

1ο Εργαστήριο Σχεδίασης Ψηφιακών Συστημάτων

1^η Διάλεξη

Εισαγωγή

Βασιλόπουλος Διονύσης

Ε.ΔΙ.Π. Τμήματος Πληροφορικής & Τηλεπικοινωνιών

Αντικείμενο Εργαστηρίου

- Πρακτικό μέρος του μαθήματος Σχεδίαση Ψηφιακών Συστημάτων
- Συνοπτική επανάληψη θεωρίας (όπου χρειάζεται)
- Προγραμματισμός στη γλώσσα VHDL
- Περιγράφουμε (σε VHDL) ένα Σύστημα Λογικού Κυκλώματος/Ψηφιακό Κύκλωμα
- Με τι προγραμματίζουμε; Με το εργαλείο Vivado
- Τι προγραμματίζουμε; Hardware (κάρτες FPGA)
- Τι κάνει το Λογικό Κύκλωμα; Δίνει λύση σε μία πραγματική ανάγκη

Αντικείμενο Μαθήματος

Πραγματικές Ανάγκες

Ψηφιακά Συστήματα



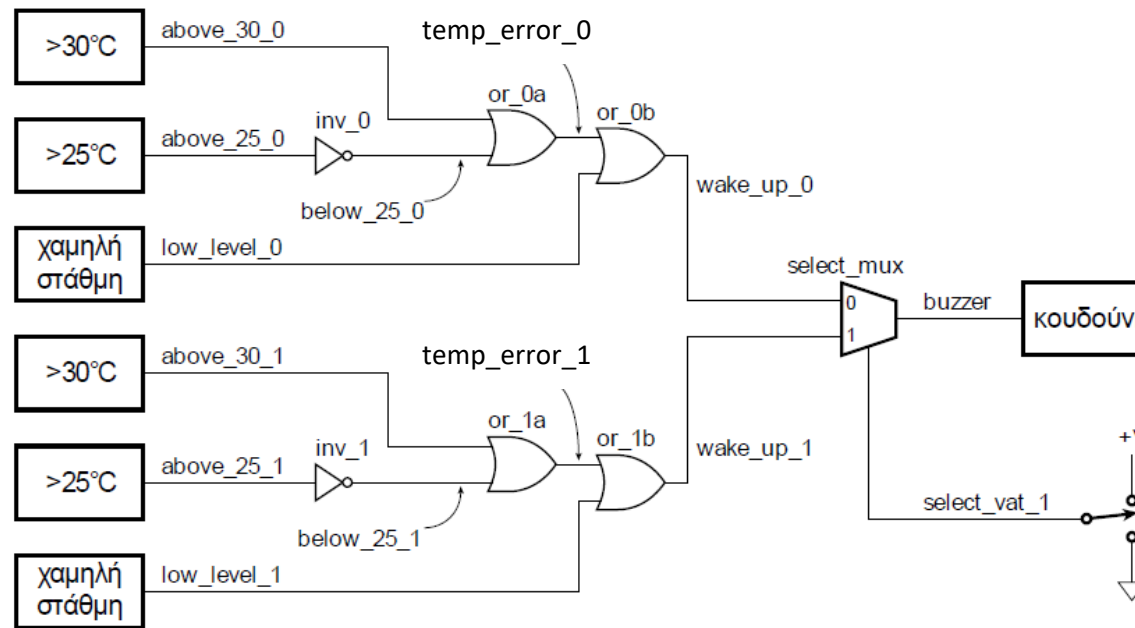
Αντικείμενο Μαθήματος

Εργαστήριο – Περιεχόμενο μαθήματος

- **Εκμάθηση Γλώσσας Προγραμματισμού VHDL**
- **Ανάπτυξη εφαρμογών** για κατανόηση του μοντέλου της VHDL
- **Χρήση του εργαλείου VIVADO** για την ανάπτυξη εφαρμογών
- **Συνοπτική παρουσίαση σε βασικά στοιχεία της θεωρίας Λογικής Σχεδίασης** (δυαδικό σύστημα αρίθμησης, λογικές πύλες, συνδυαστικά κυκλώματα, ακολουθιακά κυκλώματα, μνήμες)
- **Προγραμματισμός κάρτας FPGA** (μέσω VHDL+VIVADO) σε πραγματικές συνθήκες (εργαστήριο)

Αντικείμενο Μαθήματος

Λογικά κυκλώματα



Αντικείμενο Μαθήματος

Προγραμματισμός σε VHDL

Χρήση πρότυπης βιβλιοθήκης

package

Entity name

Port list

Port type

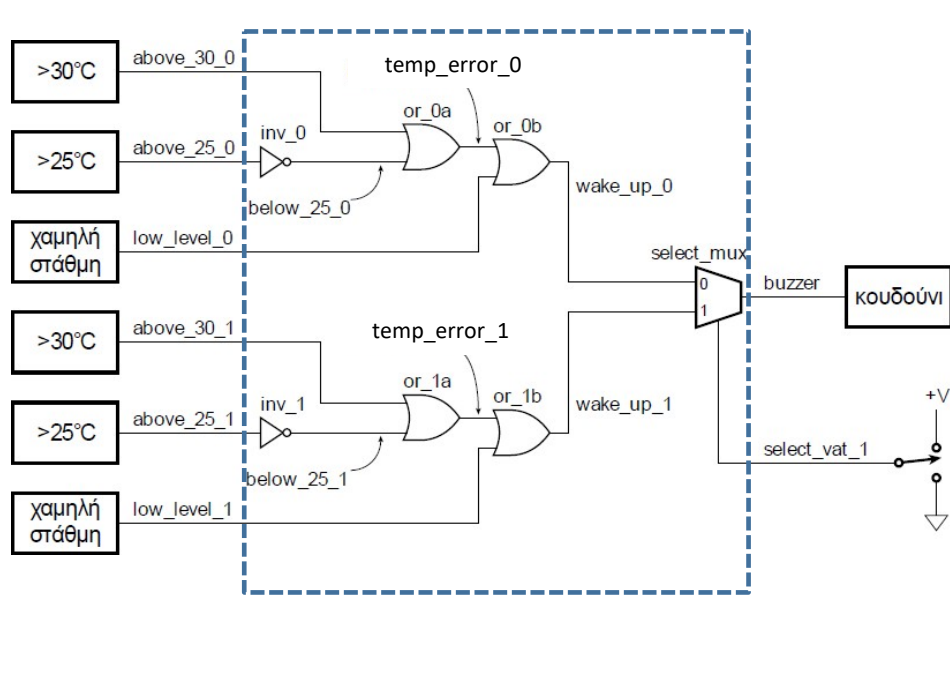
Port mode

Port name

```
library ieee; use ieee.std_logic_1164.all;
entity vat_buzzer is
  port ( above_25_0, above_30_0,
         low_level_0   : in std_logic;
         above_25_1, above_30_1,
         low_level_1   : in std_logic;
         select_vat_1  : in std_logic;
         buzzer        : out std_logic );
end entity vat_buzzer;
```

Ψηφιακά Συστήματα

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Entity: Input/Output



Entity name

Port list

```
library ieee; use ieee.std_logic_1164.all;
entity vat_buzzer is
    port (
        above_25_0, above_30_0,
        low_level_0    : in std_logic;
        above_25_1, above_30_1,
        low_level_1    : in std_logic;
        select_vat_1   : in std_logic;
        buzzer         : out std_logic );
end entity vat_buzzer;
```

Χρήση πρότυπης βιβλιοθήκης

package

Port type

Port name

Port mode

Αντικείμενο Μαθήματος

Εργαστήριο Σχεδίασης Ψηφιακών Συστημάτων – Οργάνωση μαθήματος 1/4

- Διδάσκων Εργαστηρίου: Ε.ΔΙ.Π. Διονύσης Βασιλόπουλος (denis@di.uoa.gr) – Γραφείο: Α33
 - Ώρες γραφείου: Τρίτη 15:00-16:00 και Πέμπτη 13:00 – 15:00, κατόπιν επικοινωνίας
- 3 διαλέξεις εισαγωγικές και 5 σετ Θεωρίας/εργαστηρίου (θεωρία/παράδοση-χρήση εργαλείου Vivado, επίλυση άσκησης-προγραμματισμός κάρτας) (αρχικός προγραμματισμός μαθήματος).
- Σύνολο 8 διαλέξεις στην Αίθουσα Β και 5 μαθήματα στο Εργαστήριο Ψηφιακής Σχεδίασης & ΗΥ Υψηλών Επιδόσεων (Αναγνωστήριο)

Αντικείμενο Μαθήματος

Εργαστήριο Σχεδίασης Ψηφιακών Συστημάτων – Οργάνωση μαθήματος 2/4

Ο τελικός **βαθμός** προκύπτει από:

- Εργαστηριακές ασκήσεις: μέγιστο βαθμού 7.
 - Προπτυχιακοί φοιτητές: Περιλαμβάνει 3 εργασίες (συνδυαστικά, ακολουθιακά και υλοποίηση cru 4 εντολών).
 - Μεταπτυχιακοί φοιτητές: Περιλαμβάνει 3 εργασίες (συνδυαστικά, ακολουθιακά και υλοποίηση πλήρης cru ενός κύκλου).
- Οι πρώτες δύο ασκήσεις (συνδυαστικά, ακολουθιακά) είναι κοινές.
- Γραπτές εξετάσεις: μέγιστο βαθμού 3.
 - Εξέταση μέσω eclass. Ερωτήσεις πάνω στην άσκηση αλλά και τη θεωρία.
- Εάν ο βαθμός των ασκήσεων έχει διαφορά πάνω από 30% σε σχέση με το βαθμό των γραπτών εξετάσεων τότε ισχύει ΜΟΝΟ ο βαθμός των εξετάσεων.

Αντικείμενο Μαθήματος

Εργαστήριο Σχεδίασης Ψηφιακών Συστημάτων – Οργάνωση μαθήματος 3/4

<https://eclass.uoa.gr/> (eclass)

- Ανέβασμα διαφανειών/υλικού και ασκήσεων
- Επικοινωνία μέσω eclass , email
- Παράδοση εργασιών
- Εργαλείο λογισμικού για το μάθημα: VIVADO
 - Στο eclass θα ανέβει αρχείο που περιγράφει τον τρόπο με τον οποίο θα εγκαταστήσουμε το εργαλείο στον υπολογιστή μας (απαραίτητο).
 - Υπάρχει έκδοση για Windows (**2022.2**) και Linux (όχι για Mac)
 - Η έκδοση του Vivado πρέπει **οπωσδήποτε** να είναι η 2022.2



<https://delos.uoa.gr/opendelos/>

- Καταγραφή μαθημάτων, διαθέσιμο υλικό και στο eclass το Σ/Κ.

Αντικείμενο Μαθήματος

Εργαστήριο Σχεδίασης Ψηφιακών Συστημάτων – Οργάνωση μαθήματος 4/4

Α.Παρουσιάσεις (**ΒΑΣΙΚΟ ΕΓΧΕΙΡΙΔΙΟ**), κώδικας και άλλα κείμενα που ανεβαίνουν στο eclass

Β. Προτεινόμενη Βιβλιογραφία

1. **Ψηφιακή Σχεδίαση και Αρχιτεκτονική Υπολογιστών**, S.L.Harris, D.M.Harris, Εκδόσεις ARM®, 2019
2. **Ψηφιακή Σχεδίαση. Ενσωματωμένα Συστήματα με VHDL**, Peter J. Ashenden, Επιστημονική Επιμέλεια – Μετάφραση: Μ. Ψαράκης, Ν. Κρανίτης, Δ. Γκιζόπουλος, Εκδόσεις Νέων Τεχνολογιών, 2010
3. **Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL**, S. Brown, Z.Vranesic, Εκδόσεις Τζιόλα, 2024

Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση

- **Ψηφιακή (Digital):** κυκλώματα που χρησιμοποιούν δύο επίπεδα τάσης (π.χ. 0V, +5V) για αναπαράσταση της πληροφορίας
 - Λογική (Logic): χρήση τιμών αληθείας (0/1, true/false) και κανόνων λογικής (άλγεβρα boole) για την ανάλυση των κυκλωμάτων
- **Σχεδίαση (Design):** ανταποκρίνονται σε λειτουργικές απαιτήσεις ενώ ταυτόχρονα ικανοποιούν περιορισμούς
 - Περιορισμοί: απόδοση, μέγεθος (κόστος), ισχύς, κλπ.

Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση

- **Δυαδικό Σύστημα Αρίθμησης (0,1)**
- **Λογικοί κανόνες** (Άλγεβρα Boole: ΝΑΙ/ΟΧΙ, ΙΣΧΥΕΙ/ΔΕΝ ΙΣΧΥΕΙ)
- **Πύλες** (Στοιχειώδη κυκλώματα, υλοποιούν την άλγεβρα boole σε επίπεδο υλικού, κάνουμε στοιχειώδεις πράξεις στο δυαδικό σύστημα. Στη βάση τους υπάρχει το Περνάει Ρεύμα/Δεν περνάει ρεύμα ή Υπάρχει Τάση/Γείωση)
- **Συνδυαστικά κυκλώματα** (π.χ. ένα κύκλωμα που προσθέτει δύο δυαδικούς αριθμούς). Έχει εισόδους που παράγουν κάποιες τιμές εξόδων.
- **Ακολουθιακά κυκλώματα.** Η τιμή της εξόδου εξαρτάται από τις εισόδους αλλά και από την προηγούμενη τιμή της εξόδου. Συνήθως υπάρχει συγχρονισμός στο πότε αλλάζουν οι τιμές στην είσοδο και στο πότε ελέγχουμε την τιμή στην έξοδο. Συνήθως υπάρχει αποθήκευση δεδομένων σε Flip/Flop. Παράδειγμα είναι το χρονόμετρο.

Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση – Είδη κυκλωμάτων

- **Συνδυαστικά** κυκλώματα: Οι τιμές των εξόδων εξαρτώνται μόνο από τις τιμές των εισόδων του συστήματος
- **Ακολουθιακά** κυκλώματα: Οι τιμές των εξόδων εξαρτώνται τόσο από τις τιμές των εισόδων του συστήματος όσο και από τις προηγούμενες τιμές των εξόδων (έχουμε ανάδραση).
 - **Σύγχρονα**: Η συμπεριφορά τους ορίζεται από τις τιμές των εξόδων σε διακριτές στιγμές του χρόνου. Υπάρχει σήμα συγχρονισμού (ρολόι/clock-CLK)
 - **Ασύγχρονα**: Οι τιμές των εξόδων αλλάζουν ανά πάσα στιγμή.

Ψηφιακά Συστήματα

Ψηφιακό κύκλωμα – Φυσική υλοποίηση

- Τεχνολογίες υλοποίησης
 - Application-specific ICs (ASICs): Ολοκληρωμένα κυκλώματα εξειδικευμένα για εφαρμογές (δεν προγραμματίζονται)
 - Field-programmable gate arrays (FPGAs): Επιτόπου προγραμματιζόμενοι πίνακες πυλών
- Αντιστοίχιση(mapping): καθορίζει τους πόρους για κάθε υποσύστημα
- Τοποθέτηση (placement): διευθετεί τις πύλες μέσα στα υποσυστήματα
- Δρομολόγηση (routing): ενώνει τις πύλες με αγωγούς
- Φυσική επαλήθευση (physical verification)
 - Το φυσικό κύκλωμα συνεχίζει να ικανοποιεί τους περιορισμούς
 - Χρησιμοποιεί καλύτερες εκτιμήσεις των καθυστερήσεων

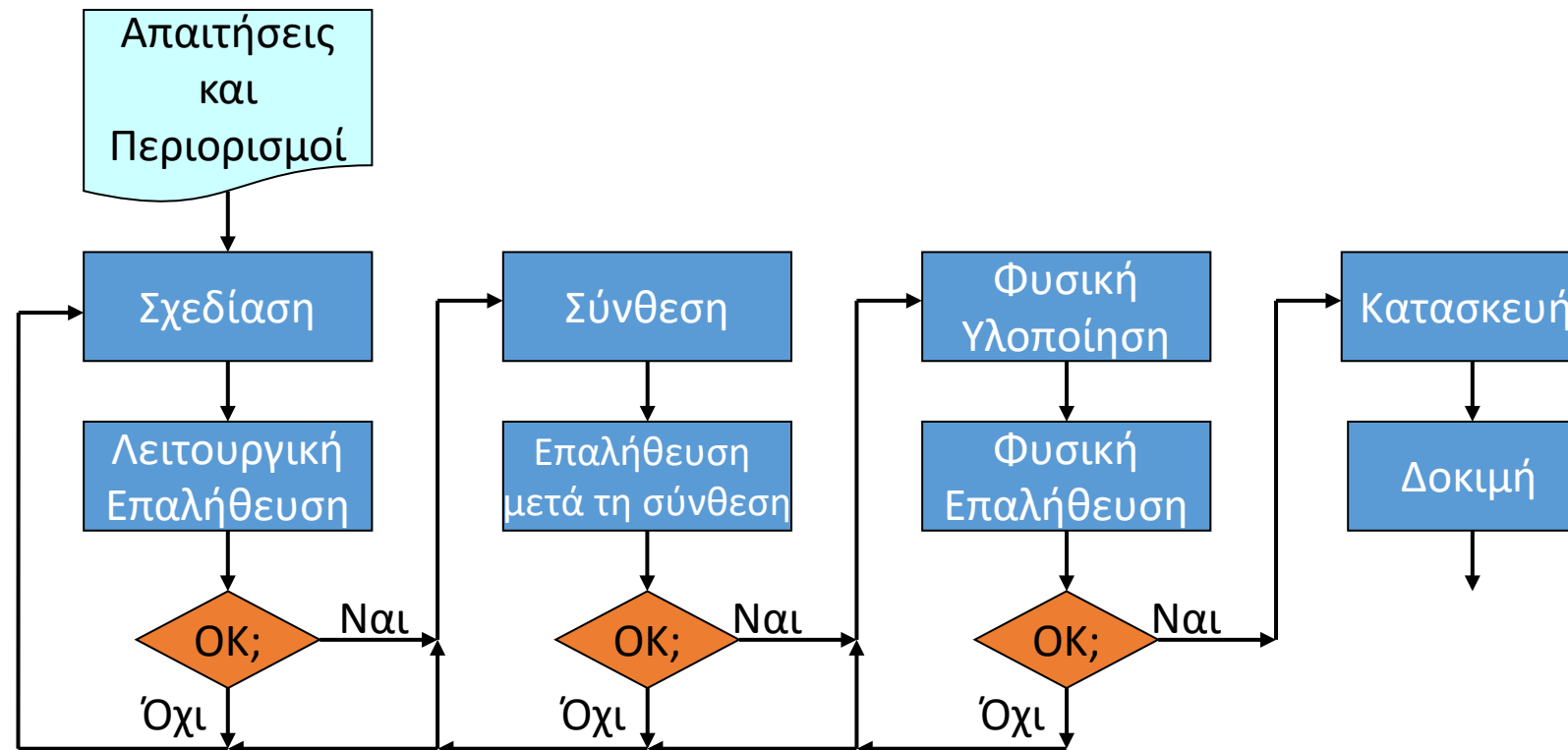
Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση – Μεθοδολογία Σχεδίασης

- Τα απλά συστήματα μπορούν να σχεδιαστούν από ένα άτομο χρησιμοποιώντας ειδικές μεθόδους
- Τα πραγματικά συστήματα σχεδιάζονται από ομάδες
 - Απαιτούν μια συστηματική μεθοδολογία σχεδίασης
- Καθορίζει
 - Τις εργασίες που αναλαμβάνουμε
 - Την πληροφορία που απαιτείται και παράγεται
 - Τις σχέσεις μεταξύ των εργασιών
 - εξαρτήσεις, καθορισμός των ακολουθιών
 - Τα εργαλεία EDA (Electronic design automation) που χρησιμοποιούνται (αλλιώς και eCAD – electronic Computer Aided Design)

Ψηφιακά Συστήματα

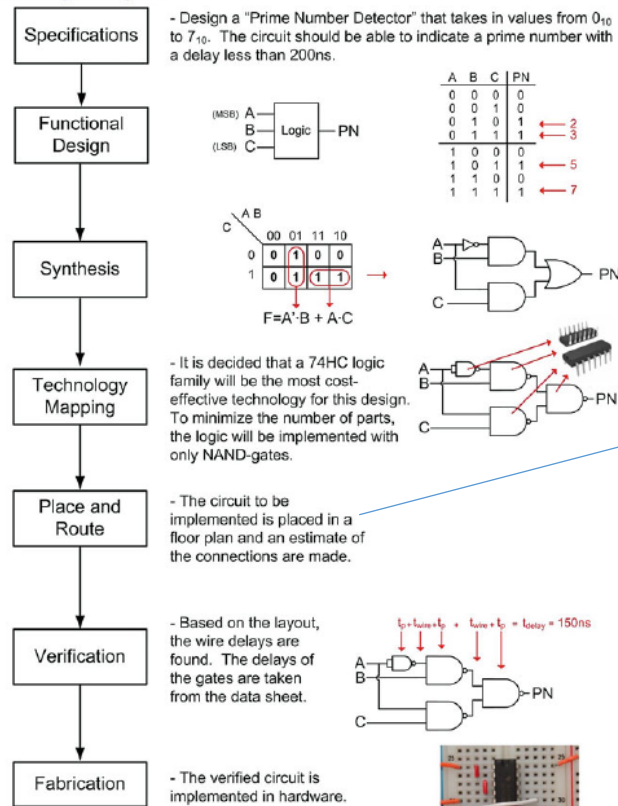
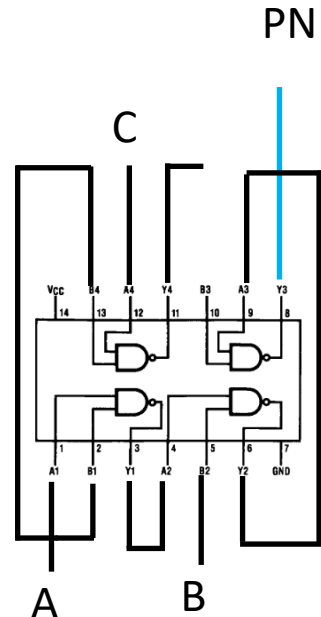
Ψηφιακή Σχεδίαση – Μεθοδολογία ανάπτυξης



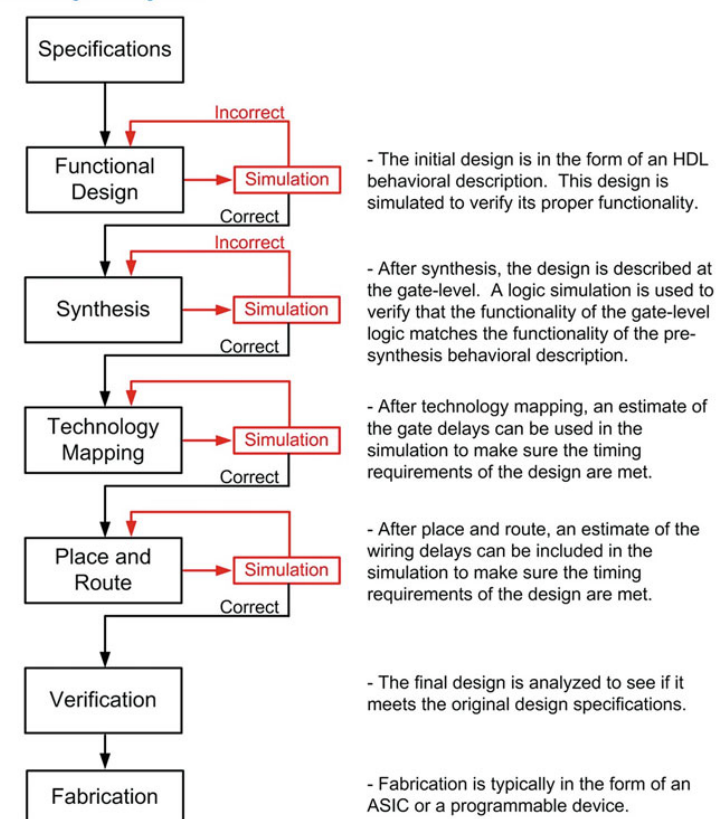
Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση – Μεθοδολογία ανάπτυξης

Classical Digital Design Flow



Modern Digital Design Flow



<https://www.futurlec.com/Datasheet/74HC/74HC00.pdf>

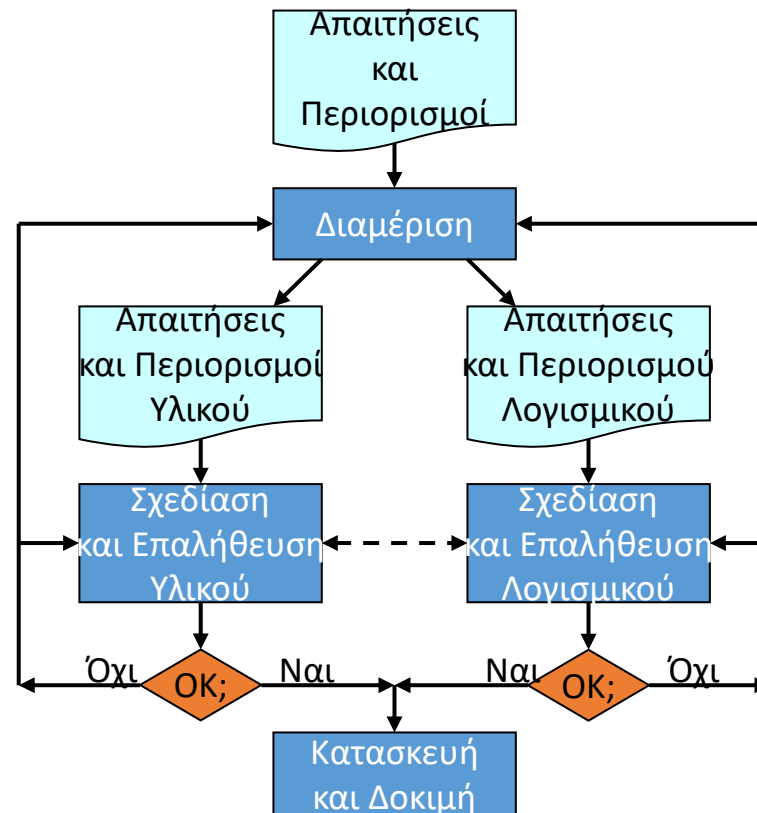
Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση – Ιεραρχική σχεδίαση

