Available online at www.sciencedirect.com

SciVerse ScienceDirect

Energy Procedia 17 (2012) 1945 – 1952

Energy Procedia

2012 International Conference on Future Electrical Power and Energy Systems

A Method to Improve the Interacting between MATLAB and NI-DAQmx

Yimeng Chen,Xuelian Li

Experiment and Training Cente,HuBei University of Technology,Wuhan, HuBei Province, China
Yimeng.chen@gmail.com

Abstract

This paper propose a method to improve the interacting between MATLAB and NI-DAQmx, we built a new Matlab toolkit for data acquisition using the NI-DAQmx drivers. With this toolkit Matlab can support the full functionality of NI-DAQmx and the data acquisition will be more stabile and effective.

Keywords:MATLAB, NI-DAQmx ANSI C API

1.Introduction

MATLAB and NI-DAQmx are both widely used softwares. MATLAB provides Toolbox to interact with NI-DAQmx , but a lot of situations should be considered while using it. Here we give a new alternative method to solve those problems.

1.1.MATLAB

MATLAB is a numerical computing environment used frequently in research and education.It allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, and Fortran.It has powerful ability in data acquisition, data analysis, and application development as a single environment. It's Data Acquisition Toolbox supports the instruments developed by National Instruments using the hardware driver NI-DAQmx or traditional NI-DAQ.

 doi:10.1016/j.egypro.2012.02.337

1.2.NI-DAQmx

NI-DAQmx is one of the best driver softwares goes far beyond a basic data acquisition driver to deliver increased productivity and performance in data acquisition and control application development. NI-DAQmx controls every aspect of the DAQ system (including NI signal conditioning devices), from configuration, to programming in LabVIEW, to low-level operating system and device control. Quickly gather real-world data with measurement-ready virtual channels and DAQ Assistant. It supports NI LabVIEW, NI LabVIEW SignalExpress, NI LabWindows/CVI, C/C++, Visual Basic, Visual Basic .NET, and C#.

1.3.NiMex

NiMex is a Matlab toolkit for data acquisition, using the NI-DAQmx drivers from National Instruments. The functionality of the drivers is exposed through and wrapped by C code. It provides relatively direct and straight-forward access to the National Instruments NI-DAQmx ANSI C driver API.

2.NiMex Overview

2.1.What is NiMex?

As we said before, NiMex provides relatively direct and straight-forward access to the National Instruments NI-DAQmx ANSI C driver API. It is made accessible to Matlab via MEX functions. The MEX functions are further wrapped by Matlab classes @nimex and @nimexEngine.
It provides access to the NI-DAQmx via a set of Matlab MEX functions that act as wrappers for the API. Whenever possible, consistent naming conventions were used.

In case of confusing, following is a note of naming conventions.

NiMex refers to whole package, including C-functions and the Matlab classes;
NIMEX refers to the C-code only;
@nimex and *@nimexEngine* are Matlab classes;
All other forms are assumed to refer to Matlab code.

2.2.The advantages of NiMex

- *Performance*

The Matlab daqtoolbox must work with drivers from multiple vendors, which is associated with considerable overhead (such as emulating hardware behavior in software, to ensure it works the same across devices).

NiMex is not subject to these constraints. NiMex therefore shows much improved efficiency and stability.

- *2. Reliability*

Event-driven programming also gets around some of the limitations imposed by Matlab's single-threadedness. However, the passing of events from NI-DAQmx into Matlab is non-deterministic in the Matlab daqtoolbox.

In contrast, NiMex event callbacks are guaranteed to execute in the order that they are generated. This is a crucial feature when writing event-driven software, which is a standard model of efficient programming for GUIs, hardware sensors, and multiple threads.

- *3. Support*

The problem we meet the most is that the adapting of Matlab daqtoolbox seldom keep up with the the new NI-DAQmx standard. Furthermore, support for the full functionality is not planned.

As an extensible, open-source package, NiMex provides access to the full capabilities of the data acqusition hardware. It also provides the possibility to add new functions or make changes based on the requirements.

