Γλώσσες Περιγραφής Υλικού

Προχωρημένα στοιχεία της VHDL

Νικόλαος Καββαδίας
nkavv@physics.auth.gr

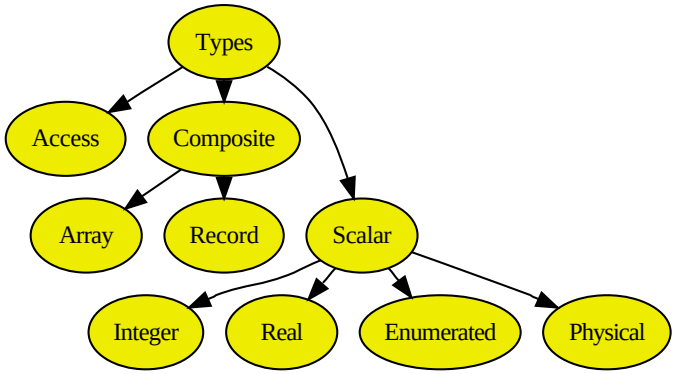
31 Μαρτίου 2009

Σκιαγράφηση της διάλεξης

- Προχωρημένα στοιχεία της VHDL
  - Τύποι και υποτύποι προκαθορισμένοι ή καθοριζόμενοι από το χρήστη (predefined or user-defined types and subtypes)
  - Μετατροπές τύπων (type conversions)
  - Ιδιότητες σημάτων και ιδιότητες καθοριζόμενες από το χρήστη (signal and user-defined attributes)
  - Ψευδώνυμα (aliases)
  - Η δήλωση GENERIC
  - Συστατικά (components)
  - Αντιστοιχίσεις θυρών και γενικών σταθερών σε στιγμιότυπα συστατικών (generic and port mappings in instances)
  - Παραδείγματα σχεδιασμού κυκλωμάτων: συγκριτής μεγαλύτερο-ή-ίσο από, δυαδικός απαριθμητής, παραμετρικός καταχωρητής, ALU με συστατικά

Ιεραρχία των τύπων δεδομένων στη VHDL

- Στη VHDL οι δηλώσεις θυρών, σημάτων και μεταβλητών πρέπει να προσδιορίζουν τον αντίστοιχό τους τύπο (type) ή υποτύπο (subtype)



Βαθμωτοί τύποι (1)

- Παραδείγματα χρήσης INTEGER (εύρος τιμών: implementation dependent με ελάχιστο το $2^{-31}-1$ ώς $2^{31}-1$)

```
variable a : INTEGER;
...
a := 1; -- OK
a := -1; -- OK
a := 1.0; -- illegal
```

- Παραδείγματα χρήσης REAL

```
variable a : REAL;
...
a := 1.3; -- OK
a := 1.7E-13; -- OK
a := 1; -- illegal
a := 5.3 ohm; -- illegal
```

- Παράδειγμα ορισμού νέου φυσικού τύπου (προκαθορισμένος μόνο ο τύπος TIME)

```
TYPE resistance is RANGE 0 to 10000000;
...
UNITS
  ohm; -- ohm
  Kohm = 1000 ohm;
  Mohm = 1000 kohm;
END UNITS;
```

Βαθμωτοί τύποι (2)

- Απαριθμητοί τύποι (enumerated types)
  - Ο χρήστης ορίζει τη λίστα των επιτρεπόμενων τιμών που μπορούν να ανατεθούν σε ένα αντικείμενο που δηλώνεται με τον συγκεκριμένο τύπο
  - Απαριθμητός τύπος δεδομένων χαρακτηριστικός για μηχανές πεπερασμένων καταστάσεων (FSM)

```
TYPE fsm_state is (idle, forward, backward, stop);
...
signal current_state := IDLE;
```

  - Απαριθμητός τύπος δεδομένων που καταγράφει τα επιτρεπόμενα χρώματα στο πρότυπο TELETEXT

```
TYPE rgb3 is (black, blue, green, cyan, red, magenta, yellow, white);
```

  - Τύπος δεδομένων για την υλοποίηση λογικής 4 επιπέδων κατά Verilog

```
TYPE verilog_mvl4 is ('0', '1', 'X', 'Z');
```

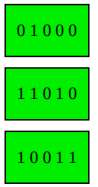

Σύνθετοι τύποι: Πίνακες (1)

- Πίνακας: συλλογή από αντικείμενα του ίδιου τύπου δεδομένων
- Τα στοιχεία ενός πίνακα τοποθετούνται σε διαδοχικές θέσεις
- Ένας πίνακας στη VHDL μπορεί να είναι μονοδιάστατος (1D), οπότε αναφέρεται και ως 'διάνυσμα' (vector), διδιάστατος (2D) ή εν γένει πολυδιάστατος (multi-dimensional)
- Επιτρέπεται ο ορισμός πινάκων των οποίων τα στοιχεία αποτελούν με τη σειρά τους πίνακες, όπως πίνακες 1D × 1D
- Αναπαραστάσεις πινάκων

