DSPExplorer User’s Manual

Introduction

DSPExplorer is an Integrated Development Environment dedicated to learning about Digital Signal Processing. It allows the user to write and run DSP software in a familiar, highly interactive self-contained environment.

As with many interactive working environments, DSPExplorer is best described using videos. However, these videos can never convey all the details the user might wish to understand. Further, there are times when the user will want a short description of a feature without wading through lengthy videos. Hence this document.

This document attempts to give a short description of each of the many features of DSPExplorer. THIS DOCUMENT IS NOT A TUTORIAL. It is assumed the reader knows how to navigate around the various screens and understands the basic workflow.

Organization of DSPExplorer

There are really 6 screens:

- Schematic
- Source
- ‘Scope
- console
- error
- log

Note that the console and error screens overlap and only one is visible at a time. DSPExplorer attempts to bring the correct one to the front but you can change it by clicking on the appropriate label.

The ‘log’ screen is essentially unused and probably should be removed.

There are three toolbars:

- Global Menu Bar (Project Module Simulation Print)
- Module Menu Bar (Single Abort file.mod)

- ‘Scope Menu Bar (sigScale vScale SpecScale ……)

Menu items are specified using a path specifier such as “Global>Open Schematic” or “Scope>Trigger>Window>Blackman”.

Some in the ‘Scope section, menu items can change their names based on their present value. This can be confusing in this document but makes sense as you are operating DSPExplorer. When this occurs, the path will be given with “[]” braces showing the various possibilities.

Finally, some menu options cannot be described in detail in this manual because their values depend on the design. A prime example would be selecting a signal to display. When this happens this document will use the “<>” braces to indicate that the selection is design dependent and an example is being shown.

FILE STRUCTURE

DSPExplorer operates in a ‘playspace’ which is a directory and all its subdirectories on the host file system. Generally speaking, files are created in that playspace (or subdirectories of that playspace). It is possible to include files from outside the playspace but those files will be read only. To ‘import’ a module from outside the playspace, simply add it to the schematic and then “Module>Clone” the module. This will create a local module with the same contents. Don’t forget to delete the original (kind of like Arnie’s “The Sixth Day”.)

Importing a schematic is problematic because DSPExplorer doesn’t take note of where the schematic is when looking for the modules in the schematic. This may get improved but improving this behavior is pretty far down on the feature list.

As stated above, the playspace can have subdirectories. There are two kinds of subdirectories. A subdirectory which has the extension ‘.lib’ is taken to be a read only directory. This is a good way to import a library of modules you don’t want to modify. A subdirectory which does not have a ‘.lib’ extension is take to be writable. Of course, if the files themselves are marked as read only by the host, they cannot be written.

DSPExplorer keeps the file on the disk ‘current’. Each time you make a modification to the design, DSPExplorer will write the update to the file. Thus, there is no ‘save’ command as things are always being saved.

The general goal of DSPExplorer is to allow you to stop what you are doing at any time and to return to that design at will WITHOUT remembering to save things. If you close the window, the goal is to have had everything already saved. If you change schematics, the schematic will already have been saved. And so forth.

EXECUTION

A quick note on execution. When told to run, DSPExplorer executes the source code of the modules. The order of execution is dictated by the physical placement of the modules in the schematic. Execution proceeds left to right. It is the position of the upper left hand corner of the module which specifies ‘position’.

Signals between modules generally go left to right as well but this is NOT a requirement. Feedback is possible. These signal traces look ugly. Sorry. They are rare enough I just didn’t bother to make them pretty.

Module outputs are ALWAYS on the right and inputs are ALWAYS on the left. There is no other way. Again, this sometimes make things a little ugly BUT there is never any ambiguity either.

Signal Selection

In the scope area there are two times when you will want to select a signal: to display the signal on the ‘scope screen and to select a signal on which to trigger. Signal selection is a cascading menu starting with the module name and followed by the ‘upper level’ signals in the module. Signals declared inside curly braces are NOT visible for triggering or display purposes.

Schematic Editing

DSPExplorer’s schematic editor is very simple. Items are added using “Module>Add”. Things are deleted by moving them to the waste basket.

Each input of a module has a default wire. The left end of the wire can be dragged to the output of any (other) module. A wire can be selected by clicking on (or very near) any piece of the wire but it is always the left end that is 'moved'… never the right end.

Modules are moved around using click/drag paradigm.

An unconnected input is assumed to be 'zero'.

