February 2005.


http://www.axoris.be
mailto:info@axoris.be

# Axoris Tool suite

# Tutorial

# Table of Content

## Table of Figures



# 1. Introduction

This document is a tutorial for the Axoris Tool suite including VirtuAL3101 and the Miss Parker Manager.

The document is split in three very distinct parts:

- User view: tutorial for who wants to play with “Miss Parker” and with the existing projects.
- Designer view: tutorial for who wants to assemble blocks in AgALag and use them with “Miss Parker”
- Developer view: tutorial for who wants to write self-made assembly code, debug it, …

Most of the explanations given about the Miss Parker manager are applicable whether the device controlled is a Miss Parker or not (AL3101 development board, self-made board…). In the rest of the text, such a device will be called “Miss Parker”.

Please note that this document is not meant to replace the help files of each individual tool. The aim of this document is to show a possible way of using the tools. You should still refer to individual helps for more information on the tools.

# 2. User tutorial

## 2.1. Introduction

As described above, a user is someone who wants to use a Miss Parker and play with available projects.

In other words, if you want to use your Miss Parker as an “out-of-the-box” piece of equipment you could buy from a shop, this part is for you…

## 2.2. Tools needed

What is basically needed in this case is the Miss Parker manager tool. This tool is available under the “Miss Parker \ Download page” of the Axoris Website (http://www.axoris.be/MissParkerDownload.htm).

You must also download the Phaser project files from the “Effect resources page” of the Axoris Website (http://www.axoris.be/Effect_Resources.htm).

## 2.3. Getting started

The Miss Parker manager binary is located in “install-dir\bin” where “install-dir” is the directory where Miss Parker manager has been installed.

There’s no fixed directory for the project files. As a consequence, projects can be stored anywhere. Two obvious options are possible:

- Use a “projects” directory in “install-dir” directory.
- Use the “projects” directory of the “VirtuAL3101” directory.

## 2.4. Basic functionality

### Launch Miss Parker manager

You should get the following screen (the look & feel may change with the theme you are using):

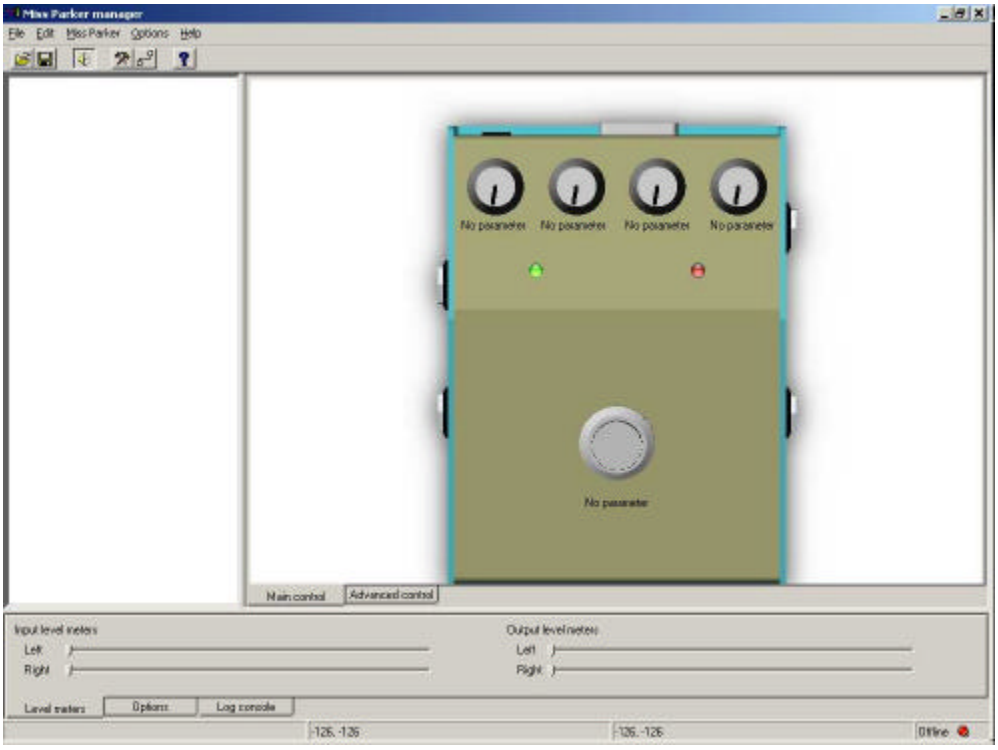



**Figure 1 – Miss Parker manager default screen**

### *Main panel*

There are two tabs in the main panel:

- Main control: is showing a Miss Parker and its controls.
- Advanced control: is giving access to all of the parameters of the current project.

The main panel and its two tabs is therefore the place to play.

### *Left panel*

The left panel is the place where the parameters of the projects will be listed.
The parameters can be grouped together. This allows to have clean and easy control of complex applications.

### *Bottom panel*

There are three tabs in the bottom panel:

- Level meters: providing information about the input and output levels as measured by the AL3101.
- Options: allowing to set options of the Miss Parker.
- Log console: console window in which some messages are printed.

### *Status bar*

The status bar displays various pieces of information (from left to right):

- Information about menu currently pointed or “Trying to connect” message
- Input levels in decibels. This is actually a textual version of the level meters window.
- Output levels in decibels.
- Hardware connection status. It is “Offline” with a red LED if no known device is detected or it is “Online” with a green LED if a known device is detected.

### *Program menus*

The “File” menu is a quite usual one with file open, save and the plain exit function.
The “Edit” menu allows to deselect parameters from potentiometers and also to clean the log console.
The “Miss Parker” menu allows all Miss Parker specific operations.
The “Options” menu allows changing the theme, communication library and output state.
The “Help” menu is of course about help but is still mostly empty at the time of writing this tutorial.

## 2.5. Practical example

### Loading the project

Let's use the Phaser project as an example. First load the project using the "File open" menu or the toolbar button. You should see the following screen:

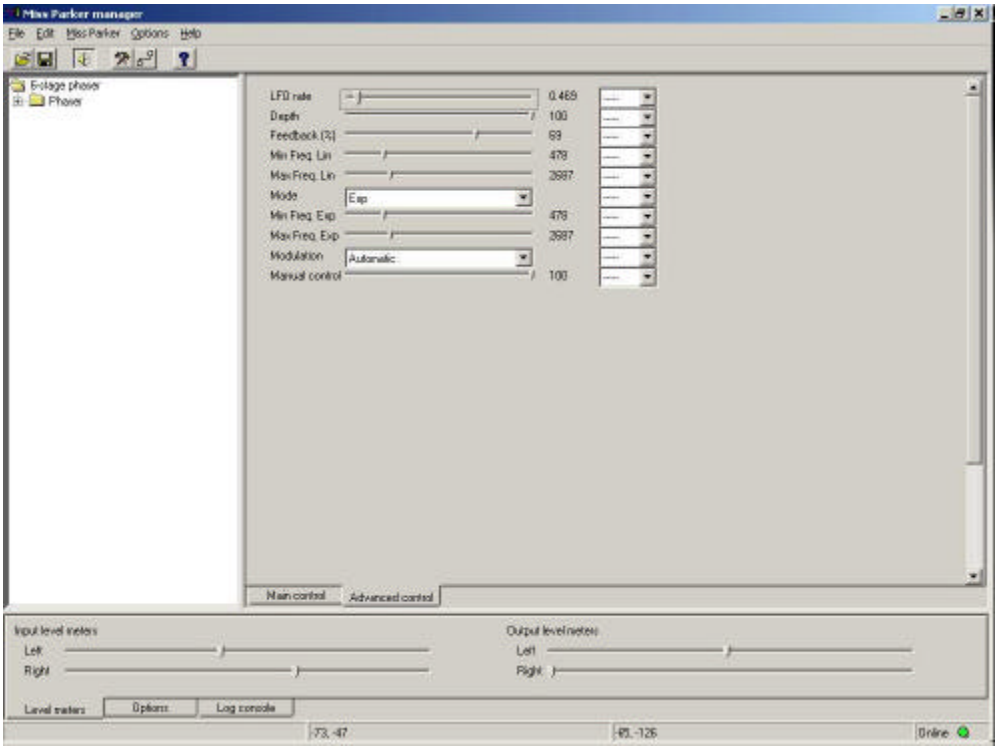



**Figure 2 – Phaser project screenshot**

**WARNING**: all actions done when a device is not yet connected like loading a project, assigning parameters, … must all be re-done as soon as a device gets connected. As a consequence, it is highly advised to work online from the start if it is the intention to work with hardware.

Note that the Right output level meter does not move because the phaser you've loaded is a mono effect only working on the left channel. It is therefore normal that the right output remains a –126 dB that corresponds to "no output".

### About Miss Parker

When looking at the main window, one can see a representation of the Miss Parker hardware.

There are four potentiometers that can be linked to the parameters of the projects.
There's the mode LED (on the left) showing if the effect is enabled or disabled. To toggle between the two states, the foot switch should be used. There's also the peak LED (on the right) showing if the peak value as defined by the user in the Options window has been reached.

The potentiometers on the screen will show the actual position of the potentiometers on the hardware.
The LEDs will also show the same behaviour on the screen as on the hardware.

### Assigning parameters to potentiometers

This can be done in two ways:

1. With the "Main" tab, drag & drop of parameters from the left tree to the potentiometers or foot switch.
2. With the "Advanced control" tab, select potentiometer or foot switch from the combo boxes.

Assigning parameters choosing any of the method will adapt the text below the potentiometer or foot switch in the "Main" tab and adapt the combo box of the parameter assigned in the "Advanced control" tab.

For example, let's assign:

- The "LFO rate" to the 1st potentiometer.
- The "Feedback" to the 2nd potentiometer.
- The "Min Freq. Exp" to the 3rd potentiometer.
- The "Max Freq. Exp" to the 4th potentiometer.
- The "Manual control" to the Expression pedal (can only be done from Advanced Control tab).
- The "Depth" to the foot switch.

Depending on the parameter, it may take more time to assign it (like the LFO rate for example) because they are implemented using tables (see 3.5 Miss Parker manager projects for details about tables).