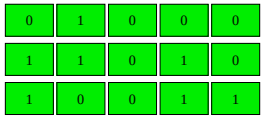
- Βαθμωτό στοιχείο

- 1D

- 1D × 1D



- 2D



Σύνθετοι τύποι: Πίνακες (2)

- Δήλωση για τον ορισμό ενός νέου τύπου πίνακα:
  TYPE <array name> **IS ARRAY** (specification) **OF** <data type>;
- Δήλωση ενός SIGNAL με τον νέο τύπο πίνακα
  SIGNAL <signal name>: <array type> [:= <initial value];
- Παραδείγματα

```
TYPE byte is ARRAY (7 downto 0) of STD_LOGIC; -- 1D
TYPE image is ARRAY (0 to 31) of byte; -- 1Dx1D
TYPE matrix2D is ARRAY (0 to 3, 1 downto 0) OF STD_LOGIC;
...
SIGNAL x: image; -- an 1Dx1D signal
SIGNAL y: matrix2D;
-- Initialization
... := "0001"; -- 1D
... := ('0','0','0','1'); -- 1D
... := (('0','1','1','1'),('1','1','1','0')); -- 1Dx1D or 2D
TYPE binary IS (ON, OFF);
-- Assignments
x(0) <= y(1)(2); -- 1Dx1D
x(0) <= v(1,2); -- 2D
x <= y(0); -- entire row
x(3 downto 1) <= y(1)(4 downto 2);
x(3 downto 0) <= (others => '1');
x(3 downto 0) <= (3 => '1', 2 => '0', others => '0');
```


Σύνθετοι τύποι: Εγγραφές

- Εγγραφή (record): συλλογή από αντικείμενα που μπορεί να ανήκουν σε διαφορετικούς τύπους
- Τα στοιχεία μιας record δεικτοδοτούνται από τα ιεραρχικά ονόματα των αντίστοιχων μελών
- Παράδειγμα ορισμού και χρήσης μιας rerord

```
TYPE binary IS (ON, OFF);
TYPE switch_info IS
  RECORD
    status : BINARY;
    IDnumber : INTEGER;
END RECORD;
...
VARIABLE switch : switch_info;
...
switch.status := ON;  -- status of the switch
```

Υποτύποι δεδομένων (subtypes)

- Ο υποτύπος ή δευτερεύων τύπος δεδομένων (SUBTYPE) είναι ένας τύπος δεδομένων με περιορισμούς
- Επιτρέπονται οι πράξεις μεταξύ αντικειμένων ενός υποτύπου και του αντίστοιχου βασικού του τύπου
- Ο χειρισμός των υποτύπων (αρχικοποιήσεις, αναθέσεις) είναι αντίστοιχος με αυτόν των βασικών τους τύπων
- Δήλωση για τον ορισμό ενός νέου υποτύπου:

  SUBTYPE <subtype name> **IS** (specification);

- Παραδείγματα

```
SUBTYPE natural is INTEGER RANGE 0 to +2147483647;
SUBTYPE mylogic is STD_LOGIC RANGE '0' to 'Z';
SUBTYPE caps is CHARACTER RANGE 'A' to 'Z';
```


Διανύσματα από STD_LOGIC: STD_LOGIC_VECTOR, UNSIGNED, SIGNED

- Χαρακτηριστικά των τύπων

Τύπος	Βιβλιοθήκη.Πακέτο	Λογικές πράξεις	Αριθμ. πράξεις
std_logic_vector	ieee.std_logic_1164	✓	
unsigned	ieee.numeric_std	✓	✓
signed	ieee.numeric_std	✓	✓

- Ορισμοί των τύπων στα αντίστοιχα πακέτα

```
type std_logic_vector is array (natural range <>) of std_logic;
type unsigned is array (natural range <>) of std_logic;
type signed is array (natural range <>) of std_logic;
```

- Παράδειγμα:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity vectors is
  port (
    vect : in std_logic_vector(1 downto 0);
    unsi : in unsigned(7 downto 0);
    sign : out signed(15 downto 0)
  );
end vectors;
```


Παράδειγμα: Συγκριτής μεγαλύτερο-ίσο (greater-than or equal) για διανύσματα τύπου UNSIGNED

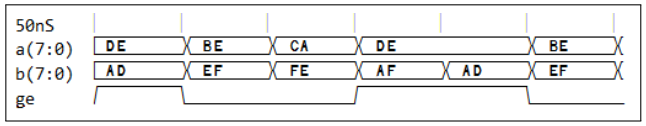
- Κώδικας σε VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity compge is
  port (
    A : in unsigned(7 downto 0);
    B : in unsigned(7 downto 0);
    GE : out std_logic
 );
end compge;

architecture rtl of compge is
begin
  GE <= '1' when A >= B else '0';
end rtl;
```

