Appendix to the Bachelor project entitled
"A physiologically driven computer interface using EMG"

BCI2000 v2.0 tutorial

Author:
Jeremy Davis

Supervisors:
Prof. Thierry Pun
Guillaume Chanel
Mohammad Soleymani

Computer Science Department
University of Geneva

September 16, 2008

Chapter 1

Introduction

This documentation is a tutorial for the BCI2000 software (version 2.0). It will cover every step I had to take for my project ("A physiologically driven computer interface using EMG"), including the download of the source code, building of the code, and installation. The creation of an acquisition module will not be covered, since I used Biosemi2ADC and most hardware has an acquisition module for BCI2000 ready for use.

If anything is unclear, you can always find more information on the BCI2000 wiki or the forum [1].

Chapter 2

First Steps

This chapter will explain how to download the source code, install the software, and do run a test.

2.1 Requirements

Before starting, you must have Borland C++ Builder 6 or more recent because the operator and most application modules rely on the Borland VCL library. If you do not have it, you will have to get it somehow.

This also means that you will have to use Windows XP, as it is the only operating system that supports Borland C++ Builder. In the future (no date announced), BCI2000 should be platform independent [2].

You will also need some knowledge of C++, since everything will be coded in that language. Fortunately, if you are not comfortable with it, it is easy to find documentation and tutorials.

2.2 Download

The first step is of course to download the source code. For that you will need a login and password that will be granted to you freely (after you fill a legal form) if you are part of a research institute or university. Just head on the main download page [11] and follow the instructions. If your supervisor (or someone else) already has an account, he should be able to give you access to the source code.

The source code is then accessible via Subversion (SVN). It is a version manager for projects, so people working on the same project can modify different parts without having to count them, or inform the others of the changes. It is a successor to Concurrent Versions System (CVS). The wiki [10] explains how to download and use TortoiseSVN in case you do not have a SVN client and then how to set up your client to download the source code.

Once you have downloaded the source code, you should have a folder with these contents:

- batch - Contains instructions for batch demonstrations.
- data - Contains data samples.
- doc - Contains the BCI2000 wiki in full.
- parms - Contains examples of parameter files, which are saved configurations (see section 2.5).
- prog - Contains all compiled modules. And the default folder where your compiled modules will be saved.
- src - Contains the source code.
  - buildutils - Command line utilities for compilation (mostly formatting)
  - contrib - Contains all the modules developed by third parties (that means you should save your work in here).
    - AppConnectorApplication
    - Application - For application modules
    - SignalProcessing - for signal processing modules
    - SignalSource - for acquisition modules
    - Tools
  - core - Contains all the modules developed by BCI2000. It also has the source code for the launcher and the operator. It contains basically the same folders as contrib, to keep the modules organized by role.
  - doc - Documentation creation files.
  - extlib - Contains all external libraries (.lib files).

  - shared - Contains all the code files that are meant to be used by modules. They are sorted by use (types, modules, fileio, bcistream, etc...). Most filters are here (under modules) so as to be available to more than one signal processing module or application. All the super-classes are here too (GenericFilter, ApplicationBase, etc...).

- tools - Contains miscellaneous tools.
- BCI2000_Help.
- GettingStarted.

This list omits most subfolders because they are either explicit or empty.

2.3 Installation

I recommend copying the "src","prog" and "tools" folders to another location, so as not to risk problems in case you choose to update your files with SVN. Updating the project may change more than you expect and force you too modify your code to fit the new protocol. When you compile the different modules, the executable will be created in the "prog" folder.

The next step is to install the acquisition module for Biosemi's Active2. If you do not plan on using this, you will have to find out how to install your hardware's module on your own. Start by checking the contributions page of the wiki [3].

Bisoemi2ADC is part of the BCI2000 source code you have downloaded in the previous step. You can find it in the "src/contrib/SignalSource/Biosemi" directory. There you will find a file named "Labview_DLL.dll". You must copy it to the "prog" folder.

Now that everything is installed, you are ready to test BCI2000.

