ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ

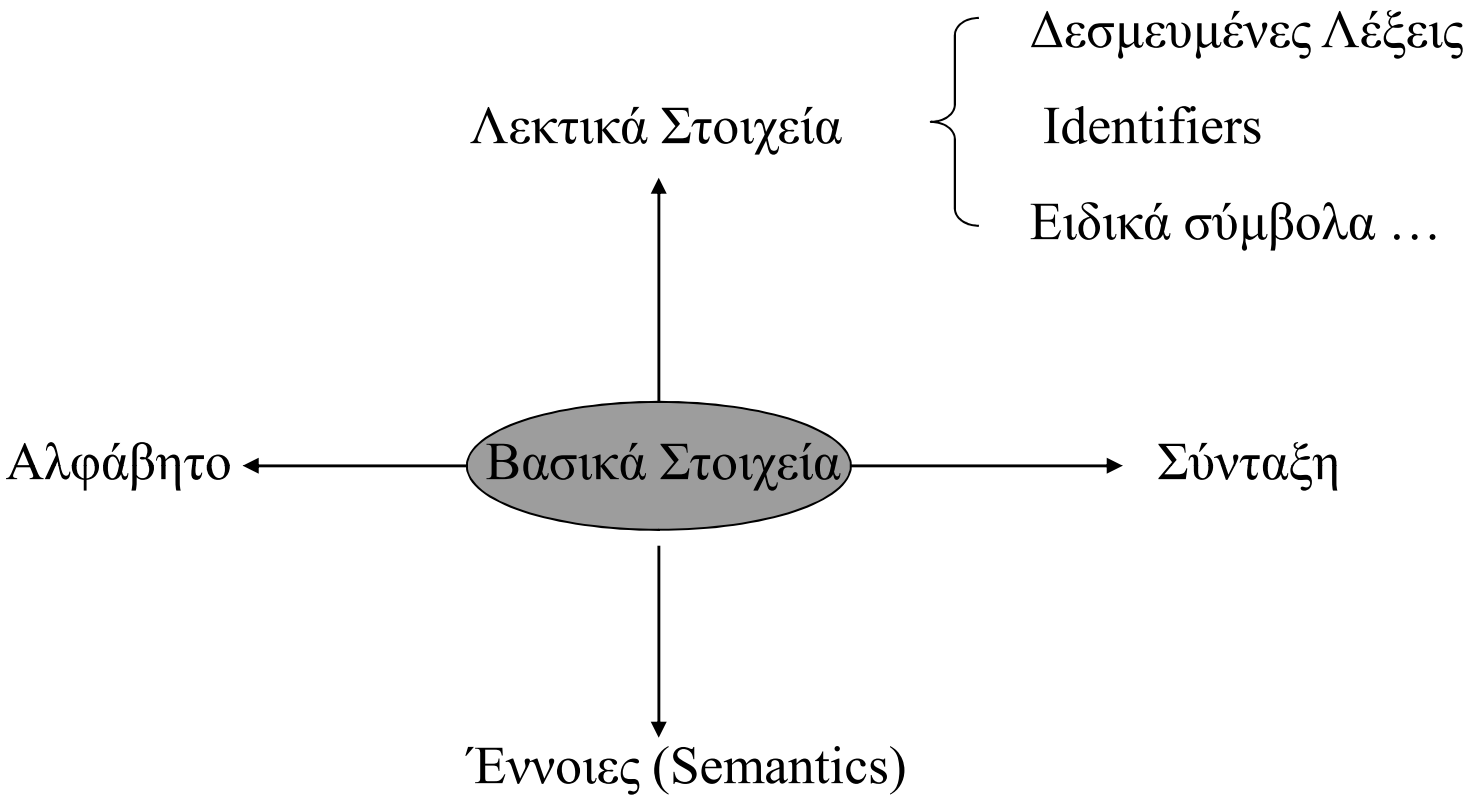
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

Βασικά στοιχεία γλώσσας περιγραφής κυκλωμάτων VHDL

(Peter Ashenden, The Students Guide to VHDL)

Βασικά στοιχεία VHDL



Λεκτικά στοιχεία της VHDL

Αλφάβητο: Όλοι οι χαρακτήρες του ISO-8 bit (μικρά, κεφαλαία, με διακριτικά – ä κλπ).

Identifiers - Βασικοί Κανόνες:

- Μπορούν να περιέχουν μόνο αλφαριθμητικούς χαρακτήρες (a…z, A…Z, 0…9) όπως και τον χαρακτήρα underscore '_'.
- Ο πρώτος χαρακτήρας πρέπει να είναι γράμμα και ο τελευταίος δεν πρέπει να είναι ο '_'.
- Δεν επιτρέπονται δύο συνεχόμενα '_'.
- Δεν έχει σημασία εάν τα γράμματα είναι μικρά ή κεφαλαία (and2=AND2=And2=…).
- Κάθε identifier μπορεί να έχει οποιοδήποτε μήκος.
- Δεν επιτρέπονται οι ειδικοί χαρακτήρες: " # & ~ ' ( ) * + , { } - / : ; < > | ! ...

