VHDL Primer

Sections:

I. Reserved Identifiers
II. Expressions, Operators and Name
III. Syntax Reference
IV. Behavioral description
V. Architecture
VI. Arrays

Reserved Identifiers

abs exit not signal
access file null shared
after for of sla
alias function on sll
all generate open sra
and generic or srl
architecture group others subtype
array guarded out then
assert if package to
attribute impure port transport
begin in postponed type
block inertial procedure unaffected
body inout process units
buffer is pure until
bus label range use
case library record variable
component linkage reject register
configuration literal rem when
constant loop reject rem while
disconnect map report with
downto mod return xnor
delay label report xor
doi new select
entity nor severity
Expressions, Operators and Names

-- indicate comments
() TypeName()
** abs not
* / mode rem
+ - &
sll srl sla sra rol ror
= /= < <= > >=
and nand or nor xor xnor
123 1_2_3 1e6 2#1110# 16#FF#
"0101" O"77" X"FF"
ABC def Ghi A123 A_B_C
Name(Expr)
Name(Expr1 to Expr2)
T'Low T'High T'Image
A'Range A'Reverse_Range
S'Event S'Stable(T) S'Delayed(T)
E'Path_Name
package Pack is
    type Enum is (Unknown, '0', '1');
    type Int is range 0 to 255;
    type float is range 0.0 to 1.0;
    type Byte is array (7 downto 0) of Bit;
    type Mem is array (Integer range <> ) of Int;
    type Intx is record
        Value : Integer;
        Defined : Boolean;
    end record;
    constant C1 : Int := 255;
    constant CV2 : Mem (0 to 511) := (1, 2, 3, others => 4);
    procedure P (Const : T; Var: out T; signal Sig : inout T);
    function "+" (L, R: T) return T;
end Pack;

package body Pack is
    procedure P (Const : T; Var: out T; signal Sig : inout T);
        begin
        -- sequence of statements
    end P;
    function "+" (L, R: T) return T is
        begin
        -- sequence of statements
            return Expr;
        end "+";
    end Pack;

library Lib;
use Lib.Pack.all;

entity Ent is
    generic (G : Time := 0 ns);
    port ( P1 in T;
          P2 : out T := '0';
          P3, P4 : inout T);
end Ent;
Behavioral description

architecture A 1 of Ent is
  signal Sig1, Sig2: Typ := Init;
begin
  Proc: process (Ck, D)
  -- declarations
  begin
  -- sequence of statements
  end process Proc;

  process
  variable Var : Typ := Init;
  begin
  wait until Rising_edge (Ck);
  wait for 10 ns;
  wait;
  Sig <= Expr after Delay;
  Sig <= Expr1 after D1, Expr2 after D2, Expr3 after D3;
  Var := Expr;
  case C is
  when C1 =>
  -- sequence of statements
  when C2 | C3 | C4 to C8 =>
  -- sequence of statements
  when other =>
    null;
  end case;
  if Reset then
  -- sequence of statements
  elsif Ck'Event and Ck = '1' then
  -- sequence of statements
  else
  -- sequence of statements
  end if;
  for i in 1 to N loop
  -- sequence of statements
  exit
  -- sequence of statements
  end loop;
  while C loop
  -- sequence of statements
  end loop;
  assert D'stable (10 ns)
  report "Setup error"
  severity warning;
  report "End of simulation"
end process;

  L1 : Sig1 <= A + B after 100 ns;
  L2 : Sig2 <= A * B after 1 us;
end A1;
architecture A2 of Ent is
    component Comp
        generic (G: Time := 0 ns);
        port (A, B: in T := '0';
            F : out T);
    end component;
signal S1, S2, S3: Typ := Init;
begin
    L0 entity Lib.Ent2(Arch)
        port map (S1, S2, S3);
    L1: Comp port map (S1, S2, S3);
    L2: Comp generic map (G => 5 ns)
        port map (S1, S2, S3);
end A2;

configuration Cfg1 of Ent is
    for A1
    end for;
end Cfg1;

library Lib;
use Lib.all;
configuration Cfg2 of Ent is
    for A2
        for all: Comp
            use configuration Lib.Cfg3;
        end for;
    end for;
end Cfg2;
Architecture

Defines the internal view of a block of hardware, i.e. the functionality, behaviour or structure of the hardware. Belongs with an entity, which defines the interface. An entity may have several alternative architectures.

Syntax

```
architecture ArchitectureName of EntityName is
    Declarations...
begin
    ConcurrentStatements...
end ArchitectureName;
```

Rules

All the architectures of a particular entity must have different names, but the architectures of two different entities can have the same name.

Example

```
architecture BENCH of TEST_MUX4 is
    subtype V2 is STD_LOGIC_VECTOR(1 downto 0);

    -- Component declaration...
    component MUX4
        port (SEL, A, B, C, D: in V2;
            F : out V2);
    end component;

    -- Internal signal...
    signal SEL, A, B, C, D, F: V2;

    begin
        P: process
            begin
                SEL <= "00";
                wait for 10 NS;
                SEL <= "01";
                wait for 10 NS,
                SEL <= "10";
                wait for 10 NS,
                SEL <= "11";
                wait for 10 NS;
                wait;
                end process P;

        -- Concurrent assignments...
        A <= "00";
        B <= "01";
        C <= "10";
        D <= "11";

        -- Component instantiation...
        M: MUX4 port map (SEL, A, B, C, D, F);

    end BENCH;
```
Arrays

A data type which consists of a vector or a multi-dimensional set of values of the same base type. Can be used to describe RAMs, ROMs, FIFOs, or any regular multi-dimensional structure.

Syntax

type NewName is -- unconstrained
array (IndexTypeName range <> , ...) of DataType;

type NewName is -- constrained
array (Range, ...) of DataType;

Rules

a) The base type *DataType* must not be an unconstrained array type.
b) A signal or variable cannot be an unconstrained array, unless it is a generic, port or parameter.

tips

a) Large arrays should be variables or constants, rather than signals. A large signal array would be inefficient for simulation.
b) The values within an array can be read or written using an indexed name or a slice name.

Example

subtype Word is Std_logic_vector(15 downto 0);
type Mem is array (0 to 2**12-1) of Word;
variable Memory: Mem := (others => Word'(others=>'U'));

... 

if MemoryRead then
    Data <= Memory(To_Integer(Address));
elsiF MemoryWrite tHen
    Memory(To_Integer(Address)) := Data;
end if;