

# Laboratório de Sistemas Digitais

## Aula Teórico-Prática 12

Ano Letivo 2016/17

As construções “**for...generate**” e  
“**if...generate**” em VHDL

Atributos pré-definidos em VHDL

*Reference cards* de VHDL e STD\_LOGIC\_1164

# Conteúdo

- A construção **for...generate** de VHDL para replicação de circuitos lógicos
  - Exemplos com
    - Instanciação de entidades
    - Atribuições concorrentes (condicionais)
    - Processos
- Atributos pré-definidos em VHDL
- *Reference cards* de VHDL e *package 1164*

# A Construção **for...generate** de VHDL para Replicação de Hardware – Ciclo Estrutural

```
label : for <index> in <KMIN> to <KMAX> generate
```

    <circuito lógico a replicar descrito com

        atribuição(ões) concorrente(s) condicional(is)

        processos(s)

        instanciação de entidade(s)>

```
end generate;
```

em que:

**<index>** - índice inteiro (com declaração implícita)

**<KMIN>** e **<KMAX>** – constantes inteiras (valor definido em *compile time*)

A construção **for...generate** deve ser escrita no corpo de uma arquitetura (e fora de processos!)

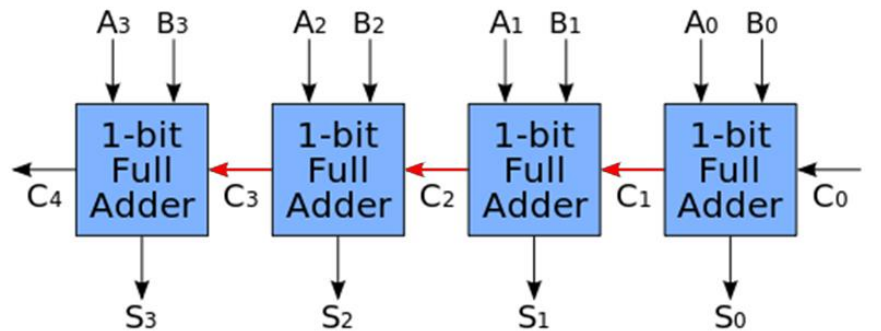
Também pode ser escrita na forma:

```
label : for <index> in <KMAX> downto <KMIN>
```

# A Construção **for...generate** de VHDL

## Exemplo de Motivação

- Somador de 4 bits (guião prático 3)
- Construído com 4 somadores completo de 1 bit em cascata
- Esqueleto do somador completo (retirado do guião)



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity FullAdder is
    port(a, b, cin : in  std_logic;
          s, cout  : out std_logic);
end FullAdder;
```

```
architecture Behavioral of FullAdder is
begin
```

```
    -- Especifique aqui as equações lógicas para as saídas "s" e "cout"
end Behavioral;
```

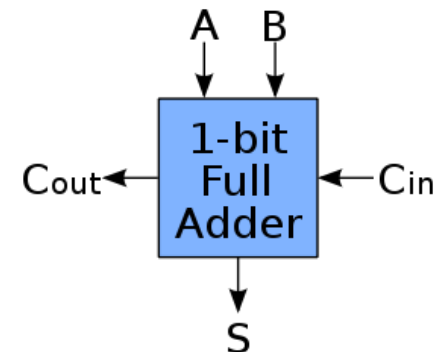


Figura 2 – Esqueleto do código VHDL da entidade **FullAdder** e respetiva arquitetura **Behavioral**.

# A Construção **for...generate** de VHDL

## Exemplo de Motivação

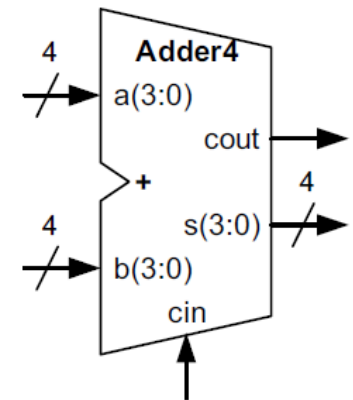
- Esqueleto do somador de 4 bits – cascata (retirado do guião)

