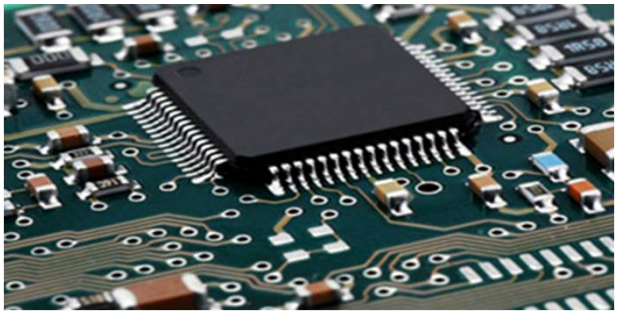
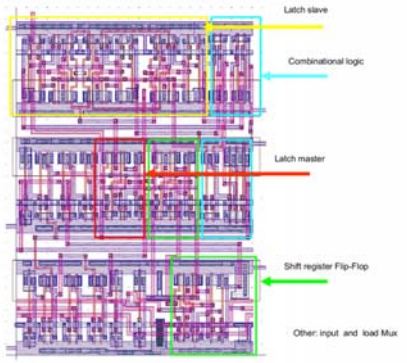
SCHOOL OF INFORMATICS





Ψηφιακή Σχεδίαση

Διάλεξη 8 – Γλώσσες Περιγραφής Υλικού (HDL)

Γεώργιος Κεραμίδας, Επίκουρος Καθηγητής

2ο Εξάμηνο, Τμήμα Πληροφορικής

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

Hardware Description Languages (HDLs)

- **Γλώσσες περιγραφής υλικού**
- **Γιατί υπάρχουν ?**
  - **Οι γλώσσες προγραμματισμού δε μας καλύπτουν**
  - **Το υλικό έχει μια εγγενή παραλληλία**
  - **Στις γλώσσες που ξέρουμε υποθέτουμε εκτέλεση σε μια ακολουθιακή μηχανή**
  - **Η εκτέλεση στο υλικό δεν είναι ακολουθιακής φύσης**
- **Γιατί χρειάζονται ?**
  - **Μεταφερσιμότητα μεταξύ τεχνολογιών**
  - **Κοινό υπόβαθρο συνεννόησης μεταξύ σχεδιαστών**
  - **Εύκολη μετάβαση από περιγραφή σε υλοποίηση**

Γλώσσες περιγραφής υλικού

- **Τι επιπλέον προσφέρουν ?**
  - **Αναπαραστάσεις σε διάφορες μορφές που χρησιμοποιεί ένας σχεδιαστής υλικού : λογικά διαγράμματα, συναρτήσεις Boole, FSMs …**
  - **Εξομοίωση (Simulation)**
  - **Σύνθεση (Synthesis)**

Εξομοίωση

- **Λογική Εξομοίωση (Logic Simulation)**
  - **Πιστοποίηση της σωστής δομής και συμπεριφοράς ενός κυκλώματος με τη χρήση υπολογιστή και κατάλληλου s/w, πριν τη πραγματική του υλοποίηση**
  - **Simulator (Εξομοιωτής)**
  - **Είσοδοι :**
    - **1) περιγραφή κυκλώματος σε HDL**
    - **2) διανύσματα εισόδου (stimuli)**
  - **Stimulus file (test bench)**
  - **Εξοδος : Τιμές εξόδων του κυκλώματος**

  - **Σε τι είναι γραμμένο ένα stimulus file ?**
  - **Φυσικά, σε HDL !!!**

Σύνθεση

- **Logic synthesis**
  - **Αυτοματοποιημένη διαδικασία παραγωγής του δικτυώματος (netlist) ενός κυκλώματος που έχουμε περιγράψει σε HDL, για κάποια συγκεκριμένη τεχνολογία. Το netlist είναι η λίστα των σχεδιαστικών κυττάρων που χρησιμοποιούμε καθώς και η διασύνδεσή τους για τη στοχευόμενη λειτουργικότητα.**
  - **Synthesizer / Synthesis tool**
  - **Είσοδοι :**
    - **1) περιγραφή κυκλώματος σε υποσύνολο της HDL**
    - **2) επιθυμητά χαρακτηριστικά λειτουργίας.**
  - **Εξοδος : Δικτύωμα του κυκλώματος.**

Παραλληλία στις HDLs

- **Κλασσικές γλώσσες προγραμματισμού :**
  - **A=B; C=A; => C=B**
  - **C=A; A=B; => C=A**
  - **Η σειρά αναγραφής των εντολών ΕΧΕΙ ΣΗΜΑΣΙΑ γιατί ο προγραμματιστής λαμβάνει υπόψη του το ακολουθιακό μοντέλο εκτέλεσης**
- **Αρχή της παραλληλίας στις HDLs :**
  - **A=B; C=A; => C=B**
  - **C=A; A=B; => C=B**
  - **Η σειρά αναγραφής των εντολών ΔΕΝ ΕΧΕΙ ΣΗΜΑΣΙΑ γιατί περιγράφουμε H/W που εκ φύσεως είναι παράλληλο.**

VHDL (1)

- **Ροή Σχεδίασης**
  - **Τα βήματα μιας σχεδίασης σε VHDL ομαδοποιούνται σε front-end και back-end**
- **Σχεδίαση**
  - **“Μικρά” κυκλώματα μπορούν να σχεδιασθούν με το κατάλληλο λογισμικό (παρόμοιο με το logicism). Μεγαλύτερα κυκλώματα μπορούν να σχεδιασθούν ιεραρχικά από μικρότερα κυκλώματα (δομικές μονάδες). Λεπτομέρειες της σχεδίασης αργότερα χρησιμοποιώντας τη γλώσσα VHDL (text based)**
- **Συγγραφή**
  - **Κώδικας VHDL περιγράφει επακριβώς τα δομικά στοιχεία, τον τρόπο αλληλεπίδρασης μεταξύ τους και λεπτομέρειες της εσωτερικής τους δομής**

VHDL (2)

- **Compilation**
  - **Ο VHDL compiler βρίσκει συντακτικά λάθη και λάθη διασύνδεσής με τμήματα του κώδικα από τον οποίο εξαρτάται. Επίσης, δημιουργούνται πληροφορίες για την φάση της προσομοίωσης του κυκλώματος**
- **Προσομοίωση**
  - **Στον VHDL Simulator ορίζονται τιμές για τις εισόδους του κυκλώματος και προσδιορίζονται οι τιμές των εξόδων, χωρίς να κατασκευαστεί το φυσικό κύκλωμα.**
  - **Σε μικρά κυκλώματα, οι είσοδοι δίνονται χειροκίνητα**
  - **Σε μεγάλα κυκλώματα, η VHDL παρέχει τη δυνατότητα δημιουργίας ενός test-bench, που θέτει αυτόματα διάφορες τιμές στις εισόδους και τις συγκρίνει με την αναμενόμενη τιμή εξόδου**

VHDL (3)

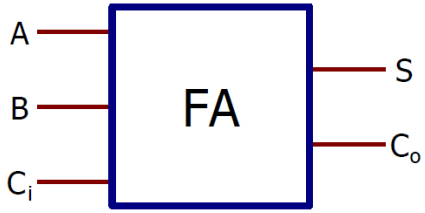
- **Verification**
  - **Η πιο σημαντική διαδικασία. Διεξοδικώς έλεγχος της ορθής λειτουργίας του κυκλώματος. Καθορισμός σεναρίων για έλεγχο του κυκλώματος σε μεγάλο εύρος λογικών συνθηκών**
  - **Functional Verification**
    - **Στο functional verification, η λειτουργία του κυκλώματος ελέγχεται ανεξάρτητα από το χρόνο. Όλοι οι παράμετροι χρόνου θεωρούνται μηδέν (π.χ. καθυστερήσεις στις πύλες)**
  - **Timing Verification**
    - **Ελέγχεται η λειτουργία του κυκλώματος λαμβάνοντας υπόψη την αναμενόμενη χρονική καθυστέρηση των πυλών και των flip-flops.**
    - **Συνήθως ολοκληρώνεται το functional simulation, προτού αρχίσουν τα βήματα που αναφέρονται σαν back-end**
    - **Περιορισμένη ακρίβεια στο timing simulation στο σημείο αυτό γιατί το αποτέλεσμα είναι ισχυρά εξαρτώμενο από τις επόμενες διαδικασίες της Σύνθεσης (synthesis) και της Προσαρμογής (Fitting)**

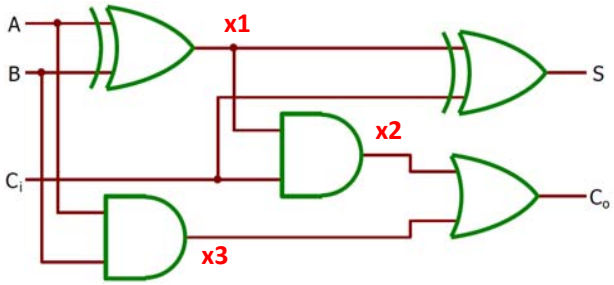
VHDL (4)

