

2ο Εργαστήριο Σχεδίασης Ψηφιακών Συστημάτων

2^η Διάλεξη

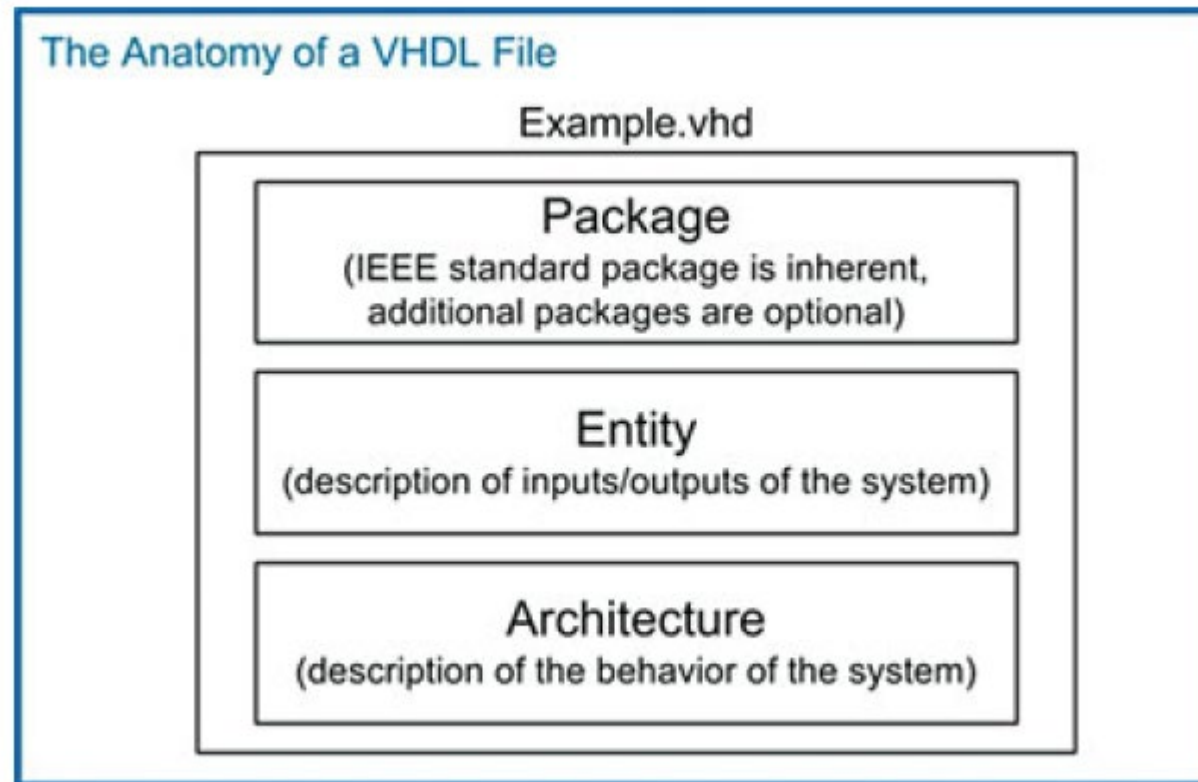
Ορισμός Οντότητας – Ορισμός/Είδη Αρχιτεκτονικής