The main tab and the advanced control tabs should look as follows now:

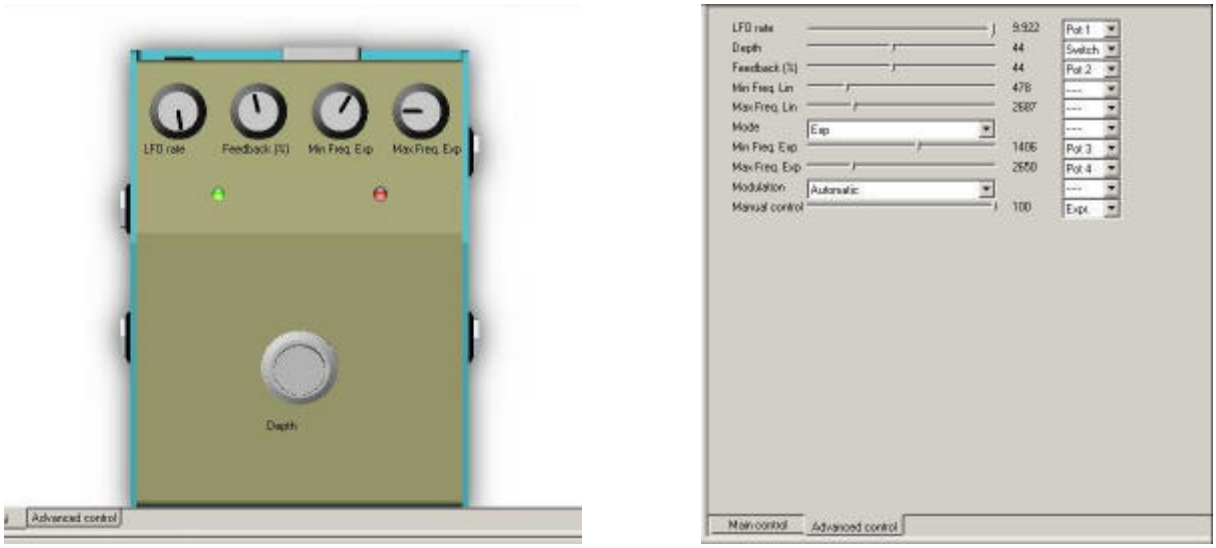



**Figure 3 - Phaser with parameters assigned**

To switch between an automatic modulation based on the LFO and the expression pedal, one should change the "Modulation" parameter from "Automatic" to "Manual".

Once parameters are assigned to hardware controls, it is not possible anymore to control them from the PC interface. To get again access from the PC to these parameters, you must use either the "Edit / Potentiometer disable" menu or choose "----" in the combo box next to the parameter in the "Advanced control" view. You can also assign another parameter to the given hardware control of course.

## *Adapting the audio levels*

The Miss Parker is equipped with amplifiers both on the input side and on the output side. With these amplifiers, the Miss Parker is able to cope with very different audio levels from a microphone level of a guitar to a line level of a synthesizer.

The amplification amount is set in the "Options" tab of the bottom panel:

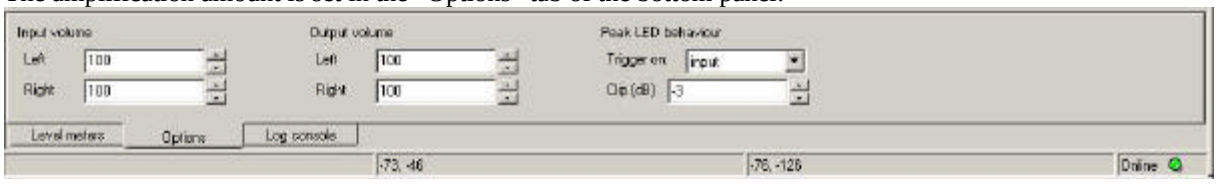



**Figure 4 - Options window**

A value of "0" means that there's no amplification while a value of "100" means that the amplification is at maximum. It is possible to set independently each channel on both input and output side.

It is also possible to specify when the peak LED should start lighting. It is possible to specify whether the LED should be triggered on the input or on the output.

## 2.6. Miss Parker stand-alone operation

Once you've played with the project and you're happy with the values of some of the parameters and the assignment to hardware controls of others, it is time to prepare the Miss Parker for stand-alone operation.

Indeed, all actions done in the previous chapter have not overwritten the internal memory of the Miss Parker. As a consequence, once the power will be removed and put back again, the Miss Parker will load its previously stored project, parameters, hardware assignments and levels. Let's know look how to store everything in the Miss Parker so that your new settings will be used later on.

First, let's store the project in Miss Parker. This is done using the "Miss Parker / Store project" menu. This procedure may take some time, be patient during the transfer. At that point, the program, parameters and assignment of parameters on hardware controls are stored in the Miss Parker.

Finally, let's store the level of the audio amplifiers. This is done using the "Miss Parker / Store Options" menu. If you would forget to do it, the program should automatically warn you at the moment you quit the software.

## 2.7. Enhanced features

### Support for other devices

Other devices than the Miss Parker can be supported. The "Options / Communication" menu should be used to select which hardware device to use.


**Figure 5 - Communication window**

There are currently two hardware devices supported: the Miss Parker of course and the AL3101 development board. Important things to know are:

- For these two devices, once selected, the program will automatically detect if such a device is connected to the computer or not.
- The program will re-scan if a device has been connected or not at regular intervals; everything works therefore plug & play. The interval is device dependent (every 2 seconds for Miss Parker and every 10 seconds for the AL3101 development board).
- If the device is not a Miss Parker (Miss Parker support field), some of the graphical elements of the program will be hidden (see "Miss Parker specific items" for more details).

When selecting the "Advanced" view, it is possible to change every single field of the two control words of the DSP. As its name suggests it, it is an advanced feature that should only be used by advanced users.

Support for more devices can be added by means of more communication libraries. Making of these libraries is out of the scope of this document, please contact the Axoris team for more information on the subject.

### *Upgrading the firmware*

It is possible to upgrade the firmware of the Miss Parker with new versions from the Website for example. This procedure is equivalent to upgrading the BIOS of a PC.

**WARNING**: The procedure may take some time and should never be interrupted.

### *Miss Parker specific items*

If the device connected is not a Miss Parker, some items will disappear:

- The “Main” tab view depicting a Miss Parker
- The combo boxes in “Advanced” tab view allowing to attach a parameter to a potentiometer.
- The “Options” tab view which allows setting Miss Parker specific options
- The “Miss Parker” menu that basically groups features specific to the Miss Parker.
- The “Potentiometer disable” item of the “Edit” menu.

The graphics are re-activated as soon as a Miss Parker device is again selected as target device.

## *2.8. Tips & Tricks*

### *About memory maps*

The Miss Parker manager is able to cope with the memory map files as generated by VirtuAL3101.

If the memory is initialized by a memory map or by initialization code is completely invisible for the end user. Only the time it takes to download the project in the Miss Parker will differ.

# 3. Designer tutorial

## 3.1. Introduction

As described above, a designer is someone who wants to build AL3101 projects based on existing blocks.

In other words, if you want to play with the Axoris application generator AgALag, this part is for you…

## 3.2. Tools needed

In order to build applications, the VirtuAL3101 suite is needed. These tools are available under the "VirtuAL3101 / Download page" of the Axoris Website (http://www.axoris.be/VirtuAL3101-Download.htm).

The VirtuAL3101 tool suite is made out of several tools:

- AgALag application generator made of a GUI front-end "AgALag" and of a command-line build tool "mkalproj".
- Assembler "asm" and Disassembler "disasm"
- Command-line simulator "alksim"

While the simulator and disassembler won't be used, the assembler and mkalproj tools will be used through the use of AgALag. Actually, the only tool to be used by the designer is AgALag.

## 3.3. Getting started

The AgALag binary is located in "install-dir\bin" where "install-dir" is the directory where VirtuAL3101 has been installed.

Two important directories in "install-dir" are:

- "Effects": this is where the building blocks are stored
- "Projects": this is where your projects are / will be stored.

### Launch AgALag

You should get the following screen (the look & feel may change with the theme you are using):

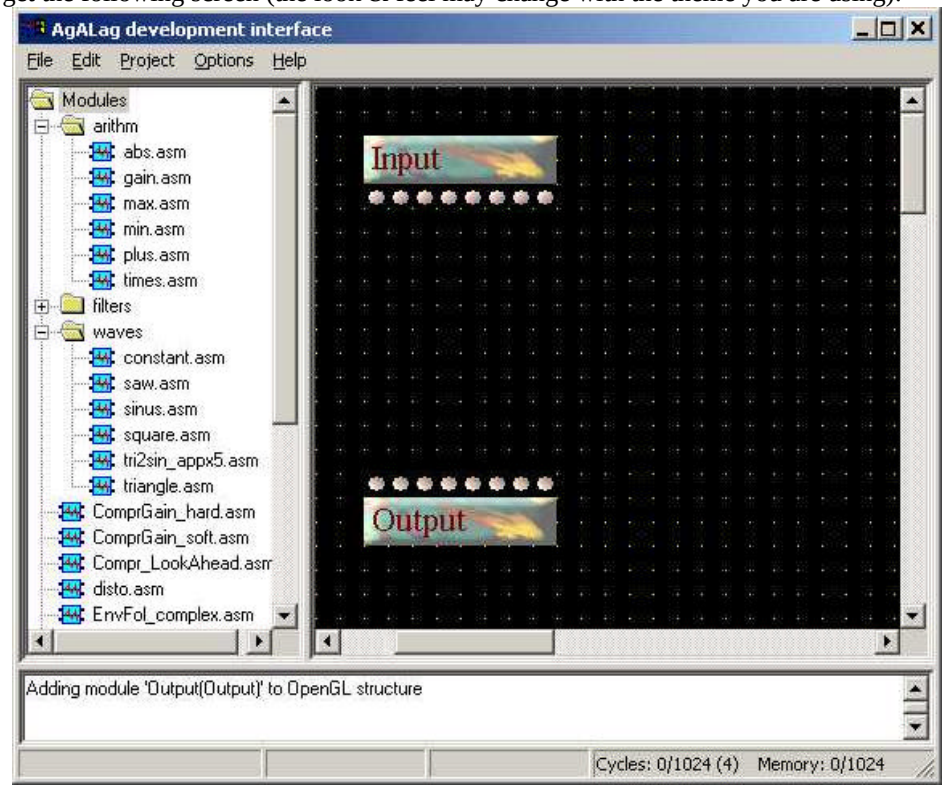



**Figure 6 - AgALag default screen**

The tree on the left side shows what building blocks are available from the “Effects” directory. Each of the items in that list is a piece of assembly code that will be used to build more complex applications. You can double-click on those items to get information about the block.

The window on the bottom is providing some information about what AgALag is doing while the status bar provides information about the amount of cycles and memory left in the AL3101.

The Input and Output blocks represent the 8 inputs and 8 outputs of the AL3101. You have to know what IOs are actually used in your hardware to know which IOs to use here. For example, Miss Parker and the AL3101 development board use the two leftmost inputs and outputs.

## *Program menus*

The “File” menu is a quite usual one with file new, open, save, print and the plain exit function.
The “Edit” menu only allows cleaning the log console.
The “Project” menu will be quite described in the next pages.
The “Options” menu allows changing the theme and the directories.
The “Help” menu is of course about help but is still empty at the time of writing this tutorial.

