All Cver System tasks and Functions

INTRO

These man pages document every system task and system function recognized by Cver including those not implemented. System tasks and functions are mixed together and appear in alphabetical order. Similar tasks and system functions are intermixed. Functionally similar tasks and functions are combined into one section and alphabetized according to main task or function.

For system functions, the SYNOPSIS section contains normal user function declaration notation to define return type and argument type even though system functions are never really declared. There is no concept of separate function header in Verilog. In the 2005 P1364 LRM most system function are indicated as returning integer, now that signed values are supported, system functions must be assigned to an integer or a "signed [31:0]" reg.

System tasks appear in the SYNOPSIS section using the task invocation form (i.e. task keyword omitted) and usually with meta descriptions of parameters. In Verilog system tasks, unlike user defined tasks, can always take a variable number of arguments and optional but not required arguments can be omitted either with ",," or by ending the argument list. Some system functions also have optional arguments that can be omitted.

In Verilog all strings except $display type format specifiers can be either expressions that are interpreted as strings (high 0 bits trimmed off and then interpret each byte as a character) or literal strings (enclosed by double quotation marks). Any [file name] can be either type of string.

The 2005 P1364 LRM timing checks are not documented here because they do not vary between simulators and are not really system functions. See 2005 P1364 LRM section 15 and A.7.5 for documentation of timing checks, edge control specifiers, and notifiers.

NAME

$bitstoreal − convert 64 bit register to real
$realtobits − convert real to 64 bit register
$itor − convert integer to real
$rtoi − convert real to integer by rounding

SYNOPSIS

```
function real $bitstoreal;
  input [63:0] bit_val;
function [63:0] $realtobits;
  input real real_val;
function real $itor;
  input integer int_val;
function integer $rtoi;
  input real real_val;
```

DESCRIPTION

System functions to convert to and from reals. Use $realtobits and $bitstoreal to pass reals across module ports. Use $itor and $rtoi to convert between integer and real by rounding. In Verilog, assignments also implicitly convert to or from a real depending on the left hand side Lvalue type. Cver conversion to real from wider than 32 bit values attempts to preserve as many bits as possible (usually 51 or 52 depending on hardware platform).

REFERENCE

Defined in 2005 P1364 LRM section 17.8.

NAME

$cleartrace – turn off statement tracing
$settrace – turn on statement tracing
$clearevtrace – turn off declarative event tracing
$setevtrace – turn on declarative event tracing
$tracefile – set separate output file for trace output

SYNOPSIS

$cleartrace;
$settrace;
$clearevtrace;
$setevtrace;
$tracefile([file name]);

DESCRIPTION

Cver supports separate control of statement and declarative event tracing. The options -t starts simulation with statement tracing on. The option minus -et start simulation with event tracing on. The -t option and $settrace and $cleartrace in some other simulators enable and disable both types of tracing. To approximate full tracing in other simulators use both -t and -et options or call both $settrace and $setevtrace.

The $tracefile system task set the output file for tracing (both types) to [file]. The $tracefile argument can be a variable that is treated as a string with high 0 bits removed. An alternative way to set a separate trace output file is with the +tracefile [file] command line option. Executing $tracefile replaces any +tracefile set file. If $tracefile is not used, trace output is written to STDOUT and the log file (if it exists). The tracefile can be the string "stdout" that has the effect of restoring trace output to default STDOUT and the log file. If a trace file is set, trace output is not written to stdout.

REFERENCE

These system tasks are not mentioned in the P1364 LRM because they apply to the interactive environment not addressed by the standard.

NAME

$cos – compute cosine of real input
$sin – compute sin of real input
$tan – compute tangent of real input
$acos – compute arc cosine of real input
$asin – compute arc sin of real input
$atan – compute arc tangent of real input
$acosh – compute hyperbolic arc cosine of real input
$asinh – compute hyperbolic arc sine of real input
$atanh – compute hyperbolic arc tangent of real input
$sgn – compute sign of real input (returns integer)
$int – convert real input to 32 bit integer (uses C not Verilog conversion)
$ln – compute nature logarithm of real input
$log10 – compute base 10 logarithm of real input
$abs – compute absolute value of real input
$pow – compute exponent of first real input to second real argument power
$sqrt – compute square root real input
$exp – compute e raised to power of real input
$min – compute minimum of 2 real inputs
$min – compute maximum of 2 real inputs

SYNOPSIS

System function identical to IEEE math functions except for added $ prefix

DESCRIPTION

This is non digital P1364 Cver enhancement that is available in digital Cver because it is required for Verilog-AMS version of Cver. Functions are defined in Verilog-AMS 2.0 OVI LRM. Behavior is identical to behavior documented by Unix math function man pages.

Cver also allows use of system functions with constant arguments in constant expressions so parameter definitions such as "parameter cos2 = $cos(2.0);" are legal in Cver and are part of new Verilog 2001 standard. Cver does not yet support constant argument user functions in constant expressions.

The following Hspice math library variants are also supported: $hsqrt, $hpow, $hpwr, $hlog, $hlog10 $hdb, and $hsign. See Hspice documentation for behavior and required arguments.

REFERENCE

See Verilog-AMS 2.0 OVI standard LRM. Standard IEEE math functions. See any Hspice documentation.