Λεκτικά στοιχεία της VHDL

Παραδείγματα μη έγκυρων αναγνωριστικών:

- 8_bit_counter: ξεκινάει με ψηφίο
- A+B: χρησιμοποιεί τον ειδικό χαρακτήρα +
- Bit_vector_: χρησιμοποιεί τον χαρακτήρα υπογράμμισης στο τέλος
- 8_bit__counter: χρησιμοποιεί δύο διαδοχικούς χαρακτήρες υπογράμισης

Ετικέτες

Χρησιμοποιούνται για να δηλώσουν προτάσεις, διεργασίες, απεικονίσεις θύρας κλπ.

Λεκτικά στοιχεία της VHDL

Εκτεταμένοι Κανόνες:

- Ένας εκτεταμένος identifier περικλείεται ανάμεσα σε χαρακτήρες '\' και έχει σημασία εάν είναι με κεφαλαία ή μικρά.
- Κάθε εκτεταμένος identifier μπορεί να είναι ίδιος με μία δεσμευμένη λέξη.
- Ανάμεσα στους δύο χαρακτήρες '\' μπορεί να χρησιμοποιηθεί οποιοσδήποτε χαρακτήρας σε οποιαδήποτε σειρά, εκτός από τον ίδιο τον '\'.
- Σε περίπτωση που θέλουμε να χρησιμοποιήσουμε τον χαρακτήρα '\' θα πρέπει να τοποθετήσουμε άλλον έναν πριν από αυτόν. Για παράδειγμα το όνομα Adder\output θα γραφεί \Adder\\output\.

Σχόλια: Με δύο συνεχόμενες παύλες αγνοείται το περιεχόμενο της υπόλοιπης γραμμής.

Λεκτικά στοιχεία της VHDL

Δεσμευμένες λέξεις

abs	entity	next	select
access	exit	nor	severity
after	file	not	signal
alias	for	null	shared
all	function	of	sla
and	generate	on	sll
architecture	generic	open	sra
array	group	or	srl
assert	guarded	others	subtype
attribute	if	out	then
begin	impure	package	to
block	in	port	transport
body	inertial	postponed	type
buffer	inout	procedure	unaffected
bus	is	process	units
case	label	pure	untill

Λεκτικά στοιχεία της VHDL

Ειδικοί Χαρακτήρες:

Χρησιμοποιούνται για λόγους διαχωρισμού, ένωσης, στίξης και ως τελεστές: & ‘ ( ) * + , - . / : ; < = > |

και ως ζεύγη: => ** := /= >= <= <>

Χαρακτήρες:

- Ένας χαρακτήρας μπορεί να χρησιμοποιηθεί μέσα σε απόστροφους, πχ. ‘a’, ‘b’ κλπ.
- Ένα string τοποθετείται σε διπλές αποστρόφους όπως “Παράδειγμα”.
- Ένα bit string ορίζει μία ακολουθία από bits. Πριν το string τοποθετείται ένα από τα γράμματα B(bit), X(hexagonal), O(octal) για να δηλώσουν το είδος της αναπαράστασης πχ. B”11010101” ή X”D5” ή O”325”.

Κυριολεκτήματα (literals)

Κυριολεκτήματα:

- Αριθμητικά: 7 55_300(η υπογράμμιση αγνοείται) 3E2 4#322
- Χαρακτήρων: 'a'   '1'
- Αλφαριθμητικά: "abcd1"
- Δυαδικά ψηφία: B(binary), O(Octal), X(hexadecimal)

  X"fed" B"011" O"017"

Λεκτικά στοιχεία της VHDL

Αριθμοί:

- Το προκαθορισμένο σύστημα αναπαράστασης είναι το δεκαδικό.
- Μπορούν όμως να χρησιμοποιηθούν και πραγματικοί σε εκθετική ή κανονική μορφή (285.2 , 8.23E-5).
- Ένας αριθμός μπορεί να εκφραστεί και σε άλλη βάση (πλην του 10) με την σύνταξη base#number# (πχ 2#1001# είναι ο αριθμός 9 στο δυαδικό)

Τύποι Δεδομένων

Τύποι Δεδομένων:

- Ο τύπος ενός αντικειμένου καθορίζει το σύνολο των τιμών και το είδος των λειτουργιών του.
- Η VHDL είναι μία γλώσσα ισχυρών τύπων και απαιτεί κάθε αντικείμενο να έχει συγκεκριμένο τύπο.
- Δεν επιτρέπεται η ανάθεση μίας τιμής ενός τύπου σε ένα αντικείμενο άλλου τύπου εκτός αν γίνει ρητή μετατροπή τύπου.
- Τύποι δεδομένων: βαθμωτοί, σύνθετοι, αρχείων κλπ.
- Οι βαθμωτοί τύποι αναπαριστούν μία τιμή και διατάσσονται έτσι ώστε οι σχεσιακοί τελεστές να μπορούν να εφαρμοστούν σε αυτούς (ακέραιοι, πραγματικοί και τύποι απαρίθμησης)

