

Αρχιτεκτονική Η/Υ ΙΙ

**Ασκήσεις Πράξης:** Ενότητα 1

Πετρέλλης Νικόλαος

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Μηχανικών Πληροφορικής Τ.Ε.

**Χρηματοδότηση**

* Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Τ.Ε.Ι. Θεσσαλίας**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
* Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



**Σκοποί ενότητας**

Σκοπός της ενότητας αυτής είναι να κατανοήσουμε τη δομή και τον τρόπο λειτουργίας των μονάδων, και της αρχιτεκτονικής ενός υπολογιστικού συστήματος μέσω τη σχεδίασης, του προγραμματισμού, και της προσομοίωσης βασικών λειτουργικών μονάδων ενός υπολογιστικού συστήματος, με χρήση της γλώσσας περιγραφής υλικού **VHDL** (**V**HSIC **H**ardware **D**escription **L**anguage).

**Περιεχόμενα**

[1. Εισαγωγή 5](#_Toc411441798)

[2. Η γλώσσα VHDL 5](#_Toc411441799)

[2.1. Στοιχεία της γλώσσας VHDL 5](#_Toc411441800)

[2.1.1. Σχόλια (*Comments*) 5](#_Toc411441801)

[2.1.2. Αναγνωριστικά (*Identifiers*) 5](#_Toc411441802)

[2.1.3. Αριθμοί (*Numbers*) 5](#_Toc411441803)

[2.1.4. Χαρακτήρες (*Characters*) 6](#_Toc411441804)

[2.1.5. Αλφαριθμητικά (*Strings*) 6](#_Toc411441805)

[2.1.6. Bit αλφαριθμητικά (*Bit Strings*) 6](#_Toc411441806)

[2.1.7. Ακέραιοι τύποι (*Integer Types*) 6](#_Toc411441807)

[2.1.8. Φυσικοί τύποι (*Physical Types*) 7](#_Toc411441808)

[2.1.9. Τύποι κινητής υποδιαστολής (*Floating Point Types*) 7](#_Toc411441809)

[2.1.10. Πίνακες (*Arrays*) 7](#_Toc411441810)

[2.1.11. Εγγραφές (*Records*) 8](#_Toc411441811)

[2.1.12. Δηλώσεις αντικειμένων (*Object Declarations*) 8](#_Toc411441812)

[2.1.13. Εκφράσεις και τελεστές (*Expressions & Operators*) 9](#_Toc411441813)

[2.1.14. Η έκφραση IF (*If Statement*) 9](#_Toc411441814)

[2.1.15. Η έκφραση Case (*Case Statement*) 10](#_Toc411441815)

[2.1.16. Η έκφραση Loop (*Loop Statement*) 10](#_Toc411441816)

[2.1.17. Η έκφραση Null (*Null Statement*) 11](#_Toc411441817)

[2.1.18. Διαδικασίες και συναρτήσεις (*Procedures & Functions*) 12](#_Toc411441818)

[2.1.19. Πακέτο (*Package*) 12](#_Toc411441819)

[2.1.20. Το πακέτο Use (*Package Use*) 13](#_Toc411441820)

[2.1.21. Δηλώσεις οντοτήτων (*Entity Declarations*) 14](#_Toc411441821)

[2.1.22. Δηλώσεις αρχιτεκτονικής (*Architecture Declarations*) 14](#_Toc411441822)

[2.1.23. Δηλώσεις σημάτων (*Signal Declarations*) 15](#_Toc411441823)

[2.1.24. Μπλοκ (*Blocks*) 15](#_Toc411441824)

[2.1.25. Δηλώσεις στοιχείων-συνιστωσών (*Component Declarations*) 16](#_Toc411441825)

[2.1.26. Διεργασίες και η έκφραση Wait (*Processes & Wait Statement*) 16](#_Toc411441826)

[3. Ασκήσεις 16](#_Toc411441827)

[3.1. Άσκηση 1 17](#_Toc411441828)

[3.2. Άσκηση 2 17](#_Toc411441829)

[3.3. Άσκηση 3 18](#_Toc411441830)

[3.4. Άσκηση 4 19](#_Toc411441831)

[3.5. Άσκηση 5 20](#_Toc411441832)

[3.6. Άσκηση 6 21](#_Toc411441833)

[Σημειώματα 22](#_Toc411441834)

**Πίνακες**

[*Πίνακας 1* 9](#_Toc411441835)

[*Πίνακας 2* 17](#_Toc411441836)

[*Πίνακας 3* 18](#_Toc411441837)

[*Πίνακας 4* 20](#_Toc411441838)

**Εικόνες**

[*Εικόνα 1* 17](#_Toc411441839)

[*Εικόνα 2* 18](#_Toc411441840)

[*Εικόνα 3* 19](#_Toc411441841)

[*Εικόνα 4* 20](#_Toc411441842)

# Εισαγωγή

Σε αυτή τη ενότητα, γίνεται μία συνοπτική παρουσίαση των δυνατοτήτων της γλώσσας, έτσι ώστε να εξοικειωθούμε με τη γλώσσα VHDL (χωρίς να δίνεται ιδιαίτερη βαρύτητα στην εκμάθηση της, αλλά περισσότερο ως εργαλείο για την κατανόηση της αρχιτεκτονικής των υπολογιστικών συστημάτων), ενώ στη συνέχεια, στις ενότητες που ακολουθούν, αναλύονται περισσότερες εφαρμογές.