Βασιλόπουλος Διονύσης

Ε.ΔΙ.Π. Τμήματος Πληροφορικής & Τηλεπικοινωνιών

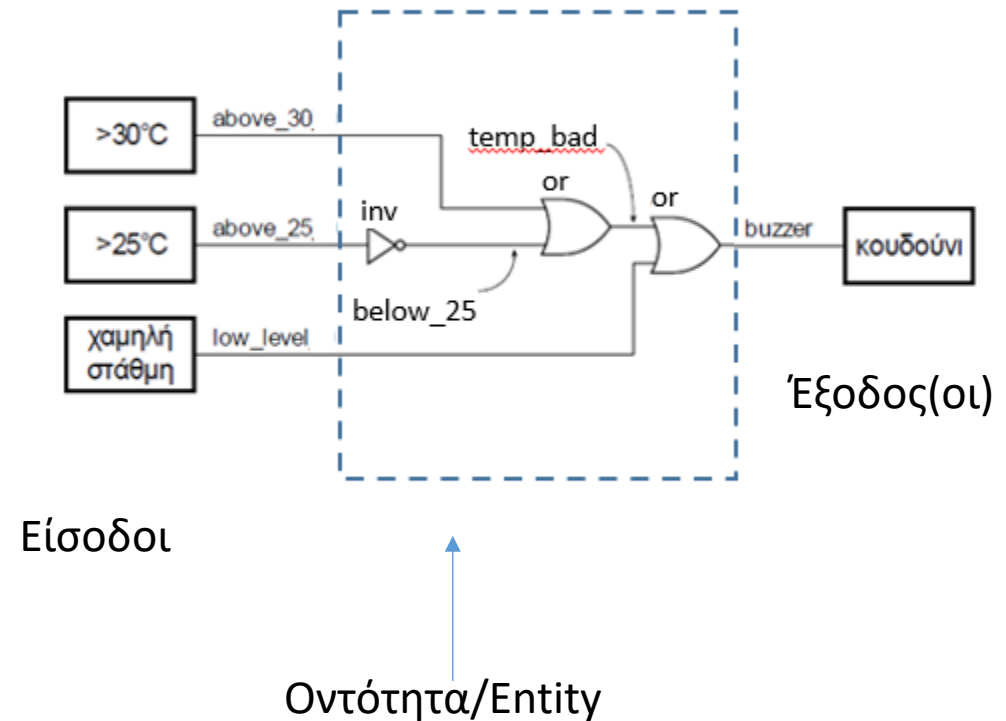
VHDL - Vivado

Ψηφιακό κύκλωμα – VHDL: Δομή Αρχείου

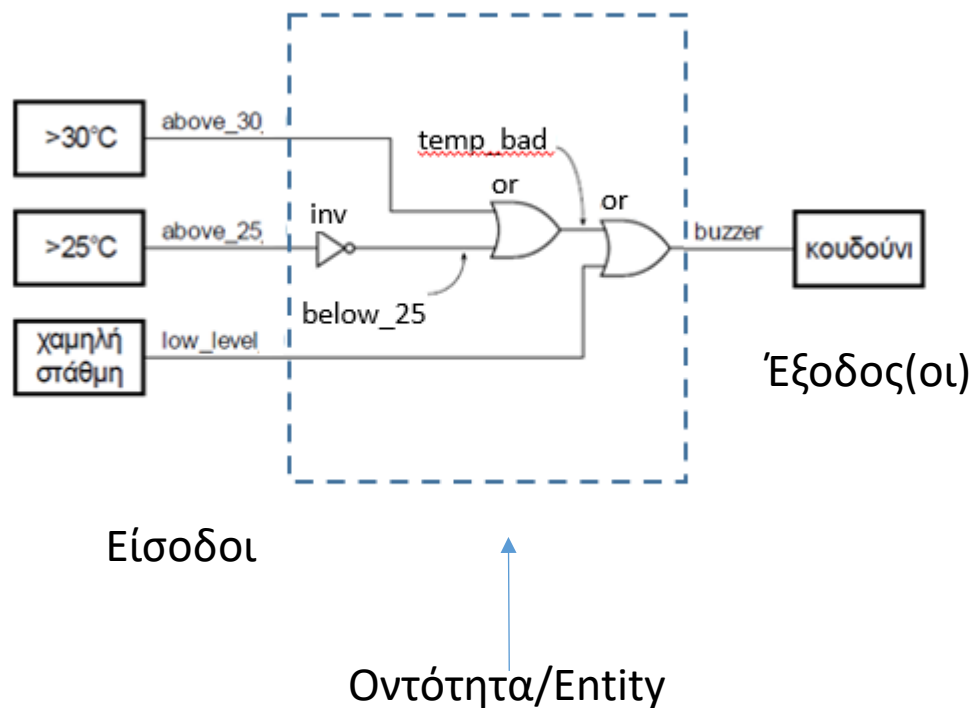


Ψηφιακό κύκλωμα - Παράδειγμα

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