Τύποι Δεδομένων

Types defined in the Package *Standard* of the *std* Library		
Type	**Range of values**	**Example**
bit	'0', '1'	signal A: bit :=1;
bit_vector	an array with each element of type bit	signal INBUS: bit_vector(7 downto 0);
boolean	FALSE, TRUE	variable TEST: Boolean :=FALSE'
character	any legal VHDL character (see package standard); printable characters must be placed between single quotes (e.g. '#')	variable VAL: character :='$';
file_open_kind*	read_mode, write_mode, append_mode	
file_open_status*	open_ok, status_error, name_error, mode_error	
integer	range is implementation dependent but includes at least $-(2^{31}-1)$ to $+(2^{31}-1)$	constant CONST1: integer :=129;
natural	integer starting with 0 up to the max specified in the implementation	variable VAR1: natural :=2;

Τύποι Δεδομένων

positive	integer starting from 1 up the max specified in the implementation	variable VAR2: positive :=2;
real*	floating point number in the range of -1.0×10^{38} to $+1.0 \times 10^{38}$ (can be implementation dependent. *Not supported by the Foundation synthesis program.*	variable VAR3: real :=+64.2E12;
severity_level	note, warning, error, failure	
string	array of which each element is of the type character	variable VAR4: string(1 to 12):= "@$#ABC*()_%Z";
time*	an integer number of which the range is implementation defined; units can be expressed in sec, ms, us, ns, ps, fs, min and hr. . *Not supported by the Foundation synthesis program*	variable DELAY: time :=5 ns;

Τύποι Δεδομένων

Τύποι Ακεραίων (integer):

- Το standard της VHDL απαιτεί ότι ο τύπος ακεραίου περιλαμβάνει τουλάχιστον τους αριθμούς από –2.147.483.647 έως +2.147.483.647
- Μπορεί να γίνει και ανάθεση ενός υπο-τύπου ακεραίου:

type day : **is range** 0 to 31;

(Η αρχική τιμή σε αυτήν την περίπτωση είναι 0 - η αριστερότερη)

Τύποι Κινητής Υποδιαστολής (real):

- Το standard της VHDL απαιτεί ότι περιλαμβάνει τουλάχιστον το εύρος από –1.0E+38 έως +1.0E+38 και τουλάχιστον 6 δεκαδικά ψηφία ακρίβεια
- Μπορεί να γίνει και ανάθεση ενός υπο-τύπου πραγματικού:

type level : **is range** –10.5 to 10.5;

Τύποι Δεδομένων

Φυσικοί Τύποι (physical):

- Αναπαριστούν πραγματικές ποσότητες.
- Έχουν κύρια μονάδα και πιθανώς δευτερεύουσες μονάδες που είναι πολλαπλάσια της κύριας μονάδας

```
type current is range 0 to 2E-12
units
        A;
        mA=1E-3A;
        uA=1E-6A;
        nA=1E-9A;
        pA=1E-12A;
end units current
```

Τα αποτελέσματα πράξεων με ακεραίους και πραγματικούς δίνουν τιμές ίδιου τύπου.

Προσοχή: η διαίρεση δύο ίδιων φυσικών τύπων δίνει ακέραιο ή πραγματικό (5mA / 2mA) = 2.5

Για πράξεις μεταξύ φυσικών τύπων είναι προτιμότερη η μετατροπή τους

Τύποι Δεδομένων

Τύπος Χρόνου (time):

- Είναι ιδιαίτερα σημαντικός στην VHDL καθώς αναπαριστά καθυστέρηση.
- Η κύρια μονάδα είναι το fs που είναι το όριο ακρίβειας εξομοίωσης

type time **is range** *implementation defined*
units
fs;
ps = 1000 fs;
ns = 1000 ps;
us = 1000 ns;
ms= 1000 us;
sec= 1000 ms;
min= 60 sec;
hr= 60 min;
end units time;

Τύποι Δεδομένων

Τύποι Απαρίθμησης: αποτελούνται από ακολουθίες χαρακτήρων και identifiers.

type logic_value **is** (‘0’, ‘1’, ‘2’, ‘Z’, ‘X’);

type instruction **is** (add, sub, store, div, mul);

type hex_digit **is** (‘0’,‘1’,‘2’,‘3’,‘4’,‘5’,‘6’,‘7’,‘8’,‘9’,‘A’,‘B’,‘C’,‘D’,‘E’,‘F’)

- Εάν η μεταβλητή δεν αρχικοποιηθεί από τον χρήστη τότε παίρνει την αριστερότερη τιμή της λίστας.
- Δύο διαφορετικοί τύποι μπορεί να έχουν και ίδια στοιχεία
- Ένας τύπος απαρίθμησης που χρησιμοποιείται πολύ συχνά στην VHDL είναι ο STD_ULOGIC που ορίζεται στο πακέτο std_logic_1164.

```
type STD_ULOGIC is (
        'U',        -- uninitialized
        'X',        -- forcing unknown
        '0',        -- forcing 0
        '1',        -- forcing 1
        'Z',        -- high impedance
        'W',        -- weak unknown
        'L',        --  weak 0
        'H',        -- weak 1
        '-');       -- don't care
```

Τύποι Δεδομένων

Τύπος Χαρακτήρα (Character):

