dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

Εργαστήριο Λογικής Σχεδίασης

Τύποι Σημάτων – Δομή Process – Τύποι Αρχιτεκτονικής

Βασιλόπουλος Διονύσης

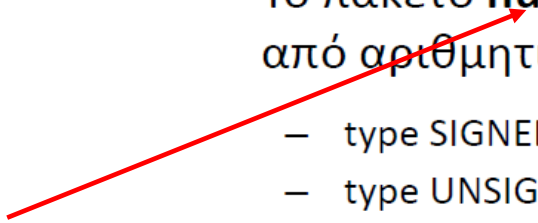
Ε.ΔΙ.Π. Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Τύποι σημάτων

Signed-Unsigned αριθμοί

- Το πακέτο **numeric_std** παρέχει τους τύπους signed και unsigned και μια σειρά από αριθμητικές λειτουργίες καθώς και συναρτήσεις μετατροπής
  - type SIGNED is array (NATURAL range <>) of STD_LOGIC;
  - type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;



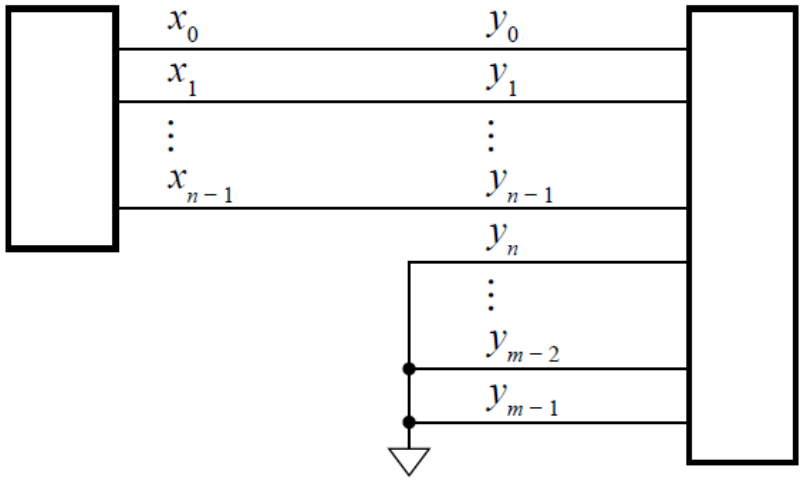
- **signal** x: **signed** (7 **downto** 0);
  - Αναπαριστά τους προσημασμένους αριθμούς (2's complement) με 8 bit
- **signal** y: **unsigned** (3 **downto** 0);
  - Αναπαριστά τους μη-προσημασμένους (θετικούς) αριθμούς με 4 bit
- Δηλώνονται σαν διανύσματα όπως ο τύπος std_logic_vector
- Επιτρέπουν την εκτέλεση αριθμητικών λειτουργιών
  - Σε αντίθεση με τα διανύσματα τύπου std_logic_vector

VHDL – Unsigned

Επέκταση μηδενός

- Για επέκταση αριθμού από *n* bit σε *m* bit
  - Πρόσθεση αρχικών bit 0
  - π.χ., 72_{10} = 1001000 = 000001001000

Σε όλες τις εντολές όσα bit είναι στο δεξί μέλος τόσο ακριβώς πρέπει να είναι και στο αριστερό




```
signal x : unsigned(3 downto 0);
signal y : unsigned(7 downto 0);

y <= "0000" & x;
```

```
y <= resize(x, 8);
```

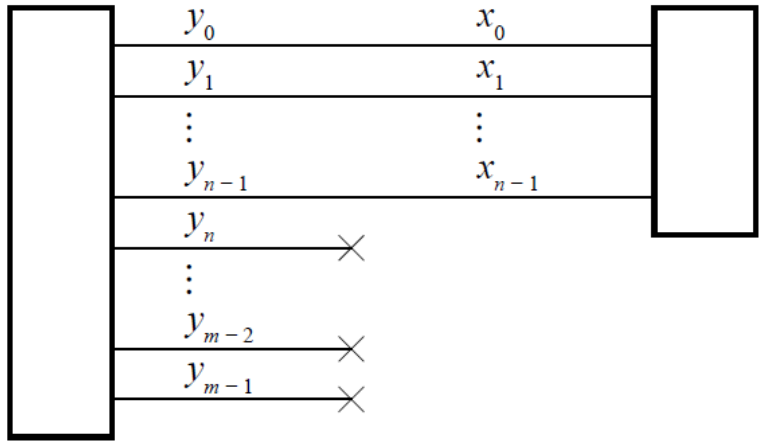
Συνένωση - concatenation

VHDL – Unisigned

Αποκοπή

Για αποκοπή από *m* bit σε *n* bit

- Απορρίπτουμε τα αριστερότερα bit
- Η τιμή διατηρείται εφόσον τα bit που απορρίπτονται είναι 0
- Το αποτέλεσμα είναι το *x* mod 2^n