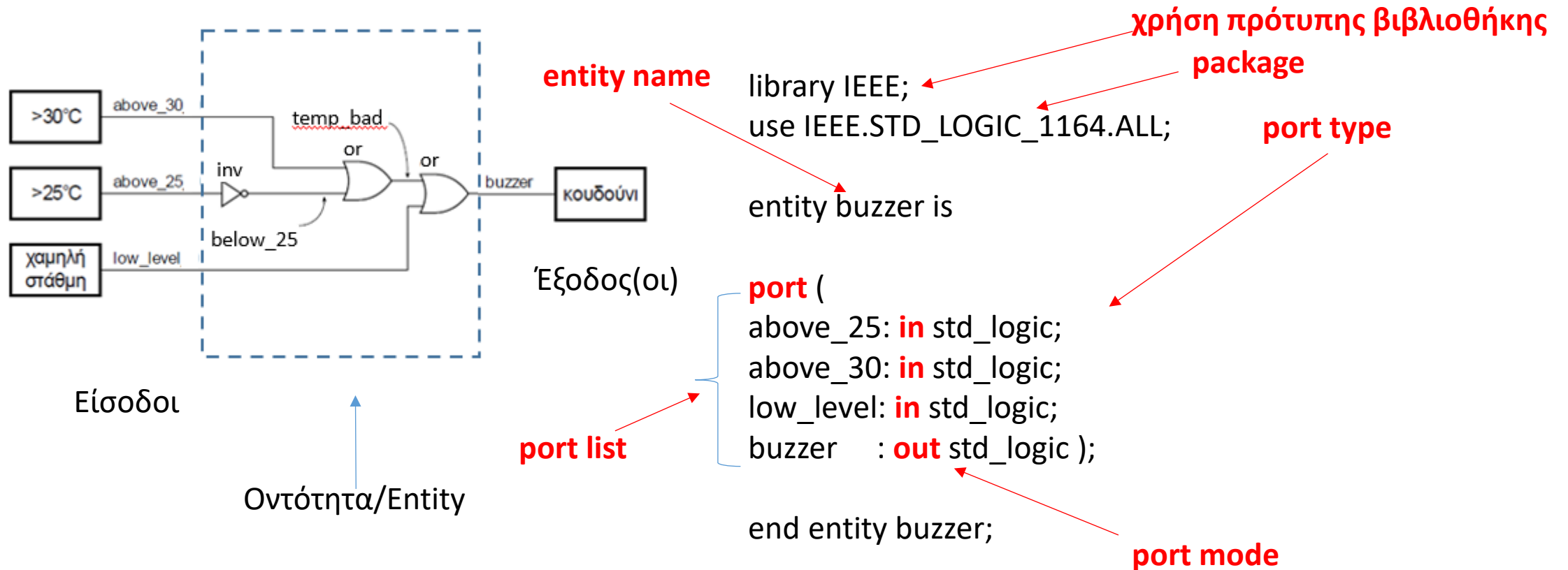
entity buzzer is

Περιγραφή Οντότητας

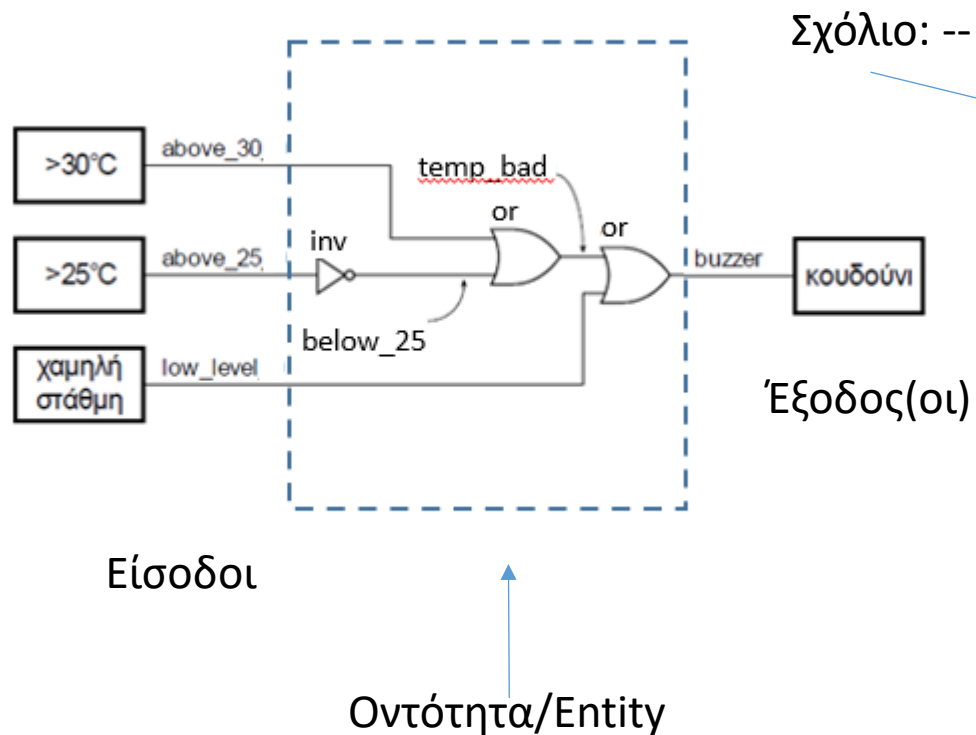
port (
above_25: **in** std_logic;
above_30: **in** std_logic;
low_level: **in** std_logic;
buzzer : **out** std_logic);

end entity buzzer;