- Τα κυκλώματα είναι αρκετά πολύπλοκα για να σχεδιάσουμε όλες τις λεπτομέρειες με τη μία
- Σχεδιάζουμε υποσυστήματα για απλές λειτουργίες
- Συνθέτουμε το σύστημα από τα υποσυστήματα
 - Αντιμετωπίζουμε τα υποκυκλώματα ως «μαύρα κουτιά»
 - Επαληθεύουμε ανεξάρτητα, και έπειτα επαληθεύουμε τη σύνθεση
- Σχεδίαση top-down (από πάνω προς τα κάτω) ή bottom-up (από κάτω προς τα πάνω)

Ψηφιακά Συστήματα

Ψηφιακή Σχεδίαση – Μεθοδολογία Συσχεδίασης



Κάθε ομάδα υλοποιεί ένα ξεχωριστό Module του συστήματος

Ψηφιακά Συστήματα

Δυαδική Αναπαράσταση

- Δεκαδικό σύστημα αναπαράστασης αριθμών (0-9)

Στήλη των 1
Στήλη των 10
Στήλη των 100
Στήλη των 1000
6598₁₀

Κάθε στήλη ενός δεκαδικού αριθμού έχει δεκαπλάσιο βάρος από την προηγούμενη στήλη. Ξεκινώντας από τα δεξιά προς τα αριστερά τα βάρη είναι: $10^0, 10^1, 10^2, 10^3, \dots$

- Δυαδικό σύστημα αναπαράστασης αριθμών (0-1)

Στήλη των 1
Στήλη των 2
Στήλη των 4
Στήλη των 8
1101₂

Κάθε στήλη ενός δυαδικού αριθμού έχει διπλάσιο βάρος από την προηγούμενη στήλη. Ξεκινώντας από τα δεξιά προς τα αριστερά τα βάρη είναι: $2^0, 2^1, 2^2, 2^3, \dots$

Ψηφιακά Συστήματα

Δυαδική Αναπαράσταση

- Δεκαδικό σύστημα αναπαράστασης αριθμών (0-9)

Στήλη των 1
Στήλη των 10
Στήλη των 100
Στήλη των 1000

$$6598_{10} = 6 \times 10^3 + 5 \times 10^2 + 9 \times 10^1 + 8 \times 10^0$$

Έξι Πέντε Εννέα Οκτώ
Χιλιάδες Εκατοντάδες Δεκάδες Μονάδες

- Δυαδικό σύστημα αναπαράστασης αριθμών (0-1)

Στήλη των 1
Στήλη των 2
Στήλη των 4
Στήλη των 8

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$$

Μια Μια Μηδέν Μια
Οκτάδα Τετράδα Δυάδες Μονάδα

Δυαδικό Σύστημα

Αριθμοί χωρίς πρόσημο (Unsigned – Μόνο θετικοί)

Πρόσθεση

0	0	1	1
+0	+1	+0	+1
---	---	---	---
0	1	1	10
Κρατούμενο/(C)arry)			
0	0	0	1

00	01	11	11
01	+01	+01	+11
----	----	-----	-----
01	10	100	110
Κρατούμενο/(C)arry)			
0	0	1	1

Δυαδικό Σύστημα

Αριθμοί χωρίς πρόσημο (Unsigned – Μόνο θετικοί)

Πολλαπλασιασμός/Διαίρεση με πολλαπλάσια του δύο

$$10 = 2(\text{δεκαδικό}) \quad (A)$$

Εφαρμόζουμε αριστερή ολίσθηση και γεμίζουμε δεξιά με το 0

$$10\mathbf{0} = 4(\text{δεκαδικό}) \quad (B) = 2 * (A)$$

$$10\mathbf{00} = 8(\text{δεκαδικό}) \quad (C) = 2 * (B) = 4 * (A)$$

$$1000 = 8(\text{δεκαδικό}) \quad (A)$$

Εφαρμόζουμε δεξιά ολίσθηση διαγράφοντας το πιο δεξί ψηφίο

$$100 = 4(\text{δεκαδικό}) \quad (B) = (A)/2$$

$$10 = 2(\text{δεκαδικό}) \quad (C) = (B)/2 = (A)/4$$

Ψηφιακά Συστήματα

Δυαδική Αναπαράσταση

- Συστήματα που αναπαριστούν την πληροφορία με δύο τιμές (0,1 ή True,False)
- Βασικές ψηφιακές λογικές πύλες και πίνακες αληθείας



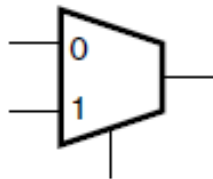
AND gate



OR gate



inverter



multiplexer

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

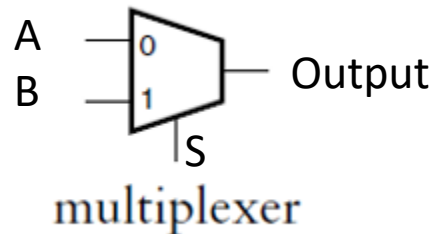
x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	$\bar{x}$
0	1
1	0

Ψηφιακά Συστήματα

Δυαδική Αναπαράσταση

- Σε κάθε Πύλη ή Κύκλωμα αντιστοιχεί ένας Πίνακας Αληθείας (Truth Table).
- Ο Πίνακας Αληθείας καθορίζει την τιμή εξόδου για κάθε συνδυασμό εισόδων στην Πύλη ή το Κύκλωμα

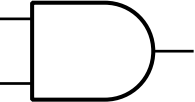
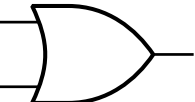
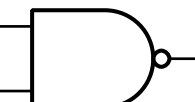
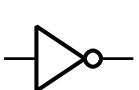


S	Output
0	A
1	B

Αν $A=0$, $B=1$, $S=1$, Output= ;

Αν $A=1$, $B=0$, $S=1$, Output= ;

Λογικοί Τελεστές

<code>a and b</code>	$a \cdot b$	
<code>a or b</code>	$a + b$	
<code>a nand b</code>	$\overline{a \cdot b}$	
<code>a nor b</code>	$\overline{a + b}$	
<code>a xor b</code>	$a \oplus b$	
<code>a xnor b</code>	$\overline{a \oplus b}$	
<code>not a</code>	$\overline{a}$	

- Προτεραιότητα
 - το **not** έχει την υψηλότερη
 - οι υπόλοιποι τελεστές έχουν ίση προτεραιότητα
 - από αριστερά προς τα δεξιά
 - χρησιμοποιούμε παρενθέσεις για να διακρίνουμε τη σειρά υπολογισμού
- Τιμές bit στην VHDL
 - '0' και '1'

- **Hardware Description Language (HDL)**
 - Μια γλώσσα για την μοντελοποίηση της συμπεριφοράς και της δομής των ψηφιακών συστημάτων
- **Electronic Design Automation (EDA) using HDL**
 - Σχεδίαση ηλεκτρονικών κυκλωμάτων με χρήση εργαλείων CAD (computer-aided design)
 - Εισαγωγή σχεδίασης (design entry)
 - Χρήση κώδικα αντί για σχηματικά διαγράμματα
 - Επαλήθευση (verification)
 - Προσομοίωση (simulation) του κώδικα
 - Σύνθεση (synthesis)
 - Αυτόματη παραγωγή των κυκλωμάτων
 - Φυσική υλοποίηση (implementation)
 - Υλοποίηση του κυκλώματος στην τεχνολογία επιλογής

- **Hardware Description Language (HDL)**
 - Μια γλώσσα για την μοντελοποίηση της συμπεριφοράς και της δομής των ψηφιακών συστημάτων
- **Electronic Design Automation (EDA) using HDL**
 - Σχεδίαση ηλεκτρονικών κυκλωμάτων με χρήση εργαλείων CAD (computer-aided design)
 - Εισαγωγή σχεδίασης (design entry)
 - Χρήση κώδικα αντί για σχηματικά διαγράμματα
 - Επαλήθευση (verification)
 - Προσομοίωση (simulation) του κώδικα
 - Σύνθεση (synthesis)
 - Αυτόματη παραγωγή των κυκλωμάτων
 - Φυσική υλοποίηση (implementation)
 - Υλοποίηση του κυκλώματος στην τεχνολογία επιλογής

Πλεονεκτήματα των HDLs

- Υπερτερούν από τα σχηματικά διαγράμματα
 - Η μοντελοποίηση του συστήματος μπορεί να γίνει σε όλα τα επίπεδα (από τα υψηλότερα ως τα χαμηλότερα)
 - Η περιγραφή σε HDL είναι συνήθως πιο κατανοητή από ένα σχηματικό διάγραμμα
 - Η περιγραφή σε HDL είναι ανεξάρτητη από τις βιβλιοθήκες σχεδίασης (design libraries) και τα εργαλεία CAD
- Υπερτερούν από τις γλώσσες προγραμματισμού
 - Παρέχουν δομές που περιγράφουν καλύτερα το υλικό
 - Παράλληλη εκτέλεση εντολών αντί για ακολουθιακή/σειριακή
 - Παρέχουν δυνατότητα για περιγραφή χρονισμών

Μοντελοποίηση και προσομοίωση

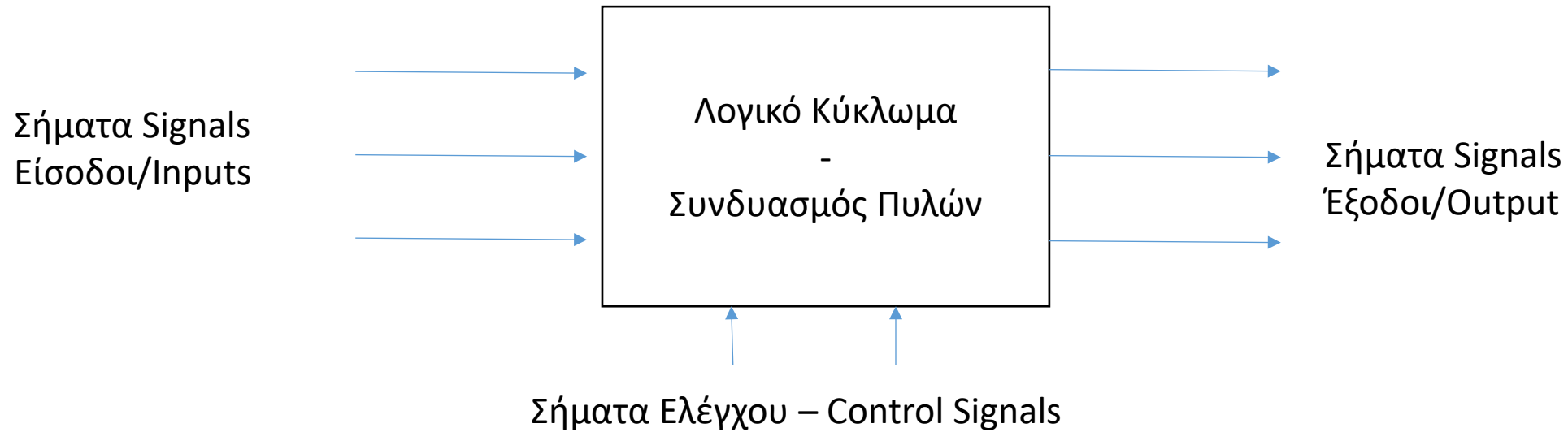
- Αρχικά οι HDLs σχεδιάστηκαν για τη μοντελοποίηση και τη προσομοίωση των συστημάτων υλικού στα υψηλότερα επίπεδα αφαίρεσης
- Χαρακτηριστικά μοντελοποίησης των HDLs:
 - παράλληλη εκτέλεση
 - ιεραρχική σχεδίαση
 - περιγραφή χρονισμών
 - περιγραφή ακολουθίας γεγονότων
 - περιγραφή σύγχρονης/ασύγχρονης συμπεριφοράς

VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Σύνθεση

- Συνήθως σχεδιάζουμε χρησιμοποιώντας HDL επιπέδου μεταφοράς καταχωρητή (Register Transfer Level - RTL)
 - υψηλότερο επίπεδο αφάιρεσης από τις πύλες
- Το εργαλείο σύνθεσης μεταφράζει το μοντέλο σε ένα κύκλωμα από πύλες που εκτελεί την ίδια λειτουργία
- Προσδιορίζουμε στο εργαλείο
 - την τεχνολογία υλοποίησης που στοχεύουμε
 - περιορισμούς στο χρονισμό, στην επιφάνεια, κλπ.
- Επαλήθευση μετά τη σύνθεση
 - το κύκλωμα που προέκυψε από τη σύνθεση ικανοποιεί τους περιορισμούς

Ψηφιακά Συστήματα

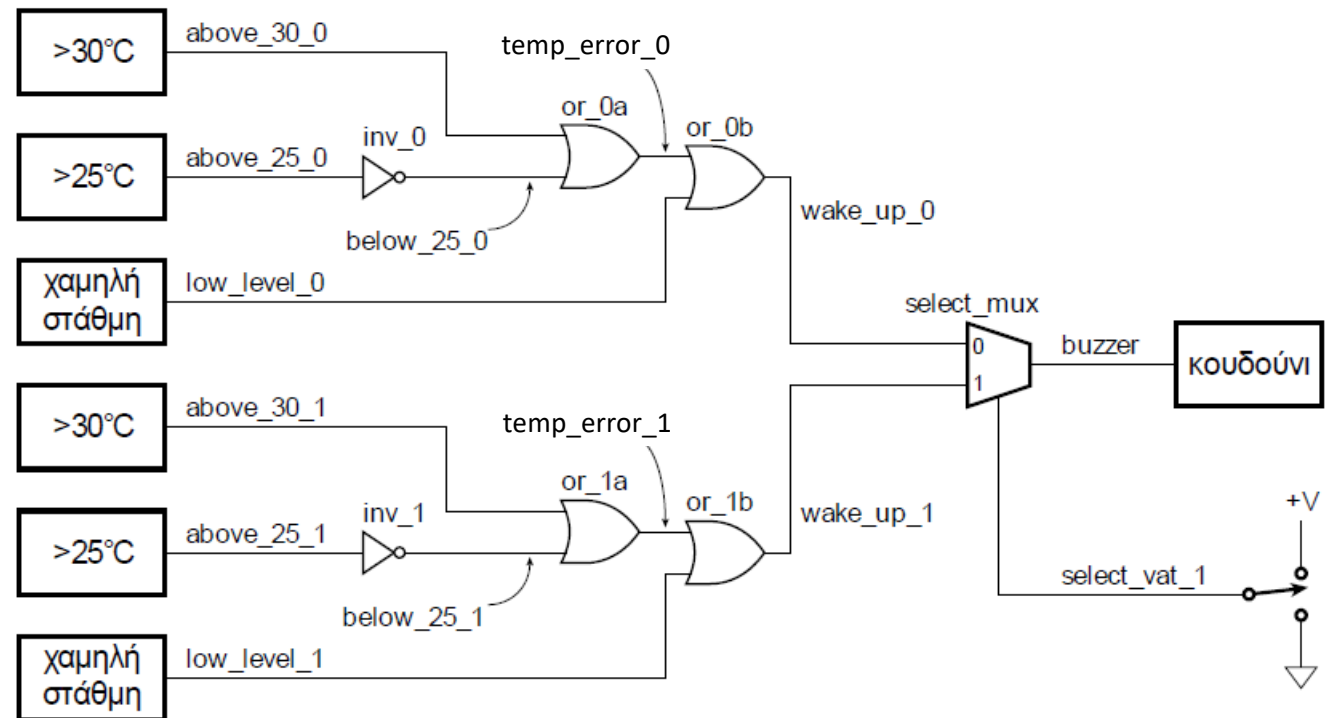


Ψηφιακά Συστήματα

Ψηφιακό κύκλωμα

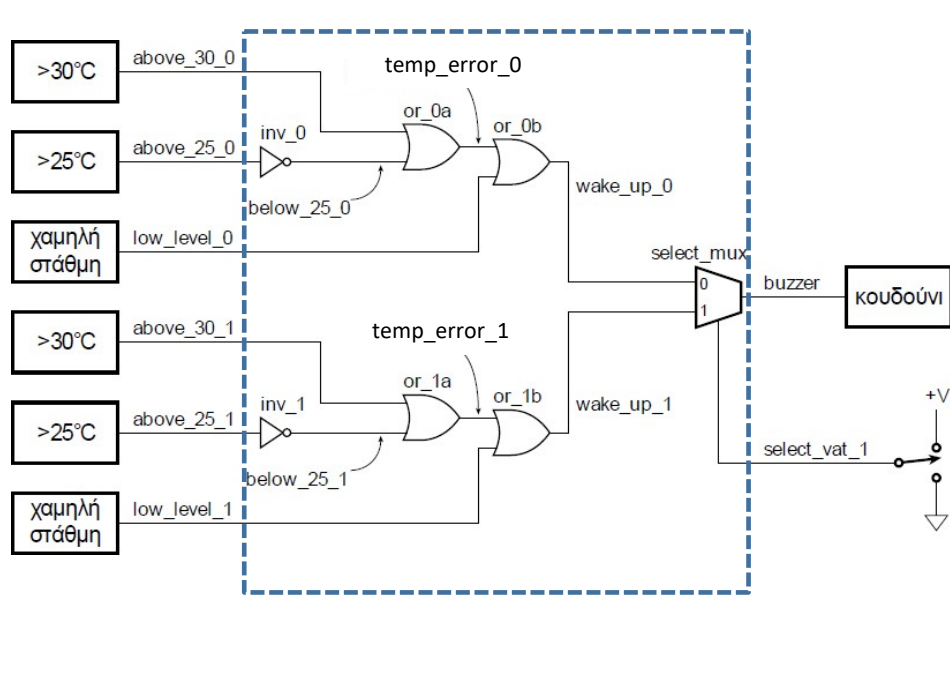
Παράδειγμα:

Εργοστάσιο με 2 δοχεία επεξεργασίας υγρών. Το κάθε δοχείο πρέπει α) να έχει θερμοκρασία μεταξύ 25 και 30 βαθμών και β) η στάθμη του πρέπει να είναι πάνω από ένα επίπεδο. Ο επόπτης επιλέγει ποιο από τα 2 δοχεία χρησιμοποιείται. Σε περίπτωση που το α) ή το β) δεν ικανοποιούνται πρέπει να ενεργοποιηθεί το κουδούνι.



Ψηφιακά Συστήματα

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Entity: Input/Output



Entity name

Port list

```
library ieee; use ieee.std_logic_1164.all;
entity vat_buzzer is
    port ( above_25_0, above_30_0,
           low_level_0   : in std_logic;
           above_25_1, above_30_1,
           low_level_1   : in std_logic;
           select_vat_1  : in std_logic;
           buzzer        : out std_logic );
end entity vat_buzzer;
```

Χρήση πρότυπης βιβλιοθήκης

package

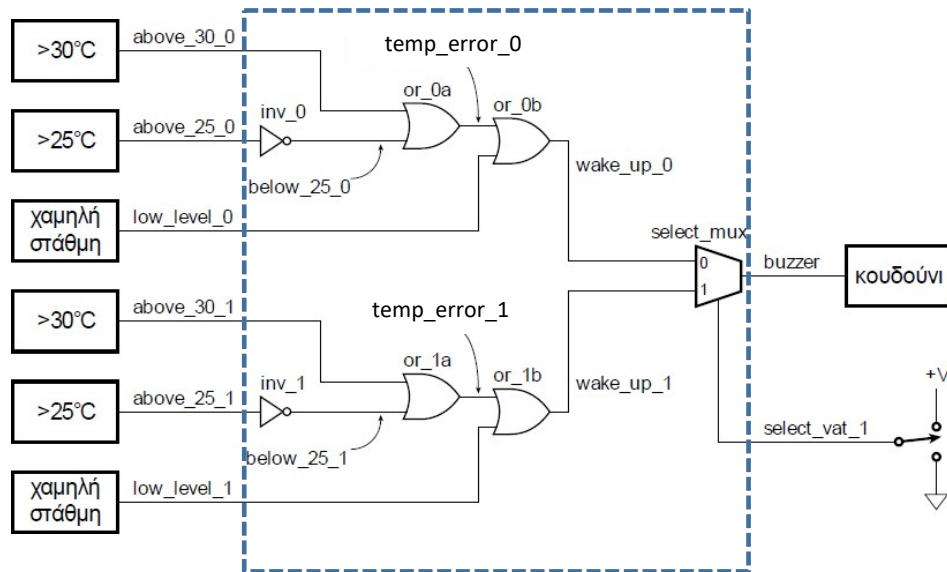
Port type

Port name

Port mode

Ψηφιακά Συστήματα

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Structural Architecture



architecture struct of vat_buzzer is

Περιγραφή δομικών στοιχείων

```
signal below_25_0, temp_error_0, wake_up_0 : std_logic;  
signal below_25_1, temp_error_1, wake_up_1 : std_logic;
```

begin

```
-- components for vat 0
```

```
inv_0 : inv port map (above_25_0, below_25_0);
```

```
or_0a : or2 port map (above_30_0, below_25_0, temp_error_0);
```

```
or_0b : or2 port map (temp_error_0, low_level_0, wake_up_0);
```

```
-- components for vat 1
```

```
inv_1 : inv port map (above_25_1, below_25_1);
```

```
or_1a : or2 port map (above_30_1, below_25_1, temp_error_1);
```

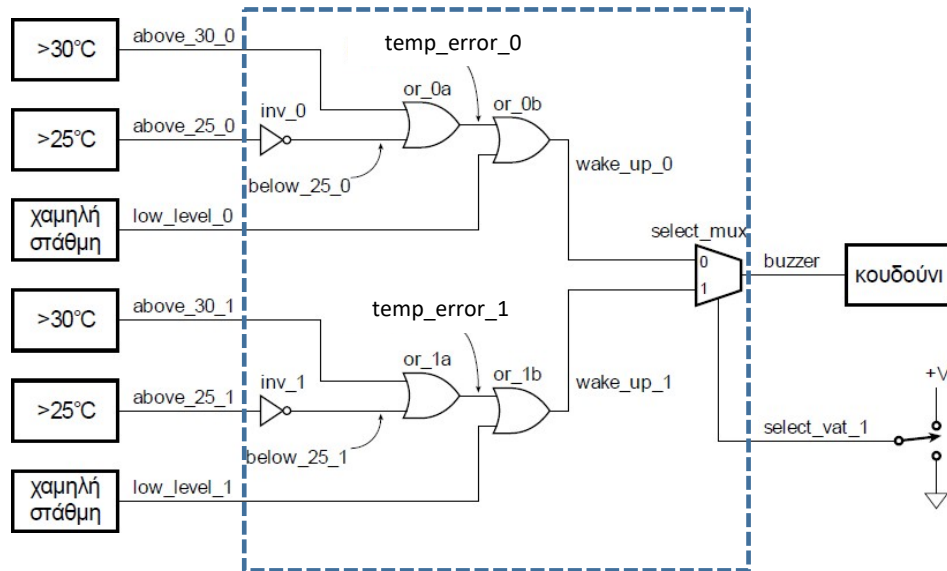
```
or_1b : or2 port map (temp_error_1, low_level_1, wake_up_1);
```

```
select_mux : mux2 port map ( wake_up_0, wake_up_1, select_vat, buzzer);
```

end architecture struct;

Ψηφιακά Συστήματα

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Dataflow Architecture



architecture Dataflow of vat_buzzer is

begin

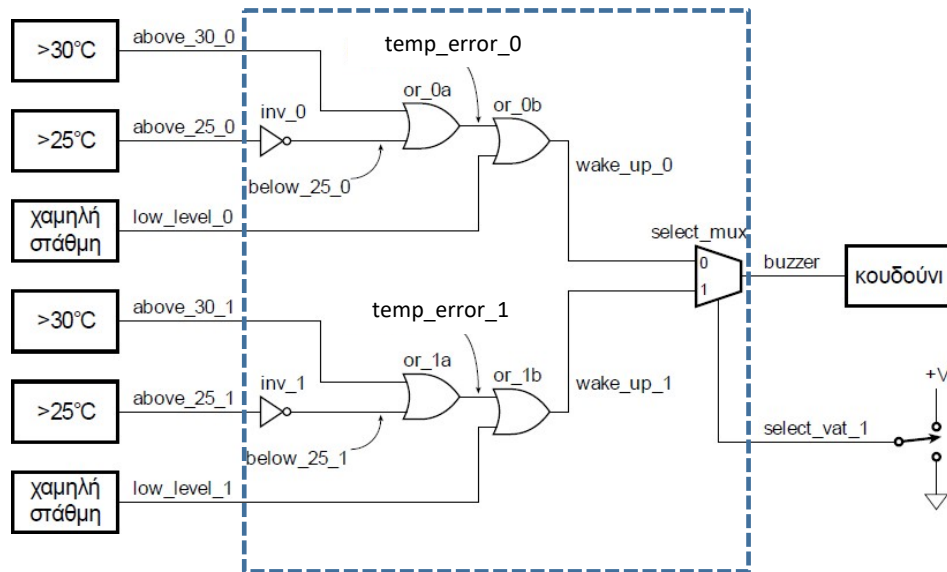
```
buzzer <= low_level_1 or (above_30_1 or not above_25_1)
when select_vat='1' else
low_level_0 or (above_30_0 or not above_25_0);
```

end architecture Dataflow;

