in order to link it with graphic controls like text edit windows, buttons, scroll bar or wheel controls etc. Virtually any signal data type is supported within the DEX signal interface, including 8- through 32-bit integers, bit fields, and floating point data types. The data type, graphic controls, as well as initial, minimum and maximum values defined for the particular signal are stored within a module file together with path to linked libraries and external signal channels that provide connection between different modules. With several modules linked with their libraries, processed within a DEX project session, complex systems can be conveniently evaluated without special need of re-compiling and linking existing software modules. A flexible sampling rate control enables evaluation of modules even for multi-kHz sampling frequencies, while keeping the graphic interface re-scan rate steady at 10Hz. Though any type of linkage between the software module and real-world embedded system can apply, DEX comes with an option for easy and almost automatic generation of an interface module to user-designed microcontroller application.

Features

- **Rapid design of embedded system control interfaces**
- **Integrated development and evaluation for multi-module systems**
- **All-in-one plant simulation, control system evaluation, process monitoring**
- **Software modules with exported signals linked as DLL libraries**
- **Module signal values available in both text and graphic form**
- **Text edits available in alphanumeric, scientific, hexadecimal notation**
- **Graphic controls include buttons, scroll bars, wheels**
- **Instant synchronization between controls**
- **8- to 32-bit integers, bit fields, floating points supported**
- **Masking and shifting for bit fields enables easy look-through**
- **Channels connect signals from different modules regardless of data type**
- **Signals with different units can be easily connected defining its unit value**
- **Sampling rate control in 0 - 10kHz range enables audio processing**
- **DexLink option connects to an user application in AVR type microcontroller**
- **Future support: recording and graphic output**

Contents

Installation

Requirements

DEX is Win32-based application, and is tested for Windows2000 and later environments. A PC with 1GHz Pentium-IV, 1280x1024 graphics and above is recommended.

Installing DEX

Full and demonstration version of DEX can be found on **www.loxtechnologies.com** web pages as a .zip package. Simply, download and unzip the dex.zip or dexdemo.zip file into a directory created on your local hard drive, e.g. in c:\DEX. When unzipped, the directory will contain dex.exe, one or more project (*.dxp) files and several other files including DLL libraries and .mod files defining the interface modules.

Limitations in the demonstration version

Demonstration version has all the functionality of the full distribution version, except **Save** commands for saving project and module interface files are not supported.

Opening and evaluation of a demo project

In the created DEX directory where the full or demonstration version is installed, run dex.exe for demonstration start. When main window appears, open project menu and click on Open Project tab. In the open dialog, select the created DEX directory. You will see one or several available project files within the directory that can be opened, select one and then open it. A project with several client module windows will be loaded. When everything is correct, signal captions will be in blue color to signalize linkage with signals in DLL libraries. This means that project simulation already runs and you can play with it.

DC servosystem simulation

This project contains 3 modules: DCMotor, SpeedController and PositionController. Both controllers are in fact of the same PI (proportional-integration) type, only the gain in integration path of PositionController is set to zero, so it is effectively a proportional controller.

Controllers have two input signals **Wished** and **Actual**, two parameters: **Kp** (proportional constant) and **Ki** (integration constant), and one **Output** signal. Input signal of the DCMotor is **Voltage**, with signals **Resistance**, **ConstantV**, **Inertia** and **Friction** being editable motor's parameters. Motor status can be read in signals **Current**, **EMFVoltage**, **Torgue**, **Speed** and **Position**. In fact, all the signals are in SI (system international) units with **Position** and **Speed** representing number of revolutions and revolutions per second, respectively.

In the cascade-shaped servosystem, proportional controller's output is connected with wished input of speed controller, which output is connected with DC motor's input (signal **Voltage**). Motor's speed is then relayed to speed controller's actual input and motor's position to actual input of position controller. Signal connections (i.e. channels) between modules are indicated with blue labels right down in the source signal frame, addressing its destination module and signal.

