SCALD 7.4.2 LOGIC SIMULATOR INCREMENTAL CHANGES

GENERAL DESCRIPTION

The 7.4.2 release of the Simulator contains several new features and added capabilities from the previous 7.25 release. Highlights of this release are listed below and are described in more detail within this document:

- o Support for different radices in tabular I/O.
- o Separate rise/fall delays for all Simulator primitives.
- o Wire delay feedback.
- o Addition of a uni-directional MOS transistor primitive.
- o User-specified time resolution.
- o Simple coverage analysis to indicate which signals have made a transition.

Existing circuits, Simulator models, command files, etc. do not need to be changed. Incompatibilities between versions 7.25 and 7.4.2 should be reported as bugs unless otherwise described in this document.

DIFFERENT RADICES IN TABULAR I/O

In Release 7.25, users were limited to using binary for tabular I/O values. The Simulator has been modified to accept values in a radix other than binary. Now, these values may also be specified in octal, decimal, or hexadecimal using a command of the following format:

TRACE <signal name>, <radix>

The radix may be specified using numerals (2, 8, 10, or 16) or characters (b, o, d, or h).

If the TRACE command is given without a radix, the default trace radix is used. This default is initially 2, but some other value may be specified using the TRACE_RADIX directive. The format for this directive is:

```
TRACE_RADIX { 2 | 8 | 10 | 16 };
```

The default radix may also be changed at any time using the new TRACE_RADIX command:

```
TRACE_RADIX [ 2 | 8 | 10 | 16 | b | o | d | h ]
```

If no argument is given, the current default trace radix is displayed.

SEPARATE RISE/FALL DELAYS

Delays associated with Simulator primitives have been modified so that different times may be specified corresponding to a rise delay and a fall delay. Specification of these delays is made through the modified DELAY property or through the new properties, RISE and FALL.

The DELAY property has been modified to accept two values, a rise delay followed by a fall delay (separated by a comma). If only one value is specified, this value is used as both the rise and fall delay. Accordingly, delay can be specified in either of the following formats:

```
DELAY <delay time>
DELAY <rise delay>, <fall delay>
```

In addition, rise and fall delays can be specified using the RISE and FALL properties. Usage of these properties is as follows:

```
RISE <rise delay>
FALL <fall delay>
```

Note that both the DELAY property and the RISE and FALL properties cannot be specified on the same body or an error will result.

A new directive has been added to the Simulator to control the use of separate RISE/FALL delays. The format of this new directive is:

```
RISE_FALL { OFF | ON }
```

If the ON state is specified, simulations are performed using both the rise and fall delays specified for parts. The default state of this directive is OFF which causes all primitives to change states after the specified delay time (if only one value is given) or after the greater of the rise and fall delays.

When separate rise/fall delays are specified, the delay used for the various transitions is as follows (where X indicates any value):

old value output	new value	delay to use
X	0	fall
X	1	rise
X	U	min(rise, fall)
0	Z	rise
1	Z	fall
U	Z	max(rise, fall)

As a result of this new feature, changes were made to the set of functions provided for UCP's. The function GET_DELAY now returns the greater of the rise and fall delays. In addition, two new functions, GET_RISE and GET_FALL have been added to return the rise and fall delays of the primitive, respectively.

WIRE DELAY FEEDBACK

Wire delay feedback has been added to the Simulator. The wire delays can be fed back in either of two ways:

1. By using a directive of the form

```
WIRE_DELAYS 'filename';
```

2. By using a command of the form

```
WIRE_DELAYS filename [;]
```

The file must be in the format described below. Basically, each element consists of a signal name (in quotes), a bit subscript (if any), and a delay element or a list of path names of components that the signal drives with a delay for each bit. These delays are added in with any other specified delay values to determine when Simulator events are scheduled for those bits.

7.4.2 Simulator Changes

```
<delay file>          ::= END. |
                          <delay list> ; END.

<delay list>          ::= <signal delay list>; |
                          <signal delay list> ; <delay list>

<signal delay list>   ::= <signal name> := <stop delay list>

<stop delay list>     ::= <stop delay>; |
                          <stop delay>; <stop delay list>

<stop delay>          ::= = <quoted rise/fall range> |
                          <quoted path name> =
                             <quoted rise/fall range>

<signal name>         ::= <quoted signal name> |
                          <quoted signal name> < <bit range> >

<bit range>           ::= <bit number>  |
                          <bit number> .. <bit number>

<bit number>          ::= <integer>

<quoted rise/fall range>
                      ::= '<delay>' |
                          '<delay range>' |
                          '<rise delay range>,
                             <fall delay range>'

<rise delay range>    ::= <min delay> - <max delay>

<fall delay range>    ::= <min delay> - <max delay>

<min delay>           ::= <fixed point number>

<max delay>           ::= <fixed point number>

<delay range>         ::= <delay>, <delay> |
                          <delay> - <delay>

<delay>               ::= <fixed point number>
```