- *4. Functionality*

NiMex supports all of the functions provided by the Matlab daqtoolbox and significantly more.

For example, NiMex supports the use of counter/timers and buffered/clocked digital I/O (as opposed to static lines that can only be updated once per functional call). These features allow a single board to do significantly more work than an equivalent board under the control of the Matlab daqtoolbox.

Some extra concepts and features were added on top of the standard NI-DAQmx model to help design event-driven (asynchronous) applications. These include new event-driven callbacks, enhanced state information, the ability to re-use a task, a flexible 'data source' model (allowing both arrays of data and runtime callbacks which generate data on the fly). Additional features, including Matlab-configurable C-level data preprocessors, generators, and listeners, could be easily implemented. Because NiMex is open source, it may be easily expanded or modified to suit a specific goal.

3.Two key parts of NiMex

As mentioned before, NiMex mainly consists of two parts, c-function and Matlab classes named of NIMEX and @nimex. There are more details of those important two parts.

3.1.NIMEX

- *How to build NIMEX*

The class used in NIMEX follows the NI-DAQmx task model. Any given data acquisition subsystem action is represented by a task. The task is created, its parameters are set (this configures the task for the desired measurement), the task is started (data is collected or sent), the task is stopped. The nimex task is re-usable, each subsequent use applies the current parameters, a task may be configured once and used many times.

Each task consists of a set of task-specific properties (task properties), channel-specific properties (channel properties), and registered callbacks. In addition, a task contains internal variables in the C code (not exposed to Matlab).In terms of design patterns, the publisher/subscriber pattern and visitor pattern are used extensively. Understanding these will help with following the general model of control flow throughout the code.

The registered callbacks require further comment: The NI-DAQmx library provides access to events which may cause code to be executed upon their occurence. Specifically, it provides an EveryN event (which is initiated after every N samples are aqcuired/sent) and a done event (which is initiated when the task has completed). The @nimex class allows multiple Matlab functions to be tied to each of these events. To tie a Matlab function to an event, a function_handle must be passed to the appropriate 'bind' function. If arguments are required for the Matlab code, a cell array may be provided instead. The cell array's first element must be a function_handle (the function to be executed), all subsequent elements will be passed into the called function as arguments. The @nimex class does not pass any of its own arguments when calling event, it is the responsibility of the registering code to specify all arguments.

- *The libraries required in NIMEX*

The National Instruments NI-DAQmx ANSI C library and the Matlab C development (MEX) library are required to build NIMEX. These are included with the National Instruments NI-DAQmx driver installation and Matlab installation, respectively.

Low-level data structures are currently handled by GLib (not to be confused with glibc, the core of the Linux kernel), which is part of the larger GTK+ and GNOME libraries. The utility library, GLib, is thinly wrapped, mostly using compile time replacements through macros. This allows it to be relatively easily swapped with another library (requiring changes in only two files). The datastructures which are used from GLib include hash tables and linked lists. In the future, multithreading may be made to depend on this library as well.

The Microsoft Windows Server 2003 Platform SDK is required for use with Windows. This is not included as part of Microsoft Visual C/C++ (nether the full version nor the Express Edition). It is a rather large and tedious download, but is a one-time affair when setting up a build environment. The requirement of the Platform SDK is for the Windows message passing functionality. It is needed to implement cross-thread callbacks to Matlab (as discussed further below).

3.2.@nimex Class

@nimex class is relatively simple in Matlab, the majority of the work and data structures exist on the C side of the implementation. The core of the class is the field called NIMEX_TaskDefinition. This is really a C pointer that has been packed into a Matlab variable, so the structures in C may be referenced across calls to different Mex functions.

The following is a rough inventory of the @nimex class, including the fields, methods and the C structures. At the end are two pieces of Demo script.

