1. Introducción

VHDL un lenguaje fuertemente tipado y **case insensitive.**
Los comentarios empiezan por dos – y hasta final de línea.

2. Unidades de Compilación

Declaración de **Library Usage** -- ref. 11
Declaración de **Entity** -- ref. 3
Declaración de **Architecture** -- ref. 4
Declaración de **Package** -- ref. 10
Declaración de **Configuration** -- ref. 14

3. Declaración de Entity

```
entity n_input_nand is
generic (n:integer := 2);
port (data :  in bit_vector( 1 to n);
       result :out bit );
end n_input_nand;
-- modo : in |out |inout |buffer |linkage
```

4. Declaración de Architecture

```
architecture behave of n_input_nand is
-- declaraciones                    -- ref. 7
begin
-- instrucciones concurrentes       -- ref. 9
end behave;
```

5. Operadores

lógicos :	**and, or, xor, nand, nor, xnor, not**
relacionales **:**	**=, /=, <, <=, >, >=**
shift left/right logicos :	**sll, srl**
shift left/rightaritmeticos:	**sla, sra**
rotar left/right logicos:	**rol, ror**
matematicos :	**+, -, *, **, /, mod, abs, rem**
concatenacion:	**&**

eg. &:concatenation, '1' **&**"10" ="110"
****** :exponentiation, 2****** 3=8
rem :remainder, 7**rem** 2=1
mod :division modulo, 5**mod** 3=2

6. Tipos de Datos

6.1 Tipos de Datos Predefinidos

bit	'0' y '1'	**integer**	32 bit con signo
bit_vector	array de "bit"	**natural**	integer >= 0
boolean	true y false	**positive**	integer > 0
character	7 bit ASCII	**real**	+1e38 a –1e38
string	array de caracteres		
time	hr,min,sec,ms,us,ns,ps,fs		

6.2 Tipos definibles por usuario

```
type distancia  is range 0 to 100000
 units
    meter;                -- unidad base
   kilometer =1000 meter;
end units distancia;
type number  is integer;
type voltage  is range 0 to 5;
type current is range 1000 downto 0;
type d_bus is array (range <> ) of  bit;
type instruccion is record
    opcode :bit;
    operand :bit;
end record;
type int_file is file of integer;
type pointer_to_integer is access integer;
subtype positive_number is integer range 0 to 100000
type fourval is (X, L, H, Z);
subtype resolve_n  is resolve twoval;
```

7. Declaraciones

```
constant bus_width :integer := 32;
variable read_flag :bit := 0;
--solo en procesos y subprogramas
signal clock : bit;
file f3 :int_file open write_mode is "test.out";
alias enable : bit is addr(31);
attribute delay :time;
component n_input_nand
  generic (n:integer := 2);
  port (data :in bit_vector (1to n);
         result :out bit );
end component n_input_nand;
function square (i:integer )return integer;
for store :use configuration latch;
```

8. Atributos

```
-- type my_array is array (9downto 0) of any_type;
-- variable an_array :my_array;
-- type fourval is ('0', '1', 'Z', 'X' );
-- signal sig :sigtype;
-- constant T:time := 10 ns;
```

Attribute	**Result type**	**Result**
my_array**'high**	any_type	9
my_array**'left**	any_type	9
my_array**'low**	any_type	0
my_array**'right**	any_type	0
my_array**'ascending**	boolean	false
my_array**'length**	integer	10
my_array**'range**	integer	9 downto 0
my_array**'reverse_range**	integer	0 to 9
fourval**'leftof**('0')	fourval	error
fourval**'leftof**('1')	fourval	'0'
fourval**'pos**('Z')	integer	2
fourval**'pred**('1')	fourval	'0'
fourval**'rightof**('1')	fourval	'Z'
fourval**'succ**('Z')	fourval	'X'
fourval**'val**(3)	fourval	'X'

sig**'active**	Boolean	Verdad si actividad en sig
sig**'delayed(T)**	sigtype	Copia de sig retrasada en T
sig**'driving_value**	sigtype	Valor del driver de sig
sig**'event**	Boolean	Verdad si evento en sig
sig**'last_active**	time	Tiempo desde ultima actividad
sig**'last_event**	time	Tiempo desde el ultimo evento
sig**'last_value**	sigtype	Valor antes del ultimo evento
sig**'quiet(T)**	Boolean	Actividad (now –T) to now
sig**'stable(T)**	Boolean	Evento (now –T) to now
sig**'transactio**	bit	Cambia la actividad de sig

9. Instrucciones

9.1 Instruccione concurrentes.

Asignación a señal simple:
Nombre_Señal <= expresión;

Asignación a señal condicional:
Nombre_señal <= expresión **when** expresión booleana **else**
expresión **when** expresión booleana **else**
expresión **when others;**

Asignación a señal selectiva:

```
with expresión select
  nombre_señal <=  valor when selección,
                   valor when selección;
```

```
state_mach :process (state ) –La etiqueta es opcional
-- declaración de variables      -- ref. 7
begin
-- instrucciones secuenciales    -- ref. 9
end process;
```

```
U1_n_input_nand :n_input_nand
generic map (n=> 2)
port map (data => my_data; result => my_res );
```

```
top_block :block
-- declaraciones                 -- ref. 7
begin
-- instrucciones concurrentes    -- ref. 9
end block;
```

```
label1 :fori i  in 1 to 3 generate
label2 :nand2( a(i), b(i), c(i) );
end generate
```

```
label3 :if (i<4) generate
      label4 :nor2( a(i), b(i), c(i) );
end generate;
```