- Διάγραμμα χρονισμού:



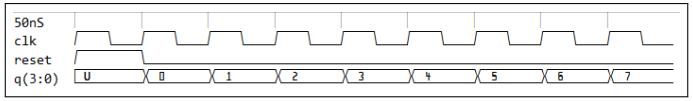
Παράδειγμα: Δυαδικός απαριθμητής των 4-bit

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter4b is
  port(
    clk : in std_logic;
    reset : in std_logic;
    q : out unsigned(3 downto 0)
  );
end counter4b;
```

```
architecture impl of counter4b is
  signal count : unsigned(3 downto 0);
begin
process (clk)
begin
  if rising_edge(clk) then
    if (reset = '1') then
      count <= (others => '0');
    else
      count <= count + 1;
    end if;
  end if;
end process;
q <= count;
end impl;
```

Διάγραμμα χρονισμού:



Αναπαράσταση ακεραίων με διανύσματα

Δυαδικό και δεκαεξαδικό

Ακέραιος	Δυαδικό	hex
0	"00000"	X"0"
1	"00001"	X"1"
2	"00010"	X"2"
3	"00011"	X"3"
4	"00100"	X"4"
5	"00101"	X"5"
6	"00110"	X"6"
7	"00111"	X"7"
8	"01000"	X"8"
9	"01001"	X"9"
10	"01010"	X"A"
11	"01011"	X"B"
12	"01100"	X"C"
13	"01101"	X"D"
14	"01110"	X"E"
15	"01111"	X"F"
16	"10000"	X"10"
17	"10001"	X"11"

Συμπλήρωμα ως προς 2

Ακέραιος	Δυαδικό	hex
-8	"1000"	X"8"
-7	"1001"	X"9"
-6	"1010"	X"A"
-5	"1011"	X"B"
-4	"1100"	X"C"
-3	"1101"	X"D"
-2	"1110"	X"E"
-1	"1111"	X"F"
0	"0000"	X"0"
1	"0001"	X"1"
2	"0010"	X"2"
3	"0011"	X"3"
4	"0100"	X"4"
5	"0101"	X"5"
6	"0110"	X"6"
7	"0111"	X"7"

Υπερφόρτωση τελεστή (operator overloading)

- Στη VHDL μπορούν να δηλωθούν (π.χ. σε πακέτα) εκ νέου τελεστές από τον χρήστη οι οποίοι να έχουν τα ίδια ονόματα με προκαθορισμένους τελεστές, αλλά να δρουν σε διαφορετικούς τύπους δεδομένων
- Η τεχνική αυτή ονομάζεται *υπερφόρτωση τελεστή*
- Υπερφόρτωση του τελεστή '+' για την πρόσθεση ενός INTEGER με ένα BIT

```
FUNCTION "+" (a:INTEGER, b:BIT) RETURN INTEGER IS
BEGIN
  IF (b='1') THEN
    RETURN a+1;
  ELSE
    RETURN a;
  END IF;
END "+";
...
SIGNAL inp1,outp: INTEGER RANGE 0 TO 15;
SIGNAL inp2: BIT;
...
outp <= inp1 + inp2;
```


Μετατροπές τύπων (type conversions)

- Αρκετά συχνά, η επαρκής χρήση της VHDL εξαρτάται από τη σωστή χρήση των κατάλληλων υπερφορτωμένων τελεστών και μετατροπών
- Η μετατροπή τύπου δεδομένων είναι απαραίτητη για την εργασία με δεδομένα διαφορετικών τύπων, καθώς η VHDL δεν επιτρέπει την απευθείας εκτέλεση πράξεων με δεδομένα διαφορετικών τύπων
- Συχνά χρησιμοποιούμενες μετατροπές
  - SIGNED, UNSIGNED $\Longleftrightarrow$ STD_LOGIC
  - SIGNED, UNSIGNED $\Longleftrightarrow$ STD_LOGIC_VECTOR
  - SIGNED, UNSIGNED $\Longleftrightarrow$ INTEGER
  - STD_LOGIC_VECTOR $\Longleftrightarrow$ INTEGER
- Ενσωματωμένοι μηχανισμοί μετατροπής τύπων
  - Αυτόματη μετατροπή
  - Μετατροπή με απόδοση (casting) τύπου
- Οι συναρτήσεις μετατροπών τύπου είναι δηλωμένες στα πακέτα numeric_std και std_logic_arith
- ☣ Χρησιμοποιούμε πάντα ΕΝΑ ΑΠΟ ΤΑ ΔΥΟ αυτά πακέτα στην ίδια περιγραφή

Αυτόματη μετατροπή τύπων

- Δύο τύποι μπορούν να μετατραπούν αυτόματα μεταξύ τους όταν αποτελούν υποτύπους του ίδιου βασικού τύπου
- Η μετατροπή μεταξύ STD_LOGIC και STD_ULOGIC είναι αυτόματη καθώς:

```
SUBTYPE std_logic is resolved std_ulogic;
```

- Τα στοιχεία των τύπων SIGNED, UNSIGNED, STD_LOGIC_VECTOR είναι τύπου STD_LOGIC
- Έστω στα παρακάτω παραδείγματα σήματα των τύπων STD_LOGIC, STD_ULOGIC, SIGNED, UNSIGNED, STD_LOGIC_VECTOR τα οποία σημειώνουμε με τις αντίστοιχες καταλήξεις sl, sul, sv, uv, slv

```
A_sl <= J_uv(0);
B_sul <= K_sv(7);
L_uv(0) <= C_sl;
M_slv(2) <= N_sv(2);
```

Μετατροπές UNSIGNED, SIGNED ⇔ STD_LOGIC_VECTOR

- Η μετατροπή τύπου με απόδοση τύπου (type casting) επιτρέπεται για πίνακες ίσων διαστάσεων όταν:
  - Τα στοιχεία τους είναι του ίδιου τύπου (π.χ. std_logic)
  - Οι δείκτες για τον προσδιορισμό θέσεων στους πίνακες έχουν κοινό τύπο (π.χ. integer)
- Οι μετατροπές γίνονται με τις προκαθορισμένες συναρτήσεις std_logic_vector(), unsigned(), signed()

```
A_slv <= std_logic_vector(B_uv);
C_slv <= std_logic_vector(D_sv);
G_uv <= unsigned(H_slv);
J_sv <= signed(K_slv);
```


Μετατροπές UNSIGNED, SIGNED ⇔ INTEGER (πακέτο numeric_std)

- Η μετατροπή από και προς INTEGER απαιτεί τη χρήση μιας συνάρτησης μετατροπής
- Μετατροπή UNSIGNED, SIGNED ⇒ INTEGER

```
Unsigned_int <= TO_INTEGER(A_uv);
Signed_int <= TO_INTEGER (B_sv);
```

- Μετατροπή INTEGER ⇒ UNSIGNED, SIGNED
  - Το εύρος bit του διανύσματος δηλώνεται ρητά

```
C_uv <= TO_UNSIGNED(Unsigned_int,8);
D_sv <= TO_SIGNED (Signed_int,8);
```

- Παράδειγμα χρήσης: Διευθυνσιοδότηση σε μνήμη ROM

```
Data_slv <= ROM( TO_INTEGER(Addr_uv) );
```

Σήματα για τα παραδείγματα:

```
signal A_uv, C_uv : unsigned (7 downto 0);
signal Unsigned_int : integer range 0 to 255;
signal B_sv, D_sv : signed(7 downto 0);
signal Signed_int : integer range -128 to 127;
```


Μετατροπές STD_LOGIC_VECTOR ⇔ INTEGER (πακέτο numeric_std)

- Η μετατροπή μεταξύ των τύπων STD_LOGIC_VECTOR και INTEGER αποτελεί διαδικασία δύο βημάτων
- Μετατροπή STD_LOGIC_VECTOR ⇒ INTEGER

```
Unsigned_int <= TO_INTEGER( UNSIGNED(A_slv) );
Signed_int   <= TO_INTEGER( SIGNED(B_slv) );
```

- Μετατροπή INTEGER ⇒ STD_LOGIC_VECTOR

```
C_slv <= STD_LOGIC_VECTOR(TO_UNSIGNED(Unsigned_int,8));
D_slv <= STD_LOGIC_VECTOR(TO_SIGNED(Signed_int,8));
```

- Παράδειγμα χρήσης: Διευθυνσιοδότηση σε μνήμη ROM

