ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΔΠΜΣ Space Technologies, Applications and seRvices (STAR)

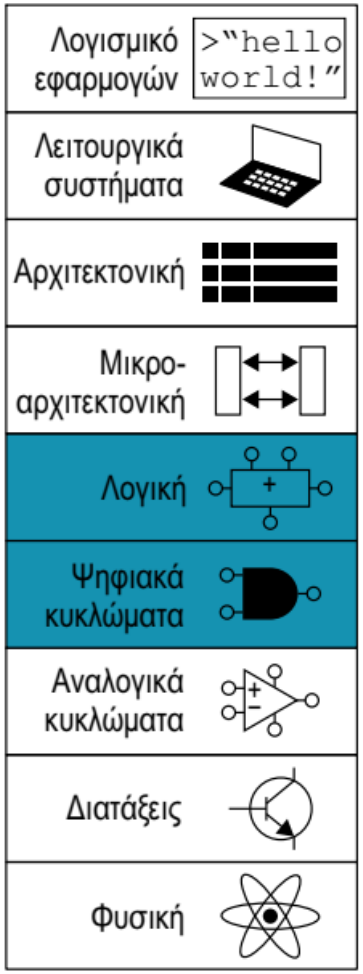
M806 Space Data Systems

VHDL

Ακαδημαϊκό Έτος 2023-2024

Νεκτάριος Κρανίτης

- Εισαγωγή
- Συνδυαστική λογική
- Καθυστερήσεις
- Ακολουθιακή λογική
- Περισσότερη συνδυαστική λογική
- Μηχανές πεπερασμένων καταστάσεων
- Τύποι δεδομένων
- Αριθμητικές πράξεις
- Παραμετροποιημένες υπομονάδες
- Προγράμματα δοκιμών



Εισαγωγή

Εισαγωγή

- Γλώσσες περιγραφής υλικού
  (Hardware description languages -HDLs):
  - Περιγραφή της επιθυμητής λειτουργίας του κυκλώματος με κώδικα
  - Τα εργαλεία Computer-Aided Design (CAD) παράγουν (*synthesize)* το κύκλωμα σε επίπεδο λογικών πυλών (gate-level)
- Οι δύο κυρίαρχες HDLs:
  - **SystemVerilog**
    - Αναπτύχθηκε το 1984 από τη Gateway Design Automation
    - IEEE standard (1364) το 1995
    - Επεκτάθηκε το 2005 (IEEE STD 1800-2009)
  - **VHDL 2008**
    - Αναπτύχθηκε το 1981 από το DoD (USA)
    - IEEE standard (1076) το 1987
    - Αναθεωρήθηκε το 2008 (IEEE STD 1076-2008)
    - Επιβάλλεται η χρήση της VHDL στις: DoD, NASA, ESA, CERN

Από τις HDL σε λογικές πύλες

- **Προσομοίωση (Simulation)**
  - Οι είσοδοι δοκιμής εφαρμόζονται στο κύκλωμα
  - Οι έξοδοι ελέγχονται για την ορθότητα
  - Η απεσφαλμάτωση (debugging) στο επίπεδο της προσομοίωσης αντί του υλικού είναι πολύ πιο οικονομική
- **Synthesis (Σύνθεση)**
  - Μετατρέπει το κώδικα HDL σε *netlist* που περιγράφει το hardware (πχ. λίστα πυλών και διασυνδέσεις)

HDL: ***Hardware*** Description Language

ΣΗΜΑΝΤΙΚΟ:

Όταν γράφουμε κώδικα σε μια HDL, πρέπει να σκεφτόμαστε το **hardware** που θα παράγει η HDL, και μετά να γράφουμε τους απαραίτητους ιδιωματισμούς που οδηγούν σε αυτό το hardware

Προσοχή στη χρήση μιας HDL σαν software υψηλού επιπέδου χωρίς να σκεφτόμαστε το hardware

HDL: ***Hardware*** Description Language

ΣΗΜΑΝΤΙΚΟ:

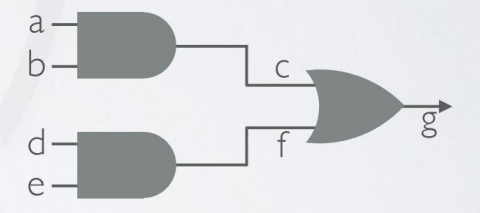
Αυτό

```
c<= a and b;
f <= d and e;
g<= c or f;
```

και αυτό

```
g<= c or f;
c<= a and b;
f <= d and e;
```

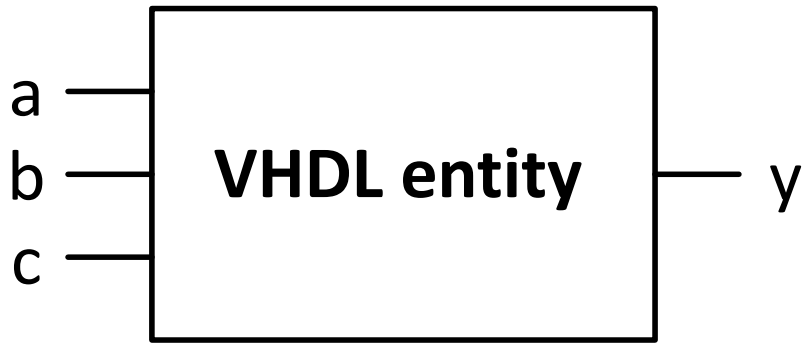
Οδηγούν στο ίδιο αποτέλεσμα!!