• *Fields*

TABLE I FIELDS OF @NIMEX

NIMEX_ TaskDefinition	The Matlab packed C pointer to the underlying structure
valid	An internal flag used to indicate if this task is properly instantiated in C
instantiationTime	A timestamp, recording when this task was created
instantiationStack	A debugging tool, the call stack leading up to the task creation

• *Method*

- **nimex.m** - *Constructor.*
```
task = nimex;
```
- **nimex_addAnalogInput.m** - *Adds an analog input channel to the task.*
```
nimex_addAnalogInput(task, '/dev1/ai0');
```
- **nimex_addAnalogOutput.m** - *Adds an analog output channel to the task.*
```
nimex_addAnalogOutput(task, '/dev1/ao0');
```
- **nimex_addDigitalOutput.m** - *Adds a digital output channel to the task.*
```
nimex_addDigitalOutput(task, '/dev1/port0/line0:7');
```
- **nimex_addDigitalInput.m** - *Adds a digital input channel to the task.*
```
nimex_addDigitalInput(task, '/dev1/port0/line0:7');
```
- **nimex_bindEveryNCallback.m** - *Registers a Matlab variable (most commonly a function_handle or a cell array whose first element is a function_handle) to be passed to `feval` when the everyN event occurs.*
```
nimex_bindEveryNCallback
(task, {@samplesAcquiredFcn, task, 1000}, 'bindEveryNCallbackImplementation', 0);
```
- **nimex_bindDoneCallback.m** - *Registers a Matlab variable (most commonly a function_handle or a cell array whose first element is a function_handle) to be passed to `feval` when the done event occurs.*

```
nimex_bindDoneCallback(task, {@acquisitionComplete, task},
'doneCallbackImplementation', 0);
```

- **nimex_delete.m** - *Clears the task from memory (frees all associated resources immediately.*

```
nimex_delete(task);
```

- **nimex_display.m** - *Displays the details of the task to the Matlab command-line.*

```
nimex_display(task);
```

- **nimex_getChannelProperty.m** - *Retrieves the current value for a channel property (or set of properties).*

```
bufferSize = nimex_getChannelProperty(task, '/dev1/ao1', 'dataBufferSize');
```

- **nimex_getTaskProperty.m** - *Retrieves the current value for a task property (or set of properties).*

```
[sampleRate, triggerSource] = nimex_getTaskProperty(task, 'samplingRate',
'triggerSource');
```

- **nimex_setChannelProperty.m** - *Sets the current value for a channel property (or set of properties)..*

```
nimex_setChannelProperty(task, '/dev1/ao1', 'mnemonicName', 'stimChan1');
```

- **nimex_setTaskProperty.m** - *Sets the current value for a task property (or set of properties).*

```
nimex_setTaskProperty(task, 'samplingRate', 10000, 'triggerSource',
'/dev1/PFI0');
```

- **nimex_readAnalogF64.m** - *Reads analog data in float64 (Matlab 'double') format.*

```
data = nimex_readAnalogF64(task, 10000);
```

- **nimex_readDigitalU32.m** - *Reads digital data in uInt32 (Matlab 'uint32') format.*

```
data = nimex_readDigitalU32(task, 10000);
```

- **nimex_writeAnalogF64.m** - *Writes analog data in float64 (Matlab 'double') format.*

```
nimex_writeAnalogF64(task, '/dev1/ao0', data);
```

- **nimex_writeDigitalU32.m** - *Writes digital data in uInt32 (Matlab 'uint32') format.*

```
nimex_writeDigitalU32(task, '/dev2/port0/line0:7', data);
```

- **nimex_sendTrigger.m** - *A convenience method for sending a digital pulse. May be deprecated in the near future.*

```
nimex_sendTrigger(task, '/dev1/port0/line0:7');
```

- **nimex_startTask.m** - *Begins executing the task (however a trigger condition may still need to be met before any data transfer actually begins).*

```
nimex_startTask(task);
```

- **nimex_stopTask.m** - *Stops a task. The task may be restarted at some point in the future.*

```
nimex_stopTask(task);
```

3.3.C Structures

The follwing are some of the important data structures in the libraries.