```
Data_slv <= ROM( TO_INTEGER(UNSIGNED(Addr_slv)) );
```

Σήματα για τα παραδείγματα:

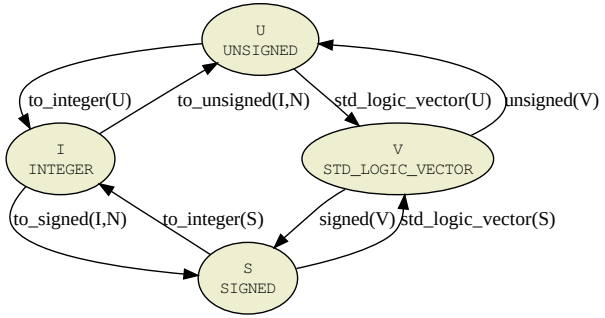
```
signal A_slv, C_slv : std_logic_vector (7 downto 0);
signal Unsigned_int : integer range 0 to 255;
signal B_slv, D_slv : std_logic_vector(7 downto 0);
signal Signed_int : integer range -128 to 127;
```


Σύνοψη των συναρτήσεων μετατροπής τύπου του πακέτου numeric_std

- Δήλωση για τη χρήση του πακέτου

```
LIBRARY ieee;
USE ieee.numeric_std.all;
```

- Γραφική αναπαράσταση των επιτρεπόμενων μετατροπών



Συναρτήσεις μετατροπής του πακέτου std_logic_arith

- Στο std_logic_arith μπορούμε να βρούμε τις συναρτήσεις μετατροπής
  conv_integer, conv_unsigned, conv_signed, conv_std_logic_vector

Κλήση συνάρτησης	Περιγραφή
conv_integer(param)	Μετατρέπει μια παράμετρο *param* τύπου INTEGER, UNSIGNED, SIGNED ή STD_ULOGIC σε μια τιμή τύπου INTEGER
conv_unsigned(param,b)	Μετατρέπει μια παράμετρο *param* τύπου INTEGER, UNSIGNED, SIGNED ή STD_ULOGIC σε μια τιμή τύπου UNSIGNED με μέγεθος *b* bit
conv_signed(param,b)	Μετατρέπει μια παράμετρο *param* τύπου INTEGER, UNSIGNED, SIGNED ή STD_ULOGIC σε μια τιμή τύπου SIGNED με μέγεθος *b* bit
conv_std_logic_-vector(param,b)	Μετατρέπει μια παράμετρο *param* τύπου INTEGER, UNSIGNED, SIGNED ή STD_ULOGIC σε μια τιμή τύπου STD_LOGIC_VECTOR με μέγεθος *b* bit

Το πακέτο std_logic_unsigned ορίζει υπερφορτωμένες εκδοχές των πρώτων τριών συναρτήσεων μετατροπής για δεδομένα τύπου STD_LOGIC_VECTOR

Παραδείγματα μετατροπών με βάση το πακέτο std_logic_arith)

- Μετατροπή UNSIGNED, SIGNED ⇒ INTEGER

```
Unsigned_int <= CONV_INTEGER( A_uv );
Signed_int   <= CONV_INTEGER( B_sv );
```

- Μετατροπή INTEGER ⇒ UNSIGNED, SIGNED

```
A_uv <= CONV_UNSIGNED(Unsigned_int,8);
B_sv <= CONV_SIGNED (Signed_int,8);
```

- Μετατροπή UNSIGNED,SIGNED ⇒ STD_LOGIC_VECTOR

```
C_slv <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER(A_uv),8);
D_slv <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER(B_sv),8);
```

- Παράδειγμα χρήσης: Διευθυνσιοδότηση σε μνήμη ROM

```
Data_slv <= ROM( CONV_INTEGER(Addr_uv) ); -- or
Data_slv <= ROM( CONV_INTEGER(Addr_slv) );
```

Σήματα για τα παραδείγματα:

```
signal A_uv : unsigned (7 downto 0) ;
signal Unsigned_int : integer range 0 to 255 ;
signal C_slv, D_slv : std_logic_vector( 7 downto 0) ;
signal B_sv : signed (7 downto 0) ;
signal Signed_int : integer range -128 to 127;
```


Ψευδώνυμα (aliases)

- Ένα ψευδώνυμο επιτρέπει την αναφορά σε αντικείμενα με εναλλακτικό τρόπο
- Τα ψευδώνυμα μπορούν να βελτιώσουν σημαντικά την ευκολία ανάγνωσης περιγραφών (readability) σε VHDL
- Τα ψευδώνυμα μπορούν να μετονομάσουν οποιοδήποτε όνομα στη VHDL εκτός από ετικέττες και παραμέτρους των εντολών LOOP και GENERATE
- Δήλωση για τον ορισμό ενός ψευδώνυμου:

```
ALIAS <object name>: <aliased data type> IS <actual data type>;
```

- Παράδειγμα