- **Σύνθεση (Synthesis)**
  - **Η περιγραφή του κυκλώματος στη VHDL μεταφράζεται σε δομικά στοιχεία που μπορούν να συναρμολογηθούν στην τεχνολογία για την οποία προορίζεται το κύκλωμα**
  - **Π.χ. για ένα FPGA δημιουργείται ένα σύνολο πυλών και ένα σύνολο από συνδέσεις (netlist) που περιγράφει πώς οι πύλες θα συνδεθούν μεταξύ τους**
- **Προσαρμογή (Fitting, Place & Route)**
  - **Κατάλληλο λογισμικό (fitter) προσαρμόζει τα δομικά στοιχεία που δημιουργήθηκαν στους διαθέσιμους πόρους της συσκευής για την οποία προορίζεται το κύκλωμα**
  - **Εισαγωγή περιορισμών για την τοποθέτηση των δομικών στοιχείων στη συσκευή και για τα pins εισόδων/εξόδων του κυκλώματος**
  - **Τελευταίο βήμα: timing verification με τους χρονικούς περιορισμούς που εισάγουν οι πύλες, το μήκος των καλωδίων, ο ηλεκτρικό φόρτος του κυκλώματος κτλ.**

Το Πρώτο Παράδειγμα

- **Παράδειγμα**
  - **Πλήρης Αθροιστής (Full Adder, FA)**
  - **Ονοματοδοσία εσωτερικών σημάτων**





$$x_1 = A \oplus B$$
$$x_2 = x_1 \cdot C_i$$
$$x_3 = A \cdot B$$
$$S = x_1 \oplus C_i$$
$$C_o = x_2 + x_3$$

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity fa_dataflow is
port (A,B,Ci: In std_logic;
S,Co: Out std_logic);
End fa_dataflow;

Architecture dataflow of fa_dataflow is
Signal x1,x2,x3: std_logic; --εσωτερικά σήματα
begin
   x1<=A xor B;
   x2<=x1 and Ci;
   x3<=A and B;
   S<=x1 xor Ci;
   Co<=x2 or x3;
End dataflow;
```

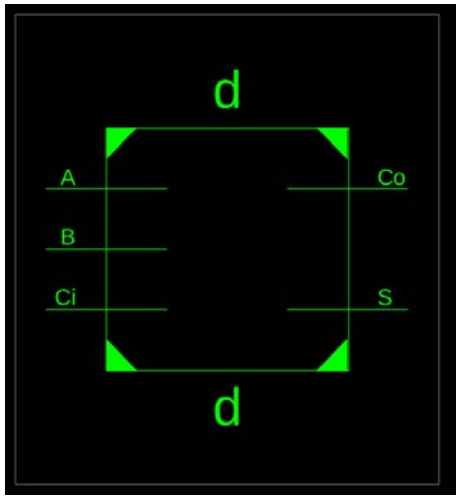
σχόλιο
(αγνοείται από τον compiler)

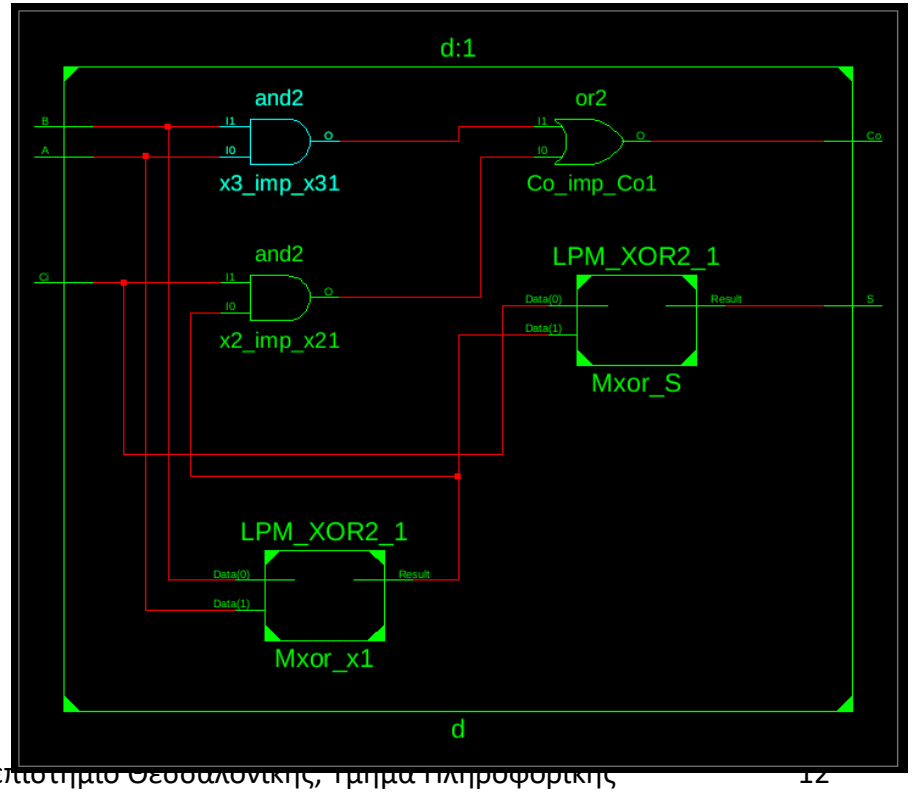
ή χωρίς εσωτερικά σήματα:

```
Architecture dataflow of fa_dataflow is
begin
   S<=(A xor B) xor Ci;
   Co<=((A xor B) and Ci) or (A and B);
End dataflow;
```

Το πρώτο παράδειγμα

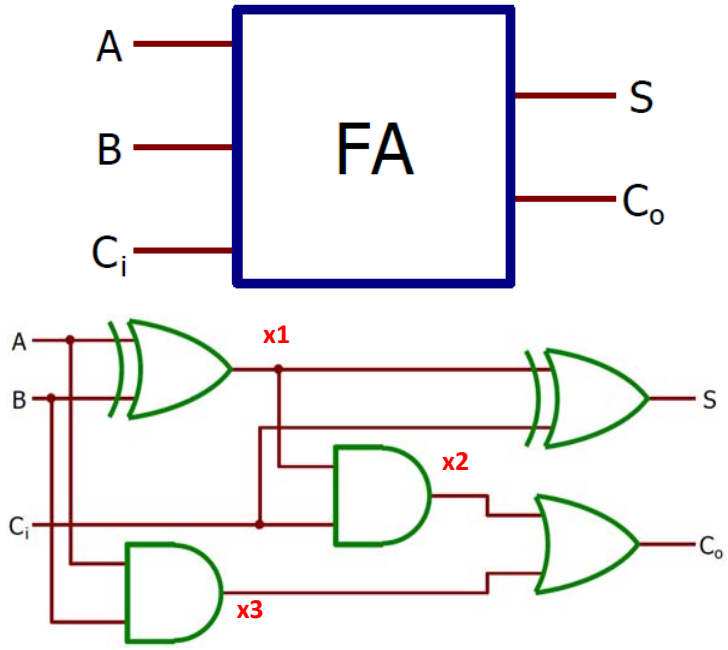
- **Εργαλείο -- Xilinx ISE 14.7**
- **Free to download**





Πρόσθεση Χρονοκαθυστερήσεων

after x ns = χρονοκαθυστέρηση Χ χρονικών μονάδων εξομοίωσης




