ValidSIM User's Guide

Steve Munich

Valid Logic Systems
Applications Engineering

ValidSIM is now available for production release. This applications note contains information about the basic philosophy of ValidSIM and how it is used in the SCALDsystem environment. Some of the differences between ValidSIM and previous versions of the logic simulator are also discussed, as well as some system configuration issues. The information contained in this applications note specifically concerns ValidSIM release 1.0.

WHAT DOES ValidSIM DO DIFFERENTLY?

The ValidSIM approach allows the SCALDsystem user to proceed directly to the simulator after entering or modifying a GED schematic. Assuming the set-up files are correct, the user types "simulate" (in UNIX or GED) or "simulate *drawingname*" (in unix) and the compiler will be called by the simulator to compile any newly written pages. Instead of writing a large expansion file for the "root drawing" out to disk (only to later read that file from the disk during simulation set-up) as was done in previous releases, ValidSIM calls a page compiler (Valid-PAGECOMP) which creates a compact and hierarchical expansion file for each page. The simulator links all the pages together as they are needed. The simulator is set up and commands are interpreted in the same way as in previous releases.

Time is saved not only because of a greatly reduced number of disk accesses, but also because only the changed pages of the design are compiled. The SCALDsystem user can expect to see a typical load time improvement of eight times, assuming some degree of hierarchy is used in the design. With a 68020 cpu board set, this will go to as high as a twenty-five times load speed improvement. Designs without hierarchy will have a turn-around time that is three to five times faster using a 68010 cpu, and ten to fifteen times faster with a 68020 board set.

ValidSIM also includes full-screen graphics waveforms, which will be described in greater detail, as well as a few other "virtual logic-analyzer" features. The greatly improved user interface, as well as the new and modified commands, are also time savers. For more complete information on the new commands, see the document called "Logic Simulator Software Changes Information" for ValidSIM 1.0. The new chapter 7 of the SCALDsystem Reference Manual also contains detailed information about ValidSIM.

ValidSIM COMPILATION ISSUES

Two types of compilers can be used by ValidSIM. One is the traditional "flat" compiler which was part of the 7.27 release. A new "page compiler" named ValidPAGECOMP is called by ValidSIM previous to setting up the simulator, and it only compiles changed pages of a design. The compiler directives for page compilation are the same as that for flat compilation, with the addition of two new ones. Linking is executed in every simulation session, and since the linker is really part of the simulator, its presence is transparent to the user.

Note that ValidPAGECOMP only supports compilation for use with ValidSIM at this time, and will support packaging and timing verification at a later date. For now, the "seplink" facility that was included as part of the 7.27 release is the most efficient method for compiling for packaging and timing verification.

PAGE COMPILATION

The files created by ValidPAGECOMP are different than previous compiler releases. The expansion file format now uses 60% less disk space than the 7.25 expansion format. Every page of every drawing compiled will have at least one expansion file. Some drawings will have more than one expansion file because they are used more than once in a given design, each time with different parameters. For example, a drawing called "multiplexor.sim" is used with size=2 and size=3 in a higher level drawing. So along with the ascii, connectivity, binary and dependency files in the drawing directory for "multiplexor", one would also find sim_exp.1.1.2 and sim_exp.1.1.3. These are the two expansion files for size=2 and size=3. There would also be a **schema** file in this directory, which is basically a list of all the contexts that a drawing has been compiled for (size=2 and size=3 are two contexts). If the drawing name was multiplexor.logic, the expansion file for page 1, size=3 would be called logic_exp.1.1.3.

The page compiler compares the time stamp of the schema file to that of the drawing's connectivity file in order to determine if the page has been changed and needs to be compiled before simulation. To do a full compilation of every page of the drawing, you can delete the schema file for each drawing in the design (rm */schema), but its best to "touch" the connectivity file of the particular drawing that you want to recompile, or write the drawing in GED. The absence of a schema file tells the compiler to generate a new compiler expansion file or set of expansion files for that page of the drawing. The old expansion files in the directory can also be deleted, but in any case they will be written over by the new ones. Deleting the schema file will cause all expansion files in the directory to be ignored by the simulator and then be regenerated by the compiler when the design using the specific context is processed by ValidSIM.

It is important that the page compiler executable (called pcomp, found in /u0/scald/compiler) is actually owned by root, otherwise the simulator will never be entered.

This will be shown as an "I/O RESULT ERROR". The permissions must also be -rwsr-xr-x on pcomp.