# Η γλώσσα VHDL

Η γλώσσα **VHDL** (**V**HSIC - **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit **HDL** - **H**ardware **D**escription **L**anguage), είναι η πρότυπη βιομηχανική γλώσσα περιγραφής ψηφιακών κυκλωμάτων. Μία πρώτη έκδοση του προτύπου αυτού, έγινε γνωστή το 1987 με το όνομα IEEE[1](http://standards.ieee.org/findstds/standard/1076-2008.html)1076, ενώ αργότερα το 1993 εμφανίστηκε μία βελτιωμένη έκδοσή της με το όνομα IEEE1164. Θα πρέπει να σημειώσουμε επίσης, ότι η γλώσσα αυτή έχει κληρονομήσει πολλά στοιχεία από τη γλώσσα προγραμματισμού ADA.

## Στοιχεία της γλώσσας VHDL

### Σχόλια (Comments)

Χρήση των συμβόλων ‘--’ .

### Αναγνωριστικά (Identifiers)

Χρήση των αναγνωριστικών ως δεσμευμένες λέξεις. Πρέπει να ακολουθούν τον κανόνα: αναγνωριστικό ::= γράμμα { [underline] γράμμα\_ή\_ψηφίο }

### Αριθμοί (Numbers)

decimal\_literal ::= integer [ . integer ] [ exponent ]

integer ::= digit { [ underline ] digit }

exponent ::= E [ + ] integer | E - integer

based\_literal ::= base # based\_integer [ . based\_integer ] # [ exponent ]

base ::= integer

based\_integer ::= extended\_digit { [ underline ] extended\_digit }

extended\_digit ::= digit | letter

1 IEEE Institute of Electrical and Electronic Engineers

Παραδείγματα:

0 1 123\_456\_789 987E6 -- integer literals

0.0 0.5 2.718\_28 12.4E-9 -- real literals

2#1100\_0100# 16#C4# 4#301#E1 -- the integer 196

2#1.1111\_1111\_111#E+11 16#F.FF#E2 -- the real number 4095.0

### Χαρακτήρες (Characters)

Εισαγωγή χαρακτήρα σε μονές αποστρόφους (*single-quote marks*).

Παραδείγματα: 'A' '\*' ''' ' '

### Αλφαριθμητικά (Strings)

Εισαγωγή χαρακτήρων σε διπλές αποστρόφους (*double-quote marks*).

Παραδείγματα:

"Ένα αλφαριθμητικό"

"" -- κενό αλφαριθμητικό.

"A string in a string: ""A string"". "

### Bit αλφαριθμητικά (Bit Strings)

Για τον καθορισμό των τιμών σε πίνακες τύπου bit.

Η σύνταξη έχει ως εξής:

bit\_string\_literal ::= base\_specifier " bit\_value "

base\_specifier ::= B | O | X

bit\_value ::= extended\_digit { [ underline ] extended\_digit }

B - binary, O -octal και X -hexadecimal.

Παραδείγματα:

B"1010110" -- το μήκος είναι 7.

O"126" -- το μήκος είναι 9, ισοδύναμο με B"001\_010\_110"

X"56" -- το μήκος είναι 8, ισοδύναμο με B"0101\_0110".

### Ακέραιοι τύποι (Integer Types)

Η σύνταξη για τον καθορισμό ακέραιων τύπων έχει ως εξής:

integer\_type\_definition ::= **range** simple\_expression **to** | **downto** simple\_expression

όπου τα όρια καθορίζονται ως: από -2147483647 έως +2147483647.

Παραδείγματα δηλώσεων ακεραίων:

**type** byte*\_int* **is range** 0 **to** 255;

**type** *signed\_*word*\_int* **is range** -32768 **to** 32767;

**type** *bit\_*index**is range** 31 **downto** 0;

Υπάρχει ένας προκαθορισμένος τύπου ακεραίου, ο τύπος **integer**.

### Φυσικοί τύποι (PhysicalTypes)

Ένας φυσικός τύπος είναι ένας αριθμητικός τύπος, ο οποίος αντιπροσωπεύει κάποια φυσική ποσότητα (*physical quantity*), όπως μάζα, μήκος, χρόνος ή τάση.