```
32  entity d is
33      Port ( A : in  STD_LOGIC;
34             B : in  STD_LOGIC;
35             Ci : in  STD_LOGIC;
36             S : out  STD_LOGIC;
37             Co : out  STD_LOGIC);
38  end d;
39
40  architecture Behavioral of d is
41  signal x1, x2, x3: STD_LOGIC;
42  begin
43  x1 <= A xor B after 2ns;
44  x2 <= x1 and Ci after 1ns;
45  x3 <= A and B after 1ns;
46  S <= x1 xor Ci after 2ns;
47  Co <= x2 or x3 after 1ns;
48
49  end Behavioral;
```

Testbench & Simulation

Stimulus file (test bench)

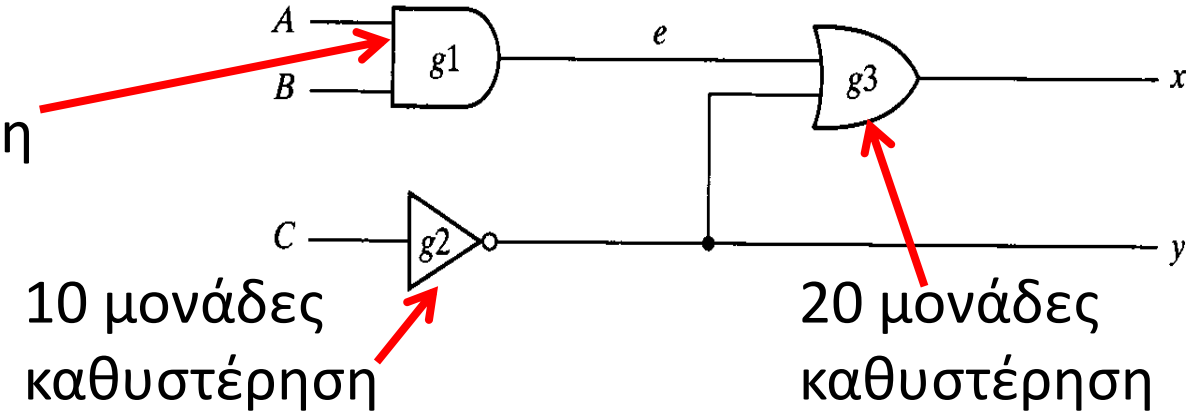
```
76    -- Stimulus process
77    stim_proc: process
78    begin
79       A<= '1';
80       B<= '0';
81       Ci<= '0';
82
83      -- hold reset state for 10 ns.
84       wait for 10 ns;
85
86       A<= '1';
87       B<= '0';
88       Ci<= '0';
89
90       wait;
91    end process;
```

κύκλωμα

```
32  entity d is
33      Port ( A : in  STD_LOGIC;
34             B : in  STD_LOGIC;
35             Ci : in  STD_LOGIC;
36             S : out  STD_LOGIC;
37             Co : out  STD_LOGIC);
38  end d;
39
40  architecture Behavioral of d is
41  signal x1, x2, x3: STD_LOGIC;
42  begin
43  x1 <= A xor B after 2ns;
44  x2 <= x1 and Ci after 1ns;
45  x3 <= A and B after 1ns;
46  S <= x1 xor Ci after 2ns;
47  Co <= x2 or x3 after 1ns;
48
49  end Behavioral;
```

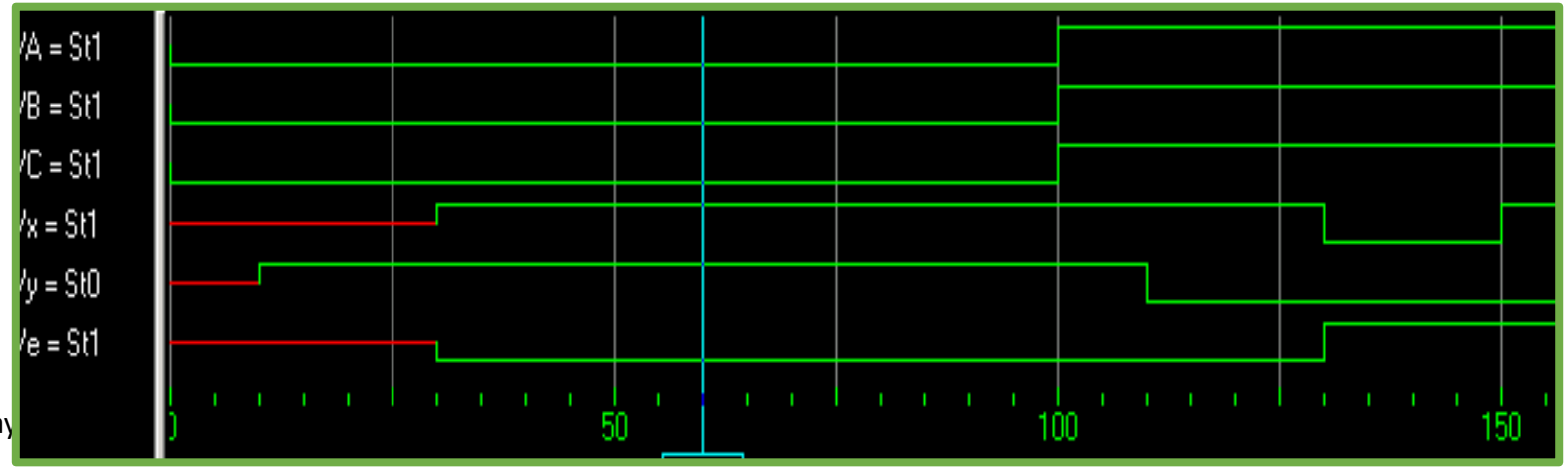
Testbench & Simulation

30 μονάδες καθυστέρηση



10 μονάδες καθυστέρηση

20 μονάδες καθυστέρηση



Δομή και Τρόποι Περιγραφής Κώδικα

- Ένα απλό πρόγραμμα
  - π.χ. πύλη AND

Δήλωση Βιβλιοθηκών (Library)

```
Library IEEE;
Use IEEE.std_logic_1164.all;
```

Δήλωση Οντότητας (Entity)

```
Entity gate_and is
port(a,b:In std_logic;
c:out std_logic);
End gate_and;
```

Δήλωση Αρχιτεκτονικής (Architecture)

```
Architecture and_arc of gate_and is
begin
c<=a and b;
End and_arc;
```

Libraries και Packages

- **Τα Libraries περιέχουν packages (components & functions) που μπορούν να χρησιμοποιηθούν από τον σχεδιαστή**
- **Τα packages είναι μια συλλογή από "κοινά" στοιχεία π.χ. components & functions για μη-προσημασμένους αριθμούς**
- **Συνήθως χρησιμοποιούμε το ieee Library και το std_logic_1164 package:**

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;
```

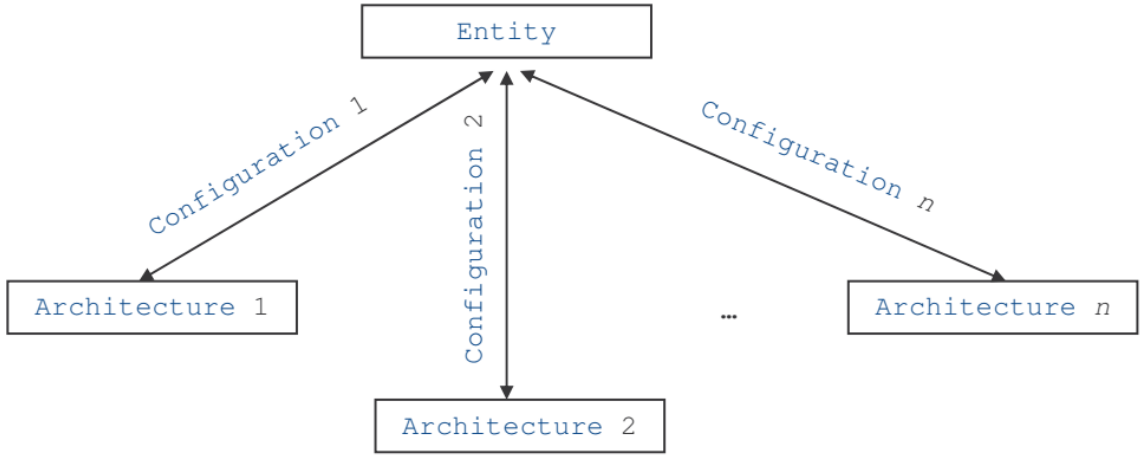
Libraries και Packages

- **Δήλωση Βιβλιοθηκών (Library)**
  - **Καθορίζουν τύπους δεδομένων, τελεστών και έτοιμων βαθμίδων που επιτρέπεται να χρησιμοποιηθούν**
  - **Κάθε βιβλιοθήκη περιέχει ορισμένα πακέτα (packages)**
- **Ενδεικτικά:**

Βιβλιοθήκη	Πακέτο	Στοιχεία
IEEE	std_logic_1164	λογικοί τελεστές (and, or, not, nand, xor, …), τελεστές σχέσεων (=, /=, <, <=, >, >=)
IEEE	std_logic_unsigned	επιτρέπει τη χρήση μη προσημασμένων αριθμών
IEEE	std_logic_signed	επιτρέπει τη χρήση προσημασμένων αριθμών
IEEE	std_logic_arith	τελεστές +, -, τελεστές σχέσεων (=, /=, <, <=, >, >=)

Entities, Architectures, Configurations

- **Ουσιαστικά η VHDL ακολουθεί τις αρχές του αντικειμενοστρεφούς προγραμματισμού**
- **Μια σχεδίαση σε VHDL αποτελείται από το εξωτερικό interface και το εσωτερικό implementation**
- **Μια σχεδίαση σε VHDL αποτελείται από entities, architectures και configurations**



Δήλωση Οντότητας (Entity)