Αρχιτεκτονική
Περιγραφή της λειτουργικότητας
που προσφέρει το λογικό
κύκλωμα

Ψηφιακά Συστήματα

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Behavioral Architecture



architecture behavioral of vat_buzzer is

begin

process (low_level_0, above_30_0, above_25_0,
low_level_1, above_30_1, above_25_1
select_vat) is

begin

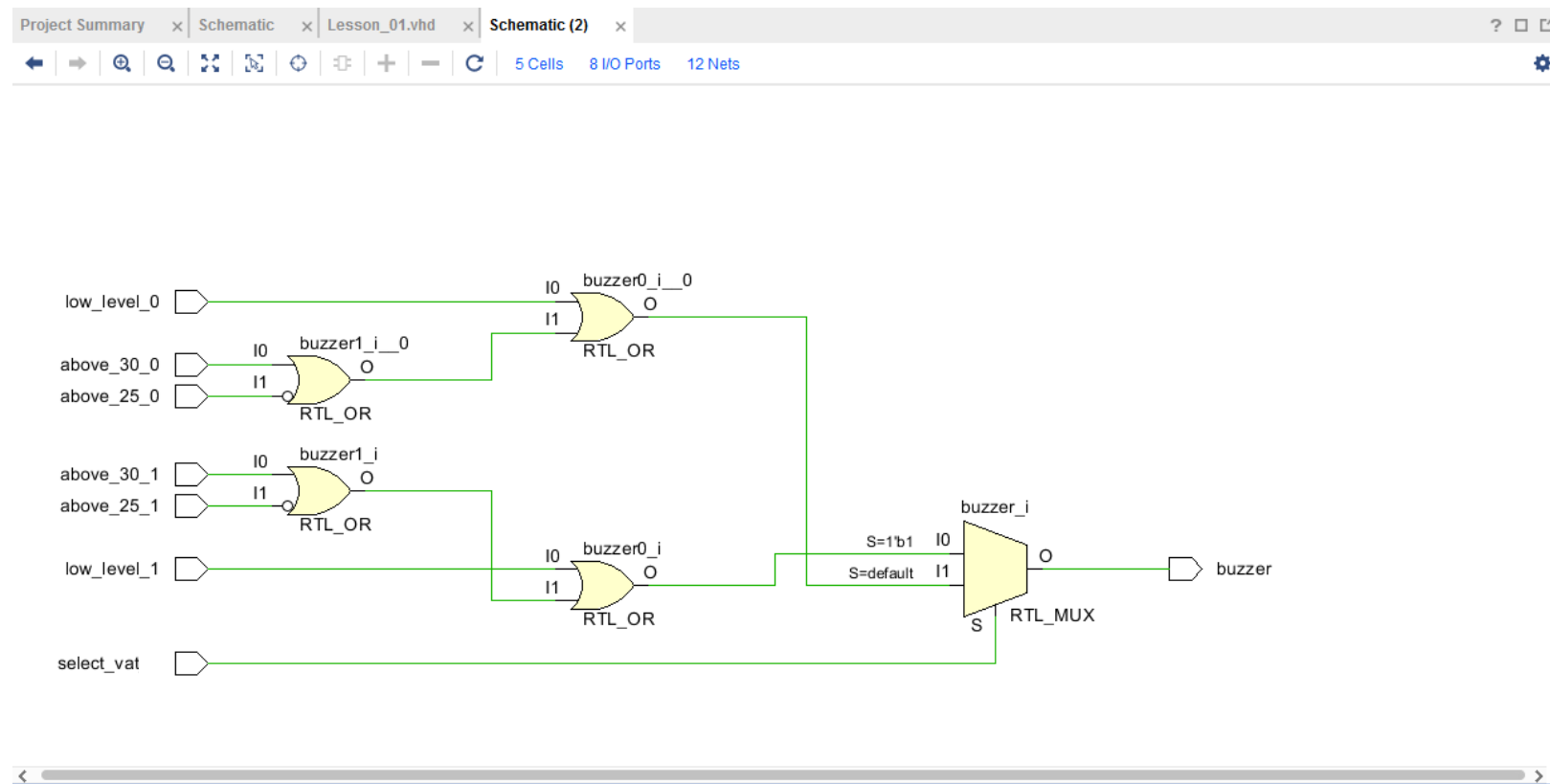
buzzer <= low_level_1 or (above_30_1 or not above_25_1)
when select_vat='1' else
low_level_0 or (above_30_0 or not above_25_0);

end process;

end architecture behavioral;

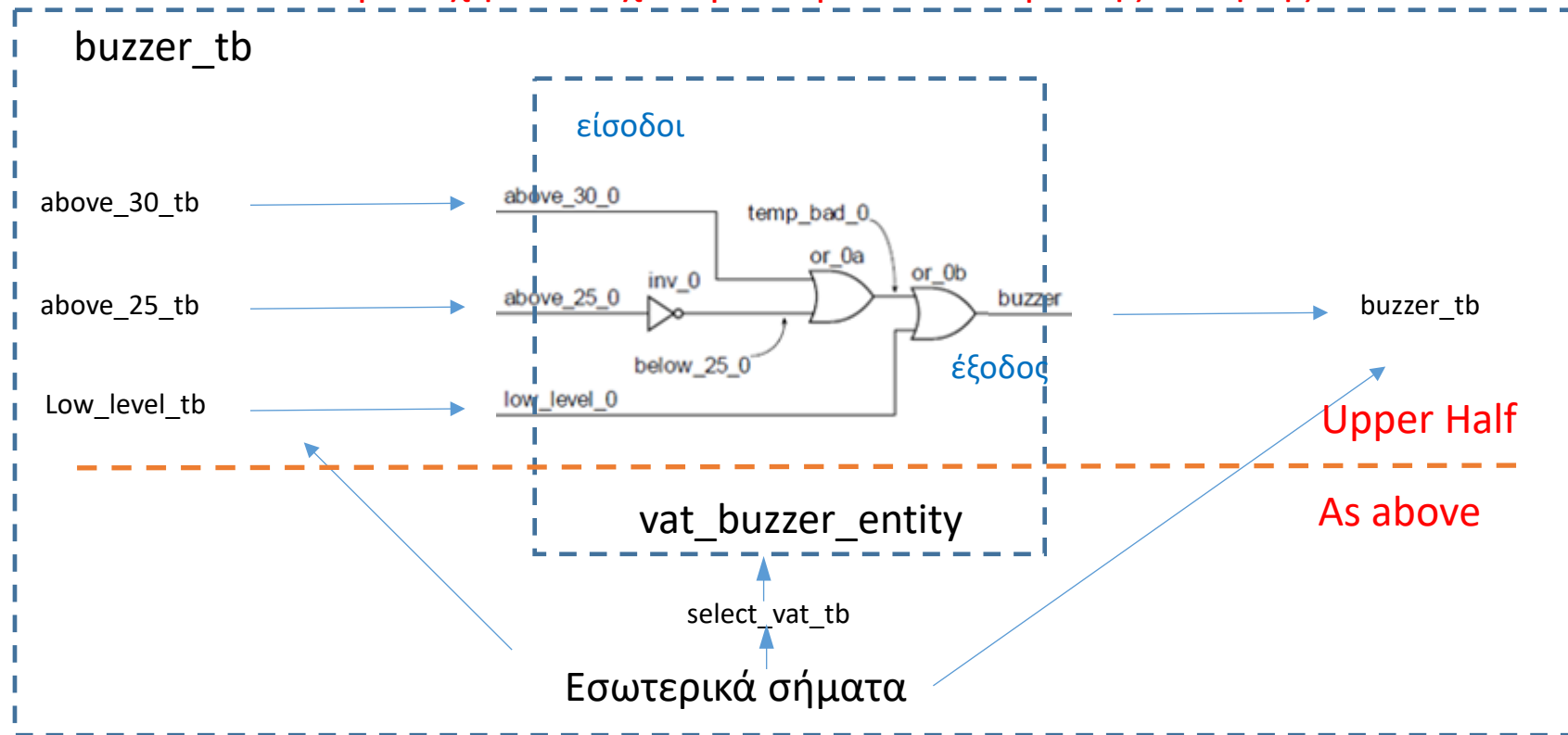
VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Λογικές πύλες



Ψηφιακό κύκλωμα – VHDL: Προσομοίωση

! Προσοχή ότι δείχνουμε το μισό κύκλωμα της άσκησης



Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Προσομοίωση

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;
```

```
entity buzzer_tb is  
end entity buzzer_tb;
```

```
architecture Behavioral of buzzer_tb is
```

```
component vat_buzzer is
```

```
port (  
  above_25_0, above_30_0, low_level_0      : in std_logic;  
  above_25_1, above_30_1, low_level_1      : in std_logic;  
  select_vat                               : in std_logic;  
  buzzer                                    : out std_logic;  
);  
end component vat_buzzer  
signal above_25_0_tb, above_30_0_tb, low_level_0_tb :std_logic;  
signal above_25_1_tb, above_30_1_tb, low_level_1_tb :std_logic;  
signal select_vat_tb                               :std_logic;  
signal buzzer_tb                                    : std_logic;
```

```
begin
```

```
duv: vat_buzzer port map (above_25_0 => above_25_0_tb, above_25_1 =>  
  above_25_1_tb, above_30_0 => above_30_0_tb, above_30_1 => above_30_1_tb,  
  low_level_0 => low_level_0_tb, low_level_1 => low_level_1_tb, select_vat =>  
  select_vat_tb, buzzer => buzzer_tb);
```

```
apply_test_cases: process is
```

```
begin
```

```
  above_25_0_tb <='0'; above_25_1_tb <='0'; above_30_0_tb <='0'; above_30_1_tb  
  <='0'; low_level_0_tb <='0'; low_level_1_tb <='0'; select_vat_tb <='0'; wait for 20 ns;  
  above_25_0_tb <='1'; above_25_1_tb <='0'; above_30_0_tb <='1'; above_30_1_tb  
  <='0'; low_level_0_tb <='0'; low_level_1_tb <='0'; select_vat_tb <='0'; wait for 20 ns;  
  above_25_0_tb <='0'; above_25_1_tb <='0'; above_30_0_tb <='0'; above_30_1_tb  
  <='0'; low_level_0_tb <='1'; low_level_1_tb <='1'; select_vat_tb <='0'; wait for 20 ns;  
  above_25_0_tb <='0'; above_25_1_tb <='0'; above_30_0_tb <='0'; above_30_1_tb  
  <='0'; low_level_0_tb <='0'; low_level_1_tb <='1'; select_vat_tb <='1'; wait for 20 ns;  
  above_25_0_tb <='0'; above_25_1_tb <='0'; above_30_0_tb <='1'; above_30_1_tb  
  <='0'; low_level_0_tb <='0'; low_level_1_tb <='0'; select_vat_tb <='1'; wait for 20 ns;  
  above_25_0_tb <='0'; above_25_1_tb <='1'; above_30_0_tb <='0'; above_30_1_tb  
  <='0'; low_level_0_tb <='1'; low_level_1_tb <='0'; select_vat_tb <='0'; wait for 20 ns;  
  above_25_0_tb <='0'; above_25_1_tb <='1'; above_30_0_tb <='1'; above_30_1_tb  
  <='1'; low_level_0_tb <='1'; low_level_1_tb <='0'; select_vat_tb <='0'; wait for 20 ns;  
  above_25_0_tb <='0'; above_25_1_tb <='0'; above_30_0_tb <='0'; above_30_1_tb  
  <='1'; low_level_0_tb <='1'; low_level_1_tb <='1'; select_vat_tb <='0'; wait for 20 ns;
```

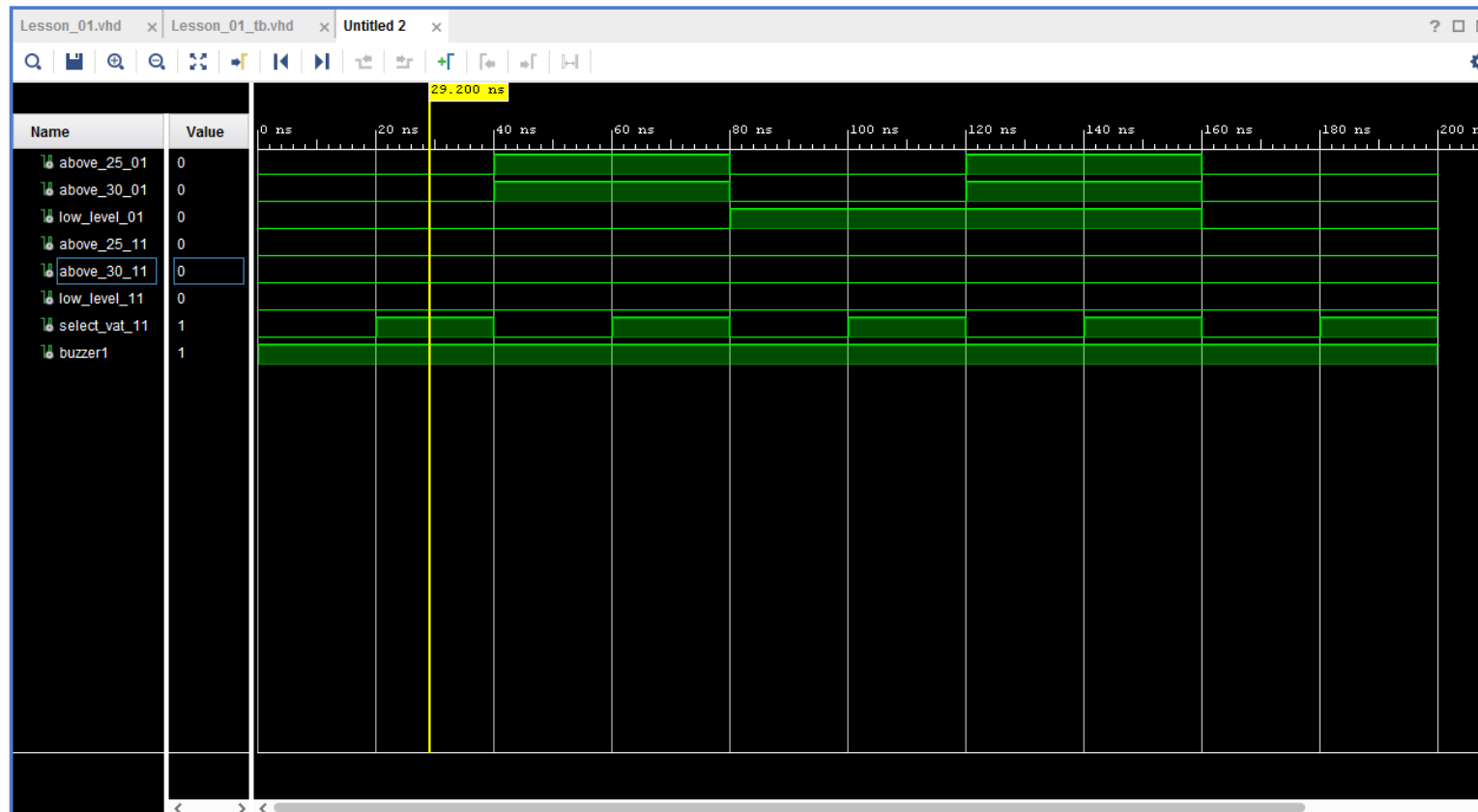
```
end process apply_test_cases;
```