Η σύνταξη της δήλωσης έχει ως εξής:

**units**

αναγνωριστικό;

{ αναγνωριστικό = [ abstract\_literal ] unit\_name }

**end units**

Παραδείγματα:

**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**;

### Τύποι κινητής υποδιαστολής (Floating Point Types)

Η ακρίβεια (precision) είναι τουλάχιστον 6 δεκαδικών ψηφίων, και με εύρος από 1E38 έως +1E38.

Η σύνταξη της δήλωσης έχει ως εξής:

floating\_type\_definition := range\_constraint

Παραδείγματα:

**type** signal\_level**is range** -10.00 **to** +10.00;

**type** probability**is range** 0.0 **to** 1.0;

Υπάρχει μία προκαθορισμένη δήλωση τύπου **real**.

### Πίνακες (Arrays)

Ο πίνακας αποτελεί σύνολο στοιχείων ίδιου τύπου.

Η σύνταξη της δήλωσης έχει ως εξής:

* *unconstrained\_*array*\_definition:*

**array** (type\_mark range <>{ , type\_mark range <>} )

**of** element\_subtype\_indication

* constrained\_array\_definition:

**array** (discr\_range{,discr\_subtype\_indication **|** range})

**of** element\_subtype\_indication

Παραδείγματα:

**type** word**is array** (31 **downto** 0) **of bit**;

**type** memory**is array** (address) **of word**;

**type** transform**is array** (1 **to** 4, 1 **to** 4) **of real**;

**type** register\_bank**is array** (**byte range** 0 **to** 132) **of integer**;

Υπάρχουν οι ενσωματωμένοι τύποι ***bit\_vector*** και ***string***,οι οποίοι είναι πίνακες από bit και χαρακτήρες αντίστοιχα. Αυτοί οι τύποι πινάκων δεν έχουν συγκεκριμένο αριθμό στοιχείων κατά τη δήλωση του τύπου τους (*unconstrained arrays*).

Ο αριθμός των στοιχείων καθορίζεται όταν δημιουργείται το αντικείμενο.

### Εγγραφές (Records)

Συλλογή στοιχείων πιθανώς διαφορετικών τύπων.

Σύνταξη:

**record**

δηλώσεις στοιχείων

{ δηλώσεις στοιχείων }

**end record**

Παράδειγμα:

**type** instruction **is**

**record**

op\_code : processor\_op;

address\_mode : mode;

operand1, operand2: **integer range** 0 **to** 15;

**end record**;

### Δηλώσεις αντικειμένων (Object Declarations)

Ένα αντικείμενο είναι ένα στοιχείο σε μία VHDL περιγραφή, το οποίο έχει τιμή συγκεκριμένου τύπου. Υπάρχουν τρεις κατηγορίες αντικειμένων (classes of objects): σταθερές (constants), μεταβλητές (variables) και σήματα (signals).

Η σύνταξη μιας δήλωσης σταθεράς έχει ως εξής:

**constant** identifier\_list : subtype\_indication [ := expression ] ;

Παραδείγματα:

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

**constant** delay : Time:= 5 ns;

**constant** max\_size: **natural**;

Η σύνταξη δήλωσης µεταβλητών:

variable identifier\_list : subtype\_indication [ := expression ] ;

Παραδείγματα:

**variable** count: **natural** := 0;

**variable** trace : trace\_array;

### Εκφράσεις και τελεστές (Expressions & Operators)

Μία έκφραση (expression) στη VHDL είναι όπως και οι εκφράσεις σε άλλες γλώσσες προγραμματισμού, δηλαδή μία παράσταση ή formula, η οποία συνδυάζει πρωταρχικά στοιχεία με τελεστές.

Πίνακας 1

| Υψηλής προτεραιότητας: | \*\* | **abs** | **not** |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | \* | / | **mod** | **rem** |  |  |
|  | + (sign) | – (sign) |  |  |  |  |
|  | + | – | & |  |  |  |
|  | = | /= | < | <= | > | >= |
| Χαμηλής προτεραιότητας: | **and** | **or** | **nand** | **nor** | **xor** |  |

Οι τελεστές πολλαπλασιασμού (\*) και διαίρεσης (/), εφαρμόζονται σε ακεραίους (integer), κινητής υποδιαστολής (floating point), και φυσικούς τύπους (physical types). Οι τελεστές modulus (mod) και υπολοίπου διαίρεσης (rem), εφαρμόζονται μόνο σε ακέραιους τύπους. Ο τελεστής *εκχώρησης* σήματος <= (signal *assignment operator*) βασικά καθορίζει μία συσχέτιση μεταξύ σημάτων.

Η απόλυτη τιμή (abs) εφαρμόζεται μόνο σε αριθμητικούς τύπους (numeric type). Ο

εκθετικός τελεστής (\*\*) (exponentiation) μπορεί να έχει ένα ακέραιο (integer) ή κινητής υποδιαστολής (floating point) αριστερό τελεστή, αλλά πρέπει να έχει ένα ακέραιο δεξιό τελεστή.

