QUALIS DESIGN CORPORATION

VHDL QUICK REFERENCE CARD

REVISION 1.1

()	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;**}]

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 | SUBTYPID**;**

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***] **function** ID
  [**(** {[**constant** | **variable** | **signal**] 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 **or** relation) |
  (relation **xor** 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**
relop	**=** | **/=** | **<** | **<=** | **>** | **=>**
shop	***sll*** | ***srl*** | ***sla*** | ***sra*** | ***rol*** | ***ror***
addop	**+** | **-** | **&**
mulop	***** | **/** | **mod** | **rem**
miscop	****** | **abs** | **not**

See reverse side for additional information.

© 1995 Qualis Design Corporation

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,} **)**];

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

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

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**'prec(**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**[**(**expr**)**]	Delayed copy of signal
SIGID**'stable**[**(**expr**)**]	Signals event on signal
SIGID**'quiet**[**(**expr**)**]	Signals activity on signal
SIGID**'transaction**[**(**expr**)**]	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

Qualis Design Corporation
Beaverton, OR USA
Phone: +1-503-531-0377 FAX: +1-503-629-5525
E-mail: info@qualis.com

© 1995 Qualis Design Corporation

QUALIS DESIGN CORPORATION

1164 PACKAGES QUICK REFERENCE CARD

REVISION 1.0

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

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

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**	u/l,uv,lv
bitwise-or	u/l,uv,lv	**or**	u/l,uv,lv
bitwise-xor	u/l,uv,lv	**xor**	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,ul	lv	**TO_STDLOGICVECTOR(**from**)**
bv,lv	uv	**TO_STDULOGICVECTOR(**from**)**

1.5. PREDICATES

RISING_EDGE(SIGID**)**	Rise edge on signal ?
FALLING_EDGE(SIGID**)**	Fall edge on signal ?
IS_X(OBJID**)**	Object contains 'X' ?

2. IEEE'S NUMERIC_STD

2.1. PREDEFINED TYPES

UNSIGNED(na **to** | **downto** na**)**
SIGNED(na **to** | **downto** na)
Arrays 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

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**)**
in	sg	**TO_SIGNED(**from**)**

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**)**

See reverse side for additional information.