```
ALIAS data_bus: high_byte(7 downto 0) is
                short_word(15 downto 8);
```


Προκαθορισμένες ιδιότητες (predefined attributes)

- Οι προκαθορισμένες ιδιότητες προσφέρουν τη δυνατότητα περιγραφής γενικών τμημάτων κώδικα για οποιοδήποτε μέγεθος διανύσματος ή πίνακα
- Κατηγορίες ιδιοτήτων
  - Ιδιότητες δεδομένων: Επιστρέφουν πληροφορία σχετικά με ένα διάνυσμα δεδομένων
  - Ιδιότητες σημάτων: Επιστρέφουν πληροφορία σχετικά με την τρέχουσα κατάσταση ή το ιστορικό μεταβολής ενός σήματος
- Σύνταξη προκαθορισμένων ιδιοτήτων:
  <data or signal name>'<attribute name>
- ℹ Από τις προκαθορισμένες ιδιότητες, ορισμένες μόνο είναι συνθέσιμες
- ℹ Μη συνθέσιμες ιδιότητες σημάτων: ACTIVE, QUIET, LAST_EVENT, LAST_ACTIVE, LAST_VALUE

Αναλυτικός πίνακας προκαθορισμένων ιδιοτήτων

- Έστω d το όνομα ενός διανύσματος ή πίνακα και s το όνομα ενός σήματος

Ιδιότητα	Επιστρεφόμενη τιμή
Σε συνηθισμένα δεδομένα (όχι απαριθμητά)	
d'LOW	Ο χαμηλότερος δείκτης του πίνακα
d'HIGH	Ο υψηλότερος δείκτης του πίνακα
d'LEFT	Ο αριστερός δείκτης του πίνακα
d'RIGHT	Ο δεξιός δείκτης του πίνακα
d'LENGTH	Το μέγεθος του διανύσματος
d'RANGE	Το εύρος του διανύσματος
d'REVERSE_RANGE	Το αντίστροφο εύρος του διανύσματος
Σε απαριθμητά δεδομένα	
d'VAL(pos)	Η τιμή στην καθοριζόμενη θέση
d'POS(value)	Η θέση (διεύθυνση) της καθοριζόμενης τιμής
d'LEFTOF(value)	Η τιμή στη θέση στα αριστερά της καθοριζόμενης τιμής
d'VAL(row,column)	Η τιμή στην καθοριζόμενη θέση σε ένα διδιάστατο πίνακα
Σε αντικείμενα τύπου SIGNAL	
s'EVENT	Αληθές όταν προκύπτει συμβάν στο s
s'STABLE	Αληθές όταν δεν προκύπτει συμβάν στο s
s'ACTIVE	Αληθές αν το s είναι σε υψηλή στάθμη

Παραδείγματα χρήσης προκαθορισμένων ιδιοτήτων

- Έστω το ακόλουθο σήμα:
  `SIGNAL d: STD_LOGIC_VECTOR(7 downto 0);`
- Τότε:

```
d'LOW=0, d'HIGH=7, d'LEFT=7, d'RIGHT=0, d'LENGTH=8,
d'RANGE=(7 downto 0), d'REVERSE_RANGE=(0 to 7)
```

- Για το ίδιο σήμα, οι ακόλουθες εντολές στις οποίες γίνεται επίκληση προκαθορισμένων ιδιοτήτων είναι ισοδύναμες και συνθέσιμες:

```
FOR i IN RANGE (0 to 7) LOOP ...
FOR i IN d'RANGE LOOP ...
FOR i IN RANGE (d'LOW to d'HIGH) LOOP ...
FOR i IN RANGE (0 to d'LENGTH-1) LOOP ...
```


Ιδιότητες οριζόμενες από το χρήστη (user-defined attributes)

- Για τη χρησιμοποίηση μιας ιδιότητας η οποία καθορίζεται από το χρήστη χρειάζεται η ΔΗΛΩΣΗ της και ο ΠΡΟΣΔΙΟΡΙΣΜΟΣ της
- Δήλωση ιδιότητας:
  ATTRIBUTE <attribute name>: <attribute type>;
- Προσδιορισμός ιδιότητας:
  ATTRIBUTE <attribute name> **OF** <target name>: <category> **IS** <value>;
- Για παράδειγμα:

```
ATTRIBUTE number_of_inputs: INTEGER;
ATTRIBUTE number_of_inputs of nand3: SIGNAL is 3;
...
-- qualified expression for accessing the attribute
inputs <= nand3'number_of_pins;
```

- Πρακτικό παράδειγμα: η ιδιότητα enum_encoding για την κωδικοποίηση των καταστάσεων ενός FSM

```
TYPE color is (red, green, blue, black, white);
ATTRIBUTE enum_encoding OF color: TYPE IS "00001 00010 00100 01000 10000";
```

- Μια ιδιότητα οριζόμενη από το χρήστη δεν μπορεί να δηλωθεί σε σώμα πακέτου (package body)

Δήλωση γενικής σταθεράς (GENERIC)

- Οι γενικές σταθερές προσφέρονται για την παραμετρική περιγραφή κυκλωμάτων
- Ορίζουν μία στατική παράμετρο με εμβέλεια την entity και τις architecture του κυκλώματος
- Επαυξάνουν τη δυνατότητα επαναχρησιμοποίησης κώδικα σε διαφορετικά κυκλώματα
- Σύνταξη μιας GENERIC
  GENERIC (
  <parameter name> : <parameter type> [:= <initialization];
  ...
  );
- Παράδειγμα:

```
entity test is
  generic (n : integer := 8);
  port (...);
end test;

architecture archtest of test is
...
end archtest;
```

Παράδειγμα χρήσης GENERIC: Παραμετρικός καταχωρητής

```
library IEEE;
use IEEE.std_logic_1164.all;

entity reg is
  generic (
    WIDTH : integer := 32);
  port (
    clk, rst : in  std_logic;
    d_i   : in  std_logic_vector(WIDTH-1 downto 0);
    d_o   : out std_logic_vector(WIDTH-1 downto 0));
end reg;

architecture rtl of reg is
  signal d_r : std_logic_vector(WIDTH-1 downto 0);
begin
  process (clk)
  begin
    if (clk'EVENT and clk='1') then
      if (rst = '1') then
        d_r <= (others => '0');
      else
        d_r <= d_i;
      end if;
    end if;
  end process;
  d_o <= d_r;
end rtl;
```


Συστατικό στοιχείο (COMPONENT)

- Το συστατικό στοιχείο (COMPONENT) αποτελεί μια ειδικής μορφής δήλωση μιας αυτόνομης κυκλωματικής περιγραφής η οποία επιτρέπει τη χρησιμοποίησή της από τρίτα κυκλώματα με τη μορφή υπομονάδας
- Δηλώσεις συστατικών χρησιμοποιούνται στην ιεραρχική σχεδίαση
- Σύνταξη της δήλωσης COMPONENT

```
COMPONENT <component name> IS
  [GENERIC (
    <generic name> : <type> [:= <initialization>];
    ...
  )]
  PORT (
    <port name> : [direction] <signal type>;
    ...
  );
END COMPONENT;
```


Στιγμιότυπο ενός συστατικού στοιχείου

- Το στιγμιότυπο (instance) ενός συστατικού στοιχείου (COMPONENT) αποτελεί ένα αντίτυπό του που χρησιμοποιείται στα πλαίσια της δομικής περιγραφής ενός κυκλώματος
- Σε μία κυκλωματική περιγραφή μπορούν να χρησιμοποιηθούν περισσότερα από ένα στιγμιότυπα του ίδιου COMPONENT με διαφορετικές, ενδεχόμενα, αντιστοιχίσεις σημάτων στις εισόδους και εξόδους του
- Δημιουργία στιγμιότυπου (χρήση) από COMPONENT

```
<label>: <component name>
  [GENERIC MAP (
    [<generic name> =>] <expression>,
    ...
  );
  PORT MAP (
    [<port name> =>] <expression>,
    ...
  );
```


Αντιστοίχιση θυρών σε στιγμιότυπα συστατικών: Κατά δεδηλωμένη θέση

- Στην περίπτωση αυτή, υπονοείται ότι ένα σήμα στο χάρτη θυρών (port map) αντιστοιχίζεται με τη θύρα που βρίσκεται στην ίδια θέση, στη δήλωση του συστατικού

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder2b is
  port (
    A, B : in unsigned(1 downto 0);
    C : out unsigned(2 downto 0)
  );
end adder2b;

architecture impl of adder2b is
component full_adder
  port (
    a, b, c : in std_logic;
    sum, carry : out std_logic
  );
end component;
```

```
signal carry : std_logic;
begin
bit0 : full_adder port map (
  A(0), B(0), '0',
  C(0), carry
  );
bit1 : full_adder port map (
  A(1), B(1), carry,
  C(1), C(2)
  );
end impl;
```

Αντιστοίχιση θυρών σε στιγμιότυπα συστατικών: Κατά όνομα

- Στην περίπτωση αυτή, στο χάρτη θυρών καταγράφεται ρητά η αντιστοίχιση (port map) ενός σήματος με το πρότυπο όνομα μιας θύρας

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder2b is
  port (
    A, B : in unsigned(1 downto 0);
    C : out unsigned(2 downto 0)
  );
end adder2b;

architecture impl2 of adder2b is
component full_adder
  port (
    a, b, c : in std_logic;
    sum, carry : out std_logic
  );
end component;
```