Γιατί οι HDLs περιγράφουν λογικά κυκλώματα!!

Οι εντολές HDL είναι ταυτόχρονες (concurrent)!
Σκεφτείτε με βάση το hardware!!

Οντότητα (entity) VHDL



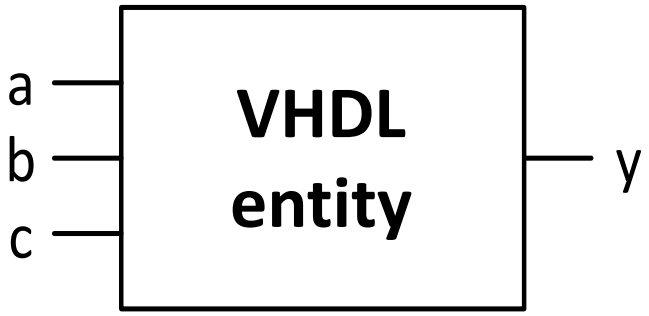
Δύο τύποι:

- **Behavioral:** περιγράφει **ΤΙ** κάνει το entity
- **Structural:** περιγράφει **ΠΩΣ** χτίζεται από πιο απλά entities

Δήλωση οντότητας (entity declaration)

VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
port(a, b, c: in STD_LOGIC;
y: out STD_LOGIC);
end;
```



- Προσδιορίζεται το **όνομα** του entity (εδώ sillyfunction)
- Απαριθμούνται οι θύρες (ports), δηλαδή οι είσοδοι και οι έξοδοί της

Περιγραφή συμπεριφοράς (behavioral description)

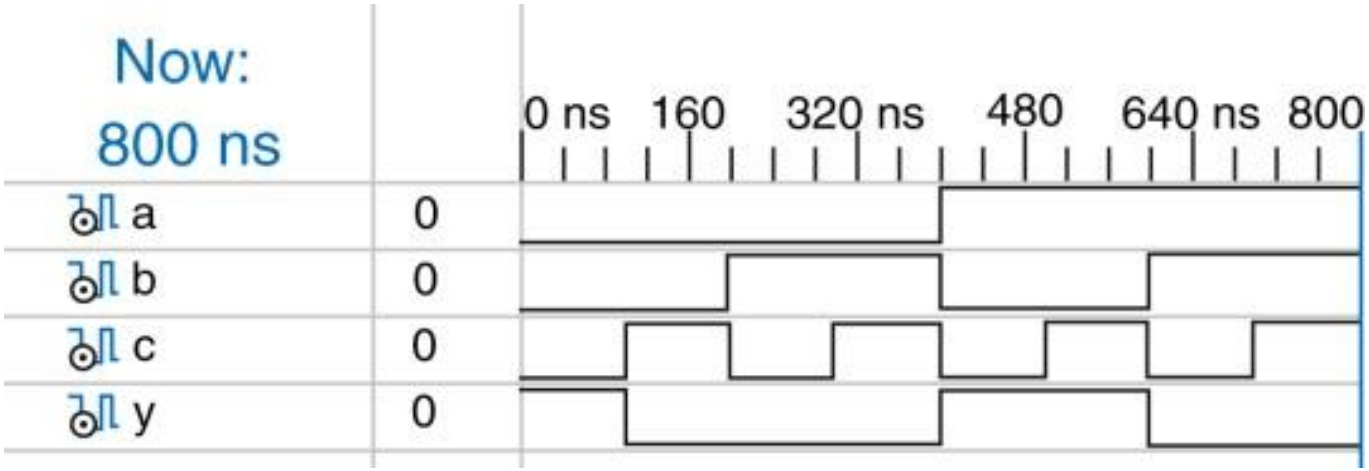
```
architecture synth of sillyfunction is
begin
y <= (not a and not b and not c) or
      (a and not b and not c) or
      (a and not b and c);
end;
```

- Στο σώμα (architecture) ορίζεται το τι κάνει το entity
- Τα σήματα της VHDL (πχ είσοδοι, έξοδοι, εσωτερικά), πρέπει να διαθέτουν μια **δήλωση τύπου (type declaration)**
- Τα ψηφιακά σήματα πρέπει να δηλώνονται με τον τύπο STD_LOGIC που μπορούν να έχουν την τιμή '0' ή '1', καθώς επίσης κυμαινόμενες (δηλαδή «μετέωρες») και μη ορισμένες τιμές
- Στη VHDL, η προτεραιότητα εκτέλεσης των πράξεων AND και OR δεν είναι σαφώς ορισμένη, οπότε οι εξισώσεις Boole πρέπει να μπαίνουν σε παρενθέσεις

Προσομοίωση HDL

- Παρακάτω φαίνεται η κυματομορφή που προέκυψε από μία προσομοίωση του entity που υλοποιεί την εξίσωση Boole:

$$Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$$

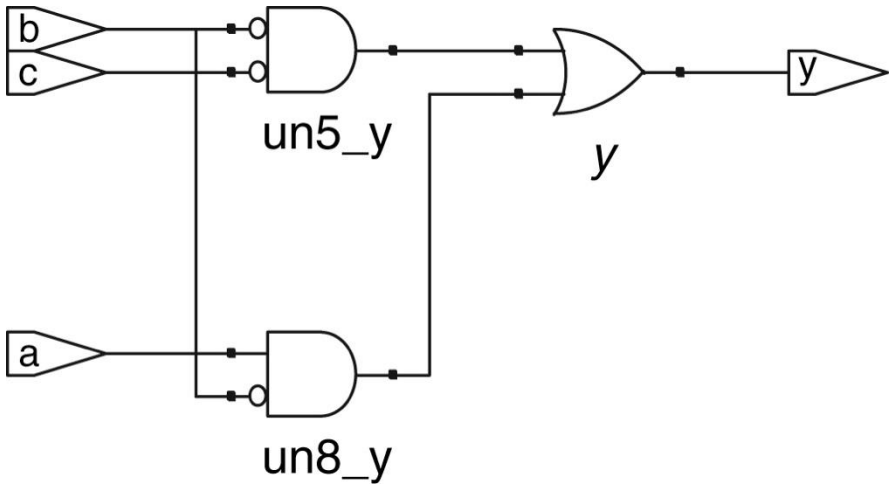


- Εξετάζοντας το διάγραμμα χρονισμού καταλήγουμε ότι η υπομονάδα όντως λειτουργεί σωστά
  - Η έξοδος Y έχει την τιμή TRUE όταν οι είσοδοι A, B και C είναι 000, 100 ή 101, όπως δηλαδή ορίζει η εξίσωση Boole

HDL Synthesis

- Στο παρακάτω σχήμα φαίνεται το **αποτελέσμα της σύνθεσης** της λογικής για την υπομονάδα που υλοποιεί την εξίσωση Boole:

$$\mathbf{Y = \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C}$$



- Οι τρεις πύλες AND, με τρεις εισόδους η καθεμία, <u>ελαχιστοποιούνται</u> σε δύο πύλες AND των δύο εισόδων

$$\mathbf{Y = \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C = B\bar{C} + A\bar{B}}$$

Συντακτικό VHDL

- **Case insensitive**
  - **Παράδειγμα:** Τα `y1` και `Y1` είναι το ίδιο σήμα
- Δεν πρέπει να ονόματα να ξεκινούν με νούμερα
  - **Παράδειγμα:** Το `2mux` είναι **invalid** όνομα
- **Σχόλια:**
  - `// single line comment`
  - `/* multiline`
    `comment */`

Περιγραφή δομής (structural description)

- Μοντελοποίηση δομής για τη κατασκευή ενός πολυπλέκτη 2:1 από ένα ζεύγος απομονωτών τριών καταστάσεων (tristate buffers)