Before starting to use AgALag, one may check that the directories are set correctly:

- Effects should point to “../effects”
- Assembler should point to “.” or  to “../bin”
- Projects should point to “../projects”
- Skins should point to “install-dir/resources/AgALag/themes”

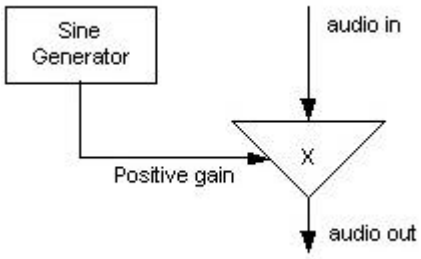
### 3.4. Practical example

For the sake of the example, we will build a Tremolo feature. This is done in phases:

1. Drop the building blocks needed to perform the task.
2. Link the blocks together
3. Adapt the parameters of each block

## Principle of the Tremolo

A tremolo is done the following way:




A sinus is generated with an output between two positive numbers.

The output of the sine generator is then used as a gain on the audio signal.

The minimum and maximum values of the sine generator define the strength of the effect.

## Add the building blocks

You should follow these next steps:

- Go in “waves” and drag & drop the “sinus” and “constant” blocks on the main window
- Go in “arithm” and drag & drop the “times”, “plus” and “gain” blocks on the main window

Your screen should look like something like this:

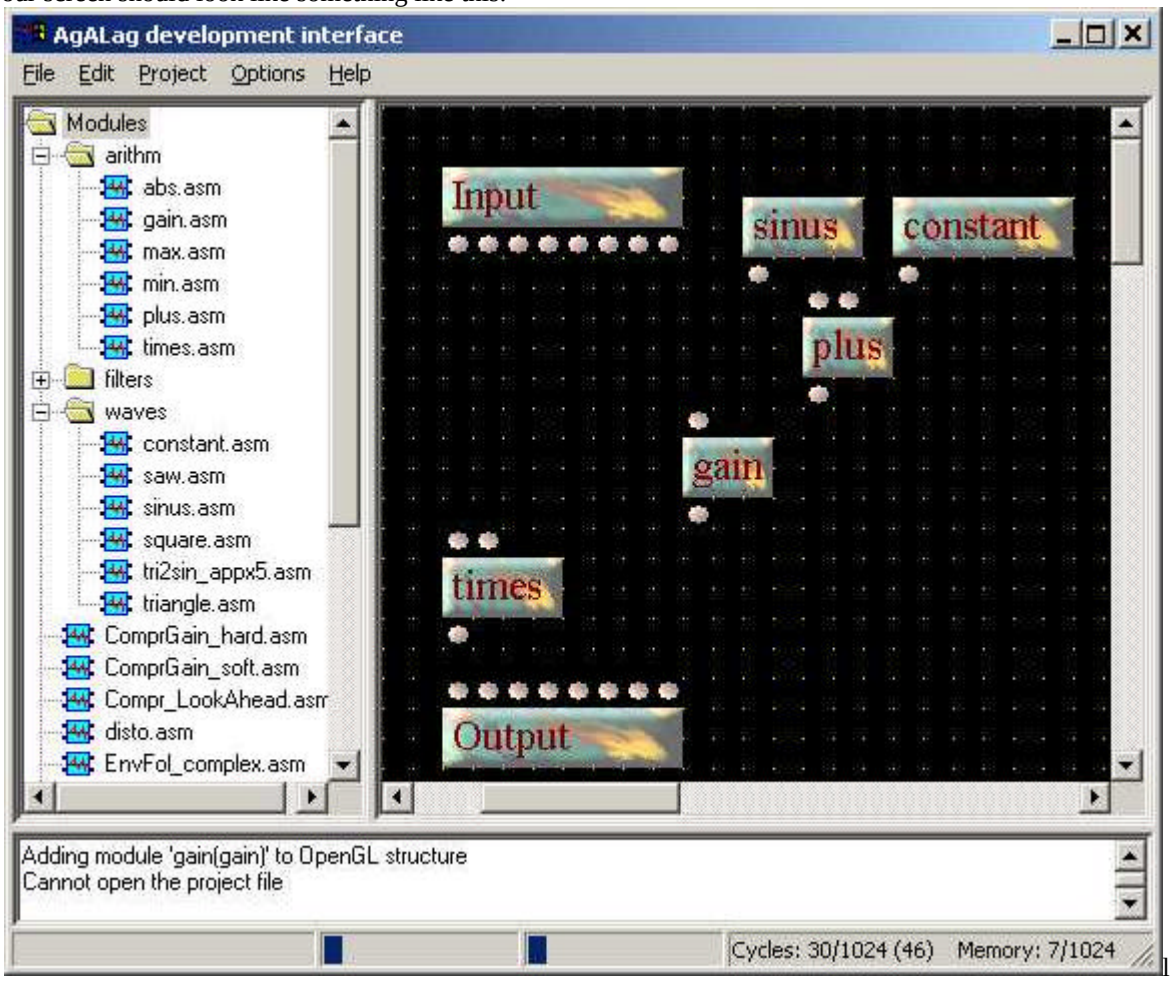



**Figure 7 - Screenshot of the Tremolo blocks**

You can get information on the various blocks by double-clicking on any leaf of the tree on the left.

### *Save the project*

You can save the project at this stage. This is simply done by going into the “File \ Save” menu.

A standard “Save file” window is then asking you where to store the project and how to name it.

Let’s call our project “Tremolo” and save it into the standard “Projects” directory.

### *Add connection links*

The blocks should then be connected. This is done by clicking on one pad and clicking on a second one; the order does not matter, AgALag will not which direction makes sense.

The following links should be added:

- From the “sinus” output to one input of the “plus” block.
- From the “constant” output to the other input of the “plus” block.
- From the “plus” output to the input of the “gain” block.
- From the “gain” output to one input of the “times” block.
- From the leftmost input pad of the “Input” block to the other input of the “times” block.
- From the “times” output to the leftmost output pad of the “Output” block.

The screen should then look like that:

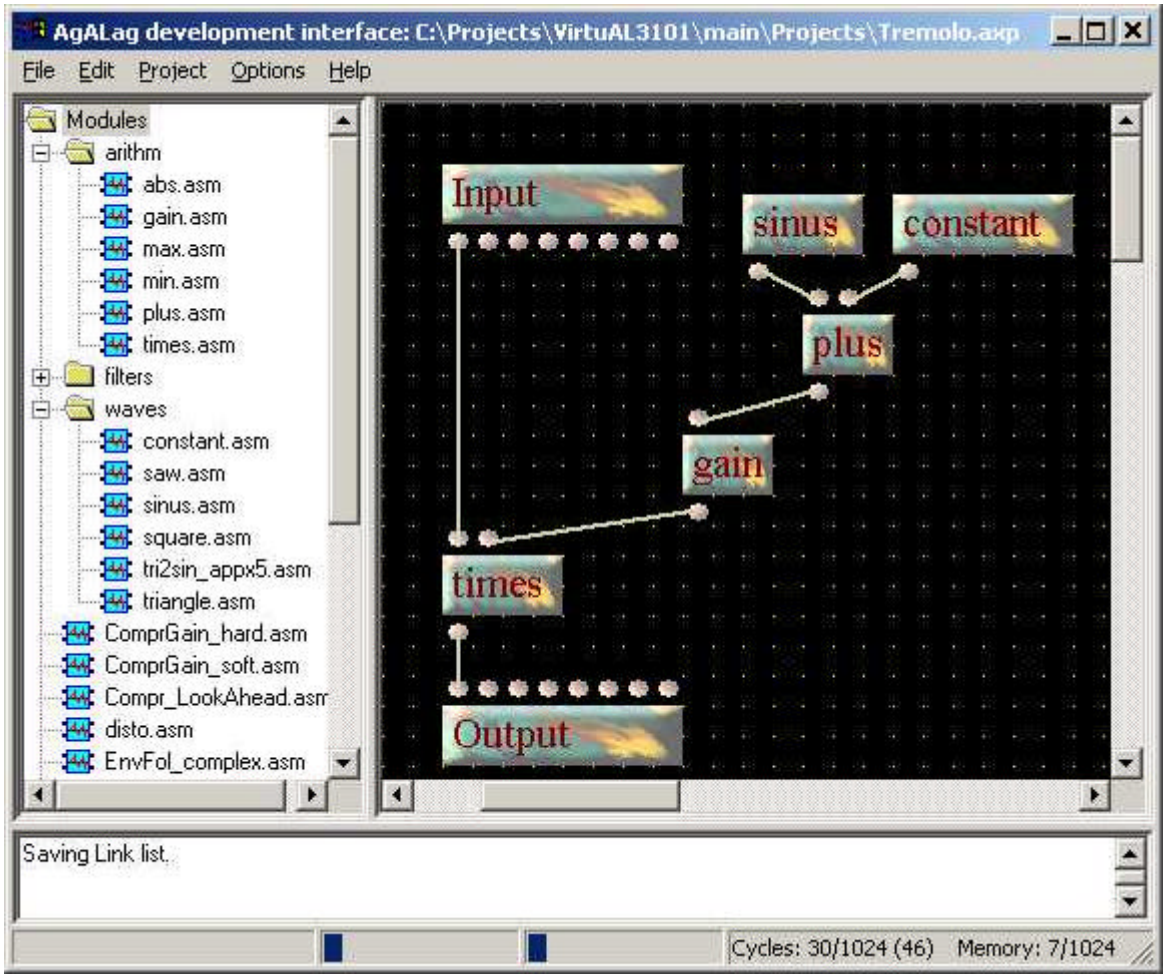



**Figure 8 – Screenshot of the Tremolo links**

One can see that our simple example requires 30 cycles out of 1024 available on the chip. With the init section, 46 cycles are required (more details about init section in “About memory maps vs. initialization code”). Finally, 7 data RAM locations are needed.

The project is now ready to be built…

### *Building the project*

The project related commands can be found in the “Projects” directory:

- Make – builds the .asm file of the application based on the project
- Assemble – assembles the .asm file into a .obj file.
- Make & Assemble – builds and assembles the project in one shot.
- Settings – allows the user to specify the settings of the project.
- Options – allows to set options for making & assembling.