At present, the Simulator does not support the following:

```
<stop delay>        ::= <quoted path name> =
                            <quoted rise/fall range>
```

In other words, the delay specified for a signal is applied to all of its inputs. Note that if only <rise delay range> or only <fall delay range> is specified, the maximum delay is applied. The following is an example of a wire delay file:

```
'FOO' <5..0>: = '2.3, 3.4';

'BAR' <2> : = '3.7 - 4.8';

'FOO BAR' : = '5.1';

END.
```

PERFORMANCE ENHANCEMENTS IN MOS

The performance of NMOS simulation has been enhanced through the addition of a uni-directional MOS transistor primitive, UNI PASS TRANSISTOR. This addition not only increases the speed of simulation for MOS circuits, but improves the readability of drawings where fully bi-directional gates are not required. Pins and properties of the UNI PASS TRANSISTOR primitive are identical to those of the PASS TRANSISTOR primitive - a G pin controls whether the A and B pins are connected; however, since the transistor described is now uni-directional, the A pin is an input pin rather than an output pin.

The performance of MOS simulation has also been improved by making the decay time feature associated with MOS primitives default to "no decay". That is, unless the user explicitly specifies a decay time (using the DECAY_TIME directive or DECAY_TIME command), MOS signal strengths do not decay over time.

USER-SPECIFIED TIME RESOLUTION

A user can now specify the time resolution to be used by the Simulator through a new directive, RESOLUTION. This feature allows a user to specify a finer resolution when such capabilities are needed, or to increase the speed of simulations when a more coarse resolution is being used. The format of this directive is:

```
RESOLUTION <time>
```

7.4.2 Simulator Changes

The time resolution should be specified as a real number of nanoseconds and need not be a power of ten (e.g., 50 picoseconds is expressed as 0.05 and 2 microseconds is expressed as 2000). The default value is 1 nanosecond, the resolution previously used by the Simulator.

The current resolution used by the Simulator is indicated in the display area as a fixed point value labeled "Scale:". In BUS mode, this indication appears on the second line of the screen. In WAVEFORMS mode, the resolution appears near the bottom of the display area below the tick marks and time scale.

The addition of this feature affects the user interface in several areas. The most obvious of these is that the time scale indicated on the screen in WAVEFORMS mode no longer represents nanoseconds, but must be scaled by the indicated scale factor. For example, by specifying "RESOLUTION 20", each tick (formerly 1 ns) now represents 20 ns.

This feature also impacts time values entered into the Simulator or used by the Simulator. Some values are scaled based on the time resolution specified in this directive; these include the clock period, signal histories, signal delays, and decay times. These values are typically specified in nanoseconds, and this remains true; however, since the display may not be in nanoseconds, the values must be appropriately scaled by the Simulator. The following examples will help clarify time resolution. With the same scale factor of 20 used above:

1. A clock period of 500 ns divided into 10 intervals is displayed as "Clock: 25 / 10".

2. The default for signal history remains the same (1000 ns), but since each tick on the screen now represents 20 ns, history is only retained for 50 ticks.

3. Time values specified in input files (e.g., delay), do not have to be scaled by the user. The input units remain the same, but the values are scaled by the Simulator for display on the screen. For example, a 10000 ns decay time still is specified by "DECAY_TIME 10000", but signals change value after 500 ticks.

On the other hand, screen-oriented times maintain their relation to ticks on the screen (although the "real" times associated with those ticks have changed). For example, while "WAVEFORM 0 100" displays a time scale from 0 to 100 ticks, "SIM 100" advances simulated time by 100 ticks, and "CURSOR 25" sets the cursor to a location corresponding to 25 ticks. Still using the example with a scale factor of 20, the 25 and 100 ticks correspond to "real" times of 500 and 2000 ns, respectively.

he user should exercise caution when manipulating the time resolution. Too fine a resolution decreases the execution speed of the Simulator (simulating for hundreds of ticks even when no events are being scheduled) or could result in the generation of massive amounts of signal histories. On the other hand, before decreasing the resolution, one must ensure that the specification of other time values is correspondingly coarse (i.e., it probably does not make sense to specify "RESOLUTION 50" with a 20 ns clock interval).

NEW SIMULATOR DIRECTIVES

CLOCK_ON_DRIVEN Directive

In previous releases, if the clock property was specified for a signal, the Simulator built a clock generator for that signal even if it was driven by some other signal. The new directive:

```
CLOCK_ON_DRIVEN { OFF | ON };
```

has been added for building clock generators on driven signals. The default for the directive is OFF (which only permits timing assertions to be specified on undriven signals). Thus, building a clock generator on a driven signal is no longer allowed unless this directive is specified as ON.