- **Δηλώνονται τα εξωτερικά σήματα του συστήματος: όνομα, τύπος (mode) (είσοδος, έξοδος), τύπος δεδομένων (bit, vector, signed, unsigned)**

```
Entity όνομα_οντότητας is
port (όνομα_σήματος: τύπος_σήματος τύπος_δεδομένων;
…
όνομα_σήματος: τύπος_σήματος τύπος_δεδομένων);
End όνομα_οντότητας;
```

```
Entity gate_and is
port(a,b:In std_logic;
c:out std_logic);
End gate_and;
```

Τύπος Σήματος	Περιγραφή
In	σήμα εισόδου (δηλώνεται δεξιά του τελεστή <=) (π.χ. temp <= A)
Out	σήμα εξόδου (δηλώνεται αριστερά του τελεστή <=) (π.χ. B <= temp)
Buffer	σήμα εξόδου που μπορεί να δηλωθεί και από τις δύο πλευρές του τελεστή <= (π.χ. C <= C + 1)
Inout	σήμα εισόδου και εξόδου

Αρχιτεκτονική (Architecture)

- **Η αρχιτεκτονική είναι η υλοποίηση ενός entity**
- **Μπορεί να υπάρχουν πολλές αρχιτεκτονικές για ένα συγκεκριμένο entity**
- **Για παράδειγμα, κάθε αρχιτεκτονική να είναι βελτιστοποιημένη για ένα συγκεκριμένο στόχο σχεδίασης:**
  - **Performance**
  - **Area**
  - **Power Consumption**
  - **Ease of Simulation**

Δήλωση Αρχιτεκτονικής

- **Πρότυπο**

```
Architecture όνομα_αρχιτεκτονικής of όνομα_οντότητας is
begin
        …..
        παράλληλες δηλώσεις
        process δηλώσεις
         παράλληλες δηλώσεις
        process δηλώσεις
         παράλληλες δηλώσεις
        process δηλώσεις
         παράλληλες δηλώσεις
…..

End όνομα_αρχιτεκτονικής;
```

Σήματα (signals)

- **Δήλωση εσωτερικών σημάτων του κυκλώματος ή μετάδοση τιμών εσωτερικά και εξωτερικά του κυκλώματος (συνδέουν τα στοιχεία του κυκλώματος σαν 'καλώδια')**
- **Ωστόσο στην VHDL τα signals αναπαριστούν και καλώδια και στοιχεία μνήμης**
- **Μπορεί να είναι ορατά σε ολόκληρο τον κώδικα**

Signal όνομα_σήματος : **τύπος_δεδομένων**;

Απόδοση τιμής σε σήμα

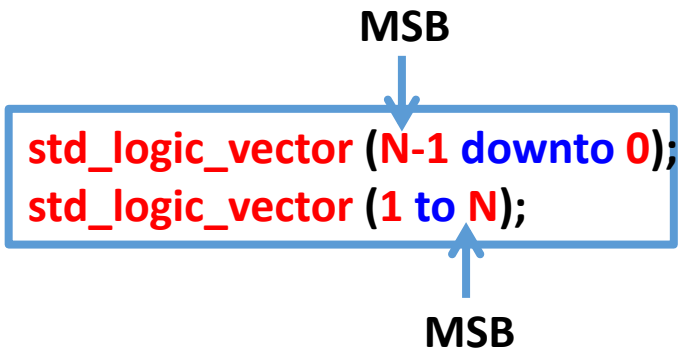
1-bit: με μονά εισαγωγικά

n-bit: με διπλά εισαγωγικά

```
signal y : std_logic;
signal counter : integer range 0 to 10;
signal reg : std_logic_vector (7 downto 0);

y <= '0'; --απόδοση 1-bit
counter <= 9; --απόδοση ακέραιου αριθμού
reg <= "11001010"; --απόδοση 8-bit διανύσματος
```

Τύπο Δεδομένων (Data Types)

- **std_logic**
  - Σήματα 1-bit
  - Απαιτεί κλήση της IEEE.std_logic_1164
  - Πολλές δυνατές τιμές, π.χ. **'0'** (λογικό 0), **'1'** (λογικό 1), **'Z'** (υψηλή εμπέδηση), **'X'** (αδιάφορη κατάσταση)



- **std_logic_vector**
  - Σήματα περισσοτέρων bits
  - Απαιτεί κλήση της IEEE.std_logic_1164
  - Αντίστοιχες τιμές, π.χ. **"0X1Z1"**, **"ZZZ"**, **"XXXX"**

```
Library IEEE;
Use IEEE.std_logic_1164.all;
...
Signal A : std_logic; -- 1-bit signal
Signal B : std_logic_vector (4 downto 0); -- 5-bit signal
...
A <= '0';
B <= "10100";
```

resolved και unresolved τύποι δεδομένων

- **std_ulogic, std_ulogic_vector**
  - **Παρόμοια με τα std_logic και std_logic_vector (δίνουν λάθος όταν διαφορετικές πηγές τους αποδίδουν διαφορετικές τιμές)**
- **resolved και unresolved τύποι δεδομένων**
  - **Σήματα τύπου unresolved δεν μπορούν να οδηγηθούν (δηλωθούν) από περισσότερες από μία διαδικασίες (σε αντίθεση με σήματα τύπου resolved)**

unresolved τύπος

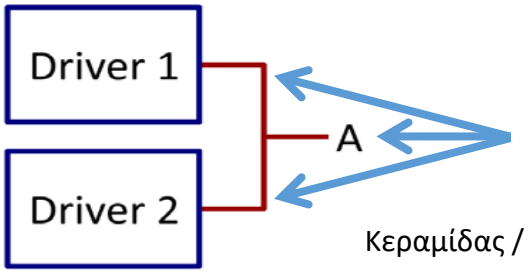
```
Signal A : std_ulogic := '0';
A <= '0'; -- Driver 1
A <= '1' after 10 ns; --Driver 2
```

δίνει λάθος`

δεν δίνει λάθος

resolved τύπος

```
Signal A : std_logic := '0';
A <= '0'; -- Driver 1
A <= '1' after 10 ns; --Driver 2
```



τα σήματα πρέπει να είναι τύπου **resolved**

Build-In Data Types

- **Παρέχονται από την ίδια την VHDL. Οχι σαν μέρος της IEEE.std_logic_1164 βιβλιοθήκης**

Data Type	Characteristics
BIT	Binary, Unresolved
BIT_VECTOR	Binary, Unresolved, Array
INTEGER	Binary, Unresolved, Array
REAL	Floating Point

- **Δεν είναι κατάλληλα για synthesis**
- **Συνήθως τα χρησιμοποιούμε εσωτερικά σε ένα architecture και όχι σαν «εξωτερικά» pins**

Bit & Bit_vector

- **bit, bit_vector**
- **Σήματα 1-bit (‘0’ ή ‘1’) ή περισσοτέρων bits (π.χ. “01101”)**
- **Δεν απαιτούν κλήση βιβλιοθήκης**

```
signal f : bit;
signal w1 : bit_vector (3 downto 0);
...
f <= '1';
w1 <= "1100";
```

Synthesis και Simulation

- **Μια υλοποίηση που μπορεί να γίνει synthesis ονομάζεται synthesizable design**
- **Οτιδήποτε γράφουμε στην VHDL μπορεί να γίνει simulation αλλά δεν είναι αναγκαστικά synthesizable design**
- **Παράδειγμα:**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY simple_buffer IS
      PORT (   din      : IN     std_logic;
               dout     : OUT    std_logic );
END simple_buffer;

ARCHITECTURE behavioural1 OF simple_buffer IS
BEGIN
      dout <= din AFTER 10 ns;
END behavioural1;
```

Synthesis και Simulation

- **Η είσοδος din γίνεται assign στο dout μετά από 10ns**
- **Αυτή η αρχιτεκτονική γίνεται simulation αλλά όχι synthesis**
- **Το τι γίνεται synthesis διαφέρει ανάλογα με το tool**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY simple_buffer IS
     PORT (   din      : IN     std_logic;
              dout     : OUT    std_logic );
END simple_buffer;

ARCHITECTURE behavioural1 OF simple_buffer IS
BEGIN
     dout <= din AFTER 10 ns;
END behavioural1;
```

Λογικοί Τελεστές και Λογικές πράξεις

- **Λογικοί Τελεστές (logical) και Λογικές πράξεις**
  - **Τύποι δεδομένων: bit, bit_vector, std_logic, std_logic_vector, std_ulogic, std_ulogic_vector**

Λογική Πράξη	Τελεστής	Παράδειγμα
NOT	NOT	`Y <= NOT (A);`
AND	AND	`Y <= (A AND B);`
OR	OR	`Y <= (A OR B);`
NAND	NAND	`Y <= (A NAND B);`
NOR	NOR	`Y <= (A NOR B);`
XOR	XOR	`Y <= (A XOR B);`
XNOR	XNOR	`Y <= (A XNOR B);`