2.4 General Architecture

The BCI2000 operator is the coordinator of the different modules, as shown in figure 2.1.

This diagram also shows the communication flow between each of the three modules. The acquisition module is the interface with the hardware you are using to record the physiological signals. Its purpose is to convert the signals into a communication package that the next module will understand, following the BCI2000 protocol. The signal is sampled at a chosen rate and passed on as blocks of a chosen size. For more information on the package architecture, see the wiki page [7].

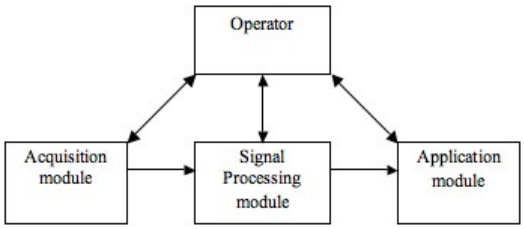

Figure 2.1: BCI2000 Diagram

The signal processing module processes each sample block to compute a new ouput vector that the application module can use as input.

2.5 BCI Launcher setup and test

With all that, we can start compiling.

Launch Borland C++ Builder and open the project named "BCI2000" in the "src" folder. Then choose **"Project" → "make all projects"** from the top menu bar. This can take quite a while (10-15 minutes, depending on your computer).

After that, try launching BCI2000 by double-clicking the "BCI2000launcher" icon in the "prog" folder. You find yourself with a window that has four compartments in the top half (Figure 2.2).

You will have to put some order in the list of programs under "others" on the far right. Select any and right-clic it. You will have the option of moving it to Source, Signal Processing, or Application. For now, you can just place SignalGenerator under Source, the DummySignalProcessing under Signal Processing and the FeedbackDemo under Application.

The bottom half of the window has two regions. The left part ("Parameters") enables you to load a parameter file before launching. This will be explained below. On the right, you can enter the IP address of the computer hosting each module. By default, all modules are on the same computer, so the IP is 127.0.0.1, for localhost. If you want to divide the modules among two or three computers, this is where you define their location, so the operator knows where to access them.

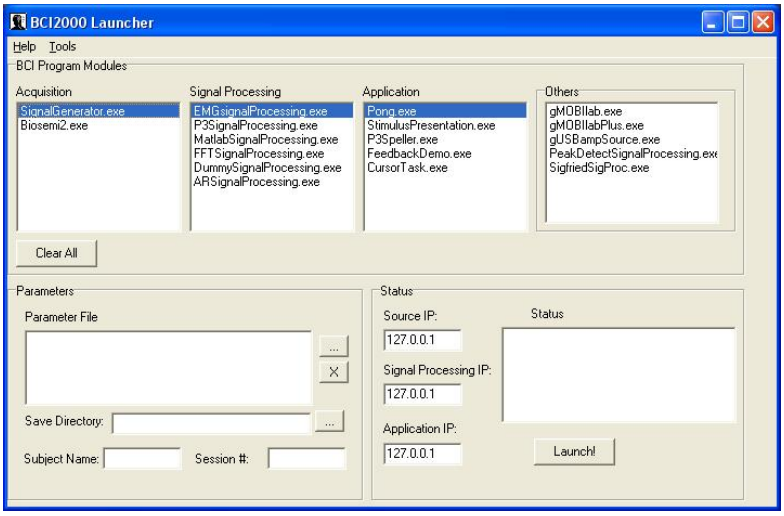

Figure 2.2: BCI2000 Launcher

Select one module of each role (acquisition, processing, application) and clic launch to set them up in a chain. You will always have to select exactly one module of each when launching BCI2000.

Now you have another window (see figure 2.3). Clic "Config" to bring up the configuration options. These are all defined by the three modules you have selected. Each module has its own set of options (or parameters). The parameters are divided under different thumbnails, depending on their definition. For instance, the parameters for visualization (mostly booleans for "visualize ...filter output") are under the "Visualize" thumbnail, while filter parameters are all under "Filtering", in different sections (Fig 2.4).