USE_SYNONYM Directive

The USE_SYNONYM directive determines if the Simulator is required to read the Compiler's synonyms file (not reading the synonyms file speeds simulation time). The syntax for this directive is:

```
USE_SYNONYM { OFF | ON };
```

Note that if the directive USE_SYNONYM OFF; is included, signals only can be referenced by their "base names." The default for the directive is ON (the synonyms file always is read).

Other Directives

Several new directives have been described previously in this document. Below is a summary of these directives (see above for a more complete description):

TRACE_RADIX { 2 | 8 | 10 | 16 };	defines default trace radix (default: 2)
RISE_FALL { OFF | ON };	enables separate primitive rise/fall delays (default: OFF)
WIRE_DELAYS 'filename';	specifies file for wire delays
RESOLUTION <time>;	specifies Simulator time resolution (default: 1 ns)

NEW/MODIFIED COMMANDS IN THE SIMULATOR

TRACE Command

The TRACE command has been modified to take advantage of the puck when running the Simulator under GED. Signals to be traced may now be specified by pointing to them with the puck using the following command format:

TRACE <point> <point> ... ;

Thus, signals can easily be specified for tracing in the default radix without typing their signal name.

ASSERTIONS Command

The ASSERTIONS command is a new command that allows timing assertions to be specified while running the Simulator. This command allows the user to specify assertions interactively rather than with the signal name given when creating the drawing in GED. Addition of this feature provides the user with an extra degree of flexibility when performing simulations since signal timing assertions are no longer fixed with the signal name and need not be compiled with the drawing. Usage of this command is as follows:

ASSERTIONS < signal name >, < timing data >

The < timing data > parameter is specified using the standard SCALD syntax for timing assertion data (e.g., 0-4). The assertion type should not be specified - the Simulator automatically adds the "!C" property to the timing data.

This command can be invoked on existing clock signals as well as any other signals in the drawing. Thus, any signal can be assigned timing assertions while in the Simulator, and assertions of existing clock signals can be re-defined. After assigning clock properties, the signal can be OPENed using either its previous or its new (with assertions) name.

COVERAGE Command

Simple coverage analysis has been added to the Simulator to enable the user to obtain a list of the signals that have made a transition during a period of simulation. This list can then be used to ensure that all signals in the circuit have been exercised.

Coverage analysis is controlled by the COVERAGE command. The format of this command is:

COVERAGE [ON | OFF]

If no parameter is given, the current status of the coverage analysis is reported. If coverage analysis is off, the Simulator will not track the number of transitions.

At any time (whether coverage analysis is enabled or not), the user can output the list of signals that have made a transition and the number of transitions that they have made by using the WRITE_COVERAGE command. This command outputs the list to a file with the specified name. The format of this command is:

WRITE_COVERAGE < filename > [, { 0 | 1 | 2 | 3 }]

If no parameters are given, the user is prompted for a file name. If the optional parameter (0 - 3) is given, the signals are processed based on the number of times that they have made a transition. The signals are sorted by the number of transitions, and the file only contains those signal names in specific groups; for example, specifying "0" indicates that only signals making 0 transitions (i.e., those that have not changed) should be output, and "1" indicates that only those signals making 0 or 1 transitions are output.

To clear the list of signals that have made a transition, the user must invoke the INIT_COVERAGE command. This command, which has no parameters, enables the user to invoke coverage analysis for different periods of simulation. Note that turning coverage analysis OFF does not clear this list - this command must be invoked each time a new list of signals is to be started (except the first, when the list is empty), regardless of the use of the COVERAGE command.

RECORD_SIGNALS Command

The RECORD_SIGNALS command causes the signal histories of all signals in the circuit to be recorded. Previously, a signal had to be OPENed in WAVEFORMS mode in order to start a recording of its history. Thus, after a period of simulation, if a signal was not OPENed, there would be no method to determine what the value of a signal was at a previous time. By invoking this command, which takes no parameters, the history of all signals is available thereafter.

Note that this command does not affect the duration of history which is maintained for all signals - history only is preserved for the interval specified using the HISTORY command (or the default). Also note that since certain storage requirements are involved in creating and maintaining history, this command should not be invoked on large circuits.

RECORD_ALL Command

The RECORD_ALL command causes the signal histories of all signals and all memories in a circuit to be recorded. This command is identical to the RECORD_SIGNALS command described above except that the history of all locations of all memories also is recorded. This command requires no parameters and has no effect on the duration of history maintained for all signals.

Note that considerable storage requirements could be involved in creating and maintaining a history of all signals and memories. Thus, this command should not be invoked on circuits with a large number of elements and/or large memories.

SCROLL Command

The SCROLL command allows the user to control the automatic scrolling feature of the Simulator. The format of this command is:

```
SCROLL [ ON | OFF ]
```

The default is ON, which causes the Simulator display to scroll in WAVEFORMS mode when a signal not currently on the screen is OPENed. Using this command to turn the feature OFF allows the user to OPEN and DEPOSIT into signals that are not on the display.