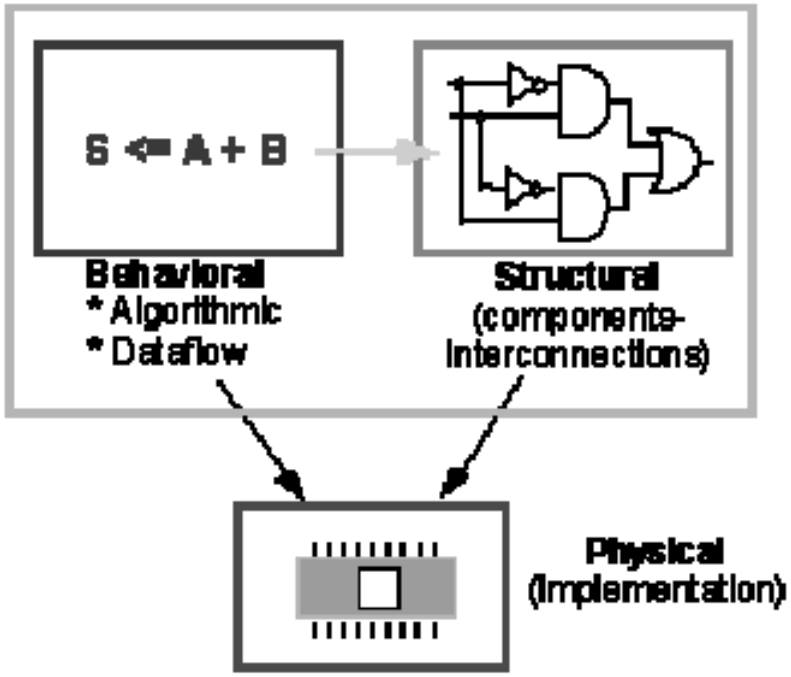
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης
ΕΛΛΑΔΑ 2008
Ανάπτυξη παντού. Ανάπτυξη για όλους.

Βασικές Δομές Μοντελοποίησης

(Peter Ashenden, The Students Guide to VHDL)

Επίπεδα Αναπαράστασης - Αφαίρεσης



Αθροιστής: SUM<=A+B

RED<=VEH_AT_MAIN´ or (VEH_AT_MAIN and VEH_AT_LOCAL);

Επίπεδα Αναπαράστασης - Αφαίρεσης

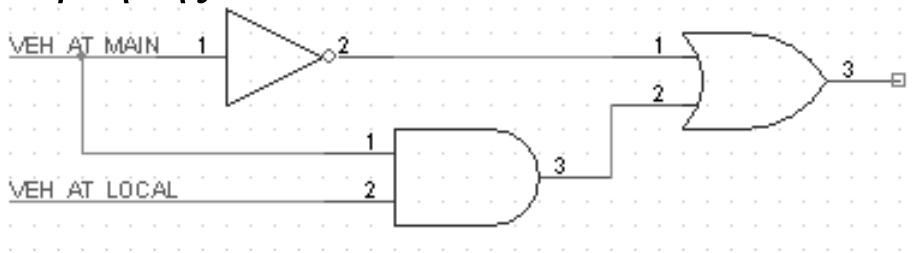
Επίπεδο Συμπεριφοράς (behavioral): Σχέση σημάτων I/O, boolean έκφραση, μεταφορά καταχωρητών, αλγόριθμος.

- Μοντέλο ροής δεδομένων
- Αλγοριθμικό Μοντέλο

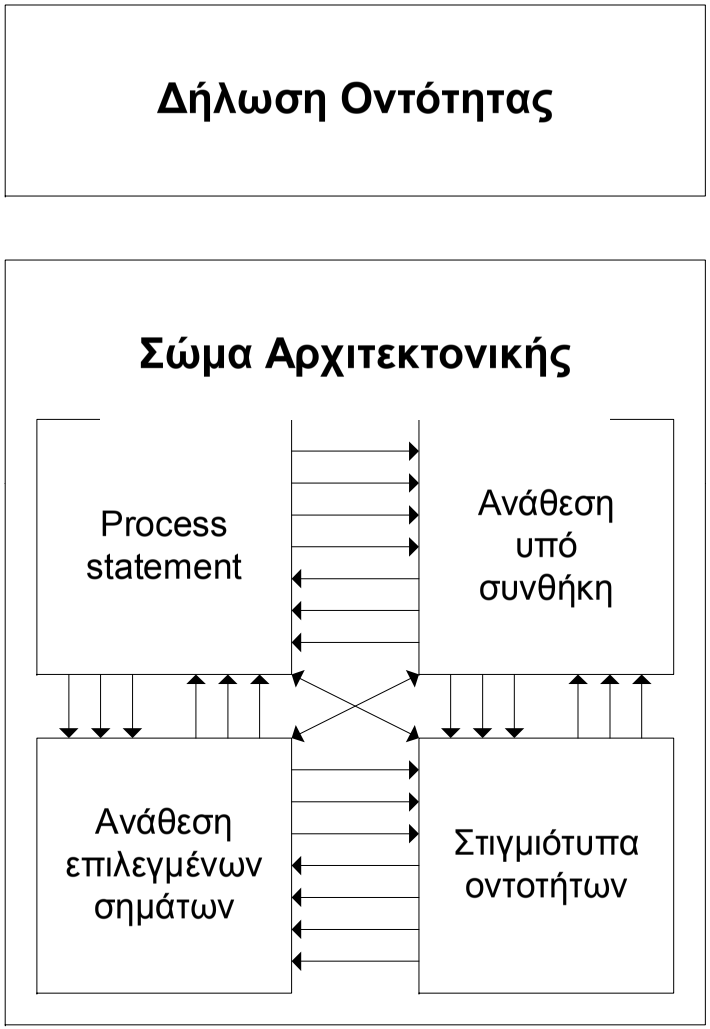
Μοντέλο ροής δεδομένων: Περιγράφει τον τρόπο με τον οποίο μετακινούνται τα δεδομένα μέσα στο σύστημα (μεταξύ καταχωρητών) – Register Transfer Level. Χρησιμοποιούνται παράλληλες εντολές.

Αλγοριθμικό Μοντέλο: Οι εντολές εκτελούνται ακολουθιακά.

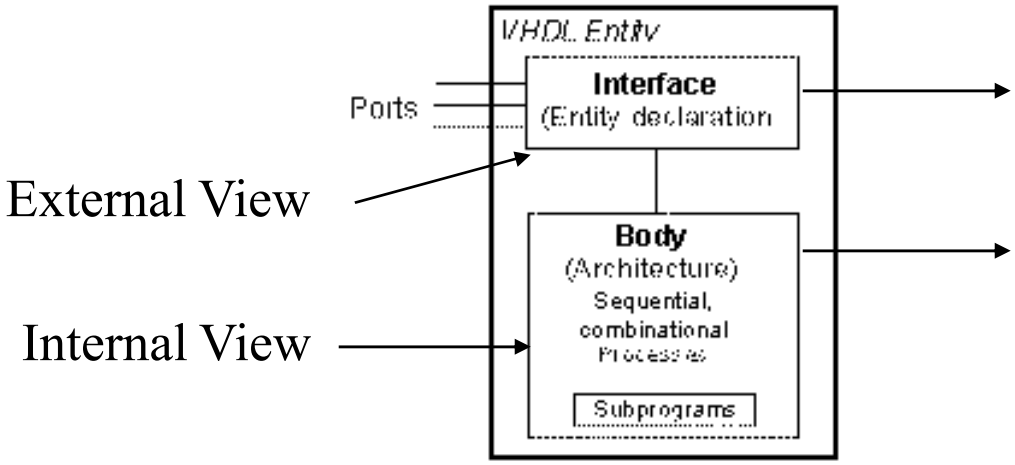
Επίπεδο Κατασκευής (structural): Συλλογή από στοιχεία διασυνδεδεμένα για την υλοποίηση επιθυμητής συνάρτησης.