The schematic screen does not pan or zoom. If you draw the schematic and then resize the window things may get lost.

There is no 'group' selection… only one thing is selected at a time.

Module Source Editing and Syntax

The module source editor is also quite simple. Unlike the schematic editor, the source editor does provide panning. The editor is what is 'built in' to java so it is primitive. The most advanced feature is probably the ability to move an entire section of text from one place to another using cut and paste or select and drag. I've found it sufficient.

Syntax checking is performed continuously. Error conditions are highlighted in red, bold italics. A hint as to why here is an error is shown in the 'error' window at the bottom right of the screen. It isn't a very cleaver syntax machine but it gives reasonable hints.

The language used in DSPExplorer is very 'C' like but there are important modifications. First, there are no functions… each module is a self contained piece of code with well defined interfaces. Inputs to the module may be signals from other modules, compile time parameters or run time 'sliders'. Outputs may go to other modules.

As of version 3.2 there are three types of variables: 'double' which is a 64 bit, IEEE754 floating point number, 'fixed' which is a fixed point number and 'complex' which is to variables packaged together. Variables may also be single dimension arrays. Inputs and outputs may be arrays, parameters and sliders may not. Parameters and sliders are always 'double's.

Originally, DSPExplorer had only doubles and complex numbers. However, several people asked where the integers and Booleans were. To address these shortcomings, DSPExplorer version 3.2 introduced 'fixed' numbers. By default, a 'fixed' number is a 32 bit, signed integer.

Fixed and complex types of variables can have a 'precision' modifier which immediately follows the type and specifies the number of bits (including a sign bit) before and after in the form b.a. For example, many fixed point DSP processors have a 16 bit word which has a sign bit and 15 'fractional' number. They can represent any number between -1 and (almost 1) with 15 bits of precision. DSP explorer would declare such a variable as:

```
        fixed 1.15 example;
        complex 1.15 twoFixeds;
```

Generally speaking, one should always use a 'double' whenever possible because this is the native data type in DSPExplorer. NOTE: 'fixed' numbers are less efficient and doubles because DSPExplorer must verify the value of a 'fixed' each time it is updated.

Booleans are implemented in the 'C' way, 0 is false and everything else is true. The only reason for a Boolean is performance and, as with integers, it just isn't that much an improvement.

The language has a fair number of built-in functions described elsewhere.

The 'println' and 'print' functions print out strings and variables. Entire arrays can be printed using a single print statement. Printing is assumed to be informative so variable precision is truncated to only a few decimal places. Variables whose values are close to (but not exactly) zero are printed as "~0" for "approximately 0". The precision changes release to release; use sparingly.

To get a feel of the language and its use, please watch a few of the videos or examine the source of one of the tutorial playspaces.

Scope Display Controls

When the scope window is resized everything in it is resized as well. All things considered, this is probably wrong and may well be changed in the future. The reason for this was to make sure things can't get lost when the window is sized downward. Again, this may change.

Signals can be added by clicking on the 'selected signal' on the scope toolbar. Signals are removed by moving them to the waste basket. Signals are selected by clicking on their LABEL, not by clicking on the actual line. This may change.

Note that the trigger signal label is on the left hand side and all other labels are on the right. The trigger signal cannot be discarded, only replaced.

The scope can display two types of information, spectra and (time domain) signals. The spectra can be independently scaled and panned. Panning takes play by moving the <Spectra> slider at the top of the window.

Time domain signals are panned using the <Signals> slider at the bottom of the screen.

The trigger level can be adjusted using the "L>" slider on the left hand side of the screen. The trigger position relative to the 'history' buffer is moved by moving the "<T>" slider at the bottom of the screen.

The value of the selected signal can be measured using the two sliders at the left hand side of the screen. They are of the form "#>" where the "#" indicates the vertical value of the selected signal. Playing with these sliders is the easiest way to understand their use.

A quick note on Triggering. DSPExplorer is a 'discrete time' simulator which means even when the signals are displayed as 'lines' as with a normal oscilloscope, the signals are NOT continuous. This complicates triggering somewhat. To make the signals stable on the display, it was necessary to 'interpolate' the value of the trigger signal.

This is all well and good if you look at things using the 'Line' display. However, when looking at things with a 'Bar' display, the 'Bar's seem to jitter left and right AND in magnitude. This is simply the nature of the beast.

MENU ITEMS

Project>New Schematic

Create a new schematic. If you select an existing schematic you will asked if you want to override the existing schematic.

Project>Open Schematic

Open an existing schematic.