```
-- Inclua as bibliotecas e os pacotes necessários

entity Adder4 is
    port(a, b : in  std_logic_vector(3 downto 0);
         cin : in  std_logic;
         s   : out std_logic_vector(3 downto 0);
         cout : out std_logic);
end Adder4;

architecture Structural of Adder4 is
    -- Declare um sinal interno (carryOut) do tipo std_logic_vector (de
    -- C bits) que interligará os bits de carry dos somadores entre si
begin
    bit0: entity work.FullAdder(Behavioral)
        port map(a    => a(0),
                 b    => b(0),
                 cin  => cin,
                 s    => s(0),
                 cout => carryOut(0));

    -- complete para os restantes bits (1 a 3)
end Structural;
```



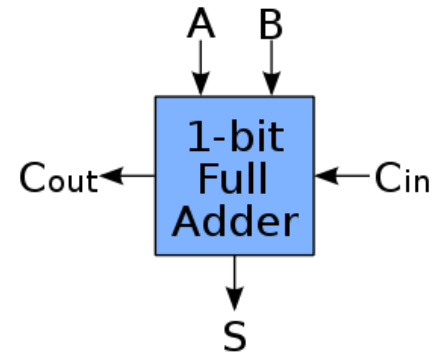
E se o somador  
tivesse 64 bits?

E se fosse  
parametrizável  
(estaticamente /  
em *compile time*)?

Figura 4 – Esqueleto do código VHDL da entidade **Adder4** e arquitetura **Structural**.

# Código Completo de Somador Completo de 1 bit

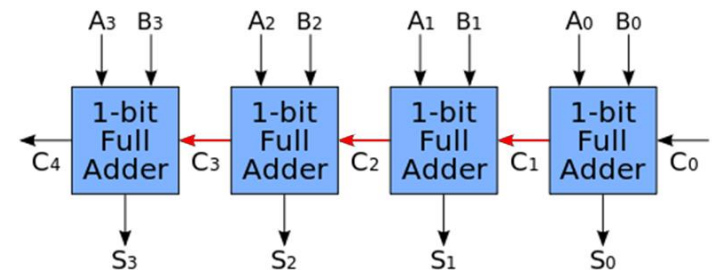
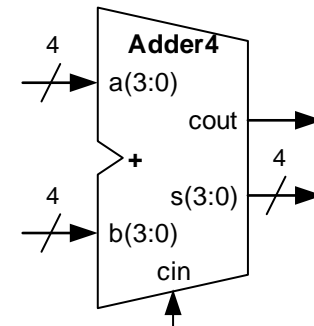
```
Text Editor - C:/Users/asroliveira/CloudStation/LSDig2017/SlidesTP/Gen...
File Edit View Project Processing Tools Window Help Search altera.com
267 268
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity FullAdder is
5     port(a, b, cin : in std_logic;
6         s, cout : out std_logic);
7 end FullAdder;
8
9 architecture Behavioral of FullAdder is
10 begin
11     s <= a xor b xor cin;
12     cout <= (a and b) or (a and cin) or (b and cin);
13 end Behavioral;
14
100% 00:00:22
```



# Instanciação de 4 Somadores Completos de 1 bit com **for...generate**

```
Text Editor - C:/Users/asoliveira/CloudStation/LSDig2017/SlidesTP/...
File Edit View Project Processing Tools Window Help Search altera.com