Επίπεδα Αναπαράστασης - Αφαίρεσης



Βασική Δομή



Δήλωση σημάτων I/O

Περιγραφή οντότητας με χρήση διαδικασιών ή διασύνδεση οντοτήτων

```
entity NAME_OF_ENTITY is [ generic generic_declarations);]
     port (signal_names: mode type;
           signal_names: mode type;
                :
           signal_names: mode type);
end [NAME_OF_ENTITY] ;
```

Δηλώσεις οντότητας

Οι θύρες χρησιμοποιούνται σαν ακροδέκτες

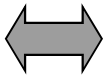
Mode
- **in**: είσοδος
- **out**: έξοδος που δεν διαβάζεται εσωτερικά
- **buffer**: έξοδος που διαβάζεται και εσωτερικά
- **inout**: είσοδος/έξοδος

Τύποι
- **bit/ bit_vector**
- **std_logic/std_logic_vector**
- **std_ulogic/std_ulogic_vector**
- **Boolean/integer/real/character**
- **time**

Η θύρα **inout** πρέπει να χρησιμοποιείται μόνο σε πραγματικά σήματα διπλής κατεύθυνσης

Παραδείγματα Δήλωσης οντότητας

```
entity adder is
    port ( a: in word;
           b: in word;
           sum : out word);
end adder;
```



```
entity adder is
    port ( a , b: in word;
           sum : out word);
end adder;
```

```
entity and_or_inv is
    port ( a1, a2, b1, b2: in bit :='1';
           y: out bit);
end and_or_inv;
```

Η αρχικοποιημένη θύρα παίρνει την αρχική τιμή της μόνο όταν μείνει ασύνδετη

```
entity adder is
end adder;
```

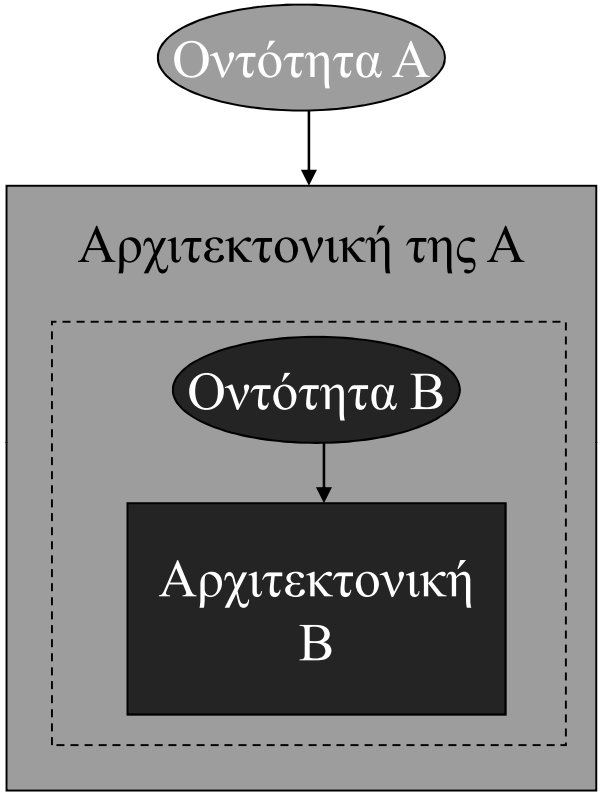
Self – contained module (top-level ιεραρχίας)

Παραδείγματα Λανθασμένης Δήλωσης οντότητας

```
entity portmode is
    port ( in_a, in_b : in bit;
           out_a, out_b : out bit);
end adder;

architecture behave of portmode is
begin
    in_a<=in_b;     --Error, can not assign to an input
    out_b<=out_a; --Error, can not read output
end behave;
```

Δυνατότητες Διασύνδεσης Θυρών

Στη θύρα της Α	**συνδέονται οι θύρες της Β**
in	in, buffer, inout
out	out, inout
buffer	buffer
inout	inout



Δηλώσεις σταθερών generics

- Οι σταθερές προσδιορίζουν μέγεθος, χρόνο κλπ
- Μεγιστοποιούν την δυνατότητα επαναχρησιμοποίησης της οντότητας

```
generic (
constant_name: type [:=value] ;
constant_name: type [:=value] ;
:
constant_name: type [:=value] );
```

Προαιρετικές

Καθορίζουν τοπικές σταθερές (χρονισμούς – μεγέθη)

Δηλώσεις σταθερών generics

```
library ieee; use ieee.std_logic_1164.all
entity full_adder is
    generic (size: integer:=8);
    port ( X,Y:   in std_logic_vector (size-1 downto 0);
           C_IN: in bit;
           SUM: out std_logic_vector (size-1 downto 0);
           C_OUT: out bit);
end full_adder;
…
Size8: full_adder port map (d=>a, q=>b);
Size10: full_adder generic map (size=>10) port map (d=>a, q=>b);
```

- Αλλάζοντας την τιμή της σταθεράς **size** αλλάζει και το μέγεθος του αθροιστή (πχ. 10 στο παράδειγμα)
- Εάν δεν δηλωθεί κατά το Instantiation η τιμή size τότε χρησιμοποιείται η τιμή 8 που είναι η default.

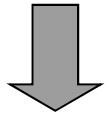
Σώμα Αρχιτεκτονικής

```
architecture architecture_name of NAME_OF_ENTITY is
-- Declarations
      -- components declarations
      -- signal declarations
      -- constant declarations
      -- function declarations
      -- procedure declarations
      -- type declarations

      :

begin
-- Statements

      :

end architecture_name;
```

Σώμα Αρχιτεκτονικής

Σώμα Αρχιτεκτονικής

- Περιγράφει την εσωτερική λειτουργία ενός module.
- Χρησιμοποιείται για περιγραφή λειτουργίας ως αλγόριθμος.
- Εφαρμόζει κάποιες λειτουργίες στις τιμές των θυρών εισόδου και παράγει τιμές των θυρών εξόδου.
- Οι λειτουργίες μπορούν να περιγραφούν με processes και ακολουθιακές εντολές, ή με συλλογή components που αναπαριστούν υποκυκλώματα.
- Ενδιάμεσες τιμές κατά την εφαρμογή των λειτουργιών μπορούν να αποθηκευτούν σε σήματα (signals).



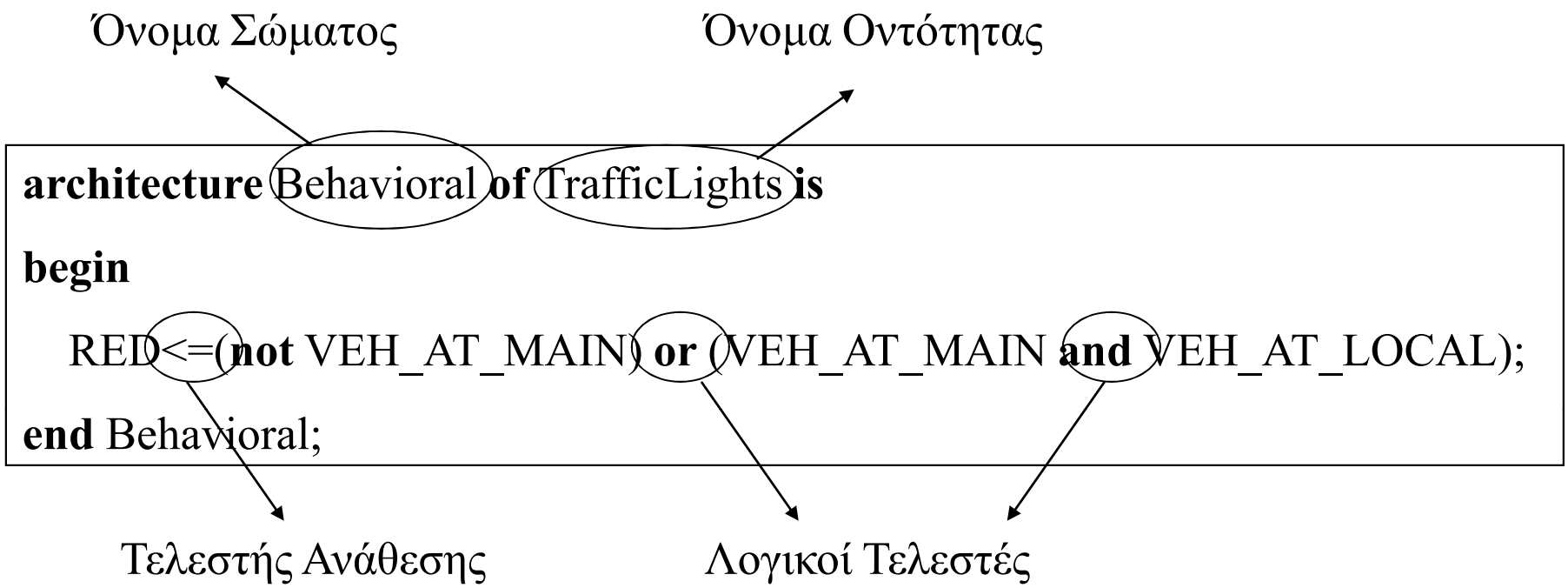
Process

Σώμα Αρχιτεκτονικής

DataFlow Statements

- Εκτελούνται ταυτόχρονα και περιγράφουν την λειτουργία του module.
- Εδώ δεν χρησιμοποιείται η “process”.
- Δεν χρησιμοποιούνται ακολουθιακές εντολές
- Χρησιμοποιούνται για συμπαγής περιγραφές

