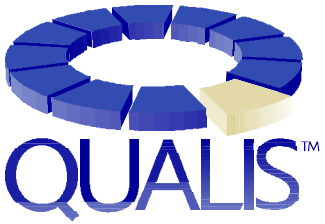

VHDL QUICK REFERENCE CARD

Revision 2.1

()	Grouping	[ ]	Optional
{}	Repeated	|	Alternative
bold	As is	CAPS	User Identifier
italic	VHDL-1993		

1. Library Units

[{use_clause}]
entity ID **is**
  [**generic (**{ID **:** TYPEID [**:=** expr];}**);**]
  [**port (**{ID **: in | out | inout** TYPEID [**:=** expr];}**);**]
  [{declaration}]
[**begin**
  {parallel_statement}]
end [***entity***] ENTITYID**;**

[{use_clause}]
architecture ID **of** ENTITYID **is**
  [{declaration}]
begin
  [{parallel_statement}]
end [***architecture***] ARCHID**;**

[{use_clause}]
package ID **is**
  [{declaration}]
end [***package***] PACKID**;**

[{use_clause}]
package body ID **is**
  [{declaration}]
end [***package body***] PACKID**;**

[{use_clause}]
configuration ID **of** ENTITYID **is**
for ARCHID
  [{block_config | comp_config}]
end for;
end [***configuration***] CONFID**;**

use_clause::=
  library ID**;**
  [{**use** LIBID**.**PKGID[**. all** | DECLID];}]

block_config::=
  for LABELID
    [{block_config | comp_config}]
  end for;

comp_config::=
  for all | LABELID **:** COMPID
    (**use entity** [LIBID.]ENTITYID [**(** ARCHID **)**]
      [[**generic map (** {GENID **=>** expr ,} **)**]
       port map ({PORTID **=>** SIGID | *expr* ,}**)**];
    [**for** ARCHID
      [{block_config | comp_config}]
    end for;]
    end for;) |
    (**use configuration** [LIBID.]CONFID
      [[**generic map (**{GENID **=>** expr ,}**)**]
      port map ({PORTID **=>** SIGID | *expr*,}**)**];)
  end for;

2. Declarations

2.1. Type declarations

type ID **is (** {ID,} **);**

type ID **is range** number **downto** | **to** number;

type ID **is array (** {range | TYPEID ,}**) of** TYPEID**;**

type ID **is record**
  {ID **:** TYPEID;}
end record;

type ID **is access** TYPEID**;**

type ID **is file of** TYPEID**;**

subtype ID **is** SCALARTYPID **range** range;

subtype ID **is** ARRAYTYPID**(** {range,}**);**

subtype ID **is** RESOLVFCTID TYPEID;

range ::=
  (integer | ENUMID **to** | **downto** integer | ENUMID) |
  (OBJID'[**reverse_**]**range**) | (TYPEID **range <>**)

2.2. Other declarations

constant ID **:** TYPEID **:=** expr;

[***shared***] **variable** ID **:** TYPEID [**:=** expr];

signal ID **:** TYPEID [**:=** expr];

file ID **:** TYPEID (**is in** | **out** string;) |
  (***open read_mode*** | ***write_mode*** |
   append_mode is *string;*)

alias ID **:** TYPEID **is** OBJID;

attribute ID **:** TYPEID;

attribute ATTRID **of** OBJID | **others** | **all :** class **is** expr;

class ::=
  entity | **architecture** | **configuration** |
  procedure | **function** | **package** | **type** |
  subtype | **constant** | **signal** | **variable** |
  component | **label**

component ID [***is***]
  [**generic (** {ID **:** TYPEID [**:=** expr];} **);**]
  [**port (**{ID **: in** | **out** | **inout** TYPEID [**:=** expr];}**);**]
end component [*COMPID*];

[***impure*** | ***pure***] **function** ID
  [**(** {[**constant** | **variable** | **signal** | ***file***] ID :
    in | **out** | **inout** TYPEID [**:=** expr];}**)**]
  return TYPEID [**is**
begin
  {sequential_statement}
end [***function***] ID**;**

procedure ID[({[**constant** | **variable** | **signal**] ID :
    in | **out** | **inout** TYPEID [**:=** expr];}**)**]
[**is begin**
  [{sequential_statement}]
end [***procedure***] ID**;**

for LABELID | **others** | **all :** COMPID **use**
  (**entity** [LIBID.]ENTITYID [**(** ARCHID **)**]) |
  (**configuration** [LIBID.]CONFID)
    [[**generic map (** {GENID **=>** expr,} **)**]
     port map ({PORTID **=>** SIGID | *expr*,} **)**];