### Η έκφραση IF (If Statement)

Η έκφραση **if** επιτρέπει την επιλογή εκφράσεων προς εκτέλεση, ανάλογα με μία ή περισσότερες συνθήκες.

Σύνταξη:

**if** συνθήκη **then**

σειρά εκφράσεων

{ **elsif** συνθήκη **then**

σειρά εκφράσεων }

[**else**

σειρά εκφράσεων]

**end if** ;

Παράδειγμα:

**if** (day = sunday) **then**

weekend:= true;

**elsif** (day = saturday) **then**

weekend:= true;

**else** weekday:= true;

**end if**;

### Η έκφραση Case (Case Statement)

Η έκφραση **case** επιτρέπει την επιλογή εκφράσεων προς εκτέλεση, ανάλογα με την τιμή της έκφρασης επιλογής.

Σύνταξη:

**case** έκφραση **is**

case\_statement\_alternative

{ case\_statement\_alternative }

**end case** ;

**when** επιλογές =>

σειρά εκφράσεων

Παραδείγματα:

**case** element\_colour **of**

**when** red =>

εκφράσεις για το red;

**when** green |blue =>

εκφράσεις για το green ή το blue;

**when** orange **to** turquoise =>

εκφράσεις για αυτά τα χρώματα;

**end case**;

**case** opcode**of**

**when** X"00" => perform\_add; -- κάνε πρόσθεση

**when** X"01" => perform\_subtract; -- κάνε αφαίρεση

**when** others=> signal\_illegal\_opcode; -- σφάλμα κώδικα λειτουργίας

**end case**;

### Η έκφραση Loop (Loop Statement)

Η εντολή επαναλήψεων στη VHDL.

Σύνταξη:

[ loop\_label : ]

[ σχήμα των επαναλήψεων ] **loop**

ακολουθία εκφράσεων

**end loop** [ loop\_label ] ;

όπου το σχήμα των επαναλήψεων θα μπορούσε να είναι:

**while** condition

| **for** loop\_identifier **in** discrete\_range

Παράδειγμα:

**Loop**

do\_something;

**end loop**;

Η έκφραση επαναλήψεων με τη **while** επιτρέπει να γίνει ο έλεγχος μιας συνθήκης πριν από κάθε επανάληψη.

Παράδειγμα:

**while** index < length**and** str(index) /= ' ' **loop**

*index* := index+ 1;

**end loop**;

Η έκφραση επαναλήψεων με τη **for** καθορίζει ένα συγκεκριμένο αριθμό επαναλήψεων.

Παράδειγμα:

**for** item**in** 1 **to** last\_item**loop**

table(item) := 0;

**end loop**;

Η έκφραση **next** τερματίζει την εκτέλεση της τρέχουσας επανάληψης, και ξεκινά την επόμενη επανάληψη.

Η έκφραση **exit** τερματίζει την εκτέλεση της τρέχουσας επανάληψης, και τερματίζει το loop.

Η σύνταξη των παραπάνω εκφράσεων έχει ως εξής:

next\_statement ::= **next** [ loop\_label ] [ **when** condition ] ;

exit\_statement ::= **exit** [ loop\_label ] [ **when** condition ] ;

Παραδείγματα:

**for** i**in** 1 **to** max\_str\_len**loop**

a(i) := buf(i);

**exit when** buf(i) = NUL;

**end loop**;

outer\_loop : **loop**

inner\_loop : **loop**

do\_something;

**next** outer\_loop **when** temp = 0;

do\_something\_else;

**end loop** inner\_loop;

**end loop** outer\_loop;

### Η έκφραση Null (Null Statement)

Η έκφραση **null** δεν έχει κανένα αποτέλεσμα (no effect). Μπορεί να χρησιμοποιηθεί για να δείξει ρητά ότι δεν είναι απαραίτητη κάποια πράξη ή ενέργεια.

 Παράδειγμα:

**case** controller\_command **is**

**when** forward => engage\_motor\_forward;

**when** reverse => engage\_motor\_reverse;

**when** idle => **null**;

**end case**;

### Διαδικασίες και συναρτήσεις (Procedures & Functions)

Όπως και άλλες γλώσσες προγραμματισμού, η VHDL παρέχει δυνατότητες υποπρογραμμάτων (subprograms) στη μορφή διαδικασιών και συναρτήσεων.

Επίσης, η VHDL παρέχει και τη δυνατότητα ομαδοποίησης των δηλώσεων και των

αντικειμένων (declarations και objects) σε λειτουργικές μονάδες (modular units) με τη μορφή του πακέτου (package).