- **Στην VHDL μπορούμε φτιάξουμε και τους δικού μας operators (παρόμοια με το overloading στις αντικειμενοστραφείς γλώσσες προγραμματισμού)**

Τελεστές Ανάθεσης και Συνένωσης

- **Ανάθεση (assignment)**
  - **Απόδοση τιμής**

Τελεστής	Ανάθεση τιμής σε	Παράδειγμα
<=	σήματα	`x <= a+b;`
:=	μεταβλητές και αρχικές συνθήκες σε σήματα	`y := "001";`
=>	στοιχεία πινάκων	`z := (0=>'1', 1=>'0', others=>'0');`

- **Συνένωσης (concatenation)**
  - **&: ομαδοποιεί τιμές**

```
Signal a,b : std_logic_vector (2 downto 0);
Signal f,g : std_logic_vector (5 downto 0);
a <= "110";
b <= "001";
f <= a & b; --f="110001"
g <= ('0','1','0','1','0','1'); --g="010101"
```

Παραδείγματα Ανάθεσης Τιμής

```
SIGNAL a, b, c              : std_logic;
SIGNAL avec, bvec, cvec     : std_logic_vector(7 DOWNTO 0);

-- Concurrent Signal Assignment Statements
-- NOTE: Both a and avec are produced concurrently
a         <= b AND c;
avec      <= bvec OR cvec;

-- Alternatively, signals may be assigned constants
a         <= '0';
b         <= '1';
c         <= 'Z';
avec      <= "00111010";              -- Assigns 0x3A to avec
bvec      <= X"3A";                   -- Assigns 0x3A to bvec
cvec      <= X"3" & X"A";             -- Assigns 0x3A to cvec
```

Εκτελούνται ταυτόχρονα

Παραδείγματα Ανάθεσης Τιμής

```
SIGNAL a, b, c, d          :std_logic;
SIGNAL avec                :std_logic_vector(1 DOWNTO 0);
SIGNAL bvec                :std_logic_vector(2 DOWNTO 0);

-- Conditional Assignment Statement
-- NOTE: This implements a tree structure of logic gates
a <=     '0'       WHEN avec = "00" ELSE
         b         WHEN avec = "11" ELSE
         c         WHEN d = '1' ELSE
         '1';

-- Selected Signal Assignment Statement
-- NOTE: The selection values must be constants
bvec <=  d & avec;
WITH bvec SELECT
a <=     '0'       WHEN "000",
         b         WHEN "011",
         c         WHEN "1--",       -- Some tools won't synthesize '-' properly
         '1'       WHEN OTHERS;
```

Ανάθεση υπό συνθήκη (when…else)

Ανάθεση υπό συνθήκη (with…select…when). Οι selected τιμές πρέπει να είναι σταθερές

Παραδείγματα Ανάθεσης Τιμής

```
SIGNAL a                    :std_logic;
SIGNAL avec, bvec           :std_logic_vector(7 DOWNTO 0);

-- Selected Signal Assignment Statement
-- NOTE: Selected signal assignments also work
--       with vectors
WITH a SELECT
avec <=  "01010101"         WHEN '1',
         bvec               WHEN OTHERS;
```

Ίδιες πράξεις με vector

Δήλωση Process

- **Συγκεκριμένα κομμάτια του design τα βάζουμε σε μια δήλωση process**
- **Περιλαμβάνουν το σύνολο των ακολουθιακών δηλώσεων του design**
- **Το εσωτερικό της εκτελείται ακολουθιακά, η ίδια όμως αποτελεί παράλληλη δήλωση (αν υπάρχουν πολλές δηλώσεις process, αυτές εκτελούνται παράλληλα χωρίς να παίζει ρόλο η σειρά τους)**

```
ετικέτα:    Process (όνομα_σήματος, όνομα_σήματος, …)
            begin
                  απόδοση τιμών σε σήματα
                  ακολουθιακές δηλώσεις
                  if δήλωση
                  case δήλωση
                  for loop δήλωση
                  ακολουθιακές δηλώσεις
            End process ετικέτα;
```

Οποιαδήποτε μεταβολή τους καθιστά ενεργή τη δήλωση process (εκτελείται ακολουθιακά το περιεχόμενό της) **(αποτελούν τη sensitivity-list)**

Δήλωση Process και Sensitivity List

- **Sensitivity list → σύνολο σημάτων και ports που μπορεί να αλλάξουν το output του process**
- **Παράδειγμα: ένα edge triggered flip-flop πυροδοτείται (είναι sensitive) στην ανερχόμενη παρυφή του ρολογιού**

```
ετικέτα:    Process (όνομα_σήματος, όνομα_σήματος, …)
            begin
                  απόδοση τιμών σε σήματα
                  ακολουθιακές δηλώσεις
                  if δήλωση
                  case δήλωση
                  for loop δήλωση
                  ακολουθιακές δηλώσεις
            End process ετικέτα;
```

Οποιαδήποτε μεταβολή τους καθιστά ενεργή τη δήλωση process (εκτελείται ακολουθιακά το περιεχόμενό της) **(αποτελούν τη sensitivity-list)**

Δήλωση Process και Δηλώσεις Ανάθεσης

- **Υπάρχουν διαφορετικοί τρόποι να κάνουμε αναθέσεις τιμών σε σήματα → διαφορετικά keywords εντός και εκτός μιας process**

Outside Processes	Inside Processes
WHEN..ELSE	IF..ELSIF..ELSE..END IF
WITH..SELECT..WHEN	CASE..WHEN..END CASE

- **Γενικός Κανόνας: Τα processes μπορούν να χρησιμοποιηθούν για την υλοποίηση συνδυαστικών κυκλωμάτων, αλλά ουσιαστικά χρησιμοποιούνται για την υλοποίηση ακολουθιακών κυκλωμάτων**

Δηλώσεις εντός μιας Process

- **Υλοποίηση D flip-flop με ασύγχρονο active low reset**

```
SIGNAL reset, clock, d, q          :std_logic;

PROCESS (reset, clock)
-- reset and clock are in the sensitivity list to
-- indicate that they are important inputs to the process
BEGIN
    -- IF keyword is only valid in a process
    IF (reset = '0') THEN
          q <= 0;
    ELSIF (clock'EVENT AND clock = '1') THEN
          q <= d;
    END IF;
END PROCESS;
```

Τα reset και clock είναι τα μόνα που μπορούν να ενεργοποιήσουν την process. Διαφορετικά το process βρίσκεται σε κατάσταση αναμονής.

Το EVENT είναι true όταν υπάρχει αλλαγή στο αντίστοιχο σήμα

Δηλώσεις εντός μιας Process

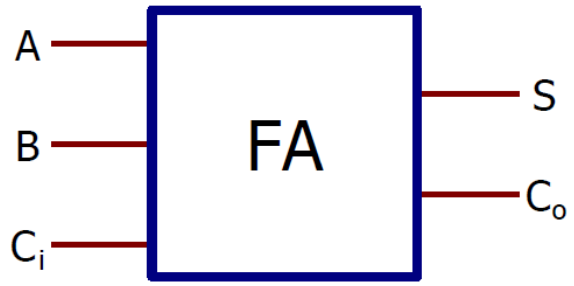
- **Υλοποίηση ενός συνδυαστικού κυκλώματος**

```
SIGNAL a, b, c, d          :std_logic;

PROCESS (a, b, d)
-- a, b, and d are in the sensitivity list to indicate that
-- the outputs of the process are sensitive to changes in them
BEGIN
   -- CASE keyword is only valid in a process
   CASE d IS
        WHEN '0' =>
                 c <= a AND b;
        WHEN OTHERS =>
                 c <= '1';
   END CASE;
END PROCESS;
```

Όχι με (with…select…when). Νοηματικά είναι το ίδιο, αλλά το (with…select…when) δεν μπορεί να χρησιμοποιηθεί μέσα σε process

Παράδειγμα Συνδυαστικού Κυκλώματος

- **Πλήρης Αθροιστής (Full Adder, FA)**
  - **Σε μορφή πίνακα αληθείας**



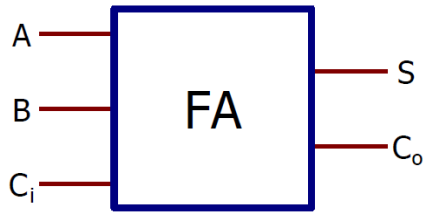
A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

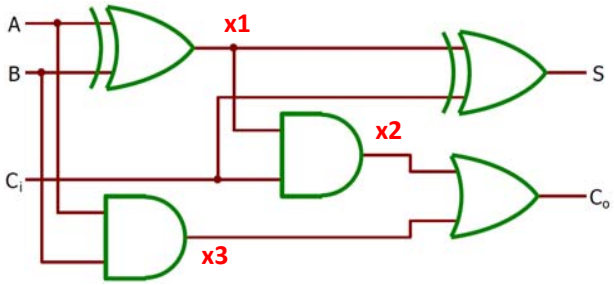
Ουσιαστικά δεν υλοποιούμε το κύκλωμα σε επίπεδο πυλών αλλά περιγράφουμε την συμπεριφορά του