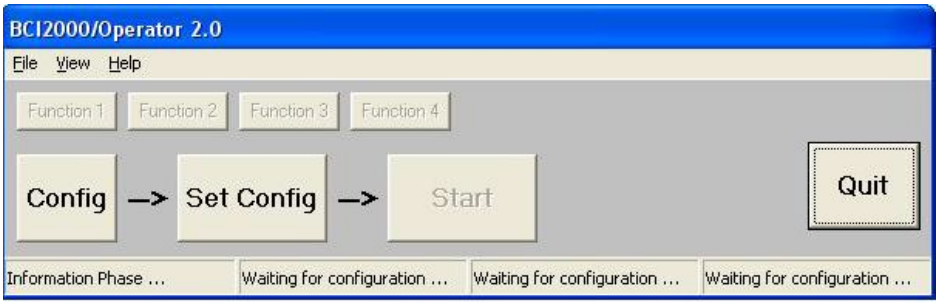

Figure 2.3: BCI2000 operator

Each time you close the operator, the configuration is lost, so be sure to save any configuration you are happy with. The saved configuration file is

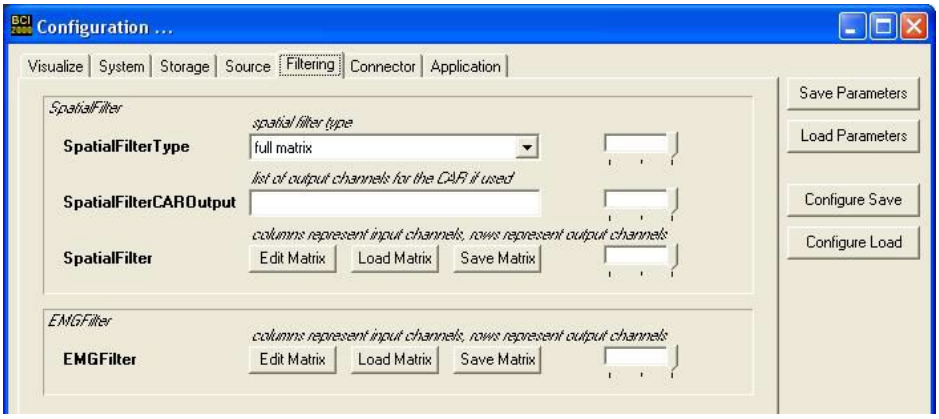

Figure 2.4: BCI2000 configuration window

then loadable from the configuration window, or beforehand in the "Parameter File" box of the BCI2000 launcher.

Now close the configuration window. If you clic on "Set Config", you will see the application screen appear, as well as different windows. You can then clic "Start" to launch the acquisiton, signal processing and application modules.

You can also start the different modules manually with the command prompt by simply calling the "start" command followed by the name of the module and its IP. You must always start with the operator though; "start operat.exe". You can also write a batch file. Some examples are in the "batch" folder.

If everything went well, we can start working on the code. If not, it is possible that your revision (the version of BCI2000 you have downloaded) is not completely functional. Posting in the forum [1] will surely get you the answer to your problem, but do not forget to check the wiki beforehand.

Chapter 3

Signal Source

As mentioned in the introduction, no modification will be made to the code of the Biosemi2ADC module. If you are using another piece of hardware, please refer to its manual [3] for installation of the acquisition module.

The Biosemi module offers quite a collection of parameters which can be found under the "Source" thumbnail in the configuration window. The most important ones are:

1. SampleBlockSize: Defines the size of each block.

2. SamplingRate: The recording rate, in Hertz. Watch out for the Nyquist-Shannon sampling theorem [12]. (To preserve a signal up to a certain frequency, the sampling rate must be at least double that frequency)

3. TransmitChList: The channels you want to forward to the filter chain. For instance, the external electrodes (EX) are relayed on channels 233 to 240 (EX1 : 233, EX2 : 234, ..., EX8 : 240).

You should also watch out for the number of channels (SourceCh) and their gain/offset.

Chapter 4

Signal Processing

4.1 Introduction

In BCI2000, a processing module just calls a number of individual filters, which makes it very easy to switch two filters or replace one by another. Moreover, one filter can be used by many modules. The signal blocks are transmitted automatically from one filter to another. Figure 4.1 shows a diagram representing the structure of a signal processing module.

