September 2004.

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

ALkSim
AL3101 DSP Simulator

Command-line help

1. Introduction

This documents describes the command-line version of the 1K DSP simulator and provides help for its commands.

This command-line version of the simulator contains all the functionality required to make code debugging and simulation. It basically allows users to test their code before they have access to a development board.

The simulator has been designed in such a way that it compares bit-true with execution on the chip. There are however some limitations on the hardware supported; these limitations are described later in this document. More information on the 1K DSP chip can be found on http://www.wavefrontsemi.com (formerly Alesis Semiconductors).

2. Basic features

The command-line simulator (*will*) contain the following features (*features in italic are not implemented yet*):
Input can be either assembly files or object files; assembler and disassembler are included.

- Stepping by any step in the code.
- Easy watching of variables, memory locations, …
- Breakpoints on Program Counter and on condition on a value (register or memory).
- Display of currently executed instruction.
- File support for .wav and .pcm sound files for floating-point ASCII files and for decimal/hexadecimal ASCII files*:*
    - Read from a file to any input or output location, any register or any memory location.
    - Write to a file from any input or output location, any register or any memory location (Tracing).
- Generator tools for generating sine, square, triangle or sawtooth waves at the 'inputs of the chip'.
- Static amplitude curve measurement tool (via sine wave with sweeping amplitude).
- *Frequency analysis tool for inputs and outputs.*
- Transfer function measurement tool (giving amplitude and phase characteristic).
- Scripting in order to execute a list of commands and ease the complete simulation of an application (load input files, specify output files, run, …).
- Script names can be given as argument to the simulator so that simulation can be done via batches. This helps when making extensive tests or when validating modules.
- Aliases for command names.

These features are (*will be*) supported by various commands that are described later in the document.

When simulating reasonably sized code, the simulator is also capable of making streaming from/to a sound board; the reasonably is of course highly dependent on the power of the computer used. The streaming from/to the sound board is relying on the PortAudio library which is an open-source multi-platform library; more information can be found on http://www.portaudio.com.

Last but not least, the simulator has been designed and written to be multi-platform; it can work on Windows operating systems and on Linux operating system as well. This is achieved with the help of the wxWidgets library (formerly wxWindows); more information can be found on http://www.wxwidgets.org.

Multi-threading has been used in this simulator. It allows, for example, entering commands while some long simulation is running which is of interest if one wants to stop a simulation or to set some parameters to other values while audio streaming is ongoing. See 6.Known problems, for more details.

3. Limitations

The simulator has been designed and written to give bit-true results compared to the execution on the chip (except for the LOG and EXP functions). The simulator does however NOT simulate all the hardware present on the chip. The following features are missing :
- External control i.e. read from status register, write to control registers,…
- Memory offset counter.
- Pin accesses via read/write to memory locations made for that (addresses in the range $421 to $42F).
- Peak meters.

Some features are present in order to increase the ease of use:
- Initial value in control register such that simulation can begin.
- To ease stepping through the code, only the cycles needed (the one containing real code) in the 1024 range are executed.

These features can of course be customized or disabled and are stored via a “configuration” mechanism i.e. registry on Windows platform and configuration files on Linux.

Even though the simulator has been written to work as fast as possible, there will be no support for real-time operation i.e. using sound cards to listen real-time to the simulation running.

4. About the simulator “philosophy”

This paragraph describes a bit the simulator “philosophy” before going in detail on all the available commands. This is supposed to give a kind of overview on what to do to use, in practice, the simulator.

First of all, the very first obvious action is to load a program (assembly code or object file) in the simulator; this is done with the “load” command. The second action can be to load some memory content in the data memory; this is also handled via the “load” command.

The second step will certainly be to associate input(s) and/or output(s) to some memory locations where memory location stands for any register or any memory location of the chip. Two different ways of providing IOs to the simulator exists where the difference stands in when the data is read/written.
- The first and most obvious way of reading/writing data is to use the “+input” and “+output” commands. These commands read/write data every 1024 cycles i.e. as it happens on the chip with the IN and OUT memory locations.
- The second way of reading/writing data is to use the “+signal” and “+trace” commands. These commands read/write data every cycle.

The first one is especially suited to simulate the hardware IOs or to track what is happening in a filter state while the second one is especially suited to track what is happening all over the time in a register or temporary memory location.
Actually, another way of providing IOs to the simulator exists; a tool can be used to compute a transfer function, … by applying some data and computing the results. This is done with the “+tool” function and the read/write happen each 1024 cycles.

Then finally, the code can be executed using the “go” command. When simulation is over, a simple “exit” command will exit the simulator.

When it comes to the debugging issue, some commands have been added to the simulator to ease the life of the programmer. Stepping through the code is allowed via the “step” command while different kind of breakpoints on instructions can be used with the “+break”, “-break” and “+tempbreak” command. Breakpoints on values can also be used via the “+valbreak” and “-valbreak” commands.
Some display routines are also provided to help debugging like printing of the executed line with the “dasm” command and the likes but also like displaying the content of any memory location with the “show” command.

Some user parameters can finally be set by using the “setuser” command. These parameter and the aliases (+alias, -alias, showalias, -allalias) will help the user to customize his working environment.