Σύνταξη διαδικασιών και συναρτήσεων:

**procedure** designator [ ( formal\_parameter\_list ) ]

| **function** designator [ ( formal\_parameter\_list ) ] **return** type\_mark

Η σύνταξη των τυπικών παραμέτρων (formal parameters) ενός υποπρογράμματος είναι:

Παράδειγμα δήλωσης διαδικασίας µε παραμέτρους:

**procedure** increment\_reg(**variable** reg: **inout** word\_32;

**constant** incr: **in integer** := 1);

Παραδείγματα κλήσης σε υποπρόγραμμα:

increment\_reg(index\_reg, offset–2); -- add value to index\_reg

increment\_reg(prog\_counter); -- add 1 (default) to prog\_counter

increment\_reg(incr => offset–2, reg => index\_reg);

increment\_reg(reg => prog\_counter);

Παράδειγμα δήλωση συνάρτησης:

**function** byte\_to\_int(**byte** : word\_8) **return integer**;

Όταν ένα υποπρόγραμμα καλείται, οι εκφράσεις στο σώμα του θα εκτελεστούν έως ότου βρεθεί το τέλος της λίστας των εκφράσεων, ή εκτελεστεί η εντολή **return**.

Η σύνταξη της έκφρασης **return** έχει ως εξής:

return\_statement ::= **return** [ expression ] ;

Παράδειγμα σώματος συνάρτησης:

**function** byte\_to\_int(**byte** : word\_8) **return integer is**

**variable** result: **integer** := 0;

**begin**

**for** index **in 0 to 7 loop**

result **: =** result**\*2 +bit’pos(byte(**index**));**

**end loop**;

**return** result;

**end** byte\_to\_int;

### Πακέτο (Package)

Ένα πακέτο είναι ένα σύνολο τύπων, σταθερών, υποπρογραμμάτων, και άλλων στοιχείων, συνήθως με στόχο να υλοποιήσουν κάποια συγκεκριμένη εργασία, ή για την ομαδοποίηση μιας ομάδας σχετιζόμενων στοιχείων. Ένα πακέτο μπορεί να έχει δύο τμήματα: μία δήλωση πακέτου (package declaration), και ένα σώμα πακέτου (package body).

Σύνταξη μιας δήλωσης πακέτου:

**package** αναγνωριστικό **is**

τμήμα δηλώσεων του πακέτου

**end** [ απλό όνομα του πακέτου ] ;

Παράδειγμα δήλωσης πακέτου:

**package** data\_types**is**

**subtype** address**is** bit\_vector(24 **downto** 0);

**subtype** data**is** bit\_vector(15 **downto** 0);

**constant** vector\_table\_loc : address;

**function** data\_to\_int(value: **data**) **return integer**;

**function** int\_to\_data(value: **integer**) **return data**;

**end data\_types**;

Η σύνταξη του σώματος πακέτου είναι:

**package body** απλό όνομα του πακέτου **is**

τμήμα δηλώσεων του σώματος του πακέτου

**end** [απλό όνομα του πακέτου] ;

Παράδειγμα:

Το σώμα του πακέτου data\_types που δόθηκε παραπάνω μπορεί να γραφεί ως εξής:

**package body** data\_types**is**

**constant** vector\_table\_loc : address:= X"FFFF00";

**function** data\_to\_int(value: **data**) **return integer is**

**body of** data\_to\_int

**end** data\_to\_int;

**function** int\_to\_data(value: **integer**) **return data is**

**body of** int\_to\_data

**end** int\_to\_data;

**end** data\_types;

### Το πακέτο Use (*Package Use*)

Όταν ένα πακέτο έχει δηλωθεί, τα στοιχεία που έχουν δηλωθεί μέσα σε αυτό, μπορούν να χρησιμοποιηθούν τοποθετώντας μπροστά από το όνομά τους, το όνομα του πακέτου.

Παράδειγμα:

**variable** PC : data\_types.address;

int\_vector\_loc := data\_types.vector\_table\_loc + 4\*int\_level;

offset := data\_types.data\_to\_int(offset\_reg);

Συχνά είναι βολικό να μπορούμε να αναφερόμαστε στα ονόματα ενός πακέτου, χωρίς να χρειάζεται να αναφέρουμε την κάθε χρήση τους με το όνομα του πακέτου. Αυτό μπορεί να γίνει κάνοντας χρήση της έκφρασης **use,** στο τμήμα των δηλώσεων.

Η σύνταξη έχει ως εξής:

use\_clause ::= **use** selected\_name { , prefix . suffix } ;

Εάν όλα τα δηλωμένα ονόματα σε ένα πακέτο πρόκειται να χρησιμοποιηθούν, μπορεί να χρησιμοποιηθεί στο τέλος η έκφραση all.

Παράδειγμα:

**use** data\_types**.all**;

### Δηλώσεις οντοτήτων (Entity Declarations)

Ένα ψηφιακό σύστημα, συνήθως σχεδιάζεται ως μία ιεραρχική συλλογή από μονάδες (modules). Κάθε μονάδα έχει ένα σύνολο από θύρες (ports), οι οποίες αποτελούν τη διασύνδεσή του. Στη γλώσσα VHDL, η οντότητα (entity) είναι μία τέτοια μονάδα που μπορεί να χρησιμοποιηθεί σαν ένα στοιχείο (component) σε ένα σχέδιο, ή μπορεί να αποτελεί τη μονάδα στο υψηλότερο επίπεδο της σχεδίασης (top level module).

