

```

for at : nand_2
  use entity work.nand_2( n_nand_arch );
end for;
end for;
end input_8;

```

15. Non-synthesizable Constructs

Most tools will not synthesize :
access, **after**, **alias**, **assert**, **bus**, **disconnect**, **file**,
guarded, **inertial**, **impure**, **label**, **linkage**, **new**,
on, **open**, **postponed**, **pure**, **reject**, **report**,
severly, **shared**, **transport**, **units**, **with**.

16. Standard Packages

Samplings of a subset of standard packages the language provides.

16.1 IEEE.STD_LOGIC_1164 Package

```

type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'V');
--MVL9

```

```

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 bit_vector;
function to_stdlogicvector( b : bit_vector )
  return std_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 is 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; l : out_line );
procedure writeline( file f : text; l : in_line );
procedure read( l : inout_line;
  value : out_bit;
  good : out_boolean );
procedure write( l : inout_line;
  value : in_bit;
  justified : in_side := right;
  field : in_width := 0 );
-- The type of "value" can be bit_vector | boolean |
-- character | integer | real | string | time.
-- There is no standard package for textio operations
-- on std_logic. Tool vendors may provide their own.

```

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 unsigned;
-- other functions: shift_right(), rotate_left(),
rotate_right()
function rsize ( arg : signed;
  new_size: natural )
  return signed;

```

VHDL Quick Reference Card is intended for quick reference.
Please use VHDL LRM of 1993 for details.

Better ROI

Technology Tracking
Infrastructure Mgmt.
Training & Attrition Mgmt.
Process Guarantee
Project Responsibility


+

Engineering (Design, Development, Test)
Chips, Boards, Software & Systems

= VALUE ADDITION

↓

1 = Pre-manufacturing
2 = Post-definition



COMIT SYSTEMS
The Contract Engineering Company
www.comit.com

Download White Paper to find out how.



VHDL Quick Reference Card

1. Introduction

VHDL is a **case insensitive** and **strongly typed** language. Comments start with two adjacent hyphens (--) and end at the end of line.

2. Compilation Unit

```

Library Usage Declarations -- ref. 11
Entity Declarations -- ref. 3
Architecture Declarations -- ref. 4
Package Declarations -- ref. 10
Configuration Declarations -- ref. 14

```

3. Entity Declaration

```

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

```

4. Architecture Declaration

```

architecture behave of n_input_nand is
  -- declarations -- ref. 7
begin
  -- concurrent statements -- ref. 9
end behave;

```

5. Operators

logical operators : **and**, **or**, **xor**, **nand**, **nor**, **xnor**, **not**
relational operators : =, /=, <, <=, >, >=
shift left/right logical operators : **sl**, **srl**
shift left/right arithmetic operators : **sla**, **sra**
rotate left/right logical operators : **rol**, **ror**
other operators : +, -, &, *, **, /, mod, abs, rem
eg. & : concatenation, '1' & '10' = '110'
** : exponentiation, 2 ** 3 = 8
rem : remainder, 7 rem 2 = 1
mod : division modulo, 5 mod 3 = 2

6. Data Types

6.1 Predefined Data Types

```

bit '0' and '1'
bit_vector Array of "bit"
boolean true and false
character 7-bit ASCII
integer signed 32 bit at least
natural integer >= 0
positive integer > 0
real Floating point, min: +1e38 to -1e38

```

```

string      Array of characters.
time       hr, min, sec, ms, us, ns, ps, fs
file_open_kind  read_mode, write_mode, append_mode
file_open_status  open_ok, status_error, name_error,
mode_error
(MIDL 93)

```

6.2 User Defined Data Types

- **type distance is range 0 to 100000**
units
meter;
kilometer = 1000 meter;
end units distance;
- **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 instruction 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. Declarations

- **constant** bus_width : integer := 32;
- **variable** read_flag : bit := '0'; -- only in processes
-- and subprograms
- **signal** clock : bit;
- **file** f3 : in_file open write_mode is "test.out";
- **alias** enable : bit is addr(31);
- **attribute** delay : time;
- **attribute** delay of my_signal : signal is 8 ns;
- **component** n_input_nand
generic (n : integer := 2);
port (data : in bit_vector (1 to n);
result : out bit);
- **end component** n_input_nand;
- **function** square (i : integer) return integer;
- **for** store : use configuration latch;

8. Attributes

- **type** my_array is array (9 downto 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 |
| an_array/length | integer | 10 |
| an_array/range | integer | 9 downto 0 |
| an_array/reverse_range | integer | 0 to 9 |
| fourval/leftof('0') | fourval | error |
| fourval/rightof('1') | fourval | '0' |

```

fourval_pos(Z)      integer      2
fourval_pred('1')  fourval      '0'
fourval_rightof('1') fourval    'X'
fourval_succ(Z)    fourval    'X'
fourval_val(3)     fourval    'X'
sig_active          boolean     True if activity on sig
sig_delayed(T)     sigtype      Copy of sig delayed by T
sig_driving_value  sigtype      Value of driver on sig
sig_event          boolean     True if event on sig
sig_last_active    time         Time since last activity
sig_last_event     sigtype      Value before last event
sig_last_value     boolean     Activity ( now - T ) to now
sig_quiet(T)       boolean     Event ( now - T ) to now
sig_stable(T)      boolean     Toggles on activity on sig
sig_transaction    bit

```

9. Statements

- ### 9.1 Concurrent Statements
- **state_mach : process (state)** -- label is optional
-- variable declarations -- ref. 7
begin
-- sequential statements -- 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** -- declaration -- ref. 7
begin
-- concurrent statements -- ref. 9
end block;
 - **label1 : for 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 Sequential Statements
- **null;** -- does nothing
 - **wait on** sig1, sig2 until (sig = '1') for 30 ns;
 - **wait until** (clock/event and clock = '1');
 - **read_flag := 0;** -- read_flag is a variable
 - **if (x < y) then** max := y; -- max is a variable
 - **elsif (x > y) then** max := x; -- optional
 - **else** max := x; -- optional
 - **end if;**
 - **case** a is
when '1' | '0' => d <= '1'; -- d is a signal
when 'Z' => d <= '0';
when others => d <= 'X'; -- optional
end case;
 - **while (x < y) loop**
next when (x > 5);
x := x + 1;
end loop;
 - **for** i in (0 to 100) loop
x := x + 1;
exit when (x = 0);
end loop;

9.3 Concurrent and Sequential Statements

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

10. Package Declarations

- **package** two_level is
-- type, signal, function declarations -- ref. 7
end two_level;
- **package** body two_level is
--subprogram definitions
end two_level;

11. Library Usage Declarations

- using the two_level package.
- library** work;
- use** work.two_level.all; -- all objects used
- use** work.two_level.vcc; -- only the "vcc" object used

12. Subprograms

```

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
-- example of side effects in a procedure
global_clk <= local_clk after clk_skew;
local_clk <= clk_pin;
end clock_buffer;

```

13. Predefined Subprograms

- **enable** <= '1' when (now < 2 ns) else '0';
- **variable** ptoi : pointer_to_integer;
- **ptoi** := new integer;
- **deallocate**(ptoi);
- **variable** status : file_open_status;
- **file** my_file : int_file;
- **file_open**(status, my_file, "in.dat", read_mode);
- **endfile**(my_file);
- **variable** int_var : integer;
- **read**(my_file, int_var);
- **file_close**(my_file);

14. Configuration Declarations

- **configuration** input_8 of n_nand is
for customizable