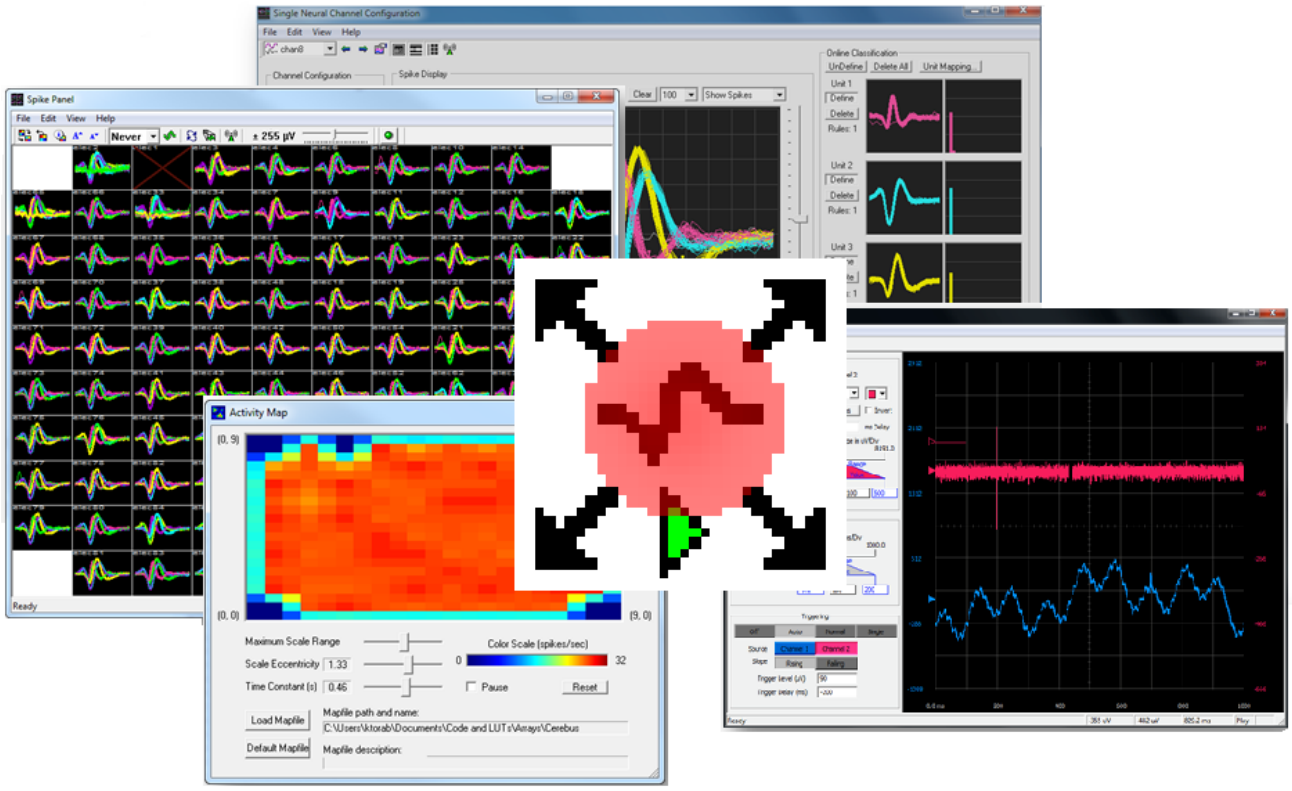
NSP Extension Code

User's Manual



(Revision 1.00)

Blackrock Microsystems®, LLC

630 Komas Drive • Suite 200
Salt Lake City • UT 84108

T: +1 801 582 5533
www.blackrockmicro.com
support@blackrockmicro.com

1.0 Table of Contents

2.0 Introduction

The latest revision of the Neural Signal Processor (NSP) for the Cerebus® System *includes* additional power to run a user defined extension that can access neural data samples in real-time for custom processing. Through an included API the user defined extension will also be able to transmit comments, using the NSP, to Central for recording and/or viewing or further processing. This API also gives an extension limited control of digital and analog output ports on the NSP including triggering analog or digital outputs, and full control of analog output channels.