NAME

$countdrivers – return 1 if bus contention because more than one driver

SYNOPSIS

```
function integer $countdrivers;
  // must be wire
  input net;
  output integer net_is_forced;
  output integer number_of_01x_drivers;
  output integer number_of_0_drivers;
  output integer number_of_1_drivers;
  output integer number_of_x_drivers;
```

DESCRIPTION

System function return 0 if there is no more than one driver (all wire fan-in is tri-stated (in hiZ state) or a most one fan-in wire (or select of wire) is non tri-stated). Returns 1 if there is more than one non-tri-stated fan-in. The first net argument is required. If a second argument is given, it must be a reg lvalue expression that is set to the total number of fan-in any state other than hiZ. If a third argument is given, it must be a reg lvalue expression that is set to the total number of fan-in that drives 0. If a fourth argument is given, it must be a reg lvalue expression that is set to the total number of fan-in that drives 1. If a fifth argument is given, it must be a reg lvalue expression that is set to the total number of fan-in that drives x. If an argument is omitted, but an argument to its right is needed, use ",,". Notice a statement: "always @w numdriv = $countdrivers(w);" does not work since the number of drivers may change without the wire's value changing especially for strength wires. Notice that in the LRM, driver usually means non tri-stated fan-in. Sometimes driver means instead any fan-in whether or not it is tri-stated.

REFERENCE

Defined in 2005 P1364 LRM Appendix C.1

NAME

$display, $displayb, $displayh, $displayo – write formatted value to stdout
$write, $writeb, $writeh, $writeo – write formatted value to stdout

SYNOPSIS

```
$display([intermixed list of format strings and expressions]);
$display[bho]([intermixed list of format strings and expressions]);
$write([intermixed list of format strings and expressions]);
$write[bho]([intermixed list of format strings and expressions]);
```

DESCRIPTION

Write formatted values to standard out (and the log file). The format is similar to C language printf format except the format string must be literal, more than one format string can appear followed by values to replace format references with, only qualifier is 0 that causes trimming of the value to narrowest field possible. The %g format for reals is identical to C language printf %g format. $display always appends a new line to the end of the displayed string. For $write, any new line must be explicitly written. By default, variable that do not match any format specifier are written in decimal. Use $displayb and $writeb to change the default to binary, $displayh and $writeh to change default to hex, and $displayo and $writeo to change to octal.

REFERENCE

Defined in P1364 LRM section 17.1.

SEE ALSO

The same format is used by the $fdisplay and $fwrite tasks which write to file(s), the $monitor and $fmonitor tasks that write changed expressions at the end of a time slot, and $strobe and $fstrobe that always write value of an expression at the end of a time slot.

NAME

$dist_chi_square – return random 32 bit value in Chi Square distribution
$dist_erlang – return random 32 bit value in Erlangian distribution
$dist_exponential – return random 32 bit value in Exponential distribution
$dist_normal – return random 32 bit value in Standard Normal distribution
$dist_poisson – return random 32 bit value in Poisson distribution
$dist_t – return random 32 bit value in Standard T distribution
$dist_uniform – return random 32 bit value uniformly distributed in range

SYNOPSIS

```
function integer $dist_chi_square;
  inout integer seed;
  input integer degree_of_freedom;
function integer $dist_erlang;
  inout integer seed;
  input integer k_stage;
  input integer mean;
function integer $dist_exponential;
  inout integer seed;
  input integer mean;
function integer $dist_normal;
  inout integer seed;
  input integer mean;
  input integer standard_deviation;
function integer $dist_poisson;
  inout integer seed;
  input integer mean;
function integer $dist_t;
  inout integer seed;
  input integer degree_of_freedom;
function integer $dist_uniform;
  inout integer seed;
  input integer start;
  input integer end;
```

DESCRIPTION

These are the random deviate generation functions defined in P1364 LRM. Algorithm follow "Numerical Recipes in C". However they are somewhat unusual in taking integer arguments and producing scaled integer outputs rather than the more normal real number inputs and outputs between 0.0 (sometimes -1.0) and 1.0. Each distribution function takes a first argument that is a seed that sets cyclical starting point in the distribution sequence. Each function returns a random value according to the distribution type and updates the seed so that if a passed seed is saved and reused, both the same value will be returned, and the same new seed will be set. This properly allows repeatability for debugging.

The $dist_chi_square and $dist_t degree_of_freedom values determine the shape of the distribution (larger values widen the distribution). The mean parameter used in $dist_erlang, $dist_exponential, $dist_normal and $disp_poisson cause the average value returned to converge to the passed mean. The $dist_normal function standard deviation input parameter determines shape of the standard normal distribution (larger value widens the distribution). The $dist_uniform start and end determine the range within which the uniformly distributed random number fits. Start and end may be negative but start must be less than end. The mean, k_stage and degree_of_freedom parameters must be positive

REFERENCE

Defined 2005 in P1364 LRM section 17.9.2.

SEE ALSO

See any statistics textbook or CRC Handbook of Standard Math Tables for definitions of the formulas. Also see "Numerical Recipes in C" for algorithms.