3. Expressions

expression ::=
  (relation **and** relation) | (relation **nand** relation) |
  (relation **or** relation) | (relation **nor** relation) |
  (relation **xor** relation) | (*relation* ***xnor*** *relation*)

relation ::= shexpr [relop shexpr]

shexpr ::= sexpr [*shop sexpr*]

sexpr ::= [**+**|**-**] term {addop term}

term ::= factor {mulop factor}

factor ::=
  (prim [****** prim]) | (**abs** prim) | (**not** prim)

prim ::=
  literal | OBJID | OBJID'ATTRID | OBJID**(**{expr,}**)**
  | OBJID**(**range**)** | **(**{[choice [{**|** choice}] **=>**] expr,}**)**
  | FCTID**(**{[PARID **=>**] expr,}**)** | TYPEID'**(**expr**)** |
  TYPEID**(**expr**)** | **new** TYPEID[**'(**expr**)**] | **(** expr **)**

choice ::= sexpr | range | RECFID | **others**

3.1. Operators, increasing precedence

logop	**and** | **or** | **xor** | **nand** | **nor** | ***xnor***
relop	**=** | ***/=*** | **<** | **<=** | **>** | **>=**
shop	***sll*** | ***srl*** | ***sla*** | ***sra*** | ***rol*** | ***ror***
addop	**+** | **-** | **&**
mulop	***** | ***/*** | **mod** | **rem**
miscop	****** | **abs** | **not**

4. Sequential Statements

wait [**on** {SIGID,}] [**until** expr] [**for** time];

assert expr
  [**report** string]
  [**severity note** | **warning** | **error** | **failure**];

report *string*
  [severity note | ***warning*** | ***error*** | ***failure];***

SIGID **<=** [**transport**] | [[***reject*** *TIME*] ***inertial***]
    {expr [**after** time],};

VARID **:=** expr**;**

PROCEDUREID[**(**{[PARID **=>**] expr,}**)**];

[*LABEL:*] **if** expr **then**
  {sequential_statement}
[{**elsif** expr **then**
  {sequential_statement}}]
[**else**
  {sequential_statement}]
end if [*LABEL*];

[*LABEL:*] **case** expr **is**
{**when** choice [{**|** choice}] **=>**
  {sequential_statement}}
end case [*LABEL*];

[LABEL**:**] [**while** expr] **loop**
  {sequential_statement}
end loop [LABEL];

[LABEL**:**] **for** ID **in** range **loop**
  {sequential_statement}
end loop [LABEL];

next [LOOPLBL] [**when** expr];

exit [LOOPLBL] [**when** expr];

return [expression];

null;

5. Parallel Statements

LABEL**:** **block** [***is***]
  [**generic (** {ID **:** TYPEID;} **);**
    [**generic map (** {[GENID **=>**] expr,} **);**]]
  [**port (** {ID **: in | out | inout** TYPEID } **);**
    [**port map (** {[PORTID **=>**] SIGID | *expr*,} **)**];]
  [{declaration}]
begin
  [{parallel_statement}]
end block [LABEL];

[LABEL**:**] [***postponed***] **process** [**(** {SIGID,} **)**]
  [{declaration}]
begin
  [{sequential_statement}]
end [***postponed***] **process** [LABEL];

[LBL**:**] [***postponed***] PROCID**(**{[PARID **=>**] expr,}**);**

[LABEL**:**] [***postponed***] **assert** expr
  [**report** string]
  [**severity note** | **warning** | **error** | **failure**];

[LABEL**:**] [***postponed***] SIGID **<=**
  [**transport**] | [[***reject*** *TIME*] ***inertial***]
  [{{expr [**after** TIME,]} | ***unaffected*** **when** expr **else**}]
  {expr [**after** TIME,]} | ***unaffected***;

[LABEL**:**] [***postponed***] **with** expr **select**
  SIGID **<=** [**transport**] | [[***reject*** *TIME*] ***inertial***]
    {{expr [**after** TIME,]} | ***unaffected***
      when choice [{**|** choice}]};