There are two additional directives that are only supported by the page compiler:

The first is called "page_synonym on/off;". If this directive is turned "on", the file cmpsyn.dat will contain synonyms listed page by page under "DRAWING FOO.LOGIC.1.1", for example, which contains the synonyms for page 1 of the drawing called FOO. The synonyms for page 2 would have the header "DRAWING FOO.LOGIC.1.2". If this directive is turned off, the synonyms will now be listed in a drawing by drawing fashion. For example, the synonyms for all pages of "FOO" will be contained under the header "DRAWING FOO.LOGIC". There is a small speed advantage if this directive is turned off. The default for this directive is on. The "page_synonym" directive has no meaning if synonyms are not being generated, and will be ignored.

The other new directive for "compile" is "default_filter on/off;". When turned "on", this directive filters out all properties except those specified with the "pass_property xxxx" directive. Conversely, if "default_filter off;" is declared, all properties will be passed except those specified with the "filter_property xxxx;" directive. This directive is default "off".

FLAT COMPILATION

For those times when flat compilation is needed, the 7.27 compiler generates flat expansion and synonym files. This can be accessed by typing compile [<root drawing> [<sim>]]. This method will completely recompile the drawing in one, large expansion file. It does not use the small expansion files in the drawing directories.

GATHERING ERROR LISTINGS

ValidSIM puts its error listing and statistical information in the same files as in previous simulator versions. These files are simlst.dat for simulation error listings, and simlog.dat for both error listing and and statistical information.

There are some changes with respect to compiler/linker error gathering. Whenever either ValidSIM (which calls the page compiler) or the "compile" facility (which is the "flat" compiler) is invoked, the compiler error descriptions are found in cmplst.dat, and statistical information relating to compilation and linking are found in cmplog.dat. Cmplog.dat contains errors that do not relate to a particular page, but instead are general compiler errors. These errors will appear on the screen.

The generation of cmplst.dat and cmplog.dat are normally transparent to the user. A program called "comperr" is called automatically to create cmplst.dat. This file will contain compiler and linker errors relating to a particular page. If a page or drawing in a design contains only warnings or oversights or both, no information concerning that module will be automatically placed in cmplst.dat. The "comperr" program reads compiler and linker listing files that are located in the drawing directories for each drawing. There are separate listing files for each page and each context, just as there are separate expansion files for each page and context. (An example of a listing file for a page compiled for simulation is found in the drawing directory as sim_lst.1.1; a linker listing file would be called sim_lli.1.1.1.) If removal of these listing files in the drawing directories is desired, it is best to do this after comperr has been run.

LIBRARIES

Library parts are treated the same way as any other drawing in the design in the sense that a schema file is generated for each part, as well as expansion files for each context of that part's use. There is no need for library permissions to change, and the user will not need to have write access to libraries. As before, the libraries must be owned by "lib" for all users to have access to them. The first time a library is compiled in a design, you will notice that the compiler calls up each previously uncompiled library element. This does not add much compile time since each context of each part usually takes no longer than a few seconds to compile, and also because the library parts are only compiled once. All libraries released after the production release of ValidSIM will include expansion files compiled for size = 1. Although the compiler expansion files generated by ValidPAGECOMP are usually no larger than 300 bytes and the listing file in the same directory is approximately 1 kbyte, it may be desirable to clean out these files on a periodic basis. It is advisable that the expansion files should only be removed from the library directory if unusual contexts have been compiled (size=5 is an example of an unusual context). Listing files are only created when a new context is compiled in a design, so it is best to remove these after the initial compile. Since both the listing and expansion files are owned by the library owner, the cleanup function will normally not be performed by the user but by the system manager.

If a particular cluster is used as a file server for libraries over the network, it is important that that node have the file "xefss" installed under /u0/scald/compiler. This executable is not normally found on SCALDsystems without ValidSIM, so it is necessary for it to be copied from any node that has ValidSIM. "Xefss" stands for "extended file system". This will allow the page compiler on a remote machine to write schema and expansion files in libraries residing on the file server.

ValidSIM SIMULATION ISSUES

ROOT DRAWING DIRECTIVE

Here is a typical directives file for use with ValidSIM:

```
root_drawing 'multiplexor';
use_synonym off;
terminal gcluster;
signame_chars 14;
clock_period 200;
clock_interval 10;
session_log on;
command_file 'newsim.scr';
output list, command_log;
end.
```

Note the new simulator directive called "root_drawing '*drawingname*';". The actual name of the drawing can be used as the argument ('multiplexor' in the example directives file above). This directive has the effect of telling the simulator not to look for a flat expansion file, as done in previous releases, but to check the schema file of the root drawing and invoke the compiler as previously described. The "root drawing" is normally the top level drawing of a hierarchical design, but it is possible to compile and simulate at any level of the design by merely changing the argument of this directive. If a schema file does not already exist, this directive causes it to be created. Leaving this directive out of simulate.cmd causes the simulator to look for the specified flat expansion file.