- Περιλαμβάνει όλους τους χαρακτήρες στο σύνολο ISO-8bit
- Η δήλωση γίνεται με την λέξη **character** και η ανάθεση με αποστρόφους:

variable terminator: character := ‘A’;

Τύπος Boolean:

- Έχει δύο τιμές true – false και δίνεται ως αποτέλεσμα συγκρίσεων

123 = 123 δίνει τιμή true ενώ 123 > 123 δίνει τιμή false

Τύπος Bit:

- Έχει δύο τιμές ‘0’, ‘1’ και χρησιμοποιείται για την αναπαράσταση λογικών επιπέδων hardware (συμμετέχει σε λογικές εκφράσεις)

Τύποι Δεδομένων

Τύποι Οριζόμενοι από τον χρήστη:

type identifier **is** type_definition

Πάνω σε έναν οριζόμενο τύπο μπορεί να δηλωθεί ένα υποσύνολό του ως υπο-τύπος (subtype):

- Οι οριζόμενοι από τον χρήστη τύποι θα πρέπει είτε να μπουν στο σώμα της αρχιτεκτονικής και να οριστούν, ή να μπουν σε ένα πακέτο.
- Η δεύτερη προσέγγιση είναι η προτιμότερη, αφού δεν απαιτεί να ξαναγράφονται όταν χρησιμοποιούνται.

```
package user_types is
…
type declarations
…
end package user_types
```

Τύποι Δεδομένων

Υπο-Τύποι (Subtypes):

subtype small_int **is** integer **range** –128 to 127;

Ο υπο-τύπος χρησιμοποιείται για δήλωση πολλών μεταβλητών.

Type Qualification:

- Χρησιμοποιείται για τον διαχωρισμό του τύπου τιμών που ανήκουν σε περισσότερους από έναν τύπο (Overloaded)

type logic_level **is** (unknown, low, undriven, high);

type system_state **is** (unknown, ready, busy);

Ο διαχωρισμός γίνεται με τα statements

logic_level'(unknown) και system_state'(unknown)

Τύποι Δεδομένων

Μετατροπές Τύπων:

Η VHDL είναι γλώσσα ισχυρών τύπων (απαιτείται η μετατροπή τύπων)

Conversions supported by std_logic_1164 package	
Conversion	**Function**
std_ulogic to bit	to_bit(*expression*)
std_logic_vector to bit_vector	to_bitvector(*expression*)
std_ulogic_vector to bit_vector	to_bitvector(*expression*)
bit to std_ulogic	To_StdULogic(*expression*)
bit_vector to std_logic_vector	To_StdLogicVector(*expression*)
bit_vector to std_ulogic_vector	To_StdUlogicVector(*expression*)
std_ulogic to std_logic_vector	To_StdLogicVector(*expression*)
std_logic to std_ulogic_vector	To_StdUlogicVector(*expression*)

- Η σύνταξη για την μετατροπή τύπου είναι όνομα_τύπου(έκφραση)
- Απαραίτητη προϋπόθεση είναι να δίνει η έκφραση αποτέλεσμα το οποίο να μπορεί να μετατραπεί στον επιθυμητό τύπο.
- Τύποι απαρίθμησης δεν μπορούν να μετατραπούν.

Ιδιότητες Βαθμωτών Τύπων

Δίνουν πληροφορίες για τις τιμές ενός τύπου

Η σύνταξη είναι: όνομα τύπου <απόστροφος> όνομα ιδιότητας

Attribute	**Value**
scalar_type'**left**	returns the first or leftmost value of scalar-type in its defined range
scalar_type'**right**	returns the last or rightmost value of scalar-type in its defined range
scalar_type'**low**	returns the lowest value of scalar-type in its defined range
scalar_type'**high**	returns the greatest value of scalar-type in its defined range
scalar_type'**ascending**	True if T is an ascending range, otherwise False
scalar_type'**value(s)**	returns the value in T that is represented by s (s stands for string value).

Ιδιότητες Βαθμωτών Τύπων

```
type resistance is range 0 to 1E19
  units
        ohm;
        kohm = 1000 ohm;
        Mohm = 1000 kohm;
  end units resistance;
```

```
type set_index_range is range 21 downto 11;
type logic_level is (unknown, low, undriven,
high);
```

resistance'left = resistance'low = 0 ohm
resistance'right = resistance'high = 1E19 ohm
resistance'ascending = true
resistance'value("5 Mohm")=5000000ohm
set_index_range'left = set_index_range'high = 21
set_index_range'right = set_index_range'low = 11
set_index_range'ascending = false
set_index_range'value("20") = 20

logic_level'left = unknown
logic_level'right = high
logic_level'low = unknown
logic_level'high = high
logic_level'ascending = true
logic_level'value("low") = low

Ιδιότητες Βαθμωτών Τύπων

Σε διακριτούς και βαθμωτούς τύπους υπάρχουν οι ακόλουθες ιδιότητες:

Type'pos(x)	Θέση του x
Type'val(n)	Τιμή στην θέση n
Type'succ(x)	Τιμή στην επόμενη μεγαλύτερη θέση του x
Type'pred(x)	Τιμή στην επόμενη μικρότερη θέση του x
Type'leftof(x)	Τιμή μία θέση αριστερά του x
Type'rightof(x)	Τιμή μία θέση δεξιά του x