```
end architecture Behavioral;
```

Συνήθως ενδεικτικές τιμές. Όχι όλος ο πίνακας αληθείας. Υπάρχει και πιο δόκιμος τρόπος

VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Προσομοίωση Χρονική



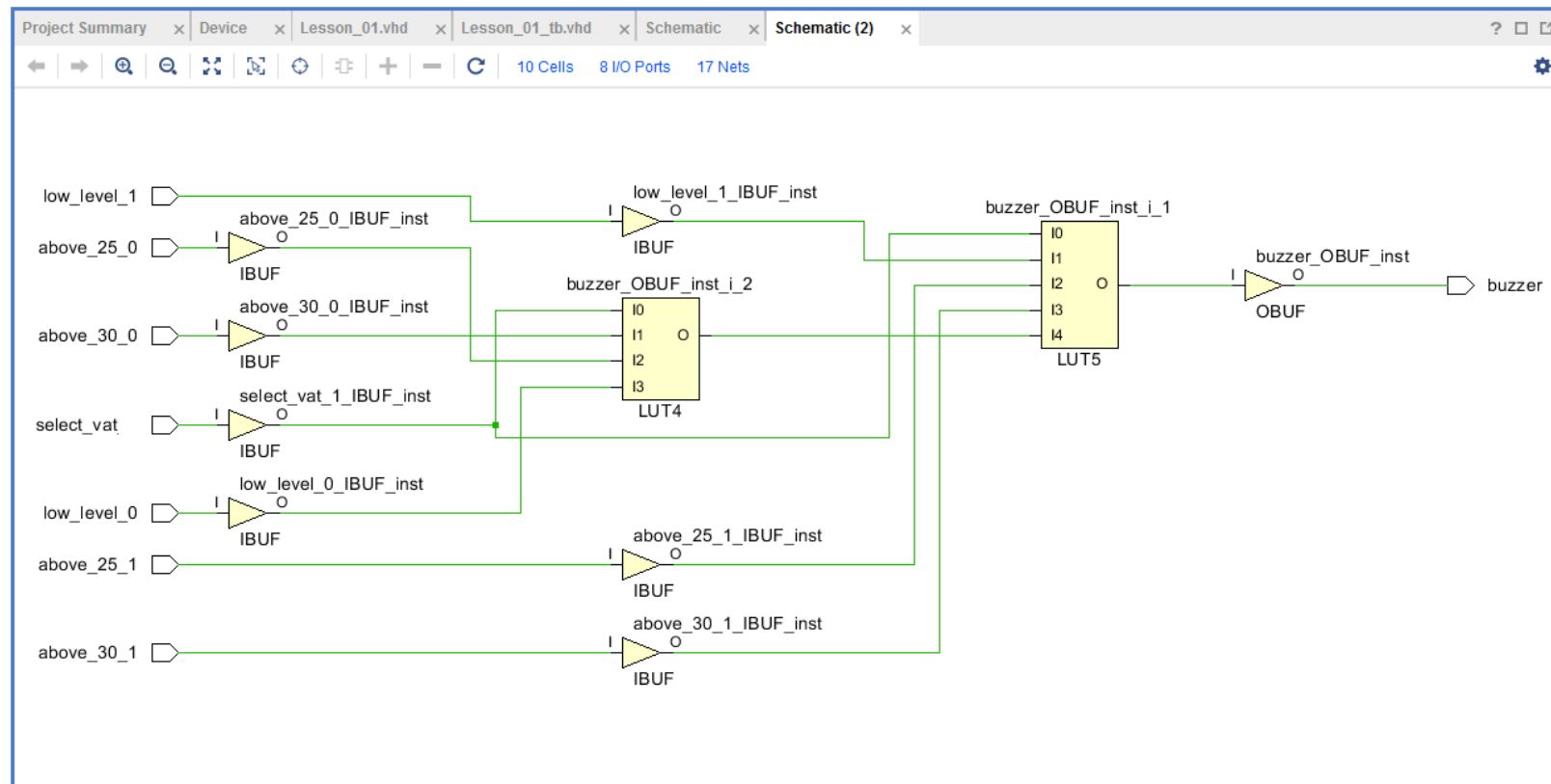
VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Σύνθεση

- Το εργαλείο σύνθεσης μεταφράζει το μοντέλο σε ένα κύκλωμα από ψηφιακά στοιχεία (πύλες, πολυπλέκτες, καταχωρητές, LUT) που εκτελεί την ίδια λειτουργία
- Προσδιορίζουμε στο εργαλείο
 - την τεχνολογία υλοποίησης που στοχεύουμε
 - περιορισμούς στο χρονισμό, στην επιφάνεια, κλπ.
- Επαλήθευση μετά τη σύνθεση
 - το κύκλωμα που προέκυψε από τη σύνθεση ικανοποιεί τους περιορισμούς

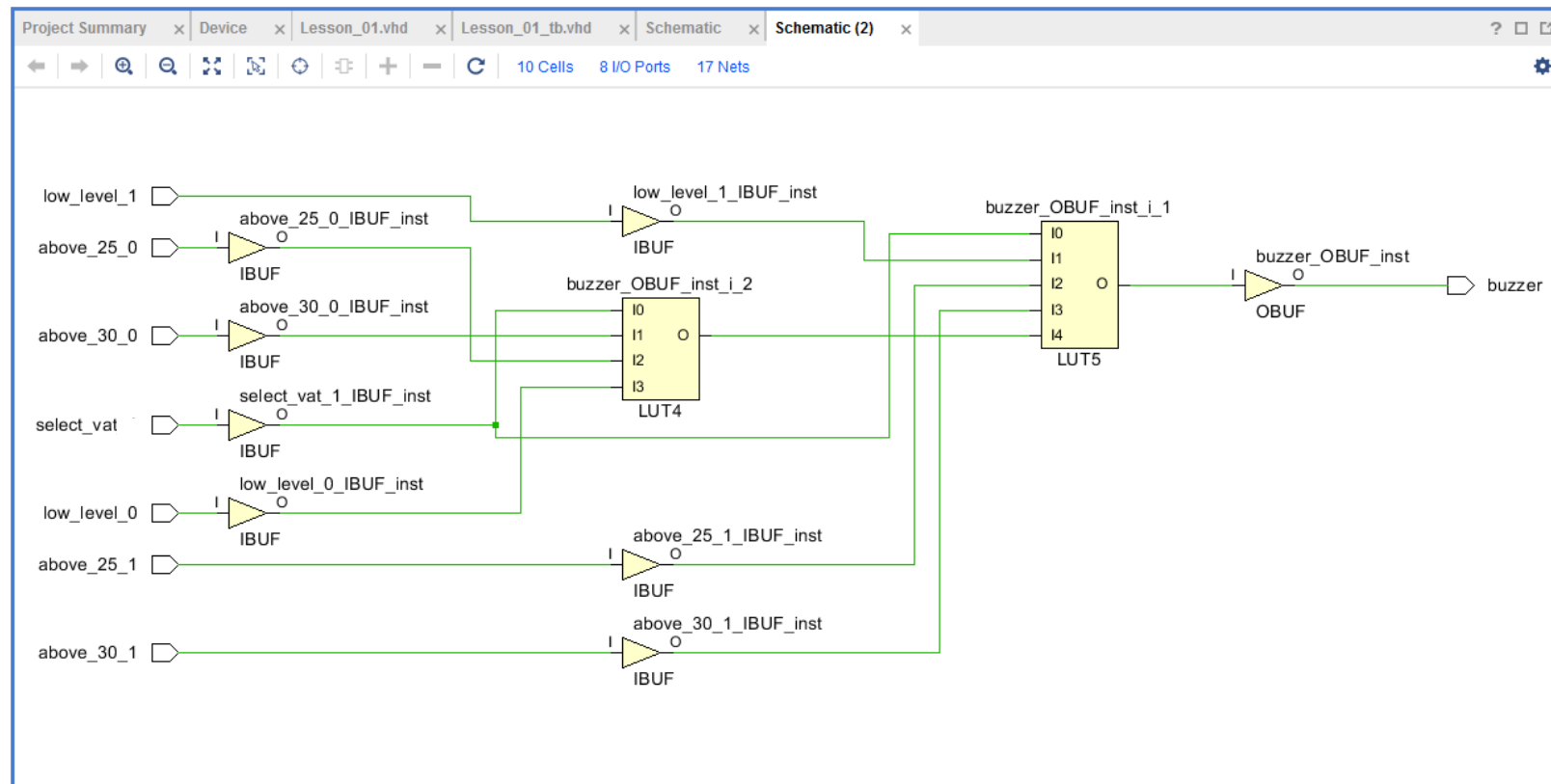
VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Σύνθεση



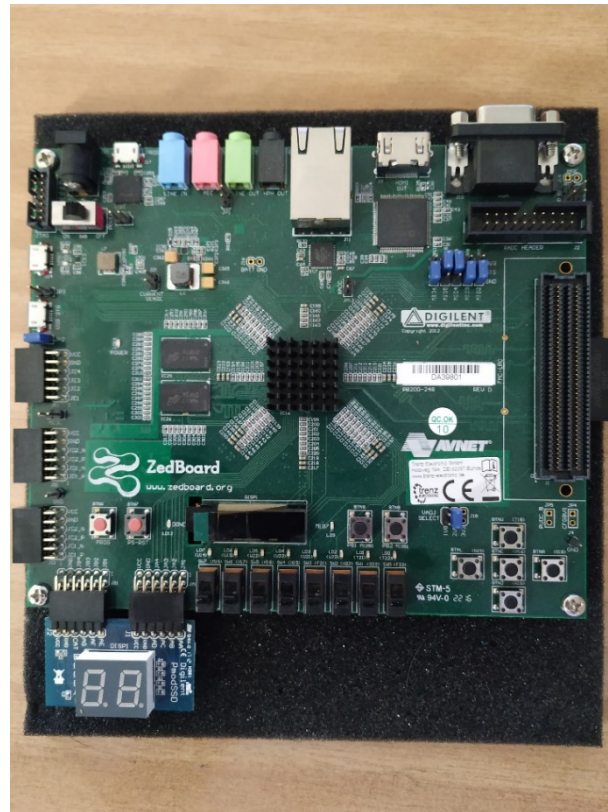
VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – Υλοποίηση



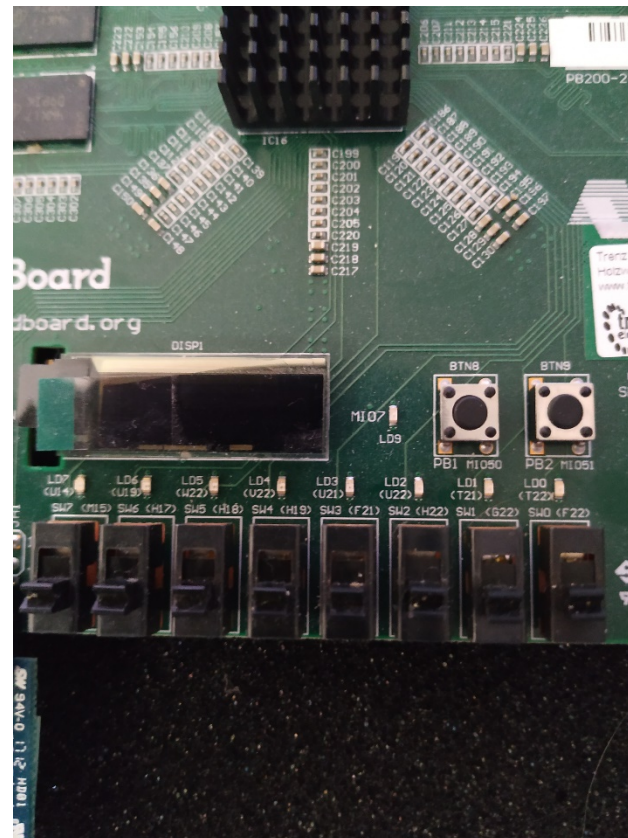
VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – FPGA



VHDL - VIVADO

Ψηφιακό κύκλωμα – Αναπαράσταση VHDL – FPGA



Παροχές ΕΚΠΑ

- <https://delos365.grnet.gr/>
 - Microsoft office (Word, Excel, Powerpoint,...) σε online και σε desktop έκδοση. Έχετε χώρο και στο onedrive (1TB). Σύνδεση απλά με τα ακαδημαϊκά σας credentials.
- google.com (ΠΡΟΣΟΧΗ: σύνδεση ως sdixxxxxxx@uoa.gr και μετά θα σας ζητηθούν τα ακαδημαϊκά σας credentials)
 - Εφαρμογές google + 50GB (google drive)
- <https://azureforeducation.microsoft.com/devtools>
 - Windows 10, 11 και εργαλεία ανάπτυξης λογισμικού της Microsoft (ΠΡΟΣΟΧΗ: σύνδεση ως sdixxxxxxx@o365.uoa.gr και μετά θα σας ζητηθούν τα ακαδημαϊκά σας credentials)

Προσοχή! Δεν είναι βέβαιο ότι θα ισχύουν για όλα τα account και ότι είναι ενεργή η συνδρομή του ΕΚΠΑ για αυτές τις υπηρεσίες

Περίληψη

- Εισαγωγή στο «Εργαστήριο Σχεδίαση Ψηφιακών Συστημάτων - VHDL»
- Μεθοδολογία σχεδίασης
- Εισαγωγή στο δυαδικό σύστημα αρίθμησης και στις λογικές πύλες
- Εισαγωγική παρουσίαση VHDL και Vivado
- Διαβάσετε το κεφάλαιο 1 (εκτός του 1.3) και τις παραγράφους 2.1 και 3.1 από το βιβλίο του Ashenden.
- Διαβάστε τις παραγράφους 1.1, 1.2, 1.3, 1.4.1 – 1.4.5, 1.5, 4.1 από το βιβλίο των Harris.
- Διαβάστε τις παραγράφους 1.1, 1.2, 1.3, 1.5, 1.5.1, 1.5.2, 2.1, 2.2, 2.3, 2.4, 2.5, 2.5.2, 2.5.3, 2.9, 2.10, 2.10.1 από το βιβλίο των Brown - Vranesic
- VHDL Reference Guide:
https://peterfab.com/ref/vhdl/vhdl_renerta/source/vhd00000.htm

2ο Εργαστήριο

Σχεδίασης Ψηφιακών Συστημάτων

2^η Διάλεξη

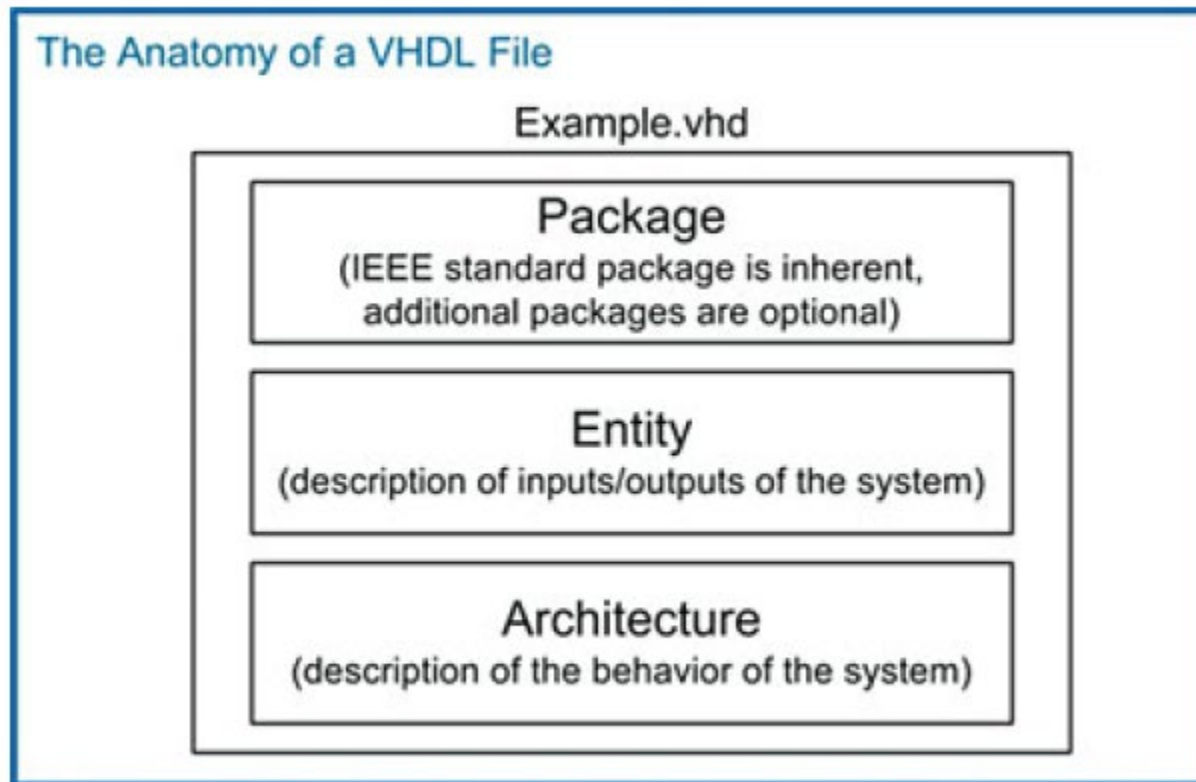
Ορισμός Οντότητας – Ορισμός/Είδη Αρχιτεκτονικής

Βασιλόπουλος Διονύσης

Ε.ΔΙ.Π. Τμήματος Πληροφορικής & Τηλεπικοινωνιών

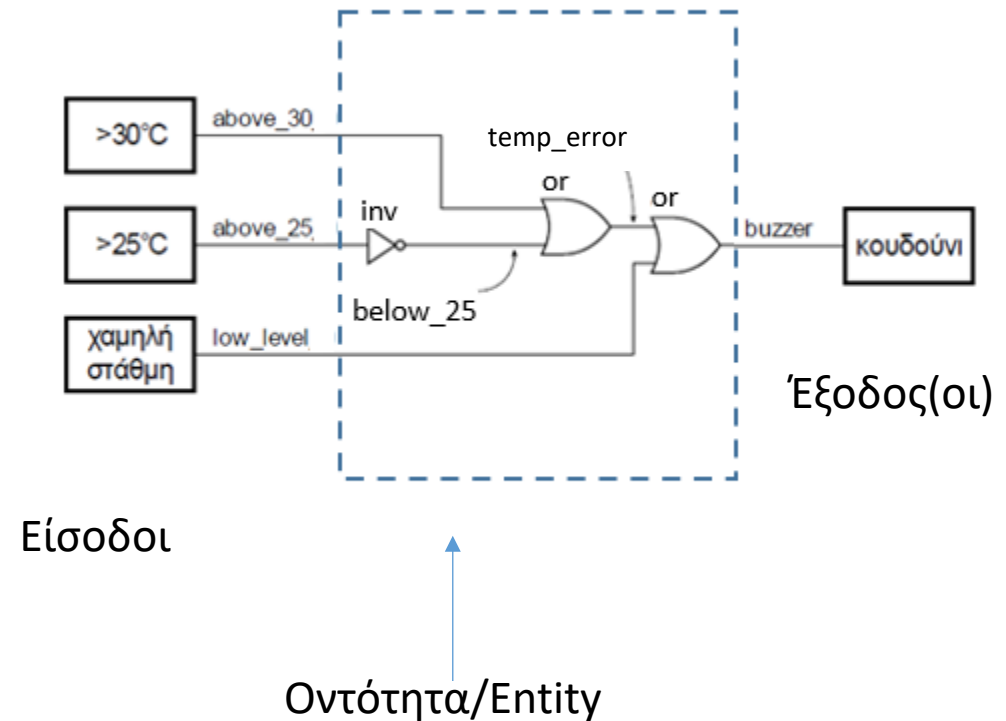
VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Δομή Αρχείου

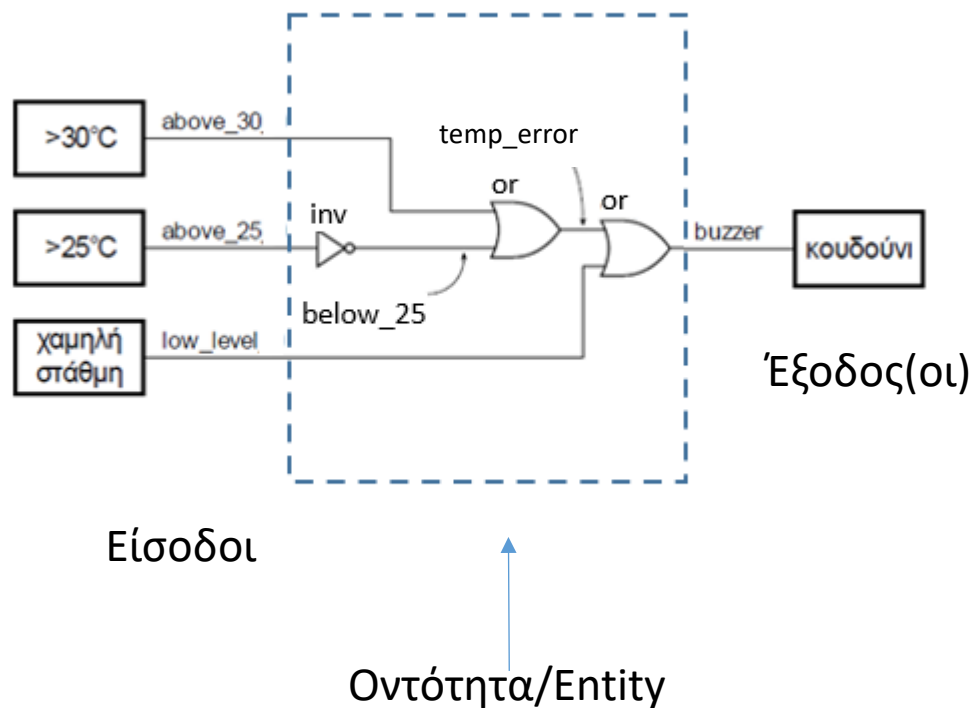


Ψηφιακό κύκλωμα – VHDL: Παράδειγμα

Παράδειγμα: Εργοστάσιο έχει δοχείο επεξεργασίας υγρών. Το δοχείο πρέπει α) να έχει θερμοκρασία μεταξύ 25 και 30 βαθμών και β) η στάθμη του πρέπει να είναι πάνω από ένα επίπεδο. Σε περίπτωση που το α) ή το β) δεν ικανοποιούνται πρέπει να ενεργοποιηθεί ένα κουδούνι. Στη διάθεσή μας έχουμε αισθητήρες (θερμόμετρα) που δείχνουν αν η θερμοκρασία ξεπερνά ένα όριο (το οποίο ορίζεται από εμάς για κάθε αισθητήρα) και αισθητήρα που μας ενημερώνει εάν η στάθμη του δοχείου είναι κάτω από ένα επίπεδο ασφαλείας. Περιγράψτε το σύστημα.



Ψηφιακό κύκλωμα – VHDL: Entity (Input/Output PORTS)



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

Υποχρεωτικές Βιβλιοθήκες

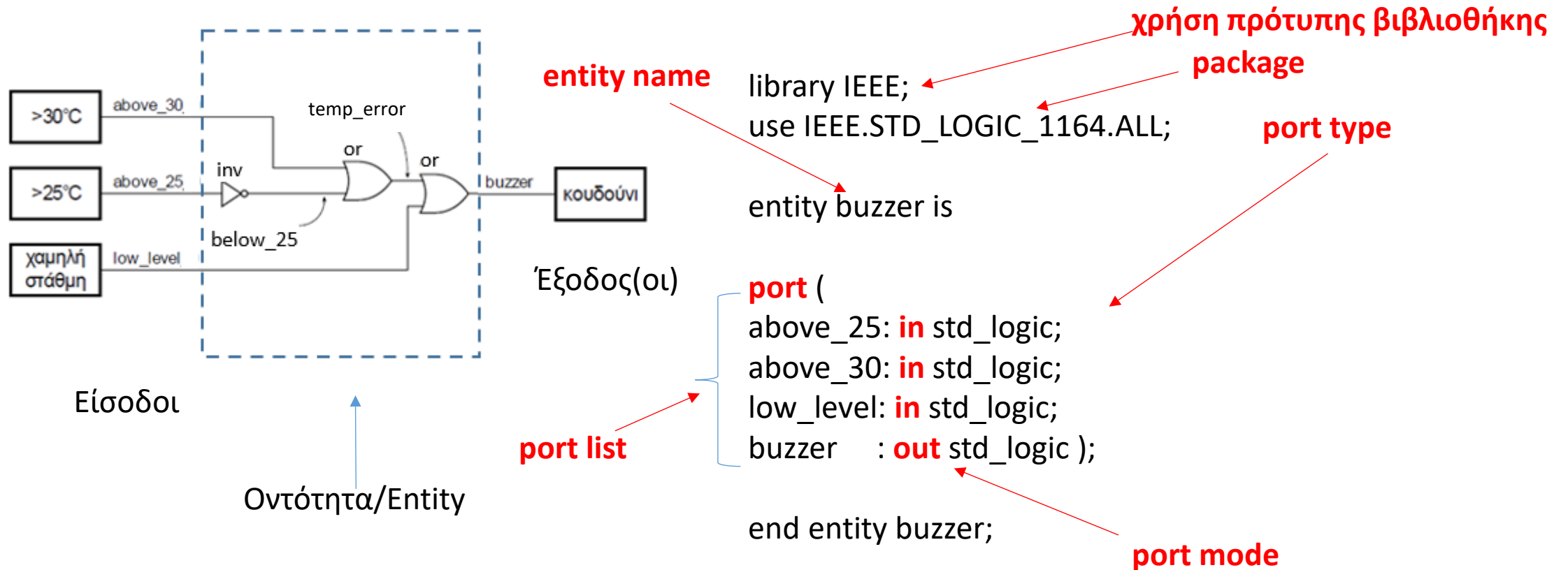
entity buzzer is

Περιγραφή Οντότητας

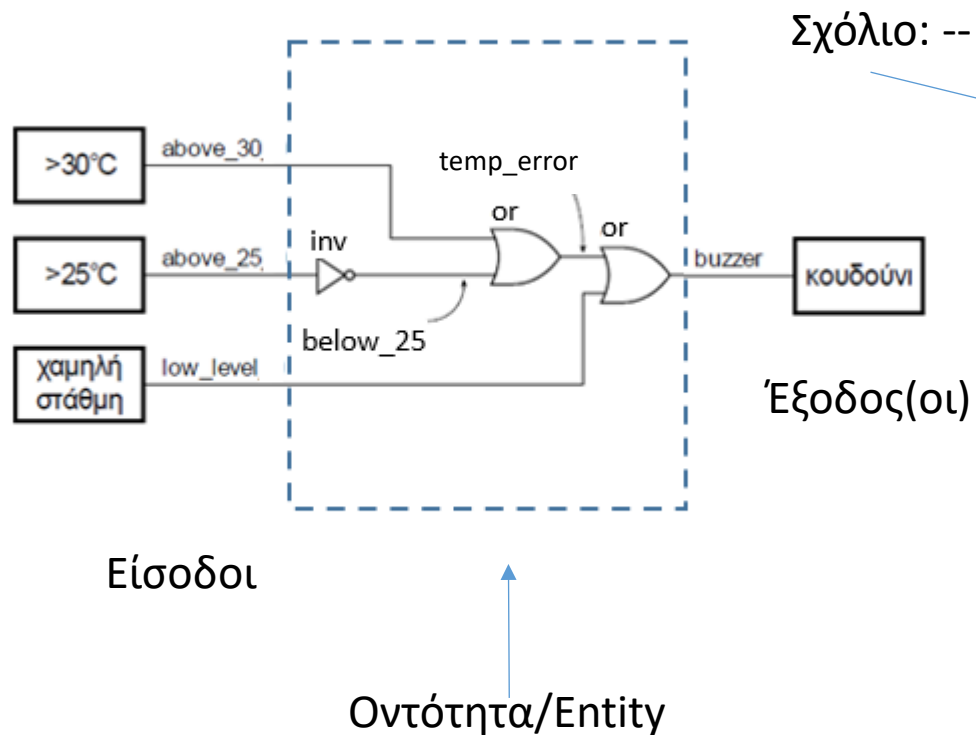
port (
above_25: **in** std_logic;
above_30: **in** std_logic;
low_level: **in** std_logic;
buzzer : **out** std_logic);

end entity buzzer;

Ψηφιακό κύκλωμα – VHDL: Entity (Input/Output PORTS)



Ψηφιακό κύκλωμα – VHDL: Entity (Input/Output PORTS)



Χωρίς τις Βιβλιοθήκες

```
--library IEEE;  
--use IEEE.STD_LOGIC_1164.ALL;
```

Περιγραφή Οντότητας

```
entity buzzer is  
  
  port (  
    above_25: in bit;  
    above_30: in bit;  
    low_level: in bit;  
    buzzer   : out bit);  
  
end entity buzzer;
```

Ο τύπος **bit**, (αλλά και **bit_vector**, ...) υποστηρίζεται χωρίς την ανάγκη κλήσης βιβλιοθηκών

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Entity: Input/Output

- Περιγράφει τη διασύνδεση μίας λογικής μονάδας, χωρίς να προσδιορίζει τη συμπεριφορά της (μαύρο κουτί - black box)
- Η διασύνδεση της μονάδας περιγράφεται με μία δήλωση των **διαύλων/θυρών επικοινωνίας (ports - signals)**

```
entity entity_name is -- σχόλια
    port (
        signal_name: mode
    signal_type;
        signal_name: mode
    signal_type;
        ...
        signal_name: mode
    signal_type);
end entity entity_name;
```

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Entity: Input/Output

- **entity_name**: το όνομα της οντότητας
- **signal_name**: το όνομα του σήματος
(εάν είναι πολλά σήματα χωρίζονται με κόμμα)
- **mode**: η κατεύθυνση του σήματος
 - **in**: είσοδος της οντότητας
 - **out**: έξοδος της οντότητας
 - **inout**: είσοδος ή έξοδος της οντότητας (bidirectional), (ΔΕΝ θα μας απασχολήσει στο μάθημα)
- **signal_type**: ο τύπος του σήματος (STD_LOGIC ή άλλος)

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Ονόματα & Ετικέτες

- Είναι **μοναδικά** μέσα σε μία συγκεκριμένη οντότητα (και αρχιτεκτονική)
- Τα σχόλια σε μία γραμμή έπονται του "--"
- Χρησιμοποιούνται οι χαρακτήρες: **a-z, A-Z, 0-9, "_"**
- Δεν χρησιμοποιούνται οι χαρακτήρες, όπως: **+, -, !, &**
- Δεν χρησιμοποιούνται ούτε **σημεία στίξης** στα ονόματα και τις ετικέτες, ούτε διπλό **"_"**, δηλαδή **"__"**
- Δεν διαχωρίζονται κεφαλαία γράμματα από μικρά
- Ο πρώτος χαρακτήρας είναι **αλφαβητικός**
- **Προσοχή στις δεσμευμένες λέξεις**

VHDL – Τύποι σημάτων

std_logic

Τιμή	Modeling for simulation	Synthesis
U	Uninitialized	Uninitialized
X	Strong driven unknown	Don't care/Multiple Values
0	Strong driven 0	0
1	Strong driven 1	1
Z	High impedance	High impedance/Δεν περνάει ρεύμα
W	Weakly driven unknown	Don't care
L	Weakly driven 0	0
H	Weakly driven 1	1
-	Don't care	Don't care

default initial
value

VHDL – Τύποι σημάτων

Std_logic_vector

```
signal_in_0: in std_logic;  
signal_in_1: in std_logic;  
signal_in_2: in std_logic;  
signal_in_3: in std_logic;
```

Εναλλακτικά

```
signal_in : in std_logic_vector (3 downto 0);
```

```
signal_in <= "0101";
```

```
signal_in(0) <= '1'  
signal_in(1) <= '0'  
signal_in(2) <= '1'  
signal_in(3) <= '0'
```

Προσοχή σε “ και ‘

VHDL – Τύποι σημάτων

Std_logic_vector

- Ο τύπος του λογικού διανύσματος (μονοδιάστατου array) **STD_LOGIC_VECTOR** είναι μέρος του πακέτου **IEEE.std_logic_1164** της βιβλιοθήκης **IEEE**
- Προσδιορίζει ένα διατεταγμένο σύνολο από σήματα (μεταβλητές) τύπου **STD_LOGIC**.
- Η διάταξη μπορεί να είναι αύξουσα
STD_LOGIC_VECTOR (0 to 7)
ή φθίνουσα
STD_LOGIC_VECTOR (7 downto 0)
- Οι δείκτες των στοιχείων του array είναι τύπου **natural**
- Προσοχή, δεν είναι ακέραιος δυαδικός αριθμός

VHDL – Τύποι σημάτων

Std_logic_vector

- Δήλωση τιμών για το 8-ψήφιο λογικό διάνυσμα V
 - `V <= "11110000"`
 - `V <= (others => '0')` -- όλα-0
- Συγκρίσεις:
 - `V = "00000000"` για σύγκριση **ολόκληρου** του διανύσματος
 - `V(3 downto 0) = "0000"` για σύγκριση **μέρους** του διανύσματος
 - Προσοχή. **Μη επιτρεπτή σύγκριση**: `V = "---0000"`
 - το '-' δεν εκλαμβάνεται σαν don't care κατά τη σύγκριση

Προσοχή. Σε όλα τα προγράμματα τα PORT στον ορισμό της Οντότητας θα είναι MONO STD_LOGIC ή STD_LOGIC_VECTOR

VHDL – Τύποι σημάτων

Std_logic_vector

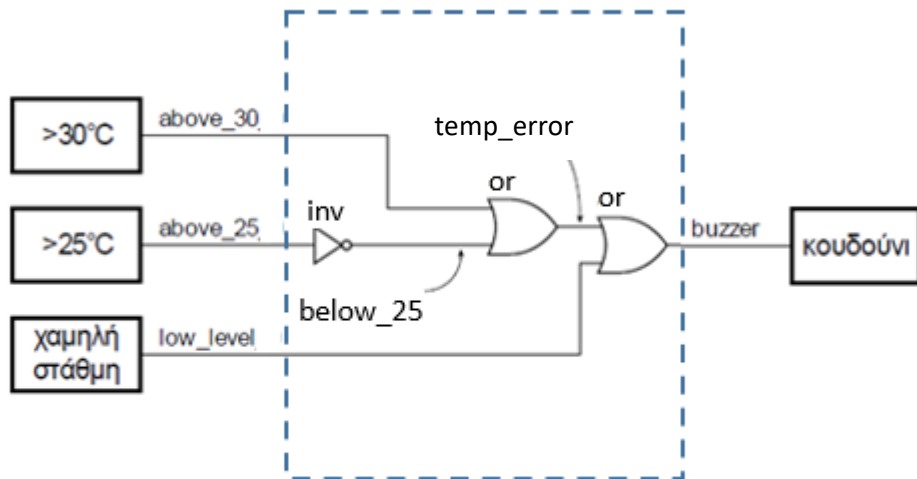
- Μπορούμε να έχουμε και δήλωση σήματος ως std_logic_vector χωρίς index range.
- Σε αυτή την περίπτωση υπονοείται από άλλη οντότητα
- Μπορείτε να βρείτε παράδειγμα :

<https://electronics.stackexchange.com/questions/622590/does-vhdl-allow-a-std-logic-vector-port-with-no-bounds>

VHDL: Architecture (Dataflow)

```
architecture arch_name of entity_name is  
    signal signal_name: signal_type;  
    ...  
begin  
    concurrent_component_statement;  
    ...  
    concurrent_component_statement;  
end architecture arch_name;
```

VHDL: Architecture (Dataflow)



architecture Dataflow of buzzer is

begin

buzzer<= low_level or (above_30 or not above_25);

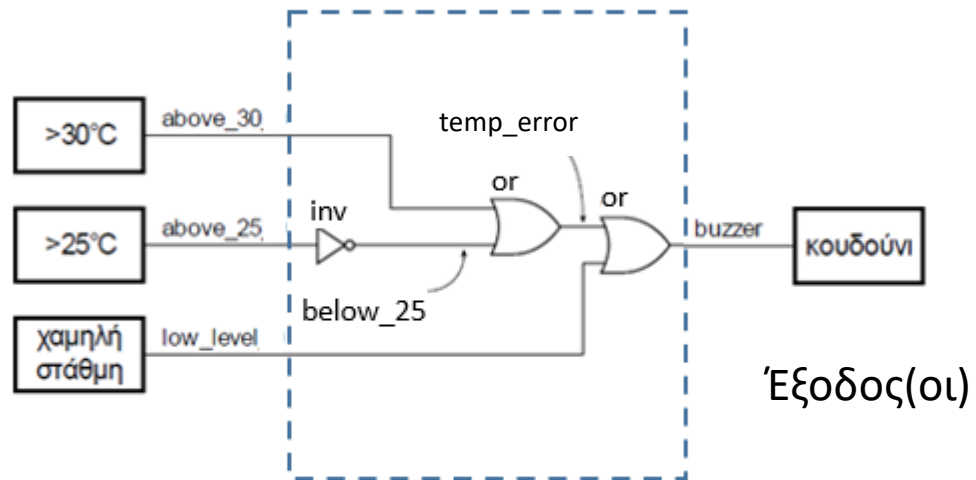
end architecture Dataflow;

Αρχιτεκτονική

Περιγραφή της λειτουργικότητας που προσφέρει το λογικό κύκλωμα

Τα σήματα εξόδου (out) ΠΑΝΤΑ αριστερά, τα σήματα εισόδου (in) ΠΑΝΤΑ δεξιά.

VHDL: Entity (Εσωτερικά σήματα)



Είσοδοι

Οντότητα/Entity

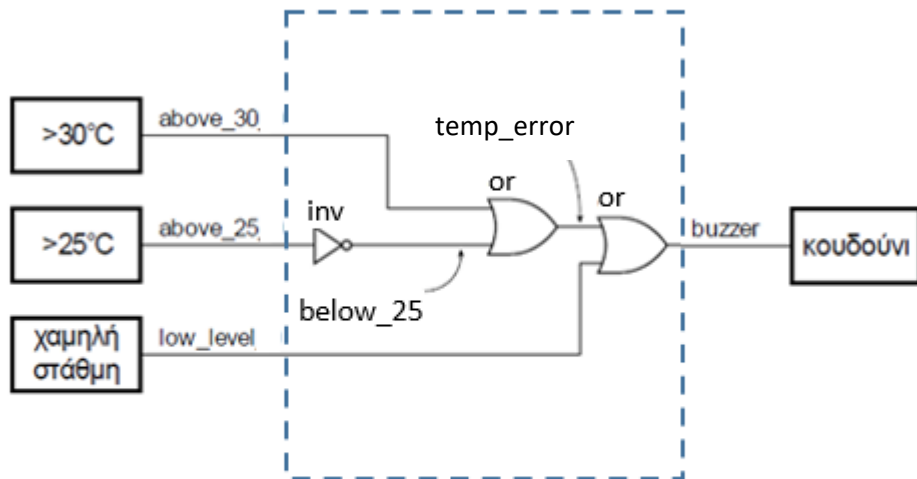
Εσωτερικά σήματα

temp_error

-

below_25

VHDL: Architecture (Dataflow)



architecture Dataflow of buzzer is
signal temp_error, below_25: std_logic;
begin
below_25<=not above_25;
temp_error<=above_30 or below_25;
buzzer<= temp_error or low_level;
end architecture Dataflow;

Περιγραφή της λειτουργικότητας που προσφέρει το λογικό κύκλωμα με **χρήση εσωτερικών σημάτων**

concurrent statements

Τα εσωτερικά σήματα μπορούν να είναι και αριστερά και δεξιά

VHDL: Architecture (Dataflow)

- **Εκτέλεση** ταυτόχρονων εντολών ανάθεσης σήματος (concurrent_signal_assignment_statements)