Μοντέλο Συμπεριφοράς



Ο Τελεστής Ανάθεσης <= εκτελείται όταν τουλάχιστον ένα από τα σήματα στο δεξιό μέρος αλλάζει τιμή.

Η καθυστέρηση διάδοσης μπορεί να μοντελοποιηθεί.

Η σειρά των γεγονότων καθορίζει την σειρά εκτέλεσης των αναθέσεων

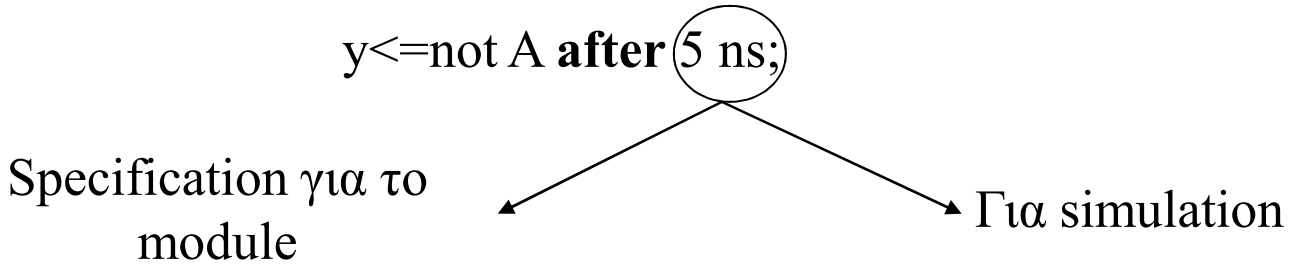
Σήματα

- Η ανάθεση σημάτων είναι η βασική λειτουργία των processes.
- Μία process ενεργοποιείται με την αλλαγή κάποιων σημάτων στις θύρες της, στα οποία είναι ευαίσθητη.

signal_assignment_statement <=

[label :] name <= [delay_mechanism] waveform;

waveform <= (value_expression [**after** time_expression]) {, …}



- Μία process που αναθέτει τιμή σε ένα σήμα ορίζει έναν οδηγό του σήματος. Κάθε σήμα πρέπει να έχει έναν οδηγό εκτός ειδικών περιπτώσεων.

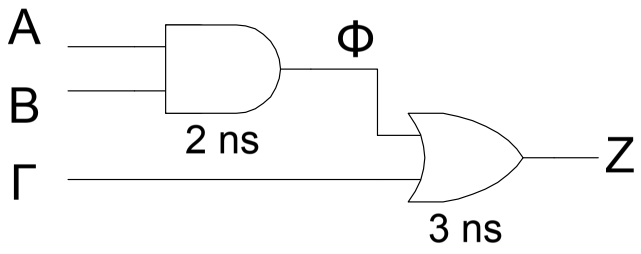
Discrete Event Simulation

Εξομοίωση διακριτών γεγονότων:

- Ο χρόνος εξομοίωσης ξεκινά από το 0 και αυξάνεται με διακριτά βήματα όσο συμβαίνουν γεγονότα (events).
- Όταν γίνεται μία ανάθεση σήματος, ο χρόνος καθυστέρησης προστίθεται στον τρέχοντα και καθορίζει πότε θα γίνει η ανάθεση (προγραμματισμός ανάθεσης).
- Όταν ο χρόνος εξομοίωσης φθάσει στον προγραμματισμένο χρόνο, η τιμή του σήματος ενημερώνεται και εάν είναι διαφορετική τότε έχουμε ένα γεγονός στο σήμα.

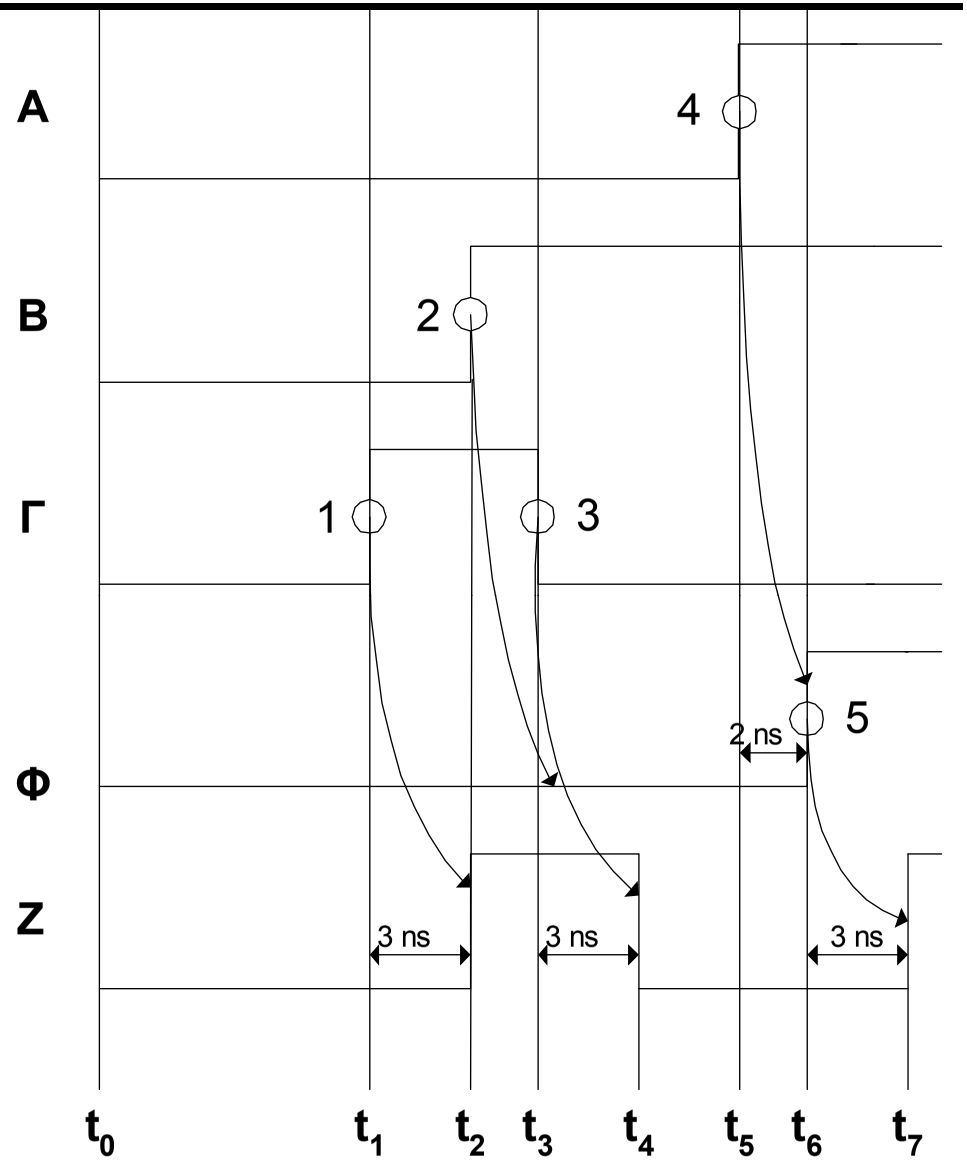
```
clock_gen : process (clk) is
begin
    if clk='0' then
        clk<='1' after T_pw, '0' after 2*T_pw;
    end if;
end process;
```