The “Settings” window looks as follows (when filled-in):

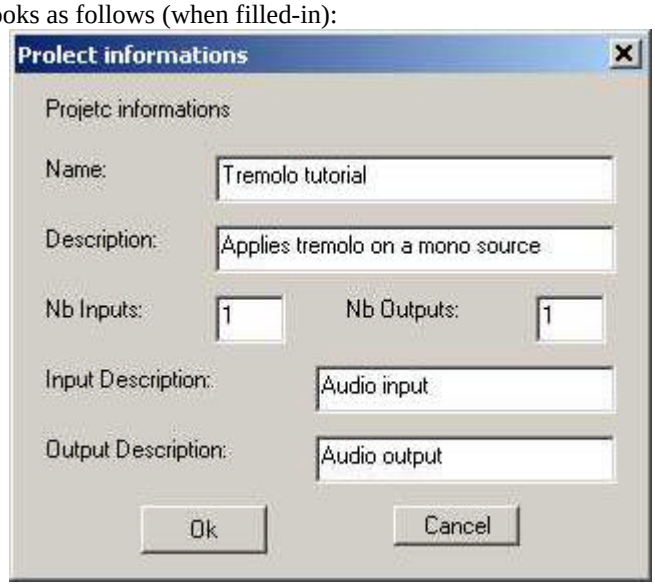



**Figure 9 - Settings window**

The fields in this window will be used to fill-in the header of the assembly file generated. The header structure is defined in [CODING RULES].
It is useful to fill-in those fields as the code generated is AgALag compliant i.e. the assembly code generated can be re-used directly as a building block (when .asm is copied from the Projects to the Effects directory).

The “Options” window looks as follows:

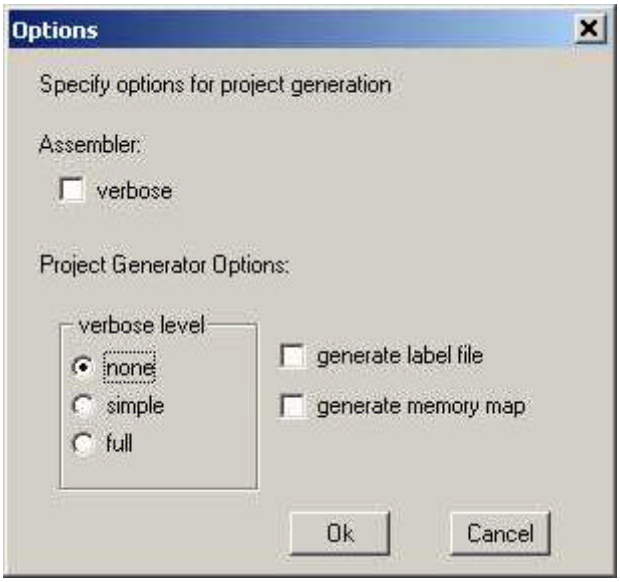



**Figure 10 - Options window**

The verbose flags can be used to ask the assembler and mkalproj for some detailed information. By default, these flags can be set to none as it wouldn’t bring significant and relevant information for Designers.

A memory map file (replacing the initialization section) can be generated as well as a label file. See [CODING RULES] for more details.

We're now fully ready to make the project. This can be done using the "Make" menu of course.
Once it is done, the log console should contain the following information:

```
7 modules
6 links to read.
Scanning sections in source files.
Determination of memory offsets
Creating temporary IO
Creating assembler code
Generating Offset table
Success.
Building MissParker project
```

You can now browse the "Projects" directory and you can find (at least) the following files:

- Tremolo.axp: this is the AgALag project file.
- Tremolo.asm: the freshly generated assembly file.
- Tremolo.prj: the Miss Parker manager project file (see 3.5 Miss Parker manager projects).
- Tremolo.off: this is an offset file generated but used "internally" by the tools.

You can open the assembly file "Tremolo.asm" to take a look at it if you wish.

## *Adapt the parameters*

You can start by renaming the modules if you like to give them more human names. Let's do that once for the sake of the example.

If you right-click on a module, the following pop-up menu appears:

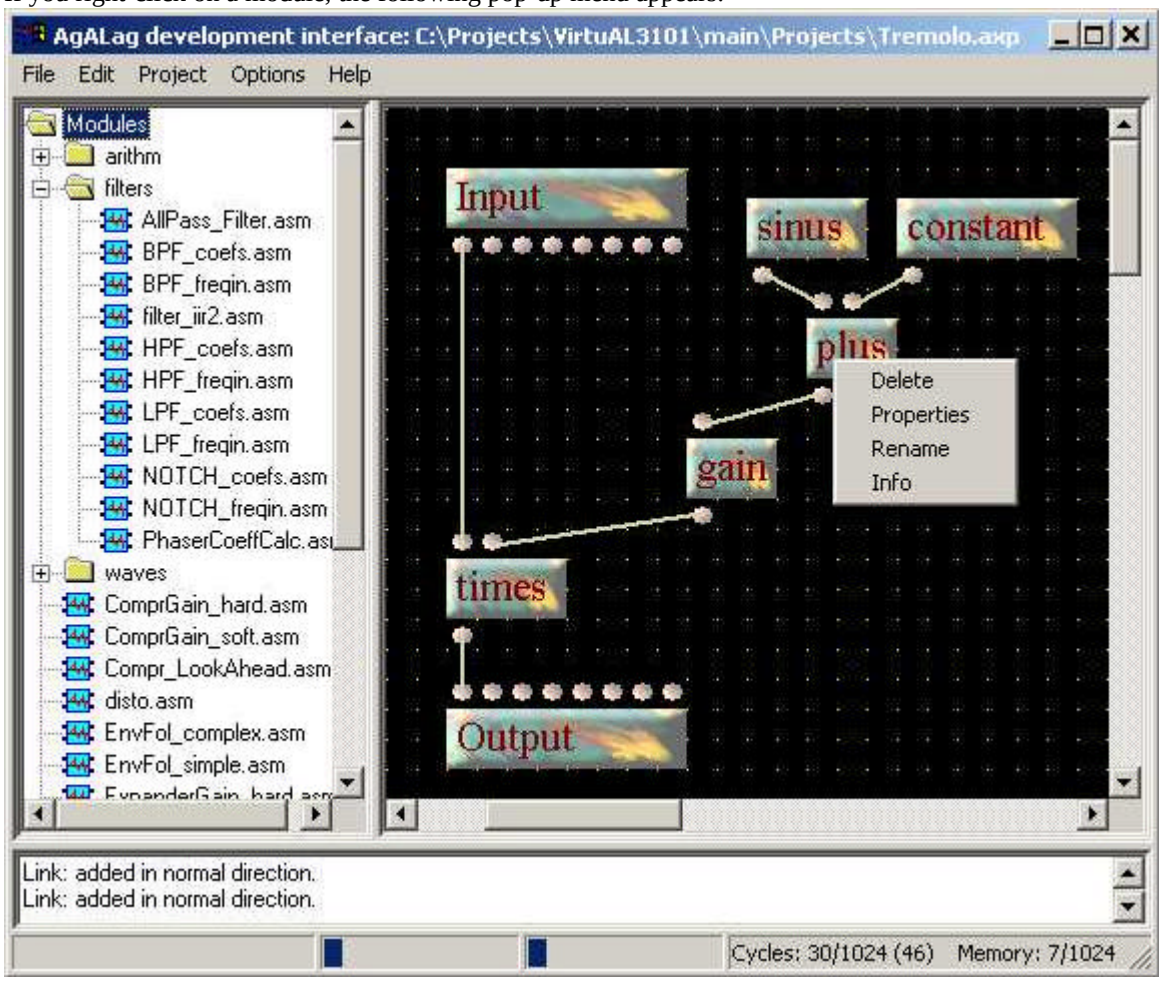



**Figure 11 - Module pop-up menu**

Rename the "constant" block into "offset block" as depicted above.
Choosing "info" is another way of getting the same information as when double-clicking on the left tree items.

The parameters can now be changed using the same pop-up menu and by choosing “Properties” this time. You should enter the following parameters:

- Sinus – Ampl – 0.5
- Sinus – Delta – 0x2BB. This parameter is given by (4 * Sine frequency / Sample rate) so a sine at 0.5Hz will lead to a delta of 4 * 0.5 / 48000. In S3.24 format (so multiplied by $2^{24}$), it leads to 0x2BB.
- Offset – cst – 0.5
- Gain – gain – 1.5

You **must** perform a “make” again so that the changes are made in the assembly file.

## *Simulate*

It is possible to simulate a project using the “Project / Simulate” menu.

The following window will appear asking you for an input and output file. Please note that the input file **must** be a stereo file.

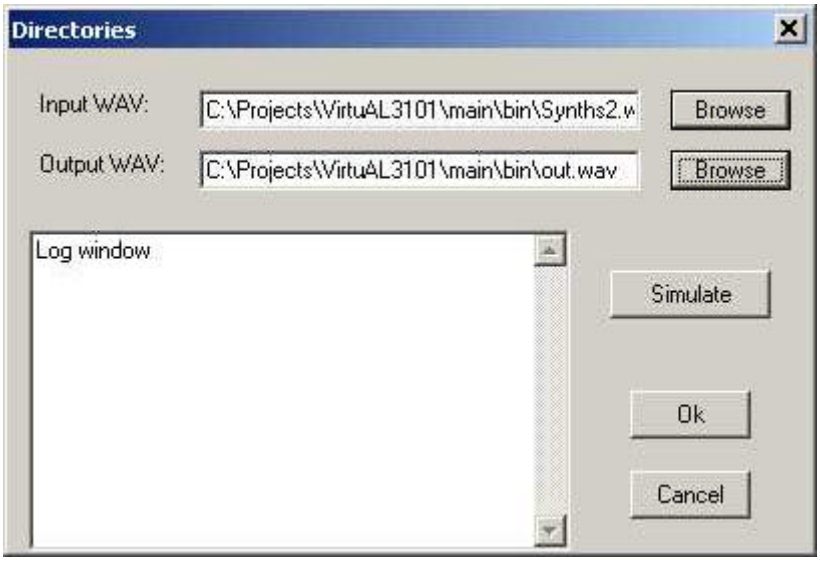



**Figure 12 - Simulation window**

The log window should contain the following text after simulation is done.

```
Simulation running.
1KSM>
Simulation complete.
1KSM>
1KSM> exit

User exited within script.

Exiting simulator.
Simulation done.
```

The output file chosen should also contain a “tremoloized” version of the input. The settings chosen are quite extreme ones but at least the effect should be quite obvious.

## 3.5. Miss Parker manager projects

Let's create now a Miss Parker manager project from the Tremolo example. Please keep in mind that the tool can be used with Miss Parker as well as for the AL3101 development board. Much other hardware could be handled by means of a dynamic library.

Open the Miss Parker manager project dialog box with the "Project / Miss Parker" menu.
Then you should perform the following actions:

- Select in the three the Gain parameter of the Gain block.
- Select the "Enable parameter" checkbox in the bottom left of the window.
- Enter "Amount of effect" as description
- Leave the Computation Type and Progression Type untouched
- Enter 50 as the Value for Display. This value will be used as the default one.
- Enter 0 and 100 as min and max for the Display values.
- Enter 0.0 and 2.0 as min and max for the Internal values. These values will be used as extremes on the DSP. Actually, the 0 to 100 display range will be converted to the 0.0 to 2.0 DSP range.
- Change the display type to "int" as we've used a range from 0 to 100 to represent the parameter.
- **Never forget** to click on "Store" when the modifications are done.