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity fa_behavioral is
port (A,B,Ci: In std_logic;
S,Co: Out std_logic);
End fa_behavioral;

Architecture behavioral of fa_behavioral is
begin

Process (A,B,Ci)
begin
   If (A='0' and B='0' and Ci='0') then
      S<='0';
      Co<='0';
   elsif (A='0' and B='0' and Ci='1') then
      S<='1';
      Co<='0';
   elsif (A='0' and B='1' and Ci='0') then
      S<='1';
      Co<='0';
   elsif (A='0' and B='1' and Ci='1') then
      S<='0';
      Co<='1';
   elsif (A='1' and B='0' and Ci='0') then
      S<='1';
      Co<='0';
   elsif (A='1' and B='0' and Ci='1') then
      S<='0';
      Co<='1';
   elsif (A='1' and B='1' and Ci='0') then
      S<='0';
      Co<='1';
   elsif (A='1' and B='1' and Ci='1') then
      S<='1';
      Co<='1';
   else
      null;
   End if;
End process;

End behavioral;
```

Και χωρίς Process

- **Παράδειγμα**
  - **Πλήρης Αθροιστής (Full Adder, FA)**
  - **Ονοματοδοσία εσωτερικών σημάτων**





$$x_1 = A \oplus B$$
$$x_2 = x_1 \cdot C_i$$
$$x_3 = A \cdot B$$
$$S = x_1 \oplus C_i$$
$$C_o = x_2 + x_3$$

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity fa_dataflow is
port (A,B,Ci: In std_logic;
S,Co: Out std_logic);
End fa_dataflow;

Architecture dataflow of fa_dataflow is
Signal x1,x2,x3: std_logic; --εσωτερικά σήματα
begin
   x1<=A xor B;
   x2<=x1 and Ci;
   x3<=A and B;
   S<=x1 xor Ci;
   Co<=x2 or x3;
End dataflow;
```

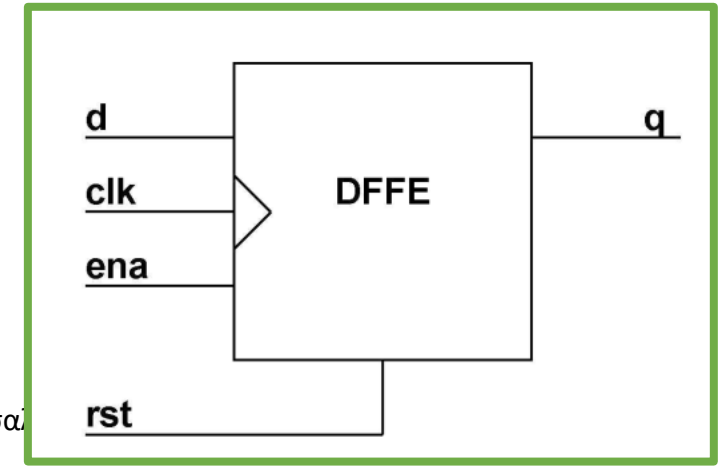
σχόλιο
(αγνοείται από τον compiler)

ή χωρίς εσωτερικά σήματα:

```
Architecture dataflow of fa_dataflow is
begin
   S<=(A xor B) xor Ci;
   Co<=((A xor B) and Ci) or (A and B);
End dataflow;
```

Παράδειγμα: D Flip-Flop

- **Βασικά στοιχεία:**
  - **Χρήση του EVENT και χρήση process**
  - **Process: χρησιμοποιείται για να κάνουμε το στοιχείο μνήμης sensitive στο ρολόι**
  - **EVENT: χρησιμοποιείται για να κάνουμε το στοιχείο μνήμης sensitive στην ανερχόμενη ή κατεχόμενη παρυφή του ρολογιού**



VHDL Υλοποίηση ενός D Flip-Flop

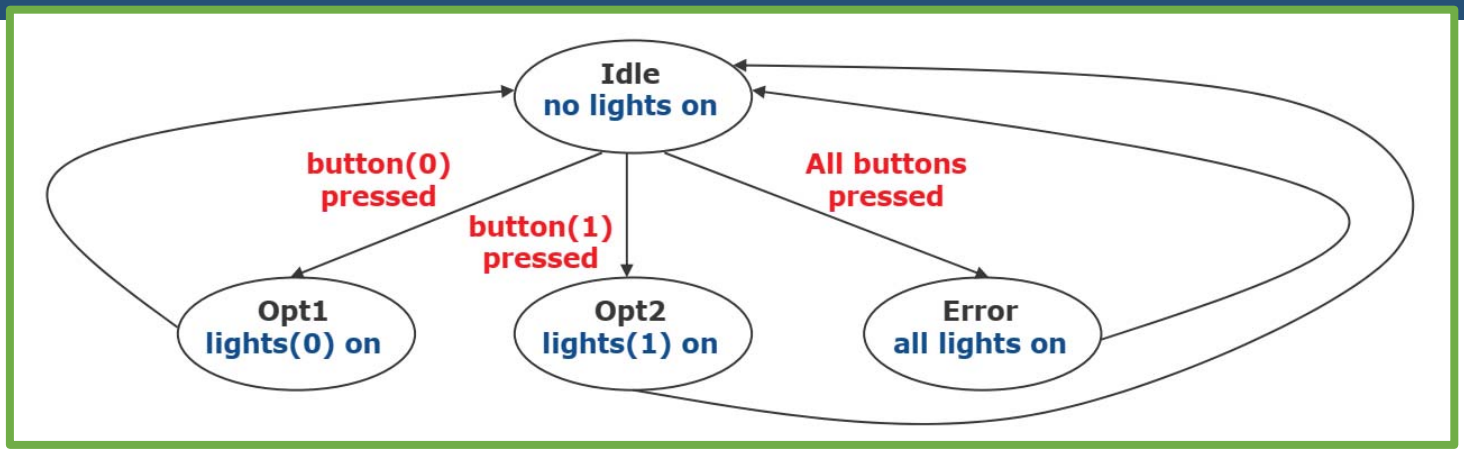
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY dffe IS
    PORT( rst, clk, ena, d : IN     std_logic;
          q                : OUT    std_logic );
END dffe;

ARCHITECTURE synthesis1 OF dffe IS
BEGIN
    PROCESS (rst, clk)
    BEGIN
          IF (rst = '1') THEN
                    q <= '0';
          ELSIF (clk'EVENT) AND (clk = '1') THEN
                    IF (ena = '1') THEN
                              q <= d;
                    END IF;
          END IF;
    END PROCESS;
END synthesis1;
```

D Flip-Flop

Ακολουθιακά Κυκλώματα

- Όπως έχουμε αναφέρει πολύπλοκα ακολουθιακά κυκλώματα σχεδιάζονται με τα διαγράμματα καταστάσεων (FSMs ή Finite State Machines)
- Τα FSMs υλοποιούνται με processes και με χρήση του CASE
- Ακολουθούν παραδείγματα

Παράδειγμα FSM




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY vending IS
   PORT(
         reset             : IN     std_logic;
         clock             : IN     std_logic;
         buttons           : IN     std_logic_vector(1 DOWNTO 0);
         lights            : OUT    std_logic_vector(1 DOWNTO 0)
   );
END vending;
```

Παράδειγμα FSM

```
ARCHITECTURE synthesis1 OF vending IS
   TYPE          statetype IS (Idle, Opt1, Opt2, Error);
   SIGNAL        currentstate, nextstate   : statetype;
BEGIN
   fsm1: PROCESS( buttons, currentstate )
   BEGIN
        CASE currentstate IS
                WHEN Idle =>
                        lights <= "00";
                        CASE buttons IS
                                WHEN "00" =>
                                        nextstate <= Idle;
                                WHEN "01" =>
                                        nextstate <= Opt1;
                                WHEN "10" =>
                                        nextstate <= Opt2;
                                WHEN OTHERS =>
                                        nextstate <= Error;
                        END CASE;
                WHEN Opt1 =>
                        lights <= "01";
                        IF buttons /= "01" THEN
                                nextstate <= Idle;
                        END IF;
```

