Laboratório de Sistemas Digitais

Aula Teórico-Prática 11

Ano Letivo 2020/21

As construções "**`for`**…**`generate`**" e "**`if`**…**`generate`**" em VHDL

Atributos pré-definidos em VHDL

Reference cards de VHDL e STD_LOGIC_1164

Universidade de Aveiro

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) concurrente(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*)

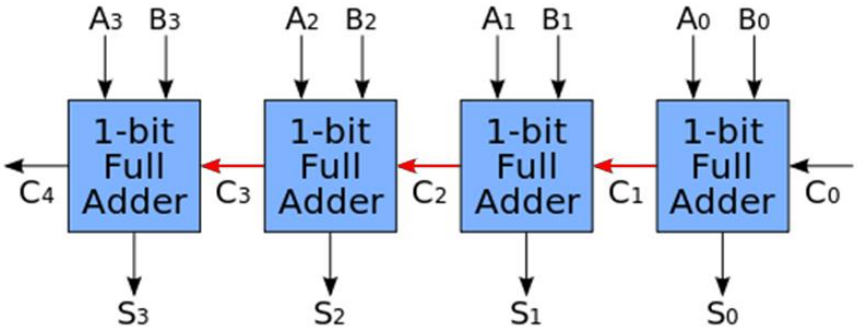
A construção **`for`…`generate`** deve ser escrita no corpo de uma arquitetura (e fora de processos!)

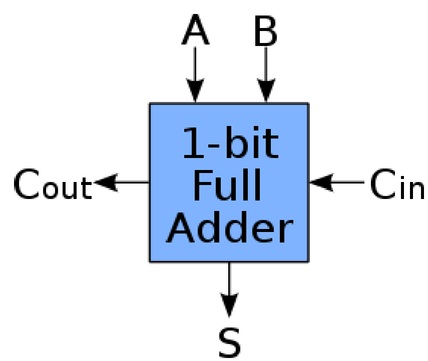
Também pode ser escrita na forma:

`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 completos 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;
```

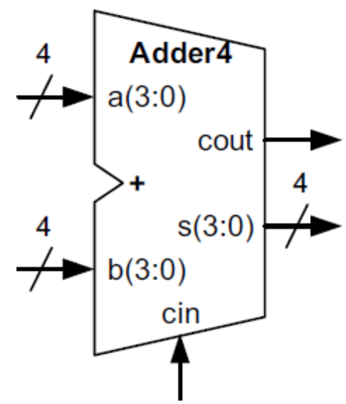
Figura 2 – Esqueleto do código VHDL da entidade **FullAdder** e respetiva arquitetura **Behavioral**.

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;
```

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

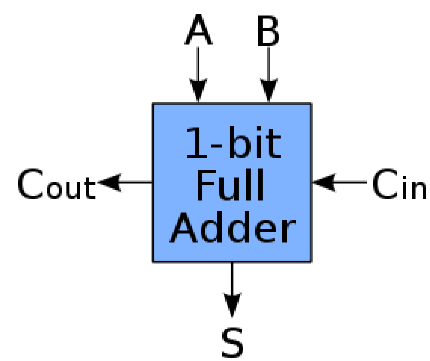


E se o somador tivesse 64 bits?

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

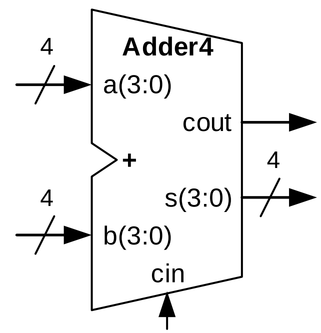
Código Completo de Somador Completo de 1 bit

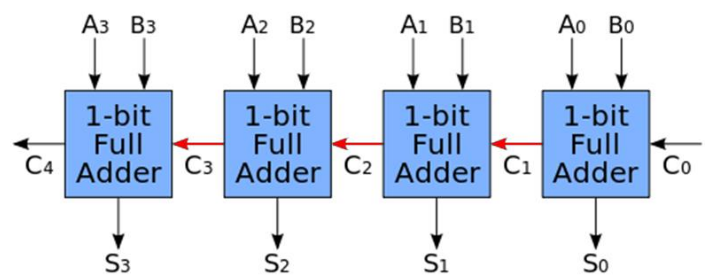
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

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

architecture Behavioral of FullAdder is
begin
	s    <= a xor b xor cin;
	cout <= (a and b) or (a and cin) or (b and cin);
end Behavioral;
```



Instanciação de 4 Somadores Completos de 1 bit com **`for…generate`**

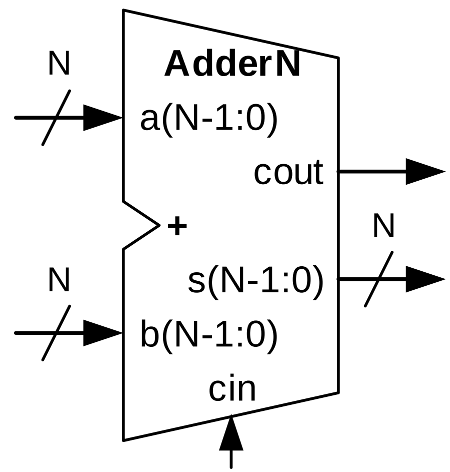
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

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

    signal s_carry : std_logic_vector(4 downto 0);

begin
    s_carry(0) <= cin;

    adder_loop : for i in 0 to 3 generate
        cell: entity work.FullAdder(Behavioral)
            port map(a     => a(i),
                     b     => b(i),
                     cin   => s_carry(i),
                     s     => s(i),
                     cout  => s_carry(i+1));
    end generate;

    cout <= s_carry(4);
end Structural;
```