```
library IEEE; use IEEE.STD LOGIC 1164.all;

entity mux2 is
  port(d0, d1: in  STD_LOGIC_VECTOR(3 downto 0);
       s:      in  STD LOGIC;
       y:      out STD LOGIC VECTOR(3 downto 0));
end;

architecture struct of mux2 is
  component tristate
    port(a:  in  STD LOGIC VECTOR(3 downto 0);
         en: in  STD LOGIC;
         y:  out STD LOGIC VECTOR(3 downto 0));
  end component;
  signal sbar: STD_LOGIC;
begin
  sbar <= not s;
  t0: tristate port map(d0, sbar, y);
  t1: tristate port map(d1, s, y);
end;
```

Component declaration

Component instantiation

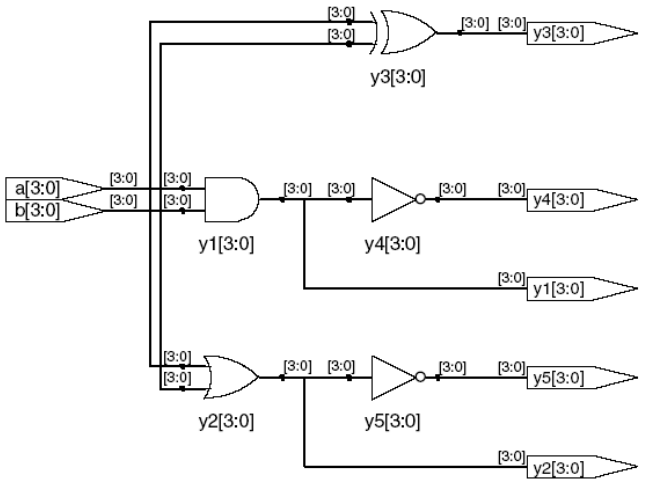
Συνδυαστική λογική

Πράξεις ανά bit (Bitwise Operators)

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity gates is
port(a, b: in STD LOGIC_VECTOR(3 downto 0);
     y1, y2, y3, y4,
     y5: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of gates is
begin
  -- πέντε διαφορετικές λογικές πύλες
  -- δύο εισόδων που εφαρμόζονται σε διαύλους
  -- των 4 bit
  y1 <= a and b;
  y2 <= a or b;
  y3 <= a xor b;
  y4 <= a nand b;
  y5 <= a nor b;
end;
```