5. Commands

Here is finally about the real stuff: how can I use this simulator?

The commands are not given in an alphabetical order but more in a “logical” order of use in real life. Only implemented commands are described.

Parameters between brackets are optional parameters.

5.1. Basic commands

quit / exit

Syntax : quit / exit
Feature : As one may guess, this command simply exits from the simulator.
Params : None.
Issues : If a simulation is running, the simulation will be stopped and the files will be saved (except for the tool reports).

load

Syntax : load filename [complete]
Feature : Loads a “program” file into the simulator (and the associated memory map file).
Params : *filename* is the file to open; it can be an assembly (.asm), object (.obj) or script file (.cmd).
: *complete* is specifying what should be done with the memory left (when program size less than 1024):
- when equal to “UNCHANGED”, the memory left is unchanged.
- when not specified, the memory left will be reset.

Issues : The files must have one of the following extensions: .asm, .obj, .cmd in order to be recognized.
The simulator looks for a memory map file with the same name but with a .map extension.
Refer to AgALag / mkALproj documentations for more details on the memory map file.
If the file is not found, the simulator says so by a warning.

help

Syntax : help [command]
Feature : Prints help on available commands.
Params : *command* specifies for which command help is required. If omitted, the list of commands is shown.
Issues : None.

loadscript

Syntax : loadscript filename
Feature : Loads and execute a script file with no restrictions on the file extensions as in the “load” command.
Params : *filename* is the script file to open.
Issues : Since the “go” command is not blocking, the “exit” command should not be used in scripts.

5.2. Miscellaneous commands

version

Syntax : version
Feature : Prints version number of the program.
Params : None.
Issues : None.

copyright:

Syntax : copyright
Feature : Prints copyright notice for the simulator.
Params : None.
Issues : None.

5.3. Program flow commands

start (or "go")

Syntax : start [blocking] [address]
Feature : Starts a simulation (continues after stop or steps occurred).
Params : *blocking* is specifying if the simulation should block the command-line or not.
- when equal to "blocking", the command-line is blocking while simulating.
- when not specified, the control is given back to the user.

: *address* is specifying the Program Counter value where simulation should start.
Issues : Setting of PC address when starting is not recommended as it can lead to non-read or non-written data.
A reset might be performed at completion of a simulation (see 5.8.User parameter commands).

step

Syntax : step [blocking] [n]
Feature : Steps through instructions in the program.
Params : *blocking* is specifying if the simulation should block the command-line or not.
- when equal to "blocking", the command-line is blocking while simulating.
- when not specified, the control is given back to the user.

: *n* is the number of steps to be executed; if omitted, a step of a single instruction is made.
Issues : The +dasm and –dasm commands allows printing or not of disassembled line when stepping.

stop

Syntax : stop
Feature : Stops a running simulation (launched with start or with step and a pretty big number of steps).
Params : None.
Issues : None.

reset

Syntax : reset [type]
Feature : Resets the simulation by cleaning data memory, closing files and/or resetting Program Counter.
Params : *type* specifies on what the reset should be applied.
- when equal to "IO", all the IOs will be reset.
- when equal to "IO_reopen", the IOs will be reset and reinitialized again.
- when equal to "Prog", the program memory and breakpoints will be reset.
- when equal to "Prog_only", only the program memory will be reset.
- when equal to "Data", the data memory and watches will be reset.
- when equal to "Data_only", only the data memory will be reset.
- when equal to "DSP", reset as it would happen in hardware i.e. program and data memory.
- when equal to "All", everything is reset (program, data memories and IOs).
- when equal to "All_reopen", everything is reset and IOs are reinitialized again.
- when not specified, a "reset all" will be done.

Issues : With all types of reset, the internal DSP state (A, B, U, program counter) will be reset.
The IOs are reinitialized with the same command-line as the one used for the initialization.
Aliases are never reset. The "-allalias" command should be used to perform this task.
The rounding flag is never affected by the reset.

5.4. IO related commands

These commands are not limited to inputs and outputs as strictly defined in the DSP specification. Registers and any memory locations can be used as input or output locations.

+input

Syntax : +input [type] [type-specific parameters]
Feature : Sets an input of the simulator to be loaded before the 1st cycle (of the 1024) is executed.
Params : *type* describes which kind of "input device" will be attached to an input location; it can be
- file: the "input device" to connect is a file (see *5.9."+input file" command* for details).

Issues : Attaching multiple files to a single input is not forbidden, user should take care it doesn't happen.

+output

Syntax : +output [type] [type-specific parameters]
Feature : Sets an output of the simulator to be saved after the 1024th cycle is executed.
Params : *type* describes which kind of “output device” will be attached to an output location; it can be
- file: the “output device” to connect is a file (see *5.10.“+output file” command* for details).

Issues : Attaching multiple files to a single output is not forbidden, user can exploit that feature if needed.

+signal

Syntax : +signal [type] [type-specific parameters]
Feature : Sets a signal for the simulator (a kind of input) to be loaded after each instruction is executed.
Params : *type* describes which kind of “signal device” will be attached to an input location; it can be
- file: the “signal device” to connect is a file (see *5.11.“+signal file” command* for details).