Discrete Event Simulation



Χρήση της ιδιότητας σημάτων

signal‘event



Παράδειγμα

```
library ieee;
use ieee.std_logic_1164.all;
entity DFF_CLEAR is
     port (CLK, CLEAR, D : in std_logic;
           Q : out std_logic);
end DFF_CLEAR;

architecture BEHAV_DFF of DFF_CLEAR is
begin
DFF_PROCESS: process (CLK, CLEAR)
     begin
          if (CLEAR = '1') then
               Q <= '0';
          elsif (CLK'event and CLK = '1') then
               Q <= D;
          end if;
     end process;
end BEHAV_DFF;
```

Μοντελοποίηση χρονικών καθυστερήσεων

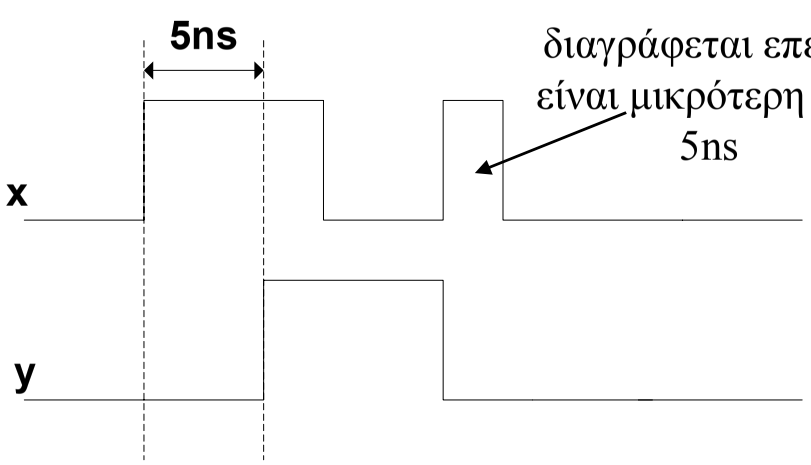
Δύο τύποι χρονικών καθυστερήσεων

Καθυστέρηση αδράνειας
(inertial delay)

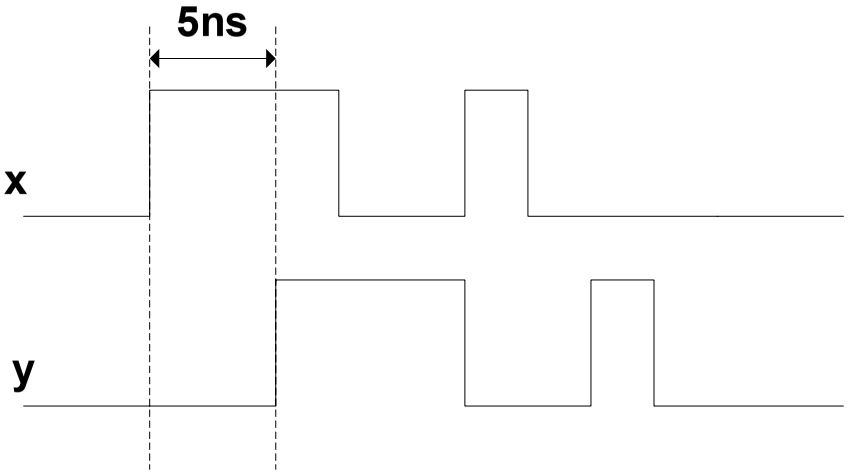
x<=y after 5ns;

Καθυστέρηση μεταφοράς
(transport delay)

x<=transport y after 5ns;



(default)



Wait statements

- Καθορίζουν πότε αντιδρούν οι processes σε αλλαγές στις τιμές σημάτων.
- Το wait statement αναγκάζει μία process να σταματήσει και ορίζει υπό ποια συνθήκη θα ξεκινήσει πάλι.

wait_statement <=

[label:] **wait** [**on** signal_name {,…}] [**until** bool_expr] [**for** time_expr]

- Μπορούμε να συμπεριλάβουμε κάθε συνδυασμό ή και κανέναν από αυτούς σε ένα wait statement.

Λίστα ευαισθησίας - Wait statements

- Ξεκινάει με την λέξη **on** και καθορίζει μία λίστα από σήματα στα οποία "αντιδρά" η process.
- Εάν συμβεί ένα γεγονός (αλλαγή τιμής) σε ένα σήμα που ανήκει σε λίστα ευαισθησίας τότε η process ξεκινάει αμέσως (***Μοντελοποίηση Συνδυαστικής Λογικής***).

```
half_add : process is
begin
    sum<=a xor b after T_pd;
    carry<= a and b after T_pd;
    wait on a,b;
end process;
```

```
half_add : process(a,b) is
begin
    sum<=a xor b after T_pd;
    carry<= a and b after T_pd;
end process;
```

- Ένα wait statement με μία λίστα ευαισθησίας είναι ισοδύναμο με μία process που εμπεριέχει λίστα ευαισθησίας

Λίστα ευαισθησίας - Wait statements

```
entity mux2 is port ( a, b, sel : in bit; z : out bit );
end entity mux2;

architecture behavioral of mux2 is
  constant prop_delay : time := 2 ns;
begin
  slick_mux : process is
  begin
    case sel is
      when '0' =>
        z <= a after prop_delay;
        wait on sel, a;
      when '1' =>
        z <= b after prop_delay;
        wait on sel, b;
    end case;
  end process slick_mux;
end architecture behavioral;
```

Λογική Συνθήκη - Wait statements

- Ένα wait statement με την λέξη until καθορίζει την συνθήκη με την οποία ξαναρχίζει μία process που είχε ανασταλεί.
- Η συνθήκη ελέγχεται μόνο όταν συμβαίνει κάποιο γεγονός στα σήματα της συνθήκης και εάν είναι αληθής τότε ξεκινά η διαδικασία.
- Ακόμη και αν η συνθήκη είναι αληθής την στιγμή που εκτελείται το wait statement τότε δεν εκτελείται εάν δεν έχει συμβεί γεγονός στα σήματα που την απαρτίζουν.

```
clock_gen : process is
begin
    clk<='1' after T_pw, '0' after 2*T_pw;
    wait until clk='0';
end process;
```

- Εάν υπάρχει λίστα ευαισθησίας και λογική συνθήκη τότε η συνθήκη ελέγχεται μόνο σε αλλαγή σήματος της λίστας ευαισθησίας και όχι σήματος της συνθήκης

Χρονικός περιορισμός - Wait statement

- Ξεκινάει με την λέξη **for** και καθορίζει το μέγιστο χρονικό διάστημα στο οποίο θα παραμείνει ανενεργή η process.

wait for 1 ms;

- Εάν συμπεριληφθεί και λογική συνθήκη ή λίστα ευαισθησίας με χρονικό περιορισμό τότε οποιοδήποτε συμβεί πρώτο θα προκαλέσει εκκίνηση.

wait until triger='1' **for** 1 ms;

```
clock_gen : process is
begin
    clk<='1' after T_pw, '0' after 2*T_pw;
    wait for 2*T_pw;
end process;
```

- Η εντολή "wait;" χωρίς λίστα-συνθήκη-χρόνο προκαλεί μόνιμη αναστολή της process (για stimulus).