Διαφοροποιούνται ανάλογα με την διάταξη (ascending / descending)

logic_level'pos(unknown) = 0

logic_level'val(3) = high

logic_level'succ(unknown) = low

logic_level'pred(undriven) = low

Αντικείμενα

Αντικείμενα:

- Ένα αντικείμενο δημιουργείται από μία δήλωση και έχει τιμή και τύπο.
- Μπορεί να είναι σταθερά, μεταβλητή, σήμα ή αρχείο.
- Τα σήματα μπορούν να θεωρηθούν ως γραμμές διασύνδεσης σε ένα σχηματικό και παίρνουν τιμές με κάποια συνάρτηση ανάθεσης.
- Οι σταθερές και οι μεταβλητές μοντελοποιούν την συμπεριφορά ενός κυκλώματος και χρησιμοποιούνται σε διαδικασίες και συναρτήσεις, όπως σε μία γλώσσα προγραμματισμού.

Σταθερές

Σταθερά: οποιουδήποτε τύπου αλλά παίρνει μία μόνο τιμή. Χρησιμοποιείται για ευαναγνωσιμότητα (αντικαθιστά αριθμούς)

constant identifier **:** subtype_indication **:=** expression

Παραδείγματα

constant number_of_bytes **:** integer **:=** 4;

constant number_of_bits **:** integer **:=** 8*number_of_bytes;

constant e **:** real **:=** 2.71828;

constant prop_delay **:** time **:=** 3 ns;

constant size_limit, count_limit **:** integer **:=** 255;

Μεταβλητές

Μεταβλητή: οποιουδήποτε τύπου με οσεσδήποτε τιμές.

variable identifier **:** subtype_indication **:=** expression

- Μία μεταβλητή ορίζεται εντός μίας διαδικασίας (*process*).
- Η αρχική τιμή της είναι η αριστερότερη τιμή του τύπου.
- Η καταχώρηση μεταβλητής γίνεται γενικά ως ακολούθως:

variable_name := *expression;*

- Με την καταχώρηση τιμής σε μία μεταβλητή αυτή ενημερώνεται ακαριαία (σε αντιδιαστολή με τα σήματα που η τιμή τους ενημερώνεται σε μελλοντική χρονική στιγμή)

variable index **:** integer **:=** 0;

variable sum **:** real **:=** 0.5;

variable s **:** bit_vector (0 to 15) **:=** "FFFF";

Παράδειγμα χρήσης μεταβλητών

```
architecture sample of ent is
  constant pi : real := 3.14159;
begin
  process
    variable counter : integer;
  begin
     . . .
    counter:=counter+1;
     . . .
  end process;
end architecture sample;
```

Παράδειγμα χρήσης μεταβλητών

```
entity show_var is
port (a: in bit; b:out bit);
end show_var;
architecture example of show_var is
  signal f1,f2 : bit := '0';
  signal f3,f4 : bit_vector (1 downto 0);
begin
  p0: process (f1, f2, f3, f4)
    variable s1, s2 : bit;
    variable s3, s4 : bit_vector(1 downto 0);
    variable s5: bit_vector (3 downto 0);
  begin
     s1:=not (f1 or f2);
     s2:='1';
     s3:="01";
     s4:=f4 or "10";
     s5:=(3=>'1', 2=>'1', others=>'0');
  end process;

end architecture example ;
```

Σήματα

Απόδοση τιμής σε σήμα

Ακολουθιακή

Συντρέχουσα (concurrent)

Η τιμή ενός σήματος είναι συνάρτηση των τιμών του οδηγού σήματος

Η απόδοση τιμής σε ένα σήμα προκαλεί αλλαγή στις τιμές των οδηγών άλλων σημάτων

Όλες οι αναθέσεις σε ένα σήμα σε μία διεργασία δημιουργούν έναν οδηγό

Κάθε σήμα έχει έναν ΜΟΝΟ οδηγό → Ανάθεση τιμής σε μία μόνο διεργασία

Σήματα

Η συνάρτηση ισοδυναμίας ExNOR

```
entity equivalence is
  port (a,b: in bit; x:out bit);
end equivalence;
architecture behave of equivalence is
begin
  p0: process (a, b)
  begin
    if (a=b) then x<='1';
    elsif (a/=b) then x<='0';
    endif;
  end process;
end;
```

Σήματα

Σήματα:

Η τιμή τους ανανεώνεται όταν εκτελεστεί μία εντολή ανάθεσης σε σήμα με προβλεπόμενη καθυστέρηση, πχ. Sum <= (A xor B) after 10ns;

Όταν η καθυστέρηση λείπει υπονοείται 0ns (delta delay)

Ένα σήμα συλλέγει οδηγούς σημάτων και όχι τιμές και παίρνει την τιμή της συνάρτησης των τιμών των οδηγών.

```
signal list_of_signal_names: type [ := initial value] ;

    signal SUM, CARRY: std_logic;
    signal CLOCK: bit;
    signal TRIGGER: integer :=0;
    signal DATA_BUS: bit_vector (0 to 7);
    signal VALUE: integer range 0 to 100;
```

signal wave: std_logic;
Wave <= ‘0’, ‘1’ **after** 5ns, ‘0’ **after** 10ns, ‘1’ **after** 20ns;