Issues : Attaching multiple signals to a single location is not forbidden, user should take care it doesn’t happen.

+trace

Syntax : +trace [type] [type-specific parameters]
Feature : Sets a trace of the simulator (a kind of output) to be saved after each instruction is executed.
Params : *type* describes which kind of “output device” will be attached to an output; it can be
- file: the “output device” to connect is a file (see *5.12.“+trace file” command* for details).

Issues : Attaching multiple traces to a single location is not forbidden, user can exploit that feature if needed.

+tool

Syntax : +tool [type] [type-specific parameters]
Feature : Specify usage of tool with the simulator. A tool can read and/or write to locations from $410 to $417.
Params : *type* describes which tool will be used; it can be
- AmpSweep: the static amplitude curve measurement tool will be used.
  More information can be found in *5.15*.

"+tool AmpSweep" command.
- FreqResp: the frequency response curve measurement tool will be used.
More information can be found in *5.16. "+tool FreqResp" command*.

Issues : Interaction (for memory read/write) between +tool and +input and/or +output is not checked.

-input

Syntax : -input InputIndex
Feature : Removes the input given by its index.
Params : *InputIndex* is the index of the input to remove; index is found from the "showinput" command.
Issues : Use the "showinput" command to find the index of the input to remove.
: The same stands for the "-output", "-signal", "-trace" and "-tool" commands.

-allinput

Syntax : -allinput
Feature : Removes all the inputs.
Params : None.
Issues : The same stands for the "-alloutput", "-allsignal", "-alltrace" and "-alltool" commands.

showinput

Syntax : showinput
Feature : Displays all the inputs created with their initial command string.
Params : None.
Issues : This command associates index with inputs; this index has to be used with the "-input" command.
: The same stands for the "-alloutput", "-allsignal", "-alltrace" and "-alltool" commands.

loopinput

Syntax : loopinput InputNumber [Mode]
Feature : Allows looping on input files. The command has no effect on generators.
Params : *InputNumber* is the number of the input which mode should be modified.
: *Mode* specifies whether to loop ("on") or not to loop ("off").
Issues : Use the "showinput" command to find the index of the input to modify.
: Anything else than "off" will be considered as "on" thus enablin g the looping.

5.5. Streaming related commands

First of all, by streaming is meant streaming from/to a sound card. The implementation of streaming in this simulator is made based on the PortAudio library.

This library allows streaming on all kind platforms using the same API but different source files. As a consequence:

- Under Windows, there will be one executable for the standard Multimedia extension (MME) drivers and one executable using ASIO drivers if available.
- Under Linux, there will be one executable for the OSS drivers and one executable for the ALSA drivers.

+startstream

The startstream command starts recording the input, simulating the code on this input and then playing the output.

The startstream command can be used in two ways:

- Specifying the inputs and outputs to be used
- Without specifying any input or output.

Syntax : startstream SamplingFrequency NumberOfChannels Input1 … Input8 Output1 … Output8
Feature : Start streaming.
Params : *SamplingFrequency* specifies the sampling frequency to use.
If set to "0", the sampling frequency known by the simulator (if available) will be reused.
NumberOfChannels specifies the number of channels (in the range [1…8]) to use for streaming.
InputLocation1…8 specifies the input location.
OutputLocation1…8 specifies the output location.
Issues : The same number of channels is used for both recording and playback.
The same syntax as the syntax of the "show" command can be used to specify the IOs.
The memory specifier by default for input is "in" and for output is "out".
Obviously the number of channels chosen must match the capabilities of the sound board.

Syntax : startstream [SamplingFrequency]
Feature : Start streaming.
Params : *SamplingFrequency* specifies the sampling frequency to use.
If set to "0", the sampling frequency known by the simulator (if available) will be reused.
Issues : The inputs and outputs which will be used are:

- o Either the one used previously with the IO specification in the startstream command.
- o Or stereo streaming is performed with "in 0" and "in 1" as inputs and "out 0" and "out 1" as outputs.

Warning: The reading of the stream inputs is made before the reading of the files, generators and tools; this means that the stream inputs can be overwritten by the other inputs if care is not taken.

Examples:

startstream	will start streaming with previous/default settings.
startstream 0	will start streaming with previous/default settings.
startstream 48000	will start streaming at 48kHz with previous/default IO settings.
startstream 48000 2 0 data 1 0 in 0	will start streaming at 48kHz with "in 0" and "data 1" as inputs and "out 0" and "in 0" as outputs.

As can be seen in this example, the "in" from "in 0" is omitted in the command as "in" is the default memory specifier for the inputs (resp. "out" from "out 0" for the outputs).

It can also be seen that it is possible to have on the left channel the processed sound coming from "out 0" while having on the right the unprocessed original signal coming from "in 0".

+stopstream

Syntax : stopstream
Feature : Stops streaming started with the startstream command.
Params : *none*.
Issues : The stopstream command cannot be used to stop a simulation started with the start command.
The stop command cannot be used to stop a simulation started with the startstream command.