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity Adder4 is
5     port(a, b : in std_logic_vector(3 downto 0);
6         cin : in std_logic;
7         s : out std_logic_vector(3 downto 0);
8         cout : out std_logic);
9 end Adder4;
10
11 architecture Structural of Adder4 is
12     signal s_carry : std_logic_vector(4 downto 0);
13
14 begin
15     s_carry(0) <= cin;
16
17     adder_loop : for i in 0 to 3 generate
18     cell: entity work.FullAdder(Behavioral)
19     port map(a => a(i),
20             b => b(i),
21             cin => s_carry(i),
22             s => s(i),
23             cout => s_carry(i+1));
24
25     end generate;
26
27     cout <= s_carry(4);
28 end Structural;
29
```

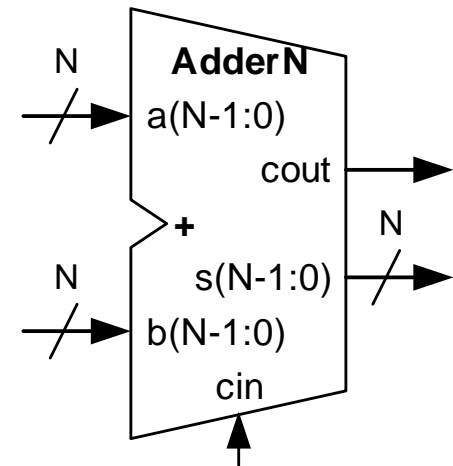


# Construção de um Somador Parametrizável de N bits com **for...generate**

```
Text Editor - C:/Users/asoliveira/CloudStation/LSDig2017/SlidesTP/Generate...
File Edit View Project Processing Tools Window Help Search altera.com
267 268

