



**dscal**  
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

# Εργαστήριο Λογικής Σχεδίασης

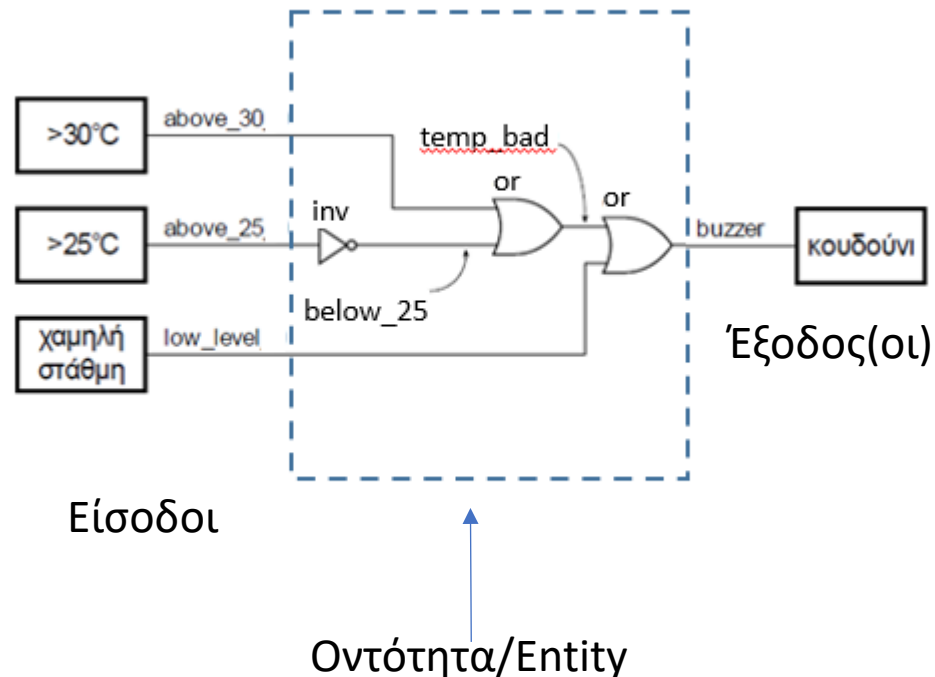
3<sup>η</sup> Διάλεξη

**Εισαγωγή στη VHDL**

**Βασιλόπουλος Διονύσης**

Ε.ΔΙ.Π Τμήματος Πληροφορικής & Τηλεπικοινωνιών

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



library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;

entity buzzer is

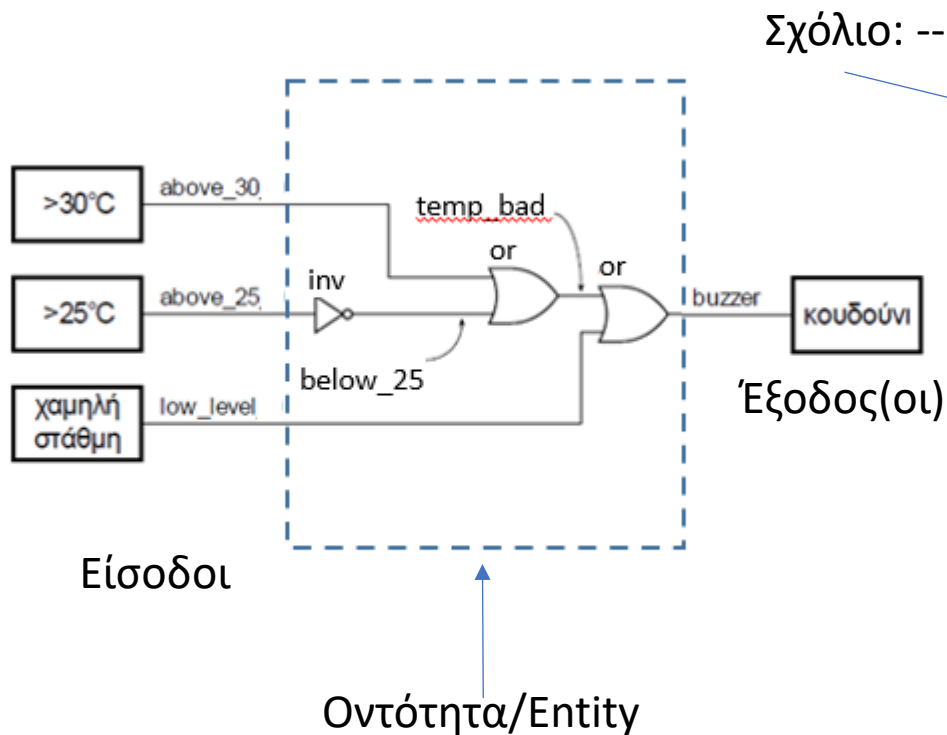
**port** (  
above\_25\_0: **in** std\_logic;  
above\_30\_0: **in** std\_logic;  
low\_level\_0: **in** std\_logic;  
buzzer : **out** std\_logic );