Για κάθε currentstate καθορίζουμε την επόμενη κατάσταση

ΣΥΝΕΧΕΙΑ

Παράδειγμα FSM

```
                WHEN Opt2 =>
                        lights <= "10";
                        IF buttons /= "10" THEN
                                nextstate <= Idle;
                        END IF;
                WHEN Error =>
                        lights <= "11";
                        IF buttons = "00" THEN
                                nextstate <= Idle;
                        END IF;
        END CASE;
    END PROCESS;

    fsm2: PROCESS( reset, clock )
    BEGIN
        IF (reset = '0') THEN
                currentstate <= Idle;
        ELSIF (clock'EVENT) AND (clock = '1') THEN
                currentstate <= nextstate;
        END IF;
    END PROCESS;
END synthesis1;
```

Δεύτερο process που ενεργοποιεί το πρώτο μέσω του σήματος currentstate

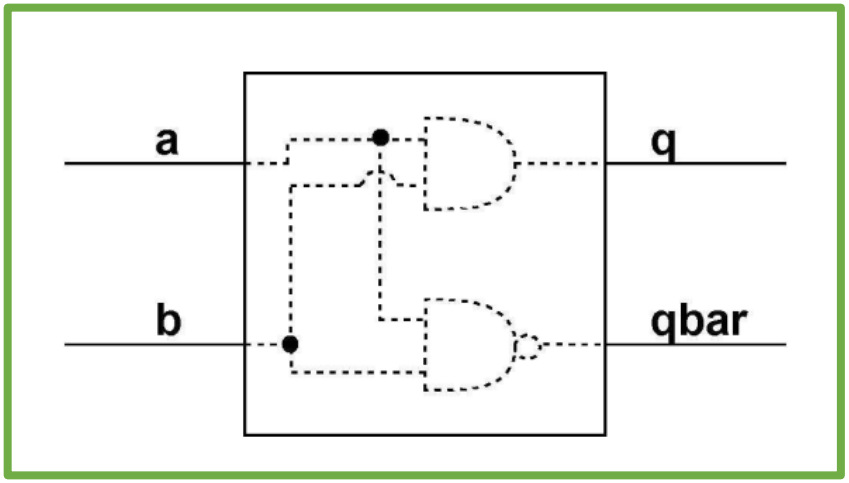
Επιπλέον Παραδείγματα

- Συνδυαστικό κύκλωμα (Παράδειγμα 1)
- Συνδυαστικό κύκλωμα με MUX (Παράδειγμα 2)
- Οδηγός 7-segment display (Παράδειγμα 3)
- Ακολουθιακό κύκλωμα (καταχωρητής) (Παράδειγμα 4)
- Ακολουθιακό κύκλωμα (μετρητής) (Παράδειγμα 5)

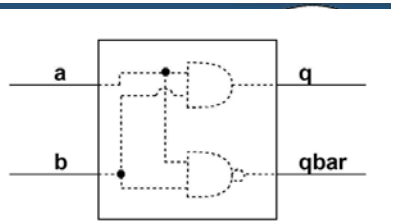
- Όλες οι υλοποιήσεις που ακολουθούν οδηγούν σε "synthesizable designs"

Παράδειγμα 1 – Εκφώνηση

- **Υλοποίηση του παρακάτω κυκλώματος. Χρησιμοποιείστε std_logic για τους τύπους των σημάτων εισόδου-εξόδου**



Παράδειγμα 1 – Λύση



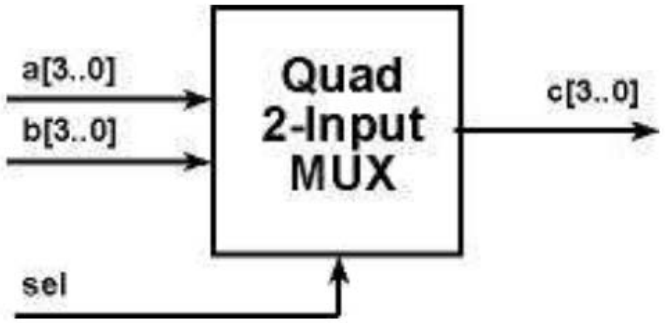

```
ENTITY andnand IS
   PORT (      a       : IN   std_logic;
               b       : IN   std_logic;
               q       : OUT  std_logic;
               qbar    : OUT  std_logic );
   END andnand;
```

```
ARCHITECTURE synthesis1 OF andnand IS
BEGIN
   q    <= a AND b;
   qbar<= a NAND b;
END synthesis1;
```

Ταυτόχρονη εκτέλεση.
Αν το είχαμε μέσα σε process, δεν θα μπορούσαμε να το πετύχουμε αυτό

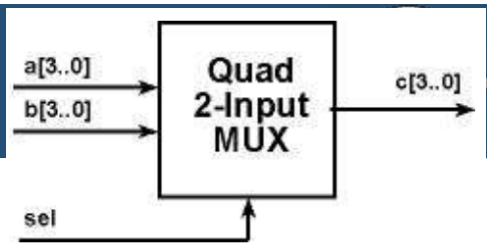
Παράδειγμα 2 – Εκφώνηση

- **Σχεδιάστε ένα VHDL entity που να υλοποιεί ένα 2-input MUX**



Port	Type	Width	Direction
a	std_logic_vector	4	IN
b	std_logic_vector	4	IN
sel	std_logic	1	IN
c	std_logic_vector	4	OUT

Παράδειγμα 2 – Λύση

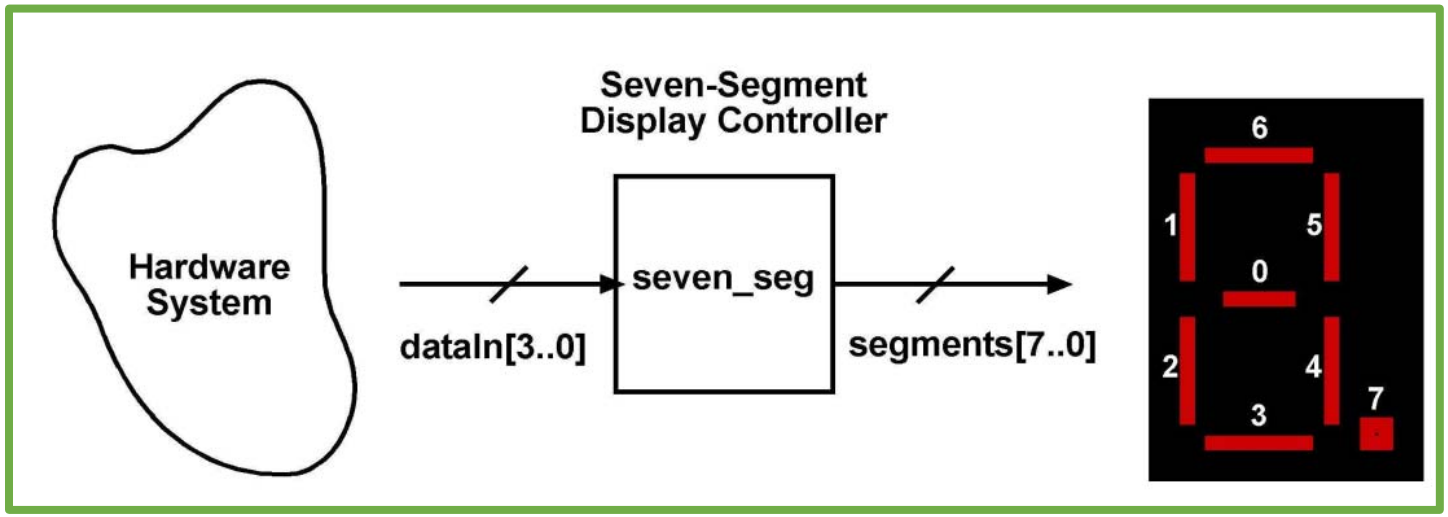



```
ENTITY quadmux IS
   PORT (     a      : IN   std_logic_vector(3 DOWNTO 0);
              b      : IN   std_logic_vector(3 DOWNTO 0);
              sel    : IN   std_logic;
              c      : OUT  std_logic_vector(3 DOWNTO 0) );
   END quadmux;
```

```
ARCHITECTURE synthesis1 OF quadmux IS
BEGIN
    WITH sel SELECT
    c   <=     a WHEN '0',
               b WHEN OTHERS;
END synthesis1;
```

Παράδειγμα 3 – Εκφώνηση

- **Σχεδιάστε έναν οδηγό για 7-segment display σύμφωνα με το παρακάτω σχήμα (το ON είναι active low)**