Synthesis:



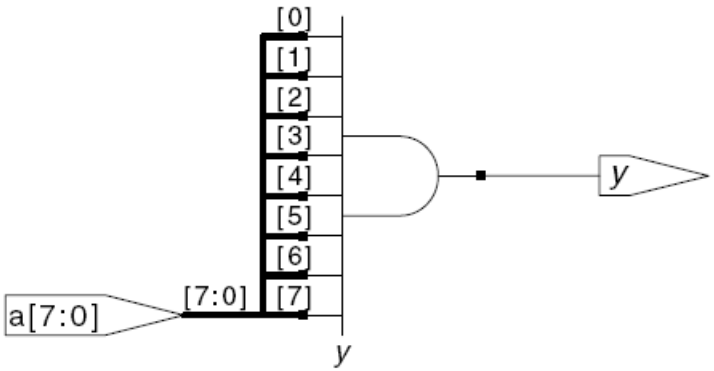
Τελεστές μείωσης (Reduction Operators)

VHDL:

```
library IEEE; use IEEE.STD LOGIC 1164.all;

entity and8 is
  port(a: in  STD LOGIC VECTOR(7 downto 0);
       y: out STD_LOGIC);
end;

architecture synth of and8 is
begin
  y <= and a;
  -- Επίσης πιο εύκολο είναι να γράψουμε a αντί
  -- για y <= a(7) and a(6) and a(5) and a(4) and
  --           a(3) and a(2) and a(1) and a(0);
end;
```

Οι τελεστές μείωσης συνεπάγονται την ύπαρξη μιας πύλης με πολλές εισόδους η οποία εφαρμόζεται σε έναν μόνο δίαυλο

Synthesis:



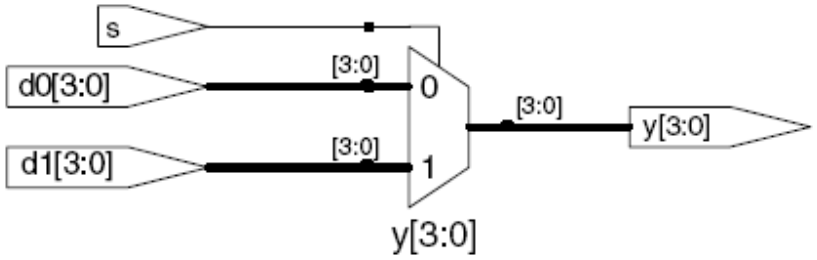
Ανάθεση υπό συνθήκη

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2 is
port(d0, d1: in  STD_LOGIC_VECTOR(3 downto 0);
     s:      in  STD_LOGIC;
     y:      out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of mux2 is
begin
  y <= d1 when s else d0;
end;
```

Πριν την αναθεώρηση VHDL 2008, ήμαστε υποχρεωμένοι να γράφουμε `when s = '1'` αντί απλώς `when s`

Synthesis:



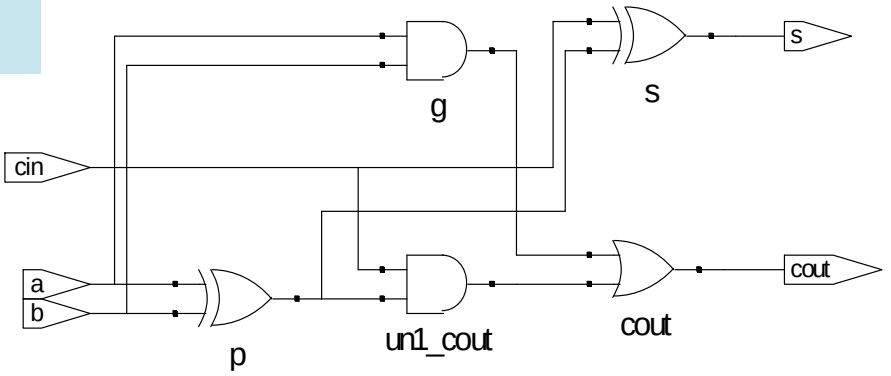
Εσωτερικά σήματα

VHDL:

```
library IEEE; use IEEE.STD LOGIC 1164.all;

entity fulladder is
  port(a, b, cin: in  STD LOGIC;
       s, cout:   out STD_LOGIC);
end;

architecture synth of fulladder is
  signal p, g: STD LOGIC;
begin
  p <= a xor b;
  g <= a and b;

  s <= p xor cin;
  cout <= g or (p and cin);
end;
```

Στη VHDL, τα εσωτερικά σήματα συνήθως δηλώνονται ως `signal` για την αναπαράσταση εσωτερικών σημάτων των οποίων οι τιμές ορίζονται με **εντολές ταυτόχρονης ανάθεσης (concurrent statement)** τιμής σήματος όπως η
`p <= a xor b;`

Synthesis:



Προτεραιότητα τελεστών

Υψηλότερη → Χαμηλότερη

Τελεστής	Σημασία
`not`	NOT
`*, /, mod, rem`	MUL, DIV, MOD, REM
`+, -`	PLUS, MINUS
`rol, ror, srl, sll`	Περιστροφή, λογική ολίσθηση
`<, <=, >, >=`	Σχετική σύγκριση
`=, /=`	Σύγκριση ισότητας
`and, or, nand, nor, xor, xnor`	Λογικές πράξεις (εκτελούνται από αριστερά προς τα δεξιά)

Αριθμοί

Μορφή: NB”value”

N = μέγεθος σε bits, **B** = base

Αριθμοί	Bit	Βάση	Τιμή	Αποθηκεύεται ως
3B"101"	3	2	5	101
B"11"	2	2	3	11
8B"11"	8	2	3	00000011
8B"1010_1011"	8	2	171	10101011
3D"6"	3	10	6	110
60"42"	6	8	34	100010
8X"AB"	8	16	171	10101011
"101"	3	2	5	101
B"101	3	2	5	101
X"AB"	8	16	171	10101011

Τα **others => '0'** και **others => '1'** είναι οι ιδιωματισμοί της VHDL για τη συμπλήρωση όλων των bit με τις τιμές 0 και 1, αντίστοιχα.

Ανάδευση bit

VHDL:

```
y <=(c(2 downto 1), d(0), d(0), d(0), c(0), 3B"101");
```

- Ο τελεστής συνάθροισης () χρησιμοποιείται για τη συνένωση διαύλων
- Το y πρέπει να είναι STD_LOGIC_ VECTOR με μέγεθος 9 bit
- Η δύναμη των συναθροίσεων (aggregations) της VHDL φαίνεται και από το ακόλουθο παράδειγμα
- Παράδειγμα
  - Αν υποθέσουμε ότι το z είναι STD_LOGIC_VECTOR με μέγεθος 8 bit
  - Δίνεται η τιμή 10010110 στο z με χρήση της παρακάτω συνάθροισης εντολών:

```
z <= ("10", 4 => '1', 2 downto 1 =>'1',others =>'0')
```

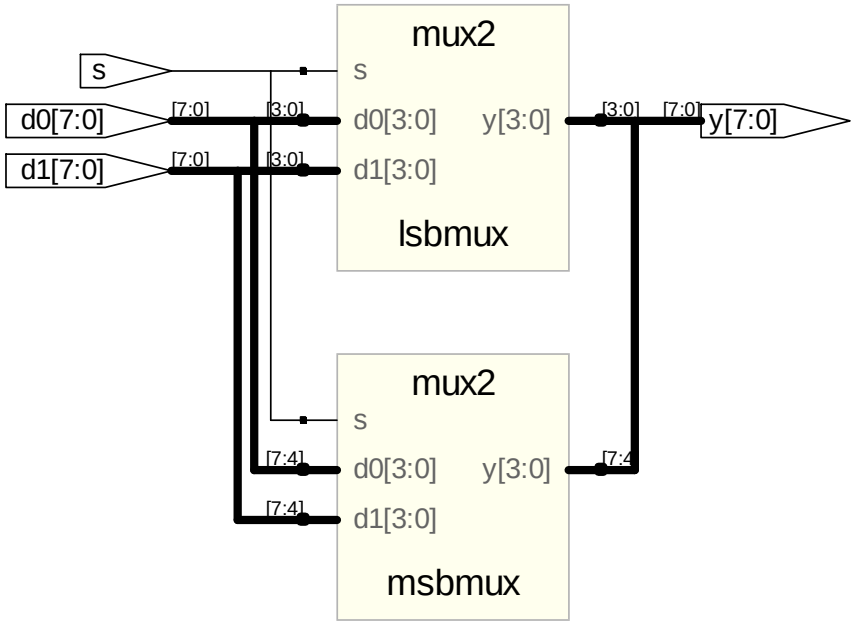
  - Το "10" τοποθετείται στα δύο πρώτα bit
  - Στο 4ο bit, καθώς επίσης στο 1ο και το 2ο bit, τοποθετούνται άσοι
  - Τα υπόλοιπα bit έχουν την τιμή 0

Προσπέλαση τμήματος των διαύλων

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2_8 is
  port(d0, d1: in  STD_LOGIC_VECTOR(7 downto 0);
       s:      in  STD_LOGIC;
       y:      out STD_LOGIC_VECTOR(7 downto 0));

end;

architecture struct of mux2_8 is
  component mux2
    port(d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
         s:  in  STD_LOGIC;
         y:  out STD_LOGIC_VECTOR(3 downto 0));
  end component;
begin
  lsbmux: mux2
    port map(d0(3 downto 0), d1(3 downto 0),
             s, y(3 downto 0));
  msbmux: mux2
    port map(d0(7 downto 4), d1(7 downto 4),
             s, y(7 downto 4));
end;
```