```
signal_name <= expression;
```

- Οι ταυτόχρονες εντολές ανάθεσης σήματος εκτελούνται μόνο, όταν υπάρξει αλλαγή τιμής στις εισόδους (στα σήματα της δεξιάς πλευράς της ταυτόχρονης εντολής ανάθεσης σήματος).
- Δεν προσδιορίζεται καθυστέρηση διάδοσης άλλη, εκτός από μία απειροελάχιστη καθυστέρηση διάδοσης, την **καθυστέρηση δέλτα δ_{delay}** , που δεν επηρεάζει τον χρονισμό του κυκλώματος
- Η πραγματική καθυστέρηση διάδοσης θα προσδιορισθεί με την υλοποίηση σε μία συγκεκριμένη τεχνολογία

Η **καθυστέρηση δέλτα δ_{delay}** δεν είναι πραγματική καθυστέρηση που επηρεάζει την προσομοίωση, αλλά απλώς ιεραρχεί τις μεταβάσεις που συμβαίνουν στα σήματα την ίδια χρονική στιγμή.

VHDL: Architecture (Dataflow) – After (glitches)

- **Εντολή `after`** : εκτέλεση ταυτόχρονης εντολής με χρονική καθυστέρηση

```
signal_name <= expression after <time>;
```

- **`C <= D and E after 10ns;`**
- Δεν πρέπει να αλλάξει η τιμή κανενός σήματος στα δεξιά της εντολής (expression) πριν περάσει ο χρόνος <time>. Σε περίπτωση όμως που αυτό συμβεί τότε θα είναι απροσδιόριστη η τιμή του signal_name στα αριστερά της εντολής. (Συνήθως η τιμή πριν την εκτέλεση της εντολής).
- Χρησιμοποιείται για προσομοίωση πραγματικών καταστάσεων. Επηρεάζεται μόνο το design model.
- Η `after` δεν είναι synthesizable οπότε αγνοείται στη φάση της σύνθεσης/υλοποίησης (π.χ. ίσως έχουμε λάθος αποτελέσματα σε design model και σωστά σε σύνθεση/υλοποίηση!)

VHDL: Architecture (Dataflow) – transport after

- Εντολή **transport after** : εκτέλεση ταυτόχρονης εντολής με χρονική καθυστέρηση

```
signal_name <= transport expression after <time>;
```

- **C <= transport D and E after 10ns;**
- Το signal name παίρνει την αναμενόμενη τιμή, ανάλογα με την τιμή του expression, με χρονική καθυστέρηση <time>.
- Χρησιμοποιείται για προσομοίωση πραγματικών καταστάσεων. Επηρεάζεται μόνο το design model.
- Η transport after δεν είναι synthesizable οπότε αγνοείται στη φάση της σύνθεσης/υλοποίησης (π.χ. ίσως έχουμε λάθος αποτελέσματα σε design model και σωστά σε σύνθεση/υλοποίηση!)

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Dataflow)

1. Αποτελείται από απλές εντολές ανάθεσης τιμών σε σήματα
2. Κάθε εντολή εκτελείται όταν μεταβληθεί η τιμή ενός σήματος στο αριστερό μέρος.
3. Όλες οι εντολές ανάθεσης εκτελούνται ταυτόχρονα (παράλληλα)
4. Οι εντολές ανάθεσης αντιστοιχούν σε λογικές πράξεις της άλγεβρας Boole.
5. Το RTL διάγραμμα που προκύπτει είναι μία απεικόνιση της άλγεβρας Boole που εκφράζουν οι εντολές ανάθεσης σε στοιχειώδεις λογικές πύλες (AND, OR, NOT) και πολυπλέκτες.

VHDL – Process

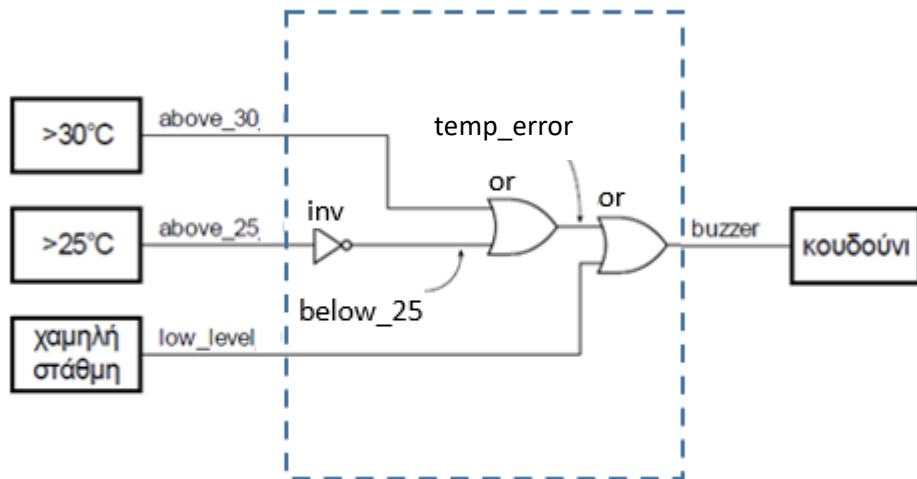
Υλοποίηση Αρχιτεκτονικής (Behavioral)

- **process**: η διεργασία είναι μία ομάδα από εντολές που εκτελούνται ακολουθιακά μετά από κάποιο γεγονός (event: αλλαγή τιμής σήματος)

Sensitivity list

```
architecture arch_name of entity_name is
begin
  label: process (signal_name, ..., signal_name)
    variable variable_name: variable_type;
  begin
    sequential_statement;
    ...
    sequential_statement;
  end process;
end arch_name;
```

Ψηφιακό κύκλωμα – VHDL: Architecture (Behavioral)



architecture behavioral of vat_buzzer is

begin

```
test: process (above_30, above_25, low_level ) is
begin
  buzzer <= low_level or (above_30 or not above_25);
end process test;
end architecture behavioral;
```

Ψηφιακό κύκλωμα – VHDL: Architecture (Structural)

```
architecture arch_name of entity_name is
    signal signal_name: signal_type;
    component comp_name
        port (
            signal_name: mode signal_type;
            ...
            signal_name: mode signal_type);
    end component;
begin
    label_1: comp_name port map (signal_name, ..);
    ...
    label_n: comp_name port map (signal_name, ..);
    concurrent_component_statement;
    ...
    concurrent_component_statement;

end architecture arch_name;
```

Ψηφιακό κύκλωμα – VHDL: Architecture (Structural)

- **arch_name**: το όνομα της αρχιτεκτονικής
- **entity_name**: το όνομα της οντότητας
- **comp_name**: το όνομα του **στοιχείου (component)** που χρησιμοποιείται στην αρχιτεκτονική της οντότητας.
 - Το στοιχείο είναι μία ήδη προκαθορισμένη οντότητα. Ξέρουμε τη λειτουργικότητα που προσφέρει. Όταν επαναχρησιμοποιούμε μια οντότητα (entity) την ονομάζουμε **component**.
- **signal_name**: το όνομα του σήματος (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
 - στις δηλώσεις σημάτων (μετά το **is** και πριν το **begin**) το σήμα είναι μία **εσωτερική διασύνδεση** της αρχιτεκτονική της οντότητας
 - στις δηλώσεις των διαύλων του στοιχείου (component) το σήμα είναι είσοδος, έξοδος του στοιχείου, όπως ακριβώς προκύπτει από τη δήλωση των διαύλων της οντότητας του συγκεκριμένου στοιχείου
- **signal_type**: ο τύπος του σήματος (STD_LOGIC ή άλλος)
- **Ταυτόχρονες εντολές ανάθεσης σήματος** (Όπως σε αρχιτεκτονική Dataflow)

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Αρχιτεκτονική – Components (1/2)

- Ταυτόχρονες εντολές στοιχείων (concurrent component statements)

```
label: comp_name port map (signal_name, ..) ;
```

- **label**: οι μοναδικές ετικέτες των στοιχείων. Δημιουργείται ένα στιγμιότυπο (**instance**) του component με το όνομα **label**. Η χρήση ενός ή παραπάνω component στην αρχιτεκτονική του συστήματος, ονομάζεται **instantiation**.
- **comp_name**: το όνομα του στοιχείου (υποκύκλωμα) που χρησιμοποιείται στην αρχιτεκτονική της οντότητας
- **port_map**: περιγράφει τη σύνδεση των σημάτων της οντότητας με τα port του component. ς
- **signal_name**: το όνομα του σήματος που συνδέεται στο υποκύκλωμα (comp_name) (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
 - το σήμα είναι μία διασύνδεση που αφορά τη συγκεκριμένη αρχιτεκτονική της οντότητας που χρησιμοποιεί το στοιχείο
 - αντιστοιχεί **αμφιμονοσήμαντα** στο αντίστοιχο σήμα της δήλωσης των port του υποκυκλώματος (comp_name) (**θέλει προσοχή η σειρά των σημάτων**)

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Structural – Δήλωση Οντοτήτων)

```
entity OR_gate is  
  port(A : in std_logic;  
        B : in std_logic;  
        O : out std_logic);  
end entity OR_gate;
```

```
architecture Dataflow of OR_gate is  
begin  
  O<=A or B;  
end Dataflow;
```

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Structural – Δήλωση Οντοτήτων)

```
entity NOT_gate is  
  port(A : in std_logic;  
        O : out std_logic);  
end entity NOT_gate;
```

```
architecture Dataflow of NOT_gate is  
begin  
  O<=not A;  
end Dataflow;
```

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Structural – Δήλωση Αρχιτεκτονικής Κύριας Οντότητας)

```
architecture Structural of buzzer is
    begin

    component OR_gate is
        port(A : in std_logic;
             B : in std_logic;
             O : out std_logic);
    end component OR_gate;

    component NOT_gate is
        port(A : in std_logic;
             O : out std_logic);
    end component NOT_gate;

    signal below_25: std_logic;
    signal temp_error: std_logic;

    comp_not: NOT_gate port map(A=>above_25, O=>below_25);
    comp_or1: OR_gate port map (A=>above_30, B=>below_25, O=>temp_error);
    comp_or2: OR_gate port map (A=>low_level, B=>temp_bad, O=>buzzer);

end architecture Structural;
```

Υλοποίηση Αρχιτεκτονικής (Structural)

1. Για την υλοποίηση της αρχιτεκτονικής χρησιμοποιούμε τη λειτουργικότητα άλλων Οντοτήτων που έχουμε ήδη υλοποιήσει.
2. Τις οντότητες που θα χρησιμοποιήσουμε τις δηλώνουμε (όνομα και port) ανάμεσα στο `is` και το `begin` της αρχιτεκτονικής. Όμως πλέον έχουν την έννοια της συνιστώσας (component).
3. Η αρχιτεκτονική πλέον αποτελείται και από εντολές που καλούν(δημιουργούν) τα components, τα οποία μπορεί και να επικοινωνούν μεταξύ τους (συνηθέστερη περίπτωση).
4. Άρα μια οντότητα μπορεί να χρησιμοποιεί άλλες οντότητες (με τη μορφή component), οι οποίες μπορεί να είναι υλοποιημένες με οποιοδήποτε από τις 3 αρχιτεκτονικές. Στη περίπτωση της structural δομής σχηματίζεται ένα δέντρο, τα φύλλα του οποίου είναι άλλες οντότητες. Τα φύλλα που είναι τερματικά έχουν δομή Dataflow ή Behavioral.
5. ΔΕΝ αλλάζει τίποτα στην προσομοίωση

Μπορείτε να δείτε και το: <https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/>

VHDL – Packages

Package: IEEE.std_logic_1164.all

Ορίζει 4 τύπους:

std_logic
std_logic_vector



Resolved: Αν υπάρχουν 2 εντολές που δίνουν διαφορετικές τιμές στο ίδιο σήμα, η VHDL θα επιλύσει τη διαφορά επιλέγοντας μια συγκεκριμένη τελική τιμή για το σήμα

std_ulogic
std_ulogic_vector

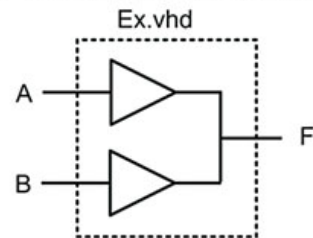
UnResolved: Αν υπάρχουν 2 εντολές που δίνουν διαφορετικές τιμές στο ίδιο σήμα, η VHDL ΔΕΝ θα μπορέσει να επιλύσει τη διαφορά και θα προκύψει compiler error

VHDL – Packages

Package: IEEE.std_logic_1164.all

STD_LOGIC_1164 Unresolved vs. Resolved Conflict Handling

The std_logic_1164 package has data types that support this type of topology.



```
architecture Ex_arch of Ex is
begin
    F <= A;
    F <= B;
end architecture;
```

*standard Package
with types bit*

```
entity Ex is
    port (A,B : in bit;
          F : out bit);
end entity;
```

NOT ALLOWED. This will
result in compiler error
"unresolved data type".

*std_logic_1164
package with types
std_ulogic*

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

entity Ex is
    port (A,B : in std_ulogic;
          F : out std_ulogic);
end entity;
```

NOT ALLOWED. This will
result in compiler error
"unresolved data type".

*std_logic_1164
package with types
std_logic*

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

entity Ex is
    port (A,B : in std_logic;
          F : out std_logic);
end entity;
```

ALLOWED. The output F
will be determined using a
resolution function.

VHDL – Packages

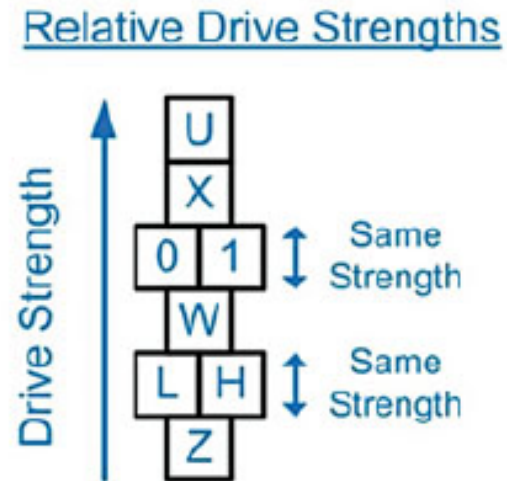
Package: IEEE.std_logic_1164.all,
Std_logic – Συνδυασμοί δύο σημάτων

U	X	0	1	Z	W	L	H	-	
U	X	X	1	1	1	1	1	X	1
U	X	0	X	0	0	0	0	X	0
U	U	U	U	U	U	U	U	U	U
U	X	X	X	X	X	X	X	X	X
U	X	0	1	Z	W	L	H	X	Z
U	X	0	1	W	W	W	W	X	W
U	X	0	1	L	W	L	W	X	L
U	X	0	1	H	W	W	H	X	H
U	X	X	X	X	X	X	X	X	-

resolution table για σήμα std_logic με δύο drivers
https://vhdlwhiz.com/std_logic/

VHDL – Packages

Package: IEEE.std_logic_1164.all
Std_logic – Relative Drive Strengths



VHDL – Packages

Package: IEEE.std_logic_1164.all

Logical Operators

and, nand, or, nor, xor, xnor, not

Edge Detection

rising_edge(), falling_edge()

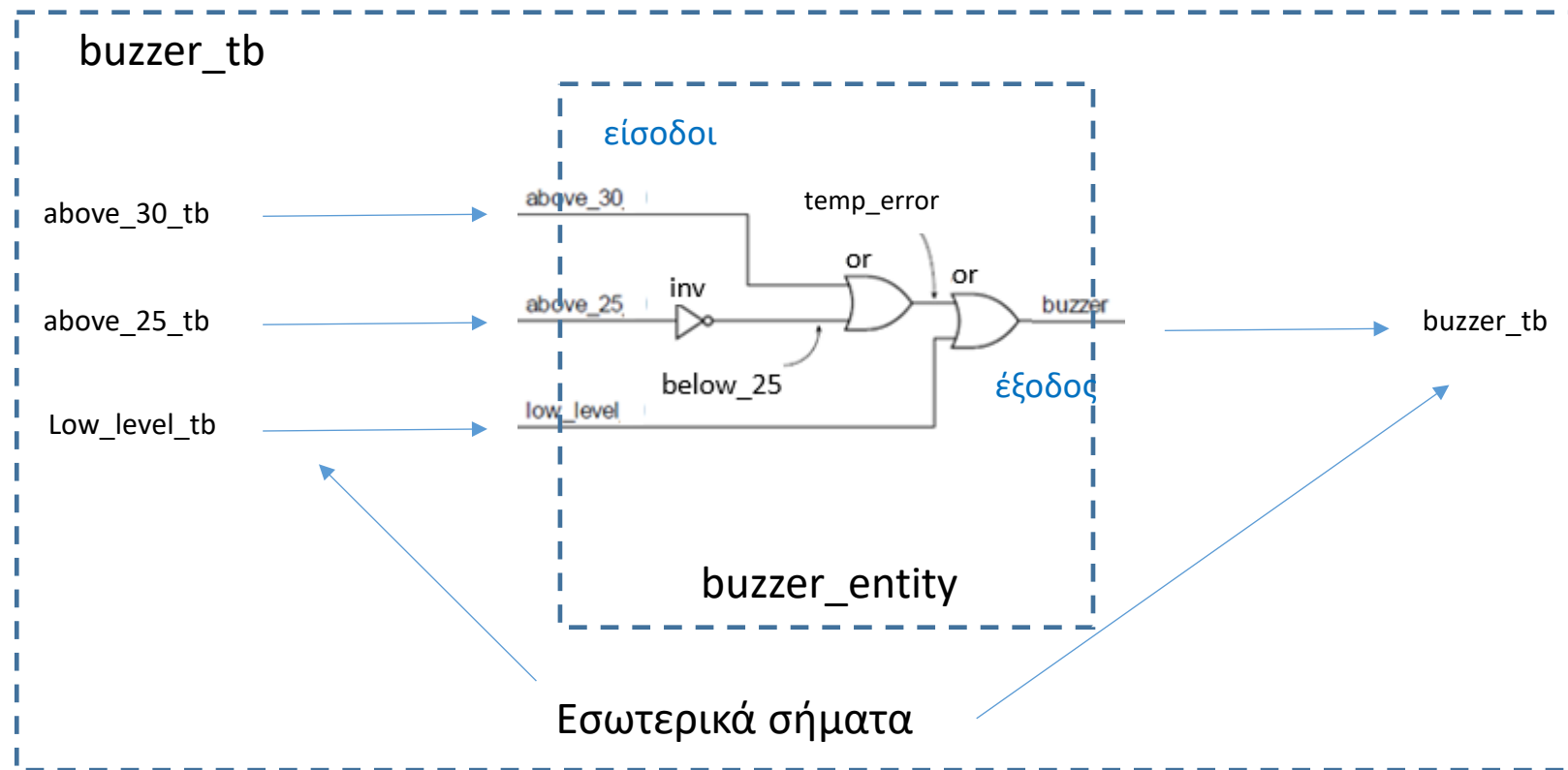
VHDL – Package: IEEE.std_logic_1164.all

Type conversions

Name	Input type	Return type
To_bit()	std_ulogic	bit
To_bitvector()	std_ulogic_vector	bit_vector
To_bitvector()	std_logic_vector	bit_vector
To_StdULogic()	bit	std_ulogic
To_StdULogicVector()	bit_vector	std_ulogic_vector
To_StdULogicVector()	std_logic_vector	std_ulogic_vector
To_StdLogicVector()	bit_vector	std_logic_vector
To_StdLogicVector()	std_ulogic_vector	std_logic_vector

Όταν υπάρχουν ίδια ονόματα συναρτήσεων, αυτές ξεχωρίζουν από τον διαφορετικό τύπο του ορίσματος

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση



Ψηφιακό κύκλωμα – VHDL: Προσομοίωση – Πίνακας Αληθείας

Πίνακας Αληθείας της συνάρτησης που εκφράζει το σήμα buzzer

above_30	above_25	low_level	buzzer
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Testbench

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
```

```
entity buzzer_tb is  
end buzzer_tb ;
```

```
architecture Behavioral of buzzer_tb is
```

```
component buzzer is port(  
  above_30: in std_logic;  
  above_25: in std_logic;  
  low_level: in std_logic;  
  buzzer: out std_logic);  
end component buzzer;
```

```
signal  above_25_tb, above_30_tb, low_level_tb  : std_logic;  
signal  buzzer_tb                               : std_logic;
```

```
begin  
 uut: buzzer port map (above_30_tb, above_25_tb, low_level_tb,  
  buzzer_tb);
```

```
  apply_test_cases: process is  
begin
```

```
  above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;  
  above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;  
  above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;  
  above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;  
  above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;  
  above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;  
  above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;  
  above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;  
end process apply_test_cases;
```

```
end architecture Behavioral;
```

Όταν γράφουμε μόνο τα εσωτερικά σήματα, η σειρά τους καθορίζει έμμεσα την αντιστοίχιση με τα port του component

POSITIONAL PORT MAPPING

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Testbench

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
```

```
entity buzzer_tb is  
end buzzer_tb ;
```

```
architecture Behavioral of buzzer_tb is
```

```
component buzzer is port(  
  above_30: in std_logic;  
  above_25: in std_logic;  
  low_level: in std_logic;  
  buzzer: out std_logic);  
end component buzzer;
```

```
signal  above_25_tb, above_30_tb, low_level_tb  : std_logic;  
signal  buzzer_tb                                : std_logic;
```

```
begin
```

```
 uut: buzzer port map (  
  above_25 => above_25_tb,  
  above_30 => above_30_tb,  
  low_level => low_level_tb,  
  buzzer => buzzer_tb);
```

```
apply_test_cases: process is  
begin
```

```
  above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;  
  above_30_tb <='0'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;  
  above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;  
  above_30_tb <='0'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;  
  above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='0'; wait for 20 ns;  
  above_30_tb <='1'; above_25_tb <='0'; low_level_tb <='1'; wait for 20 ns;  
  above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='0'; wait for 20 ns;  
  above_30_tb <='1'; above_25_tb <='1'; low_level_tb <='1'; wait for 20 ns;  
end process apply_test_cases;
```

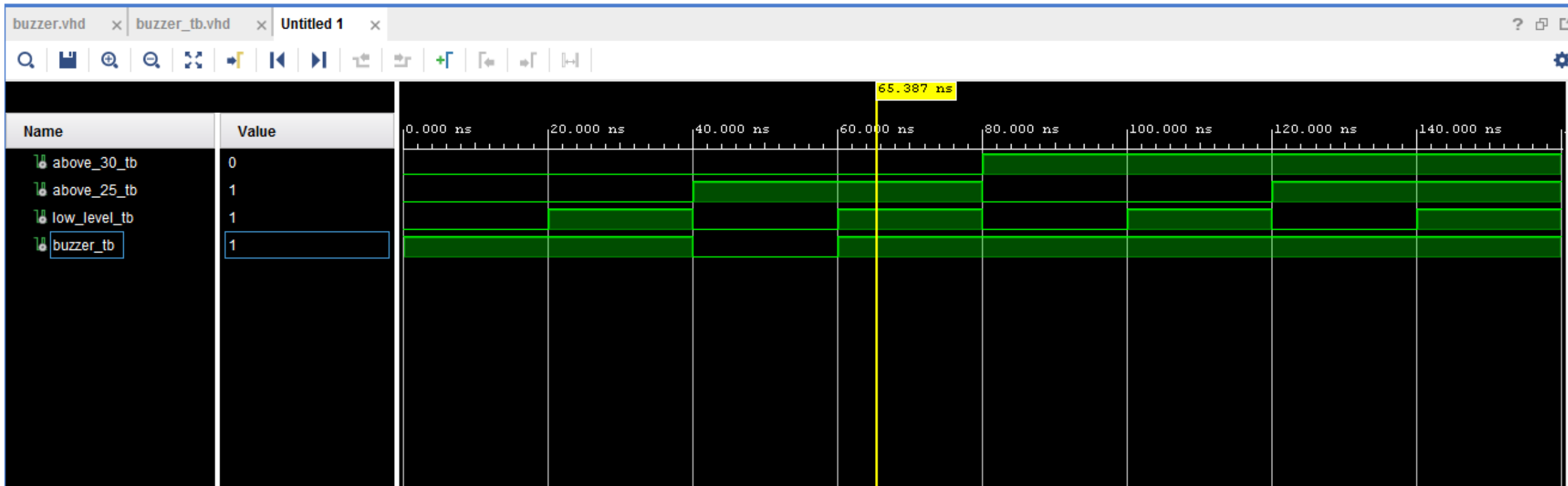
```
end architecture Behavioral;
```

Όταν ορίζουμε άμεσα την αντιστοίχιση των εσωτερικών σημάτων με τα port, η σειρά με την οποία το κάνουμε είναι αδιάφορη

EXPLICIT PORT MAPPING

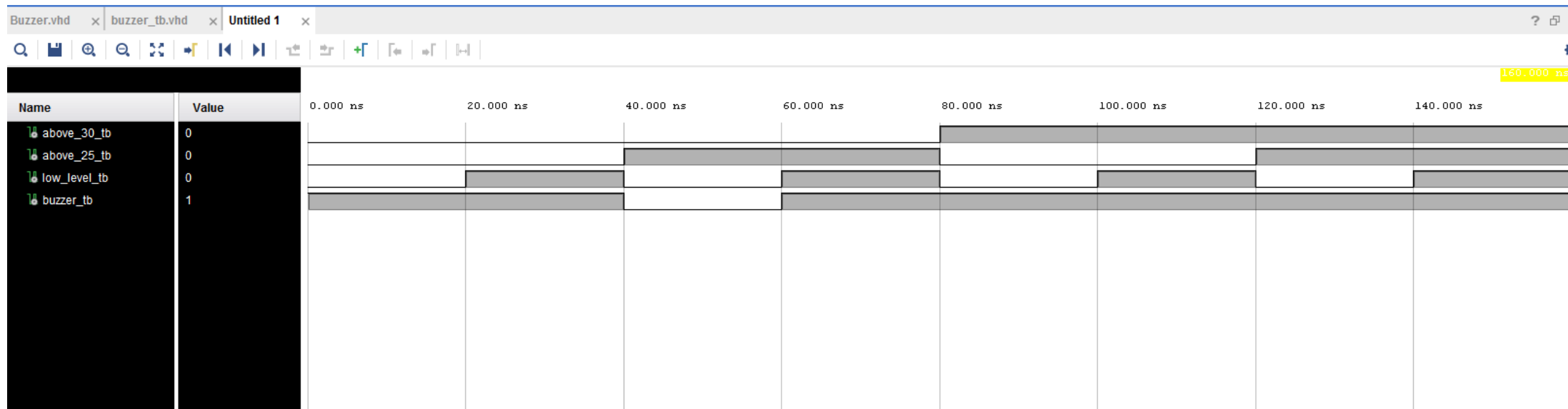
VHDL

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Χρονοσειρά



VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Χρονοσειρά



Boards

- Boards που μπορείτε να ερευνήσετε για home use:

Without ARM

<https://www.digikey.gr/en/products/detail/digilent-inc/410-183/4970275>

With ARM (Zynq7000)

<https://www.digikey.gr/en/products/detail/digilent-inc/410-370/9445909>

<https://www.digikey.gr/en/products/detail/digilent-inc/6003-410-017/9839382>

<https://www.digikey.gr/en/products/detail/digilent-inc/410-351-10/7652757>

Περίληψη

- Παράδειγμα ανάπτυξης εφαρμογής σε VHDL
- Δηλώσεις Οντότητας (entity), Αρχιτεκτονικής (architecture).
- Ports και Εσωτερικά σήματα
- Παράδειγμα των 3 ειδών αρχιτεκτονικής (Dataflow, Behavioral, Structural)
- Πακέτο std_logic_1164.all
- Τύποι σημάτων (std_logic, std_logic_vector)
- Διαβάζετε τις παραγράφους 2.1, 2.6, 4.2.5, 4.2.8, 4.2.9, 4.9, από το βιβλίο των Harris.
- Διαβάζετε τις παραγράφους 12.1, 12.2.2, 12.2.3, 12.2.4, 12.4.1, 12.5, 12.6, 12.6.1 από το βιβλίο των Brown - Vranesic
- Συνοπτικός οδηγός VHDL:
https://redirect.cs.umbc.edu/portal/help/VHDL/summary_one.html

3^ο Εργαστήριο

Σχεδίαση Ψηφιακών Συστημάτων

3^η Διάλεξη

Γνωριμία με τη VHDL και το VIVADO

Βασιλόπουλος Διονύσης

Ε.ΔΙ.Π. Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

VHDL – Παράδειγμα

Άσκηση

Σχεδιάστε ένα ψηφιακό κύκλωμα που θα δέχεται 2 εισόδους data1_in και data2_in των 2 bit η κάθε μία και θα έχει δύο εξόδους data1_out του ενός (1) bit και data2_out των 2 bit. Η έξοδος data1_out θα είναι '1' όταν το πλήθος των bit που είναι '1' και των δύο εισόδων είναι άρτιο, αλλιώς θα είναι '0'. Η έξοδος data2_out θα έχει το λογικό xor των δύο εισόδων. Δημιουργήστε ένα νέο project με το όνομα DSD_Lab3, με όνομα οντότητας σχεδίασης DSD_Lab3, όνομα οντότητας simulation DSD_Lab3_tb. Οι αντίστοιχες αρχιτεκτονικές θα λέγονται DF και DF_tb.

Σχεδιάστε και υλοποιήστε το λογικό κύκλωμα που περιγράφει το ανωτέρω πρόβλημα.

Σχεδιάστε το κύκλωμα και με αρχιτεκτονικές Behavioral και Structural

VHDL – Παράδειγμα

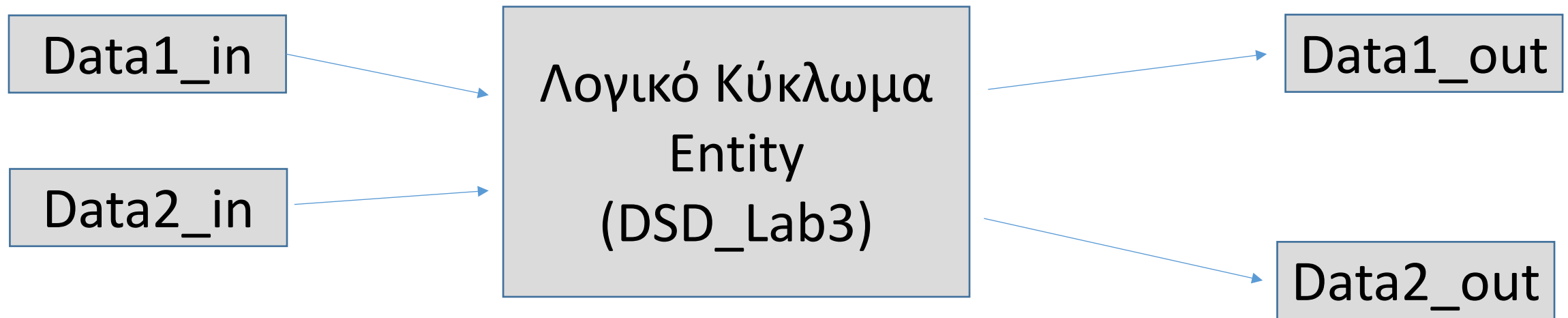
Πραγματικό πρόβλημα – Βήματα επίλυσης

1. Δημιουργία νέου project
2. Δημιουργία Entity – Εντοπισμός Input/Output του συστήματος
3. Εύρεση πίνακα αληθείας για κάθε έξοδο του συστήματος
4. Δημιουργία Architecture – Θα έχετε τουλάχιστον τόσες εντολές όσες είναι και οι έξοδοι του συστήματος. Κάθε μία εντολή αντιστοιχεί σε μία έξοδο.
5. Δημιουργία RTL αναπαράστασης
6. Προσομοίωση
7. Σύνθεση

(Αναλυτικές οδηγίες στον Οδηγό Vivado στο eclass)

VHDL – Παράδειγμα

Απλοποιημένη μορφή κυκλώματος – Είσοδοι/Εξοδοι



VHDL – Παράδειγμα

Βήμα 2: Περιγραφή Οντότητας

Χρήση βιβλιοθηκών
και πακέτων

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

Περιγραφή Οντότητας