```
signal carry : std_logic;
begin
bit0 : full_adder port map (
  a => A(0), b => B(0),
  c => '0',
  sum => C(0), carry => carry
  );
bit1 : full_adder port map (
  a => A(1), b => B(1),
  c => carry,
  sum => C(1), carry => C(2)
  );
end impl2;
```


Απ΄ ευθείας χρήση στιγμιοτύπου χωρίς τη δήλωση συστατικού (direct instantiation)

- Αντί του ονόματος του συστατικού, γίνεται χρήση του ιεραρχικού ονόματος της αντίστοιχης ENTITY
- Στο παράδειγμα υποθέτουμε ότι το συστατικό `full_adder` βρίσκεται στη βιβλιοθήκη `work`

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder2b is
  port (
    A, B : in unsigned(1 downto 0);
    C : out unsigned(2 downto 0)
  );
end adder2b;

architecture impl3 of adder2b is
signal carry : std_logic;
begin
bit0 : entity work.full_adder port map (
  A(0), B(0), '0', C(0), carry
  );
bit1 : entity work.full_adder port map (
  A(1), B(1), carry, C(1), C(2)
  );
end impl3;
```


ALU αποτελούμενη από συστατικά (1)

- Περιγραφή της αριθμητικής-λογικής μονάδας (ALU) της 2ης διάλεξης με χρήση συστατικών
- Δημιουργία ξεχωριστών περιγραφών για τις επιμέρους υπομονάδες: λογική μονάδα (logic.vhd), αριθμητική μονάδα (arith.vhd) και πολυπλέκτης διανυσμάτων (vmux.vhd)
- Διασύνδεση των επιμέρους υπομονάδων για την περιγραφή του συνολικού κυκλώματος στο αρχείο alu.vhd

ALU αποτελούμενη από συστατικά (2)

- Περιγραφή του πολυπλέκτη διανυσμάτων

```
-- vmux.vhd
library ieee;
use ieee.std_logic_1164.all;

entity vmux is
  port (
    a, b: in std_logic_vector(7 downto 0);
    sel : in std_logic;
    x   : out std_logic_vector(7 downto 0)
  );
end vmux;

architecture rtl of vmux is
begin
  with sel select
    x <= a when '0',
         b when others;
end rtl;
```

ALU αποτελούμενη από συστατικά (3)

- Περιγραφή της αριθμητικής μονάδας

```
-- arith.vhd
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity arith is
  port (
    a,b : in std_logic_vector(7 downto 0);
    cin : in std_logic;
    sel : in std_logic_vector(2 downto 0);
    x : out std_logic_vector(7 downto 0)
  );
end arith;

architecture dataflow of arith is
begin
  with sel select
    x <= a when "000",
         a+1 when "001",
         a-1 when "010",
         b when "011",
         b+1 when "100",
         b-1 when "101",
         a+b when "110",
         a+b+cin when others;
end dataflow;
```


ALU αποτελούμενη από συστατικά (4)

- Περιγραφή της λογικής μονάδας

```
-- logic.vhd
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity logic is
  port (
    a,b : in std_logic_vector(7 downto 0);
    sel : in std_logic_vector(2 downto 0);
    x : out std_logic_vector(7 downto 0)
  );
end logic;

architecture dataflow of logic is
begin
  with sel select
    x <= not a when "000",
         not b when "001",
         a and b when "010",
         a or b when "011",
         a nand b when "100",
         a nor b when "101",
         a xor b when "110",
         a xnor b when others;
end dataflow;
```


ALU αποτελούμενη από συστατικά (5)

```
-- alu.vhd
library IEEE;
use IEEE.std_logic_1164.all;

entity alu is
  port (
    a,b : in std_logic_vector(7 downto 0);
    cin : in std_logic;
    sel : in std_logic_vector(3 downto 0);
    y : out std_logic_vector(7 downto 0)
  );
end alu;

architecture structural of alu is
component arith is
  port (
    a,b: in std_logic_vector(7 downto 0);
    cin: in std_logic;
    sel: in std_logic_vector(2 downto 0);
    x  : out std_logic_vector(7 downto 0)
  );
end component;
component logic is
  port (
    a,b: in std_logic_vector(7 downto 0);
    sel: in std_logic_vector(2 downto 0);
    x  : out std_logic_vector(7 downto 0)
  );
end component;
```

```
component vmux is
  port (
    a,b: in std_logic_vector(7 downto 0);
    sel: in std_logic;
    x  : out std_logic_vector(7 downto 0)
  );
end component;
--
signal x1, x2: std_logic_vector(7 downto 0);
--
begin
  U1: arith port map (
    a   => a,  b   => b, cin => cin,
    sel => sel(2 downto 0),
    x   => x1
  );

  U2: logic port map (
    a   => a,  b   => b,
    sel => sel(2 downto 0),
    x   => x2
  );

  U3: vmux port map (
    a   => x1, b   => x2,
    sel => sel(3),
    x   => y
  );
end structural;
```