Αντικείμενα

Example of a process using Variables

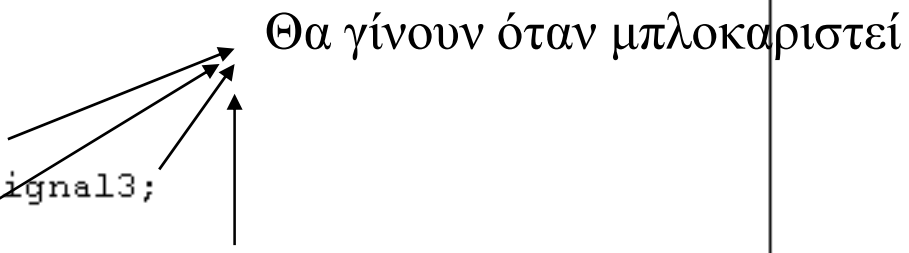
```
architecture VAR of EXAMPLE is
     signal TRIGGER, RESULT: integer := 0;
begin
     process
          variable variable1: integer :=1;
          variable variable2: integer :=2;
          variable variable3: integer :=3;
     begin
          wait on TRIGGER;
          variable1 := variable2;
          variable2 := variable1 + variable3;
          variable3 := variable2;
          RESULT <= variable1 + variable2 + variable3;
     end process;
end VAR
```

Ο υπολογισμός των μεταβλητών γίνεται σειριακά: variable1=2, variable2=2+3=5, variable3=5 και άρα Result=2+5+5=12.

Σήματα

Example of a process using Signals

```
architecture SIGN of EXAMPLE is
     signal TRIGGER, RESULT: integer := 0;
      signal signal1: integer :=1;
     signal signal2: integer :=2;
     signal signal3: integer :=3;
begin
     process
     begin
           wait on TRIGGER;
           signal1 <= signal2;
           signal2 <= signal1 + signal3;
           signal3 <= signal2;
           RESULT  <= signal1 + signal2 + signal3;
     end process;
end SIGN;
```



Τα σήματα υπολογίζονται ταυτόχρονα την χρονική στιγμή που φτάνει το σήμα Trigger και με τις παλιές τιμές των σημάτων. Έτσι signal1=2, signal2=1+3=4, signal3=2 και Result=1+2+3=6

Σήματα

Η σειρά ανάθεσης στις μεταβλητές έχει μεγάλη σημασία αφού εκτελούνται ακολουθιακά.

Η σειρά ανάθεσης στα σήματα δεν έχει καμία σημασία αφού εκτελούνται όλες μαζί μετά από χρόνο δ ή μεγαλύτερο.

Ας δούμε ένα παράδειγμα:

Σήματα

```
architecture behave of seq_sig_assgn is
signal f1:bit:='1';
signal x:bit:='0';
begin
      p0: process(x,a)
        variable f2:bit:='0';
        variable f3, f4: bit:='0';
      begin
        f3:=f2 or f1;
        f4:=f3 and f1;
        x<= (f3 and f4) after 5ns;
        a<=x after 5ns;
      end process;
end behave;
```

f3=1 άμεσα

f4=1 άμεσα

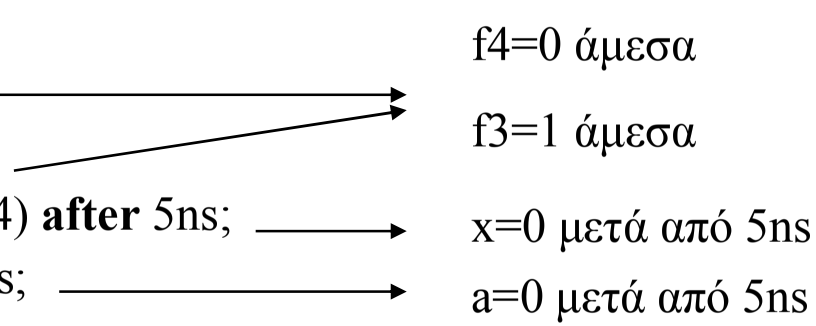
x=1 μετά από 5ns

a=0 μετά από 5ns,
1 μετά από 10ns (επανάκληση process)

Σήματα

Η εναλλαγή των αναθέσεων των μεταβλητών άλλαξε το αποτέλεσμα.

```
architecture behave of seq_sig_assgn is
signal f1:bit:='1';
signal x:bit:='0';
begin
    p0: process(x,a)
      variable f2:bit:='0';
      variable f3, f4: bit:='0';
    begin
      f4:=f3 and f1;
      f3:=f2 or f1;
      x<= (f3 and f4) after 5ns;
      a<=x after 5ns;
    end process;
end behave;
```