LABEL**:** COMPID
    [[**generic map ({**GENID **=>** expr,} **)**]
    port map ({[PORTID **=>**] SIGID | *expr*,} **)**];

*LABEL**: entity** [LIBID.]ENTITYID [**(**ARCHID**)**]*
    *[[**generic map (** {GENID **=>** expr,} **)***]
    port map (** {[PORTID **=>**] SIGID | expr,} **)];

*LABEL**: configuration** [LIBID.]CONFID*
    *[[**generic map (** {GENID **=>** expr,} **)***]
    port map (** {[PORTID **=>**] SIGID | expr,} **)];

LABEL**:** **if** expr **generate**
  [{parallel_statement}]
end generate [LABEL];

LABEL**:** **for** ID **in** range **generate**
  [{parallel_statement}]
end generate [LABEL];

6. Predefined Attributes

TYPID**'base**	Base type
TYPID**'left**	Left bound value
TYPID**'right**	Right-bound value
TYPID**'high**	Upper-bound value
TYPID**'low**	Lower-bound value
TYPID**'pos(**expr**)**	Position within type
TYPID**'val(**expr**)**	Value at position
TYPID**'succ(**expr**)**	Next value in order
TYPID**'pred(**expr**)**	Previous value in order
TYPID**'leftof(**expr**)**	Value to the left in order
TYPID**'rightof(**expr**)**	Value to the right in order
*TYPID**'ascending***	*Ascending type predicate*
*TYPID**'image(**expr**)***	*String image of value*
*TYPID**'value(**string**)***	*Value of string image*
ARYID**'left[(**expr**)]**	Left-bound of [nth] index
ARYID**'right[(**expr**)]**	Right-bound of [nth] index
ARYID**'high[(**expr**)]**	Upper-bound of [nth] index
ARYID**'low[(**expr**)]**	Lower-bound of [nth] index
ARYID**'range[(**expr**)]**	'left down/to 'right
ARYID**'reverse_range[(**expr**)]**	'right down/to 'left
ARYID**'length[(**expr**)]**	Length of [nth] dimension
*ARYID**'ascending[(**expr**)]***	*'right >= 'left ?*
SIGID**'delayed[(**TIME**)]**	Delayed copy of signal
SIGID**'stable[(**TIME**)]**	Signals event on signal
SIGID**'quiet[(**TIME**)]**	Signals activity on signal
SIGID**'transaction**	Toggles if signal active
SIGID**'event**	Event on signal ?
SIGID**'active**	Activity on signal ?
SIGID**'last_event**	Time since last event
SIGID**'last_active**	Time since last active
SIGID**'last_value**	Value before last event
*SIGID**'driving***	*Active driver predicate*
*SIGID**'driving_value***	*Value of driver*
*OBJID**'simple_name***	*Name of object*
*OBJID**'instance_name***	*Pathname of object*
*OBJID**'path_name***	*Pathname to object*

7. Predefined Types

BOOLEAN	True or false
INTEGER	32 or 64 bits
NATURAL	Integers >= 0
POSITIVE	Integers > 0
REAL	Floating-point
BIT	'0', '1'
BIT_VECTOR(NATURAL)	Array of bits
CHARACTER	7-bit ASCII
STRING(POSITIVE)	Array of characters
TIME	hr, min, sec, ms, us, ns, ps, fs
DELAY_LENGTH	*Time >= 0*

8. Predefined Functions

NOW Returns current simulation time

DEALLOCATE(ACCESSTYPOBJ**)**
Deallocate dynamic object

FILE_OPEN(*[status], FILEID, string, mode****)***
Open file

FILE_CLOSE(*FILEID****)*** *Close file*

9. Lexical Elements

Identifier ::= letter { [underline] alphanumeric }

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

based literal ::=
integer # hexint [**.** hexint] # [**E[+|-]** integer]

bit string literal ::= **B|O|X "** hexint **"**

comment ::= **--** comment text

1164 PACKAGES QUICK REFERENCE CARD

Revision 2.2

()	Grouping	[ ]	Optional
{}	Repeated	|	Alternative
bold	As is	CAPS	User Identifier
italic	VHDL-93	c	commutative

b	::= BIT
bv	::= BIT_VECTOR
u/l	::= STD_ULOGIC/STD_LOGIC
uv	::= STD_ULOGIC_VECTOR
lv	::= STD_LOGIC_VECTOR
un	::= UNSIGNED
sg	::= SIGNED
in	::= INTEGER
na	::= NATURAL
sm	::= SMALL_INT (subtype INTEGER range 0 to 1)