While the system is online users will have the ability to send commands to their extension through the use of comments either from Central, ExtensionLoader, or cbmex/cbsdk (CereLink). Users will also have limited access to the NSP system through a special profile. This profile is meant for loading and building extensions directly on the NSP. Commands for utilizing this profile are issued through ExtensionLoader or CereLink.

In this manual you will find *information* on system setup, loading and building an extension, and how to interact with the extension including an example script. For further information on the API please refer to the searchable API reference at http://www.blackrockmicro.com/extension/.

3.0 System Setup

3.1 SYSTEM SCHEMATIC

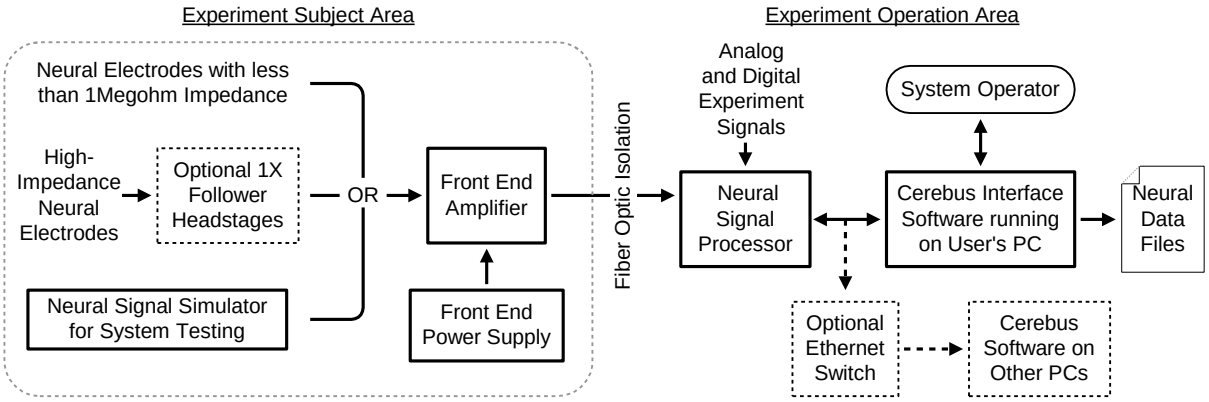


Figure 1 - System Overview

Connect your NSP to the PC running Cerebus® Central Software as displayed in the schematic above.

4.0 Commands

4.1 OPEN

Description

Opens the connection to the NSP.

Format

ExtensionLoader:	Click the Connect button
cbMEX	cbmex('open')

Inputs

None

4.2 CLOSE

Description

Closes the connection to the NSP

Format

ExtensionLoader:	Click the Disconnect button
cbMEX	cbmex('close')

Inputs

None

4.3 SYSTEM

Description

Sets the current state of the NSP.

Format

ExtensionLoader:	buttons – State: Run, State: Standby
cbMEX	cbmex('system', command);

Inputs

Reset	Performs a soft reset of the NSP and sets it to 'running'
Shutdown	Shuts down the NSP.
Standby	Sets the NSP into standby. This mode is necessary for uploading files to the NSP.

4.4 UPLOAD

Description

Uploads files to the NSP. This is used to build and manage extensions directly on the NSP. The ExtensionLoader allows you to browse to a file to upload. In addition, you can specify the destination directory by simply adding a folder to the end of the file to upload then click the Upload button. The dropdown contains a history of files. E.g. *c:\projects\myExtension\myExtension.c myExtension* will upload the file myExtension.c to the folder myExtension. This will allow you to have multiple projects on the NSP.