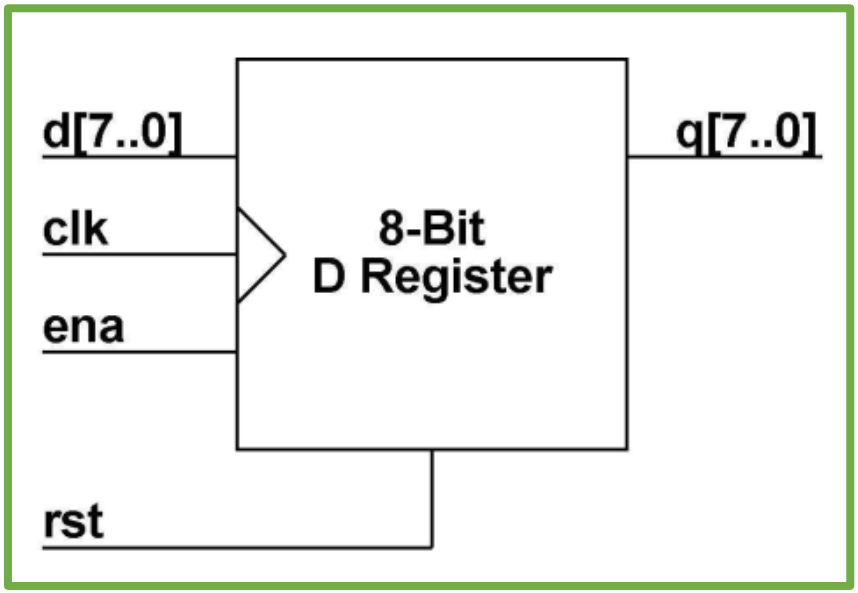
Παράδειγμα 3 – Λύση

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY seven_seg IS
   PORT( dataIn          : IN     std_logic_vector(3 DOWNTO 0);
         segments        : OUT    std_logic_vector(7 DOWNTO 0) );
END seven_seg;

ARCHITECTURE synthesis1 OF seven_seg IS
BEGIN
   WITH dataIn SELECT
         segments <=     "10000001" WHEN "0000",   -- 0
                         "11001111" WHEN "0001",   -- 1
                         "10010010" WHEN "0010",   -- 2
                         "10000110" WHEN "0011",   -- 3
                         "11001100" WHEN "0100",   -- 4
                         "10100100" WHEN "0101",   -- 5
                         "10100000" WHEN "0110",   -- 6
                         "10001111" WHEN "0111",   -- 7
                         "10000000" WHEN "1000",   -- 8
                         "10000100" WHEN "1001",   -- 9
                         "11111111" WHEN OTHERS;
END synthesis1;
```

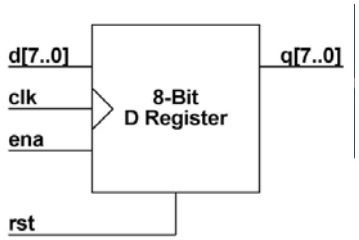
Σιγά-σιγά φεύγω από τις λεπτομέρειες του κυκλώματος.
Για παράδειγμα, δεν με νοιάζει πως θα γίνει η υλοποίηση σε πύλες

Παράδειγμα 4 – Εκφώνηση

- **Σχεδιάστε σε VHDL έναν 8-bit register με σήμα enable και ασύγχρονο reset**



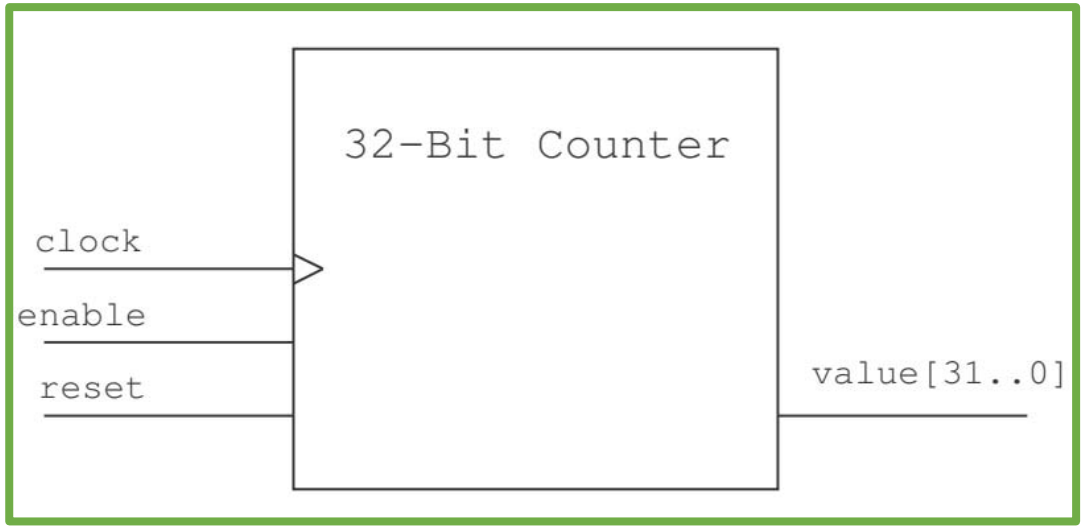
Παράδειγμα 4 – Λύση




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY dregister IS
   PORT( rst, clk, ena    : IN     std_logic;
         d                : IN     std_logic_vector(7 DOWNTO 0);
         q                : OUT    std_logic_vector(7 DOWNTO 0) );
END dregister;

ARCHITECTURE synthesis1 OF dregister IS
BEGIN
   PROCESS (rst, clk)
   BEGIN
         IF (rst = '1') THEN
                  q <= X"00";
         ELSIF (clk'EVENT) AND (clk = '1') THEN
                  IF (ena = '1') THEN
                           q <= d;
                  END IF;
         END IF;
   END PROCESS;
END synthesis1;
```

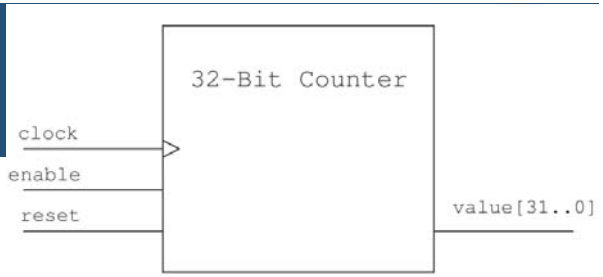
8-bit D Flip-Flop

Παράδειγμα 5 – Εκφώνηση



- **Σχεδιάστε σε VHDL έναν 32-bit μετρητή με σήμα enable και ασύγχρονο reset**

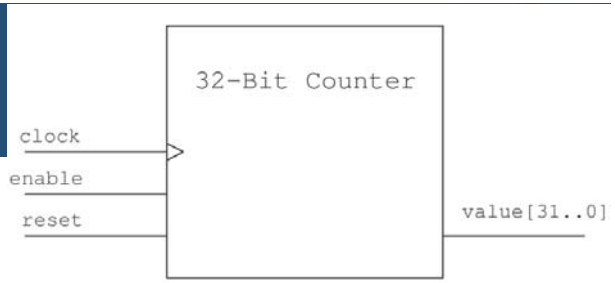
Παράδειγμα 5 – Λύση




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY counter IS
   PORT(
         reset             : IN     std_logic;
         clock             : IN     std_logic;
         enable            : IN     std_logic;
         value             : OUT    std_logic_vector(31 DOWNTO 0)
   );
END counter;
```

Δήλωση entity

Παράδειγμα 5 – Λύση




```
ARCHITECTURE synthesis1 OF counter IS

                                        -- The unsigned type is used
   SIGNAL count  : unsigned(31 DOWNTO 0); -- so that unsigned arithmetic
                                        -- will be synthesized

BEGIN
   PROCESS (reset, clock)
   BEGIN
        IF (reset = '1') THEN
                count <= X"00000000";
        ELSIF (clock'EVENT) AND (clock = '1') THEN
                IF (enable = '1') THEN
                        count <= count + 1;
                END IF;
        END IF;
   END PROCESS;
                                        -- Here, the count value is
   value <= std_logic_vector(count);    -- converted to std_logic_vector
                                        -- using a conversion function

END synthesis1;
```

Προσοχή

Πρότυπο Θεμάτων

- **Άλγεβρα Boole (~2.5 μονάδες)**
  - Απλοποίηση συναρτήσεων, Υπολογισμός αντιστρόφου, συνάρτησης ως άθροισμα ελαχιστόρων / γινόμενο μεγιστόρων, Υλοποίηση με πύλες, Πίνακας αληθείας
- **Συνδυαστικά Κυκλώματα: Χάρτης Karnaugh (~2.5 μονάδες)**
  - Απλοποίηση συνάρτησης με χάρτη Karnaugh, Σχεδίαση σε πύλες
- **Συνδυαστικά Κυκλώματα: Υλοποίηση κυκλωμάτων με MUX, DEMUX, Decoders, Encoders, ROMS (~1.5 μονάδες)**
- **Ακολουθιακά Κυκλώματα: D-FF, JK-FF, T-FF (~2.5 μονάδες)**
  - Σχεδίαση καταχωρητών/μετρητών, Διαγράμματα καταστάσεων
- **VHDL (~2 μονάδες)**
  - Κώδικας VHDL για απλά συνδυαστικά και ακολουθιακά κυκλώματα