Most signals are limited which can be seen on a scroll bar graphic control limits. When a graphic control applies for a signal that is module's input or parameter, and not connected by channel with other module output, then the graphic control can be used to modify the signal value – this applies to all here involved graphic controls i.e. buttons, scroll bars and wheels. You can also use text edit window in the signal frame to modify its value, by entering new numerical value (you must press ENTER after editing).

To bring the servosystem into motion, simply try to turn the control wheel of the **Wished** signal in PositionController with left-button-down mouse. You will see a number of signals changing and the DCMotor Position wheel rotating in the right direction and exactly to the position value you choose. You can also enter the wished motor position by entering a numerical value in the text edit control of the PositionController **Wished** signal. You can switch OFF the PositionCotroller clicking on the corresponding controller's **Mode.ENABLED** button and obtain manual control over servosystem's speed, by modifying **Wished** signal value on the SpeedController. Finally, you can switch both controllers OFF to have control over motor's **Voltage**. Observe motor's reaction when you turn the motor's shaft (position wheel) by zero applied voltage: this is just the way a DC motor will behave with shortened contacts!

Working with module

How does DEX interact with a software module?

Low-overhead, easy-to-understand DEX/module interaction

For DEX, module's interface, i.e. the way it exchanges data with its environment (modules included within the project), is achieved simply by accessing a collection of global variables that represents input, output, and module parameters. There are several reasons for this. First, it is simple and therefore available for wide range of software developers, even for those not familiar with more sophisticated interfaces common in object-oriented programming. Secondly, OOP often fails to apply in embedded software development, for larger system resources often needed or simply because no OOP compiler is available or affordable for the particular design. Moreover, it is doubtful if any sophisticated interface, consuming both machine and development resources, will be of any advantage, when rapid, low-cost development and evaluation process is asked. And finally, DLL is well-known, low-overhead standard suited just for this type of interface.

The addresses of global variables of a software module are exported as is common in a DLL interface. Then, DEX by loading the DLL module reads addresses of those global variables. Because signals in DEX are represented as pointers to some data types, DEX reads corresponding global variable address from the DLL module and sets this as the value of the signal data pointer. Any access or modification of the signal value is achieved through

reading or writing to the signal data pointer that addresses just the corresponding global variable of the DLL module. And that's all!

Of course, elevated precaution is required for proper setting of signal data types in DEX that must unconditionally correspond with data types in the DLL! This is the price for the simple and straightforward interface we have.

Instant interactivity with DEX build-in technologies

Internally, DEX implements a sophisticated method of channel-based synchronization of data within a signal frame, where a change in the data at some place is immediately relayed to all connected components (signal frame text and graphic controls, controls at extended signal information window etc.). The advanced mechanisms here, that includes avoidance of endless loops, brings a level of comfort not commonly available for the end user: a change in a parameter spreads immediately over the whole project, without need of pressing any 'apply' button or something.

Synchronization of signal values and control components

Beyond the DEX/module interface, however, the channel data exchange mechanisms cannot reach. Instead, any change in DLL global variable data is recorded by buffering old signal values, and if change occurs, DEX signal frame is noticed about the change, which is then broadcasted using the DEX channel mechanism. The opposite case, when data change originates in the DEX, needs not be specially treated, because DEX signal frame data seats actually in the DLL module.

A DLL module is evaluated at a constant rate by calling its exported eval() function. When a signal change has been initiated in DEX, the DLL can detect the change in its data by comparing to its own buffered values, and then correspondingly behave. Or, DLL can simply use the status in the global variables to compute new status in a defined way. After DLL evaluation ends (all project DLL modules are evaluated one by one as a block), DEX detects the changes in signal data, and if occurred, starts spreading the changes over the channel network. In such a way, synchronization of graphic components with the actual module variables is achieved.