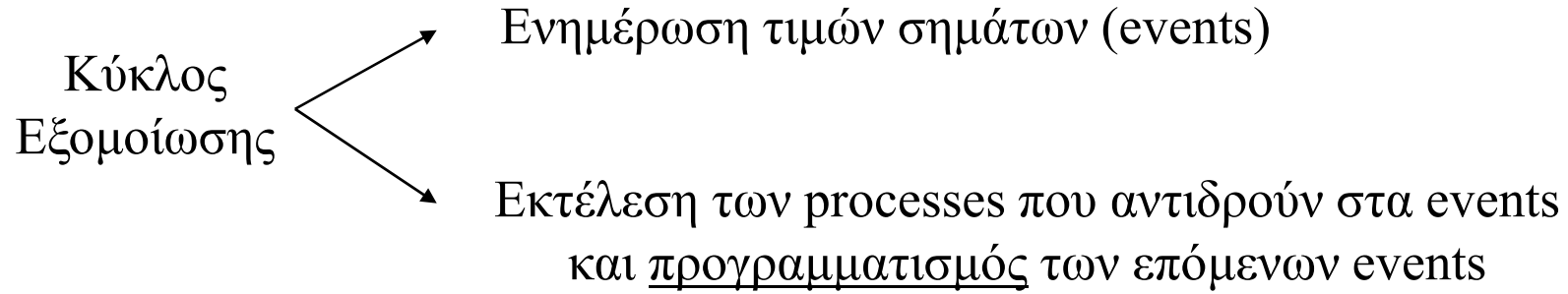
Delta Delay

- Η ανάθεση μίας τιμής σε ένα σήμα χωρίς χρονική καθυστέρηση γίνεται μόλις η διαδικασία ανασταλεί. Η καθυστέρηση των 0fs σε ένα σήμα ονομάζεται καθυστέρηση delta.

  Παράδειγμα: έστω ότι σε μία process έχουμε τις αναθέσεις

```
…                          …
a<='1';                    a<='1';
b<=a;                      if a='1' then …
…                          …
```

  εάν αρχική τιμή του a είναι '0' τότε η τιμή του b είναι '0' και όχι '1', και το if δεν εκτελείται.



Process

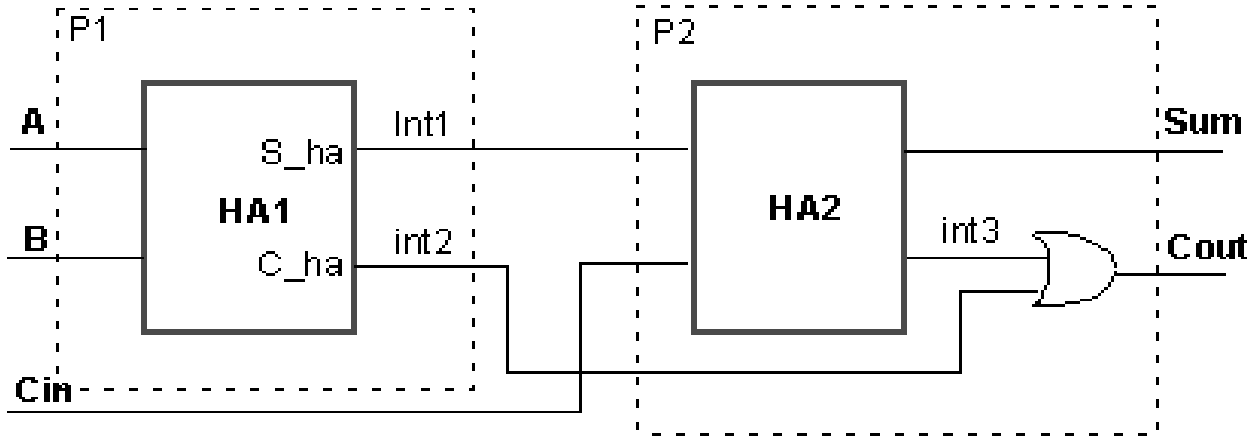
- Η δομή process δηλώνεται μέσα σε μία αρχιτεκτονική και εκτελείται παράλληλα με άλλες δομές (Concurrent).
- Οι εντολές εσωτερικά στην δομή εκτελούνται ακολουθιακά.
- Στην δομή process υπάρχει μία λίστα με σήματα στα οποία είναι "ευαίσθητη" η διαδικασία process.
- Κάθε αλλαγή στην τιμή αυτών των σημάτων προκαλεί την άμεση εκτέλεση της διαδικασίας.
- Εναλλακτικά μπορεί να συμπεριληφθεί μία εντολή WAIT.
- Μεταβλητές/σταθερές ορίζονται στο τμήμα δηλώσεων της, πριν το begin.
- Οι εντολές της εκτελούνται ακολουθιακά.
- Οι αναθέσεις των μεταβλητών εκτελούνται χωρίς εσωτερική καθυστέρηση (delay).

Η δομή process

Μία Process χωρίς wait statement εκτελείται για πάντα

[*process_label*:] **process** [(*sensitivity_list*)] [**is**]
[*process_declarations*]
begin
list of sequential statements such as:
signal assignments
variable assignments
case statement
exit statement
if statement
loop statement
next statement
null statement
procedure call
wait statement
end process [*process_label*];

Παράδειγμα



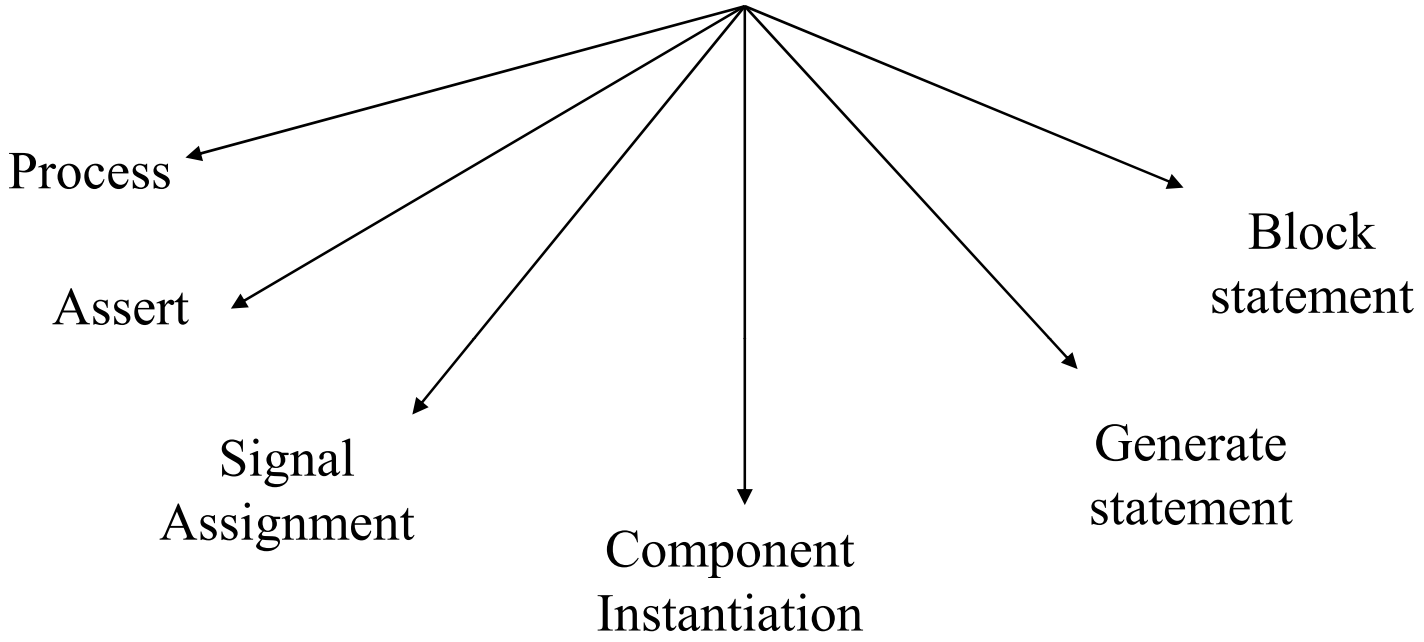

```
library ieee;
use ieee.std_logic_1164.all;
entity FULL_ADDER is
     port (A, B, Cin : in std_logic;
           Sum, Cout : out std_logic);
end FULL_ADDER;

architecture BEHAV_FA of FULL_ADDER is
signal int1, int2, int3: std_logic;
begin
```

Παράδειγμα

```
-- Process P1 that defines the first half adder
P1: process (A, B)
     begin
          int1<= A xor B;
          int2<= A and B;
     end process;
-- Process P2 that defines the second half adder and the OR -- gate
P2: process (int1, int2, Cin)
     begin
          Sum <= int1 xor Cin;
          int3 <= int1 and Cin;
          Cout <= int2 or int3;
     end process;
end BEHAV_FA;
```