+setstream

Syntax : setstream BlockSize NumberOfBlocks [InputDeviceID] [OutputDeviceID]
Feature : Sets streaming-specific parameters.
Params : *BlockSize* is the obviously the size of blocks to use.
NumberOfBlocks is the number of blocks to be used by the system for buffering.
When set to 0, the optimal number of blocks is used.
InputDeviceID specifies the device ID of the input device to be used.
OutputDeviceID specifies the device ID of the output device to be used.
Issues : The streaming functions use the default input and output devices by default.

Example: “setstream 128 0” will specify usage of optimal number of blocks of 128 default wave devices.

There’s no special need for using this command, it is provided for those who wants to customize the way streaming is done.

The InputDeviceID and OutputDeviceID can be used when multiple sound cards are present on the system and when the “non-default” one should be used.

5.6. Breakpoint related commands

+break

Syntax : +break [address]
Feature : Sets a breakpoint to break execution when reaching a given Program Counter value.
Params : *address* is equal the PC (Program Counter) value where breakpoint should occur.
If omitted, current Program Counter will be used as address.
Issues : None.
Example: "*+break 128*" will set a breakpoint at a Program Counter of 128.

+tempbreak

Syntax : +tempbreak HitNumber [address]
Feature : Sets a temporary breakpoint to break execution for a given number of hits.
Params : *HitNumber* is the number of time the execution will stop before clearing the temporary breakpoint.
: *address* is equal the PC value where breakpoint should occur. If omitted, current PC is used.
Issues : None.
Example: "*+tempbreak 10*" will set a temporary breakpoint for 10 hits at current Program Counter.

-break

Syntax : -break [address]
Feature : Clears a breakpoint (breaking on Program Counter values).
Params : *address* is equal to the PC value where breakpoint should not occur anymore.
If omitted, current PC is used.
Issues : None.
Example: "*-break 10*" will reset breakpoint (temporary or not) at a Program Counter of 10.

+valbreak

Syntax : +valbreak value Reg/Mem_Id [address]
Feature : Sets a breakpoint to break execution when a given memory location reachs a given value.
Params : *value* is the value (given as decimal value) on which the execution should stop.
Reg/Mem_Id specifies the register or the memory bank to use and are defined in "Show" command.
address is equal the location in the memory bank.
It is only required when addressing memory i.e. thus not for registers.
If omitted, the first position of the memory bank will be used (in[0] if parameter is "in").
Issues : None.
Example: "*+valbreak 120 data 100*" will set a breakpoint which will stop execution when Data[100] = 120.

-valbreak

Syntax : -valbreak Reg/Mem_Id [address]
Feature : Clears a breakpoint (breaking on given values).
Params : *Reg/Mem_Id* specifies the register or the memory bank to use and are defined in "Show" command.
address is equal the location in the memory bank.
It is only required when addressing memory i.e. thus not for registers.
If omitted, the first position of the memory bank will be used (in[0] if parameter is "in").
Issues : None.
Example: "*-valbreak data 100*" will delete the above mentioned breakpoint.

-allbreak

Syntax : -allbreak
Feature : Clears all breakpoints (breaking on instruction and on values).
Params : None.
Issues : None.

showbreak

Syntax : showbreak
Feature : Displays information on all existing breakpoints (breaking on instructions and on values).
Params : None.
Issues : None.

5.7. Display related commands

show

This command can be used to display both registers and memory locations. The syntax of the command is a bit different in the two cases as explained below

<u>*Showing of registers:*</u>

Syntax : show Register [DisplayFormat]
Feature : Displays a given register in a given format.
Params : *Register* is specifying which register to show. It can be A, B or U.
: *DisplayFormat* specifies how to display the data. It can be:

- float : use of the floating-point format.
- dS3.24 : data is considered as being a S3.24 number shown in decimal.
- dS3.18 : data is considered as being a S3.18 number shown in decimal.
- hS3.24 : data is considered as being a S3.24 number shown in hexadecimal.
- hS3.18 : data is considered as being a S3.18 number shown in hexadecimal.
- bS3.24 : data is considered as being a S3.24 number shown in binary.
- bS3.18 : data is considered as being a S3.18 number shown in binary.
- dec : data is a S3.24 number shown in decimal (mapped on dS3.24).
- hex : data is a S3.24 number shown in hexadecimal (mapped on hS3.24).
- bin : data is a S3.24 number shown in binary (mapped on bS3.24).

The float format is used as default i.e. when format is not specified.

Issues : It is advised to use the integer format to display flags as their possible values are 0 or 1.

<u>*Showing of memory locations:*</u>

Syntax : show MemoryType [FirstAddress] [SecondAddress] [DisplayFormat]
Feature : Displays a given range of memory in a given format.
Params : *MemoryType* is specifying which memory piece to use. It can be:

- In : for the DSP inputs (0x410 to 0x417 in read mode)
- Out : for the DSP outputs (0x410 to 0x417 in write mode)
- Data (or Mem) : for the Data memory also called Sample Memory (0x000 to 0x3FF)
- Reg (or Scratch) : for the registers also called Direct Access Memory (0x400 to 0x40F)

Where addresses are given for the internal memory mapping as described in the 1K DSP specification.