Order in which DLL modules are evaluated is not of importance because DEX-mediated data interchange between them occurs after all the DLLs are evaluated in one block; in such a way, complexity resulting from module's interconnections is avoided at a price of effectively embedding one sample-period time delay in each signal connection between DLL modules. These time delays may not be of any disadvantage when properly involved into the overall system design.

Building a software module

Compilers for building a software module

You will need a compiler, presumably C/C++ or Pascal one, able to create DLLs (dynamically linked libraries, which is a standard on Microsoft platforms). Compiler GUIs are often equipped with wizards that simplify overheads in creation of DLL-enabled code. There are totally free command-line compilers available that are fine for that task, like Borland's BC command-line compiler. Moreover, DEX Demo package contains source code for modules

involved in the demonstration project, so you can reverse-engineer this source file to create your own module.

Required composition of a module

The software module is obliged for export two functions: function init() and function eval(). Function init() is invoked after the DLL is linked; it can be used to allocate memory and initialize any internal data before the processing begins. Function eval() is invoked in regular time intervals and represents inner data processing within the software module.

Function init() has no arguments and returns a char (signed byte) that contains number of invocations of this function; this serves as an indication of how many module interfaces are linked with the same DLL, to avoid multiple invocation of the eval() function. Multiple allocations of memory by possible multiple invocations of init() have to be avoided internally in the init() function.

Function eval() has no arguments and no return value.

When memory has been allocated in the init() function, it is strongly recommended to free it in a deinit() function, which has no arguments and no return value. When an exported deinit() function is found by DEX, then it will be invoked when the software module is released from being linked with DEX, e.g. at DEX project close. While init() and eval() are mandatory functions, deinit() need not be present in the module.

Exporting signal variables

For each signal of the software module you need to control or monitor by DEX, simply define a global variable of the corresponding type in the source code, e.g.

```
float Voltage=0;
```

and, declare it in the header file like

```
extern "C" __declspec(dllexport) float Voltage;
```

which ensures the compiler is informed about your desire to export this variable through the DLL's interface. You can also write the declaration within the source code before the variable definition is taking place, when no header file is present for your software module.

Exporting module functions

Assuming you already defined the mandatory functions init() and eval() within the source code of your software module, simply add

```
extern "C" __declspec(dllexport) char init(void);

extern "C" __declspec(dllexport) void eval(void);
```

into the header file, or in the source code before definition of those functions is taking place. The same holds for the non-mandatory function deinit().

Guide to compile a demo module

Will be added later.

Building module interface

Module file

Module file (*.mod) is the user interface to the evaluated software module in the DLL library file. Module file stores information about available signals, their types, formats, utilized graphic controls, signal interconnections and more.

Any evaluated software module must have corresponding module file open and must be linked with it. Only one DLL can be linked with the module file at a time, however, more module files can link to the same DLL.

To create a new, open an existing, close, save or rename a module file, use controls in the Module menu, or speed buttons on the toolbar. When open, module file will appear as a document window in the DEX working space.

Adding signals

Module input and output as well as inner parameters can be viewed and edited as signals. To add them into the module file, click **Add** in the **Signal** menu. **Add Signal** dialog will appear where signal name, type, and controls can be chosen. Then, signal frame will be placed into the current module window.

Signals within the module window can be selected or unselected by clicking on label (signal name). The selected signal will appear with label in bold type.

To delete a signal from module file, select a signal frame and click **Delete** in **Signal** menu.

Selecting signal data types

You can chose signal name, signal data type and graphic controls in the **Add Signal** dialog which appear as you try to add signal into the current module.

First, **Add Signal** dialog prompts for signal **name**. Chose the name of the desired global variable in the DLL module which should be evaluated. Note that the signal name is case-sensitive.

Alternatively, add postfix to the variable name separated with comma character, to indicate special meaning of the signal; e.g. a bit within a bit field. DEX will interpret and search the DLL only for the first part of the name up-to the comma character.

In the next, chose variable type. Always select **isatom** option, then select **atomtype** as **CHAR** for character variable (char), **INT** for all integer types (including 8-bit types), or **FLOAT** for all floating-point types (including double precision).