As an example, the Spatial Filter is almost always used. Each module that needs it just calls it. There is no need to copy it or duplicate the code. More details can be found in Section 4.3.4.

We will start by creating the signal processing module, which groups the different filters together, and see how we can use already created filters. Then we'll see how to create a new filter and integrate it to our signal processing module.

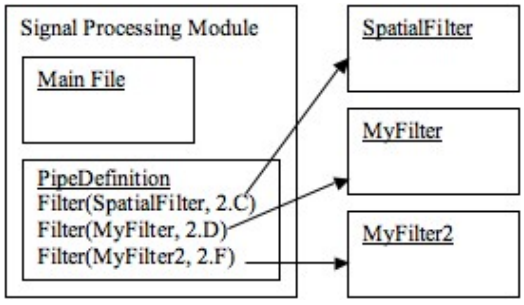

Figure 4.1: Signal Processing module

4.2 The main module

To create a signal processing module, launch C++ Builder and create a new project by clicking **"Project" → "Add New Project..."** and then "Application". This will also create a file and a form. You can close the form and the file. You will be prompted with a message to save the file. Do not; it is useless. Instead, create a new file by clicking **"File" → "New" → "Other..."** and then "Cpp file". Copy the following code in the file (you can copy/paste from almost any other Signal Processing main file), then save the project (**"File" → "Save Project as"**) in its own folder (under "src/contrib/signalprocessing/") and name it accordingly (the project is the module). You will also be asked to save the file. You should give it the same name.

Listing 4.1: YourProject.cpp

```
#include "PCHIncludes.h"
#pragma hdrstop

#include <vcl.h>
#include "CoreModuleVCL.h"

WINAPI
WinMain( HINSTANCE, HINSTANCE, LPSTR, int ) {
  try {
    Application->Initialize();
    CoreModuleVCL().Run(_argc, _argv );
  }
  catch (Exception &exception) {
    Application->ShowException(&exception);
  }
  return 0;
}
```

The PCHincludes header is necessary for your module to be able to call BCI2000 functions in order to follow the protocol (such as the "Filter" function without which you could not use your filters). The pragma instruction is a question of precompilation options. The "vcl" includes are for the user interface (Borland VCL Library).

The next step is to create a "PipeDefinition" file. The name of the file is important. Create a new "Cpp file" in the same project and start by writing

the following lines:

Listing 4.2: PipeDefinition.cpp

```
#include "PCHIncludes.h"
#pragama hdrstop

#include "SpatialFilter.h"

Filter( SpatialFilter, 2.C );
```

This file is the description of the filter chain. Include all the filters you want or need, and call them with the "Filter" function. The "2.C" part is the order in which the filters will be set. The first digit should follow the following protocol [4]:

1. filters in source modules
2. filters in processing modules
3. filters in applications

This means all filters called in the signal processing module will have a rank of 2 (while application filters have a rank of 3). The filters are then applied according to the relative alphabetical position of their letter (2.B comes before 2.E).

For now, the Spatial Filter is enough to test your module. You can add other filters later.

The last step is to set up the links. Choose **"Project" → "Add to project"** and choose the PipeDefinition file you've just created. Then repeat this for all cpp files in the different folders under "src/shared". The reason for this is because the "PCHincludes" file you included is the tip of the iceberg that is the hierarchy of files that compose BCI2000. It is possible that you will not be using all of them, but it is easier to add everything than to sort through them. If you miss any, you will be notified by C++ Builder at the end of the compilation. It will say "[Linker Error] unresolved external..." and then the name of the missing file, followed by error codes.

You will also have to make sure the following options are correct. Clic **"Project" → "Options"**.