end entity buzzer;

Υποχρεωτικές Βιβλιοθήκες

Περιγραφή Οντότητας

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



Σχόλιο: --

Χωρίς τις Βιβλιοθήκες

Περιγραφή Οντότητας

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

```
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;
```

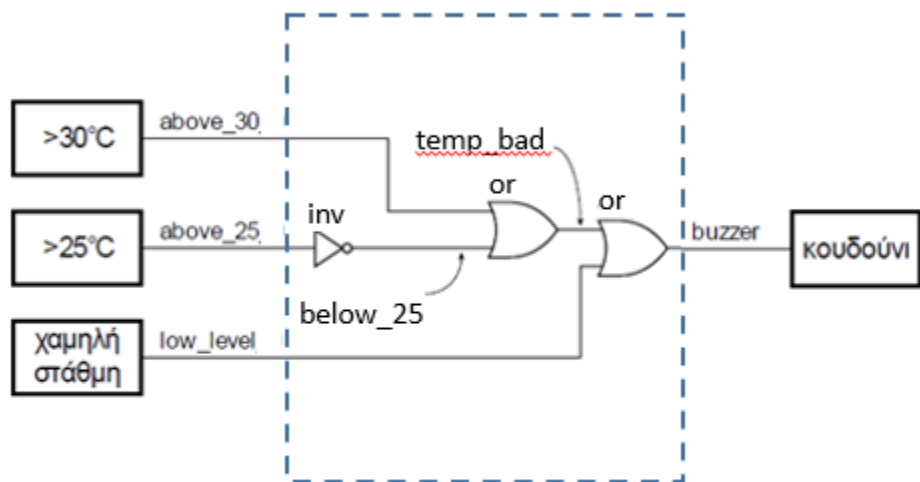
## Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Entity: Input/Output

- **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**

## Ψηφιακό κύκλωμα – VHDL: Architecture (Dataflow)



architecture Dataflow of buzzer is

begin

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

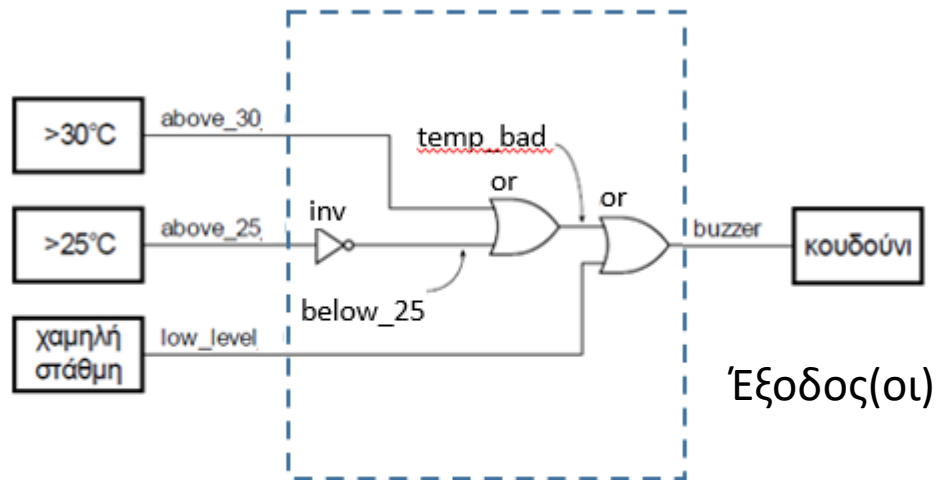
end architecture buzzer;