: *FirstAddress* is the first element to show. This value is 0-based.
It is given as an offset from the beginning of the piece of memory used.
Default value is 0 when FirstAddress and SecondAddress are not specified.

: *SecondAddress* is the last element to show. This value is 0-based.
It is given as an offset from the beginning of the piece of memory used.
Default value is FirstAddress when SecondAddress is not specified.

: *DisplayFormat* specifies how to display the data. It can be:

- float : use of the floating-point format.
- dS3.24 : data is considered as being a S3.24 number shown in decimal.
- dS3.18 : data is considered as being a S3.18 number shown in decimal.
- hS3.24 : data is considered as being a S3.24 number shown in hexadecimal.
- hS3.18 : data is considered as being a S3.18 number shown in hexadecimal.
- bS3.24 : data is considered as being a S3.24 number shown in binary.
- bS3.18 : data is considered as being a S3.18 number shown in binary.
- dec : data is a S3.24 number shown in decimal (mapped on dS3.24).
- hex : data is a S3.24 number shown in hexadecimal (mapped on hS3.24).
- bin : data is a S3.24 number shown in binary (mapped on bS3.24).

The float format is used as default i.e. when format is not specified.
The format can be specified regardless of the number of specified addresses.

Issues : A check is made on the addresses provided to the function fobidding commands like *"show in 16"* .

setval

This command can be used to set a value of both registers and memory locations. The syntax of the command is a bit different in the two cases as explained below

Setting of registers:

Syntax : setval Register Value
Feature : Sets the value of the given register.
Params : *Register* is specifying which register to show. It can be A, B, U, n (flag), v (flag) or z (flag).
: *Value* gives the value to use. The following format are supported:

- float : value written in floating-point format.
- hex : value written as a S3.24 hexadecimal number.
- octal : value written in octal format.
- bin : value is written in binary form.

The format is recognized with the help of type specifiers such as 0x, $ or B and follow the conventions used in the assembler.

Issues : It is advised to use the integer format to display flags as their possible values are 0 or 1.

Setting of memory locations:

Syntax : setval MemoryType Address Value1 … ValueN
Feature : Sets a given range of memory location with given N values.
Params : *MemoryType* is specifying which memory piece to use. It can be:

- In : for the DSP inputs (0x410 to 0x417 in read mode)
- Out : for the DSP outputs (0x410 to 0x417 in write mode)
- Data (or Mem) : for the Data memory also called Sample Memory (0x000 to 0x3FF)
- Reg (or Scratch) : for the registers also called Direct Access Memory (0x400 to 0x40F)

Where addresses are given for the internal memory mapping as described in the 1K DSP specification.

: *Address* is the first element to set. This value is 0-based.
It is given as an offset from the beginning of the piece of memory used.
: *Value1* is the value to set in the first memory location (the one pointed by MemSpecifier:Address).
: *ValueN* is the last value to set in the memory i.e. in (Address + N).

Issues : A check is made to prevent writing out of the boundaries.
The same formats as for the “register case” are supported.
These formats can be mixed from a value to the other.

dasm

Syntax : dasm [BeginAddress] [EndAddress]
Feature : Displays source code (or disassembled code) in the specified range.
Params : *BeginAddress* is the lower limit of the specified range for displaying.
: *EndAddress* is the upper limit of the specified range for displaying. If omitted, BeginAddress is used.
Issues : None.

+dasm

Syntax : +dasm
Feature : Turns on display of executed line when stepping through the code.
Params : None.
Issues : See 6.Known problems, for more information.

-dasm

Syntax : -dasm
Feature : Turns off display of executed line when stepping through the code.
Params : None.
Issues : See 6.Known problems, for more information.

+watch

Syntax : +watch [parameters] (see Show command)
Feature : Adds a watch on a register or on memory locations.
Params : *parameters* are the parameters as defined in the Show command.
Issues : None.

The syntax is basically the same as the one used in the show command; the functionality is also not so different. Indeed, the Show command shows the value of a register or memory locations while the watch command will "perform" the same task but it will happen when stepping and for every step.

-watch

Syntax : -watch WatchIndex
Feature : Removes the watch given by its index.
Params : *WatchIndex* is the index of the watch to remove; index is found from the "showwatch" command.
Issues : Use the "showwatch" command to find the index of the watch to remove.

-allwatch

Syntax : -allwatch
Feature : Removes all the watches.
Params : None.
Issues : None.

showwatch

Syntax : showwatch
Feature : Displays all the watches created in the simulator.
Params : None.
Issues : This command associates index with watches; this index has to be used with the "-watch" command.

5.8. User parameter commands

+alias

Syntax	: +alias AliasName CommandName
Feature	: Adds an alias: AliasName of the given command CommandName.
Params	: *AliasName* is the name of the alias to use.
	CommandName is the name of the command to map on the alias.
Issues	: Alias are stored in the configuration file and thus remembered from one execution to the other.

-alias

Syntax	: -alias AliasIndex
Feature	: Removes the alias given by its index.
Params	: *AliasIndex* is the index of the alias to remove; index is found from the "showalias" command.
Issues	: Use the "showalias" command to find the index of the alias to remove.
	The command will have impact on the configuration file (alias removed from there as well).