The window should now look like that:

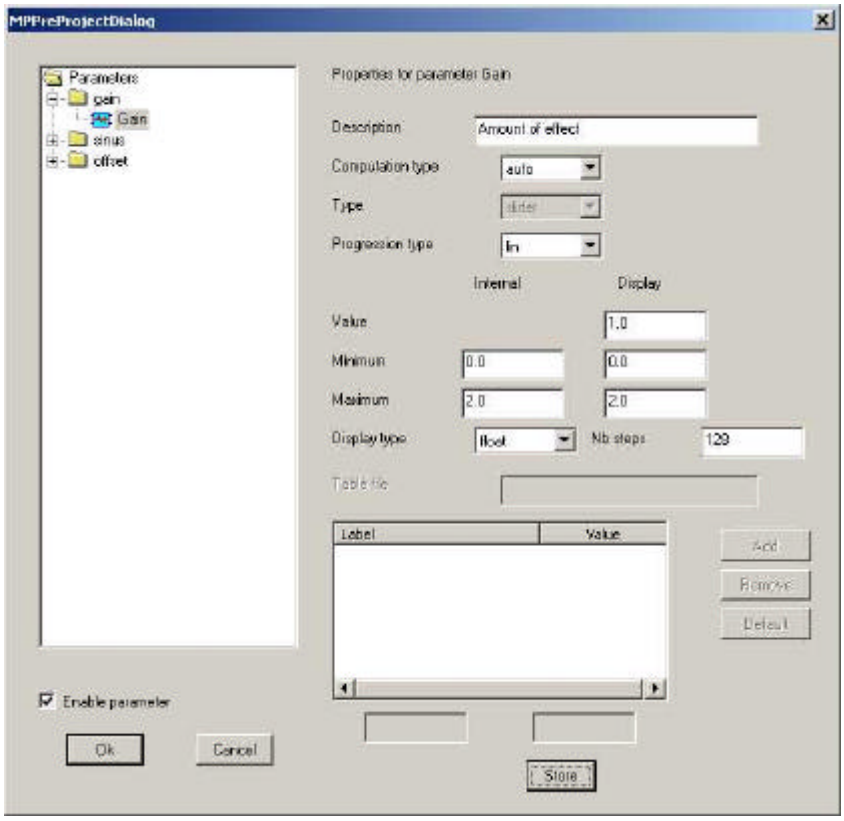



**Figure 13 - Parameter setting for Miss Parker manager project**

The progression type can be set to "lin", "log" and "exp" depending on how the range should be covered. If "log" is chosen for example, the range from min to max will be covered in a logarithmic way.

Now perform the following actions:

- Select the Cst parameter of the Offset block
- Select the "Enable parameter" checkbox in the bottom left of the window.
- Enter "Mid reference amplitude" as description
- Leave the Computation Type and Progression Type untouched
- Enter 0.5 as the Value for Display. This value will be used as the default one.
- Enter 0.0 and 1.0 as min and max for the Display values.
- Enter 0.0 and 1.0 as min and max for the Internal values (Internal and Display ranges can match).
- Leave the display type to "float" as we've used a range from 0.0 to 1.0 to represent the parameter.
- Save modification by clicking on "Store"

Then:

- Select the Ampl parameter of the sinus block
- Select the "Enable parameter" checkbox in the bottom left of the window.
- Enter "Moving amplitude" as description
- Leave the Computation Type and Progression Type untouched
- Enter 0.5 as the Value for Display. This value will be used as the default one.
- Enter 0.0 and 1.0 as min and max for the Display values.
- Enter 0.0 and 1.0 as min and max for the Internal values.
- Leave the display type to "float".
- Save modification by clicking on "Store"

Finally perform the next actions:

- Select the Ampl parameter of the sinus block
- Select the "Enable parameter" checkbox in the bottom left of the window.
- Enter "LFO Frequency" as description
- Change the Computation Type to "table"
- Leave the Type to "Slider". The "Combo" type can be used to make combo boxes like "on / off".
- Leave the progression type untouched.
- Enter 0.5, 0.0 and 10.0 as the Value, Min and Max for Display.
- Enter "SineFreqTable.txt" as the name of the table of Sine frequencies. The "root" directory for tables is the "Projects" directory (as chosen in Program menus).
- Save modification by clicking on "Store"

The screen should look like follows:

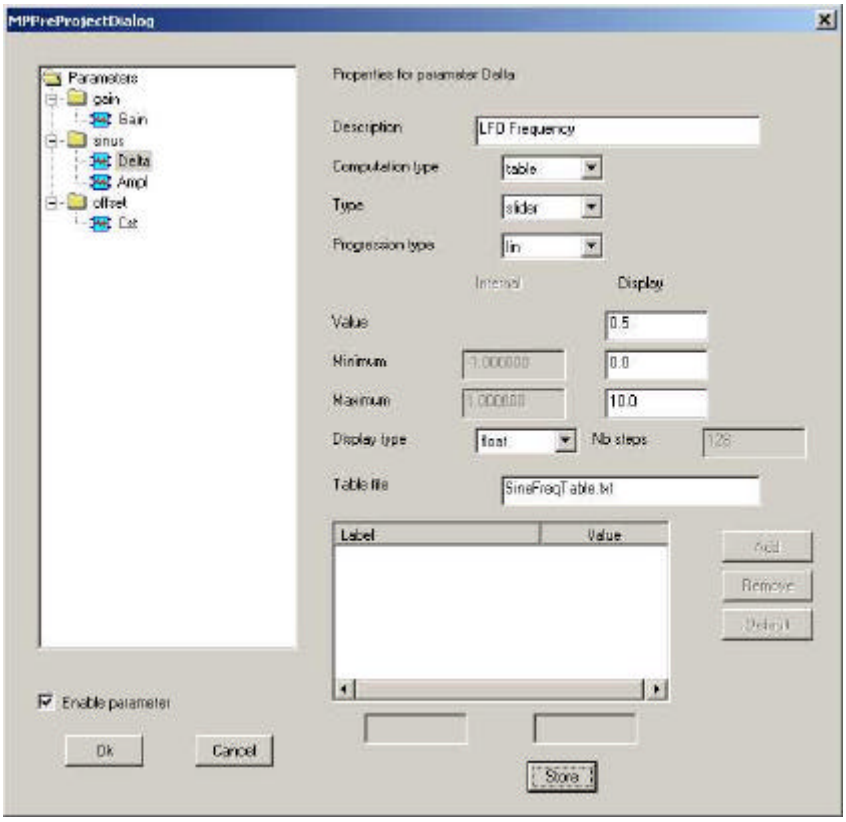



**Figure 14 - Parameter setting with an external table**

There are two main reasons to use tables:

1. The behavior is not "linear" and cannot be mimicked by mapping a range on another.
2. The values used as Internal values are too small to allow a good interpolation of the values. In the future, the Miss Parker manager will detect possible problems automatically and will then also generate the tables. For the moment, if the difference between Max and Min is smaller than $10^{-3}$, a table should be used.

The LFO frequency is an example of the second case as 10Hz is represented on the DSP (using sinus.asm) by $8.333*10^{-3}$.

The tables are a succession of ASCII string giving each of the 128 steps as an integer number. The integer number is given by float_number * $2^{24}$.
The table therefore looks like follows:

```
0
110
220
330
…
```

Please note that the table **must** have a size of 128 values, no more, no less. This is actually the number of steps that is stored in the Miss Parker eeproms therefore the current limitation.

**WARNING:** the tables must be located in the directory where the .prj is stored or any other but not in the "Tremolo" directory. Indeed, this directory gets destroyed during each build process so the table file would be destroyed as well. As said above, when specifying the file name, the program assumes that the current directory is the "Project" directory.

Finally, you can close the Miss Parker manager project dialog box.

The last action to perform is to "make" the project so that the Miss Parker Manager project is created.
The "tremolo.prj" file now contains links to files to find in the "Tremolo" directory. These files are the descriptions of each parameter as you have entered them during the reading of the previous pages.

# 4. Before opening with the Miss Parker manager tool, you must have created the SineFreqTable.txt file in the Tremolo directory. A copy of that table is available from 6.Complete project from scratch

Part to be written soon...

Last but not least, the newly created Miss Parker manager project can now be opened with the Miss Parker manager tool. For a tutorial on that part, please refer to 2.User tutorial.

## 4.1. Tips & Tricks

### About memory maps vs. initialization code

AgALag compliant codes are controlled using parameters stored in Data memory. There are two ways to initialise those parameters at start-up:

1. Initialise them from the program memory using initialisation sections.
2. Initialise them from a micro-controller by downloading the correct values in Data memory.

While the first is very portable as the DSP has everything it needs to get started, the second has a much reduced cycle count and relies on particular micro-controller features.

The ALksim simulator and the Miss Parker manager are able to cope with both methods i.e. they are able to load memory map files that are snapshots of what the memory should be at start-up.

The AgALag can generate the assembly codes in two ways following the two different methods. The way it is done is defined by the “Generate memory map” checkbox in the Options windows (see Figure 10 - Options window).

Both have pros & cons and one should choose based on cycle constrains, hardware capabilities, download tool features what is the best appropriate method to use.

### *How to implement a feedback loop?*

AgALag does not allow feedback loop (yet, the feature is on the "to-do list"). However, there's a simple work-around for that.

The aim is to use the "Feedback_part1" and "Feedback_part2" modules:

- The part1 is meant to mix the input signal with a state variable. A gain called Feedback is applied on the state variable before the addition.
- The part2 is basically copying its input into another state variable.

The trick is to modify the generated assembly file so that the state variables of both parts have the same address in memory. That will ensure that what is copied by the part2 gets re-injected by that part1.