Έστω: signal y : unsigned(7 downto 0);
signal x : unsigned(3 downto 0);

```
x <= y(3 downto 0);
```

```
x <= resize(y, 4);
```

Ποια είναι πιο ευέλικτη;

x <= resize(y, x'length)

Και για επέκταση και για αποκοπή.

VHDL – Unsigned

Πρόσθεση

```
library ieee; use ieee.numeric_std.all;
...
signal a, b, s: unsigned(7 downto 0); …
s <= a + b;
```

+: Παράγει αποτέλεσμα ίδιου μήκους.
Δεν ανιχνεύει υπερχείλιση
Θα μπορούσαμε με προφανώς να έχουμε και - .

Επέκταση με μηδενικά στα a και b κατά 1-bit
Το c είναι το MSB του αποτελέσματος

```
signal tmp_result : unsigned(8 downto 0);
signal c : std_logic;
…
tmp_result <= ('0' & a) + ('0' & b);
c <= tmp_result(8);
s <= tmp_result(7 downto 0);
```

Λύση
Χρειαζόμαστε ένα επιπλέον bit για το κρατούμενο (**C**arry)

VHDL – Unsigned

Αφαίρεση

```
library ieee; use ieee.std_logic_1164.all, ieee.numeric_std.all;
entity adder_subtracter is
  port ( x, y          : in unsigned(11 downto 0);
         s             : out unsigned(11 downto 0);
         mode          : in std_logic;
         ovf_unf       : out std_logic );
end entity adder_subtracter;

architecture behavior of adder_subtracter is
signal s_tmp : unsigned(12 downto 0);
begin
  s_tmp <= ('0' & x) + ('0' & y) when mode = '0' else
           ('0' & x) - ('0' & y);
  s <= s_tmp(11 downto 0);
  ovf_unf <= s_tmp(12);
end architecture behavior;
```

Υπό συνθήκη

VHDL – Unsigned

Increments

- Απλά, πρόσθεση ή αφαίρεση του 1

```
signal x, s: unsigned(15 downto 0);
...

s <= x + 1;  -- increment x

s <= x - 1;  -- decrement x
```

- Σημείωση: 1 (ακέραιος), όχι '1' (bit)

VHDL – Unsigned

Γινόμενο

- Μεγαλύτερο αποτέλεσμα για τελεστέους των n bit :