NAME

$dumpvars – set up variables that are written to VCD file
$dumpall – dump current value of all variables set up for dumpvaring
$dumpflush – make OS call to flush VCD file buffer to file
$dumplimit – set maximum size of VCD file
$dumpoff – turn off dumping to VCD file on variable changes
$dumpon – turn on dumping to VCD file on variable changes

SYNOPSIS

$dumpvars([level], [list of scopes and/or variables to dump]);
$dumpall;
$dumpfile([file name]);
$dumpflush;
$dumplimit([limit number of bytes]);
$dumpoff;
$dumpon;

DESCRIPTION

The $dumpvars system task causes first a header to be written to the VCD (value change dump) file that defines a short code for each variables. Then whenever a variable changes, the variable's code and value are written to the VCD file. When $dumpvars is first called the value of all variables is written to the VCD file.

$dumpvars must be called before any writing of changed variable values to the VCD dump file can occur to set up the variables that are dumped. The $dumpfile system task can change the name of the VCD dump file from the default verilog.dump. It must be called before or at the same time as $dumpvars. When $dumpvars is called actual dumping is set up at the end of that time slot. It is an error for $dumpvars to be called more than once.

$dumpvars takes either scopes or variable names as arguments. A Variable name (or hierarchical reference) causes just that variable to be written to the VCD file. a Scope causes all variables in the scope and all variables in [level] scopes down to be written to the VCD file on change. [level] value 0 means descend to the bottom of the hierarchy. If $dumpvars is called with no arguments, all variables in the design are written to the VCD file.

$dump limits set the maximum size of the VCD dump file to [limit]. Once [limit] bytes have been written, no more writing to the dumpvars file occurs. $dumpoff stop dumping of variables and writes every dumped variable to the VCD dump file with unknown (x) value. $dumpon restarts dumping of variable changes and starts by writing the current value of every variable to the VCD dump file.

The +dumpvarsextended added Cver option writes a non standard VCD file but allows strengths (not just the value part) to be written and uses hex format instead of binary where possible to reduce the VCD file size.

REFERENCE

Defined in 2005 P1364 LRM section 18.

NAME

$fdisplay, $fdisplayb, $fdisplayh, $fdisplayo − write formatted value to file(s)
$fwrite, $fwriteb, $fwriteh, $fwriteo − write formatted value to file(s)

SYNOPSIS

$fdisplay([multi-channel or fd], [intermixed list of format strings and expressions]);
$fdisplay[bho]([multi-channel or fd], [intermixed list of format strings and expressions]);
$fwrite([multi-channel or fd], [intermixed list of format strings and expressions]);
$fdisplay[bho]([multi-channel or fd], [intermixed list of format strings and expressions]);

DESCRIPTION

Same as $display and $write but writes to either a Unix style file stream (abbreviated fd below) or all file descriptors selected by on bits in passed 32 bit multi-channel descriptor. For multi-channel descriptors if a bit is on but the file corresponding to the bit position is not opened with $fopen system function, no write occurs. The multi-channel descriptor idea allows one $fdisplay to write to more than one file in contrast to requiring multiple writes.

The modern Unix OS I/O stream form allows opening a file using the stream two argument form of $fopen and then using that strean descriptor (fd) for Unix OS style fwriting. The old file descriptor with bit 31 turned on is now used to indicate the new Unix OS style fd stream.

REFERENCE

Defined in 2005 P1364 2005 LRM section 17.2.2.

SEE ALSO

See related $display and $write system tasks. See $fopen and $fclose for opening and closing new Unix OS file streams and assigning multi-channel descriptors.

NAME

$finish − exit Cver

SYNOPSIS

$finish;
$finish([message level]);

DESCRIPTION

Exit Cver and return control to the host operating system. if a value is passed, if it is 0 (same as no argument) prints nothing, if 1 is passed prints normal exit message, and if 1 prints message as if +verbose option were selected.

REFERENCE

Defined in 2005 P1364 LRM section 17.4.1.

NAME

$flushlog – flush log and trace file internal OS buffers

SYNOPSIS

$flushlog;

DESCRIPTION

Flush the log file OS buffers. If the $tracefile system task has been used to set a separate trace output file, that stream also is flushed.

REFERENCE

Not defined in P1364 LRM but commonly implemented.

NAME

$fmonitor, $fmonitorb, $fmonitorh, $fmonitoro – write changed formatted value to file(s)

SYNOPSIS

$fmonitor([multi-channel or fd], [intermixed list of format strings and expressions]);
$fmonitor[bho]([multi-channel of fd], [intermixed list of format strings and expressions]);

DESCRIPTION

If any expression in the format variable list changes, format and write the value to the new Unix OS file descriptor (fd) or a multi-channel descriptor file list at the end of the simulation time slot. If a time returning system functions such as $time appears in the list, it does not cause a change. Format is same as $fdisplay. Any number of $fmonitors may be used and if more than one changed the format list for each changed $fmonitor is written at time slot end.

REFERENCE

Defined in new 2005 P1364 LRM section 17.2.2. Also 17.1.3 for $fmonitor.

SEE ALSO

$monitor is same but writes to stdout. See $display for format definitions.