1. IEEE's STD_LOGIC_1164

1.1 Logic Values

'U'	Uninitialized
'X'/'W'	Strong/Weak unknown
'0'/'L'	Strong/Weak 0
'1'/'H'	Strong/Weak 1
'Z'	High Impedance
'-'	Don't care

1.2 Predefined Types

STD_ULOGIC Base type

Subtypes:

STD_LOGIC	Resolved STD_ULOGIC
X01	Resolved X, 0 & 1
X01Z	Resolved X, 0, 1 & Z
UX01	Resolved U, X, 0 & 1
UX01Z	Resolved U, X, 0, 1 & Z

STD_ULOGIC_VECTOR(na **to | downto** na**)** Array of STD_ULOGIC

STD_LOGIC_VECTOR(na **to | downto** na**)** Array of STD_LOGIC

1.3 Overloaded Operators

Description	Left	Operator	Right
bitwise-and	u/l,uv,lv	**and**, **nand**	u/l,uv,lv
bitwise-or	u/l,uv,lv	**or**, **nor**	u/l,uv,lv
bitwise-xor	u/l,uv,lv	**xor**, **xnor**	u/l,uv,lv
bitwise-not		**not**	u/l,uv,lv

1.4 Conversion Functions

From	To	Function
u/l	b	**TO_BIT(**from**[,** xmap**])**
uv,lv	bv	**TO_BITVECTOR(**from**[,** xmap**])**
b	u/l	**TO_STDULOGIC(**from**)**
bv,uv	lv	**TO_STDLOGICVECTOR(**from**)**
bv,lv	uv	**TO_STDULOGICVECTOR(**from**)**

2. IEEE's NUMERIC_STD

2.1 Predefined Types

UNSIGNED(na **to | downto** na**)** Array of STD_LOGIC

SIGNED(na **to | downto** na**)** Array of STD_LOGIC

2.2 Overloaded Operators

Left	Op	Right	Return
	abs	sg	sg
	-	sg	sg
un	**+,-,*,/,rem,mod**	un	un
sg	**+,-,*,/,rem,mod**	sg	sg
un	**+,-,*,/,rem,mod** $_c$	na	un
sg	**+,-,*,/,rem,mod** $_c$	in	sg
un	**<,>,<=,>=,=,/=**	un	bool
sg	**<,>,<=,>=,=,/=**	sg	bool
un	**<,>,<=,>=,=,/=** $_c$	na	bool
sg	**<,>,<=,>=,=,/=** $_c$	in	bool

2.3 Predefined Functions

SHIFT_LEFT(un, na**)**	un
SHIFT_RIGHT(un, na**)**	un
SHIFT_LEFT(sg, na**)**	sg
SHIFT_RIGHT(sg, na**)**	sg
ROTATE_LEFT(un, na**)**	un
ROTATE_RIGHT(un, na**)**	un
ROTATE_LEFT(sg, na**)**	sg
ROTATE_RIGHT(sg, na**)**	sg
RESIZE(sg, na**)**	sg
RESIZE(un, na**)**	un
STD_MATCH(u/l, u/l**)**	bool
STD_MATCH(uv, uv**)**	bool
STD_MATCH(lv, lv**)**	bool
STD_MATCH(un, un**)**	bool
STD_MATCH(sg, sg**)**	bool

2.4 Conversion Functions

From	To	Function
un,lv	sg	**SIGNED(**from**)**
sg,lv	un	**UNSIGNED(**from**)**
un,sg	lv	**STD_LOGIC_VECTOR(**from**)**
un,sg	in	**TO_INTEGER(**from**)**
na	un	**TO_UNSIGNED(**from, size**)**
in	sg	**TO_SIGNED(**from, size**)**