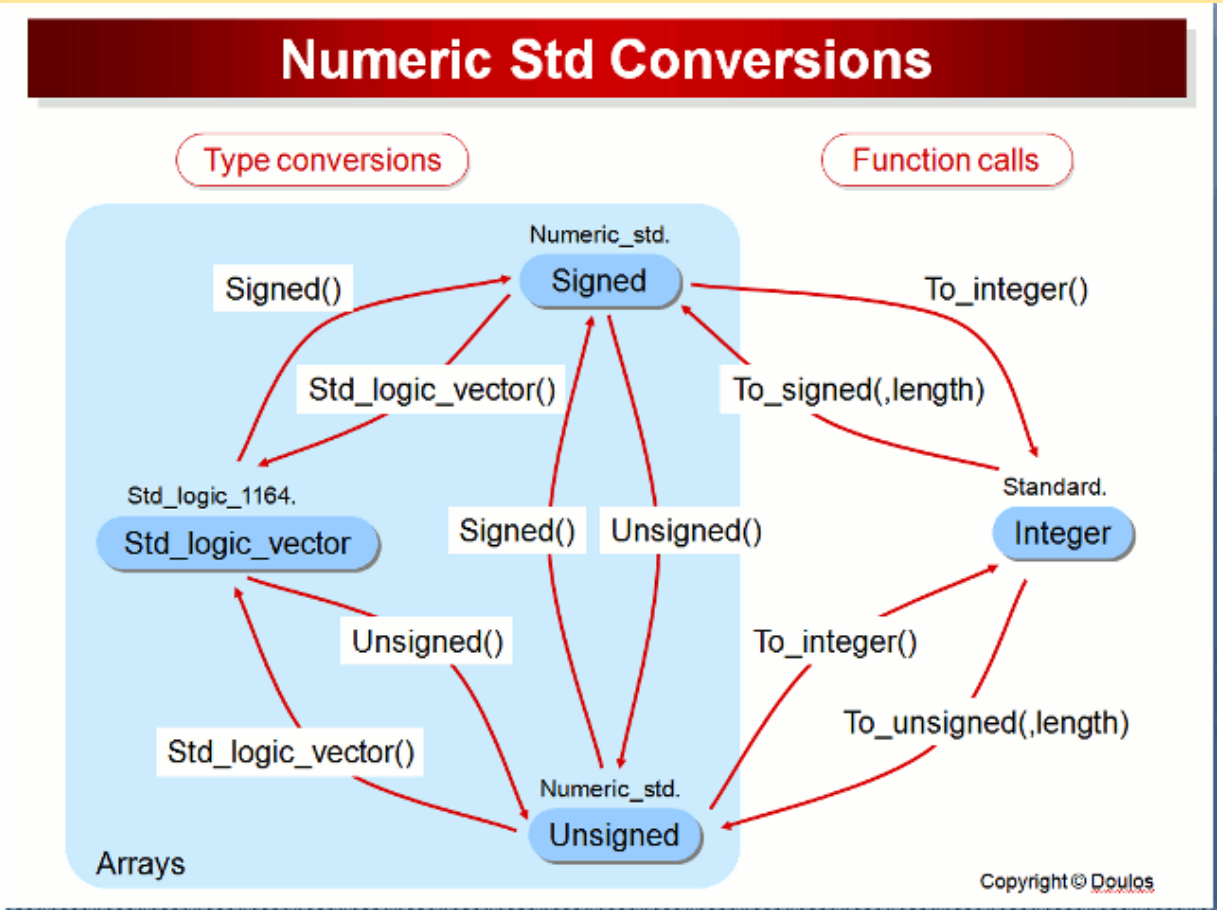
$$(2^n-1)(2^n-1)=2^{2n}-2^n-2^n+1=2^{2n}-\left(2^{n+1}-1\right)$$

  - Απαιτεί 2^{2n} bit για αποφυγή υπερχείλισης
  - Γινόμενο τελεστέων των n bit και m bit
    - Απαιτεί $n+m$ bit

```
signal x : unsigned(7 downto 0);
signal y : unsigned(13 downto 0);
signal p : unsigned(21 downto 0);
…
p <= x * y;
```

VHDL – Τύποι σημάτων

Μετατροπές



VHDL – Τύποι σημάτων

Μετατροπές

Έστω οι δηλώσεις:
signal x: std_logic_vector(3 downto 0);
signal y: unsigned(3 downto 0);

Για να μετατρέψουμε τον ένα τύπο στον άλλο:
x<=std_logic_vector(y);
ή
y<=unsigned(x);

Έστω οι δηλώσεις:
signal x: std_logic_vector(5 downto 0);
signal y: unsigned(3 downto 0);

Για να μετατρέψουμε τον ένα τύπο στον άλλο:
x<=std_logic_vector(resize(y,x'length));
ή
y<=unsigned(x(3 downto 0));

σωστό π.χ. και το y<=unsigned(x(4 downto 1));

Όσα bit είναι αριστερά μιας εντολής, τόσα πρέπει να είναι και δεξιά

VHDL – Process

- **process**: η διεργασία είναι μία ομάδα από εντολές που εκτελούνται ακολουθιακά

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;
```

VHDL – Process

Variables (Μεταβλητές)

variable_name: το όνομα της μεταβλητής
(εάν είναι πολλές μεταβλητές χωρίζονται με κόμμα)

- *μέσα στις διεργασίες ορίζονται τοπικές μεταβλητές και ΟΧΙ εσωτερικά σήματα*
- *στις δηλώσεις μεταβλητών (μετά το variable) προσδιορίζονται μεταβλητές που μπορεί να μην έχουν τη φυσική σημασία του σήματος*

variable_type: ο τύπος της μεταβλητής (STD_LOGIC)

VHDL – Process

Variables (Μεταβλητές)

```
variable my_number: natural;

variable my_logic: std_logic:='1';

variable my_vector :  std_logic_vector(3 downto 0):="1110";
```

Αρχικοποίηση τιμών. Μπορεί να γίνει και σε ΟΛΑ τα σήματα.

Σε process, function, procedure …

VHDL – Process

Variables (Μεταβλητές)