Let's build the following project:

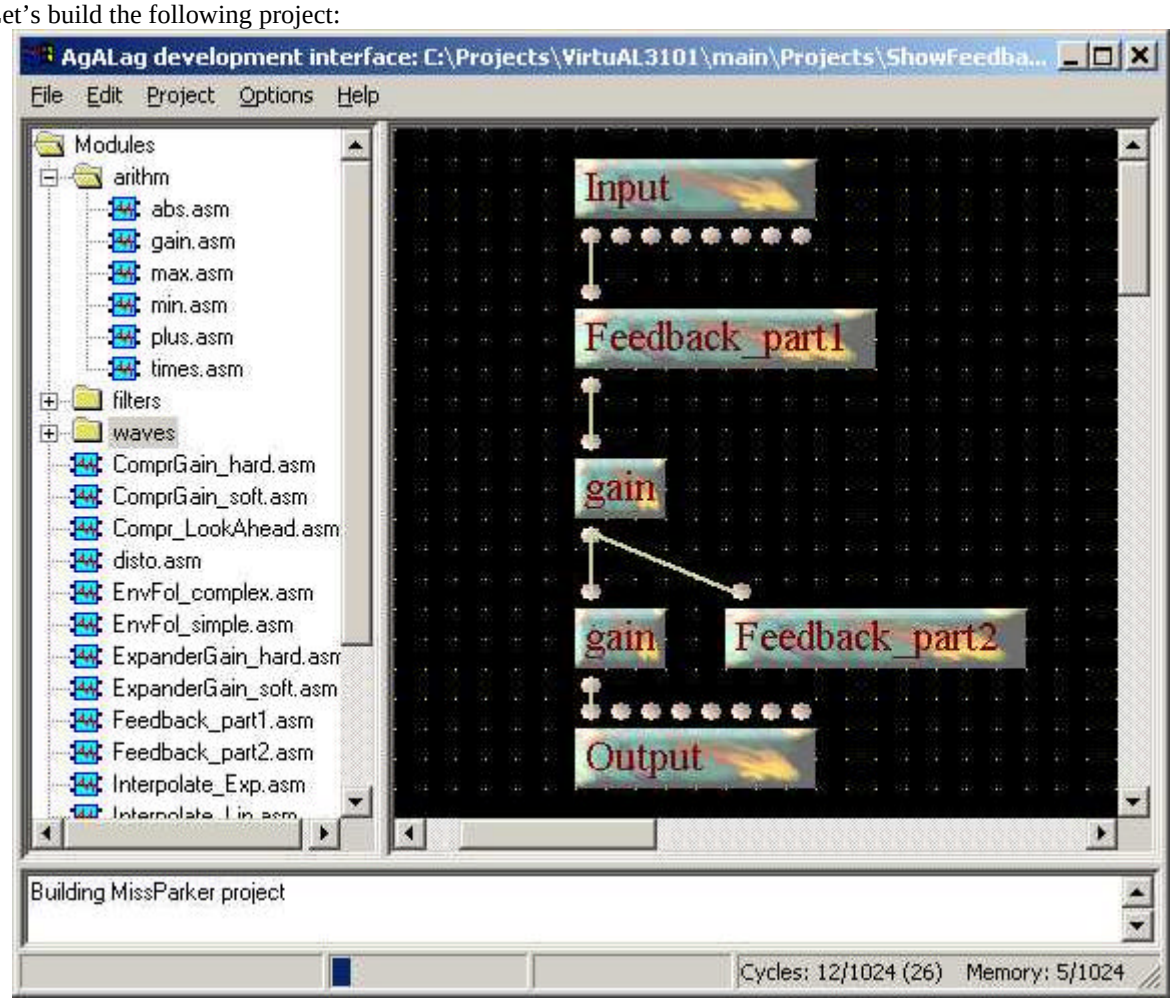



**Figure 15 - Feedback loop example**

The following piece of code can be found in SEC4 which is the section for state variables:

```
        ;; SEC4 Memory variables definition
ABS     Feedback_part14FbState  0x0
ABS     Feedback_part22FbState  0x3
```

This code should be replaced by the following:

```
        ;; SEC4 Memory variables definition
ABS     Feedback_part14FbState  0x0
ABS     Feedback_part22FbState  0x0
```

When "playing" with this trick, please remember:

1. To adapt the .asm code every time a "make" is done.
2. To assemble the .asm into a .obj (if needed) after the modification.

### *How to probe signals for "debug"?*

As AgALag is handling itself the connection of the different modules between each other and most often does it re-using some internal variables, it is not easy to "place" a probe on a intermediate signal.

The solution is to add a "Feedback_part2" module where the signal needs to be probed.

For example, let's try to probe the output of the offset added to the sine wave in the tremolo project. To do this, add a "Feedback_part2" module at the output of the "plus" module as shown below:

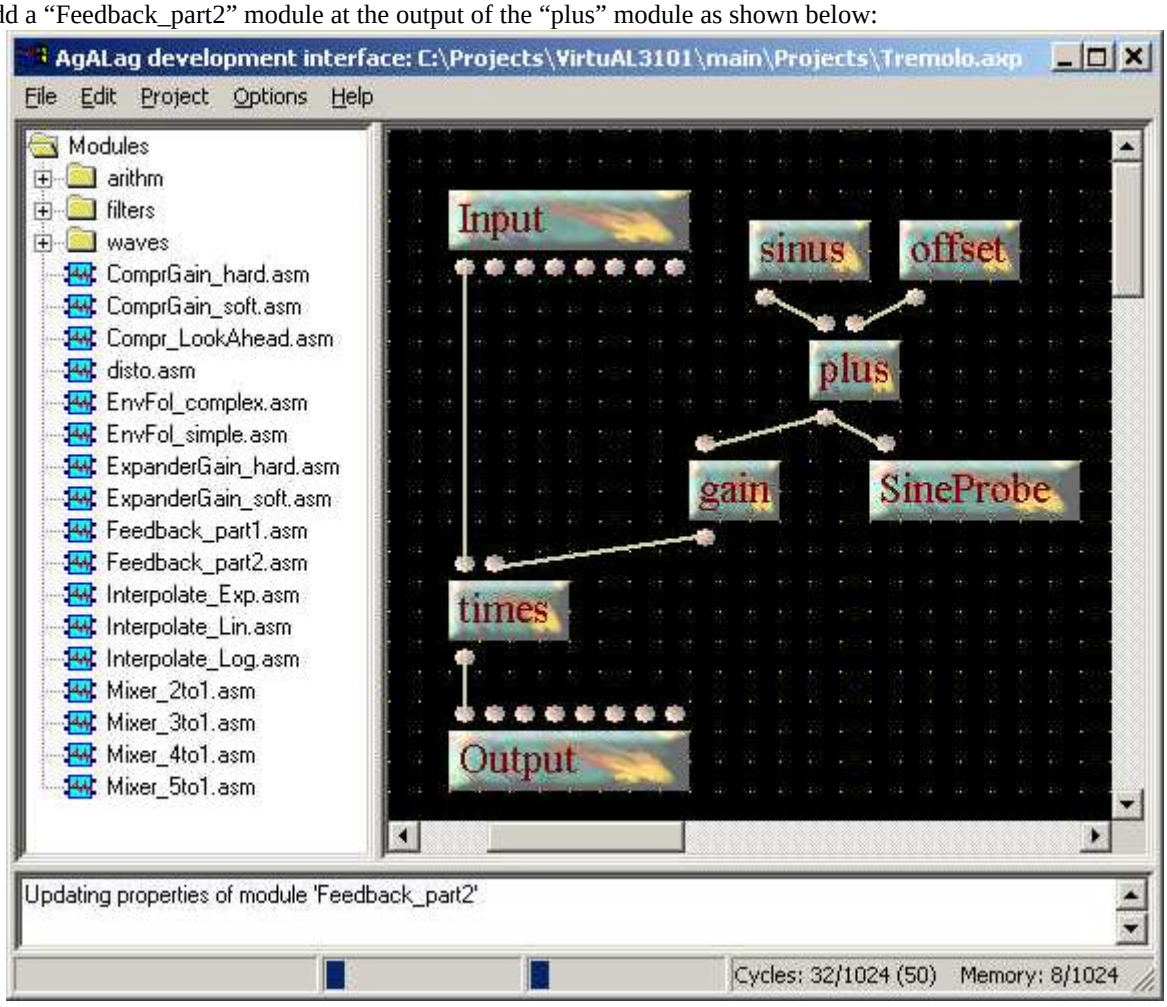



**Figure 16 - Example of a probe**

The assembly code contains the following state variable section:

```
        ;; SEC4 Memory variables definition
ABS     sinus5Up        0x3
ABS     sinus5Value     0x4
ABS     sinus5Tmp       0x5
ABS     SineProbe1FbState       0x6
```

The SineProbe1FbState is the variable that should be monitored in the simulator. This is done by linking an output file to "Data 6".

# 5. Developer tutorial

## 5.1. Introduction

As described in the introduction, a developer is someone who wants to program the AL3101 DSP in assembly and most probably wants to debug the code with the simulator.

In other words, if you want to play with the ALksim simulator, this part is for you…

## 5.2. Tools needed

In order to write and debug code, the VirtuAL3101 suite is needed. These tools are available under the "VirtuAL3101 / Download page" of the Axoris Website (http://www.axoris.be/VirtuAL3101-Download.htm).

The VirtuAL3101 tool suite is made out of several tools:

- Command-line simulator "alksim"
- Assembler "asm" and Disassembler "disasm" (stand-alone and pre-integrated into the simulator)
- AgALag application generator as described in previous chapter.

The tools to be used by the developer are typically a text editor, the ALksim for validation of the code and the assembler in order to get an object file.

## 5.3. Getting started

The ALksim binary is located in "install-dir\bin" where "install-dir" is the directory where VirtuAL3101 has been installed.

An important directory in "install-dir" is the "Effects" directory. This is where the available assembly files are stored. They can be used 1) for inspiration and 2) as example of AgALag-compliant modules.

## 5.4. Basic functionality

The simulator is capable of various basic operations:

- Load an assembly or object file and the related memory map file as generated by AgALag
- Link input and output files to certain memory locations (data, IO, scratch, registers)
- Show and set values in the DSP memories (data, scratch, registers)
- Set breakpoints in many ways
- Step into the code

## 5.5. Practical example

### Simple simulation

Let's use a practical example to get acquainted with the above-mentioned commands… We'll also write a bugged code on purpose in order to show how to use the simulator to debug some piece of wrong code.

First thing first, let's write a small assembly code that we can simulate and save it under "test.asm":

```
CM    0.5   IN1
SCA   0.0   OUT1
CM    0.25  0x0
SCA   0.0   OUT2
```

A test file (called "in.dat") should be created. We'll choose the .dat format for easiness reasons as one only needs to write floating-point numbers in a text file. Let's then fill in "in.dat" with the following data:

```
1.0
0.5
-0.25
2.0
```

Let's now load the different files by using the following commands ("screenshot" of the ALksim):

```
ALESIS AL3101/1KM SIMULATOR v0.1 , AXORIS (c) Copyright
1KSM> load test.asm
Assembling ...
Assembling successfully completed

... Sucess assembling : 4 lines
Cannot find a memory map file, initialization done by program itself
1KSM> +input file in.dat 48000 0
1KSM> +input file in.dat 0 data 10
1KSM>
```

First, we load the assembly file with the "load" command. The simulator tells that it cannot find a memory map file so any initialization of memory has to be done in the program itself. The simulator can open object files as well.