3. IEEE's NUMERIC_BIT

3.1 Predefined Types

UNSIGNED(na **to | downto** na**)** Array of BIT

SIGNED(na **to | downto** na**)** Array of BIT

3.2 Overloaded Operators

Left	Op	Right	Return
	abs	sg	sg
	-	sg	sg
un	**+,-,*,/,rem,mod**	un	un
sg	**+,-,*,/,rem,mod**	sg	sg
un	**+,-,*,/,rem,mod** $_c$	na	un
sg	**+,-,*,/,rem,mod** $_c$	in	sg
un	**<,>,<=,>=,=,/=**	un	bool
sg	**<,>,<=,>=,=,/=**	sg	bool
un	**<,>,<=,>=,=,/=** $_c$	na	bool
sg	**<,>,<=,>=,=,/=** $_c$	in	bool

3.3 Predefined Functions

SHIFT_LEFT(un, na**)**	un
SHIFT_RIGHT(un, na**)**	un
SHIFT_LEFT(sg, na**)**	sg
SHIFT_RIGHT(sg, na**)**	sg
ROTATE_LEFT(un, na**)**	un
ROTATE_RIGHT(un, na**)**	un
ROTATE_LEFT(sg, na**)**	sg
ROTATE_RIGHT(sg, na**)**	sg
RESIZE(sg, na**)**	sg
RESIZE(un, na**)**	un

3.4 Conversion Functions

From	To	Function
un,bv	sg	**SIGNED(**from**)**
sg,bv	un	**UNSIGNED(**from**)**
un,sg	bv	**BIT_VECTOR(**from**)**
un,sg	in	**TO_INTEGER(**from**)**
na	un	**TO_UNSIGNED(**from**)**
in	sg	**TO_SIGNED(**from**)**

4. SYNOPSYS' STD_LOGIC_ARITH

4.1 PREDEFINED TYPES

UNSIGNED(na **to | downto** na**)**	Array of STD_LOGIC
SIGNED(na **to | downto** na**)**	Array of STD_LOGIC
SMALL_INT	Integer subtype, 0 or 1

4.2 OVERLOADED OPERATORS

Left	Op	Right	Return
	abs	sg	sg,lv
	-	sg	sg,lv
un	**+,-,***	un	un,lv
sg	**+,-,***	sg	sg,lv
sg	**+,-,***	un	sg,lv
un	**+,- c**	in	un,lv
sg	**+,- c**	in	sg,lv
un	**+,- c**	u/l	un,lv
sg	**+,- c**	u/l	sg,lv
un	**<,>,<=,>=,=,/=**	un	bool
sg	**<,>,<=,>=,=,/=**	sg	bool
un	**<,>,<=,>=,=,/= c**	in	bool
sg	**<,>,<=,>=,=,/= c**	in	bool

4.3 PREDEFINED FUNCTIONS

SHL(un, un**)**	un	**SHR(**un, un**)**	un
SHL(sg, un**)**	sg	**SHR(**sg, un**)**	sg
EXT(lv, in**)**	lv	zero-extend	
SEXT(lv, in**)**	lv	sign-extend	

4.4 CONVERSION FUNCTIONS

From	To	Function
un,lv	sg	**SIGNED(**from**)**
sg,lv	un	**UNSIGNED(**from**)**
sg,un	lv	**STD_LOGIC_VECTOR(**from**)**
un,sg	in	**CONV_INTEGER(**from**)**
in,un,sg,u	un	**CONV_UNSIGNED(**from, size**)**
in,un,sg,u	sg	**CONV_SIGNED(**from, size**)**
in,un,sg,u	lv	**CONV_STD_LOGIC_VECTOR(**from, size**)**

5. SYNOPSYS' STD_LOGIC_UNSIGNED

5.1 OVERLOADED OPERATORS

Left	Op	Right	Return
	+	lv	lv
lv	**+,-,***	lv	lv
lv	**+,-c**	in	lv
lv	**+,- c**	u/l	lv
lv	**<,>,<=,>=,=,/=**	lv	bool
lv	**<,>,<=,>=,=,/= c**	in	bool

5.2 CONVERSION FUNCTIONS

From	To	Function
lv	in	**CONV_INTEGER**(from)

6. SYNOPSYS' STD_LOGIC_SIGNED

6.1 OVERLOADED OPERATORS

Left	Op	Right	Return
	abs	lv	lv
	+,-	lv	lv
lv	**+,-,***	lv	lv
lv	**+,-c**	in	lv
lv	**+,- c**	u/l	lv
lv	**<,>,<=,>=,=,/=**	lv	bool
lv	**<,>,<=,>=,=,/= c**	in	bool