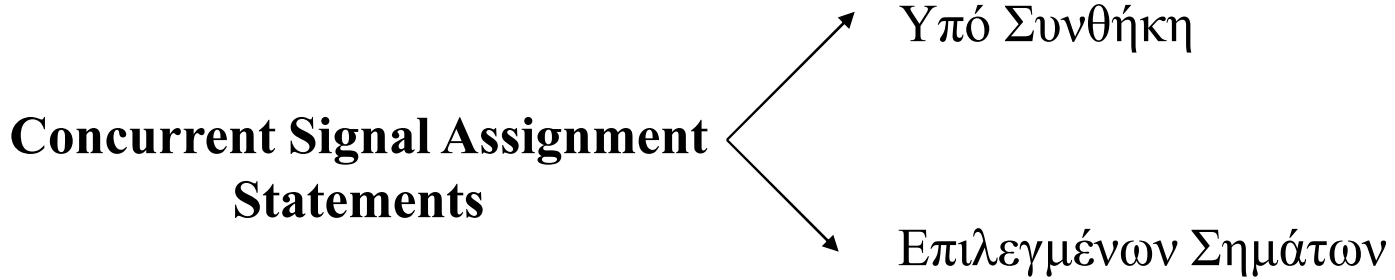
Συντρέχουσες Προτάσεις (Concurrent)

Πρόκειται για διεργασίες που τρέχουν παράλληλα (η σειρά δεν έχει σημασία).



Αναθέσεις σημάτων Concurrent

- Η δομή process χρησιμοποιείται για μοντελοποίηση συμπεριφοράς αλλά για απλές περιπτώσεις απαιτείται μία απλούστερη δομή.
- **Concurrent Signal Assignment Statements**: δομή που επιτρέπει απλή συνδυαστική μετατροπή εισόδων σε εξόδους.
- Μπορεί να συμπεριληφθεί στο σώμα αρχιτεκτονικής ανεξάρτητα από οποιαδήποτε process και εκτελείται παράλληλα με αυτές.
- Η μοντελοποίηση με αυτόν τον τρόπο ονομάζεται μοντελοποίηση ροής δεδομένων



Ροή Δεδομένων (Data Flow)

Example of a Four bit Adder using concurrent/behavioral modeling

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

entity ADD4 is
    port (
        A: in STD_LOGIC_VECTOR (3 downto 0);
        B: in STD_LOGIC_VECTOR (3 downto 0);
        CIN: in STD_LOGIC;
        SUM: out STD_LOGIC_VECTOR (3 downto 0);
        COUT: out STD_LOGIC
    );
end ADD4;

architecture ADD4_concurnt of ADD4 is

-- define internal SUM signal including the carry
signal SUMINT: STD_LOGIC_VECTOR(4 downto 0);

begin
  -- <<enter your statements here>>

  SUMINT <=  ('0' & A) + ('0' & B) + ("0000" & CIN);
  COUT <= SUMINT(4);
  SUM <= SUMINT(3 downto 0);
end ADD4_concurnt;
```

Ανάθεση σημάτων

```
entity AND2 is
      port (in1, in2: in std_logic;
       out1: out std_logic);
end AND2;

architecture behavioral_2 of AND2 is
begin
            out1 <= in1 and in2;
end behavioral_2;
```

```
entity XNOR2 is
     port (A, B: in std_logic;
      Z: out std_logic);
end XNOR2;

architecture behavioral_xnor of XNOR2 is
     -- signal declaration (of internal signals X, Y)
     signal X, Y: std_logic;
begin
           X <= A and B;
           Y <= (not A) and (not B);
           Z <= X or Y;
End behavioral_xnor;
```

Όλες οι αναθέσεις γίνονται ταυτόχρονα

Αναθέσεις υπό συνθήκη

```
Target_signal <= expression when Boolean_condition else
                 expression when Boolean_condition else
                     :
                 expression;
```

- Η ανάθεση υπό συνθήκη μας επιτρέπει να καθορίσουμε ποια κυμματομορφή θα οδηγήσει το σήμα.
- Το πλεονέκτημα της δομής είναι η περιγραφική της απλότητα.
- Η δομή είναι ευαίσθητη σε αλλαγή οποιουδήποτε σήματος είτε των εκφράσεων, είτε των συνθηκών.
- Απλουστεύεται η περιγραφή απλών συνδυαστικών κυκλωμάτων όπως ο πολυπλέκτης.

Αναθέσεις υπό συνθήκη

Πολυπλέκτης

```
entity MUX_4_1_Conc is
   port (S1, S0, A, B, C, D: in std_logic;
            Z: out std_logic);
   end MUX_4_1_Conc;
architecture concurr_MUX41 of MUX_4_1_Conc is
begin
      Z <= A when S1='0' and S0='0' else
           B when S1='0' and S0='1' else
           C when S1='1' and S0='0' else
           D;
end concurr_MUX41;
```

Αναθέσεις επιλεγμένων σημάτων

```
with choice_expression select
      target_name <= expression when choices,
      target_name <= expression when choices,
            :
      target_name <= expression when choices;
```

- Η ανάθεση υπό συνθήκη μας επιτρέπει να καθορίσουμε ποια κυμματομορφή θα οδηγήσει το σήμα εξετάζοντας μία μόνο έκφραση.
- Το πλεονέκτημα της δομής είναι η περιγραφική της απλότητα.
- Η δομή είναι ευαίσθητη σε αλλαγή οποιουδήποτε σήματος είτε των εκφράσεων, είτε της έκφρασης επιλογής.
- Οι κανόνες που ισχύουν για την δομή case ισχύουν και στην ανάθεση επιλεγμένων σημάτων

Αναθέσεις επιλεγμένων σημάτων

```
with choice_expression select
      target_name <= expression when choices,
      target_name <= expression when choices,
            :
      target_name <= expression when choices;
```

```
entity MUX_4_1_Conc2 is
   port (A, B, C, D: in std_logic;
            SEL: in std_logic_vector(1 downto 0);
            Z: out std_logic);
   end MUX_4_1_Conc2;
architecture concurr_MUX41b of MUX_4_1_Conc2 is
begin
     with SEL select
           Z <= A when "00",
                B when "01",
                C when "10",
                D when "11";
end concurr_MUX41b;
```

Αναθέσεις επιλεγμένων σημάτων

```
entity FullAdd_Conc is
   port (A, B, C: in std_logic;
         sum, cout: out std_logic);
   end FullAdd_Conc;
architecture FullAdd_Conc of FullAdd_Conc is
     --define internal signal: vector INS of the input signals
     signal INS: std_logic_vector (2 downto 0);

begin
     --define the components of vector INS of the input signals
     INS(2) <= A;
     INS(1) <= B;
     INS(0) <= C;

     with INS select
          (sum, cout) <=  std_logic_vector'("00") when "000",
                          std_logic_vector'("10") when "001",
                          std_logic_vector'("10") when "010",
                          std_logic_vector'("01") when "011",
                          std_logic_vector'("10") when "100",
                          std_logic_vector'("01") when "101",
                          std_logic_vector'("01") when "110",
                          std_logic_vector'("11") when "111",
                          std_logic_vector'("11") when others;