Format

ExtensionLoader:	Filename, Browse button, and Upload button
ExtensionLoader:	upload, filename
cbMEX:	cbmex('ext', 'upload', 'filename');

Inputs

Filename	The full path to the file as a string (e.g., 'c:\projects\myExtension.c').

Examples

Upload the tar file extension_simple.tar

ExtensionLoader:	browse to extension_simple.tar, click Upload
cbMEX:	cbmex('ext', 'upload', 'extension_simple.tar')

Upload the file extension_simple.c to the folder myExtension

ExtensionLoader:	browse to extension_simple.c, add myExtension to the upload line, click Upload
cbMEX:	cbmex('ext', 'upload', 'extension_simple.c') cbmex('ext', 'command', 'mv extension_simple.c myExtension')

4.5 TERMINATE

Description

It sends a kill signal to the current running process (if there is one).

Format

ExtensionLoader:	terminate,
cbMEX:	cbmex('ext', 'terminate');

Inputs

None

4.6 QUIT PLUGIN

Description

Unloads the plugin for the current running session of the NSP. Next time the NSP boots, it will execute the plugin again.

Format

ExtensionLoader: quitplugin
cbMEX: cbmex('ext', 'quitplugin');

Inputs

None

4.7 COMMAND

Description

Tells the NSP to interpret the command string as a linux command.
ExtensionLoader assumes any command typed will be a linux command so if any of the

Format

ExtensionLoader: command, linuxcommand
cbMEX: cbmex('ext', 'command', '*linuxcommand*');

Examples

Run the make command on the Makefile in the myExtension directory
ExtensionLoader: make –C myExtension
ExtensionLoader: command, make -C myExtension
cbMEX: cbmex('ext', 'command', 'make -C myExtension')

List the contents of myExtension directory
ExtensionLoader: ls myExtension
ExtensionLoader: command, ls myExtension
cbMEX: cbmex('ext', 'command', 'ls myExtension');

Clear the display
ExtensionLoader: clear
cbMEX: cbmex('ext', 'command', 'clear');

Remove an extension
ExtensionLoader: rm lib/nspext_default.so
ExtensionLoader: command, rm lib/nspext_default.so
cbMEX: cbmex('ext', 'command', 'rm lib/nspext_default.so');

4.8 COMMENT

Description

The comment can be used to send variable data that can be interpreted by an extension however it chooses. This is the main method by which to communicate with a extension while the NSP is online.

Format

ExtensionLoader:	comment, rgba, charset, comment
cbMEX:	cbmex('comment', rgba, charset, comment)
Central:	click the Add Comment button and type a comment (Note: rgba & charset are both set to 0)
Central or Raster:	Start typing a comment

Inputs

rgba	32-bit integer color coding in RGBA format
charset	8-bit character set. 0 - ASCII, 1 - UTF-16
comment	The comment string up to 127 characters

Examples

Change the output waveform to analog output 2

ExtensionLoader:	comment, 0, 0, aoutchan 2
cbMEX:	cbmex('ext', 'comment', 'aoutchan 2')

5.0 Building and Loading an Extension

5.1 Prebuild Requirements

In order to build an extension, certain requirements must be met for the extension to build and be recognized by the NSP. Please note that the included sample files demonstrate these requirements.

- Any source file accessing the API includes 'nspChanTrigExtension.h' and 'nspExtension.h'.
- Make sure all common files are included in the Makefile at the section "CommonDir".
- Make sure all source files are included in the Makefile at the section "PLUGIN_SRC".
- It is recommended to place the following 3 files in a common folder: 'nspChanTrigExtension.h', 'nspExtension.h' and 'libnsp.a'. (see steps below)

To load the common files, start the ExtensionLoader application. It is an icon on the Windows start menu under the default Blackrock Microsystems group. Perform the following steps:

1. Click connect
2. Click State: Standby
3. In the command line at the bottom of the windows, type "mkdir ExtensionCommon" and press Enter
4. Click the … button and browse to the location of libnsp.a
5. Add " ExtensionCommon" (be sure to include the preceding space) to the line containing the file
6. Click Upload
7. Repeat steps 4-6 for nspChanTrigPlugin.h and nspPlugin.h

5.2 Building Procedure

Once the above requirements have been met the extension can be compiled. The following steps describe how to run the build procedure.