```
variable_name := expression;
```

ΠΡΟΣΟΧΗ
Είναι **:=** και όχι **<=**

- **variable_name**: το όνομα της μεταβλητής
- **expression**: έκφραση με σήματα, μεταβλητές και τελεστές
  - στις ακολουθιακές (sequential) εντολές ανάθεσης μεταβλητής :
    - στην έκφραση προσδιορίζονται **σήματα**, που ανήκουν ή δεν ανήκουν στη λίστα ευαισθησίας, και **μεταβλητές** που δηλώνονται κατά τη δήλωση μεταβλητών
    - στο αριστερό μέρος της εντολής προσδιορίζεται **μεταβλητή** που δηλώνεται κατά τη δήλωση των μεταβλητών

VHDL – Process

Variables (Μεταβλητές)

- Διαφορά μεταβλητής και σήματος μέσα σε μία διεργασία
  - η μεταβλητή παίρνει νέα τιμή **στιγμιαία** με τον τελεστή ανάθεσης **:=**, αμέσως μόλις εκτελεστεί η αντίστοιχη εντολή μέσα στη διεργασία
  - σε αντίθεση, το σήμα παίρνει νέα τιμή **με καθυστέρηση δέλτα δ_{delay}**, με τον τελεστή ανάθεσης **<=**, στο **τέλος** της εκτέλεσης της διεργασίας

Η χρήση των μεταβλητών μειώνει σημαντικά το χρόνο της προσομοίωσης

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Behavioral)

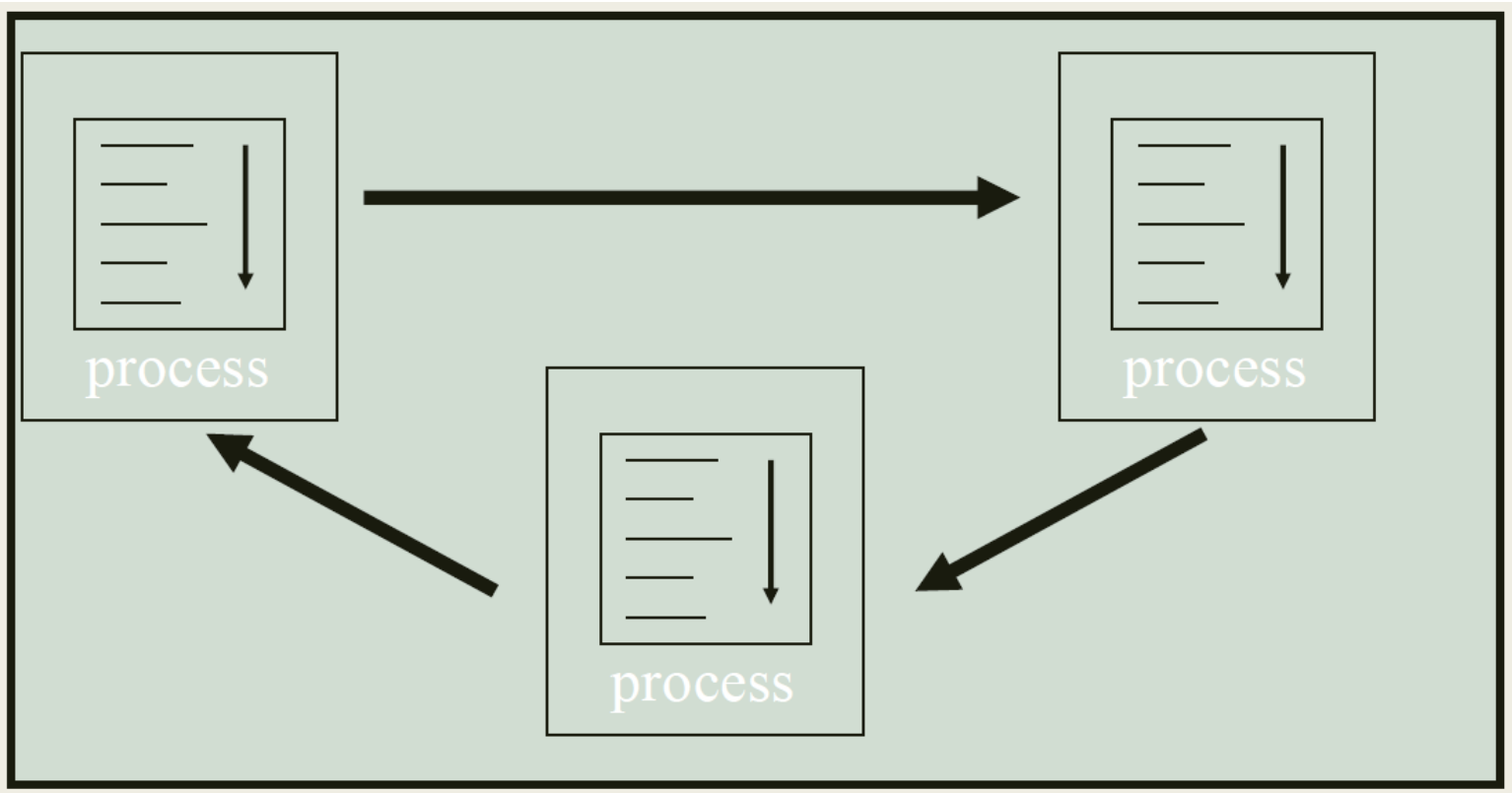
```
architecture Behavioral of buzzer is
begin