For **INT** types, dialog prompts for **issigned**, **islimited**, **ismasked** and **isbitfield** flags and for the **inttype** bit-length (**BYTE**=8bit, **SHORT**=16bit, **INT**=32bit, **LONG**=32bit). When the DLL variable is signed integer type, select **issigned** flag. If you desire to limit its range with minimum and maximum values, select **islimited**. When variable is a bit field from which only part is to be viewed and edited, select **ismasked** and **isbitfield**. Then, select proper **inttype** for the variable as is defined in the DLL source file. Note that the numerical type bit-length selection is crucial because it tells the DEX how long is the variable representation in the memory, so be cautious in this matter.

For **FLOAT** types, you can choose **islimited** to be enabled to set the lower and upper limits of the signal. Then select the proper **floattype** bit-length (**SINGLE**=32-bit, **DOUBLE**=64-bit). Again, be careful in choosing the proper signal bit-length.

Select **inrecord** when recording option should be available for the current signal.

Note that once selected, signal type (**atomtype**, **inttype**, **floattype**) cannot be altered anytime later!

Adding graphic controls to signals

By default, signal frame comes with text editing box. To add other graphic controls, select **iscontrol** in the **Add Signal** dialog and then choose one graphic control that will be associated with the signal frame.

*DEX V1.1: Only **BUTTON**, **SCROLL** and **WHEEL** controls are available*

Graphic control selection cannot be altered later!

Connecting signals using channels

You can "wire" signals within the module or whole project using channel interconnections. Changes in one source signal will then be immediately relayed to all destination signals. To build a one-direction channel between two signals, select the source signal, the click to **Add Channel** in the **Signal** menu, and then click to the destination signal in either module present within the project. Channel will be then indicated as blue label in right-lower corner of the source signal frame with target module and signal indicated in **module::signal** form. For bi-directional connection, simply select the previous destination signal and make him source of a channel for the reverse signal path.

A signal frame can source up-to 4 user-defined channels.

Linking the module

Linking the DLL module

To link software DLL module to a module file, click to **DLL Link** on the **Module** menu or on the toolbar. To release the DLL link, click **DLL Release**.

By linking, DEX searches for signal names found within the module window in the DLL module, and if successful, actual variable address in the signal frame of that name will be replaced with variable address from the linked DLL module. The successful signal linkage is indicated with signal frame name in blue color.

When linked, DLL module is steadily re-evaluated at a rate defined by sampler (default value is 10-times per second). This can be a problem when editing signal parameters like limits or appearance; so release the DLL link when attempting to edit those parameters.

Linkage to DLL, as well as other module- and subsequent signal-related parameters are saved within the module file, so when module is again opened, it will be automatically linked to its DLL.

Initialization options

Initial values for module signals can be chosen independently of initial settings in the DLL module. First, set signal values in the non-linked module to the desired initial values and then click to **Set Actual Data as Initial** in the **Module** menu. This stores actual edited values as initial. Then, select **Initialize after Link** option and save the module. Then link the DLL, and DEX will initialize it with previously stored initial values.

Working with project

Project file

A DEX project is collection of open module files linked to the corresponding DLL software modules. You can create, open an existing, save, or rename project file (*.dxp) using items in the **Project** menu or speed buttons on the toolbar.

By saving project file, all currently open module windows with their settings are saved, and information about names and locations of current module files is saved in the *.dxp project file. Opening an existing project will force DEX to open all the involved module files with linked DLL modules.

Note that project file does not contain module files. So when distributed, DEX project must contain *dxp file, as well as all the *.mod and *.dll files involved in.

Adding and removing modules within a project

To add a module into a current project, simply create or open existing module. To remove module, close the corresponding module window.

Sampler module

Opening sampler module that can be found in the DEX directory (sampler.mod) enables you to control sampling rate for all modules found within the project.