1. For using the Makefile the following commands have been provided; type the bold terms into the command line to initiate the associated action:
    a. **Make -C *foldername*** - This will compile and build the extension or display any errors should there be any. The built extension will be named 'nspext_default.so' (this is necessary for the nsp to recognize it) and will be placed into the proper folder to run next time the NSP is rebooted.
    b. **make -C *foldername* clean** - This will remove the built extension and any temporary object files. Useful for making a 'fresh' build.

5.3 SAMPLE EXTENSION 1

This extension shows the main structure of extension code, but is not very useful. It will send a log packet containing "Hello World!" to Central when it receives a comment from the user.

To build and run it, do the following:

1. Click connect
2. Click State: Standby
3. In the command line at the bottom of the windows, type "mkdir SampleExtension1" and press Enter
4. Click the … button and browse to the location of SampleExtension1.c
5. Add " SampleExtension1" (be sure to include the preceding space) to the line containing the file
6. Click Upload
7. Repeat steps d through e for "Makefile"
8. In the command line at the bottom of the window, type "make –C SampleExtension1"
9. Click the "Reboot" button to restart the NSP and the extension will run

To interact with this extension, do the following

1. Start Central then click Connect on the ExtensionLoader
2. The only interaction with this extension is that it will display a message in ExtensionLoader when a comment is received. To show that it works, In Central, type a comment and you should see "Hello World!" on the ExtensionLoader window.

5.4 SAMPLE EXTENSION 2

This extension does the following:

1. Trigger a digital output based on receiving a comment packet
2. Trigger an analog output based on receiving a comment
3. Invert the signal of a front end channel on an analog output channel
4. Send a log packet containing "Test Operation…" when a comment of "Test Operation" is received
5. Return the proctime when a comment of "Proctime" is received

To build and run it, follow the steps below:

1. Click connect
2. Click State: Standby
3. In the command line at the bottom of the windows, type "mkdir SampleExtension2" and press Enter
4. Click the … button and browse to the location of SampleExtension2.c
5. Add " SampleExtension2" (be sure to include the preceding space) to the line containing the file
6. Click Upload
7. Repeat steps d through e for "Makefile"
8. In the command line at the bottom of the window, type "make –C SampleExtension2"
9. Click the "Reboot" button to restart the NSP and the extension will run

To build and run it, follow the steps below:

1. Connect a BNC cable from Digital Output 1 to Analog Input 1
2. Connect a BNC cable from Analog Output 1 to Analog Input 2
3. Connect a BNC cable from Analog Output 2 to Analog Input 3
4. Start Central
5. Configure Front End channel 1 to continuous 30kS/s
6. Configure Analog Inputs 1-3 to continuous 30kS/s
7. Start Raster
8. Configure Raster to show chan1, ainp1-3
9. Configure Digital Output 1 to be Triggered by Extension and Samples High set to 30000
10. Configure Analog Output 1 to Sine Waveform, Triggered by Extension with an Amplitued of 5000 and a Frequency of 1Hz and Repeats of 5
11. Configure Analog Output 2 to Function Extension
12. Click Connect on the ExtensionLoader
13. In Central, type "Proctime" and you should see the current proctime on the ExtensionLoader window
14. In Central, type "Digital Trigger" and watch ainp1 on Raster. You should see a positive going 1 second pulse
15. In Central, type "Analog 1 Trigger" and watch ainp2 on Raster. You should see a series of 5 sine waves
16. In Central, type "Analog 2 Output" and watch ainp3 on Raster. You should see ainp3 is inverted signal of chan1