It is also possible to use this directive without specifying the drawing name in the directives file. By using 'name' as the argument of the root_drawing directive, ValidSIM will call the compiler and simulate the design whose name is specified with the command "simulate *drawingname*". The drawingname specified with this command will override the argument of the "root_drawing" directive in simulate.cmd.

SYNONYMS ON/OFF

The simulator directive "**use_synonym on/off;**" has an effect on performance with ValidSIM. This allows the simulator to load slightly faster because synonym information is not used if the "use_synonym off" directive is specified. If the directive is not included in simulate.cmd, the default is on. The greatest advantage of turning off the synonyms is in saving memory space so that more primitives can be loaded. For both load speed and memory utilization, performance improvement of synonyms on versus synonyms off is very much a function of hierarchy in the design. This is because deeply hierarchical designs can be very synonym intensive. The trade-off for the memory utilization and speed improvement is that only *base signal names* will be recognized by the simulator. These are the signal names that are referenced by the expansion file. Since the compiler chooses the most global signal as its base signal, this means that the signal name at the top of the hierarchy will be chosen over lower level interface signals connected to the same net, and that the lower level signal names are therefore not accessible with the synonyms turned off. For reasons of convenience, its usually desirable to leave synonyms turned on. For more information on how the compiler chooses base signal names, see chapter 4 of the Valid SCALDsystem Reference Manual.

Note that when synonyms are turned on, there is no "synonyms file" created. This is because the linker now finds the synonym information in the expansion file for the page. Due to the compact design of the expansion files, it is now impossible for the user to understand the synonym information from the expansion file or any other file unless he compiles flat with *output synonym;* as one of the compiler directives.

For more information on synonyms in hierarchy and the how this affects memory utilization, see the section discussing memory in SYSTEM CONFIGURATION ISSUES.

NEW "VIRTUAL LOGIC ANALYZER" (VLA) FEATURES

Another new directive is "*terminal gcluster;*", which allows a new graphics cluster terminal type. This allows a simulation to run from a UNIX shell with full-screen graphics waveforms. To access full screen graphics waveforms if the older "terminal cluster;" directive is used, just type "t gc" as a simulator command. This will redraw the screen of ascii waveforms with the same data in graphics form, and the simulator will proceed in graphics mode. To toggle back to ascii waveforms, type in "t cl". A total of 48 waveforms can be displayed in one full screen in the stand-alone mode, while 200 signals can be opened at any one time. There is no limit to the number of signals that can be viewed during the simulation, only that unused signals must be removed when the 200 signal limit is reached. A "bussed" signal is the equivalent of a single-bit signal as far as waveform display space is concerned.

There is also a new directive that allows a variation of the number of signalname characters displayed in the graphics cluster mode of simulation. The displaced characters are then replaced by more waveform information. The directive is "**signame_chars #;**", where "#" represents a single-digit number from 9 to 24. The display will default to the maximum (24) if a number greater than 24 is used. If the argument is less than 9 is specified in this directive, the display will default the minimum (9). The new "peek" command allows the truncated signalname to be viewed in the graphics cluster mode.

One of the new VLA features allows the time between two points in the waveform display area to be more easily calculated. Select the DELTA_TIME box from the simulator menu, then choose the two points in the display area with the puck. The elapsed time between the two points will be returned.

Another new feature of the graphics waveforms mode is the ability to use the puck to zoom in on the waveform display area. First select the "WAVEFORM" box in the menu, then select two points on the time axis. The expanded waveforms will be immediately displayed. Zooming out is accomplished in the same manner as before, by typing "wave <time> <time>". Right-to-left panning can be accomplished using the puck by selecting the "waveform" menu box, then selecting the beginning time with the puck on the waveform display, then selecting the ";" box at the bottom of the menu to terminate the command.