Then, we attach the input file to the IN1 memory location using the "+input file" command. The input file format is determined by the extension of the file; in this case, we use an ASCII floating-point file format. After the file name, one can find the sample rate that is going to be used. This parameter has no direct impact on the simulation itself, it is only used when writing output files of .wav format for example. The last parameter specifies that we will put the file content in IN1 (by default, we read to Input memories and we specified an offset of 0).

Finally, we attach the same input file to the IN2 memory location. This time, the sample rate parameter is 0 meaning that the already defined sample rate will be used. Now, we've also specified that we want the file to be put in the Data[10] memory location.

Let's now attach the output files and launch the simulation:

```
1KSM> +output file outL.dat 0 0
1KSM> +output file outR.dat 0 out 1
1KSM> go
Simulation running.
1KSM> 1KSM>
Simulation complete.
1KSM> reset all
1KSM>
```

The "+output file" command works in the same way as the "+input file" one. The Output memory is chosen by default here.

We can use the "go" command to launch the simulation. The simulation will stop when one of the input files will be exhausted. You therefore have to take care that an input file (or a tool as we'll see later) is loaded otherwise the simulation won't stop.

Finally the "reset" command should be used to close the input and output files. In this case, we've reset everything (files, DSP memories, registers, breakpoints …) by specifying we want a "reset all".

Let's take a look at the output files to check that our code has worked properly:

| **OutL.dat :** | **OutR.dat :** |
|---|---|
| 0.500000 | 0.250000 |
| 0.250000 | 0.125000 |
| -0.125000 | -0.062500 |
| 1.000000 | 0.500000 |

### *Debugging using ALksim*

Let's now write a bugged code to get an occasion of using the debugging features of the simulator…
Therefore, let's replace the previous "test.asm" file by the following that is supposed to compute the absolute value of the input (the aim is not to exercise on an "intelligent" bug but to exercise simply):

```
CM    1.0   IN1   ; A contains the input
SKIP  N     1     ; if the number is positive don't do next line
CAD   -1.0  0.0   ; invert A if number if negative
SCA   1.0   OUT1  ; store the absolute value
```

The bug is located on the second line, we should have "!N" in there instead of "N".

Let's use the following commands:

```
1KSM> load test.asm
Assembling ...
Assembling successfully completed

... Sucess assembling : 4 lines
Cannot find a memory map file, initialization done by program itself
1KSM> +input file in.dat 48000 0
1KSM> +output file out.dat 0 0
1KSM> go
Simulation running.
1KSM> 1KSM>
Simulation complete.
1KSM> reset all
1KSM>
```

The output shall be as follows:

```
-1.000000
-0.500000
-0.250000
-2.000000
```

As we can see, it's not really the result we were hoping for. Again, the reason looks pretty obvious but let's try to investigate why using the following command sequence…

```
1KSM> load test.asm
Assembling ...
Assembling successfully completed

... Sucess assembling : 4 lines
Cannot find a memory map file, initialization done by program itself
1KSM> +input file in.dat 48000 0
1KSM> +dasm
1KSM> step

#0001: SKIP     N 1

1 step done.
1KSM> 1KSM> show a
A = 1.000000
1KSM> step

#0002: CAD      -1.0 0.0
1KSM>
1 step done.
1KSM> show a
A = 1.000000
1KSM>
```

After usual load of assembly and input file, we've used the "+dasm" command to enable printing of following assembly line to be executed on next cycle.

Then we simply step into the code using the "step" command. A number of steps to be executed can be specified as a parameter.

Between each step, let's use the "show" command to display the value of the A register.

Doing all this, we can directly see that an input of 1.0 leads to the branch where the input gets inverted. We now know where out bug stands. Let's change the "SKIP N 1" line into "SKIP !N 1" so that our code works properly.

We'll use this time the "+watch" command so that the A register gets displayed at each step.

```
1KSM> load test.asm
Assembling ...
Assembling successfully completed

... Sucess assembling : 4 lines
Cannot find a memory map file, initialization done by program itself
1KSM> +input file in.dat 48000 0
1KSM> +dasm
1KSM> +watch a
1KSM> step

A = 1.000000 *

#0001: SKIP     !N 1

1 step done.
1KSM> 1KSM> step

A = 1.000000

#0003: SCA      1.0 OUT1

1 step done.
1KSM> 1KSM>
```

This time we see that the branch is correctly taken and that the A register has the correct value.

One may note the * next to the first displayed value of A; this is to show that the value has changed in the last cycle.

Another option for debugging bigger code is to use breakpoints. Breakpoints on program addresses or on memory locations having a given value are possible. On top of that, temporary breakpoints with a limited hit number can be used.

```
1KSM> +break 3
1KSM> +tempbreak 4 2
1KSM> +valbreak 0.5 data 10
1KSM>
1KSM>
1KSM> showbreak
Temporary breakpoint #1 at address 2. 4 breaks remaining.
Breakpoint #1 at address 3
Breakpoint on value #1 on Data[10] = 0
1KSM>
```

The "+break 3" sets a breakpoint at program address 3 while the "+tempbreak 4 2" sets a breakpoint at address 2 but that will be breaking only for 4 times. The "+valbreak 0.5 data 10" will break execution when the Data[10] memory location will contain 0.5.
At every moment, the breakpoints can be listed using the "showbreak" commands.
To delete breakpoints, you should use the "-break" and "-valbreak". The same kind of approach is to be used with watches, inputs and outputs.

## 5.6. Enhanced features

The simulator is also capable of more:

- Audio streaming via MME or ASIO on Windows systems and via OSS on Linux systems
- Script files
- Trace and signal files
- Compute the gain of a given block via an integrated tool (amplitude sweep sine wave)
- Compute the transfer function of a given block via an integrated tool

### Audio streaming in ALksim

On Linux, streaming is implemented through OSS. On Windows, three versions of the simulator are available: the first without streaming at all, one using the standard MME drivers (to use by default) and one using ASIO drivers.

First of all, starting streaming under ALksim is equivalent to launching a simulation with the "go" command. This means that all tools, file IO, … will be handled in the very same way regardless of how the simulation is started.

The streaming can be started in two ways: a very simple one allowing stereo in and out only and a more flexible one allowing more channels.

By default the streaming is recording an audio input and playing then back an audio output. Let's modify a bit the original "test.asm" file:

```
CM    1.0   IN1         ; load the left input in A
AMC   0.0   0x0         ; let's apply a gain on the signal
SCA   1.0   OUT1        ; and save the result in left output
SCA   0.0   OUT2        ; then in the right output
```

This code will simply copy the input, apply a gain and copy the results in the left and right outputs. The following sequence of commands can then be used:

The streaming is by default performed from IN1/IN2 to OUT1/OUT2. That's the reason why the code has been written in this way.

```
1KSM> load test.asm
Assembling ...
Assembling successfully completed

... Sucess assembling : 4 lines
Cannot find a memory map file

1KSM> setval data 0 0.5
1KSM> startstream 48000
1KSM>
```

The "setval" command allows to set the Data[0] memory location to 0.5 which basically means that a gain of 0.5 will be applied to the signal.

One may note that the "startstream" command gives the hand back to the user (just like the "go" command by the way). This is a pretty interesting feature for two reasons:

- The "stopstream" command can be called to stop the streaming on user's request
- The "setval" command can be called while streaming allowing for a real-time change of parameters

It is also possible to perform file to audio output streaming using a special trick of the simulator. Indeed, the streaming copy to memory occurs before the file IOs are made. This basically means that a file can overwrite the audio input. This feature can be used in the following way:

```
1KSM> load test.asm
Assembling ...
Assembling successfully completed

... Sucess assembling : 4 lines
Cannot find a memory map file
1KSM> +input file synths.wav 0 0
1KSM> startstream
```

The "+input file" command used in this way will open the "synths.wav" file (a mono file at 44.1kHz in this case) and will link it to IN1 (Input memory with offset 0) and will use the sample rate as defined in the file.

The more complicated way of starting streaming and of specifying stream parameters is explained in the ALksim user's guide.

## *Script files*

It is possible to use scripts with the simulator. A script is basically an ASCII text file containing a succession of commands.

That's a very interesting feature in two cases at least:

- Perform repeatable tasks before debugging like loading the files, setting watches and breakpoints and launch simulation. The user then gets the simulator in a known state ready for some debugging…
- Perform some automated tests. This is made feasible by the fact that the simulator can take a script file as argument when it's launched.

In order to run automated tests, one would however need to be able to launch a simulation in "blocking mode" so that the simulator does not exit directly after the "go" command because of the "exit" command called right after. This is solved by the "go blocking" command that does not give the hand back to the user. Please be careful with this feature as there's no other way of stopping the simulation (other than file exhausted of course) than killing the program with Ctrl-C.

If an error occurs during the execution of a script launched within the simulator, the script will be stopped.
If an error occurs during the execution of a script launched via the argument of the executable, there are two options:

- Only the script name was specified, then the simulator will give the hand back to the user as if the script would have been called from within the simulator
- The "exit_if_error" keyword is provided as a second argument, the simulator will exit allowing for continuation of batches for example.

## *Trace and signal files*

On top of the known input and output files, traces and signals can be used. These are basically the same file IO mechanism but for which samples are read/written every single cycle instead of every 1024 cycles.

The "+trace" and "+signal" files are used in exactly the same way as the "+output" and "+input" files.

### *AmpSweep tool usage*

The “AmpSweep” tool can be used to measure the static gain function between the input and the output of a block. This is extremely useful when measuring the static response curve of a compressor or of a distortion module.

The tool works by generating a sine wave sweeping in amplitude and measuring the output while comparing the level of the input with the level of the output. A timing should be provided specifying an initial time during which no measurement time is done and also a measurement time during which the measurement is actually done.

The tool will generate two reports (will happen when “reset” is done or when exiting simulator):

- An amplitude report providing the input amplitude in the first column and the output amplitude in the second column. This can be used to display usual compression curves.
- A gain report providing the input amplitude in the first column and the gain between input and output in the second column.