Η σύνταξη δήλωσης μιας οντότητας έχει ως εξής:

**entity** identifier **is**

επικεφαλίδα της οντότητας

τμήμα δηλώσεων της οντότητας

[ **begin**

τμήμα εκφράσεων της οντότητας]

**end** [ entity\_simple\_name ] ;

Παραδείγματα δηλώσεων οντοτήτων:

**entity** processor**is**

**generic** (max\_clock\_freq: frequency:= 30 MHz);

**port** (clock: **in bit**;

address: **out integer**;

data: **inout** word\_32;

control: **out** proc\_control;

ready: **in bit**);

**end** processor;

### Δηλώσεις αρχιτεκτονικής (Architecture Declarations)

Περισσότερες από μία εφαρμογές οντοτήτων μπορούν να περιγραφούν σε σώματα αρχιτεκτονικής. Κάθε σώμα αρχιτεκτονικής μπορεί να περιγράψει μία διαφορετικά άποψη της οντότητας.

Η σύνταξη της δήλωσης σώματος αρχιτεκτονικής έχει ως εξής:

**architecture** αναγνωριστικό **of** όνομα οντότητας **is**

τμήμα δηλώσεων της αρχιτεκτονικής

**begin**

τμήμα εκφράσεων της αρχιτεκτονικής

**end** [ απλό όνομα αρχιτεκτονικής ] ;

Οι δηλώσεις στο σώμα της αρχιτεκτονικής, ορίζουν στοιχεία που μπορούν να χρησιμοποιηθούν στη δόμηση της περιγραφής σχεδίασης.

### Δηλώσεις σημάτων (Signal Declarations)

Τα σήματα χρησιμοποιούνται για τη σύνδεση υπο-μονάδων σε μία σχεδίαση.

Σύνταξη:

**signal** λίστα αναγνωριστικών : subtype\_indication [**register** | **bus**] [ := expression ] ;

### Μπλοκ (Blocks)

Οι υπο-μονάδες σε ένα σώμα αρχιτεκτονικής μπορούν να περιγραφούν ως μπλοκ.

Σύνταξη:

ετικέτα του μπλοκ:

**block** [ ( guard\_expression ) ]

επικεφαλίδα του μπλοκ

τμήμα δηλώσεων του μπλοκ

**begin**

τμήμα εκφράσεων του μπλοκ

**end block** [ ετικέτα του μπλοκ ] ;

Παράδειγμα μιας δομής αρχιτεκτονικής μιας οντότητας επεξεργαστή:

**architecture** block\_structure **of** processor **is**

**type** data\_path\_control is … ;

**signal** internal\_control : data\_path\_control;

**begin**

control\_unit : **block**

**port** (clk : **in** bit;

bus\_control : **out** proc\_control;

bus\_ready : **in** bit;

control : **out** data\_path\_control);

**port map** (clk => clock, bus\_control => control, bus\_ready =>

ready;

control => internal\_control);

declarations for control\_unit

**begin**

statements for control\_unit

**end block** control\_unit;

data\_path : **block**

**port** (address : **out** integer;

data : **inout** word\_32;

control : **in** data\_path\_control);

**port map** (address => address, data => data, control =>

internal\_control);

declarations for data\_path

**begin**

statements for data\_path

**end block** data\_path;

**end** block\_structure;

### Δηλώσεις στοιχείων-συνιστωσών (Component Declarations)

Ένα σώμα αρχιτεκτονικής μπορεί να περιέχει και να κάνει χρήση και άλλων οντοτήτων οι οποίες περιγράφονται ξεχωριστά, και τοποθετούνται σε σχεδιαστικές βιβλιοθήκες (design libraries). Για να επιτευχθεί αυτό στο σώμα της αρχιτεκτονικής, πρέπει να δηλωθεί ένα στοιχείο (component), το οποίο μπορεί να θεωρηθεί ως ένα πρότυπο (template).

Σύνταξη:

**component** αναγνωριστικό

[ local\_generic\_clause ]

[ local\_port\_clause ]

**end component** ;

Παράδειγμα:

**component** nand3

**generic** (Tpd: Time:= 1 ns);

**port** (a, b, c: **in** logic\_level;

*y* : **out** logic\_level);

**end component**;

### Διεργασίες και η έκφραση Wait (Processes & Wait Statement)

Η πρωταρχική μονάδα σε μία περιγραφή λειτουργίας (behavioral description) στη γλώσσα VHDL είναι η διεργασία (process). Μία διεργασία είναι μία ακολουθιακή ενότητα κώδικα, η οποία μπορεί να ενεργοποιηθεί µε τις αλλαγές στην κατάστασή της.