### Αρχιτεκτονική

Περιγραφή της λειτουργικότητας που προσφέρει το λογικό κύκλωμα

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

## Ψηφιακό κύκλωμα – VHDL: Entity (Εσωτερικά σήματα)



Εσωτερικά σήματα

temp\_bad

-

below\_25

Είσοδοι

Οντότητα/Entity



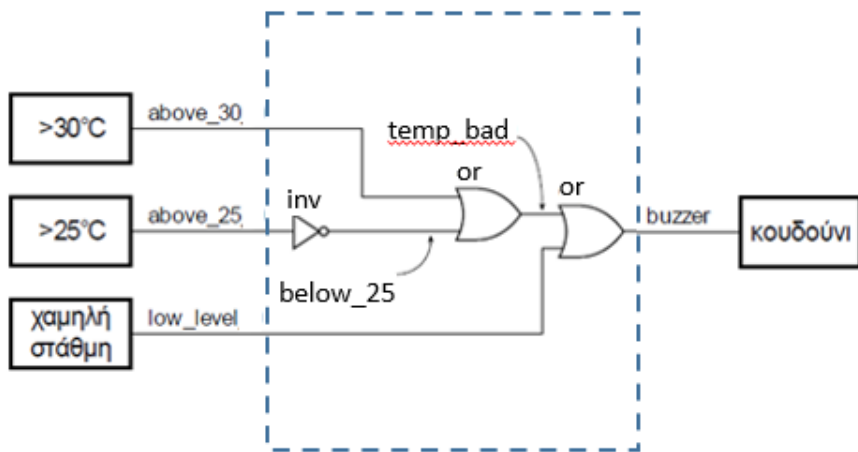
## Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Architecture

```
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  
  concurrent_component_statement;  
  ...  
  concurrent_component_statement;  
end architecture arch_name;
```

## Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Architecture

- **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.
- **signal\_name**: το όνομα του σήματος (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
  - στις δηλώσεις σημάτων (μετά το **is**) 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)

## Ψηφιακό κύκλωμα – VHDL: Architecture (Dataflow)



architecture Dataflow of buzzer is

```
signal temp_bad, below_25: std_logic;
```

```
begin
```

```
below_25<=not above_25;
```

```
temp_bad<=above_30 or below_25;
```

```
buzzer<= temp_bad or low_level;
```

```
end architecture buzzer;
```

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

**concurrent statements**

**Τα εσωτερικά σήματα μπορούν να είναι και αριστερά και δεξιά**

## Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Architecture

- Ταυτόχρονες εντολές ανάθεσης σήματος (concurrent\_signal\_assignment\_statements)

```
signal_name <= expression;
```

- **expression**: έκφραση με σήματα και τελεστές
- **signal\_name**: το όνομα του σήματος
  - στις ταυτόχρονες εντολές ανάθεσης σήματος :
    - στην **έκφραση (expression)** προσδιορίζονται σήματα που είναι **είσοδοι** στην οντότητα και δηλώνονται κατά τη δήλωση των διαύλων της οντότητας, και **εσωτερικές διασυνδέσεις (εσωτερικά σήματα)** που δηλώνονται κατά τη δήλωση σημάτων
    - στο αριστερό μέρος της εντολής προσδιορίζεται σήμα που είναι **έξοδος** της οντότητας και δηλώνεται κατά τη δήλωση των διαύλων της οντότητας, ή **εσωτερική διασύνδεση (εσωτερικό σήμα)** που δηλώνεται κατά τη δήλωση σημάτων

## Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Architecture

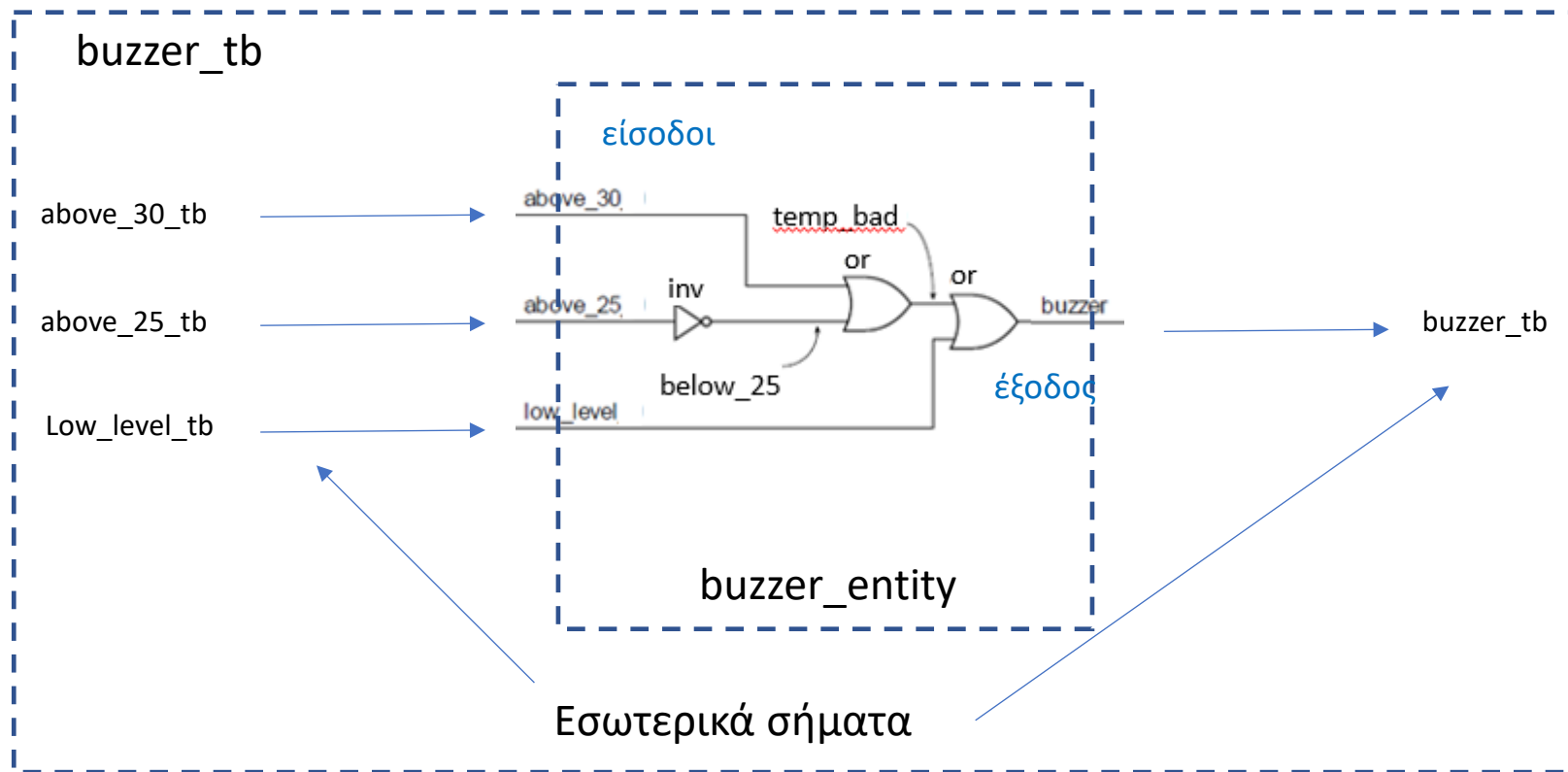
- **Εκτέλεση** ταυτόχρονων εντολών ανάθεσης σήματος (concurrent\_signal\_assignment\_statements)

```
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  $\delta_{\text{delay}}$** , που δεν επηρεάζει τον χρονισμό του κυκλώματος
- Η πραγματική καθυστέρηση διάδοσης θα προσδιορισθεί με την υλοποίηση σε μία συγκεκριμένη τεχνολογία

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

## 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

## Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Testbench

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

entity buzzer_tb is
end buzzer_tb ;

architecture Beh_tb of buzzer_tb is

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

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 Beh_tb;
```

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



# VHDL - Vivado

## Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Testbench

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

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

```
architecture Beh_tb of buzzer_tb is
```

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

```
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 Beh_tb;
```

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

## Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Αρχιτεκτονική – Components