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity AdderN is
5      generic(N : integer := 8);
6      port(a, b : in std_logic_vector(N-1 downto 0);
7          cin : in std_logic;
8          s : out std_logic_vector(N-1 downto 0);
9          cout : out std_logic);
10 end AdderN;
11
12 architecture Structural of AdderN is
13     signal s_carry : std_logic_vector(N downto 0);
14
15 begin
16     s_carry(0) <= cin;
17
18     adder_loop : for i in 0 to (N-1) generate
19         cell: entity work.FullAdder(Behavioral)
20         port map(a => a(i),
21                 b => b(i),
22                 cin => s_carry(i),
23                 s => s(i),
24                 cout => s_carry(i+1));
25     end generate;
26
27     cout <= s_carry(N);
28 end Structural;
29
30
```

Ln 1 Col 1 VHDL File 100% 00:00:21





# Exemplo de `for...generate` com uma Atribuição Condicional

- Descodificador binário de  $N \rightarrow 2^N$  bits, com  $N$  parametrizável

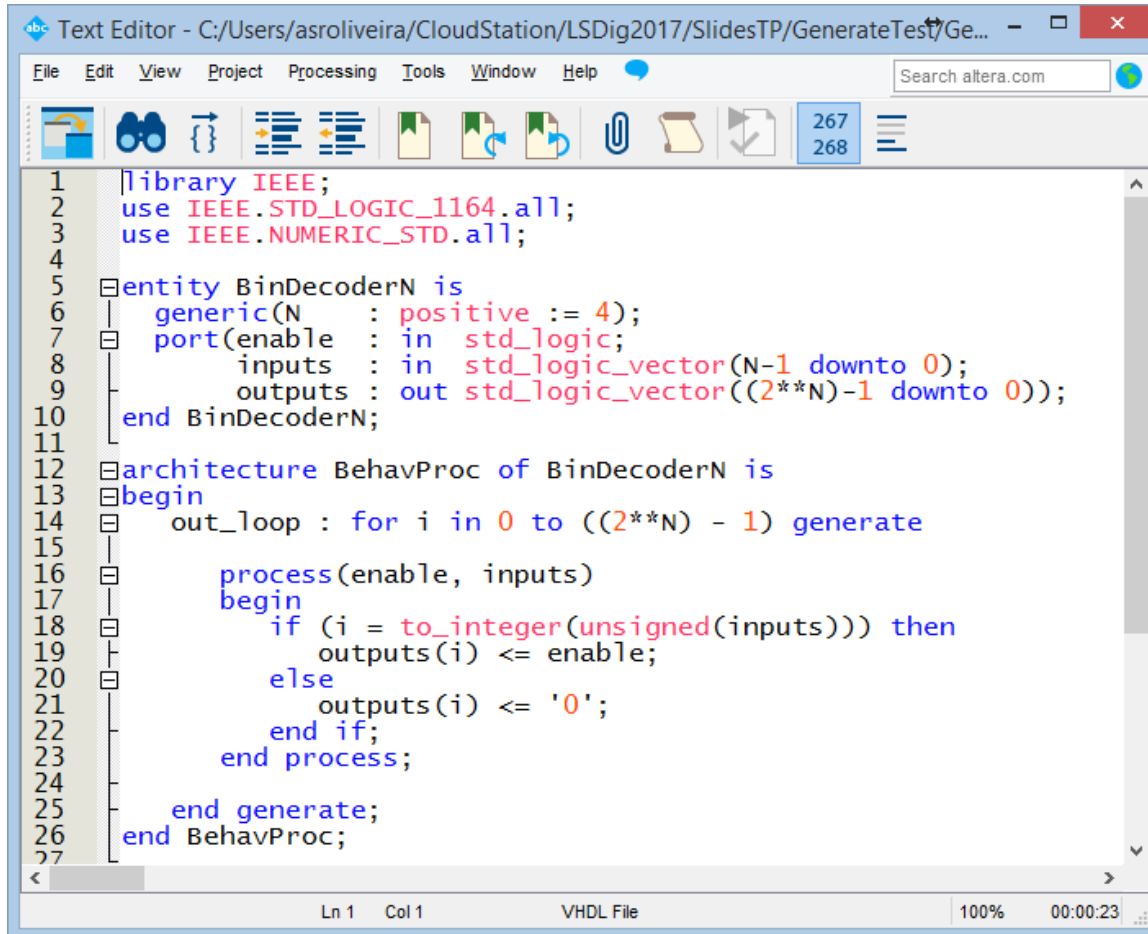
The image shows a text editor window titled "Text Editor - C:/Users/asoliveira/CloudStation/LSDig2017/SlidesTP/GenerateTest/Genera" with a menu bar (File, Edit, View, Project, Processing, Tools, Window, Help) and a toolbar. The code is as follows:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3 use IEEE.NUMERIC_STD.all;
4
5 entity BinDecoderN is
6     generic(N : positive := 4);
7     port(enable : in std_logic;
8           inputs : in std_logic_vector(N-1 downto 0);
9           outputs : out std_logic_vector((2**N)-1 downto 0));
10 end BinDecoderN;
11
12 architecture BehavAssign of BinDecoderN is
13 begin
14     out_loop : for i in 0 to ((2**N) - 1) generate
15     --
16         outputs(i) <= enable when (i = to_integer(unsigned(inputs))) else
17             '0';
18     --
19     end generate;
20 end BehavAssign;
21
```

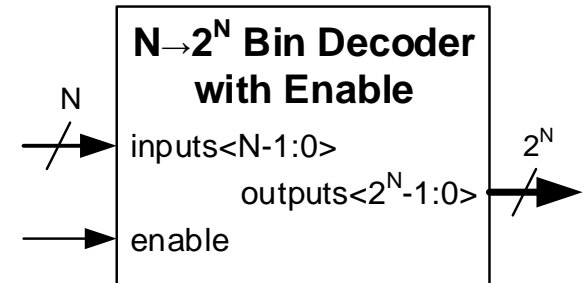
To the right of the code is a block diagram for the "N → 2<sup>N</sup> Bin Decoder with Enable". It has an input "inputs<N-1:0>" with a bus width of  $N$ , an input "enable", and an output "outputs<2<sup>N</sup>-1:0>" with a bus width of  $2^N$ .