Σύνταξη:

[ ετικέτα διεργασίας : ]

**process** [ ( sensitivity\_list ) ]

*τμήμα δηλώσεων της διεργασίας*

**begin**

τμήμα εκφράσεων της διεργασίας

**end process** [ετικέτα διεργασίας] ;

Μία διεργασία μπορεί να αναβληθεί προσωρινά η εκτέλεσή της, με την εντολή **wait**.

**wait** [**on** signal\_name { , signal\_name }] [**until** condition] [**for** time\_expression] ;

# Ασκήσεις

## Άσκηση 1

Ας θεωρήσουμε ότι μας δίνεται το παρακάτω κύκλωμα με εξίσωση

 ( $f=x\_{1}x\_{2}+\overbar{x\_{2}}x\_{3}$ ), και πρέπει να γράψουμε ένα πρόγραμμα σε VHDL που το περιγράφει.

Πίνακας 2

| **X1** | **X2** | **X3** | **f** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



Εικόνα 1

Για την υλοποίηση της άσκησης δημιουργούμε το παρακάτω πρόγραμμα σε VHDL:

**ENITITY** example1 **IS**

**PORT** (x1, x2, x3 : **IN** BIT ;

f : **OUT** BIT ) ;

**END** example1 ;

**ARCHITECTURE** LogicFunc **OF** example1 **IS**

**BEGIN**

f <= (x1 **AND** x2) **OR** (**NOT** x2 **AND** x3) ;

**END** LogicFunc ;

Το πρόγραμμα αυτό µε όνομα example1.vhd***,*** αποτελείται από δύο βασικά τμήματα: το τμήμα της οντότητας και της αρχιτεκτονικής.

Η **οντότητα** (**ENTITY**) με το όνομα example1, ορίζει τα σήματα εισόδου και εξόδου του κυκλώματος. Παρατηρούμε επίσης, ότι η οντότητα αυτή διαθέτει τρεις θύρες (**ports**) εισόδου (**in**), και μία θύρα εξόδου (**out**). Οι θύρες εισόδου αντιστοιχούν στις μεταβλητές x1, x2, x3, που αποτελούν τα σήματα εισόδου και έχουν τύπο **bit**, ενώ η θύρα εξόδου ονομάζεται f και είναι επίσης τύπου **bit**.

Η **αρχιτεκτονική** (**ARCHITECTURE**) της άσκησής μας με το όνομα LogicFunc*,*παρέχει πληροφορίες που εξηγούν τι παριστάνει το κύκλωμα, δηλαδή ορίζει τη λειτουργικότητα του σχεδίου με τη βοήθεια μιας λογικής έκφρασης.

## Άσκηση 2

Δίνεται το ακόλουθο λογικό κύκλωμα.



Εικόνα 2

Να γράψετε τον κώδικα σε VHDL που το περιγράφει.

Ο κώδικας VHDL για το λογικό κύκλωμα 4 εισόδων έχει ως εξής:

**ENTITY** examlpe2 **IS**

**PORT** ( x1, x2, x3, x4 : **IN** BIT;

f, g : **OUT** BIT ) ;

**END** example2 ;

**ARCHITECTURE** LogicFunc **OF** example2 **IS**

**BEGIN**

f <= (x1 **AND** x3) **OR** (**NOT** x3 **AND** x2) ;

g <= (**NOT** x3 **OR** x1) **AND** (**NOT** x3 **OR** x4) ;

**END** LogicFunc ;

## Άσκηση 3

Δίνεται το ακόλουθο λογικό κύκλωμα, καθώς και η λογική συνάρτηση που το περιγράφει $f= \overbar{x\_{2}}x\_{3}+ x\_{1}\overbar{x\_{3}}$ καθώς και ο πίνακας αληθείας του.

Πίνακας 3

| Row number | x1 | x2 | x3 | $$f(x\_{1}, x\_{2}, x\_{3})$$ |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 0 |



Εικόνα 3

Να γράψετε τον κώδικα σε VHDL που το περιγράφει.

Ο κώδικας VHDL για το λογικό κύκλωμα 3 εισόδων έχει ως εξής:

**ENTITY** func1 **IS**

**PORT** ( x1, x2, x3 : **IN** BIT ;

f : **OUT** BIT) ;

**END** func1 ;

**ARCHITECTURE** LogicFunc **OF** func1 **IS**

**BEGIN**

f <= (**NOT** x1 **AND** **NOT** x2 **AND** x3) **OR**

(x1 **AND** **NOT** x2 **AND** **NOT** x3) **OR**

(x1 **AND** **NOT** x2 **AND** x3) **OR**

(x1 **AND** x2 **AND** **NOT** x3) ;

**END** LogicFunc ;

Μία άλλη υλοποίηση του κώδικα VHDL για το παραπάνω κύκλωμα, χρησιμοποιώντας αντικείμενα τύπου STD\_LOGIC.