9.2 Instrucciones Secuenciales

```
null;       -- no hace nada
wait on sig1, sig2 until (sig ='1' ) for 30 ns;
wait until (clock'event and clock ='1' );
read_flag := 0;  -- asignación de variable

if (x<y) then max := y;
elsif (x>y)then max := x;         -- opcional
else max := x; -- optional
end if;

case a is
   when '1' | '0' => d <= '1';
   when 'Z'       => d<= '0';
   when others  => d<= 'X';   -- opcional
end case;

while (x<y) loop
     next  when (x>5);       -- uso de next
      x:= x+1;
end loop;

for iin (0to 100 ) loop
      x:= x+i;
     exit  when (x=0);        -- uso de  exit
end loop;
```

9.3 Instrucciones Concurrentes y Secuenciales

```
enable <= select after 1ns;
 assert (a=b)
 report "ais not equal to b"
 severity note;
-- severity levels :note |warning |error |failure
```

10. declaración de Package

```
package two_level is
-- Declaracion de  type, signal, functions       -- ref. 7
end two_level;

package body two_level is
-- definición de subprogramas                    -- ref. 7
end two_level;
```

11. Uso de librerías

```
library work;
use work.two_level.all;          -- todos los objetos
use work.two.level.vcc;          -- solo el objeto  "vcc"
```

12. Subprogramas

```
function bool_2_2level (boolean :in_bool ) return two_level is
   variable return_val :two_level;
begin
if (in_bool =true )then
  return_val := high;
else return_val := low;
end if;
return return_val;
end bool_2_2level;

procedure clock_buffer
(
signal local_clk :inout bit;
signal clk_pin :in bit;
constant clock_skew :in time )is
begin
-- ejemplo de efectos en   un procedure

global_clk <= local_clk after clk_skew;
local_clk <= clk_pin;
end clock_buffer;
```

13. Subprogramas predefinidos

```
variable ptoi : pointer_to_integer;
ptoi := new integer;    -- uso de  new
deallocate (ptoi );

variable status :file_open_status;
file my_file :int_file;
file_open( status, my_file, "in.dat", read_mode );
end_file (my_file );  -- devuelve  true/false
 variable int_var :integer;
read (my_file, int_var );
file_close (my_file );
```

14. Declaracion de Configuration

```
configuration input_8 of n_nand is
      for customizable
         for a1 : nand_2
           use entity work..nand_2(n_nand_arch );
         end for;
    end for;
end input_8;
```

15. Construcciones NO-Sintetizables

La mayoría de herramientas no sintetizan:
access, after, alias, assert, bus, disconnect, file, guarded, inertial, impure, label, linkage, new, on, open, postponed, pure, reject, report, severity, shared, transport, units, with.

16. Standard Packages

Ejemplos de paquetes standar .

16.1 IEEE.STD_LOGIC_1164 Package

```
type std_ulogic is ('U', 'X', '0', '1', 'W', 'L', 'H' );-- MLV9
type std_ulogic_vector is array(natural range <> ) of std_ulogic;
function resolved (s:std_ulogic_vector )return std_ulogic;
subtype std_logic is resolved std_ulogic;
type std_logic_vector is array(natural range <> ) of std_logic;
function to_bit (s:std_ulogic; xmap :bit := '0' ) return bit;
function to_bitvector(s:std_logic_vector; xmap :bit := '0' )
return bitvector;
function to_stdlogicvector (b:bit_vector ) return td_logic_vector;
function rising_edge (signal s:std_ulogic ) return boolean;
function falling_edge (signal s:std_ulogic ) return boolean;
function is_x(s:std_logic_vector ) return boolean;
```

16.2 STD.TEXTIO Package

```
type line access string;
type text is file of string;
type side is (right, left );
subtype width is natural;
file input :text open read_mode is "std_input";
file output :text open write_mode is "std_output";
procedure readline (file f:text; I:out line );
procedure writeline (file f:text; I:in line );
procedure read (I:inout line;value :out bit;good :out boolean );
procedure write (I:inout line;value :in bit;justified :in side := right;field
:in width := 0);
-- El tipo de  "value" puede ser  bit_vector |boolena |-- character |
integer |real |string |time.

-- En el package estandard no estan contempladas  operaciones  textio
-- std_logic. Cada vendedor suministra las suyas.
```

16.3 IEEE.NUMERIC_STD Package

```
type unsigned is array (natural range <> ) of std_logic;
type signed is array (natural range <> ) of std_logic;
function shift_left (arg :unsigned; count :natural ) return  nsigned;
-- Otras funciones : shift_right(), rotate_left(),
rotate_right()
function rsize (arg :signed; new_size :natural ) return signed;
```

Super Txuleta de VHDL

Sistemas Electrónicos Digitales

Profesor: Mikel San Miguel