6.2 CONVERSION FUNCTIONS

From	To	Function
lv	in	**CONV_INTEGER(**from**)**

7. SYNOPSYS' STD_LOGIC_MISC

7.1 PREDEFINED FUNCTIONS

AND_REDUCE(lv | uv**)**	u/l
[X]OR_REDUCE(lv | uv**)**	u/l
[N]AND_REDUCE(lv | uv**)**	UX01
OR_REDUCE(lv | uv**)**	UX01
NOR_REDUCE(lv | uv**)**	UX01
XOR_REDUCE(lv | uv**)**	UX01
XNOR_REDUCE(lv | uv**)**	UX01

8. EXEMPLAR'S STD_LOGIC_ARITH

8.1 OVERLOADED OPERATORS

Left	Op	Right	Return
	+,-,*,	u/l	u/l
	abs	u/l	u/l

8.2 PREDEFINED FUNCTIONS

sl(u/l**,** in**)**	u/l
sl2(u/l**,** in**)**	u/l
sr(u/l**,** in**)**	u/l
sr2(u/l**,** in**)**	u/l
add(u/l**)**	u/l
add2(u/l**)**	u/l
sub(u/l**)**	u/l
sub2(u/l**)**	u/l
mult(u/l**)**	u/l
mult2(u/l**)**	u/l
extend(u/l, in**)**	u/l
extend2(u/l, in**)**	u/l
comp2(u/l**)**	u/l

8.3 CONVERSION FUNCTIONS

From	To	Function
bool	uv	**bool2elb**
uv	bool	**elb2bool**
u/l	na	**evec2int**
in	u/l	**int2evec (**size**)**
uv	na	**elb2int**

9. MENTOR'S STD_LOGIC_ARITH

9.1 PREDEFINED TYPES

UNSIGNED(na **to** | **downto** na**)**	Array of STD_LOGIC
SIGNED(na **to** | **downto** na**)**	Array of STD_LOGIC

9.2 OVERLOADED OPERATORS

Left	Op	Right	Return
	abs	sg	sg
	-	sg	sg
u/l	**+,-**	u/l	u/l
uv	**+,-,*,/,mod,rem,****	uv	uv
lv	**+,-,*,/,mod,rem,****	lv	lv
un	**+,-,*,/,mod,rem,****	un	un
sg	**+,-,*,/,mod,rem,****	sg	sg
un	**<,>,<=,>=,=,/=**	un	bool
sg	**<,>,<=,>=,=,/=**	sg	bool
	not	un	un
	not	sg	sg
un	**and,nand,or,nor,xor**	un	un
sg	**and,nand,or,nor,xor,*xnor***	sg	sg
uv	***sla,sra,sll,srl,rol,ror***	uv	uv
lv	***sla,sra,sll,srl,rol,ror***	lv	lv
un	***sla,sra,sll,srl,rol,ror***	un	un
sg	***sla,sra,sll,srl,rol,ror***	sg	sg

9.3 PREDEFINED FUNCTIONS

ZERO_EXTEND(uv | lv | un**,** na**)**	same
ZERO_EXTEND(u/l**,** na**)**	lv
SIGN_EXTEND(sg**,** na**)**	sg
AND_REDUCE(uv | lv | un | sg**)**	u/l
OR_REDUCE(uv | lv | un | sg**)**	u/l
XOR_REDUCE(uv | lv | un | sg**)**	u/l

9.4 CONVERSION FUNCTIONS

From	To	Function
u/l,uv,lv,un,sg	in	**TO_INTEGER(**from**)**
u/l,uv,lv,un,sg	in	**CONV_INTEGER(**from**)**
bool	u/l	**TO_STDLOGIC(**from**)**
na	un	**TO_UNSIGNED(**from,size**)**
na	un	**CONV_UNSIGNED(**from,size**)**
in	sg	**TO_SIGNED(**from,size**)**
in	sg	**CONV_SIGNED(**from,size**)**
na	lv	**TO_STDLOGICVECTOR(**from,size**)**
na	uv	**TO_STDULOGICVECTOR(**from,size**)**