```
entity DSD_Lab3 is  
  Port ( data1_in : in STD_LOGIC_VECTOR (1 downto 0);  
         data2_in : in STD_LOGIC_VECTOR (1 downto 0);  
         data1_out : out STD_LOGIC;  
         data2_out : out STD_LOGIC_VECTOR (1 downto 0)  
        );  
end entity DSD_Lab3;
```

VHDL – Παράδειγμα

Βήμα 3: Πίνακας αληθείας κυκλώματος για έξοδο Data1_out

Data1_in(1)	Data1_in(0)	Data2_in(1)	Data2_in(0)	Data1_out
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

VHDL – Παράδειγμα

Βήμα 3: Πίνακας Karnaugh για εύρεση και απλοποίηση συνάρτησης εξόδου

Data1_in\Data2_in	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

Data1_out=

VHDL – Παράδειγμα

Βήμα 4a: Περιγραφή Αρχιτεκτονικής - Dataflow

architecture DF of DSD_Lab3 is

begin

data1_out<= not(data1_in(0) xor data1_in(1) xor data2_in(0) xor data2_in(1)); --xnor

data2_out<= data1_in xor data2_in;

end architecture DF;

Αρχιτεκτονική

Περιγραφή της λειτουργικότητας που προσφέρει το λογικό κύκλωμα

Απόδοση τιμής σε σήμα

Εκτελούνται ταυτόχρονα

VHDL – Παράδειγμα

Βήμα 4b: Περιγραφή Αρχιτεκτονικής - Behavioral

```
architecture Beh of DSD_Lab3 is  
begin
```

```
beh_arc: process (data1_in, data2_in) is  
begin
```

```
data1_out<= not( data1_in(0) xor data1_in(1) xor data2_in(0) xor data2_in(1));    --xnor  
data2_out<= data1_in xor data2_in;
```

```
end process beh_arc;
```

```
end architecture Beh;
```

Αρχιτεκτονική Behavioral

Χρήση δομής process (οι εντολές
εντός της εκτελούνται
ακολουθιακά)

VHDL – Παράδειγμα

Βήμα 4c: Περιγραφή Αρχιτεκτονικής - Structural

architecture Structural of DSD_Lab3 is

```
component Xor_Gate is
  Port ( IO : in STD_LOGIC;
        I1 : in STD_LOGIC;
        O : out STD_LOGIC);
end component Xor_Gate;
```

```
component Not_Gate is
  Port ( IO : in STD_LOGIC;
        O : out STD_LOGIC);
end component Not_Gate;
```

```
signal temp_xor1 : STD_LOGIC;
signal temp_xor2 : STD_LOGIC;
signal temp_xor3 : STD_LOGIC;
```

begin

```
xor1: Xor_Gate port map(data1_in(0),data1_in(1), temp_xor1);
xor2: Xor_Gate port map(data2_in(0),data2_in(1), temp_xor2);
xor3: Xor_Gate port map(temp_xor1,temp_xor2, temp_xor3);
not1: Not_gate port map(temp_xor3, data1_out);
```

```
xor4: Xor_Gate port map(data1_in(0),data2_in(0),
data2_out(0));
xor5: Xor_Gate port map(data1_in(1),data2_in(1),
data2_out(1));
```

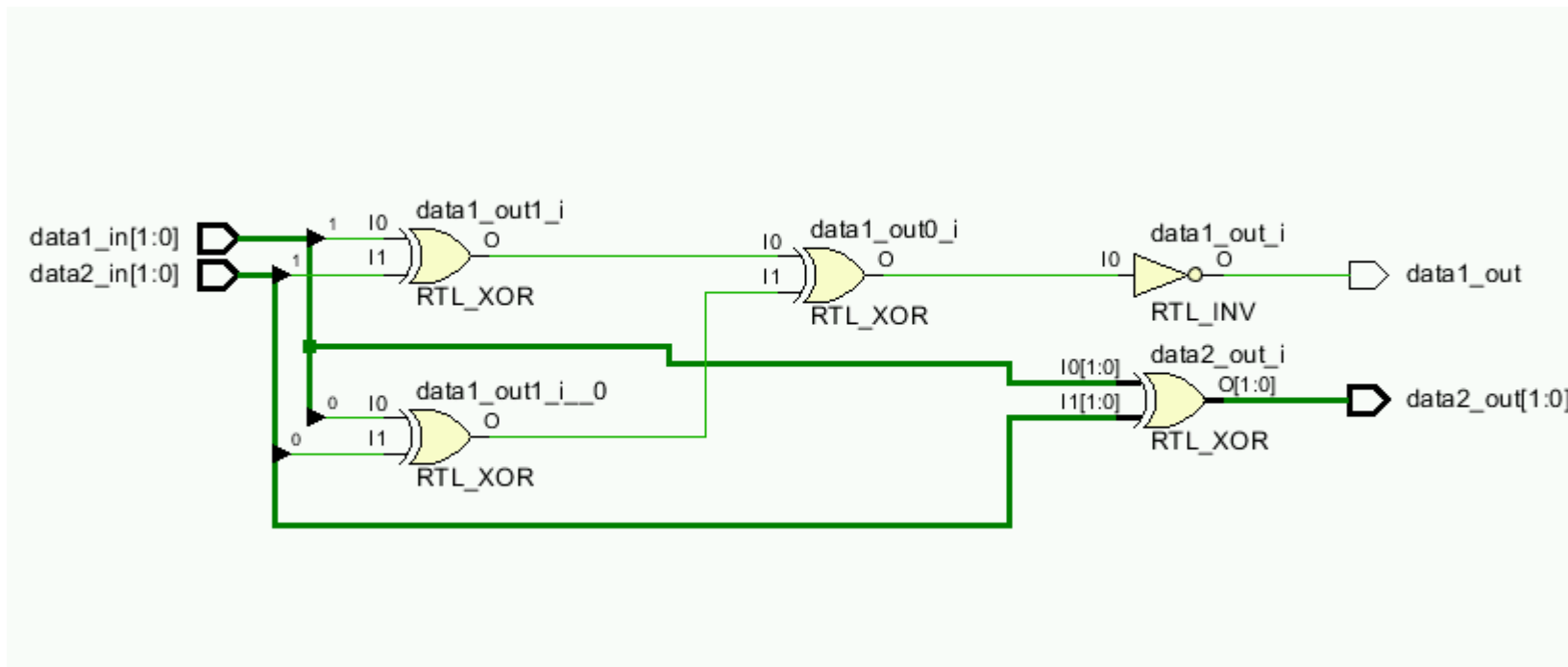
end architecture Structural;

← Αρχιτεκτονική Structural
Επαναχρησιμοποίηση οντοτήτων
που ήδη έχουμε δημιουργήσει

← Εσωτερικά σήματα:
Μόνο τύπος, όχι in/out

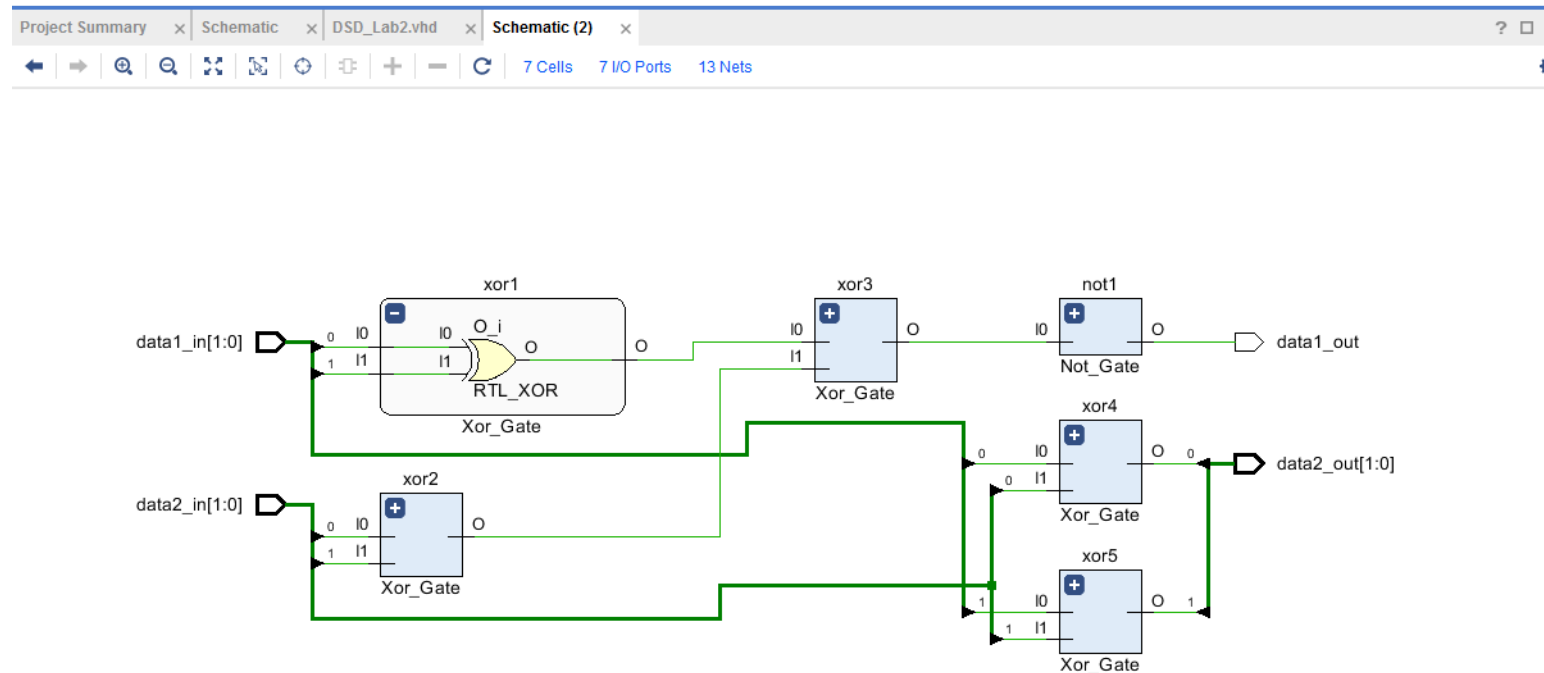
VHDL – Παράδειγμα

Βήμα 5α: RTL Αναπαράσταση – Dataflow/Behavioral



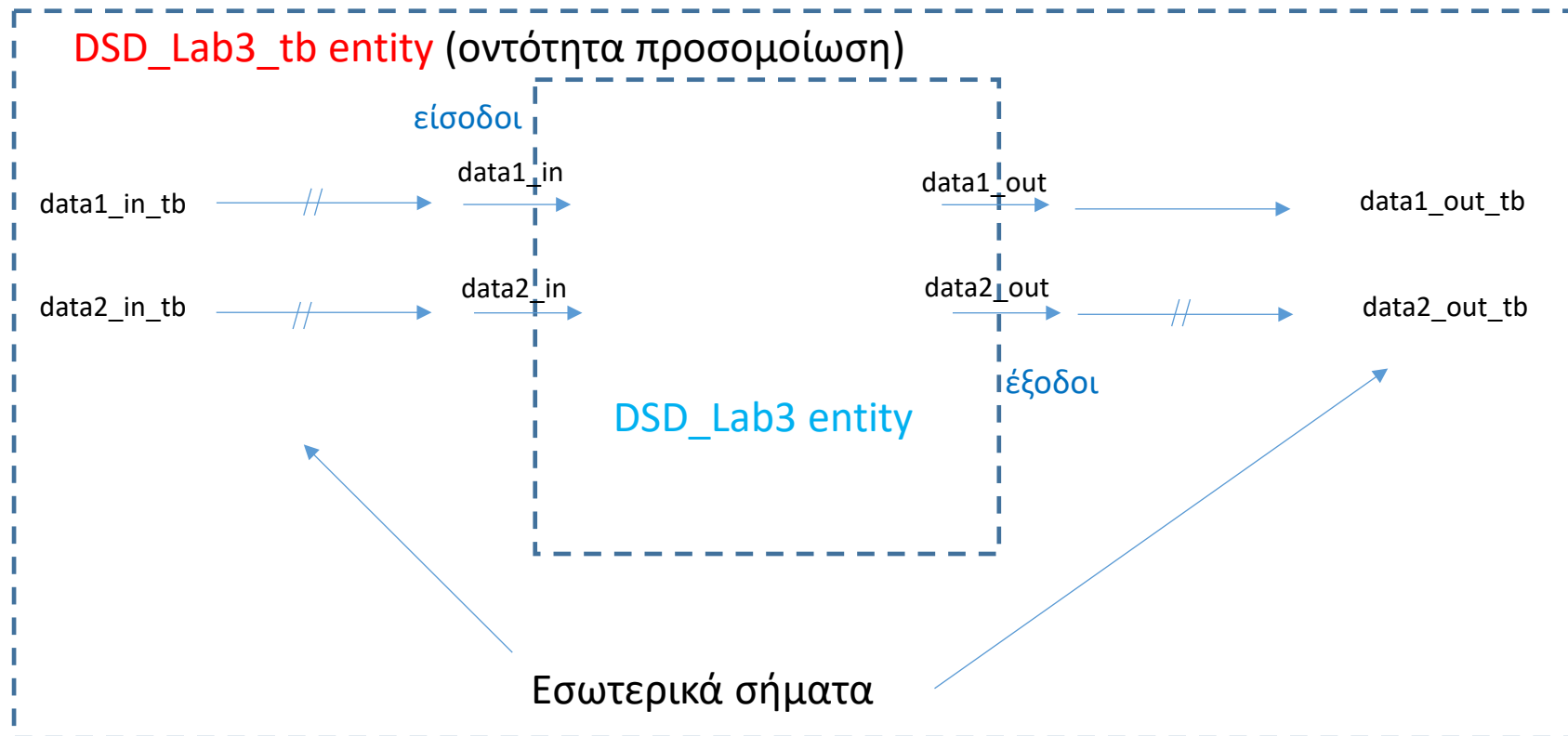
VHDL – Παράδειγμα

Βήμα 5b: RTL Αναπαράσταση – Structural



VHDL – Παράδειγμα

Βήμα 6: Simulation



VHDL – Παράδειγμα

Βήμα 6α: Simulation (Πρόγραμμα testbench)

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;  
entity DSD_Lab3_tb is  
end DSD_Lab3_tb;
```

Συνήθως έχει το όνομα της
οντότητας που
προσομοιώνουμε αλλά με
την κατάληξη _tb

```
architecture DF_tb of DSD_Lab3_tb is
```

```
component DSD_Lab3 is
```

```
    Port ( data1_in : in STD_LOGIC_VECTOR (1 downto 0);  
          data2_in : in STD_LOGIC_VECTOR (1 downto 0);  
          data1_out : out STD_LOGIC;  
          data2_out : out STD_LOGIC_VECTOR (1 downto 0)  
    );
```

```
end component DSD_Lab3 ;
```

```
-- Internal Signals: One for each port
```

```
signal data1_in_tb : STD_LOGIC_VECTOR (1 downto 0);  
signal data2_in_tb : STD_LOGIC_VECTOR (1 downto 0);  
signal data1_out_tb : STD_LOGIC;  
signal data2_out_tb : STD_LOGIC_VECTOR (1 downto 0)
```

```
begin
```

```
    uut: DSD_Lab3 port map (data1_in=>data1_in_tb,data2_in=>data2_in_tb,  
    data1_out=>data1_out_tb,data2_out=>data2_out_tb);
```

```
test_DF: process is
```

```
begin
```

```
    data1_in_tb<="00";data2_in_tb<="00";wait for 20 ns;  
    data1_in_tb<="00";data2_in_tb<="01";wait for 20 ns;  
    data1_in_tb<="00";data2_in_tb<="10";wait for 20 ns;  
    data1_in_tb<="00";data2_in_tb<="11";wait for 20 ns;  
    data1_in_tb<="01";data2_in_tb<="00";wait for 20 ns;  
    data1_in_tb<="01";data2_in_tb<="01";wait for 20 ns;  
    data1_in_tb<="01";data2_in_tb<="10";wait for 20 ns;  
    data1_in_tb<="01";data2_in_tb<="11";wait for 20 ns;  
    data1_in_tb<="10";data2_in_tb<="00";wait for 20 ns;  
    data1_in_tb<="10";data2_in_tb<="01";wait for 20 ns;  
    data1_in_tb<="10";data2_in_tb<="10";wait for 20 ns;  
    data1_in_tb<="10";data2_in_tb<="11";wait for 20 ns;  
    data1_in_tb<="11";data2_in_tb<="00";wait for 20 ns;  
    data1_in_tb<="11";data2_in_tb<="01";wait for 20 ns;  
    data1_in_tb<="11";data2_in_tb<="10";wait for 20 ns;  
    data1_in_tb<="11";data2_in_tb<="11";wait for 20 ns;
```

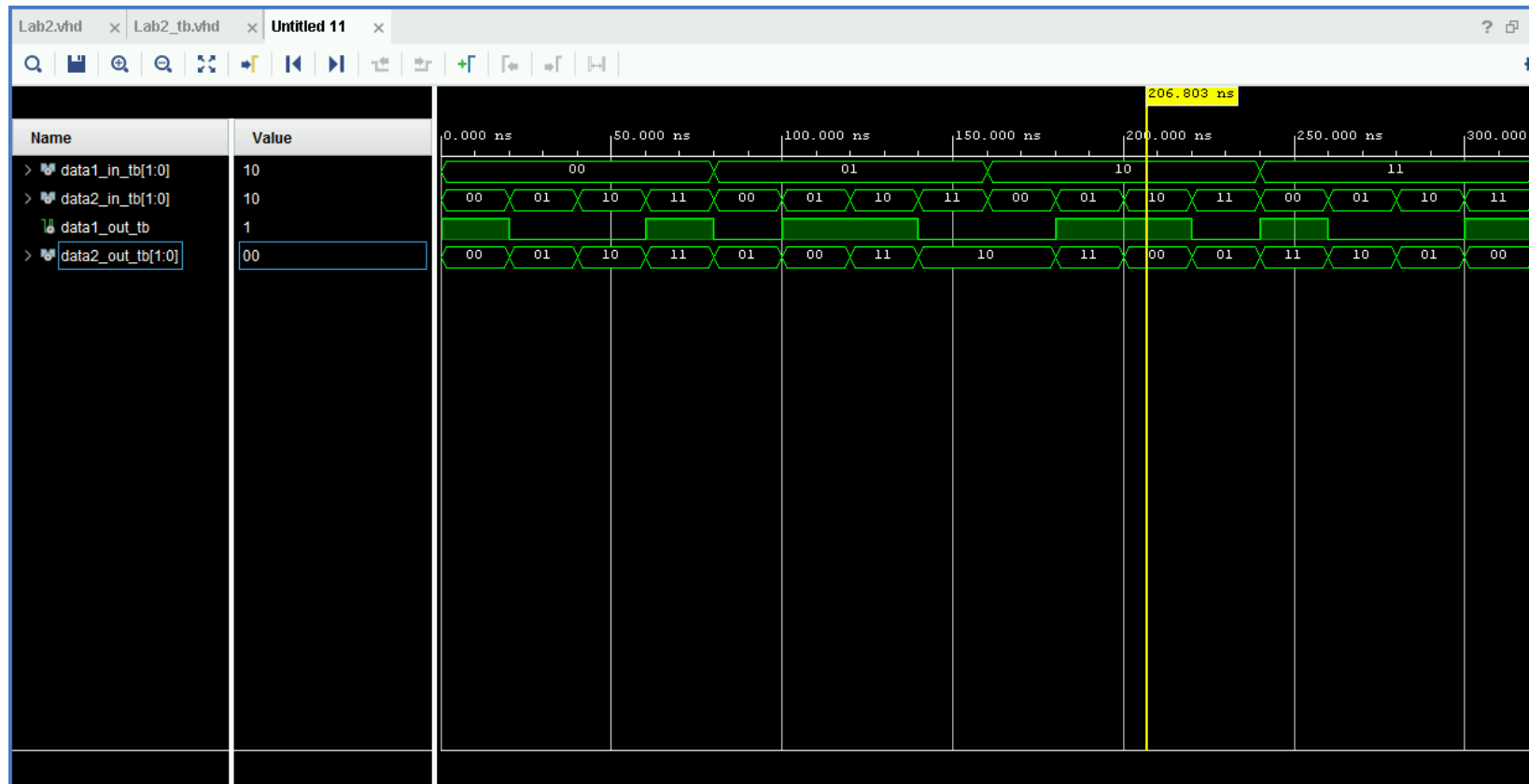
```
end process test_DF;
```

```
end DF_tb;
```

Όλοι οι δυνατοί
συνδυασμοί

VHDL – Παράδειγμα

Βήμα 6b: Waveform



VHDL – Improving code

Βήμα 6c: Simulation

```
use IEEE.NUMERIC_STD.ALL; -- package declaration part
```

```
test_DF: process is  
begin
```

```
for i in 0 to 3 loop
```

```
    data1_in_tb<=std_logic_vector(to_unsigned(i, data1_in_tb'length));
```

```
    for j in 0 to 3 loop
```

```
        data2_in_tb<=std_logic_vector(to_unsigned(j, data2_in_tb'length));wait for 20 ns;
```

```
    end loop;
```

```
end loop;
```

```
end process test_DF;
```

VHDL – Παράδειγμα

Βήμα 7α: Σύνθεση – LUT(LookUp Table), υλοποιούν Πίνακες Αληθείας (προγραμματιζόμενα μέρη της κάρτας FPGA)

Nets->Σύρματα

Cells->Ψηφιακά στοιχεία

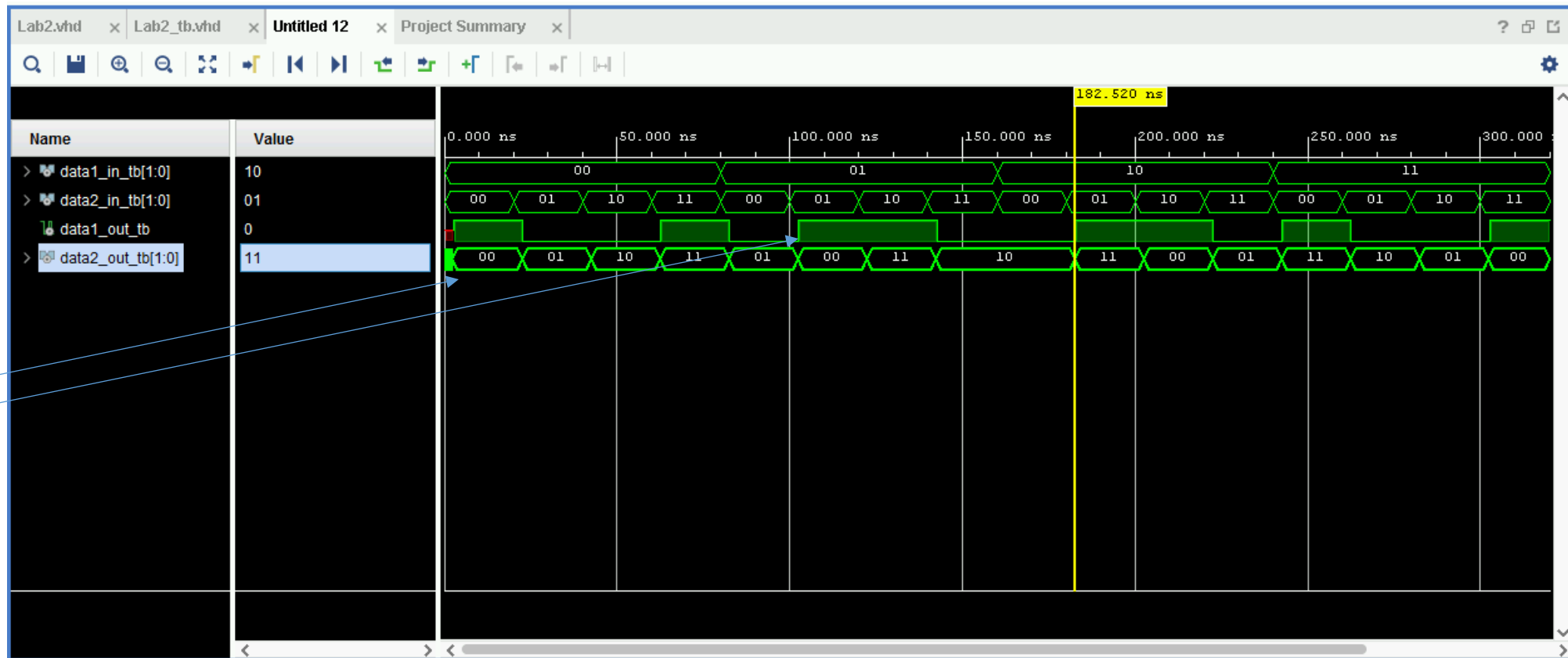
LUT: Πίνακας Αληθείας

The screenshot displays the Vivado 2022.2 interface during the synthesis phase. The left-hand 'Project Manager' pane shows the 'SYNTHESIS' step as the current active task. The central 'Netlist' pane lists the components synthesized, including input buffers (IBUF), output buffers (OBUF), and Look-Up Tables (LUT4 and LUT2). The right-hand 'Schematic' pane shows a logic diagram where a LUT4 component is configured to implement a truth table, with its inputs connected to data1_in and data2_in signals. Below the schematic, the 'Cell Properties' window for the LUT4 shows its truth table configuration. The bottom 'Tcl Console' pane displays the synthesis log, indicating that the netlist sorting is complete.