# Exemplo de **for...generate** com um Processo

- Descodificador binário de  $N \rightarrow 2^N$  bits, com  $N$  parametrizável



```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3 use IEEE.NUMERIC_STD.all;
4
5 entity BinDecoderN is
6     generic(N : positive := 4);
7     port(enable : in std_logic;
8           inputs : in std_logic_vector(N-1 downto 0);
9           outputs : out std_logic_vector((2**N)-1 downto 0));
10 end BinDecoderN;
11
12 architecture BehavProc of BinDecoderN is
13 begin
14     out_loop : for i in 0 to ((2**N) - 1) generate
15     |
16         process(enable, inputs)
17         begin
18             if (i = to_integer(unsigned(inputs))) then
19                 outputs(i) <= enable;
20             else
21                 outputs(i) <= '0';
22             end if;
23         end process;
24     end generate;
25 end BehavProc;
```



A construção **for...generate** também é usada no pacote de suporte aos projetos finais para instanciar vários módulos de *debounce* (nos ficheiros "audio\_tl.vhd" e "ps2\_vga\_text\_buffer\_tl.vhd")

# A Construção **if...generate** de VHDL para Inclusão Condicional de Hardware

```
label : if <condition> generate
    <circuito lógico a incluir se <condition> for TRUE>
else generate
    <circuito lógico a incluir se <condition> for FALSE>
end generate;
```

- A condição é avaliada em *compile time*
- O bloco **else...generate** é opcional
- O circuito lógico pode ser descrito com
  - Atribuição(ões) concorrente(s) condicional(is)
  - Processos(s)
  - Instanciação de entidade(s)
- A construção **if...generate** deve ser escrita no corpo de uma arquitetura (e fora de processos!) – tal como o **for...generate**!

# Exemplo da Construção **if...generate** de VHDL

Exemplo retirado “vga.vhd” (do pacote de suporte aos projetos finais) para instanciar um módulo gerador de *clock* se a frequência de entrada não for 50 MHz)

```
dir_clk: if (vga_freq > 49.999e6) and
           (vga_freq < 50.001e6) generate
    vga_clock <= clock_50;
else generate
    clk_gen : entity work.clock_generator(cyclone4e)
              generic map(frequency => vga_freq)
              port map(clock_50 => clock_50,
                      new_clock => vga_clock);
end generate;
```

# Atributos Pré-definidos em VHDL

- Permitem testar condições e obter informação sobre tipos e objetos (*arrays*, sinais, portos, etc.) durante a simulação e a síntese de um modelo VHDL
- Vamos considerar apenas um pequeno subconjunto dos definidos na linguagem

Atributo aplicável a sinais e portos

- **SIGID'event**                      Event on signal
  - Exemplo: `if (clk'event and clk = '1')`

# Atributos Aplicáveis a Tipos em VHDL

Úteis para escrever código mais flexível

<code>TYPID'left</code>	Left bound value
<code>TYPID'right</code>	Right-bound value
<code>TYPID'high</code>	Upper-bound value
<code>TYPID'low</code>	Lower-bound value
<code>TYPID'pos (expr)</code>	Position within type
<code>TYPID'val (expr)</code>	Value at position

# Exemplos de Atributos Aplicáveis a Tipos em VHDL

```
integer'left    = -2147483648
integer'right   = 2147483647
integer'high    = 2147483647
integer'low     = -2147483648
integer'pos(0)  = 0
integer'val(0)  = 0
```

Considerando as declarações:

```
type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
subtype std_logic is resolved std_ulogic;
```

```
std_logic'left    = 'U'
std_logic'right   = '-'
std_logic'high    = '-'
std_logic'low     = 'U'
std_logic'pos('X') = 1    (valor inteiro)
std_logic'val(4)   = 'Z'
```

# Atributos Aplicáveis a *Arrays* em VHDL

Úteis também para escrever código mais flexível

<code>ARYID[nth] 'left</code>	Left-bound of [nth] index
<code>ARYID[nth] 'right</code>	Right-bound of [nth] index
<code>ARYID[nth] 'high</code>	Upper-bound of [nth] index
<code>ARYID[nth] 'low</code>	Lower-bound of [nth] index
<code>ARYID[nth] 'range</code>	'left down/to 'right
<code>ARYID[nth] 'reverse_range</code>	'right down/to 'left
<code>ARYID[nth] 'length</code>	Length of [nth] dimension