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



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



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**

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

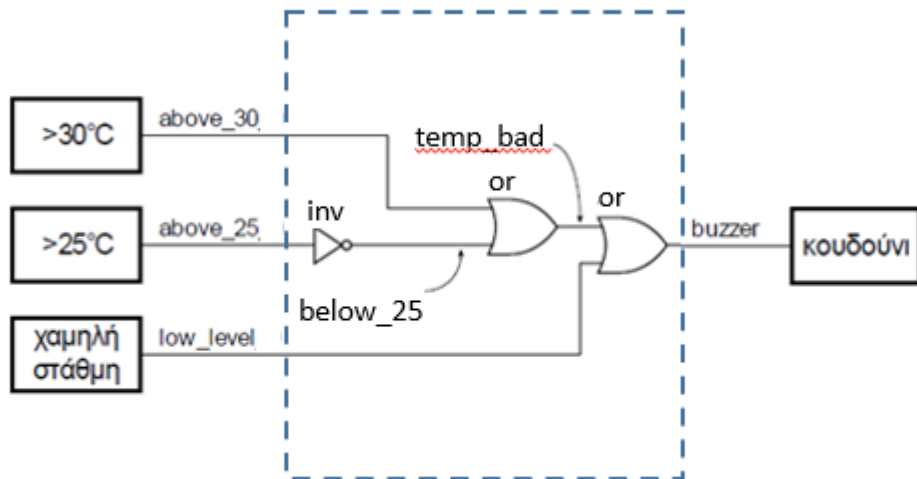
- Μπορούμε να έχουμε και δήλωση σήματος ως std_logic_vector χωρίς index range.
- Σε αυτή την περίπτωση υπονοείται από άλλη οντότητα
- Μπορείτε να βρείτε παράδειγμα :

<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);

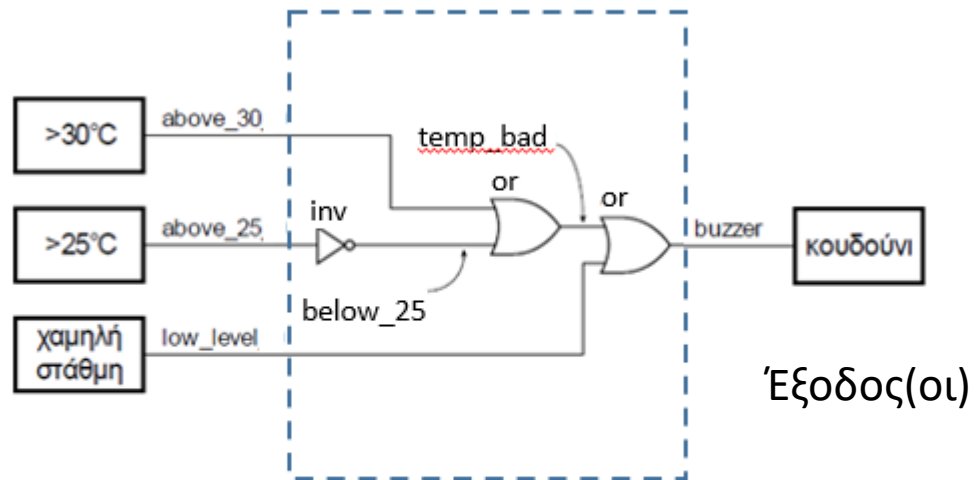
end architecture Dataflow;