Synthesis:



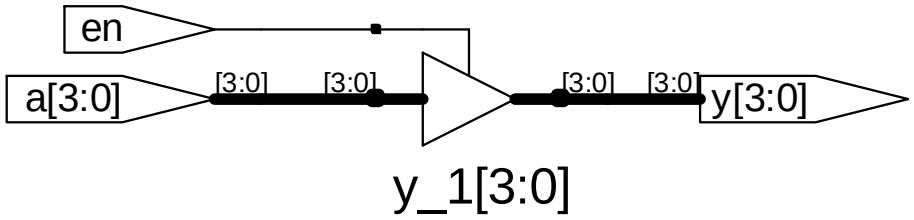
Απομονωτής τριών καταστάσεων

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity tristate is
  port(a: in  STD_LOGIC_VECTOR(3 downto 0);
       en: in  STD_LOGIC;
        y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of tristate is
begin
  y <= a when en else "ZZZZ";
end;
```

- Οι HDLs χρησιμοποιούν το **z** για να αναπαριστούν μια κυμαινόμενη (δηλαδή «μετέωρη») τιμή
- Το z είναι χρήσιμο για την περιγραφή ενός **απομονωτή τριών καταστάσεων (tristate buffer)**, του οποίου η έξοδος είναι μετέωρη όταν η είσοδος enable έχει την τιμή 0

Synthesis:



Καθυστερήσεις

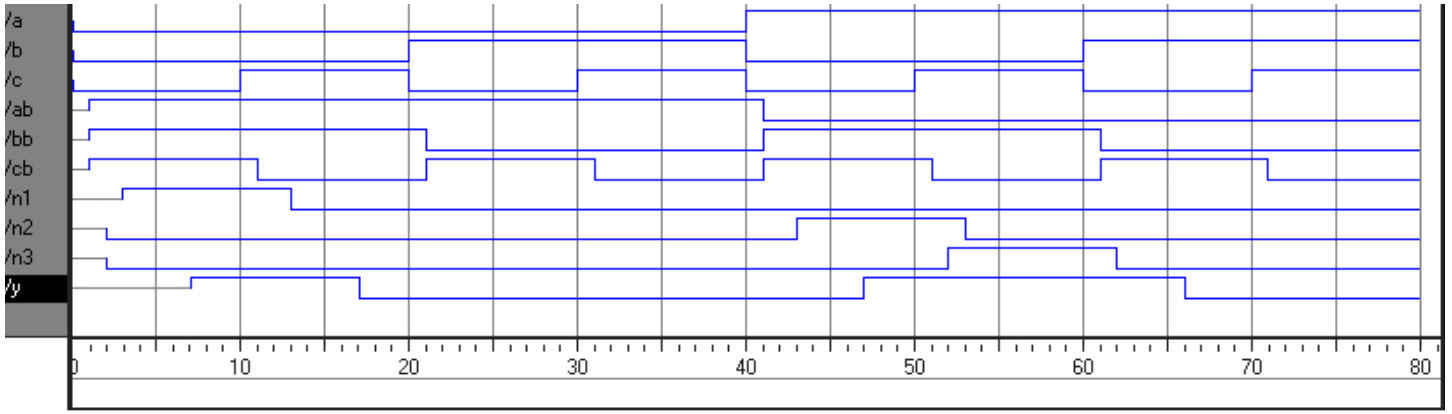
Καθυστερήσεις

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity example is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture synth of example is
  signal ab, bb, cb, n1, n2, n3: STD_LOGIC;
begin
  ab <= not a after 1 ns;
  bb <= not b after 1 ns;
  cb <= not c after 1 ns;
  n1 <= ab and bb and cb after 2 ns;
  n2 <= a and bb and cb after 2 ns;
  n3 <= a and bb and c after 2 ns;
  y  <= n1 or n2 or n3 after 4 ns;
end;
```

- Οι καθυστερήσεις χρησιμοποιούνται **αποκλειστικά για προσομοίωση!** Δεν καθορίζουν την καθυστέρηση του hardware που μοντελοποιούν !!!!
- Οι καθυστερήσεις αγνοούνται κατά τη σύνθεση· η καθυστέρηση μιας πύλης που παράγεται από το εργαλείο σύνθεσης εξαρτάται από τις προδιαγραφές της για τους χρόνους $\mathbf{t_{pd}}$ και $\mathbf{t_{cd}}$, και όχι από οποιονδήποτε αριθμό στον κώδικα HDL
- Στη VHDL, χρησιμοποιούμε την πρόταση **after** για να υποδείξουμε τη καθυστέρηση. Εδώ ως μονάδες ορίζονται τα nanoseconds (ns)