- Compiler: In the "File name" field: "..\..\..\shared\obj\bci2000.csm"

- Linker: Under Linking, the "Use Dynamic RTL" box is not checked.
- Directories/Conditionals
  - The "Include path" field has the following, in any order:
    * ..\..\..\shared
    * ..\..\..\shared\accessors
    * ..\..\..\shared\bcistream
    * ..\..\..\shared\config
    * ..\..\..\shared\fileio
    * ..\..\..\shared\gui
    * ..\..\..\shared\modules
    * ..\..\..\shared\modules\signalprocessing
    * ..\..\..\shared\types
    * ..\..\..\shared\utils
    * ..\..\..\shared\utils\Expression
    * ..\..\..\extlib\matlab
    * $ (BCB)\include
    * $ (BCB)\include\vcl
    * The folder containing your module (probably contrib\...)
  - The "Library path" field contains:
    * $ (BCB)\lib
    * $ (BCB)\lib\obj
  - "Intermediate output" = "obj"
  - "Final output" = "..\..\..\..\prog\" - This defines where your executable will be created. If you want it elsewhere, just change the path, knowing that it is relative.
  - "Conditional defines" = "MODTYPE=2" - The number is the type of module (1 for acquisition, 2 for signal processing, 3 for application)

You can now try compiling your project. Then open the BCI2000 launcher again and move your module from the "Other" box into the "Signal Processing" box. Repeat the same steps mentioned in the previous chapter.

4.3 The filters

Now that you have the signal processing module ready, you have to create your own filters. Start by creating a new unit (**"File" → "New" → "Unit"**). Save it right away to change it's name.

4.3.1 Header

Open the header (there is a thumbnail at the bottom of the window).

Include "GenericFilter.h" and define your filter as a class inheriting from "GenericFilter" (class YourFilter : public GenericFilter). This class is inherited to be sure you have implemented all necessary methods to fit the BCI2000 protocol.

The filter requires three public methods (apart from the constructor and destructor):

- virtual void Preflight(const SignalProperties&, SignalProperties&) const;
- virtual void Initialize(const SignalProperties&, const SignalProperties&);
- virtual void Process(const GenericSignal& Input, GenericSignal& Output);

These methods are part of the BCI2000 protocol. They will be called in turn when you start the whole process, the Preflight and Initialize methods when you set the configuration, the Process method for each sample block while the application runs.

You may also want to implement some optional methods of the GenericFilter class. For more information about these, check the wiki [6].

4.3.2 C++ File includes and declarations

Now switch back to the filter file (.cpp). Include "PCHIncludes.h" and any other file you might need (for those, you should open the "src/shared" folder).

Keep #pragma hdrstop right after the inclusion of "PCHIncludes.h" but delete the other #pragma. Again, this is a precompiler command. You will need "using namespace std;" to activate i/o.

Then call "RegisterFilter" with the name of your filter and its default rank: "RegisterFilter(<YourFilter>, 2.C)" for instance. You will be able to place your filter anywhere afterwards, so do not spend time thinking about the default rank.

If you have any parameters to your filter, you will have to declare them in the constructor. They will then appear in the configuration window under the specified thumbnail and section. This is done by writing

`"BEGIN_PARAMETER_DEFINITIONS"`

and

`"END_PARAMETER_DEFINITIONS"`

in the body of the constructor. Parameters are then declared between these two instructions and following this protocol:

`<Section> <type> <name>= <value> <default> <Min> <Max> // <Comment>`

This can change if the type is a list or matrix. Everything is explained in detail on the wiki [8]. The section should be Filtering, or maybe Visualize if the parameter only changes the visual output. This serves to place the parameter in the right thumbnail in the configuration interface of the operator.

Each parameter must be in between quotes and separated by a coma, even the last one before "END_PARAMETER_DEFINITIONS".

For instance:

Listing 4.3: Parameter definitions

```
BEGIN_PARAMETER_DEFINITIONS
  "Filtering int frequency= 18 18 5 50"
    "// the frequency of something in your filter",

  "Filtering int box= 0 0 0 1"
    "//this is a boolean parameter, it makes a  →
       ↪ checkbox because of the boundaries that  →
       ↪ allow only 0 or 1",
END_PARAMETER_DEFINITIONS
```

State variables, which are used somewhat like global variables for all three modules, are defined in the same way, if you need some. More details are on the wiki [9].