Project>Save Schematic As

Change the name of the working schematic and then save it under a new name.

Project>Change Playspace

Change the working playspace.

Module>New

Create a new module. If you choose an existing module you will be asked if you want to override the existing.

Module>Add

Add an existing module to the schematic. The module is added to the schematic and opened for examination/modification in the source screen. If the module is not from the working playspace it will be marked as read only.

Module>Clone

Copy the presently selected module into a new module. You will be asked to provide a new name. The module will be created in the working playspace.

Simulation>Run

Run the simulation (if there are no errors).

Simulation>Halt

Halt the simulation at the end of the given 'left to right' evaluation. You can 'continue' from this smoothly.

Simulation>AutoRun

Automatically start the simulation after a parameter has been changed.

Simulation>Continue

Continue a simulation after a Halt

Simulation>Single

Run a single step (complete left to right evaluation).

Simulation>Abort

Stop any running simulation. Useful when you have an infinite loop in your code. Unfortunately, there is no indication of where the simulation was when the abort happened. Room for enhancement.

Print>Schematic

Print the schematic

Print>Source

Print the contents of the module being edited.

Print>Scope

Print the contents of the scope screen.

Source>Single

Run the module being edited ONCE. This doesn't run the whole schematic… just the module being edited.

Source>Abort

Abort any simulation running. Good for infinite loops.

Source> {module.mod}

This is really just a label of which module is being edited. To edit a different module, select it using the schematic editor.

Scope>sigScale

Sets the horizontal scale of items displayed as 'signals'.

Scope>vScale

Sets the vertical scale of whatever is selected for edit. The item selected for edit is shown in bold on the trace screen and named in the 'signal' part of the 'Scope toolbar.

Scope>specScale

Sets the horizontal scale of items displayed as 'spectra' (spectrums?) Spectra are scaled to fit entire on the screen when the specScale is '1'.

Scope>Trigger>{signal}

The trigger menu item selects the operating mode for the 'Scope. This includes:

- Scope>Trigger>{signal} Selects the trigger signal. The value shown is the 'present' value.

- Scope>History(value): how many time samples are recorded for display. The present value is shown. This is the same number used to compute the spectrum when the signal is displayed as a spectrum. Larger values will result in more precision.
- Scope>Window(value): which windowing function will be used when computing the spectrum of signals. The present value is shown. 'none' delivers the fewest surprises when looking at signals in the frequency domain. Use of other windows can cause 'artifacts' to appear on 'pure' signals. Be on guard.
- Scope>Auto@(value): how long should the simulation run before automatically triggering a display. The present value is shown.
- Edge: set the +- edge and AC/DC coupling. Use DC coupling whenever possible. The present value is shown.

Scope>[Real Imag Complex]

For the selected signal, control what information is shown. Real is just the real part, Imag is the imaginary part, Complex is both. The Real part is always shown as a solid line, the imaginary part is always dashed.

Scope>[Line Bar Line&Bar]

For the selected signal, display the signal as line(s) or as vertical Bar(s) or as both. For the bar case, if the horizontal scale is such that the bars would merge, the 'line' format is used instead regardless of this setting.

Scope>[Sig Ref asSpec FFT IFFT Mag PeakMag]

How to display the selected signal.

- Sig: display as a normal, 'time domain' oscilloscope trace.
- Ref: as a normal 'time domain' trace. This is usually used to display arrays such as filter kernels.
- asSpec: as a spectrum. This is used to simply array as a spectrum rather than as a time domain signal.
- FFT: display the FFT of the selected signal. Used to examine the spectrum of a signal.
- IFFT: display the Inverse FFT of the selected signal. If displayed as 'Complex', the real and imaginary parts are both shown.

- Mag: display the spectrum MAGNITUDE of the selected signal. If being displayed as 'Complex', the imaginary part is the angle.
- PeakMag: keep the 'peaks' of the Magnitudes of the spectrum of the selected signal. Useful when examining the passband of filters and the like.
-

Scope>[Normal Center db Centerdb]

- Normal: display the selected trace normally.
- Center: display the selected trace (usually a spectrum) with 0 in the center rather than at the left.
- db: display the 20log10(signal)
- Center db: display the 20log10(signal) centered (usually for a spectrum)

Scope>{signal}

Specifies the signal presently selected. Clicking on this allows one to add another signal. Clicking on a signal label will change the value of this field. The selected signal is displayed with a heavy line.

Scope>Color

Change the color of the selected signal.