I3	I2	I1	I0	O
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0

VHDL – Παράδειγμα

Βήμα 7b: Post Synthesis Timing Simulation



Περίληψη

- Πρώτη επαφή με το εργαλείο Vivado
 - Ανάπτυξη βήμα-βήμα μιας απλής εφαρμογής στο Vivado
 - RTL
 - Προσομοίωση
 - Synthesis
 - Lookup tables (LUT)
-
- Κατεβάζετε και κάνετε εγκατάσταση το Vivado 2022.2 . [Οδηγίες υπάρχουν στο eclass.](#)

4^ο Εργαστήριο Σχεδίασης Ψηφιακών Συστημάτων

1^ο Εργαστηριακό Μάθημα

Βασιλόπουλος Διονύσης

ΕΔΙΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

1^ο Εργαστηριακό μάθημα

Περιβάλλον Linux

USER=guest

PASSWORD=linux!

Για να εκτελεστεί το Vivado πρέπει να γράψετε τις ακόλουθες εντολές:
(υπάρχει και στο eclass, κάνετε copy/paste κάθε γραμμή χωριστά στο **terminal**)

```
source /opt/Xilinx/Vivado/2022.2/settings64.sh
```

```
cd $home
```

```
cd VIVADO-users/
```

```
mkdir sdi2400XXX
```

```
cd sdi2400XXX
```

```
vivado
```

← Όπου sdi2400XXX είναι ο ΑΜ σας

1^ο Εργαστηριακό μάθημα

Άσκηση

Σχεδιάστε μία αριθμητική και λογική μονάδα (ALU). Στην είσοδο δέχεται ένα σήμα a των 3 bit, καθώς και ένα σήμα $Ctrl$ ενός bit. Η ALU για τιμή $Ctrl = '0'$ κάνει διαίρεση ($a/2$: ακέραιο μέρος της διαίρεσης) ενώ για τιμή $Ctrl = '1'$ κάνει διπλασιασμό του a ($a*2$). Στην έξοδο υπάρχει το σήμα $Result$ των 3 bit με το αποτέλεσμα της πράξης και ένα σήμα $Carry$ που έχει τιμή '1' σε περίπτωση που υπάρχει κρατούμενο ($Carry$). Σχεδιάστε το κύκλωμα. Σας δίδεται ο ορισμός της οντότητας:

entity ALU is

Port (

a : in STD_LOGIC_VECTOR (2 downto 0);

Ctrl : in STD_LOGIC;

Result : out STD_LOGIC_VECTOR (2 downto 0);

Carry : out STD_LOGIC);

end entity ALU;

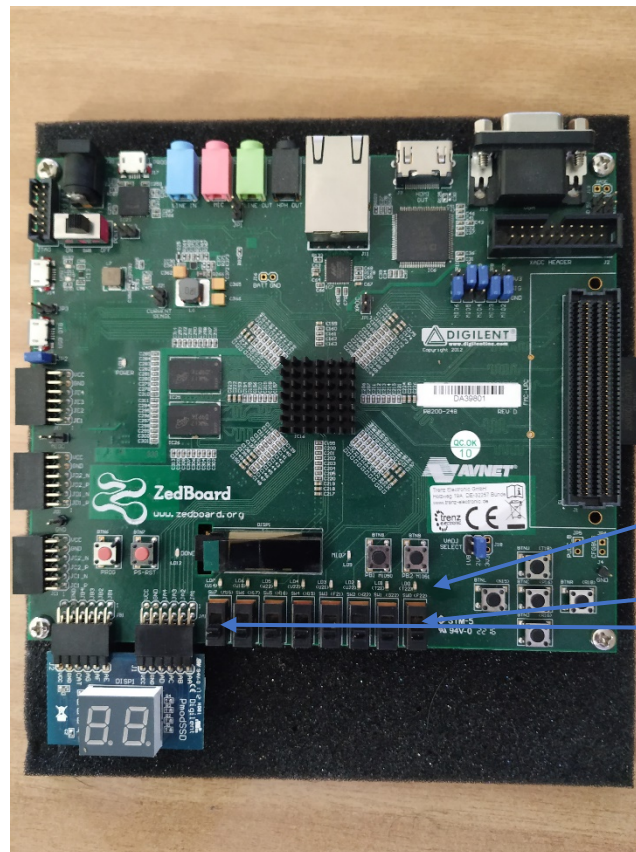
1^ο Εργαστηριακό μάθημα

Βήματα Επίλυσης

1. Εύρεση εισόδων/εξόδων συστήματος
2. Εύρεση πίνακα αληθείας για κάθε έξοδο του συστήματος (αν χρειάζεται)
3. Δημιουργία νέου project
4. Δημιουργία Entity – Εντοπισμός Input/Output του συστήματος
5. Δημιουργία Architecture – Θα έχετε τουλάχιστον τόσες εντολές όσες είναι και οι εξοδοί του συστήματος. Κάθε μία εντολή αντιστοιχεί σε μία έξοδο.
6. Δημιουργία RTL αναπαράστασης
7. Σύνθεση
8. Υλοποίηση
9. Προγραμματισμός κάρτας (Έγινε μόνο στο Εργαστήριο)
10. Προσομοίωση (Παρουσιάζεται μόνο στις διαφάνειες)

1^ο Εργαστηριακό μάθημα

Γνωριμία με την κάρτα

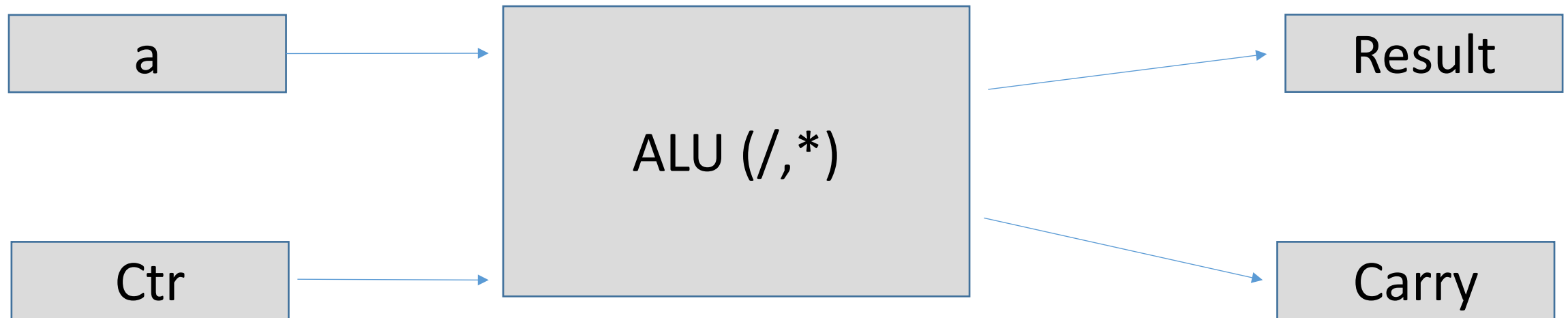


Led: ld0-ld2 (Result)
: ld7 (Carry)
SW: sw0-sw2 (a)
: sw7 (Ctr)

Το πραγματικό όνομα (pin) των led και switch είναι σε παρένθεση

1^ο Εργαστηριακό μάθημα

Απλοποιημένη μορφή – Είσοδοι/Εξοδοι



1^ο Εργαστηριακό μάθημα

Βήμα 2. Περιγραφή Οντότητας

```
entity ALU is
  Port (
    a          : in STD_LOGIC_VECTOR (2 downto 0);
    Ctr        : in STD_LOGIC;
    Result     : out STD_LOGIC_VECTOR (2 downto 0);
    Carry      : out STD_LOGIC
  );
end entity ALU;
```

1^ο Εργαστηριακό μάθημα

Συσχέτιση port με FPGA

Είσοδοι	DIP Switch
Ctr	SW7
A[2]	SW2
A[1]	SW1
A[0]	SW0

Έξοδοι	LED
Carry	LED7
Result[2]	LED2
Result[1]	LED1
Result[0]	LED0

1^ο Εργαστηριακό μάθημα

Συσχέτιση port με FPGA-2 (Αρχείο constraints: ZedBoard.xdc)

```
# ZedBoard Pin Assignments
#####
# On-board Slide Switches #
#####
set_property -dict { PACKAGE_PIN F22  IOSTANDARD LVCMOS33 } [get_ports { a[0] }];
set_property -dict { PACKAGE_PIN G22  IOSTANDARD LVCMOS33 } [get_ports { a[1] }];
set_property -dict { PACKAGE_PIN H22  IOSTANDARD LVCMOS33 } [get_ports { a[2] }];
set_property -dict { PACKAGE_PIN M15  IOSTANDARD LVCMOS33 } [get_ports { Ctr }];

#####
# On-board led          #
#####
set_property -dict { PACKAGE_PIN T22  IOSTANDARD LVCMOS33 } [get_ports { Result[0] }];
set_property -dict { PACKAGE_PIN T21  IOSTANDARD LVCMOS33 } [get_ports { Result[1] }];
set_property -dict { PACKAGE_PIN U22  IOSTANDARD LVCMOS33 } [get_ports { Result[2] }];
set_property -dict { PACKAGE_PIN U14  IOSTANDARD LVCMOS33 } [get_ports { Carry }];
```

ΠΡΟΣΟΧΗ στις **διαφορές** με τον Κώδικα VHDL

1. Τα **σχόλια** εδώ είναι με **#**
2. Τα bit του vector εδώ είναι με **[]** αντί **()**.
3. Τα ονόματα των σημάτων, πρέπει να είναι **ΑΚΡΙΒΩΣ** ίδια με τη δήλωση στην οντότητα (**case sensitive**)
4. Κάρτα Zynq 7(000) ZC702 – Evaluation Board

1^ο Εργαστηριακό μάθημα

Βήμα 3. Πίνακας Αληθείας κυκλώματος - Result

A2	A1	A0	Ctr	Result2	Result1	Result0	A2	A1	A0	Ctr	Result2	Result1	Result0
0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	0	1	1	0	1	0
0	1	0	0	0	0	1	0	1	0	1	1	0	0
0	1	1	0	0	0	1	0	1	1	1	1	1	0
1	0	0	0	0	1	0	1	0	0	1	0	0	0
1	0	1	0	0	1	0	1	0	1	1	0	1	0
1	1	0	0	0	1	1	1	1	0	1	1	0	0
1	1	1	0	0	1	1	1	1	1	1	1	1	0

1^ο Εργαστηριακό μάθημα

Βήμα 3. Πίνακας Αληθείας κυκλώματος - Carry

A2	A1	A0	Ctr	Carry
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

A2	A1	A0	Ctr	Carry
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

1^ο Εργαστηριακό μάθημα

Βήμα 4. Περιγραφή Αρχιτεκτονικής

Result(2) <= Ctr and a(1);

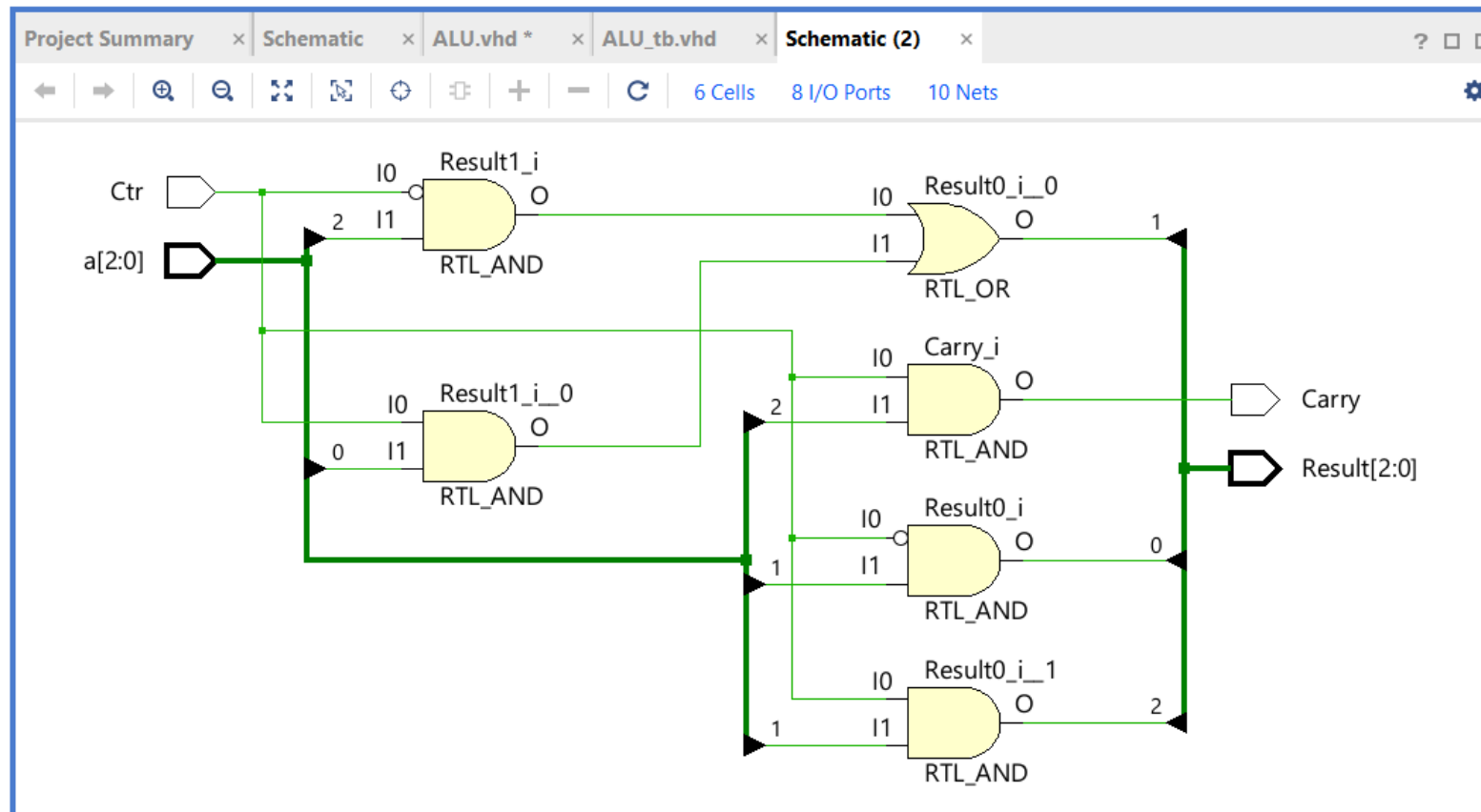
Result(0) <= not Ctr and a(1);

Result(1) <= (not Ctr and a(2)) or (Ctr and a(0));

Carry <= Ctr and a(2);

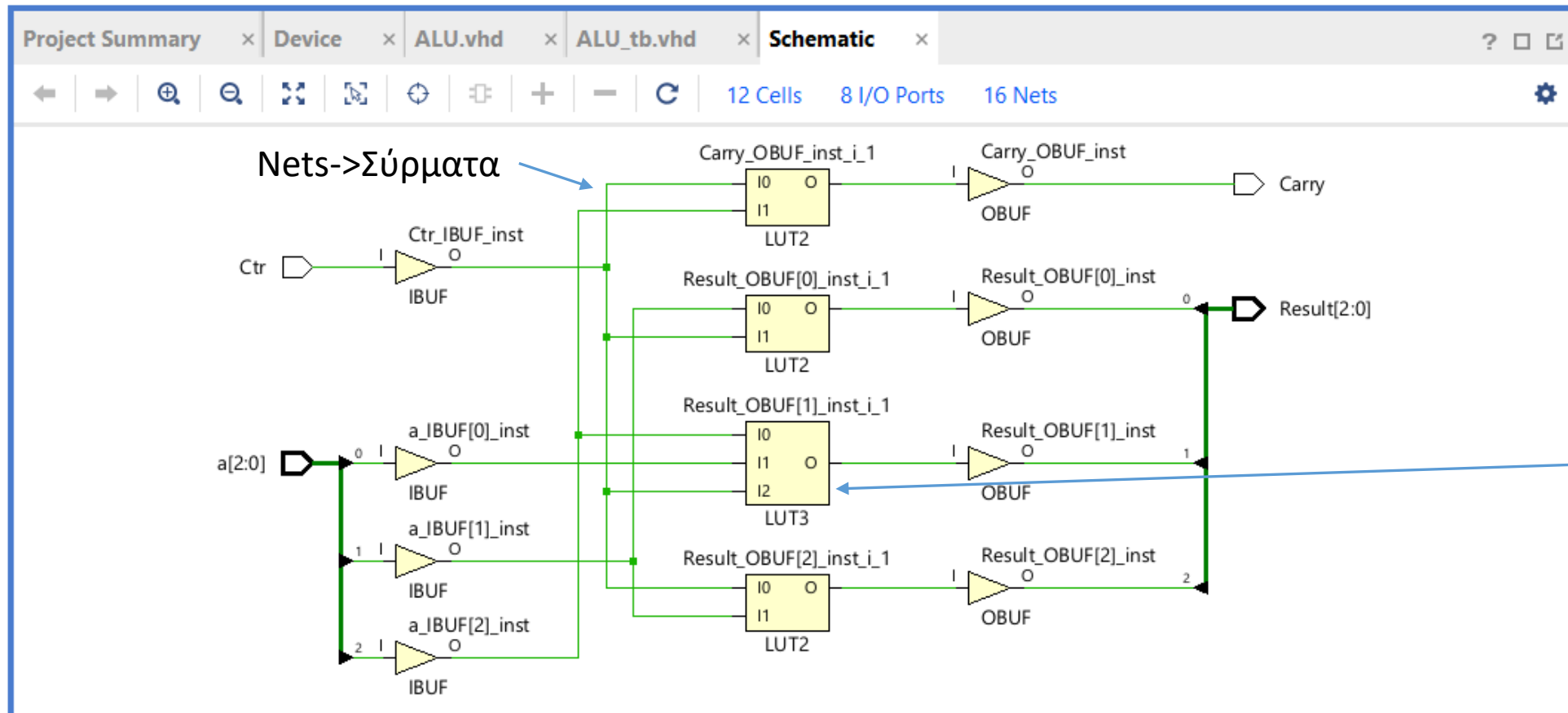
1^ο Εργαστηριακό μάθημα

Βήμα 5. Λογικό κύκλωμα: Αναπαράσταση RTL (α)



1^ο Εργαστηριακό μάθημα

Βήμα 6. Λογικό κύκλωμα: Φάση Σύνθεσης – LUT(LookUp Table), υλοποιούν Πίνακες Αληθείας (προγραμματιζόμενα μέρη της κάρτας FPGA)

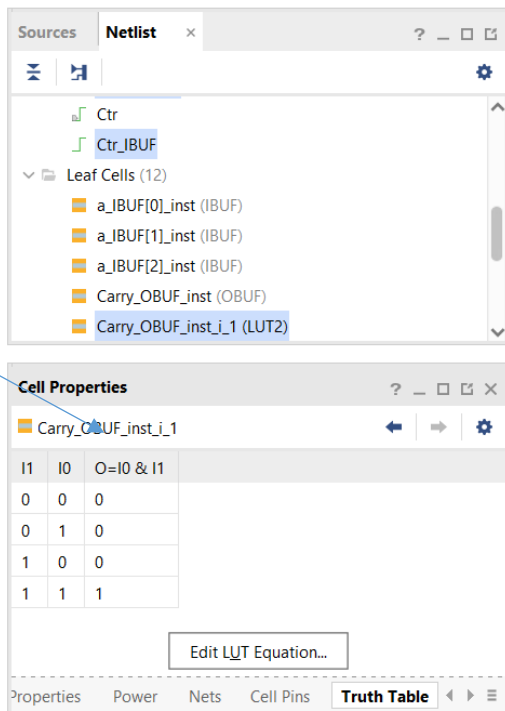
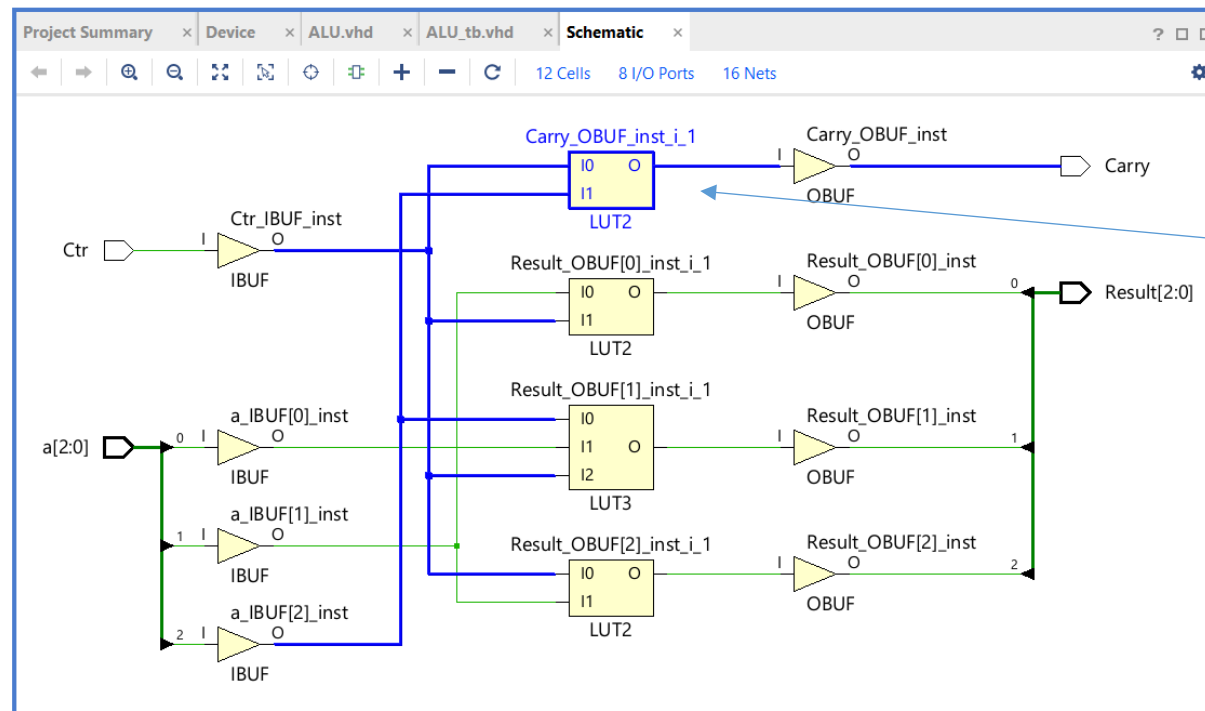


1^ο Εργαστηριακό μάθημα

Βήμα 7α. Λογικό κύκλωμα: Φάση Υλοποίησης

Συνάρτηση
LUT

Πίνακας Αληθείας



Στο LUT είσοδοι
είναι a(2), Ctr και
υπολογίζεται το
ψηφίο Carry.