4.3.3 Necessary methods

Then comes the preflight method. It is called when you set the configuration. Its purpose is mainly to check the different parameters. If you have bound conditions, you should check them by using the Parameter function with the name of the parameter. For instance: "Parameter("frequency");" You can also write special conditions, such as dependencies between two parameters. States are also checked that way, but with the State function. The last thing you should check (if it is important) is the correspondence between the input and the output properties. This is done easily: "OutputProperties = InputProperties", InputProperties and OutputProperties being the first and second parameter of the Preflight method.

The Initialize method is called once at the beginning, just after the preflight method. Any variable you need to initialize should be treated here.

Finally, the Process method is the core of your filter. This method is called for each signal block. That means you do not need to create a loop to work on the different blocks. Input is a two-dimensionnal matrix. The first dimension represents the channels, the second the samples. The size of the second dimension thus depends on the size of blocks you transmit. For instance, if you transmit 4 channels at 32 samples/block, you will get a 4x32 matrix as Input for each call of Preflight.

4.3.4 included filters

This short list of filters may prove useful. Be sure to check your own list of filters, as some may have been added.

- SpatialFilter - Applies a linear transformation to the signal.
- FFTFilter - Does a Fast Fourier Transform on the signal.
- Normalizer - Can modify the gain and offset of the signal. Can also modify the signal to have a mean of zero.
- RandomFilter - Multiplies the signal by random zero-mean noise and outputs the result to extra channels.
- LPFilter - A simple low-pass filter.

For more details on each filter, check the BCI2000 wiki [5].

There are other filters available. Just open the "src/shared/modules/signalprocessing/" folder and open them in Borland C++ Builder to see what they are about (there is a comment at the top of the code that explains the filter's use).

Chapter 5

Application

The application module is the final step. It should be the goal of your project. For instance, my work consisted in implementing a game of pong as biofeedback.

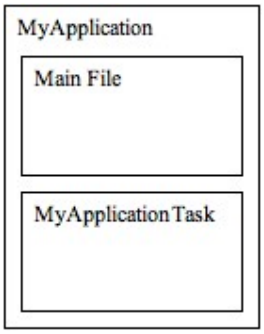

Figure 5.1: Application module

5.1 Application Main Setup

Close the existing project (**"File" → "Close all"**) before continuing. Creating a new project like you did for your signal processing module:

1. **"Project" → "Add New Project..."**
2. Close the two windows without saving.
3. Create a new cpp file with the same code as in the signal processing module.
   (you can add a title by writing *Application->Title = <your Application Title>*);
4. Add all shared files to the project, the same way you added them to your signal processing module.
5. Check that the project options are the same as for the signal processing module, except that "MODTYPE=3" should replace "MODTYPE=2"

6. "Save project as" your application name in a new folder under contrib\Application \<yourAppName>.

5.2 Application Task

You can now create a new unit. Again, you can erase the "#pragma package(smart_init)" from the cpp file.

5.2.1 Header

In the header, you will need to include "ApplicationBase.h". Then you can declare your application class. It has to inherit from ApplicationBase (public derivation). It usually has the "Task" suffix. For instance, the "Pong" application has a "PongTask" class.

In the public field, you will have to declare the inherited methods as well as the constructor and destructor:

- <Name of the class> (const GUI::GraphicDisplay* = NULL);
- virtual ~ <Name of the class>();
- virtual void Preflight(const SignalProperties&, SignalProperties&) const;
- virtual void Initialize(const SignalProperties&, const SignalProperties&);
- virtual void Process(const GenericSignal&, GenericSignal&);
- virtual void StartRun();
- virtual void StopRun();
- virtual void Halt();

These methods will be explained later on.

In the private field, you will probably want a TForm pointer to create a window and a couple of shapes and/or labels. These are all part of the Borland VCL library mentioned in the requirements section (2.1). For instance, the Pong game uses these:

class TForm * window;

class Tlabel * textLabel;

class TShape * paddle1;

class TShape * paddle2;

class TShape * ball;

5.2.2 C++ File includes and declarations

You will have to include the usual "PCHIncludes.h" and <vcl.h>. It is a good idea to include "Color.h" too. You may want to include "Localization.h" if you plan to translate your application.