end FullAdd_Conc;
```

Περιγραφή Κατασκευής (Structural)

- Στην περιγραφή κατασκευής ένα σύστημα περιγράφεται ως διασύνδεση components. Υπονοείται μία ιεραρχική δομή.
- Για κάθε χρησιμοποιούμενο component πρέπει να γίνουν τα ακόλουθα:
  1. Δήλωση του component στο σώμα αρχιτεκτονικής όπου θα χρησιμοποιηθεί.
  2. Instantiation του component όπου γίνεται και η σύνδεσή του με το περιβάλλον σύστημα.
- Η δήλωση του component είναι όμοια με την δήλωση της αντίστοιχης οντότητας.
- Το instantiation είναι ουσιαστικά η δημιουργία ενός αντιγράφου μίας οντότητας.
- Το πέρασμα παραμέτρων στις θύρες μπορεί να γίνει θεσιακά ή ονομαστικά.

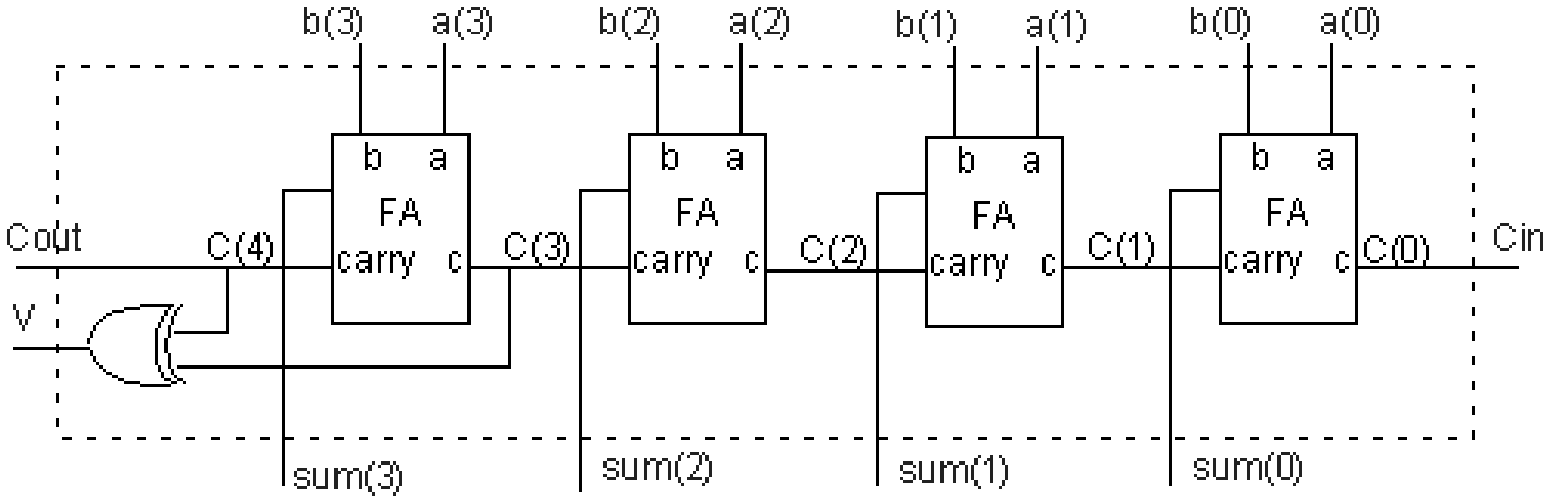
Περιγραφή Κατασκευής (Structural)

```
architecture Structural of TrafiicLights is
        component AND2
                port (in1, in2: in std_logic; out1: out std_logic);
        end component
        component OR2
                port (in1, in2: in std_logic; out1: out std_logic);
        end component
        component NOT1
                port (in1: in std_logic; out1: out std_logic);
        end component

signal VEH_AT_MAIN_INVERT, INT: std_logic;
begin
        U0: NOT1 port map (VEH_AT_MAIN, VEH_AT_MAIN_INVERT);
        U1: AND2 port map (VEH_AT_MAIN, VEH_AT_LOCAL, INT);
        U2: NOT1 port map (VEH_AT_MAIN_INVERT, INT, RED);
end Structural
```

Περιγραφή Κατασκευής (Structural)

Ιεραρχικοί σχεδιασμοί – Επαναληπτική χρήση Components

$$sum = (A \oplus B) \oplus C$$
$$carry = AB + C(A \oplus B)$$



Περιγραφή Κατασκευής (Structural)

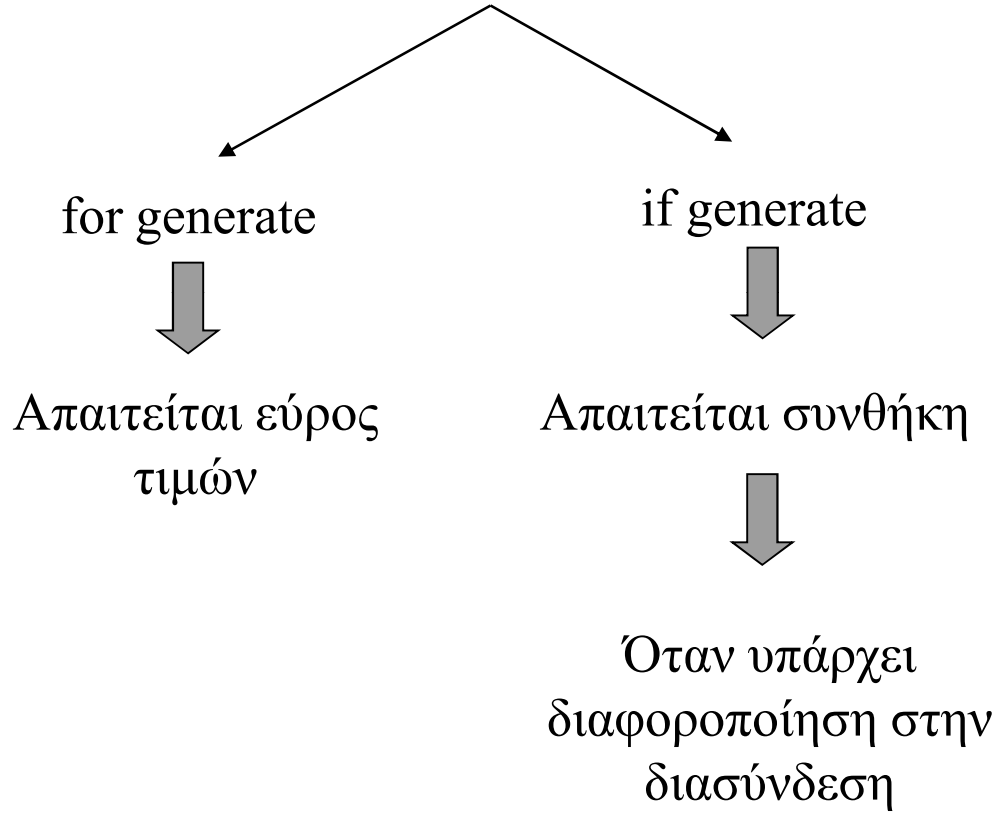
```
library  ieee;
use  ieee.std_logic_1164.all;
-- definition of a full adder
entity FULLADDER is
     port (a, b, c: in std_logic;
      sum, carry: out std_logic);
end FULLADDER;
architecture fulladder_behav of FULLADDER is
begin
      sum <= (a xor b) xor c ;
      carry <= (a and b) or (c and (a xor b));
end fulladder_behav;
-- 4-bit adder
library  ieee;
use  ieee.std_logic_1164.all;

entity FOURBITADD is
     port (a, b: in std_logic_vector(3 downto 0);
           Cin : in std_logic;
           sum: out std_logic_vector (3 downto 0);
           Cout, V: out std_logic);
end FOURBITADD;
```

Περιγραφή Κατασκευής (Structural)

```
architecture fouradder_structure of FOURBITADD is
      signal c: std_logic_vector (4 downto 0);
      component FULLADDER
            port(a, b, c: in std_logic;
                  sum, carry: out std_logic);
      end component;
begin
      FA0: FULLADDER
            port map (a(0), b(0), Cin, sum(0), c(1));
      FA1: FULLADDER
            port map (a(1), b(1), C(1), sum(1), c(2));
      FA2: FULLADDER
            port map (a(2), b(2), C(2), sum(2), c(3));
      FA3: FULLADDER
            port map (a(3), b(3), C(3), sum(3), c(4));
      V <= c(3) xor c(4);
      Cout <= c(4);
end fouradder_structure;
```

Προτάσεις παραγωγής (Generate)

- Βοηθάει στην γραφή συμπαγούς κώδικα σε περιπτώσεις κατασκευής ILAs.
- Αποφεύγουμε την δημιουργία πολλαπλών στιγμιοτύπων.



Προτάσεις παραγωγής (Generate)

```
-- 4-bit adder
library  ieee;
use  ieee.std_logic_1164.all;

entity FOURBITADD is
     port (a, b: in std_logic_vector(3 downto 0);
           Cin : in std_logic;
           sum: out std_logic_vector (3 downto 0);
           Cout, V: out std_logic);
end FOURBITADD;
```

Προτάσεις παραγωγής (Generate)