Let’s use the “disto.asm” effect as available from the “effects” directory. This can be done the following way:

```
ALESIS AL3101/1KM SIMULATOR v0.1 , AXORIS (c) Copyright
1KSM> load ./../effects/disto.asm
Assembling ...
Assembling successfully completed

... Sucess assembling : 17 lines
Cannot find a memory map file, initialization done by program itself
1KSM> +tool ampsweep 48000 1000 0.02 0.5 -90 0 5 0 0 amplitude.txt
gain.txt
1KSM> go
Simulation running.
1KSM> 1KSM>
Simulation complete.
1KSM> reset all
1KSM>
```

Here is the meaning of the “+tool ampsweep” command:

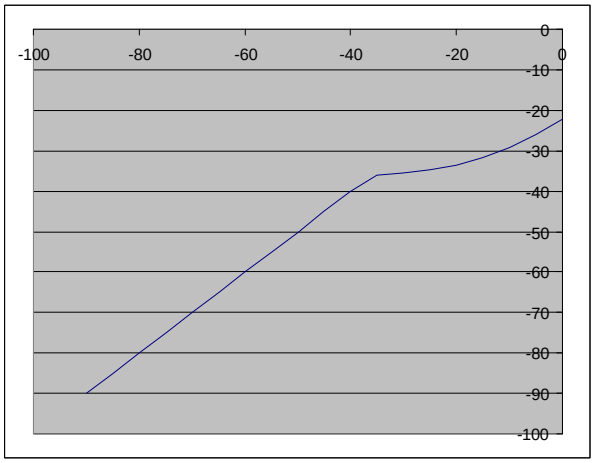
- 48000 is the sample rate to be used; this value is expressed in Hz
- 1000 is the frequency of the sine wave that will be used; this value is expressed in Hz
- 0.02 is the setup time during which no measurement is done; this value is expressed in seconds
- 0.5 is the measurement time; this value is expressed in seconds
- -90 is the lowest amplitude to be generated; this value is expressed in dB.
- 0 is the highest amplitude to be generated; this value is expressed in dB
- 5 is the amplitude increment between two steps; this value is expressed in dB
- 0 is the offset of the input in the Input memories therefore meaning IN1
- 0 is the offset of the output in the Output memories therefore meaning OUT1
- amplitude.txt is the name of the amplitude report file
- gain.txt is the name of the gain report file

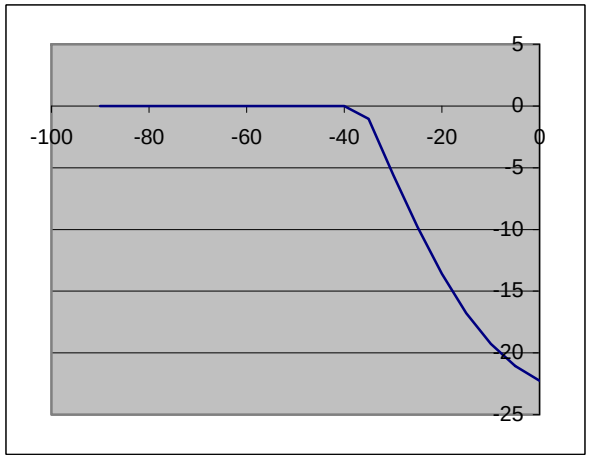
Some memory specifiers can be used if measurement should be done from other memory areas.

The reports should look as follows (except for the two first lines that are not present in the reports):

| **Amplitude report** | | **Gain report** | |
|---|---|---|---|
| Input (dB) | Output (dB) | Input (dB) | Gain (dB) |
| -90.0089 | -90.0089 | -90.0089 | 0 |
| -85.0042 | -85.0042 | -85.0042 | 0 |
| -80.0037 | -80.0037 | -80.0037 | 0 |
| -75.0013 | -75.0013 | -75.0013 | 0 |
| -70.0007 | -70.0007 | -70.0007 | 0 |
| -65.0005 | -65.0005 | -65.0005 | 0 |
| -60.0001 | -60.0001 | -60.0001 | 0 |
| -55.0002 | -55.0002 | -55.0002 | 0 |
| -50 | -50 | -50 | 0 |
| -45 | -45 | -45 | 0 |
| -40 | -40 | -40 | 0 |
| -35 | -36.0489 | -35 | -1.04892 |
| -30 | -35.5848 | -30 | -5.58483 |
| -25 | -34.8162 | -25 | -9.81621 |
| -20 | -33.5977 | -20 | -13.5977 |
| -15 | -31.7802 | -15 | -16.7802 |
| -10 | -29.2656 | -10 | -19.2656 |
| -5 | -26.0563 | -5 | -21.0563 |
| 0 | -22.2535 | 0 | -22.2535 |

The amplitude and gain can be plot using appropriate tools. This gives the following view in this case:




As it can be seen on the right graph, the attenuation applied is growing as the input level is growing. This is typical of a distortion block.

The left graph shows the output level given a certain input level. One can again notice the reduced output level for high levels at the input.

## *FreqResp tool usage*

The “FreqResp” tool can be used to measure the transfer function between the input and the output of a block. This is extremely useful when measuring the frequency response of a system containing filters.

The tool works by generating white noise and measuring the output while comparing the frequency spectrums of both signals. The tool is working with blocks of size as defined by the user; a given number of blocks can be discarded at the start to allow the system to go in “steady state”.

The tool will generate two reports (will happen when “reset” is done or when exiting simulator):

- An amplitude report providing the magnitude of the transfer function.
- A phase report providing the phase difference between the input and output.

Let’s use the “HPF_coefsw.asm” effect as available from the “effects” directory. This can be done that way:

```
1KSM> load ./../effects/filters/HPF_coefs.asm
Assembling ...
Assembling successfully completed

... Sucess assembling : 79 lines
Cannot find a memory map file, initialization done by program itself
1KSM> +tool FreqResp 48000 2048 4 -6 0 0 amplitude.txt phase.txt
1KSM> go
Simulation running.
1KSM>
Simulation complete.
1KSM> 1KSM> reset all
1KSM>
```

Here is the meaning of the “+tool freqresp” command:

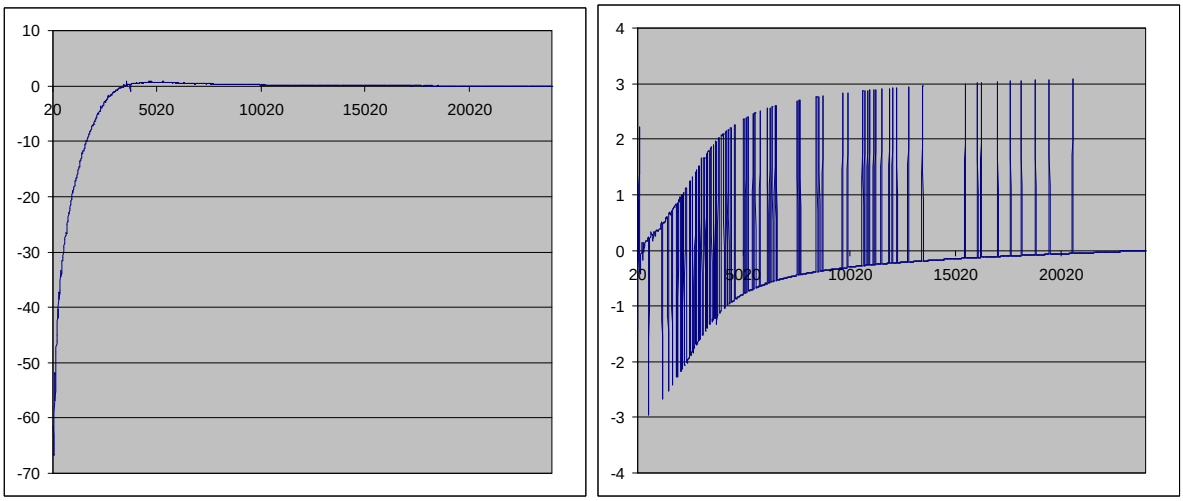
- 48000 is the sample rate to be used; this value is expressed in Hz
- 2048 is the size of the block on which the measurement is done
- 4 is the number of blocks that will be skipped in order to come to a “steady state”
- -6 is the peak amplitude to be generated; this value is expressed in dB.
- 0 is the offset of the input in the Input memories therefore meaning IN1
- 0 is the offset of the output in the Output memories therefore meaning OUT1
- amplitude.txt is the name of the amplitude report file
- phase.txt is the name of the phase report file

Some memory specifiers can be used if measurement should be done from other memory areas.

The reports should look as follows (except for the two first lines that are not present in the reports):

| **Magnitude report** | | **Phase report** | |
|---|---|---|---|
| Freq. (Hz) | Magnitude(dB) | Freq. (Hz) | Phase (radians) |
| 0 | -1.#IND | 0 | 0 |
| 23.4375 | -61.4164 | 23.4375 | -1.42321 |
| 46.875 | -64.6612 | 46.875 | 0.879427 |
| 70.3125 | -66.5221 | 70.3125 | -0.0425518 |
| 93.75 | -52.1758 | 93.75 | 2.22035 |
| 117.188 | -56.8304 | 117.188 | -0.173209 |
| 140.625 | -54.3136 | 140.625 | -0.427522 |
| 164.063 | -48.1384 | 164.063 | -0.0780397 |
| 187.5 | -48.0103 | 187.5 | -0.0409121 |
| … | … | … | … |
| 23812.5 | -4.66468e-005 | 23812.5 | -0.00276298 |
| 23835.9 | -0.000572849 | 23835.9 | -0.0024997 |
| 23859.4 | -0.00387735 | 23859.4 | -0.0023207 |
| 23882.8 | -0.00495848 | 23882.8 | -0.0014553 |
| 23906.3 | -0.00141538 | 23906.3 | -0.00142855 |
| 23929.7 | -0.00102262 | 23929.7 | -0.000992635 |
| 23953.1 | 0.000871227 | 23953.1 | -0.000524653 |
| 23976.6 | 0.000289597 | 23976.6 | -0.000372907 |
| 24000 | 0.00196585 | 24000 | 0 |

The transfer function curves can be displayed as well using appropriate tools:




On the left graph, the high-pass behaviour is clearly visible with the low frequency attenuated.

The right graph shows the phase behaviour of the system with the phase moving from -Đ to 0.0. The spikes are phase shift of Đ and can therefore be omitted. The simulator will be modified in the future so that these shifts are removed when generating the report.

## *5.7. Tips & Tricks*

Part to be written soon...

# 6. Complete project from scratch

Part to be written soon...

# 7. Appendix 1 – SineFreqTable file content

```
0
110
220
330
440
550
660
770
880
990
1100
1210
1320
1430
1540
1650
1760
1870
1980
2090
2200
2310
2420
2530
2640
2750
2860
2970
3080
3190
3300
3410
3520
3630
3740
3850
3960
4070
4180
4290
4400
4510
4620
4730
4840
4950
5060
5170
5280
5390
5500
5610
5720
5830
5940
6050
6160
6270
6380
6490
6600
6710
6820
6930
7040
7150
7260
7370
7480
7590
7700
7810
7920
8030
8140
8250
8360
8470
8580
8690
8800
8910
9020
9130
9240
9350
9460
9570
9680
9790
9900
10010
10120
10230
10340
10450
10560
10670
10780
10890
11000
11110
11220
11330
11440
11550
11660
11770
11880
11990
12100
12210
12320
12430
12540
12650
12760
12870
12980
13090
13200
13310
13420
13530
13640
13750
13860
13970
```