Σήματα

```
architecture behave of seq_sig_assgn is
signal f1:bit:='1';
signal x:bit:='0';
begin
    p0: process(x,a)
      variable f2:bit:='0';
      variable f3, f4: bit:='0';
    begin
      f3:=f2 or f1;
      f4:=f3 and f1;
      a<=x after 5ns;
      x<= (f3 and f4) after 5ns;
    end process;
end behave;
```

f3=1 άμεσα

f4=1 άμεσα

a=0 μετά από 5ns

1 μετά από 10ns

x=1 μετά από 5ns

Η αντιστροφή σειράς ανάθεσης των σημάτων δεν αλλάζει το αποτέλεσμα

Σήματα

```
entity not_xor is
    port ( a,b : in bit; x : out bit);
end not_xor;

architecture behave of not_xor is
begin
    p0: process(a,b)
    begin
      x<='0';                                          x=0 μετά από 0ns+δ
      if (a=b) then      εάν a=b τότε αντικαθίσταται η πρώτη ανάθεση με την ακόλουθη:
          x<= '1';                                     x=1 μετά από 0ns+δ
      end if;
    end process;
end behave;
```

Χαρακτηριστικά Σημάτων

Attribute	Function
signal_name'**event**	returns the Boolean value True if an event on the signal occurred, otherwise gives a False
signal_name'**active**	returns the Boolean value True there has been a transaction (assignment) on the signal, otherwise gives a False
signal_name'**transaction**	returns a signal of the type "bit" that toggles (0 to 1 or 1 to 0) every time there is a transaction on the signal.
signal_name'**last_event**	returns the time interval since the last event on the signal
signal_name'**last_active**	returns the time interval since the last transaction on the signal
signal_name'**last_value**	gives the value of the signal before the last event occurred on the signal
signal_name'**delayed(T)**	gives a signal that is the delayed version (by time T) of the original one. [T is optional, default T=0]
signal_name'**stable(T)**	returns a Boolean value, True, if no event has occurred on the signal during the interval T, otherwise returns a False. [T is optional, default T=0]
signal_name'**quiet(T)**	returns a Boolean value, True, if no transaction has occurred on the signal during the interval T, otherwise returns a False. [T is optional, default T=0]

Transaction (ανάθεση τιμής) ≠ Event (αλλαγή τιμής)

Χαρακτηριστικό s'event

Επιστρέφει boolean τιμή TRUE αν γίνει αλλαγή τιμής στο σήμα s

```
entity DFF is
  port(CLK, D : in  std_logic; Q: out std_logic);
end DFF;

architecture RTL of DFF is
begin
  seq0 : process
  begin
    wait until CLK'event and CLK = '1';
    Q <= D;
  end process;
end RTL;
```

Χαρακτηριστικό s'active

Επιστρέφει boolean τιμή TRUE αν υπάρχει σχεδιασμένο γεγονός στο σήμα s (ακόμη και αν δεν αλλάξει)

```
p0 : process (d)
variable a, b: boolean;
begin
   s<=d after 10ns;
   a:=s'active;
   b:=s'event;
end process;
```

a=TRUE αφού το s έχει προγραμματισμένη τιμή

b=FALSE αφού το s δεν αλλάζει τιμή στον τρέχων κύκλο εξομοίωσης

Χαρακτηριστικό s'last_value

Επιστρέφει τιμή τύπου ίδιου με τον βασικό τύπο του σήματος, ίση με την τελευταία τιμή που είχε το σήμα πριν το τελευταίο γεγονός.

```
p0 : process (d)
variable b: bit:='1';
begin
   s<=d;
   b:=s'last_value;
end process;
```

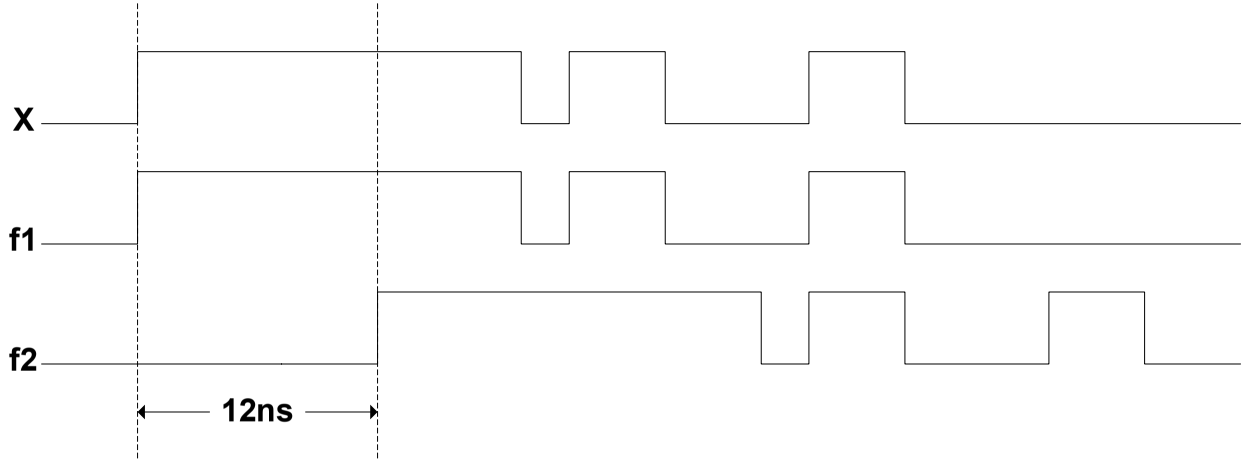
b=0 αν η αρχική τιμή του σήματος s είναι 0.