```
architecture fouradder_structure of FOURBITADD is
   component FULLADDER
         port (a,b,c: in std_logic; sum,carry: out std_logic);
   end component;
   signal c: std_logic_vector(4 downto 0);
begin
         c(0)<=Cin;
         g0: for i in 0 to 3 generate
                  u0: FULLADDER port map(a=>a(i), b=>b(i), c=>c(i),
                     sum=>sum(i), carry=>c(i+1));
         end generate;
         Cout<c(4);
end;
```

Προτάσεις παραγωγής (Generate)

Εναλλακτικά με *if generate*:

```
architecture fouradder_structure of FOURBITADD is
   component FULLADDER
            port (a,b,c: in std_logic; sum,carry: out std_logic);
   end component;
   signal c: std_logic_vector(3 downto 1);
begin
            g0: for i in 0 to 3 generate
                  g1: if i=0 generate
                        u0: FULLADDER port map(a(i), b(i), Cin, sum(i), c(i+1));
                  end generate g1;
                  g2: if i>0 and i<3 generate
                        u1: FULLADDER port map(a(i), b(i), c(i), sum(i), c(i+1));
                  end generate g2;
                  g3: if i=3 generate
                        u2: FULLADDER port map(a(i), b(i), c(i), sum(i), Cout);
                  end generate g3;
            end generate;
end;
```

Ανάθεση υποστοιχείων

- Όταν μία θύρα ενός component είναι σύνθετου τύπου τότε μπορούμε να αναθέσουμε ένα-ένα τα στοιχεία της θύρας.

```
entity reg is port (d:in bit_vector(7 downto 0); q:out bit_vector(7 downto 0); clk : in bit );
end entity reg;

…

architecture RTL of microprocessor is
  signal interrupt_req : bit;
  signal interrupt_level : bit_vector(2 downto 0);
  signal carry_flag, negative_flag, overflow_flag, zero_flag : bit;
  signal program_status : bit_vector(7 downto 0);
  signal clk_PSR : bit;
begin
  PSR : entity work.reg
    port map ( d(7) => interrupt_req, d(6 downto 4) => interrupt_level,  d(3) => carry_flag,
     d(2) => negative_flag,  d(1) => overflow_flag,  d(0) => zero_flag, q => program_status,
    clk => clk_PSR );
end architecture RTL;
```

Επεξεργασία Σχεδιασμού

- Οι στόχοι ενός σχεδιασμού είναι η εξομοίωση και πιθανώς η σύνθεση.
- Απαιτείται η δημιουργία προγράμματος εξομοίωσης ή/και netlist στοιχείων.
- Η εξομοίωση και η σύνθεση απαιτούν δύο προπαρασκευαστικά βήματα, την ανάλυση και την επεξεργασία.
- Η εξομοίωση απαιτεί την εκτέλεση του επεξεργασμένου μοντέλου.
- Η σύνθεση περιλαμβάνει την δημιουργία ενός netlist από πρωταρχικά στοιχεία της βιβλιοθήκης, το οποίο εκτελεί την ίδια λειτουργία με το επεξεργασμένο μοντέλο.

Ανάλυση

- Επιβεβαίωση ότι ο σχεδιασμός ακολουθεί τους συντακτικούς και εννοιολογικούς κανόνες.
- Μετάφραση της περιγραφής του σχεδιασμού σε μία εσωτερική μορφή πιο εύκολα επεξεργάσιμη που τοποθετείται στην βιβλιοθήκη σχεδιασμού (design library).
- Κάθε οντότητα (entity) θεωρείται κύρια σχεδιαστική μονάδα (primary design unit) και τοποθετείται στην βιβλιοθήκη ως μονάδα βιβλιοθήκης.
- Το σώμα αρχιτεκτονικής θεωρείται δευτερεύουσα σχεδιαστική μονάδα (secondary design unit). Εξαρτάται από την αντίστοιχη κύρια σχεδιαστική μονάδα και πρέπει να αναλύεται μετά από αυτήν.
- Η χρήση μονάδας (instantiation) ελέγχεται για αριθμό, τύπο, είδος θυρών.
- Μία σχεδιαστική μονάδα μπορεί να εξαρτάται από μία άλλη σχεδιαστική μονάδα (ιεραρχικότητα).
- Κάθε μονάδα εξαρτώμενη από άλλες πρέπει να αναλύεται μετά από αυτές.

Βιβλιοθήκες

- Χρησιμοποιούνται για την αποθήκευση σχεδιαστικών μονάδων. Μπορεί να είναι μία βάση δεδομένων ή ένας κατάλογος αρχείων.
- Μία από τις βιβλιοθήκες ονομάζεται βιβλιοθήκη εργασίας (working library) όπου αναλύεται και αποθηκεύεται ο σχεδιασμός μας.
- Για αναφορά στην βιβλιοθήκη σχεδιασμού χρησιμοποιείται η λέξη "**work**". Όμως η χρήση της είναι περιττή αφού υπονοείται.
- Για την αναφορά σχεδιαστικών μονάδων που βρίσκονται σε άλλες βιβλιοθήκες χρησιμοποιούμε την λέξη **library**.

```
library widget_cells, wasp_lib;

architecture cell_based of filter is
begin
    clk_pad : entity wasp_lib.in_pad
    port map ( i => clk, z => filter_clk );
    -- . . .
end architecture cell_based;
```

Κλήση μονάδας

Βιβλιοθήκες

Όταν γίνεται συχνή χρήση σχεδιαστικών μονάδων μίας βιβλιοθήκης είναι προτιμότερη η χρήση της έκφρασης **use** η οποία δίνει άμεση προσπέλαση σε όλα τα σχεδιαστικά τμήματα της βιβλιοθήκης..

library ieee;

use ieee.std_logic_1164.all } Δήλωση Βιβλιοθήκης - Πακέτου

std_logic_1164	Βασικοί τύποι δεδομένων	library ieee; use ieee.std_logic_1164.all; use ieee.std_logic_arith.all; use ieee.std_logic_unsigned.all;
std_logic_arith	Συναρτήσεις Αριθμητικές, Μετατροπής, Σύγκρισης για τους τύπους δεδομένων signed, unsigned, integer, std_ulogic, std_logic και std_logic_vector	
std_logic_unsigned	Συναρτήσεις μη προσημασμένων αριθμών	
std_logic_misc	Πρόσθετοι τύποι – συναρτήσεις	

Βιβλιοθήκες

```
-- Package declaration
library ieee;
use ieee.std_logic_1164.all;
package basic_func is
      -- AND2 declaration
      component AND2
            generic (DELAY: time :=5ns);
            port (in1, in2: in std_logic; out1: out std_logic);
      end component;
      -- OR2 declaration
      component OR2
            generic (DELAY: time :=5ns);
            port (in1, in2: in std_logic; out1: out std_logic);
      end component;
end package basic_func;
```

Βιβλιοθήκες

```
-- Package body declarations
library ieee;
use ieee.std_logic_1164.all;
package body basic_func is
      -- 2 input AND gate
      entity AND2 is
            generic (DELAY: time);
            port (in1, in2: in std_logic; out1: out std_logic);
      end AND2;
      architecture model_conc of AND2 is
            begin
                  out1 <= in1 and in2 after DELAY;
      end model_conc;
      -- 2 input OR gate
      entity OR2 is
            generic (DELAY: time);
            port (in1, in2: in std_logic; out1: out std_logic);
      end OR2;
      architecture model_conc2 of AND2 is
            begin
                  out1 <= in1 or in2 after DELAY;
      end model_conc2;
end package body basic_func;
```

Επεξεργασία (Elaboration)

- Στόχος της επεξεργασίας είναι να αφαιρεθεί η ιεραρχία και να δημιουργηθεί ένα επίπεδο σύνολο από διαδικασίες που ενώνονται με γραμμές διασύνδεσης.
- Είναι μία αναδρομική διαδικασία που ξεκινά από την κορυφαία οντότητα και κινείται προς τα κάτω:

1. Αρχικά δημιουργούνται οι θύρες της οντότητας και επιλέγεται το σώμα αρχιτεκτονικής.
2. Όλα τα components που χρησιμοποιούνται στο σώμα αποκτούν οντότητα και διασυνδέονται σε επίπεδο entities.
3. Αναδρομικά για κάθε component επιλέγεται σώμα αρχιτεκτονικής.
4. Η αναδρομή σταματά όταν φθάνουμε σε process ή σε primitive components.