NAME

$fopen – open a file and assign a Unix OS stream or multi-descriptor channel bit
$fclose – close a file and free for reuse a Unix OS stream or multi-descriptor

SYNOPSIS

function integer $fopen([file name]);
function integer $fopen([file name], [open description type]);
$fclose([multi-channel or fd]);

DESCRIPTION

Verilog allows writing to multiple open files at once using a multi-channel file descriptor. Bit 0 (least significant bit) is associated with stdout and is always open. The $fclose call closes the file associated with any on bit in the passed 32 bit multi-channel value. The file descript form of $fopen system function is passed a file name (can be any length and need not be a literal string) and returns a multi-channel descriptor with the next available bit (bit corresponding to un-opened descriptor channel) set. $fopen reuses multichannel bits freed by the file descriptor form of $fclose.

The new file stream form of fopen can be used to open a Unix OS style stream. For opening streams, the 2nd I/O type string must be present. The strings correspond exactly Unix $fopen I/O types (see 2005 LRM table 17-7). The return Unix OS stream has bit 31 turned on. Therefore bit 31 can no longer be used for old style file descriptors.

REFERENCE

Defined in 2005 P1364 LRM section 17.2.1.

SEE ALSO

See $fdisplay, $fwrite, $fmonitor, and $fstrobe routines that write to multi-channel descriptors or Unix OS streams.

NAME

$swrite[bho], $sformat - write formatted value to a string

SYNOPSIS

$swrite[bho]([ouput reg], [intermixed list of format strings and expressions]);
$sformat([ouput reg, [format string], [list of arguments]]);

DESCRIPTION

Routines to allow writing to Verilog regs. $swrite is the same as $display except the output is written to as a string to a Verilog reg. $sformat requires exactly one format string. The remaining arguments are interpreted as format values (never as format strings). This allows compile time checking of formats and is the same as the Unix OS sprintf type routine.

REFERENCE

Defined in 2005 P1364 LRM section 17.2.3.

NAME

$fgetc, $ungetc, $fgets, $fscanf, $sscanf, $fread, $fseek, $ftell, $rewind

SYNOPSIS

integer $fread([reg], [fd only]);
integer $fread([Verilog array], [fd only]);
integer $fread([Verilog array], [fd only], [start], [count]);
[other routines have same arguments and order as UNIX I/O library input routines]

DESCRIPTION

All the Unix I/O library input routine are implemented. Except for binary $fread that can read into arrays so it has extra arguments, all the read routines are the same as the Unix OS library routines in behavior and take the same arguments in the same order. See section 17.2.4.3 for a definition of the $fscanf and $sscanf legal format strings because formats are available for the Verilog two bit values and streng values (form v). Also unformatted binary data can be read in 2 bit chunks using the %z format and 1 bit values that are expanded to 4 value Verilog x/z values using the %u format.

Notice that only I/O streams (fds), not multi-channel descriptors can be used as the file argument for the file I/O read routines. Also, notice that the output value from these routines should be assigned to an integer, not an unsigned value. $fseek, $fell, $rewind and $ungetc correspond exactly to the Unix I/O library corresponding routines. All read routines return the number of items read or EOF (-1) when end-of-file is read. You must invoke $ferror to determine the cause of the error.

REFERENCE

Defined in 2005 P1364 LRM section 17.2.4.

NAME

$fflush - writes any buffer output for an open file or stream

SYNOPSIS

$flush([mcd]);
$flush([fd]);
$flush();

DESCRIPTION

These routine flush any buffered output for either an open multi-channel descriptor, a Unix OS file stream (fd) or all open multi-channel descriptor files and streams.

REFERENCE

Defined in 2005 P1364 LRM section 17.2.6.

NAME

$ferror - returns I/O error status

SYNOPSIS

integer $ferror([mcd or fd], str);

DESCRIPTION

This routine correspeonds to the Unix OS I/O library ferror routine. If an I/O error is detected, the OS error number is returned and the [str] is set to a string indicating the error reason. The reason is the same as the Unix OS returned I/O library error reason string.

REFERENCE

Defined in 2005 P1364 LRM section 17.2.7.

NAME

$getpattern − function for rapid assignment of memory bits to concatenate of scalars

SYNOPSIS

assign {<list of scalars or selects>]} = $getpattern(<memory>[<index>];

DESCRIPTION

This function must be used on the right hand side of a continuous assignment where the left hand side is a concatenate of scalars or constant bit selects. The argument must be a select of memory (normally loaded using $reaadmem) that is a variable. When the variable select index is changed the new memory value determined by the select index will be rapidly (i.e. with no need for expression evaluation) assigned to the scalars. Normally the assignment process will be driven by a for loop that increments the index. No other use of $getpattern is allowed.

REFERENCE

Defined in 2005 P1364 LRM Appendix C.1

NAME

$fstrobe, $fstrobeb, $fstrobeh, $fstrobeo − write formatted value to file at end of time slot

SYNOPSIS

$fstrobe([multi-channel or fd], [intermixed list of format strings and expressions]);
$fstrobe[bho]([multi-channel or fd], [intermixed list of format strings and expressions]);

DESCRIPTION

Same as $fdisplay but formats and writes value at the end of time time slot rather than when the $fstrobe statement is executed. Format is identical to $fdisplay and [bho] suffix letter changes default for expression that appears outside of any format as with $display. One formated string is written for every $fstrobe and $strobe executed during the time slot.

REFERENCE

Defined in 2005 P1364 LRM section 17.2.2.

SEE ALSO

$strobe is same except writes to stdout.

NAME

$history − print list previously executed interactive commands
$nokeepcommands − do not add executed interactive commands to history list
$keepcommands − add executed interactive commands to history list

SYNOPSIS

$history([number of commands to list]);
$keepcommands;
$nokeepcommands;

DESCRIPTION

$history lists either all or [number] of most recently executed interactive commands. All commands except history enable, disable, and one character abbreviation commands are entered in the history list. Added : debugger commands are also added to the history list.

Each command is numbered so it can be re-executed by entering [number] at the interactive prompt and, for scheduled and uncompleted commands, disabled by typing -[history command number]. $nokeepcommands disables collection of interactive commands into the history list and $keepcommands enables collection. Cver keeps all commands entered on the history list until a :emptyhistory added debugger command is entered at which point the history list is made empty. $input command scripts should begin with $nokeepcommands and end with $keepcommands to minimize history list size. The added debugger :history command is more flexible than $history. Multiple line commands (end with escaped new line in Cver) are printed as one command.

REFERENCE

Not defined in 2005 P1364 LRM. OVI LRM 1.0 section D.8.

SEE ALSO

See added debugger online ":help history" command for more detailed description of Cver's history mechanism.

NAME

$input − enter interactive commands from a file

SYNOPSIS

$input([file]);

DESCRIPTION

$input and the -i [file] Cver command argument cause interactive commands to be read from [file]. It can contain added debugger : commands. If an $input script file contains a $input call, command reading continues in the new script. The new script replaces the old and any un-executed interactive commands after the $input are not called. The new script is chained to not called. Interactive mode must be entered before commands can be read from the $input file so both -i and $input do nothing unless interactive mode is entered. $input should not appear in Verilog source.

REFERENCE

Defined in 2005 P1364 LRM section C.3.

SEE ALSO

See added debugger online ":help debugging" for additional documentation.

NAME (NOT SUPPORTED BY CVER)

$key − save every press key stroke to a file
$nokey − disable saving of key strokes

DESCRIPTION

Cver does not support $key because it depends on the original XL scheme for tracking asynchronous interrupts and is not compatible with Cver's added : debugger and line continuation scheme.

There are two better ways to allow restarting and simulating to a particular problem statement at a particular problem time. First, prepare a $input script and use -s with -i command options to rerun the script. Second, use the added :ibreak or :break breakpoint command with the :ignore [count] command to skip [count] break points to return to the problem time. Use :info breakpoint to determine the number of times a break point was hit. Alternative use the :breakpoint command :cond [expression] command to attach a condition to a statement break point.

REFERENCE

See 2005 P1364 LRM section C.4.

SEE ALSO

See added debugger :help online help system messages.

NAME

$list – list source reconstructed from internal data base for scope

SYNOPSIS

$list([scope]);

DESCRIPTION

List scope to stdout and the log file by reconstructing source from Cver's internal representation. All parameters and specparams are displayed as numeric constants. If no argument is given, list the current scope. If the -d command line option is used, reconstructed source for an entire design is output. If $list is executed from interactive mode, the current interactive scope (maybe set with the $scope system task is used). It is better from interactive mode to use the :list added debugger command that prints source lines exactly as they appear in the source input and allows more control of lines to list.

REFERENCE

See 2005 P1364 LRM section C.5.

SEE ALSO

Type ":help :list" in interactive mode for documentation of :list range specification.

NAME

$log – set new log output file or re-enable writing to log file
$nolog – disable writing to the log file

SYNOPSIS

$log;
$log([file name]);
$nolog;

DESCRIPTION

Normally all terminal (stdout) output is written to the log file that has name verilog.log and is over-written for each new run of Cver command option set the log file to [file]. Another way to disable writing to the log file is to use file name /dev/null on Unix and nul on OS2/DOS.

REFERENCE

See 2005 P1364 LRM section C.6.

NAME (CVER EXTENSION)

$memuse – print message giving dynamically allocated memory.

SYNOPSIS

$memuse;

DESCRIPTION

System task that can be called to output to stdout and the log file the number of bytes of dynamically allocated memory. Rather useless added system task since it is better to use the +verbose option.

REFERENCE

Cver extension not mentioned in the P1364 LRM.

NAME

$monitor, $monitorb, $monitorh, $monitoro – write changed formatted value to file(s)
$monitoroff- disable display of monitor changes
$monitoron- re-enable display of monitor changes

SYNOPSIS

$monitor([intermixed list of format strings and expressions]);
$monitor[bho]([intermixed list of format strings and expressions]);
$monitoron;
$monitoroff;

DESCRIPTION

If any expression in the format variable list changes, format and write the value to stdout and the log file at the end of the simulation time slot. Only one $monitor can be active at a time. Execution of a new monitor replaces the previous (see $fmonitor if multiple active monitors are needed, use multi-channel channel 0 to write to stdout and log file). If a time returning system functions such as $time appears in the list, it does not cause a change. Format is same as $fdisplay. $monitoroff turns off display of changed monitor values and $monitoron re-enables writing of changed formatted values.

REFERENCE

Defined in 2005 P1364 LRM section 17.1.3.

SEE ALSO

$fmonitor is same but writes to file using multi-channel descriptor, See $display for format documentation.

NAME

$q_add – place an entry on a queue
$q_exam – get selected queue status information
$q_initialize – create a new queue
$q_remove – get an entry from a queue
$q_full – return 1 if a queue if full else 0

SYNOPSIS

```
$q_add;
  input integer q_id; // unique number to identify queue
  input integer job_id; // unique number to identify job
  input integer inform_id; // user defined value added to queue
  output integer status; // completion status
$q_exam;
  input integer q_id; // unique number to identify queue
  input integer q_stat_code; // what to return in q_stat_value
  output integer q_stat_value;// returned value selected by q_stat_code
  output integer status; // completion status
```

$q_initialize
  input integer q_id; // unique number to identify queue
  input integer q_type; // type, 1=fifo, 2=lifo
  input max_length; // maximum number of elements allowed in queue
  output integer status; // completion status
$q_remove;
  input integer q_id; // unique number to identify queue
  input integer job_id; // unique number to identify job
  input integer inform_id; // user defined value removed from queue
  output integer status; // completion status
function integer $q_full;
  input integer q_id; // unique number to identify queue
  output integer status; // completion status

DESCRIPTION

In combination with stochastic random distribution (deviate) generators, these routines provide routines to model statistical queue for driving designs. Use the PLI routines for queues that must contain values more complicated that 32 bit integers (or regs).

for $q_exam, the possible request types are: 1=current queue length, 2=mean inter-arrival time, 3=maximum queue length, 4=shortest wait time ever, 5=longest wait time for jobs still in queue, and 6=average wait time in the queue. Any queue routine may set the status output parameter to: 1=OK, 2=queue full, cannot add, 3=undefined q_id, 4=unsupported queue type, cannot create queue, 5=specified length <=0, cannot create, 6=duplicate q_id, cannot create, not enough memory, cannot create.

REFERENCE

Defined in 2005 P1364 LRM section 17.6.

NAME

$random – generate signed random 32 bit value

SYNOPSIS

function integer $random;
  inout integer seed;
  integer seed;

DESCRIPTION

Cver uses good BSD random number generator that produces values with almost 32 pseudo random bits, but the sequence of generated number will probably not match the one returned by other simulators. If the the optional seed variable lvalue is given, the starting location in the 2**32 (almost) element sequence of pseudo random values is altered. Because the random generator only used 32 bit arithmetic the low bit is unrandom.

REFERENCE

See 2005 P1364 LRM section 17.9.1.

NAME

$readmemb – read binary number from memory stored in file
$readmemh – read hex number from memory stored in file
$sreadmemb – read binary number from memory stored in string
$sreadmemh – read hex number from memory stored in string

SYNOPSIS

$readmemb([file name], [memory name], [start_addr], [finish_addr]);
$readmemh([file name], [memory name], [start_addr], [finish_addr]);
$sreadmemb([memory name], [start_address], [finish_addr], [list of strings]);

sreadmemb([memory name], [start_address], [finish_addr], [list of strings]);

DESCRIPTION

These system tasks read values from either a file or a string (the $smemread[bh] routines). The format of the file or string, from which to the memory is filled, is a list of white space separated values. The values can contain digits, '_', x and z but no width specification. For $readmemb and $sreadmemb the values must be binary. For $readmemh and $sreadmemh, the values must be hex.

The basic memory filling algorithm is to read a word from the file or string, fill the current memory location, then increment the memory counter (decrement if [start_addr] is larger than [finish_addr]). The special value @[hexadecimal number] in the file or string changes the next address to write the memory data word into. If @[value] form changes to an address outside the range, memory filling stops. Memory address outside the range or outside the number of elements in the file or string are not changed. For $smemread routines, a list of strings is legal. The arguments may be arbitrary run time expressions that are converted to strings. The list of strings is concatenated into one long string and read exactly as if a file read of the string happened. The list of strings is required because Verilog does not allow strings to span line boundaries.

For $readmemb and $readmemh the [file] as a string (possibly an expression that is converted to a string) and the memory identifier are required. For $sreadmemb and $sreadmemh the memory identifier and at least one string are required. The [start_addr] and [finish_addr] are optional (must be indicated by ,, for $sreadmem functions) and give the first address in the memory to use to write the first data word from the file or string into. If the [finish_addr] is present, when that memory address is reached, filling of the memory is stopped. It is legal for [start_address] to be larger than [finish_addr] in which case the memory if filled from high to low word. If only [start_address] is given, [finish_addr] is the last (second) memory declaration range. If only [finish_addr] is given, [start_addr] is the start (first) memory declaration range.

REFERENCE

See 2005 P1364 LRM sections and 17.2.9 and C.13.

NAME

$reset − reset time to 0 and restart the simulation
$reset_value − returns value passed by most recent call of $reset
$reset_count − returns the number of times $reset has been executed

SYNOPSIS

$reset([stop_value], [reset_value], [diagnostics_value]);
function integer $reset_value;
function integer $reset_count;

DESCRIPTION

$reset allows rerunning a simulation from time 0 without re-translating a model. It can have up to 3 optional arguments. If [stop_value] is omitted or value 0, interactive mode is entered after reset. If [reset_value] is present, it is preserved across the reset and can be read with the the #reset_value system function after completing the reset. The optional [diagnostic value] argument determines the amount of diagnostic information printed after reset but prior to starting again at time 0. Value 0 causes no information to be emitted, 1 some information, and 2 is equivalent to the +verbose option. The $reset_count system function returns the number of times the $reset system task has been executed during the current run.

Cver also supports the :reset [option stop] added command debugger. It does not effect either the [reset_value] or the number of times $reset has been called so models that rely on $reset values can be debugged. In Cver, all : added debugger setting are preserved except break points and display expressions are disabled but not removed. Either type of reset removes all quasi-continuous forces and assigns. If a simulation is started with -s and -i [file], $reset will cause simulation to start over in interactive mode running the first command in [file]. Cver will never stop unless $reset_count is checked and used to cause end of simulation.

REFERENCE

See 2005 P1364 LRM section C.7.

SEE ALSO

See debugger online help for :reset added debugger command.

NAME (NOT IMPLEMENTED)

$save − save state of simulation to a file for later restart
$incsave − save only changed values from last $save to a file
$restart − restart simulation from a $save file

SYNOPSIS

$save([file]);
$incsave([file]);
$restart([file]);

DESCRIPTION

Cver does not yet support simulation check pointing that is needed for long simulations especially in cases where power or hardware failures occur. Cver design translation from source is fast enough that at least so far loading the binary data structure does not reduce load time. $incsave will probably not be supported since Cver already packs to the bit in order to support it, extra simulation event are needed.

REFERENCE

See 2005 P1364 LRM section C.8.

NAME

$scale − convert a time value from one module's time scale to another as real

SYNOPSIS

function real $scale;
  input [time hierarchical value as either real or reg];

DESCRIPTION

Given a time value as an hierarchical reference, convert to the time scale in which the $scale system task is executed. Usage: r = $scale(top.i1.i2.t1);

REFERENCE

See 2005 P1364 LRM section C.9.

NAME

$scope − change scope for use by interactive commands

SYNOPSIS

$scope([hierarchal name]);

DESCRIPTION

Use scope to change interactive scope from the default first top module for use after entering the interactive debugger. $scope is not very useful in Cver because, unless turned off by a debugger :set command, upon debugger entry (by $stop or interrupt) the interactive scope is set to the entering simulation scope. Also Cver supports an extended :scope command that allow relative movement between scopes and general reference for new scopes such as line numbers.

REFERENCE

See 2005 P1364 LRM section C.10.

SEE ALSO

See the added debugger ":help :scope" help screen.

NAME

$sdf_annotate – XL compatible sdf annotation system task

SYNPOSIS

$sdf_annotate([required name of sdf file], [optional annotation scope], [optional ignored - no config files yet], [optional ignored - no separate sdf log file], [optional mintypmax SDF override], [optional ignored - no scale factor], [optional ignored - no scale type]);

DESCRIPTION

System function alternative to +sdf_annotate command line option. Function identical to XL $sdf_annotate system task so Verilog source that in XL can be run without change in Cver. $sdf_annotate system task also allows conditional choice of sdf annotation file. Third optional configuration file argument is ignored because Configurations are not yet supported in Cver. Optional fourth argument name of separate log file is ignored because Cver writes all sdf messages to normal log file. Sdf annotation has also been changed to match XL so simulation continues even if SDF contains errors. As much annotation as possible is made if SDF contains errors. Fourth optional mintypmax override argument is supported. Legal values are one of MINIMUM, TYPICAL, or MAXIMUM. Optional sixth scale factor and seventh scale factor type arguments are ignored. Extra scaling of SDF values in Cver is not supported.

REFERENCE

De facto standardized routine. See document for any of the other Verilog simulators such as XL.

NAME

$showallinstances

SYNOPSIS

$showallinstances;

DESCRIPTION

For every module in design, print its instance and gate usage in tabular form. This system tasks prints at run time the instance design statistics table printed by the +printstats command argument.

REFERENCE

Not in P1364 LRM but commonly part of interactive environments.

NAME

$showscopes – display list of all scopes inside the current scope

SYNOPSIS

$showscopes([value]);

DESCRIPTION

List all scope objects in current scope. If invoked from interactive mode, the scope is the current interactive scope. If called during simulation, scope is current simulation scope. If value is present and non zero, print all scopes in or below the current scope to be output to stdout and the log file.

REFERENCE

See 2005 P1364 LRM section C.11.

NAME

$showvars − show information about variables
$showvariables − alternative name for $showvars task

SYNOPSIS

$showvars([optional list of variables]);

DESCRIPTION

Display information about variables. If no argument is given, display information about all variables in current scope. If a list of variables is given display information about each variable. Hierarchical references are allowed. Cver's added interactive debugger supports additional commands for examining variable values and information. Use the :help data debugger command for more information.

REFERENCE

See 2005 P1364 LRM section C.12.

SEE ALSO

See :print, :whatis, :expris, :varis added debugger commands.

NAME (CVER EXTENSION)

$snapshot − display active procedural thread tree and pending events

SYNOPSIS

$snapshot;

DESCRIPTION

Mechanism to print a snapshot of procedural location, pending events and thread execution status. If interactive debugger is disabled interrupt (ˆc) causes $snapshot to be called. Most information also generated by :where added debugger command. If you think some tasks or initial/always blocks should be active but they are not, or you think they should have completed but they have not, put $snapshot in your source or invoke from interactive mode to see the procedural active tree.

REFERENCE

Not in P1364 LRM.