Χαρακτηριστικό s'delayed

Δημιουργεί ένα σήμα καθυστερημένο κατά t≥δ σε σχέση με το σήμα s

```
entity attr_delayed is
  port(x : in bit);
end attr_delayed;

architecture behave of attr_delayed is
signal f1, f2: bit;
begin
  p0 : process (x)
  begin
    f1<=x;
    f2<=f1'delayed (12ns);
  end process;
end behave;
```



Ψευδώνυμα

- Δεν είναι νέο αντικείμενο αλλά αποτελεί διαφορετική ονομασία για υπάρχον αντικείμενο.
- Μια χρήση είναι ο προσδιορισμός μίας περιοχής τιμών ενός πίνακα.

signal instruction: std_logic_vector (15 **downto** 0);

alias opcode: std_logic_vector (3 **downto** 0) **is** instruction (15 **downto** 12);

Το opcode αποτελεί τα δυαδικά ψηφία 15 έως 12 της εντολής instruction.

Τελεστές

Class						
1. Logical operators	**and**	**or**	**nand**	**nor**	**xor**	**xnor**
2. Relational operators	=	/=	<	<=	>	>=
3. Shift operators	**sll**	**srl**	**sla**	**sra**	**rol**	**ror**
4.Addition operators	+	=	**&**			
5. Unary operators	+	-				
6. Multiplying op.	*	/	**mod**	**rem**		
7. Miscellaneous op.	**	**abs**	**not**			

- Η προτεραιότητά τους αυξάνει από την κατηγορία 1 προς την κατηγορία 7.
- Οι τελεστές ίδιας προτεραιότητας εφαρμόζονται από αριστερά προς τα δεξιά σε μία έκφραση. Η έκφραση «not X & Y xor Z rol 1» ισοδυναμεί με την έκφραση «((not X) & Y) xor (Z rol 1)»
- Για έναν τύπο απαρίθμησης, το αριστερότερο στοιχείο της λίστας θεωρείται το μικρότερο και το δεξιότερο το μεγαλύτερο

Σχεσιακοί Τελεστές

- Οι τελεστές κάθε σχεσιακής έκφρασης πρέπει να είναι ίδιου τύπου.
- Το αποτέλεσμα είναι τύπου BOOLEAN

Σε σύγκριση πινάκων ισχύουν τα ακόλουθα:

- Δύο πίνακες είναι ίσοι εάν είναι της ίδιας διάστασης και έχουν ένα προς ένα όλα τα στοιχεία τους ίσα
- Ο πίνακας A είναι μεγαλύτερος από τον B (A>B) εάν:

  1. Το πρώτο στοιχείο του A είναι μεγαλύτερο από το πρώτο στοιχείο του B.

  2. Τα πρώτα στοιχεία των A, B είναι ίδια και ο B δεν έχει άλλα στοιχεία αλλά ο A έχει.

Σχεσιακοί Τελεστές

variable x1: bit_vector (3 **downto** 0) := "1011";

variable x2: bit_vector (3 **downto** 0) := "1011";

variable x3: bit_vector (3 **downto** 0) := "0011";

variable y: bit_vector (4 **downto** 0) := "10110";

variable w: bit_vector (4 **downto** 0) := "10010";

variable z: bit_vector (4 **downto** 0) := "000000";

x1=x2, x2>x3, x2<y, x2>w, x2>z, y>z

Τελεστές Ολίσθησης

Operator	Description	Operand Type	Result Type
sll	Shift left logical (fill right vacated bits with the 0)	Left: Any one-dimensional array type with elements of type bit or Boolean; Right: integer	Same as left type
srl	Shift right logical (fill left vacated bits with 0)	same as above	Same as left type
sla	Shift left arithmetic (fill right vacated bits with rightmost bit)	same as above	Same as left type
sra	Shift right arithmetic (fill left vacated bits with leftmost bit)	same as above	Same as left type
rol	Rotate left (circular)	same as above	Same as left type
ror	Rotate right (circular)	same as above	Same as left type

Αριθμητικοί Τελεστές

Operator	Description	Left Operand Type	Right Operand Type	Result Type
*	Multiplication	Any integer or floating point	Same type	Same type
		Any physical type	Integer or real type	Same as left
		Any integer or real type	Any physical type	Same as right
/	Division	Any integer or floating point	Any integer or floating point	Same type
		Any physical type	Any integer or real t ype	Same as left
		Any physical type	Same type	Integer
mod	Modulus	Any integer type		Same type
rem	Remainder	Any integer type		Same type

Operator	Description	Left Operand Type	Right Operand Type	Result Type
**	Exponentiation	Integer type	Integer type	Same as left
		Floating point	Integer type	Same as left
abs	Absolute value	Any numeric type		Same type
not	Logical negation	Any bit or Boolean type		Same type