# Exemplos de Atributos Aplicáveis a Arrays em VHDL

Considerando as declarações:

```
subtype TDataWord is std_logic_vector(7 downto 0);  
type TMemory is array (0 to 31) of TDataWord;  
signal s_memory : TMemory;  
signal s_data : TDataWord;
```

```
s_memory'left      = 0  
s_memory'right     = 31  
s_memory'high      = 31  
s_memory'low       = 0  
s_memory'range     = (0 to 31)  
s_memory'reverse_range = (31 downto 0)  
s_memory'length    = 32
```

```
s_memory(0)'left      = 7  
s_memory(0)'right     = 0  
s_memory(0)'high      = 7  
s_memory(0)'low       = 0  
s_memory(0)'range     = (7 downto 0)  
s_memory(0)'reverse_range = (0 to 7)  
s_memory(0)'length    = 8
```

```
s_data'left      = 7  
s_data'right     = 0  
s_data'high      = 7  
s_data'low       = 0  
s_data'range     = (7 downto 0)  
s_data'reverse_range = (0 to 7)  
s_data'length    = 8
```



# VHDL QUICK REFERENCE CARD

Revision 2.2

()	Grouping	[]	Optional
{}	Repeated		Alternative
<b>bold</b>	As is	CAPS	User Identifier
<i>italic</i>	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]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

1995-2000 Qualis Design Corporation. Permission to reproduce and distribute strictly verbatim copies of this document in whole is hereby granted.

See reverse side for additional information.

#### 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}
[elseif 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.]JENTITYID ([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**(ACCESSTYOBJ)  
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

© 1995-2000 Qualis Design Corporation. Permission to reproduce and distribute strictly verbatim copies of this document in whole is hereby granted.

**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:www.qualis.com  
**Also available:** 1164 Packages Quick Reference Card  
Verilog HDL Quick Reference Card



# 1164 PACKAGES QUICK REFERENCE CARD

Revision 2.2

()	Grouping	[]	Optional
{}	Repeated		Alternative
<b>bold</b>	As is	CAPS	User Identifier
<i>italic</i>	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

<b>STD_ULOGIC</b>	Base type
Subtypes:	
<b>STD_LOGIC</b>	Resolved STD_ULOGIC
<b>X01</b>	Resolved X, 0 & 1
<b>X01Z</b>	Resolved X, 0, 1 & Z
<b>UX01</b>	Resolved U, X, 0 & 1
<b>UX01Z</b>	Resolved U, X, 0, 1 & Z
<b>STD_ULOGIC_VECTOR</b> (na to   downto na)	Array of STD_ULOGIC
<b>STD_LOGIC_VECTOR</b> (na to   downto na)	Array of STD_LOGIC

## 1.3 OVERLOADED OPERATORS

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

## 1.4 CONVERSION FUNCTIONS

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

## 2. IEEE's NUMERIC\_STD

### 2.1 PREDEFINED TYPES

<b>UNSIGNED</b> (na to   downto na)	Array of STD_LOGIC
<b>SIGNED</b> (na to   downto na)	Array of STD_LOGIC

### 2.2 OVERLOADED OPERATORS

Left	Op	Right	Return
	<b>abs</b>	sg	sg
	-	sg	sg
un	<b>+, -, *, /, rem, mod</b>	un	un
sg	<b>+, -, *, /, rem, mod</b>	sg	sg
un	<b>+, -, *, /, rem, mod c</b>	na	un
sg	<b>+, -, *, /, rem, mod c</b>	in	sg
un	<b>&lt;, &gt;, &lt;=, &gt;=, /=</b>	un	bool
sg	<b>&lt;, &gt;, &lt;=, &gt;=, /=</b>	sg	bool
un	<b>&lt;, &gt;, &lt;=, &gt;=, /= c</b>	na	bool
sg	<b>&lt;, &gt;, &lt;=, &gt;=, /= c</b>	in	bool