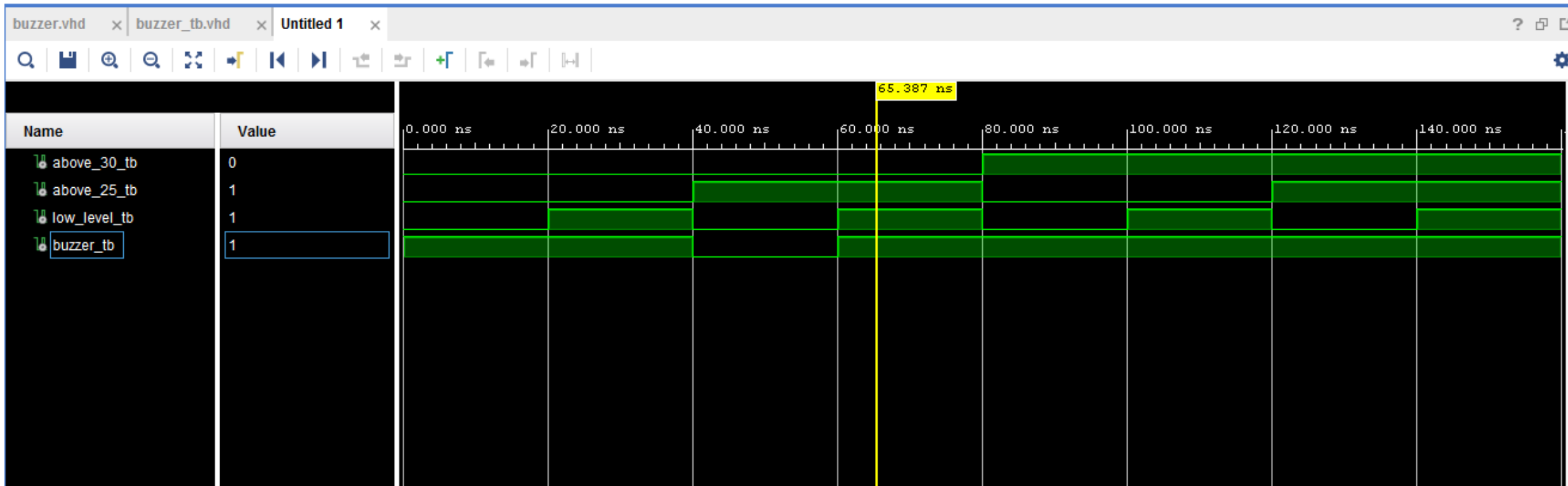
- **Ταυτόχρονες εντολές στοιχείων** (concurrent\_component\_statements)

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

- **label**: οι μοναδικές ετικέτες των στοιχείων
- **comp\_name**: το όνομα του στοιχείου (υποκύκλωμα) που χρησιμοποιείται στην αρχιτεκτονική της οντότητας
- **port\_name**: το όνομα του port της οντότητας (component) το οποίο καλούμε.
- **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**)