- NIMEX_channelDefinition

TABLE II DATA STRUCTURE I

DATA STRUCTURE	DATA
int32	channelType
int32	terminalConfig
int32	units
float64	minVal
float64	maxVal
void*	dataBuffer
uInt64	dataBufferSize
char*	mnemonicName
char*	physicalChannel

- NIMEX_taskDefinition

TABLE III DATA STRUCTURE II

DATA STRUCTURE	DATA
TaskHandle*	TaskHandle
NIMEX_ChannelList*	Channels
char*	ClockSource
int32	clockActiveEdge
char*	clockExportTerminal
mxArray*	userData
HANDLE	mutex
char*	triggerSource
int32	timeout
NIMEX_CallbackSet*	everyNCallbacks
NIMEX_CallbackSet*	doneCallbacks
int32	lineGrouping
float64	samplingRate
int32	sampleMode
uInt64	sampsPerChanToAcquire
int32	triggerEdge
uInt32	pretriggerSamples
int32	started
uInt32	everyNSamples
uInt32	repeatOutput

- *Demo Script*

  Here we provide a piece of demo scripts as an example .It shows the basic function of nimex.

```
function nimexDemo

menuString = sprintf('\nChoose a feature to demonstrate:\n\t[1]
Analog Input Demo\n\t[2] Analog Output Demo\n\t[3] Triggering
Demo\n\t[4] Clock Synchronization Demo\n\t[5] Continuous
Acquisition Demo\n\t[6] Digital Output Demo\n\t[q]
Quit\n\n\tSelection: ');

again = 1;
while again
    choice = lower(input(menuString, 's'));

    try
        switch choice
            case '1'
                analogInputDemo;
            case '2'
                analogOutputDemo;
            case '3'
                triggeringDemo;
            case '4'
                clockSyncDemo;
            case '5'
                continuousDemo;
            case '6'
                digitalOutputDemo;
            case 'q'
                clear mex;
            return;
   otherwise
  fprintf(2, '\nUnrecognized option ''%s''\n\n', choice);
        end
    catch
 fprintf(2, 'An error occurred while executing a specific
demonstration: %s', lasterr);
        clear mex;
        return;
    end
```

4.Some necessary hints

4.1.Memory Consideration

Because the code underlying the class is in C, memory management needs to be taken into consideration. Matlab does not provide explicit signals for when it is safe to free memory that has been allocated in mex functions, other than when mex functions are completely cleared. Because of this, it becomes necessary to explicitly release @nimex resources when they are no longer needed. The nimex_delete.m method will free all resources associated with a specific task. Any future attempts to use a deleted task will result in undefined behavior (and likely a segmentation violation will be issued). It is

important to delete a task when it is no longer needed, but not before.

All memory allocated by the nimex API is tracked by the API. Because variables do not naturally persist across mex files, the pointer to the memory manager is stored inside Matlab (similar to how a pointer to each nimex task is stored in a field of the class). To make this memory manager globally accesible to all nimex mex files, it is stored in a global variable. Any tampering with this variable will result in unspecified behavior (and will likely cause a segmentation violation, corrupt data structures, or crash Matlab). For reference, the variable is called "NIMEX_GLOBAL_PERSISTENCE_LIST". But, it should never be accessed outside of the nimex C code (and even then, it should never be accessed outside of NIMEX_memManagement.c).

Clearing all global variables before deleting all nimex task instances or clearing all mex files will result in a memory leak.

4.2.Necessary knowledge of NiMex

Familiarity with the daqtoolbox and/or NI-DAQmx libraries is helpful, but not necessary. Where possible, parameters in the @nimex class are mapped directly to their NI-DAQmx equivalents. An understanding of object-oriented programming in general, and Matlab's implementation of object oriented programming in particular, is required for a full understanding and effective use of the code. A discussion of these topics can be found in the Matlab documentation itself ('Using object-oriented programming in MATLAB').

References

[1] http://www.ni.com/

[2] *Data Acquisition Toolbox*, http://www.mathworks.cn/