### 2.3 PREDEFINED FUNCTIONS

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

## 2.4 CONVERSION FUNCTIONS

From	To	Function
un, lv	sg	<b>SIGNED</b> (from)
sg, lv	un	<b>UNSIGNED</b> (from)
un, sg	lv	<b>STD_LOGIC_VECTOR</b> (from)
un, sg	in	<b>TO_INTEGER</b> (from)
na	un	<b>TO_UNSIGNED</b> (from, size)
in	sg	<b>TO_SIGNED</b> (from, size)

## 3. IEEE's NUMERIC\_BIT

### 3.1 PREDEFINED TYPES

<b>UNSIGNED</b> (na to   downto na)	Array of BIT
<b>SIGNED</b> (na to   downto na)	Array of BIT

### 3.2 OVERLOADED OPERATORS

Left	Op	Right	Return
	<b>abs</b>	sg	sg
	-	sg	sg
un	<b>+, -, *, /, rem, mod</b>	un	un
sg	<b>+, -, *, /, rem, mod</b>	sg	sg
un	<b>+, -, *, /, rem, mod c</b>	na	un
sg	<b>+, -, *, /, rem, mod c</b>	in	sg
un	<b>&lt;, &gt;, &lt;=, &gt;=, /=</b>	un	bool
sg	<b>&lt;, &gt;, &lt;=, &gt;=, /=</b>	sg	bool
un	<b>&lt;, &gt;, &lt;=, &gt;=, /= c</b>	na	bool
sg	<b>&lt;, &gt;, &lt;=, &gt;=, /= c</b>	in	bool

### 3.3 PREDEFINED FUNCTIONS

<b>SHIFT_LEFT</b> (un, na)	un
<b>SHIFT_RIGHT</b> (un, na)	un
<b>SHIFT_LEFT</b> (sg, na)	sg
<b>SHIFT_RIGHT</b> (sg, na)	sg
<b>ROTATE_LEFT</b> (un, na)	un
<b>ROTATE_RIGHT</b> (un, na)	un
<b>ROTATE_LEFT</b> (sg, na)	sg
<b>ROTATE_RIGHT</b> (sg, na)	sg
<b>RESIZE</b> (sg, na)	sg
<b>RESIZE</b> (un, na)	un

### 3.4 CONVERSION FUNCTIONS

From	To	Function
un, bv	sg	<b>SIGNED</b> (from)
sg, bv	un	<b>UNSIGNED</b> (from)
un, sg	bv	<b>BIT_VECTOR</b> (from)
un, sg	in	<b>TO_INTEGER</b> (from)
na	un	<b>TO_UNSIGNED</b> (from)
in	sg	<b>TO_SIGNED</b> (from)

© 1995 - 2000 Qualis Design Corporation. Permission to reproduce and distribute strictly verbatim copies of this document in whole is hereby granted. See reverse side for additional information.



## 4. SYNOPSIS'S STD\_LOGIC\_ARITH

### 4.1 PREDEFINED TYPES

<b>UNSIGNED</b> (na to   downto na)	Array of STD_LOGIC
<b>SIGNED</b> (na to   downto na)	Array of STD_LOGIC
<b>SMALL_INT</b>	Integer subtype, 0 or 1

### 4.2 OVERLOADED OPERATORS

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

### 4.3 PREDEFINED FUNCTIONS

<b>SHL</b> (un, un)	un	<b>SHR</b> (un, un)	un
<b>SHL</b> (sg, un)	sg	<b>SHR</b> (sg, un)	sg
<b>EXT</b> (lv, in)	lv	zero-extend	
<b>SEXT</b> (lv, in)	lv	sign-extend	