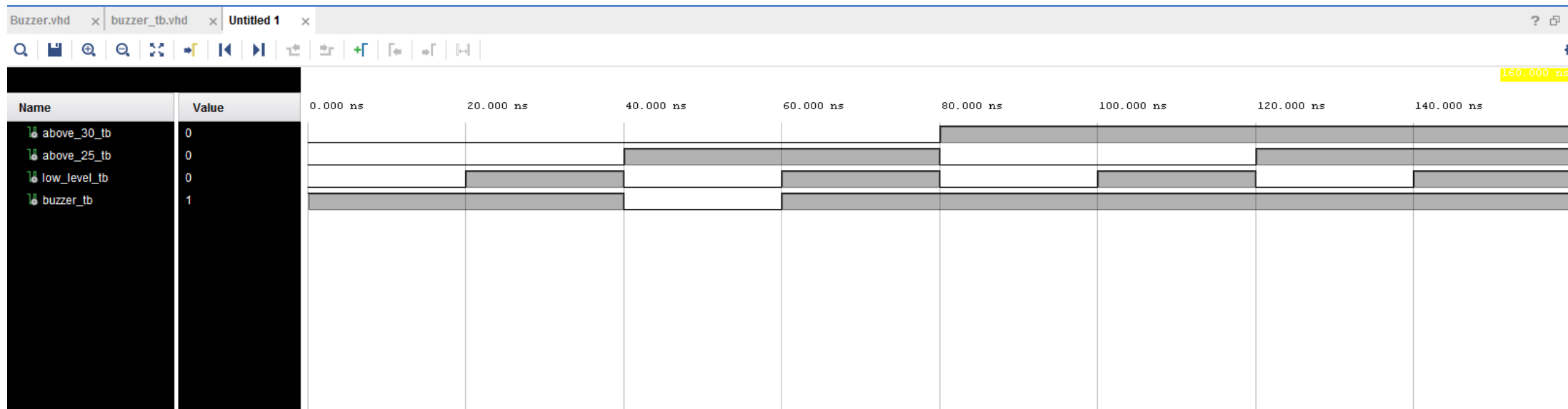
# VHDL

## Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Χρονοσειρά


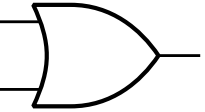
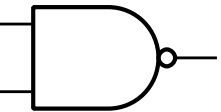
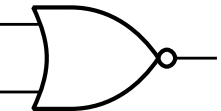
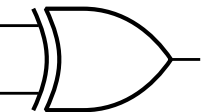
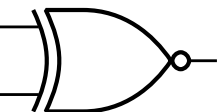
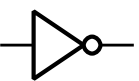


# VHDL - Vivado

## Ψηφιακό κύκλωμα – VHDL: Προσομοίωση - Χρονοσειρά



## Λογικοί Τελεστές

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

- 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'

# VHDL – Τύποι σημάτων

## Std\_logic

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

# VHDL – Τύποι σημάτων

## Std\_logic\_vector

```
signal_0: out std_logic;  
signal_1: out std_logic;  
signal_2: out std_logic;  
signal_3: out std_logic;
```

Εναλλακτικά

```
signal_all : out std_logic_vector (3 downto 0);
```

---

```
signal_all <="0101";
```

```
signal_all(0) <='1'  
signal_all(1) <='0'  
signal_all(2) <='1'  
signal_all(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 **θετικοί ακέραιοι ή μηδέν**
- 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:
  - `If V = "00000000"` για σύγκριση **ολόκληρου** του διανύσματος
  - `If V(3 downto 0) = "0000"` για σύγκριση **μέρους** του διανύσματος

**Προσοχή. Σε όλα τα προγράμματα τα PORT στον ορισμό της Οντότητας θα είναι MONO STD\_LOGIC ή STD\_LOGIC\_VECTOR**

# Εγγραφές σε Τμήματα Εργαστηρίων

- Στο διάστημα 23/10 – 1/11 θα γίνει το 1ο ΕΡΓΑΣΤΗΡΙΟ (και **ΔΕΝ** θα γίνει διάλεξη στις 29/10 στην Α2).
- Στο διάστημα 15/10 – 18/10 θα επιλέξετε το εργαστηριακό τμήμα που επιθυμείτε να παρακολουθήσετε
- Το εργαστήριο είναι στο αναγνωστήριο.
- Μπορείτε να **αλλάξετε τμήμα** εάν υπάρχουν θέσεις (απεγγραφή από παλιό, εγγραφή στο νέο). Αυτό μπορεί να γίνει έως και 18/10.
- Σε περίπτωση που χρειαστεί θα ανοίξουν και επιπλέον τμήματα. Θα υπάρξουν θέσεις για όλους.
- Όσοι φοιτητές δεν έχουν κάνει ακόμα εγγραφή στο τμήμα ή απλά δεν έχουν κωδικούς για το eclass, θα μου στείλουν email δηλώνοντας **τουλάχιστον δύο τμήματα** με διαθέσιμες θέσεις.
- Οι μέρες και ώρες που επιλέγετε για το 1ο εργαστήριο ΔΕΝ σας δεσμεύουν για το 2ο εργαστήριο (όταν αυτό ανακοινωθεί). Μπορείτε, τότε, να επιλέξετε άλλη μέρα και ώρα.

# Περίληψη

- Παράδειγμα ανάπτυξης εφαρμογής σε VHDL
- Δηλώσεις Οντότητας (entity), Αρχιτεκτονικής (architecture).
- Ports και Εσωτερικά σήματα
- Ταυτόχρονες εντολές
- Components
- Προτεραιότητες πράξεων
- Τύποι σημάτων (std\_logic, std\_logic\_vector)
- Διαβάζετε τις παραγράφους 3.2 από Ashenden και 2.1, 2.6, 4.1, 4.2.1, 4.2.2 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.
- Κατεβάζετε και κάνετε εγκατάσταση το Vivado