Simulation

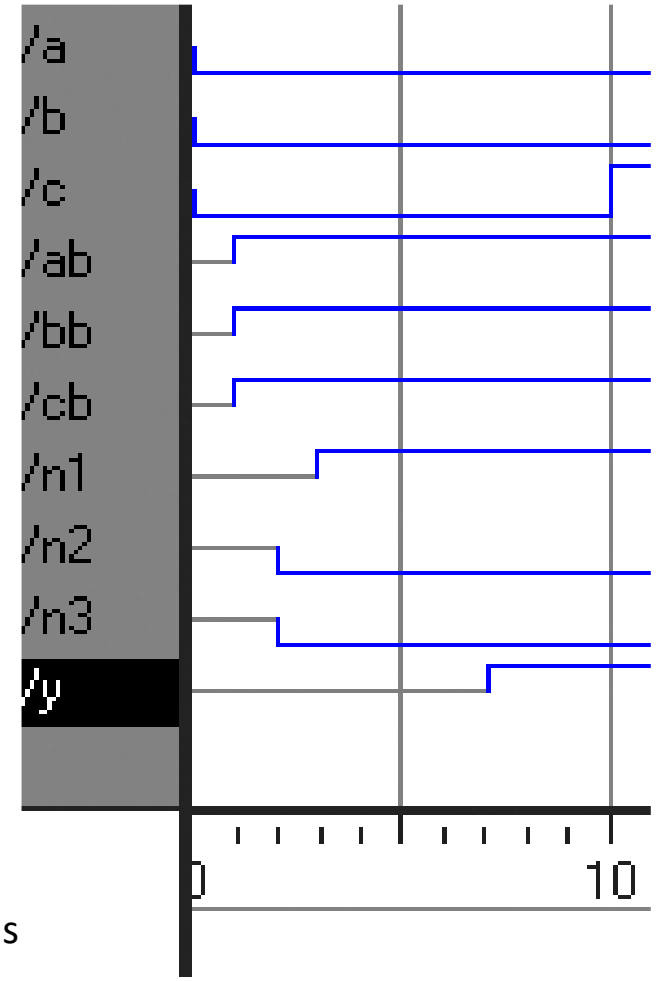


Καθυστερήσεις

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity example is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture synth of example is
  signal ab, bb, cb, n1, n2, n3: STD_LOGIC;
begin
  ab <= not a after 1 ns;
  bb <= not b after 1 ns;
  cb <= not c after 1 ns;
  n1 <= ab and bb and cb after 2 ns;
  n2 <= a and bb and cb after 2 ns;
  n3 <= a and bb and c after 2 ns;
  y  <= n1 or n2 or n3 after 4 ns;
end;
```

Simulation



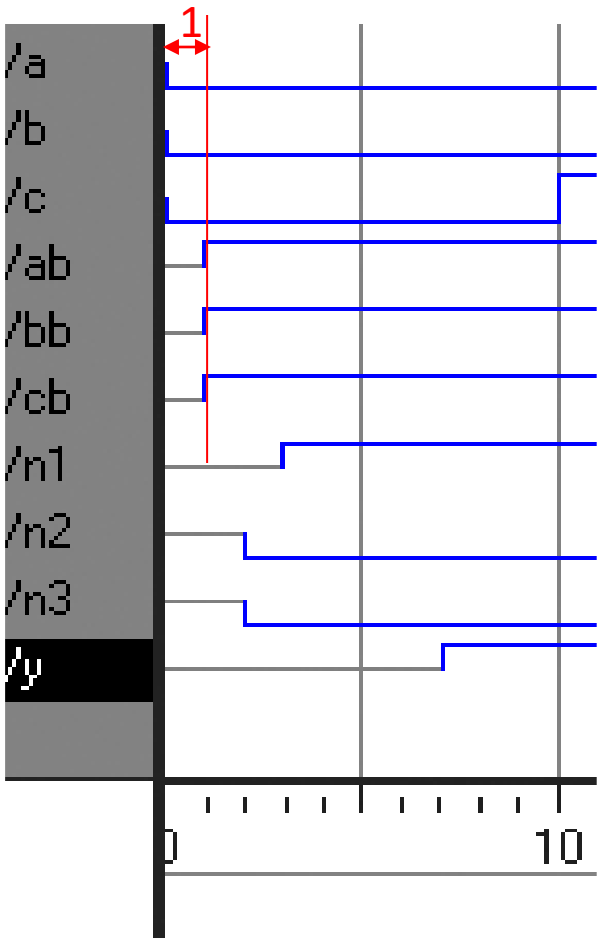
- Στον κώδικα περιγράφουμε ότι:
  - οι αντιστροφείς έχουν καθυστέρηση 1 ns
  - οι πύλες AND με τρεις εισόδους έχουν καθυστέρηση 2 ns
  - οι πύλες OR με τρεις εισόδους έχουν καθυστέρηση 4 ns

Καθυστερήσεις

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity example is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture synth of example is
  signal ab, bb, cb, n1, n2, n3: STD_LOGIC;
begin
  ab <= not a after 1 ns;
  bb <= not b after 1 ns;
  cb <= not c after 1 ns;
  n1 <= ab and bb and cb after 2 ns;
  n2 <= a and bb and cb after 2 ns;
  n3 <= a and bb and c after 2 ns;
  y  <= n1 or n2 or n3 after 4 ns;
end;
```