-allalias

Syntax	: -allalias
Feature	: Removes all the aliases.
Params	: None.
Issues	: The command will have impact on the configuration file (alias removed from there as well).

showalias

Syntax	: showaliases
Feature	: Displays all the aliases created.
Params	: None.
Issues	: This command associates index with aliases; this index has to be used with the "-alias" command.

"setuser" command

Syntax	: setuser UserParameterName UserParameterValue
Feature	: Sets a given user parameter to a given value.
Params	: *UserParameterName* specifies which user parameter will be set.
	: *UserParameterValue* gives a value to the parameter.
Issues	: The parameter value is dependent on the name of the parameter used.

These parameters are remembered from one use to the other. This is achieved by using some "configuration" mechanism i.e. registry on Windows platforms and configuration files on Linux.

The following user parameters can be set (with their respective allowed values):

-	dasm	: specifies if the disassembled line has to be displayed when stepping or not.
		: value can be "on" or "off"; any other value is seen as an "off" value.
-	showformat	: specifies which default format to use when using the "show" command.
		: value can be one of the allowed *DisplayFormat* of the "show" command.
-	truecycle	: specifies if the simulator should simulate all the 1024 cycles even though they are not "used".
		: value can be "on" or "off"; any other value is seen as an "off" value.
-	resettype	: specifies what type of reset should be performed at the end of a simulation.
		: value is a type as given to the "*reset*" command (see 5.3.Program flow commands).
		: the reset is only applied when the simulation ends itself i.e. not when "*stop*" is used.
-	maxinputs	: specifies what is the maximum number of inputs the simulator should handle.
		: value is an integer number between 0 and 1024.
-	maxoutputs	: specifies what is the maximum number of outputs the simulator should handle.
		: value is an integer number between 0 and 1024.
-	maxsignals	: specifies what is the maximum number of signals the simulator should handle.
		: value is an integer number between 0 and 1024.
-	maxtraces	: specifies what is the maximum number of traces the simulator should handle.
		: value is an integer number between 0 and 1024.
-	maxtools	: specifies what is the maximum number of tools the simulator should handle.
		: value is an integer number between 0 and 1024.

5.9. “+input file” command

This command attaches a file to a memory location (or two for stereo files); a memory location can be either a register or any piece of memory contained in the chip. A memory location is given by an identifier plus a location:

- the identifier can be chosen between the register and the memory ones of the “show” command. If not specified, the “In” type is used.
- the location is given when identifier is of memory type to know where to put the data.

The simulator supports the following file formats:

- .wav files: standard audio files on PC. A header describes the sound characteristics.
- .pcm files: header-less raw audio files.
- .dat files: floating-point ASCII files.
- .dec files: decimal integer ASCII files using S3.24 values i.e. loss-less format.
- .hex files: hexadecimal integer ASCII files using s3.24 values i.e. loss-less format.

Syntax : +input file FileName SamplingFrequency LeftOrMonoLocation [RightLocation]
Params : FileName gives the name of the file to use. The extension is used to know what is the file type.
: SamplingFrequency gives the sampling frequency to use for header-less file formats.
If set to 0, previously specified sampling frequency will be reused.
For .wav files and due to its header, it can be always safely set to 0.
: LeftOrMonoLocation (identifier and location) specifies where the left data should be put.
: RightLocation specifies where the right data should be put if the file is stereophonic.
Issues : The same sampling frequency should be used when specifying all inputs, outputs and tools.

For example, *+input file input.wav 44100 0 3* sets:

- sampling frequency to 44.1 kHz.
- data will be provided to the system at address $410 (position 0) for the left channel and at address $413 (position 3) for the right channel.
- The file which will be opened is named “input.wav”. It will be checked that this file contains stereophonic sound at 44.1kHz.

Or *+input file test.wav 0 Data 100 Scratch 5* sets:

- The file which will be opened is named “test.wav” and is expected to be a stereophonic file. It will be checked that this file contains stereophonic sound. The sampling frequency of this file will be used as sampling frequency of the system if not already set to another value.
- data will be provided to the system at address 100 of the data memory for the left channel and at address 5 of the scratch memory for the right channel.

5.10. “+output file” command

This command attaches a file to a memory location (or two for stereo files) ; a memory location can be either a register or any piece of memory contained in the chip. A memory location is given by an identifier plus a location:

- the identifier can be chosen between the register and the memory ones of the “show” command. If not specified, the “In” type is used.
- the location is given when identifier is of memory type to know where to put the data.

The simulator supports the same formats as for “+input file” command.

Syntax : +output file FileName SamplingFrequency LeftOrMonoLocation [RightLocation]
Params : FileName gives the name of the file to use. The extension is used to know what is the file type.
: SamplingFrequency gives the sampling frequency to use for file formats with headers.
If set to 0, previously specified sampling frequency will be reused.
: LeftOrMonoLocation (identifier and location) specifies where the left data should be taken from.
: RightLocation specifies where the right data should be taken from if the file is stereophonic.
Issues : The same sampling frequency should be used when specifying all inputs, outputs and tools.