proc1: process(low_level_0,  above_30_0, above_25_0) is
begin

buzzer<= low_level_0 or (above_30_0 or not above_25_0);

end process proc1;

end architecture Behavioral;
```

VHDL – Process

signal_name: το όνομα του σήματος
(εάν είναι πολλά σήματα χωρίζονται με κόμμα)

- *στις δηλώσεις των σημάτων (μετά το process) που απαρτίζουν τη λίστα ευαισθησίας (sensitivity list) το σήμα είναι είσοδος της υπομονάδας που δηλώνεται κατά τη δήλωση των διαύλων της οντότητας ή εσωτερική διασύνδεση της υπομονάδας*
- *κάθε αλλαγή τιμής σήματος εισόδου που ανήκει στη λίστα ευαισθησίας οδηγεί στην ακολουθιακή εκτέλεση των εντολών της διεργασίας μία φορά*
- *εάν περισσότερες από μία εντολές αναθέτουν τιμή σε κάποιο σήμα λαμβάνεται υπόψη μόνο η τελευταία ακολουθιακή εντολή*
- *Προσοχή στη δήλωση των σημάτων στη λίστα ευαισθησίας*
  - μία ελλιπής δήλωση σημάτων στη λίστα ευαισθησίας θα οδηγήσει είτε σε μη σωστή σύνθεση, είτε σε ασυμφωνία της προσομοίωσης πριν και μετά τη σύνθεση και την υλοποίηση

VHDL – Process



Κάθε διεργασία (process) εκτελεί τις εντολές της ακολουθιακά, ενώ πολλές διεργασίες μαζί αλληλεπιδρούν ταυτόχρονα. Επίσης, ταυτόχρονα αλληλεπιδρούν εντολές ταυτόχρονης ανάθεσης και διεργασίες.

VHDL – Process

Διαφορά στην εφαρμογή της τιμής μίας sequential εντολής ανάθεσης μεταβλητής ή σήματος μέσα σε μία διεργασία :

– η **μεταβλητή παίρνει νέα τιμή άμεσα** με τον τελεστή ανάθεσης :=, αμέσως μόλις εκτελεστεί η αντίστοιχη εντολή μέσα στη διεργασία
    * Η variable δεν χρησιμοποιείται στην υλοποίηση της λογικής

– σε αντίθεση, **το σήμα παίρνει νέα τιμή με καθυστέρηση** δέλτα δ_{delay}, με τον τελεστή ανάθεσης <=, στο τέλος της εκτέλεσης της διεργασίας

– το σήμα θυμάται την τιμή του μέχρι να φτάσει το τέλος της εκτέλεσης της διεργασίας και να λάβει μία νέα τιμή
    * Το εσωτερικό σήμα χρησιμοποιείται στην υλοποίηση της λογικής

VHDL – Process

- Όταν **δεν υπάρχει sensitivity list** οι εντολές μέσα στο process **εκτελούνται συνέχεια**. Θα πρέπει να υπάρχει **εντολή wait** ώστε να αποδοθούν τιμές στα σήματα (όπως κάνουμε στο simulation).
- Οι εντολές μέσα στο process **εκτελούνται τουλάχιστον 1 φορά**, ακόμα και εάν υπάρχει sensitivity list.
- Οι τιμές αποδίδονται στα σήματα είτε κάθε φορά που συναντάται η εντολή **wait** μέσα στο process είτε όταν φτάνουμε στο **end** του process.
- Εάν υπάρχει wait σε σήμα που δεν αλλάζει τιμή (εντός του process) τότε η εκτέλεση σταματάει στο συγκεκριμένο wait.
- Όλο το **process θεωρείται ΜΙΑ εντολή μέσα σε μια αρχιτεκτονική** και εκτελείται παράλληλα με πιθανές άλλες εντολές της.

VHDL – Process

```
test:process (a,b) is
begin
....
end process test;
```

```
test:process is
begin
....
wait on a,b;
end process test;
```

Δεν το γράφουμε αλλά ισχύει

```
test:process (a,b) is
begin
....
end process test;
```

```
test:process (a,b) is
begin
....
wait on c;
end process test;
```

Το process «κολλάει» σε αυτή την εντολή

VHDL - Παράδειγμα

Πραγματικό πρόβλημα

Σε ένα Computer Room υπάρχουν δύο (2) αισθητήρες θερμοκρασίας (**Sensor_1** και **Sensor_2**) και δύο (2) κλιματιστικά (**AirCond_1** και **AirCond_2**). Το πρώτο κλιματιστικό (AirCond_1) λειτουργεί εάν τουλάχιστον ένας από τους δύο αισθητήρες ανιχνεύσουν θερμοκρασία άνω των 35 βαθμών στο computer room. Το δεύτερο κλιματιστικό (AirCond_2) λειτουργεί εάν και οι δύο αισθητήρες ανιχνεύσουν θερμοκρασία άνω των 35 βαθμών στο computer room. Θεωρείστε ότι κάθε αισθητήρας δίνει σήμα (‘1’) μόνο όταν η θερμοκρασία που ανιχνεύει γίνει μεγαλύτερη των 35 βαθμών (>35). Σε άλλη περίπτωση ο αισθητήρας στέλνεις την τιμή ‘0’. Σχεδιάστε και υλοποιήστε το λογικό κύκλωμα που περιγράφει το ανωτέρω πρόβλημα. Το όνομα του Vivado Project θα είναι Lab_1, της οντότητας θα είναι επίσης Lab_1 ενώ το όνομα της αρχιτεκτονικής Lab1_Beh.

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

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Dataflow)

```
architecture Dataflow of CR_AC is
begin

        AirCond_1<=Sensor_1 or Sensor_2;
        AirCond_2<=Sensor_2 and Sensor_3;

end architecture Dataflow;
```

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 - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Behavioral)

```
architecture behavioral of CR_AC is
begin

room:  process (Sensor_1, Sensor_2) is
          begin
               AirCond_1<=Sensor_1 or Sensor_2;
               AirCond_2<=Sensor_1 and Sensor_2;
          end process room;

end architecture Behavioral;
```

VHDL

Υλοποίηση Αρχιτεκτονικής (Behavioral)

1. Όμοιο RTL Design με το Dataflow
2. ΔΕΝ χρειάζεται να αλλάξουμε τίποτα στην προσομοίωση
3. Εισαγωγή της δομής process (με λίστα ευαισθησίας)
4. Σε περίπτωση που υπάρχει μόνο η εντολή process, τότε ουσιαστικά οι εντολές μας εκτελούνται σειριακά
5. Σε περίπτωση που υπάρχουν και απλές εντολές ανάθεσης στην αρχιτεκτονική, τότε μπορείτε να θεωρήσετε όλη τη δομή process σαν μία εντολή που εκτελείται παράλληλα με τις εντολές ανάθεσης (που είναι εκτός process)

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

architecture Dataflow of AND_gate is
begin
   O<=A AND B;
end Dataflow;
```

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Structural – Δήλωση Αρχιτεκτονικής Κύριας Οντότητας)

```
architecture Structural of CR_AC is

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

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

begin

or_comp : OR_gate  port map (A=>Sensor_1,
B=>Sensor_2, O=>AirCond_1);
and_comp: AND_gate port map (A=>Sensor_1,
B=>Sensor_2, O=>AirCond_2);

end architecture Structural;
```

VHDL

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 αποτελείτε μόνο 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 βλέπουμε ότι σχηματίζεται ένα δέντρο, τα φύλλα του οποίου είναι 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 - Περίληψη

- Τύπος Unsigned
- Process
- Variable
- Παράδειγμα των 3 ειδών αρχιτεκτονικής (Dataflow, Behavioral, Structural)
- Διαβάζετε τις παραγράφους 2.3, 2.4, 3.1 (θεωρία και VHDL) από Ashenden και 2.2 - 2.5, 4.2.5, 4.3, 4.4.1 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.
- Συνοπτικός οδηγός VHDL: https://redirect.cs.umbc.edu/portal/help/VHDL/summary_one.html