You then have to register the filter, the same way you did the signal processing one. The rank has to be 3 since it is an application filter.

5.2.3 Constructor and Destructor

The constructor has to inherit from ApplicationBase, as well as create the class-typed attributes of your class. Here's an example from my PongTask class:

Listing 5.1: PongTask constructor

```
1 PongTask::PongTask( const GUI::GraphDisplay* inDisplay →
    ↪ )
  : ApplicationBase( inDisplay ),
3   window( new TForm( reinterpret_cast<TComponent*>( →
      ↪ NULL ) ) ),
    textLabel( new TLabel( window ) ),
5   ball( new TShape( window ) ),
```

The content of the constructor is basically the same as in the signal processing filter. Start by defining your parameters and state variables if you have any. The protocol is the same as before.

You can also start personalizing the application window (border, color, etc.).

The Destructor should at least call the "Halt" method and delete the window ("delete window;").

5.2.4 Main methods

The Preflight method is used the same way as in the filter. You should at least call each Parameter (*Parameter("ParamName");*) to check the boundaries. Check all other conditions with normal "if" instructions. Also, make good use of the "bcierr" and "bciout" output streams, knowing that the first blocks the process, the second being just a warning.

The initialize method should set every Tform, Tshape or Tlabel parameters (size, color, visibility, position, etc.), and anything else you need to be done before the application starts running. It is also a good place to "transfer" the Parameters to variables.

The StartRun and StopRun are called when you "Suspend" or "Resume" the application (StartRun is also called at the beginning). Usually they will have a "Pause" message appear or disappear in the middle of the screen. The actual pause in the process is done automatically.

The Halt method is called when the application is stopped. It is mostly used by the acquisition module, but you may have a use for it if your application initiates asynchronous operations such as executing threads or acquiring data.

The Process method is the same as for the signal processing filter. It is the core method, but do not hesitate to create private methods to "divide" the work and have clean code. Remember that the function is called for each block, so the timing is hard, since it depends on the sampling rate and the sample block size, which are determined in the source module.

Chapter 6

Final step

Once all modules are implemented, you can build (Ctrl+ F9, or **"Project"→ "Make ..."**) your module. Then open the launcher, and move your application from "Others" to the "Application" box. Then choose the source and the signal processing module and start the test!

Do not hesitate to open other applications and filters to get examples of how things are done.

Bibliography

[1] *BCI2000 forum.* http://bbs.bci2000.org.

[2] *BCI2000: Roadmap.* http://www.bci2000.org/tracproj.

[3] *BCI2000 wiki: ADC Contributions.* http://www.bci2000.org/wiki/index.php/Contributions:ADCs.

[4] *BCI2000 wiki: Filter Chain.* http://www.bci2000.org/wiki/index.php/Programming_Reference:Filter_Chain.

[5] *BCI2000 wiki: Filters.* http://www.bci2000.org/wiki/index.php/User_Reference:Filters.

[6] *BCI2000 wiki: GenericFilter Class.* http://www.bci2000.org/wiki/index.php/Programming_Reference:GenericFilter_Class.

[7] *BCI2000 wiki: Message protocol.* http://www.bci2000.org/wiki/index.php/Technical_Reference:BCI2000_Messages.

[8] *BCI2000 wiki: Parameter Definition.* http://www.bci2000.org/wiki/index.php/Technical_Reference:Parameter_Definition.

[9] *BCI2000 wiki: State Definition.* http://www.bci2000.org/wiki/index.php/Technical_Reference:State_Definition.

[10] *BCI2000 wiki: BCI2000 Source Code.* http://www.bci2000.org/wiki/index.php/Programming_Reference:BCI2000_Source_Code.

[11] *BCI2000 download page.* http://bci2000.org/BCI2000/Download.html.

[12] *Wikipedia: Nyquist Shannon sampling theorem.* http://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem.