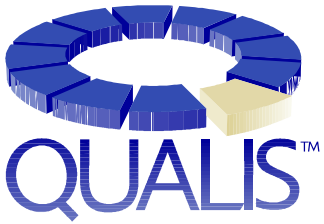

Verilog HDL QUICK REFERENCE CARD

Revision 2.1

()	Grouping	[ ]	Optional
{}	Repeated	|	Alternative
bold	As is	CAPS	User Identifier

1. MODULE

module MODID[({PORTID,})];
  [**input** | **output** | **inout** [range] {PORTID,};]
  [{declaration}]
  [{parallel_statement}]
  [specify_block]
endmodule

range ::= **[**constexpr **:** constexpr**]**

2. DECLARATIONS

parameter {PARID **=** constexpr,};

wire | **wand** | **wor** [range] {WIRID,};

reg [range] {REGID [range],};

integer {INTID [range],};

time {TIMID [range],};

real {REALID,};

realtime {REALTIMID,};

event {EVTID,};

task TASKID;
  [{**input** | **output** | **inout** [range] {ARGID,};}]
  [{declaration}]
begin
  [{sequential_statement}]
end
endtask

function [range] FCTID;
  {**input** [range] {ARGID,};}
  [{declaration}]
begin
  [{sequential_statement}]
end
endfunction

3. PARALLEL STATEMENTS

assign [**(**strength**1,** strength**0)**] WIRID **=** expr;

initial sequential_statement

always sequential_statement

MODID [**#(**{expr,}**)**] INSTID
  ([{expr,} | {.PORTID**(**expr**),**}**]);**

GATEID [**(**strength**1,** strength**0)**] [**#**delay]
    [INSTID] **(**{expr,}**);**

defparam {HIERID **=** constexpr,};

strength ::= **supply** | **strong** | **pull** | **weak** | **highz**

delay ::= number | PARID | **(** expr [**,** expr [**,** expr]] **)**

4. GATE PRIMITIVES

and (out, in_1, ..., in_N**);** **nand (**out, in_1, ..., in_N**);**
or (out, in_1, ..., in_N**);** **nor (**out, in_1, ..., in_N**);**
xor (out, in_1, ..., in_N**);** **xnor (**out, in_1, ..., in_N**);**

buf (out_1, ..., out_N, in**);** **not (**out_1, ..., out_N, in**);**

bufif0 (out, in, ctl**);** **bufif1 (**out, in, ctl**);**
notif0 (out, in, ctl**);** **notif1 (**out, in, ctl**);**

pullup (out**);** **pulldown (**out**);**

[**r**]**pmos (**out, in, ctl**);**
[**r**]**nmos (**out, in, ctl**);**
[**r**]**cmos (**out, in, nctl, pctl**);**

[**r**]**tran (**inout, inout**);**
[**r**]**tranif1 (**inout, inout, ctl**);**
[**r**]**tranif0 (**inout, inout, ctl**);**

5. SEQUENTIAL STATEMENTS

;

begin[**:** BLKID
  [{declaration}]]
  [{sequential_statement}]
end

if (expr**)** sequential_statement
[**else** sequential_statement]

case | **casex** | **casez (**expr**)**
  [{{expr,}**:** sequential_statement}]
  [**default:** sequential_statement]
endcase

forever sequential_statement

repeat (expr**)** sequential_statement

while (expr**)** sequential_statement

for (lvalue **=** expr**;** expr**;** lvalue **=** expr**)**
  sequential_statement

#(number | **(**expr**)**) sequential_statement

@ (event [{**or** event}]**)** sequential_statement

lvalue [**<**]**=** [**#**(number | **(**expr**)**)] expr**;**

lvalue [**<**]**=** [**@ (**event [{**or** event}]**)**] expr**;**

wait (expr**)** sequential_statement

-> EVENTID**;**

fork[**:** BLKID
  [{declaration}]]
  [{sequential_statement}]
join

TASKID[**(**{expr,}**)**]**;**

disable BLKID | TASKID**;**

assign lvalue **=** expr**;**

deassign lvalue**;**

lvalue ::=
  ID[range] | ID**[**expr**]** | **{{**lvalue,**}}**

event ::= [**posedge** | **negedge**] expr

6. SPECIFY BLOCK

specify_block ::= **specify**
    {specify_statement}
  endspecify

6.1. SPECIFY BLOCK STATEMENTS

specparam {ID **=** constexpr,};

(terminal **=>** terminal**) =** path_delay;

((terminal,} ***>** {terminal,}**) =** path_delay;

if (expr**) (**terminal [**+**|**-**]**=>** terminal**) =** path_delay;

if (expr**) (**{terminal,} [**+**|**-**]***>** {terminal,}**) =**
    path_delay;

[**if (**expr**)**] **(**[**posedge**|**negedge**]** terminal **=>**
    (terminal [**+**|**-**]**:** expr**)) =** path_delay;

[**if (**expr**)**] **(**[**posedge**|**negedge**]** terminal ***>**
    ({terminal,} [**+**|**-**]**:** expr**)) =** path_delay;

$setup(tevent, tevent, expr [, ID]**);**

$hold(tevent, tevent, expr [, ID]**);**

$setuphold(tevent, tevent, expr, expr [, ID]**);**

$period(tevent, expr [, ID]**);**

$width(tevent, expr, constexpr [, ID]**);**

$skew(tevent, tevent, expr [, ID]**);**

$recovery(tevent, tevent, expr [, ID]**);**

tevent ::= [**posedge** | **negedge**] terminal
    [**&&&** scalar_expr]

path_delay ::=
  expr | **(**expr, expr [, expr [, expr, expr, expr]]**)**

terminal ::= ID[range] | ID**[**expr**]**

© 1995-2000 Qualis Design Corporation

7. EXPRESSIONS

primary

unop primary

expr binop expr

expr **?** expr **:** expr

primary ::=
literal | lvalue | FCTID**(**{expr,}**)** | **(** expr **)**

7.1. UNARY OPERATORS

+, **-**	Positive, Negative
!	Logical negation
~	Bitwise negation
&, **~&**	Bitwise and, nand
|, **~|**	Bitwise or, nor
^, **~^**, **^~**	Bitwise xor, xnor

7.2. BINARY OPERATORS

Increasing precedence:

?:	if/else
||	Logical or
&&	Logical and
|	Bitwise or
^, **^~**	Bitwise xor, xnor
&	Bitwise and
==, **!=** , **===**, **!==**	Equality
<, **<=**, **>**, **>=**	Inequality
<<, **>>**	Logical shift
+, **-**	Addition, Subtraction
*, /, %	Multiply, Divide, Modulo

7.3. SIZES OF EXPRESSIONS

unsized constant		32
sized constant		as specified
i *op* j	+,-,*,/,%,&,|,^,^~	max(L(i), L(j))
op i	+, -, ~	L(i)
i *op* j	===, !==, ==, !=	
	&&, ||, >, >=, <, <=	1
op i	&, ~&, |, ~|, ^, ~^	1
i *op* j	>>, <<	L(i)
i ? j : k		max(L(j), L(k))
{i,...,j}		L(i) + ... + L(j)
{i{j,...k}}		i * (L(j)+...+L(k))
i = j		L(i)

8. SYSTEM TASKS

* indicates tasks not part of the IEEE standard but mentioned in the informative appendix.

8.1. INPUT

$readmemb("fname"**,** ID [**,** startadd [**,** stopadd]]**);**
$readmemh("fname"**,** ID [**,** startadd [**,** stopadd]]**);**
***$sreadmemb(**ID**,** startadd**,** stopadd {**,** string}**);**
***$sreadmemh(**ID**,** startadd**,** stopadd {**,** string}**);**

8.2. OUTPUT

$display[defbase]**(**[fmtstr,] {expr,}**);**
$write[defbase] **(**[fmtstr,] {expr,}**);**
$strobe[defbase] **(**[fmtstr,] {expr,}**);**
$monitor[defbase] **(**[fmtstr,] {expr,}**);**
$fdisplay[defbase] **(**fileno**,** [fmtstr,] {expr,}**);**
$fwrite[defbase] **(**fileno**,** [fmtstr,] {expr,}**);**
$fstrobe(fileno**,** [fmtstr,] {expr,}**);**
$fmonitor(fileno**,** [fmtstr,] {expr,}**);**
fileno = **$fopen(**"filename"**);**
$fclose(fileno**);**

defbase ::= **h** | **b** | **o**

8.3. TIME

$time	"now" as TIME
$stime	"now" as INTEGER
$realtime	"now" as REAL
$scale(hierid**)**	Scale "foreign" time value
$printtimescale[**(**path**)**]	Display time unit & precision
$timeformat(unit#**,** prec#**,** "unit"**,** minwidth**)**	Set time %t display format

8.4. SIMULATION CONTROL

$stop	Interrupt
$finish	Terminate
***$save(**"fn"**)**	Save current simulation
***$incsave(**"fn"**)**	Delta-save since last save
***$restart(**"fn"**)**	Restart with saved simulation
***$input(**"fn"**)**	Read commands from file
***$log**[**(**"fn"**)**]	Enable output logging to file
***$nolog**	Disable output logging
***$key**[**(**"fn"**)**]	Enable input logging to file
***$nokey**	Disable input logging
***$scope(**hiername**)**	Set scope to hierarchy
***$showscopes**	Scopes at current scope
***$showscopes(1)**	All scopes at & below scope
***$showvars**	Info on all variables in scope
***$showvars(**ID**)**	Info on specified variable
***$countdrivers(**net**)**>1	driver predicate
***$list**[**(**ID**)**]	List source of [named] block
$monitoron	Enable $monitor task
$monitoroff	Disable $monitor task
$dumpon	Enable val change dumping
$dumpoff	Disable val change dumping
$dumpfile("fn"**)**	Name of dump file
$dumplimit(size**)**	Max size of dump file
$dumpflush	Flush dump file buffer
$dumpvars(levels [**{,** MODID | VARID}]**)**	Variables to dump
$dumpall	Force a dump now
***$reset**[**(0)**]	Reset simulation to time 0
***$reset(1)**	Reset and run again
***$reset(0|1,** expr**)**	Reset with reset_value
***$reset_value**	Reset_value of last $reset
***$reset_count**	# of times $reset was used

8.5. MISCELLANEOUS

$random[**(**ID**)**]

***$getpattern(**mem**)**	Assign mem content
$rtoi(expr**)**	Convert real to integer
$itor(expr**)**	Convert integer to real
$realtobits(expr**)**	Convert real to 64-bit vector
$bitstoreal(expr**)**	Convert 64-bit vector to real

8.6. ESCAPE SEQUENCES IN FORMAT STRINGS

\n, **\t**, ****, **\"**	newline, TAB, '\', '"'
\xxx	character as octal value
%%	character '%'
%[w.d]**e**, **%**[w.d]**E**	display real in scientific form
%[w.d]**f**, **%**[w.d]**F**	display real in decimal form
%[w.d]**g**, **%**[w.d]**G**	display real in shortest form
%[0]h, **%[0]H**	display in hexadecimal
%[0]d, **%[0]D**	display in decimal
%[0]o, **%[0]O**	display in octal
%[0]b, **%[0]B**	display in binary
%[0]c, **%[0]C**	display as ASCII character
%[0]v, **%[0]V**	display net signal strength
%[0]s, **%[0]S**	display as string
%[0]t, %[**0**]**T**	display in current time format
%[0]m, **%[0]M**	display hierarchical name

9. LEXICAL ELEMENTS

hierarchical identifier ::= {INSTID **.**} identifier

identifier ::= letter | **_** { alphanumeric | **$** | **_**}

escaped identifer ::= \ {nonwhite}

decimal literal ::=
[**+**|**-**]integer [**.** integer] [**E**|**e**[**+**|**-**] integer]

based literal ::= integer **'** base {hexdigit | **x** | **z**}

base ::= **b** | **o** | **d** | **h**

comment ::= **//** comment newline

comment block ::= **/*** comment ***/**

© 1995-2000 Qualis Design Corporation

Qualis Design Corporation
Elite Training / Consulting in Reuse and Methodology

Phone: +1-503-670-7200 FAX: +1-503-670-0809
E-mail: info@qualis.com com Web: http://www.qualis.com