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

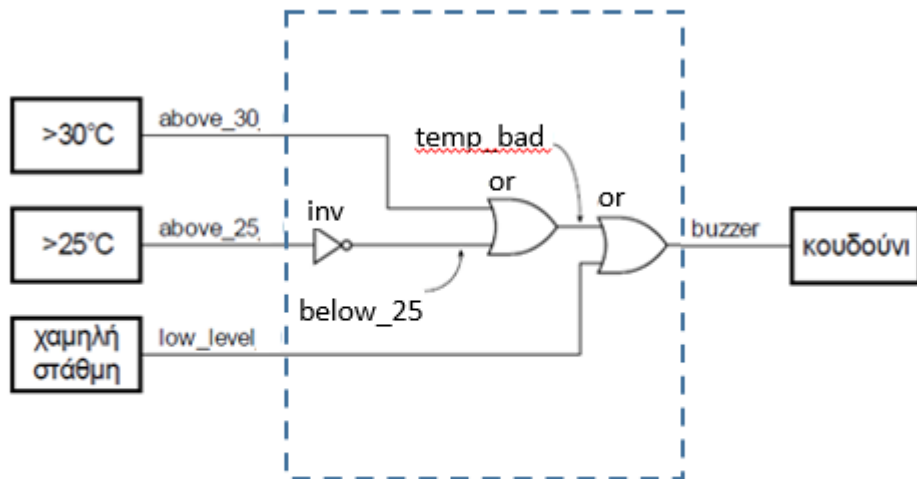
`temp_bad`

-

`below_25`

Έξοδος(οι)

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 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

- **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

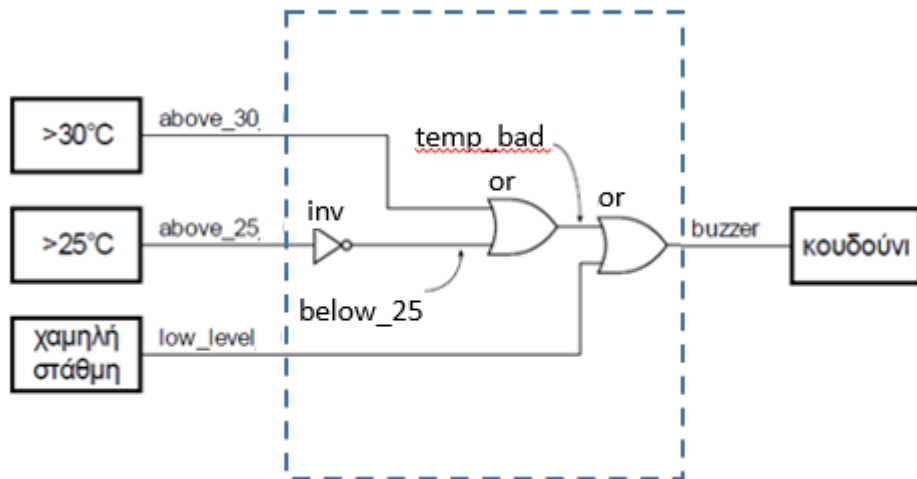
Υλοποίηση Αρχιτεκτονικής (Behavioral)

- **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

    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_bad: std_logic;

begin

    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_bad);
    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

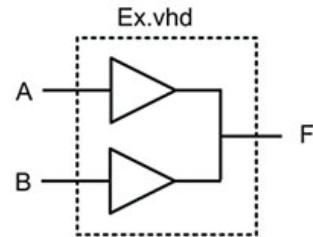
UnResolved: Αν υπάρχουν 2 εντολές που δίνουν διαφορετικές τιμές στο ίδιο σήμα, η VHDL ΔΕΝ θα μπορέσει να επιλύσει τη διαφορά και θα προκύψει compiler error

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

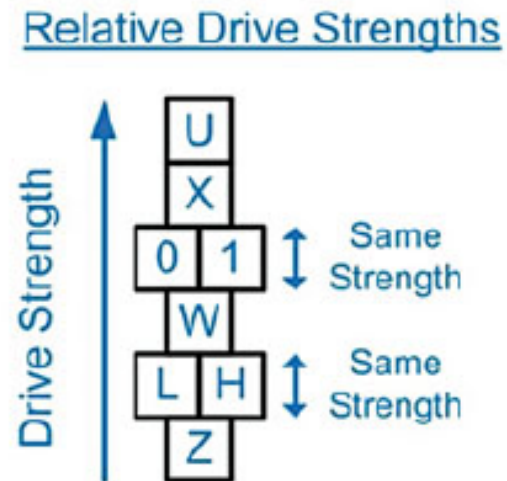
Package: IEEE.std_logic_1164.all,
Std_logic – Συνδυασμοί δύο σημάτων

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()

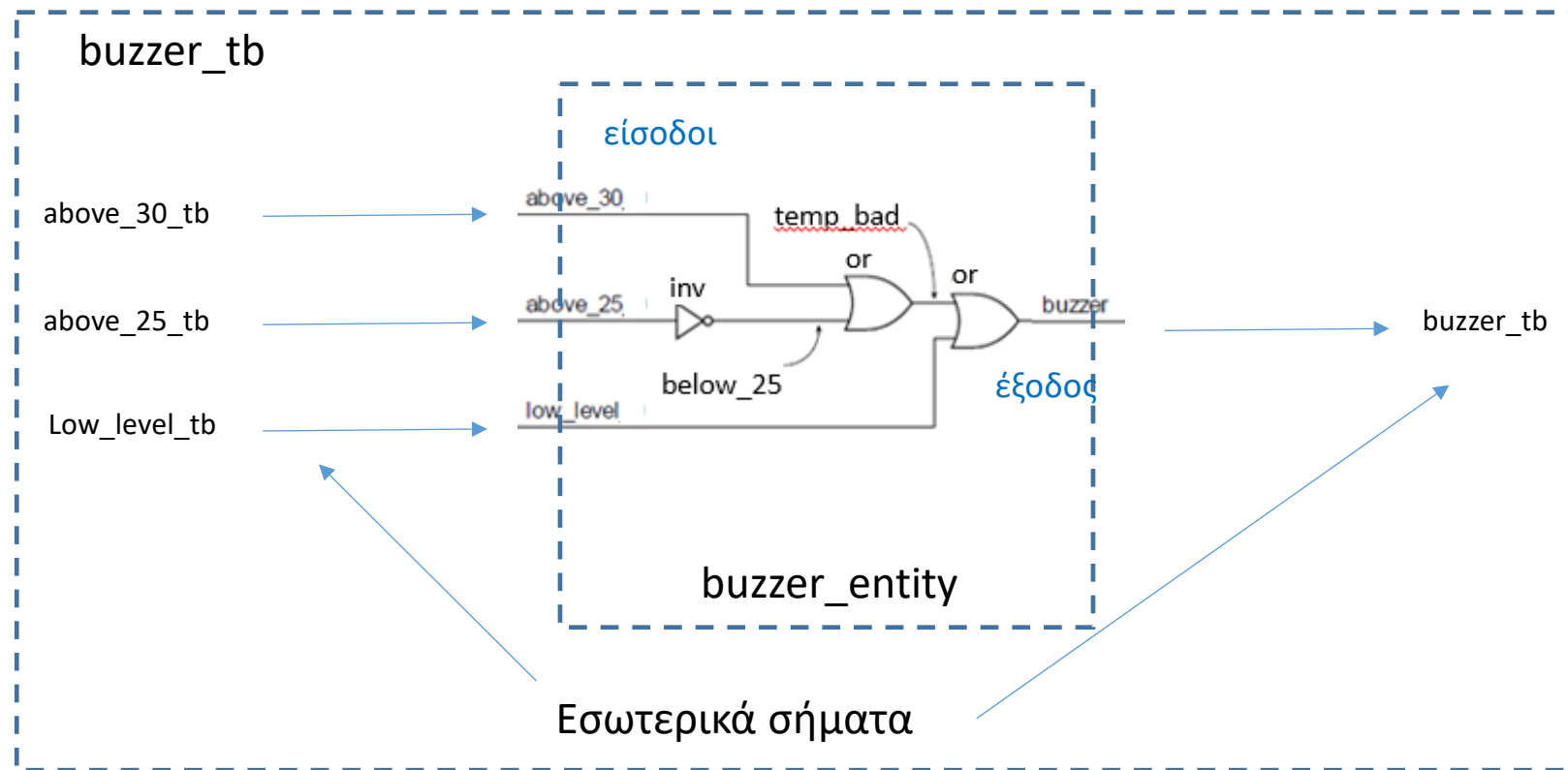
VHDL – Package: IEEE.std_logic_1164.all

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

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_0 => above_25_tb,  
  above_30_0 => above_30_tb,  
  low_level_0 => 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;
```

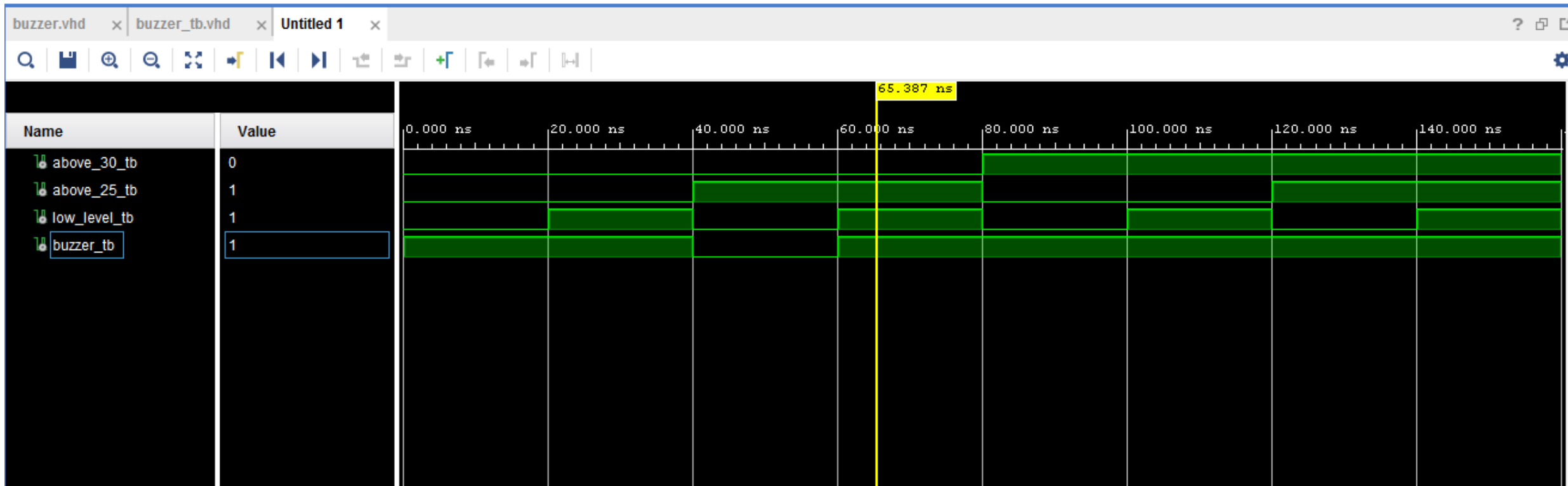
```
end architecture Behavioral;
```

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

EXPLICIT PORT MAPPING

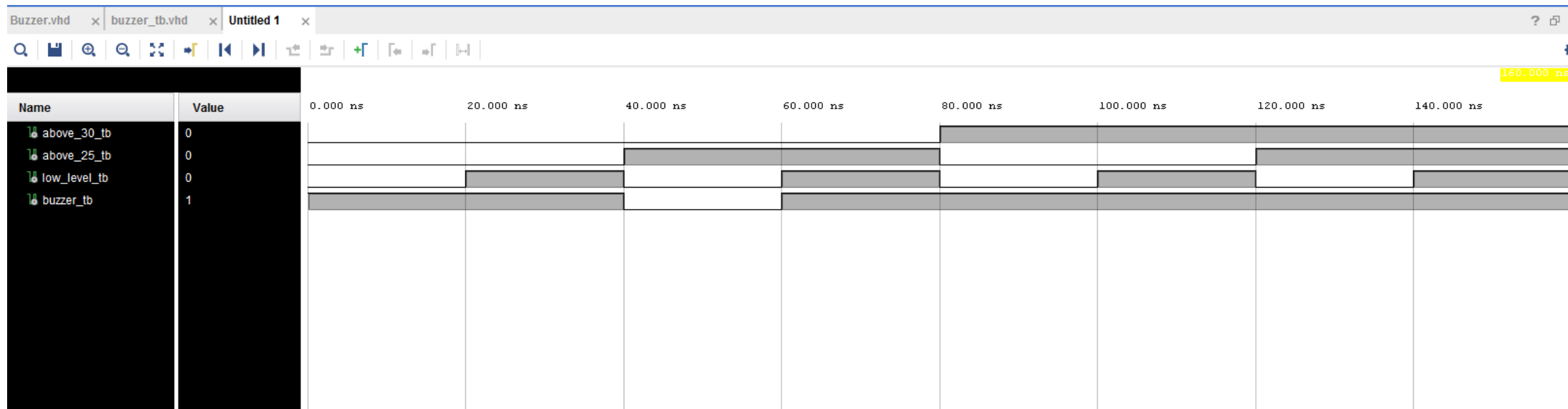
VHDL

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