Construção de um Somador Parametrizável de N bits com **`for…generate`**

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

entity AdderN is
    generic(N : integer := 8);
    port(a, b : in  std_logic_vector(N-1 downto 0);
         cin  : in  std_logic;
         s    : out std_logic_vector(N-1 downto 0);
         cout : out std_logic);
end AdderN;

architecture Structural of AdderN is

    signal s_carry : std_logic_vector(N downto 0);

begin
    s_carry(0) <= cin;

    adder_loop : for i in 0 to (N-1) generate
        cell: entity work.FullAdder(Behavioral)
            port map(a    => a(i),
                     b    => b(i),
                     cin  => s_carry(i),
                     s    => s(i),
                     cout => s_carry(i+1));
    end generate;

    cout <= s_carry(N);
end Structural;
```

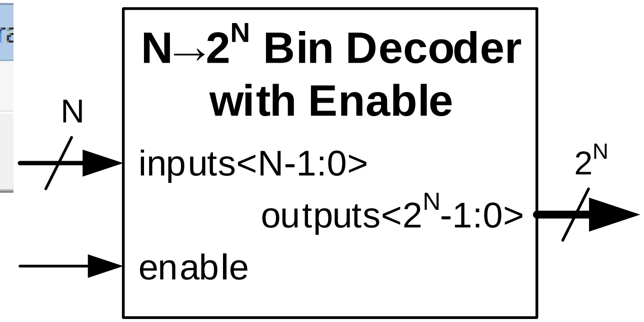


Exemplo de `for…generate` com uma Atribuição Condicional

- Descodificador binário de N→2^N bits, com N parametrizável

Text Editor - C:/Users/asroliveira/CloudStation/LSDig2017/SlidesTP/GenerateTest/Genera

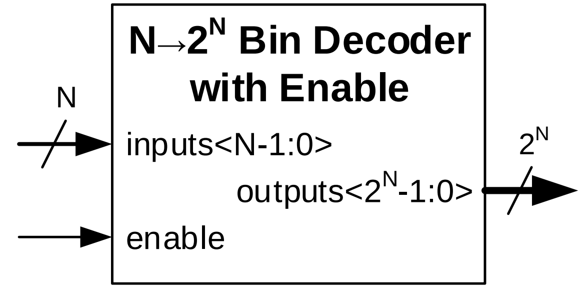
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity BinDecoderN is
  generic(N      : positive := 4);
  port(enable  : in  std_logic;
       inputs  : in  std_logic_vector(N-1 downto 0);
       outputs : out std_logic_vector((2**N)-1 downto 0));
end BinDecoderN;

architecture BehavAssign of BinDecoderN is
begin
   out_loop : for i in 0 to ((2**N) - 1) generate

      outputs(i) <= enable when (i = to_integer(unsigned(inputs))) else
                   '0';

   end generate;
end BehavAssign;
```



Exemplo de **`for…generate`** com um Processo

- Descodificador binário de N→2^N bits, com N parametrizável

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

entity BinDecoderN is
   generic(N       : positive := 4);
   port(enable  : in  std_logic;
        inputs  : in  std_logic_vector(N-1 downto 0);
        outputs : out std_logic_vector((2**N)-1 downto 0));
end BinDecoderN;

architecture BehavProc of BinDecoderN is
begin
   out_loop : for i in 0 to ((2**N) - 1) generate

      process(enable, inputs)
      begin
         if (i = to_integer(unsigned(inputs))) then
            outputs(i) <= enable;
         else
            outputs(i) <= '0';
         end if;
      end process;

   end generate;
end BehavProc;
```



A construção **`for…generate`** também é útil para os projetos finais

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`**!

Universidade de Aveiro

Exemplo da Construção `if…generate` de VHDL

```
entity AdderN is
   generic ( N : positive := 4);
   port (a, b  : in std_logic_vector(N-1 downto 0);
         cin   : in std_logic;
         s     : out std_logic_vector(N-1 downto 0);
         cout  : out std_logic);
end AdderN;

architecture Structural of AdderN is
   signal s_carry : std_logic_vector(N-1 downto 1);
begin
   adder: for i in 0 to N-1 generate
   begin
   adder_cell: if i = N-1 generate -- most-significant cell
                  add_bit: entity work.FullAdder(Behavioral)
                  port map (a => a(i), b => b(i), s => s(i), cin => s_carry(N-1), cout => cout);
               elsif i = 0 generate -- least-significant cell
                  add_bit: entity work.FullAdder(Behavioral)
                  port map (a => a(i), b => b(i), s => s(i), cin => cin, cout => s_carry(i+1));
               else generate -- middle cell
                  add_bit: entity work.FullAdder(Behavioral)
                  port map (a => a(i), b => b(i), s => s(i), cin => s_carry(i), cout => s_carry(i+1));
   end generate adder_cell;
   end generate adder;
end Structural;
```

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

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**

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

`ARYID[nth]'left`	`Left-bound of [nth] index`
`ARYID[nth]'right`	`Right-bound of [nth] index`
`ARYID[nth]'high`	`Upper-bound of [nth] index`
`ARYID[nth]'low`	`Lower-bound of [nth] index`
`ARYID[nth]'range`	`'left down/to 'right`
`ARYID[nth]'reverse_range`	`'right down/to 'left`
`ARYID[nth]'length`	`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
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**]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**

See reverse side for additional information.

© 1995-2000 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 | *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

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
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)

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 LSD