### 4.4 CONVERSION FUNCTIONS

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

## 5. SYNOPSIS'S STD\_LOGIC\_UNSIGNED

### 5.1 OVERLOADED OPERATORS

Left	Op	Right	Return
	<b>+</b>	lv	lv
lv	<b>+, -, *</b>	lv	lv
lv	<b>+, -c</b>	in	lv
lv	<b>+, -c</b>	u/l	lv
lv	<b>&lt;, &gt;, &lt;=, &gt;=, =, /=</b>	lv	bool
lv	<b>&lt;, &gt;, &lt;=, &gt;=, =, /= c</b>	in	bool

### 5.2 CONVERSION FUNCTIONS

From	To	Function
lv	in	<b>CONV_INTEGER</b> (from)

## 6. SYNOPSIS'S STD\_LOGIC\_SIGNED

### 6.1 OVERLOADED OPERATORS

Left	Op	Right	Return
	<b>abs</b>	lv	lv
	<b>+, -</b>	lv	lv
lv	<b>+, -, *</b>	lv	lv
lv	<b>+, -c</b>	in	lv
lv	<b>+, -c</b>	u/l	lv
lv	<b>&lt;, &gt;, &lt;=, &gt;=, =, /=</b>	lv	bool
lv	<b>&lt;, &gt;, &lt;=, &gt;=, =, /= c</b>	in	bool

### 6.2 CONVERSION FUNCTIONS

From	To	Function
lv	in	<b>CONV_INTEGER</b> (from)

## 7. SYNOPSIS'S STD\_LOGIC\_MISC

### 7.1 PREDEFINED FUNCTIONS

<b>AND_REDUCE</b> (lv   uv)	u/l
<b>XOR_REDUCE</b> (lv   uv)	u/l
<b>AND_REDUCE</b> (lv   uv)	UX01
<b>OR_REDUCE</b> (lv   uv)	UX01
<b>NOR_REDUCE</b> (lv   uv)	UX01
<b>XOR_REDUCE</b> (lv   uv)	UX01
<b>XNOR_REDUCE</b> (lv   uv)	UX01

## 8. EXEMPLAR'S STD\_LOGIC\_ARITH

### 8.1 OVERLOADED OPERATORS

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

### 8.2 PREDEFINED FUNCTIONS

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

### 8.3 CONVERSION FUNCTIONS

From	To	Function
bool	uv	<b>bool2elb</b>
uv	bool	<b>elb2bool</b>
u/l	na	<b>even2int</b>
in	u/l	<b>int2even</b> (size)
uv	na	<b>elb2int</b>

## 9. MENTOR'S STD\_LOGIC\_ARITH

### 9.1 PREDEFINED TYPES

<b>UNSIGNED</b> (na to   downto na)	Array of STD_LOGIC
<b>SIGNED</b> (na to   downto na)	Array of STD_LOGIC

### 9.2 OVERLOADED OPERATORS

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

### 9.3 PREDEFINED FUNCTIONS

<b>ZERO_EXTEND</b> (uv   lv   un, na)	same
<b>ZERO_EXTEND</b> (u/l, na)	lv
<b>SIGN_EXTEND</b> (sg, na)	sg
<b>AND_REDUCE</b> (uv   lv   un   sg)	u/l
<b>OR_REDUCE</b> (uv   lv   un   sg)	u/l
<b>XOR_REDUCE</b> (uv   lv   un   sg)	u/l

### 9.4 CONVERSION FUNCTIONS

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

© 1995 - 2000 Qualis Design Corporation. Permission to reproduce and distribute strictly verbatim copies of this document in whole is hereby granted.

### Qualis Design Corporation

Elite Training and Consulting in Reuse and Methodology

Phone: +1.503.670.7200

FAX: +1.503.670.0809

Email: info@qualis.com

Web: http://www.qualis.com

Also available:

VHDL Quick Reference Card

Verilog Quick Reference Card

# Comentários Finais

- No final desta aula deverá ser capaz de:
  - Utilizar as construções de VHDL
    - **for...generate**
    - **if...generate**
  - Conhecer alguns dos atributos pré-definidos de VHDL
  - Consultar em *reference cards* de VHDL as construções da linguagem abordadas em LSDig