Ctr => I0
a(2) => I1

1^ο Εργαστηριακό μάθημα

Βήμα 7b. Λογικό κύκλωμα: Φάση Υλοποίησης – Modified Truth Table (αντιστοιχίες $a(0)$, $b(0)$, $Ctrl \Leftrightarrow I$ του LUT

Πίνακας Αληθείας
Truth Table
για Carry



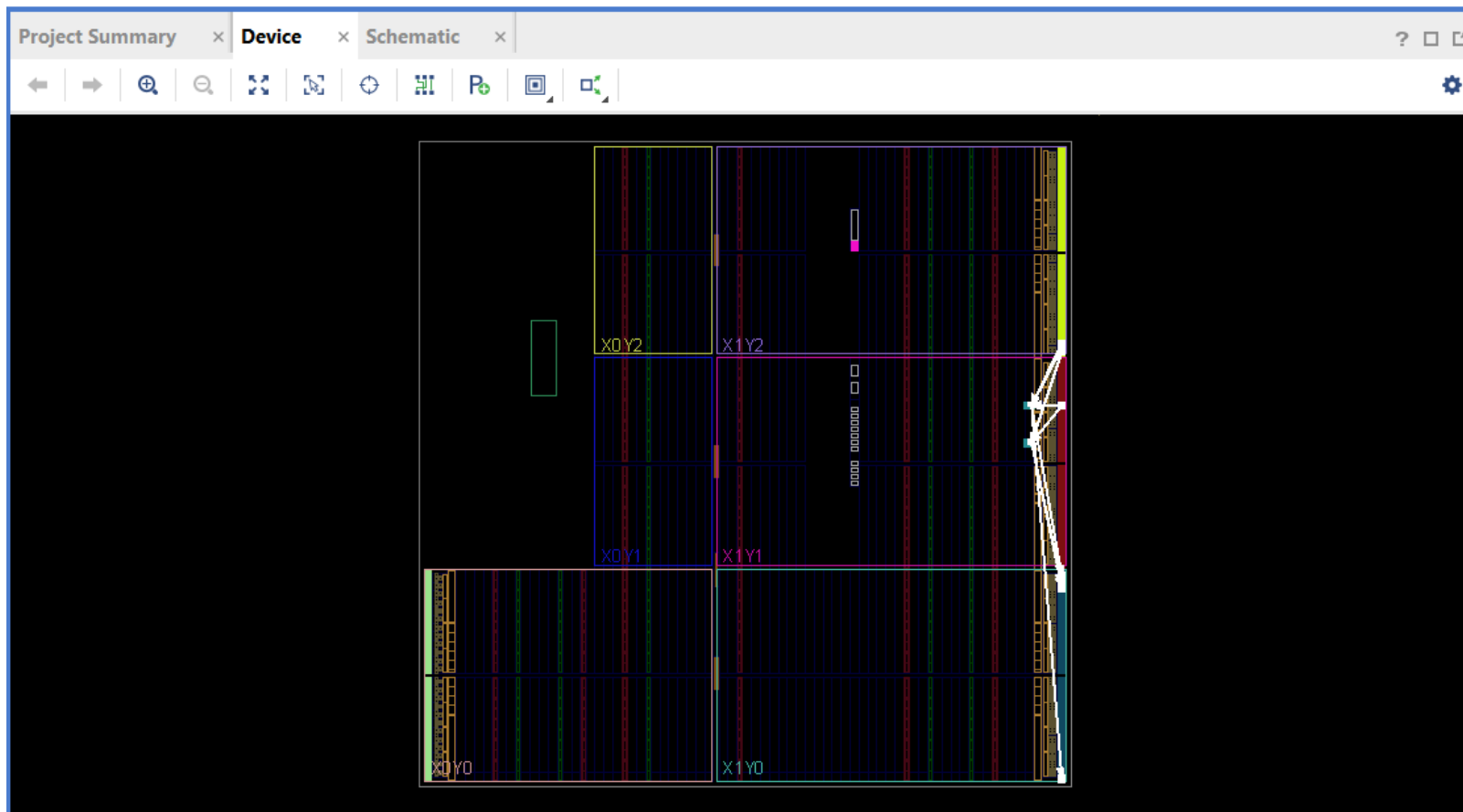
Είσοδοι		Έξοδοι
Ctrl/ I_0	$a(2)/I_1$	Carry
0	0	0
0	1	0
1	0	0
1	1	1

Ο αρχικός Πίνακας Αληθείας και ο αντίστοιχος του LUT είναι ίδιοι

$Carry \leq (a(2) \text{ and } Ctrl)$
 $O = I_0 \text{ \& } I_1$

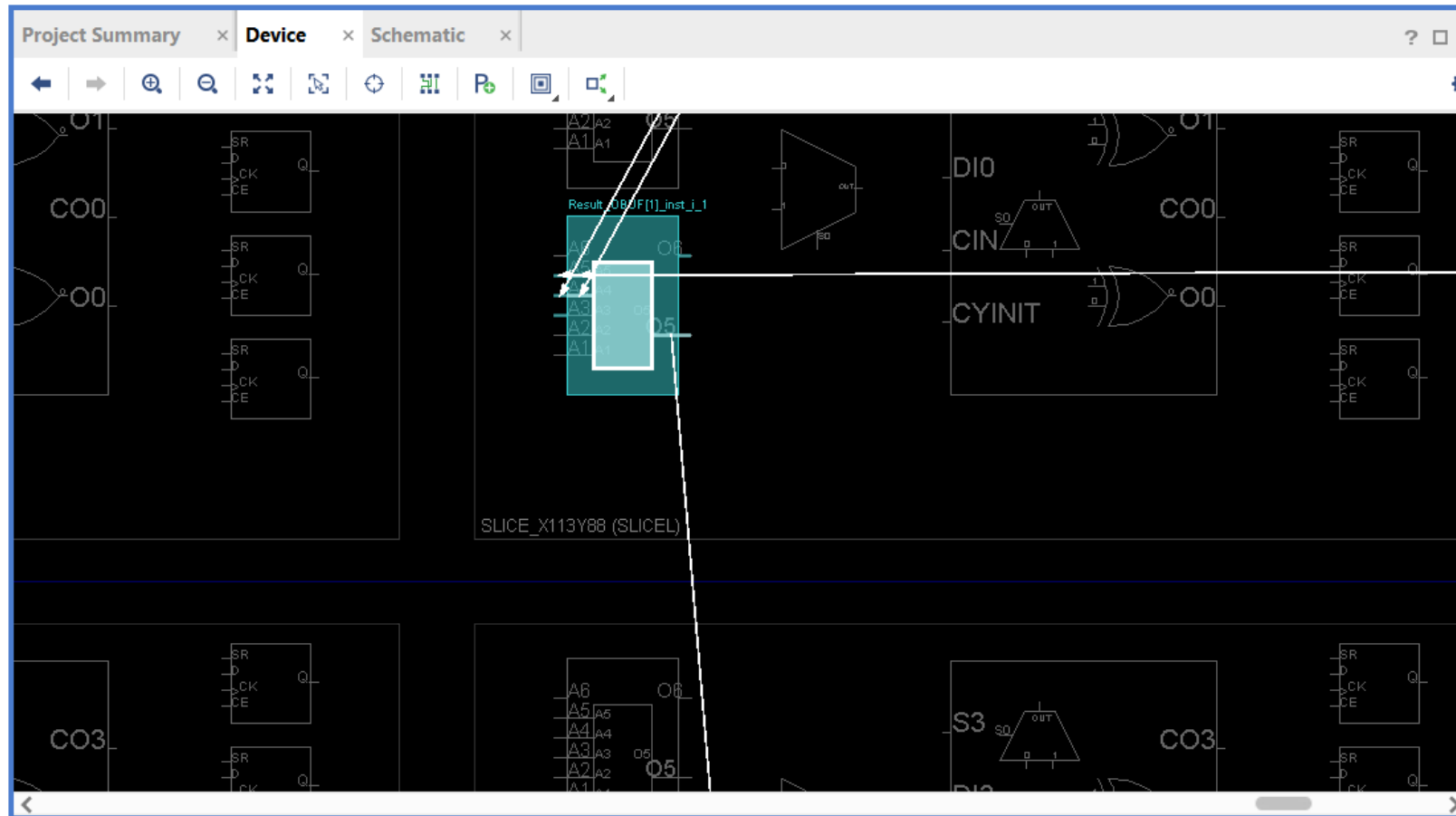
1^ο Εργαστηριακό μάθημα

Βήμα 7c. Λογικό κύκλωμα: Φάση Υλοποίησης – Design



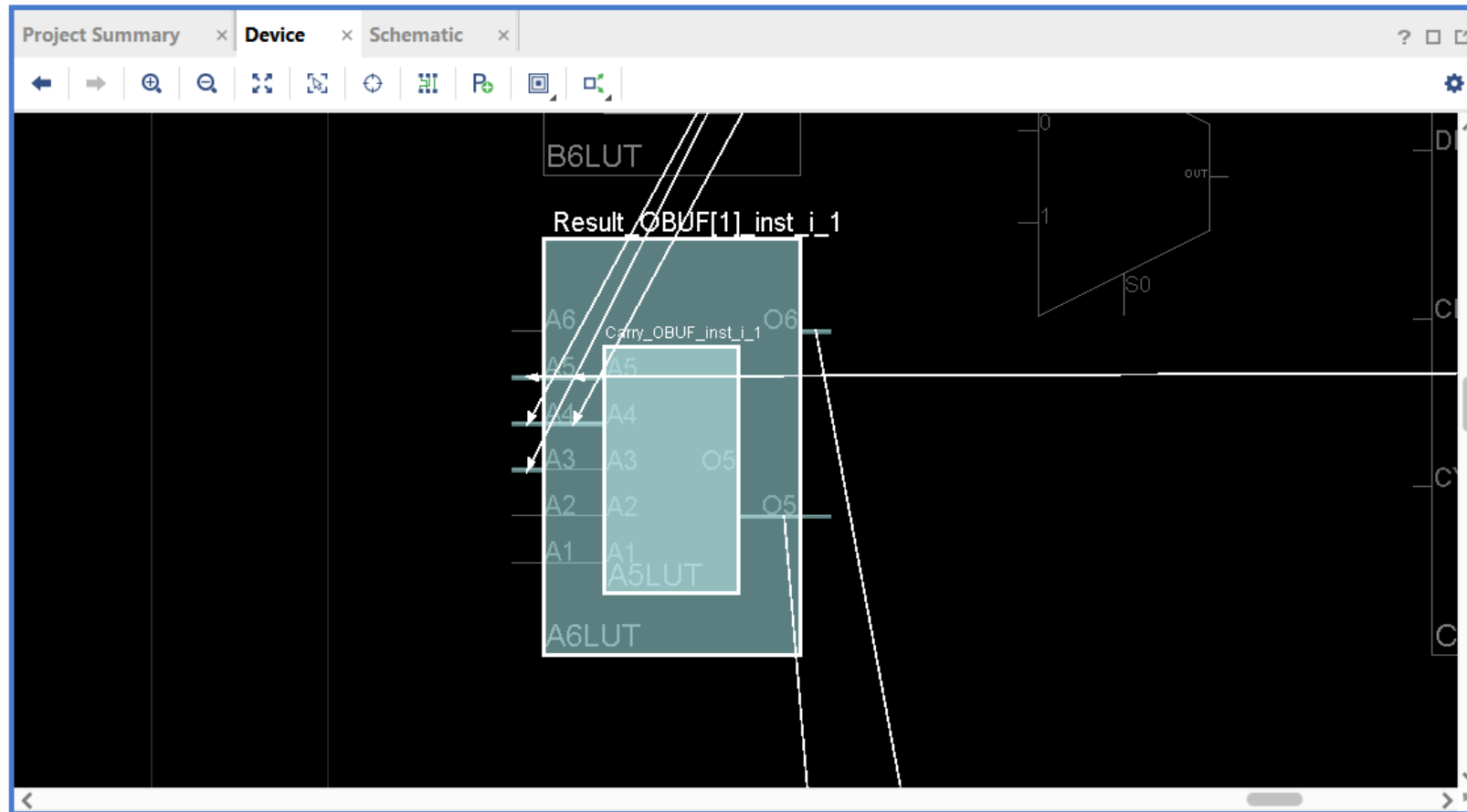
1^ο Εργαστηριακό μάθημα

Βήμα 7d. Λογικό κύκλωμα: Φάση Υλοποίησης – Design: LUT on Board (LUT2 – Carry)



1^ο Εργαστηριακό μάθημα

Βήμα 7d. Λογικό κύκλωμα: Φάση Υλοποίησης – Design: LUT on Board (LUT2 – Carry+Result bit)



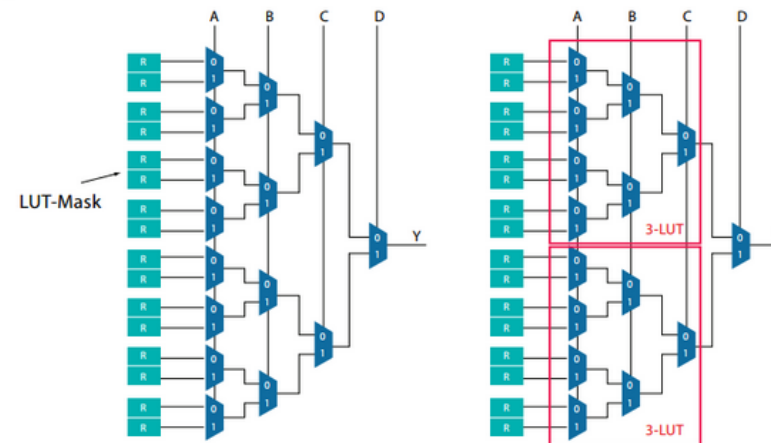
1^ο Εργαστηριακό μάθημα

LUT

Building Look-up Tables (LUTs)

An overview of how LUTs are built helps describe the key innovations in the ALM. A LUT is typically built out of SRAM bits to hold the configuration memory (CRAM) LUT-mask and a set of multiplexers to select the bit of CRAM that is to drive the output. To implement a k-input LUT (k-LUT)—a LUT that can implement any function of k inputs— 2^k SRAM bits and a $2^k:1$ multiplexer are needed. Figure 2 shows a 4-LUT, which consists of 16 bits of SRAM and a 16:1 multiplexer implemented as a tree of 2:1 multiplexers. The 4-LUT can implement any function of 4 inputs (A, B, C, D) by setting the appropriate value in the LUT-mask. To simplify the 4-LUT in Figure 2, it can also be built from two 3-LUTs connected by a 2:1 multiplexer.

Figure 2. Building a LUT



$$a'b'c'd' + abcd + abc'd' = 1000\ 0000\ 0000\ 1001 = 0x8009$$

Share Cite Follow

answered May 8, 2015 at 8:04

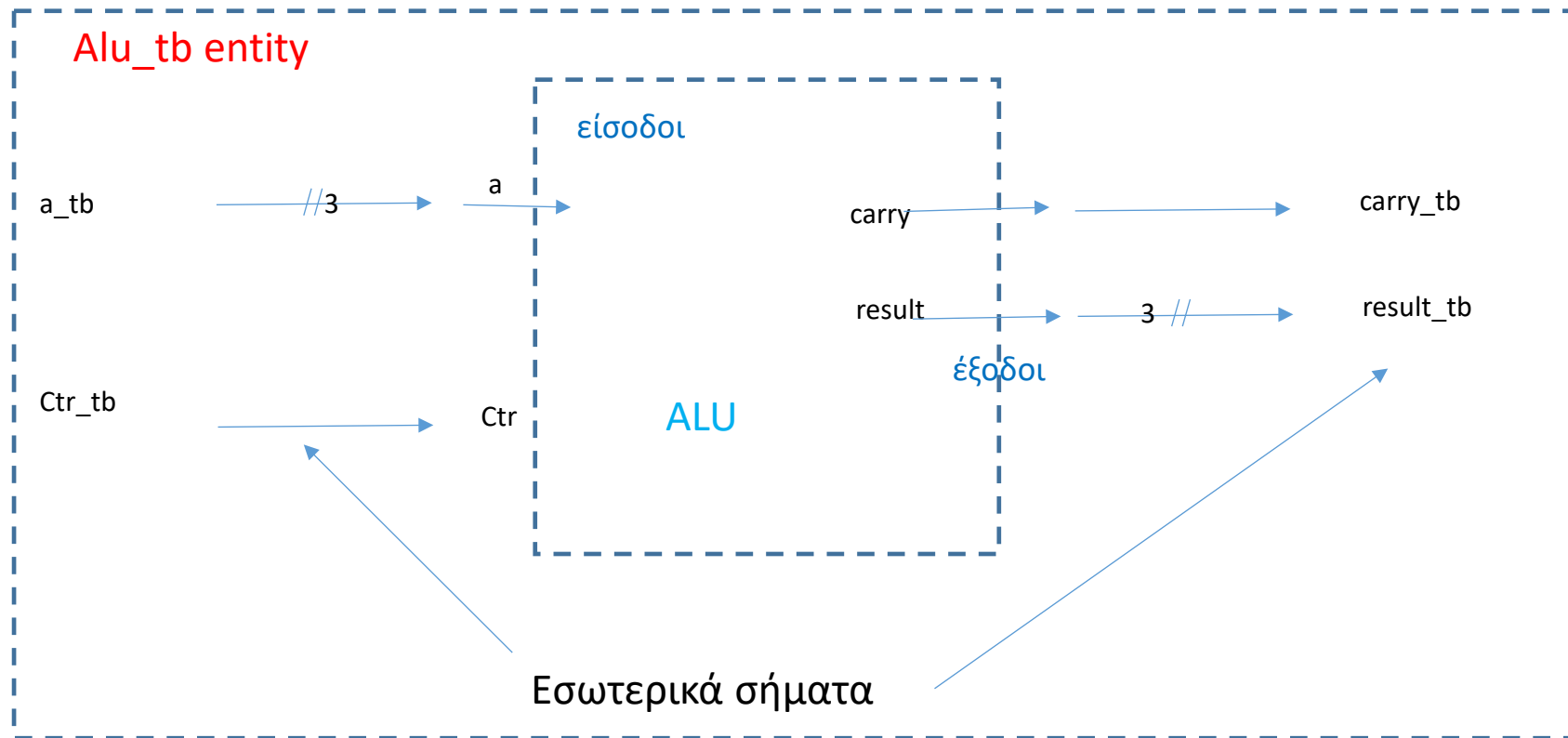
apalorohapa
8,329 ● 2 ● 28 ● 39

Add a comment

Υλοποίηση LUT

1^ο Εργαστηριακό μάθημα

Simulation



1^ο Εργαστηριακό μάθημα

Simulation - κώδικας

```
entity ALU_tb is
-- Port ( );
end ALU_tb;

architecture Behavioral of ALU_tb is
component ALU is
    Port ( a      : in STD_LOGIC_VECTOR (2 downto 0);
          Ctr     : in STD_LOGIC;
          Result  : out STD_LOGIC_VECTOR (2 downto 0);
          Carry   : out STD_LOGIC
        );
end component ALU;

signal a_tb      : STD_LOGIC_VECTOR (2 downto 0);
signal Ctr_tb    : STD_LOGIC;
signal Result_tb : STD_LOGIC_VECTOR (2 downto 0);
signal Carry_tb  : STD_LOGIC;

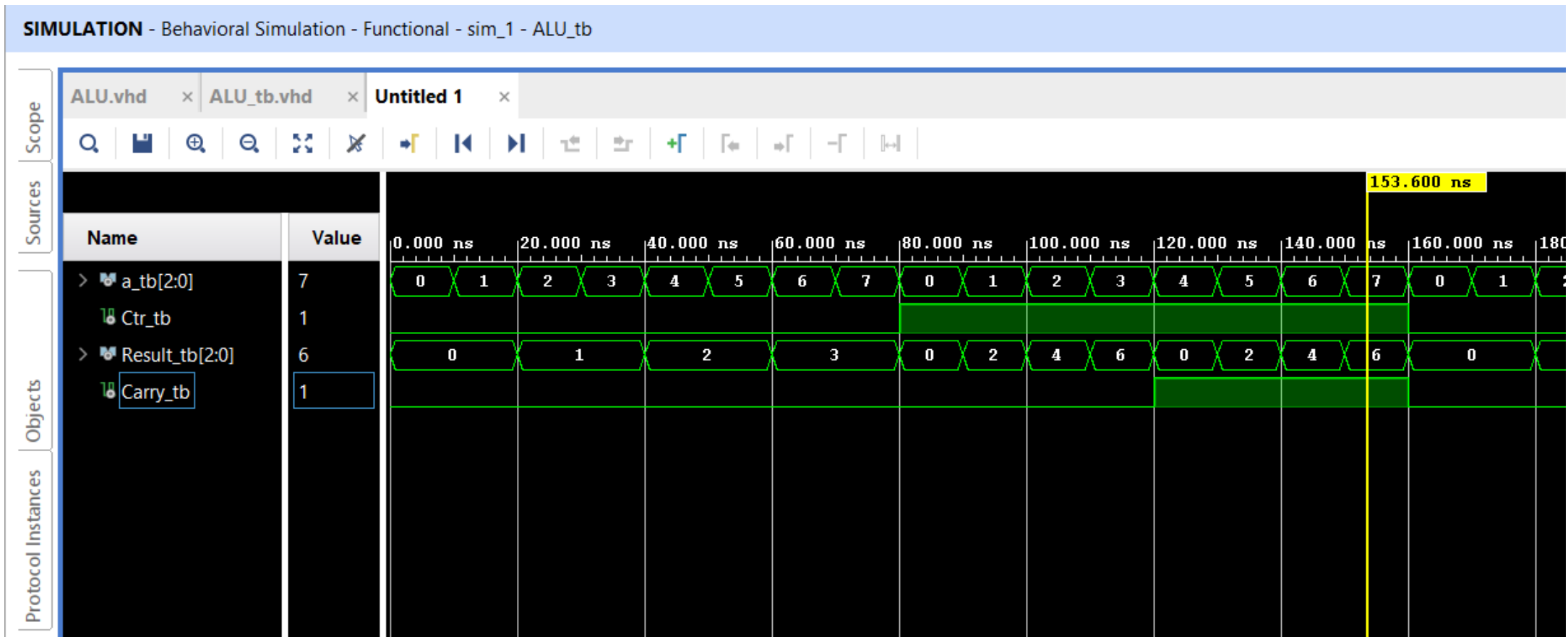
Begin
uut: ALU port map (a_tb,Ctr_tb,Result_tb, Carry_tb);
```

```
test: process is
begin
    Ctr_tb<='0';
    a_tb<="000";wait for 10ns;
    a_tb<="001";wait for 10ns;
    a_tb<="010";wait for 10ns;
    a_tb<="011";wait for 10ns;
    a_tb<="100";wait for 10ns;
    a_tb<="101";wait for 10ns;
    a_tb<="110";wait for 10ns;
    a_tb<="111";wait for 10ns;
    Ctr_tb<='1';
    a_tb<="000";wait for 10ns;
    a_tb<="001";wait for 10ns;
    a_tb<="010";wait for 10ns;
    a_tb<="011";wait for 10ns;
    a_tb<="100";wait for 10ns;
    a_tb<="101";wait for 10ns;
    a_tb<="110";wait for 10ns;
    a_tb<="111";wait for 10ns;
end process test;
```

```
end architecture Behavioral;
```

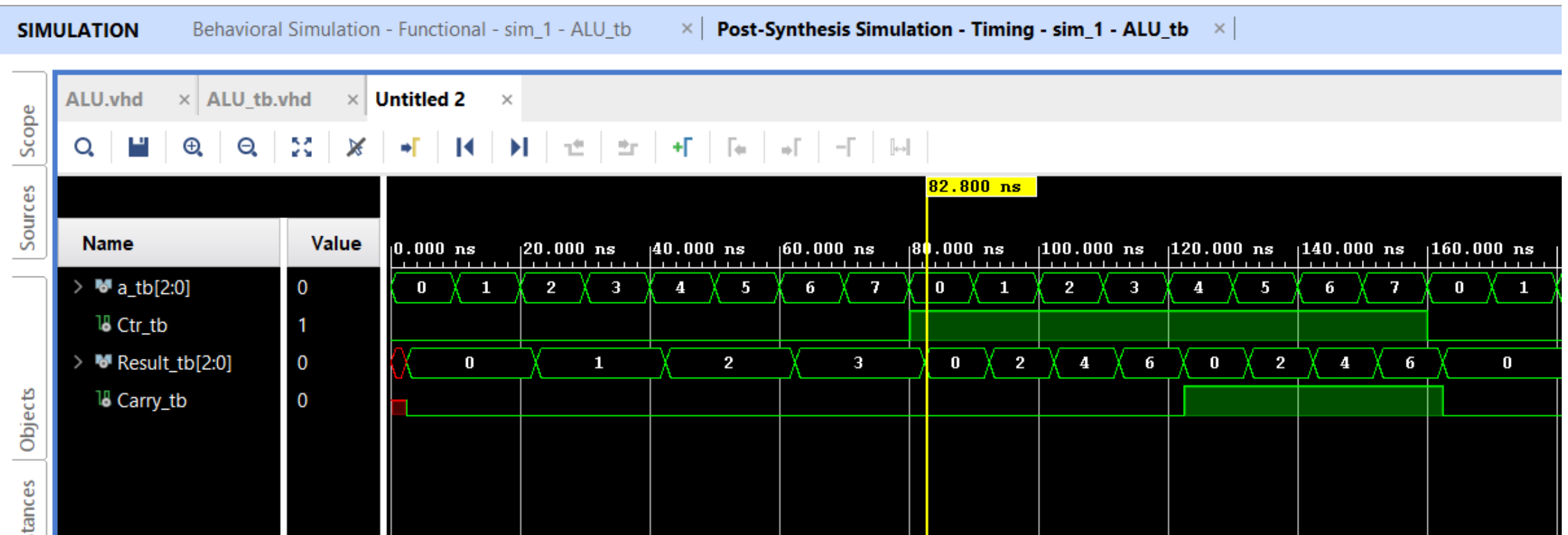
1^ο Εργαστηριακό μάθημα

Simulation – Χρονοσειρά RTL



1^ο Εργαστηριακό μάθημα

Simulation – Χρονοσειρά post Synthesis



1^ο Εργαστηριακό μάθημα

Περίληψη

- Ανάπτυξη βήμα-βήμα μιας απλής εφαρμογής στο Vivado
- RTL->Synthesis->Implementation
- Προσομοίωση
- LUT

Υλοποίηση LUT

<https://electronics.stackexchange.com/questions/169532/what-is-an-lut-in-fpga>

<https://hardwarebee.com/overview-of-lookup-tables-in-fpga-design/>

https://www.xilinx.com/htmldocs/xilinx2017_4/sdaccel_doc/yeo1504034293627.html