For example, *+output file out.pcm 0 5* sets:

- sampling frequency to the one already used by the system.
- data will be will be read from the system at 0x415 (position 5).
- The file which will be written is named “out.pcm”. It will be checked that the sampling frequency has already been set in the system.

5.11. “+signal file” command

This command behaves exactly as the “+input file” one from the syntax, parameters and issues point of view. The only difference in this case is that the data are read before each instruction is executed.

5.12. “+trace file” command

This command behaves exactly as the “+output file” one from the syntax, parameters and issues point of view. The only difference in this case is that the data are read before each instruction is executed.

5.13. “+input gen” command

This command attaches a generator to a memory location; a memory location can be either a register or any piece of memory contained in the chip. A memory location is given by an identifier plus a location:

- the identifier can be chosen between the register and the memory ones of the “show” command. If not specified, the “In” type is used.
- the location is given when identifier is of memory type to know where to put the data.

The simulator supports the following generator types:

- *wnoise*: generate some white noise
- *sin*: generate a sine wave
- *tri*: generate a triangle wave
- *saw*: generate a saw tooth wave

Syntax	: +input file wnoise SamplingFrequency MonoLocation Amplitude [Offset]
	: +input file sin SamplingFrequency MonoLocation Frequency Amplitude [Offset] [Phase]
	: +input file tri SamplingFrequency MonoLocation Frequency Amplitude [Offset]
	: +input file saw SamplingFrequency MonoLocation Frequency Amplitude [Offset]
Params	: wnoise, sin, tri and saw specifies which generator to use
	: SamplingFrequency gives the sampling frequency to use. If set to 0, previously specified sampling frequency will be reused.
	: MonoLocation (identifier and location) specifies where the generated data should be put.
	: Amplitude specifies the peak amplitude of the noise to be generated.
	: Offset specifies an offset (DC component) to be added to the generated noise.
	: Frequency specifies the frequency (in Hz) of the sine, triangle or saw tooth wave to be generated
	: Phase specifies in radians the original phase when starting generating the sine wave.
Issues	: The same sampling frequency should be used when specifying all inputs, outputs and tools.
	: Possible formats for the amplitude and offset are as for the “*setval*” command.

For example, *+input gen sin 48000 mem 10 1000 0.25 0.25 3.14* sets:

- sampling frequency to 48 kHz.
- data will be provided to the system at address 10 of the memory.
- The sine wave will have a frequency of 1000Hz, an amplitude of 0.25 and a DC offset of 0.25. The sine wave will have an original phase of PI.

Or *+input gen wnoise 0 1 0.5* sets:

- The sampling frequency will be the one already used in the system.
- The generator will generate white noise at input 1 ($411 location) and will have an amplitude of 0.5.

5.14. “+signal gen” command

This command behaves exactly as the “+input gen” one from the syntax, parameters and issues point of view. The only difference in this case is that the data are read before each instruction is executed.

5.15. *“+tool AmpSweep” command*

The AmplitudeSweep tool is intended to produce a sweeping signal in amplitude. The signal generated is a sine wave of a given frequency going from an initial level to a final level by means of level increments.
The tools tracks the output of the system, measures the output amplitude for a given input amplitude and calculates the gain from input to output.
It is thus especially suited to measure the static transfer function of a compressor, distortion, ...

Syntax : +tool AmpSweep Sampling_Frequency Sine_Frequency Transient_Time Measurement_Time
Initial_Level Final_Level Level_Increment Input_Position Output_Position
Amplitude_Report_Filename Gain_Report_Filename

Params : *Sampling_Frequency* gives the sampling frequency (in Hz) to use.
If set to 0, previously specified sampling frequency will be reused.
: *Sine_Frequency* is the frequency (in Hz) of the sine wave generated by the tool typically 1000 Hz.
: *Transient_Time* is a time (in seconds) during which the system does not track the amplitude value.
It is meant to eliminate the transients due to the change in amplitude of the input sine wave.
: *Measurement_Time* is the time (in seconds) during which the system tracks the amplitude value.
This phase takes place after *Transient_Time* has elapsed.
: *Initial_Level* is the initial amplitude (in dB) of the sine wave.
: *Final_Level* is the final amplitude (in dB) of the sine wave.
Once this amplitude will be reached, simulation will stop.
: *Level_Increment* is the increment (in dB) by which the amplitude is increased every phase.
A phase is in that case a complete "*Transient_Time* + *Measurement_Time*" simulation.
: *Input_Position* is the position in which the tool should write its input (same as for +input file).
: *Output_Position* is the position in which the tool should read the output of the system.
: *Amplitude_Report_Filename* is the name of the report file containing the amplitude information.
The first column of this file is the input amplitude (in dB)
The second column of this file is the output amplitude (in dB).
: *Gain_Report_Filename* is the name of the report file containing the gain information.
The first column of this file is the input amplitude (in dB)
The second column of this file is the gain (in dB) between input and output.

Issues: The same sampling frequency should be used when specifying all inputs, outputs and tools.