NAME

$stop − enter interactive debugger

SYNOPSIS

$stop;
$stop([message level]);

DESCRIPTION

Enter interactive debugger and if [message level] is 1 or 2, print a message. 1 prints simulation time and 2 prints +verbose simulation statistics. Interactive debugger can also be entered by pressing interrupt key (usually ˆc) or from the -s option.

REFERENCE

Defined in 2005 P1364 LRM section 17.4.

NAME

$strobe, $strobeb, $strobeh, $strobeo − write formatted value to terminal at end of time slot

SYNOPSIS

$strobe([intermixed list of format strings and expressions]);
$strobe[bho]([intermixed list of format strings and expressions]);

DESCRIPTION

Same as $display but formats and writes value at the end of time time slot rather than when the $strobe statement is executed. Format is identical to $display and [bho] suffix letter changes default for expression that appears outside of any format as with $display. One format is written to stdout and log file for every $strobe executed during the time slot.

REFERENCE

Defined in 2005 P1364 LRM section 17.1.2 and 17.2.2.

SEE ALSO

$fstrobe is same except writes to multi-channel file(s).

NAME

$suppress_warns
$allow_warns

SYNOPSIS

$suppress_warns([comma separated list of warning or inform numbers]);
$allow_warns([comma separated list of warning or inform numbers]);

DESCRIPTION

It is possible to suppress writing of run time warning and inform messages by calling the $suppress_warns system task with a list of numbers that are printed when a warning or inform message is printed. Messages of class ERROR and FATAL ERROR can not be suppressed. The +suppress_warns+[list of + separated numbers] command line option also suppresses printing of messages and is the only way to suppers. translation time messages. $allow_warns re-enables printing of run time messages.

REFERENCE

These system tasks are a Cver extension.

SEE ALSO

See +suppress_warns+[num]+[num]+... option.

NAME

$system − execute operating system command from within Cver

SYNOPSIS

$system([OS command line]);

DESCRIPTION

Execute some operating system command string by means of a shell escape. $system; with no arguments or an empty argument runs an interactive shell if one is supported for the system you are running Cver on. The semantics of this task is slightly different on Unix based and non Unix systems. If you are running with multiple shell windows, it is better to execute commands in another window because a core dump will probably also cause Cver to core dump.

REFERENCE

Not defined in P1364 LRM but commonly implemented.

NAME

$test$plusargs − test Cver for existence of command argument
$scan$plusargs − scan Cver command arguments to match prefix

SYNOPSIS

function integer $test$plusargs([string]);
function $scan$plusargs([plus option prefix as string], [string lvalue]);

DESCRIPTION

$test$plusargs returns 1 if the argument appeared in the Cver command invocation argument list. The string expression argument must not include the leading + and must match an entire argument. Only Cver command arguments that begin with + are checked for a match.

The $scan$plusargs system function is equivalent to PLI tf_ mc_scan_plusargs function. The first argument is plus option (without +) prefix as string expression. Second argument is lvalue expression that is assigned the remainder of the string to the right of the matched prefix. Returns 1 if match and assign tail, 0 if no match and no assign. In -f files, tokenization is by white space only so file=xx is legal, but depending on your shell the option may need to be quoted if given on the command line. Standard $test$plusargs supported and return 1 if entire string matches (no leading +) else 0.

REFERENCE

Not defined in P1364 LRM. $test$plusargs is commonly implemented. $scan$plusargs allows use of PLI tf_ mc_scan_plusargs without linking in the PLI.

NAME

$time − return time scaled to module as 64 bit time value
$realtime − return time scaled scaled to module as real
$stime − return time scaled to module as 32 bit value
$tickstime − return time in internal simulation ticks as 64 bit value
$stickstime − return time in internal simulation ticks as 32 bit value

SYNOPSIS

function time $time;
function real $realtime;
function [31:0] $stime;
function time $tickstime;
function [31:0] $stickstime;

DESCRIPTION

$time returns current time as 64 bit unsigned time value. It is scaled to time units of the module the $time task is invoked from. $realtime is the same as $time except the value returned is real and the $timeformat scale values are used to determine real value fraction accuracy. $stime is the same as $time except value is truncated to 32 bits. $tickstime returns current time in internal simulation ticks (smallest module time scale in design) in a 64 bit value. $stickstime is the same as $tickstime except it is truncated to fit in 32 bits. Simulation ticks are the minimum time scale unit in any module and is the value used internally during simulation.

REFERENCE

Defined in 2005 P1364 LRM section 17.7. $tickstime and $stickstime are Cver extensions.

NAME

$printtimescale − display time unit and precision for a module
$timeformat − specify format for the %t $display format specifier

SYNOPSIS

$printtimescale([hierarchical_name]);
$timeformat([units_number], [precision number], [suffix_string], [minimum_field_width]);

DESCRIPTION

The $printtimescale system task displays time unit and precision for a particular module. If the argument is omitted, the information for the current module is printed.

The $timeformat system task sets the format for the %t format specifier.

REFERENCE

See 2005 P1364 LRM section 17.3 and the ‘timescale directive discussion section 19.8.

TRADEMARKS AND COPYRIGHT

Verilog is a Trademark of Cadence Design Systems Licensed to Open Verilog International.
Cver, CVC and Vcmp are Trademarks of Pragmatic C Software Corporation.