There are several more features have been added to ValidSIM that are described in the "Logic Simulator Software Changes Document" or in the new SCALDsystem Reference Manual Chapter 7. Here is a brief list of new or modified commands not discussed above:

assertions	- allows clock assertions to be added to or modified on signals without restarting or reinitializing ValidSIM.
deposit	- modified so that signals do not have to be first opened before a value can be deposited.
display	- allows the waveform display to be turned off, saving time because the screen need not be updated unnecessarily until the results must be viewed. Especially useful when using scripts.
hardcopy	- allows hardcopying of waveforms directly from ValidSIM. Also allows for a page size argument {A - E}.
history	- default history has been increased to 10000 ns.
move	- allows signals to be graphically moved using the puck.
peek	- allows the viewing of truncated signalnames, as selected by the puck. Also returns the value of the signal as well.

remove - modified so that signals can be removed from the display area using the puck, and need not be "opened" first.

set - has many added functions, such as the ability to choose from a variety of plotter options for waveform hardcopying.

simulate - has the added feature of allowing to simulate one clock period by typing "sim c" or selecting the "sim c" menu box with the puck.

SIMULATION WITH THE GRAPHICS EDITOR

When ValidSIM is run under GED, many of the graphics cluster features are accessible. The limitation here is the size of the simulation window, allowing only twelve signals to be viewed at any one time. The window size also limits the number of menu boxes that can be accessed. The major benefit of this mode is the ability to easily open signals during the initial simulation to build a script (found in simcmd.dat) to be reused later. Future releases of ValidSIM will allow other methods for initial circuit stimulation, so the limitations imposed by ValidSIM 1.0 running under GED are temporary.

When ValidSIM is invoked under GED, only messages concerning simulation set-up are easily readable. Compilation or linker error messages can be viewed only on a single line. It is then necessary to run "comperr" to determine what errors occurred, if any. For this reason it is recommended that simulation be run under GED only when the design is known to be bug free. A method for using the schematic to create a script file for input stimulus with the "stand-alone" graphics simulator is described in the next section.

COMMUNICATION BETWEEN GED AND THE STAND-ALONE SIMULATOR

In a GED window, to choose the signals that you would like to open in the simulator display, use the "change" command with the puck. "Control v" will immediately take you into the "vi" editor, and a file called "textedit##.tmp will be edited. "##" represents a window number, and will be different with each window. The file will look something like this:

```
!PROP SIG_NAME=foo1
!PROP SIG_NAME=foo2
!PROP SIG_NAME=foo3
!PROP SIG_NAME=foo4
```

You can easily delete the first 15 characters of each line and add "open " or "o " (thats open <space>) in front. You can even add ",line#" at the end of each line, if desired. Now write the file out(use :w! *name*) as some other name and use ":q!" to quit without changing the file called textedit##.tmp. Now you have a script that can be used immediately in the simulator.

A script called "tosim" can be used instead of manually creating the file in vi. "Tosim" can look like this:

```
ed $1<<END>/dev/null
1,\$s/!PROP SIG_NAME=/o /g
w
q
END
```

This can be executed after "control v" puts you into vi by typing ":!tosim". This will create a file that now looks like this:

```
o foo1
o foo2
o foo3
o foo4
```

The file must still be written out under another name, and vi must be quit with ":q!".

SYSTEM CONFIGURATION ISSUES

FUJITSU EAGLE / 68020 CPU BOARD

Using an S-32 with 140 Mbyte of hard disk, the typical load time improvement is eight times faster than the 7.25 compile-simulate cycle. With a Fujitsu Eagle, because of faster disk access times, the improvement is typically ten times. With the new 68020 CPU board, these numbers will be 20 times on a 140 Mbyte disk system, and 25 times with an Eagle.

MEMORY CONFIGURATION

In order to minimize page faults and decrease simulator load time, it is necessary to have the correct amount of system memory, especially now that ValidSIM does fewer disk accesses and more memory accesses. A simple formula will help to determine the minimum memory configuration to avoid page faults:

$$(\#\text{ of primitives})(.00040\text{ Mbytes}) + 1.5\text{ Mbytes} = \text{MEMORY}$$

MEMORY is defined to be the maximum memory size that would benefit load time. This assumes that no other processes are running in system memory, and that synonyms are turned off. For example, a design containing 6000 primitives needs a minimum system configuration of 4.5 Mbytes. The number of primitives for a given design can be found in the heap statistics table in the simlog.dat file, under "# of elements".

If synonyms are turned on, the following equation applies:

$$(\# \text{ of primitives})(.00045 \text{ Mbytes}) + 1.5 \text{ Mbytes} = \text{MEMORY}$$

If the design is flat with synonyms turned on, there will be no extra signal names for each signal, so no extra memory will be used.

The above formula only takes into account the physical memory needed for simulation, and not for any other processes. For each instance of the graphics editor, add 1 Megabyte. The operating system also takes up about 1 Megabyte. So for the 6000 primitive design mentioned previously, the system needs a minimum of 6.5 Megabytes to avoid page faults.