For example, *+tool AmpSweep 48000 1000 0.020 0.500 -90 0 5 0 0 amplitude.txt gain.txt* sets:
- sampling frequency to 48 kHz, sine frequency to 1 kHz.
- transient time to 20 ms and measurement time to 500 ms.
- amplitude will go from -90 dB to 0 dB by steps of 5 dB.
- data will be provided to the system at adress 0x410 (position 0) and will be read from the system at 0x410 (position 0) as well.
- amplitude report will be written in the "amplitude.txt" file while gain report will be written in the "gain.txt" file.

The fact that data can be read/written from register or memory can be used to perform measurements on a part of a complete application only (i.e. when real IOs locations cannot be used).

5.16. “+tool FreqResp” command

The FrequencyResponse tool is intended to produce a white noise signal. The signal generated is expected to cover the full frequency range and that during a certain amount of time: the number of blocks. As the tool uses FFTs internally, a block size has to be specified corresponding to the size of the FFT. A given number of blocks will be generated first so that the system under test can reach the steady state.
The tool tracks the output of the system and measures the transfer function (amplitude and phase) of the system. It is thus especially suited to measure the frequency response of filters, ...

Syntax : +tool FreqResp Sampling_Frequency BlockSize NSkipBlocks NoiseLevel
Input_Position Output_Position Amplitude_Report_Filename Phase_Report_Filename
Params : *Sampling_Frequency* gives the sampling frequency (in Hz) to use.
If set to 0, previously specified sampling frequency will be reused.
: *BlockSize* is the size (power of 2 expected) on which the FFT will be computed.
This BlockSize also gives the number of output points which is equal to BlockSize / 2.
: NSkipBlocks is the number of blocks that should be processed before actually measuring.
It is meant to eliminate the transients in the system and allow it to run in steady state.
: *NoiseLevel* is the level (in dB) of the generated noise.
This parameter can help showing some limit cycles when set to a small value.
: *Input_Position* is the position in which the tool should write its input (same as for +input file).
: *Output_Position* is the position in which the tool should read the output of the system.
: *Amplitude_Report_Filename* is the name of the report file containing the amplitude information.
The first column of this file is the frequency (in Hz)
The second column of this file is the amplitude (in dB) between input and output.
: *Phase_Report_Filename* is the name of the report file containing the phase information.
The first column of this file is the frequency (in Hz)
The second column of this file is the phase (in rad) between input and output.
Issues : The same sampling frequency should be used when specifying all inputs, outputs and tools.

For example, *+tool FreqResp 48000 16384 5 -10 0 0 amplitude.txt phase.txt* sets:

- sampling frequency to 48 kHz.
- FFT size of 16384 (thus output of 8192 points), 5 blocks “skipped” before measuring.
- Average amplitude of the noise input set to –10 dB.
- data will be provided to the system at address 0x410 (position 0) and will be read from the system at 0x410 (position 0) as well.
- amplitude report will be written in the "amplitude.txt" file while phase report will be written in the "phase.txt" file.

The fact that data can be read/written from register or memory can be used to perform measurements on a part of a complete application only (i.e. when real IOs locations cannot be used).

WARNING: this tool is in a “beta release” state which means that bugs might still be found in the tool. Moreover, due to still unknown reasons, the output generated is quite “noisy” (contains peaks) even though sufficient time is spent to let the system go in steady state but also even though proper windowing is done. This problem is under investigation and a solution is expected (or should it be said ‘hoped’) in the near future.

6. Known problems, bugs and limitations

6.1. Multi-threading related issues

Since the simulation is running in a different thread than the command processing, it is possible to enter new commands while a simulation is running (except commands like start and step). It allows user to show values, set breakpoints, … and especially to stop this simulation.

1) If the display of disassembled lines has been turned on and if the simulation running has been launched with a step command and a pretty big number of steps, the screen will be continuously filled with disassembled lines. Please, note that even though the display will not reflect it (due to scrolling), commands can still be entered; in this case, the most interesting one being the stop command.
2) Due to the fact that it is pretty difficult to synchronize the different threads when it comes to write on the screen (and on the only console window available in the command-line version), the program sometimes displays twice the prompt i.e. “1KSM>”.
3) When starting a simulation with the start command, the simulator will display the “Simulation running” message. From that moment on, the simulation will run and when it will be finished (because end of file is reached or tool has finished its work), the “Simulation complete” message will be displayed.
4) Since the “go”-like commands are not blocking, the “exit” command should not be used in scripts. This will be solved in the future by implementing prioritized command queues and blocking “go”-like commands.

6.2. IO related issues

When coming to the IO topic, the simulator doesn’t check if an input has already been attached to an input device (being a file, a generator or a tool). This implies that the user has to take care that an input is not overwritten by another input.

The same situation exists for the outputs but in that case, it can be used as an interesting feature. For example, a tool like *AmpSweep* can generate a signal on a given input and takes its result from a given output; it is thus possible to write to an output file the same output as the one tracked by the tool to see what’s happening.

The reading/writing of .wav and .pcm files are limited to16-bits files only. As a matter of fact and for the sake of simplicity, it has been considered that using a 24-bits DSP with an 8-bits input wouldn’t make so much sense thus restricting the simulator to 16-bits files.