**LIBRARY** ieee ;

**USE** ieee.std\_logic\_1164.**all** ;

**ENTITY** func2 **IS**

**PORT** ( x1, x2, x3 : **IN** STD\_LOGIC ;

f : **OUT** STD\_LOGIC ) ;

**END** func2 ;

**ARCHITECTURE** LogicFunc **OF** func2 **IS**

**BEGIN**

f <= (**NOT** x1 **AND** **NOT** x2 **AND** x3) **OR**

(x1 **AND** **NOT** x2 **AND NOT** x3) **OR**

(x1 **AND NOT** x2 **AND** x3) **OR**

(x1 **AND** x2 **AND NOT** x3) ;

**END** LogicFunc ;

## Άσκηση 4

Να σχεδιαστεί σε VHDL ένα στοιχείο (component) με τις ακόλουθες εισόδους και εξόδους:



Εικόνα 4

Το στοιχείο αυτό θα πρέπει να έχει την παρακάτω συμπεριφορά:

Πίνακας 4

| **sel** | **q** |
| --- | --- |
| 00 | a **xor** b |
| 01 | a **or** b |
| 10 | a **nor** b |
| 11 | a **and** b |
| others | “XX” |

Μία λύση που παρουσιάζεται είναι µε χρήση της case (θα μπορούσε η υλοποίηση να γίνει και µε **if** ή και **when else** εκφράσεις).

**Architecture** rtl **of** c1 **is**

**begin**

**process**(a,b,sel)

**begin**

**case** sel **is**

**when** “00” =>q<= a **xor** b;

**when** “01” =>q<= a **or** b;

**when** “10” =>q<= a **nor** b;

**when** “11” =>q<= a **and** b;

**when** “others” =>q<= “XX”;

**end case**;

**end process**;

**end**;

## Άσκηση 5

Να σχεδιαστεί σε VHDL το στοιχείο NAND.

**Entity** nand\_comp **is**

**port (a\_in\_b\_in: in std\_logic; --είσοδοι**

**a\_out out std\_logic); --έξοδοι**

**end;**

**Architecture** nand\_behv **of** nand\_comp **is**

**signal** int: std\_logic; -- δήλωση εσωτερικού σήματος

**begin**

int <= a\_in **and** b\_in;

a\_out <= **not** int;

**end**;

## Άσκηση 6

Να σχεδιαστεί σε VHDL το στοιχείο XOR.

Πρώτος τρόπος υλοποίησης:

-- 2 input e**x**clusive **or**

-- Μοντελοποίηση σε RTL επίπεδο

**entity** x\_or **is**

**port** (

in1 : **in bit** ;

in2 : **in bit** ;

out1 : **out bit**);

**end** x\_or;

**architecture** rtl **of** x\_or **is**

**begin**

out1 <= in1 **xor** in2 **after** 10 ns;

**end** rtl;

-- Μοντελοποίηση σε behavioral επίπεδο.

**entity** x\_or **is**

**port** (

in1 : **in bit** ;

in2 : **in bit** ;

out1 : **out bit**) ;

**end** x\_or;

**architecture** behavior **of** x\_or **is**

**begin**

**process**(in1, in2)

**begin**

**if** in1 = in2 **then**

out1 <= '0' **after** 10 ns;

**else** out1 <= '1' **after** 10 ns;

**end if**;

**end process**;

**end behavior**;

**Τέλος Ενότητας**

Επεξεργασία: Σοφιανίδου Γεωργία

 

# Σημειώματα

**Σημείωμα Ιστορικού Εκδόσεων Έργου**

Το παρόν έργο αποτελεί την έκδοση 1.01.

**Σημείωμα Αναφοράς**

Copyright Τεχνολογικό Εκπαιδευτικό Ίδρυμα Θεσσαλίας, Πετρέλλης Χ. Νικόλαος, 2015. Πετρέλλης Χ. Νικόλαος. «Αρχιτεκτονική Η/Υ ΙΙ». Έκδοση: 1.0. Λάρισα 01/03/2015. Διαθέσιμο από τη δικτυακή διεύθυνση: <http://cdev.teilar.gr/courses/TMA112/index.php>.

**Σημείωμα Αδειοδότησης**

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λπ., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] [http://creativecommons.org/licenses/by-nc-sa/4.0/](http://creativecommons.org/licenses/by-nc-sa/4.0/deed.el)

Ως **Μη Εμπορική** ορίζεται η χρήση:

που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο,

που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο,

που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο.

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

**Διατήρηση Σημειωμάτων**

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

* το Σημείωμα Αναφοράς,
* το Σημείωμα Αδειοδότησης,
* τη Δήλωση Διατήρησης Σημειωμάτων,
* το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει).

μαζί με τους συνοδευόμενους υπερσυνδέσμους.