Simulation

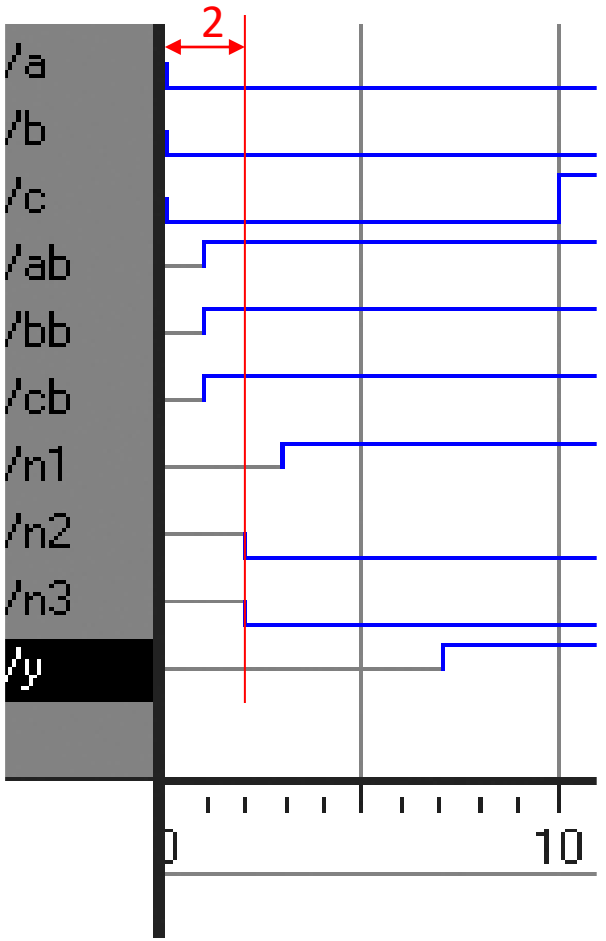


Καθυστερήσεις

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity example is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture synth of example is
  signal ab, bb, cb, n1, n2, n3: STD_LOGIC;
begin
  ab <= not a after 1 ns;
  bb <= not b after 1 ns;
  cb <= not c after 1 ns;
  n1 <= ab and bb and cb after 2 ns;
  n2 <= a and bb and cb after 2 ns;
  n3 <= a and bb and c after 2 ns;
  y  <= n1 or n2 or n3 after 4 ns;
end;
```

Simulation

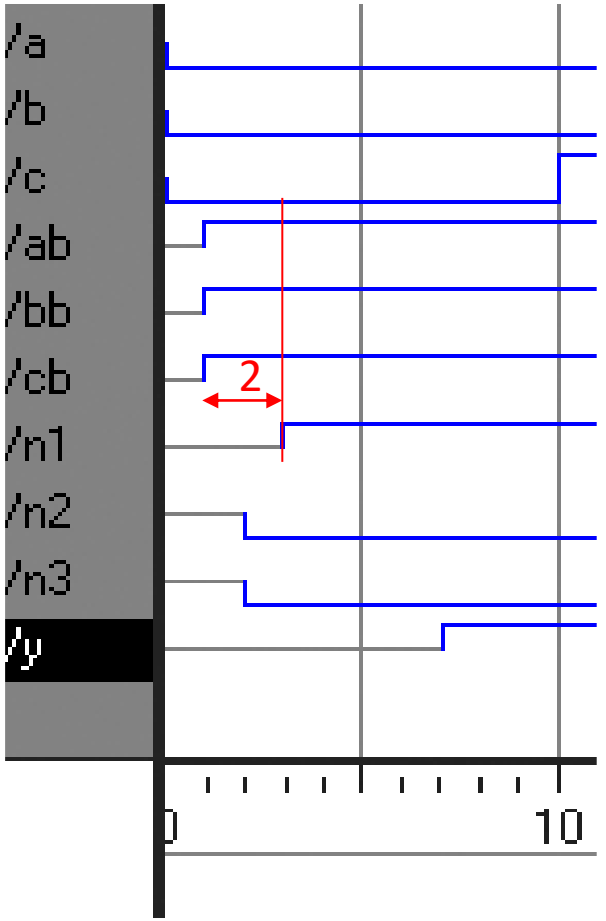


Καθυστερήσεις

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity example is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture synth of example is
  signal ab, bb, cb, n1, n2, n3: STD_LOGIC;
begin
  ab <= not a after 1 ns;
  bb <= not b after 1 ns;
  cb <= not c after 1 ns;
  n1 <= ab and bb and cb after 2 ns;
  n2 <= a and bb and cb after 2 ns;
  n3 <= a and bb and c after 2 ns;
  y  <= n1 or n2 or n3 after 4 ns;
end;
```

Simulation

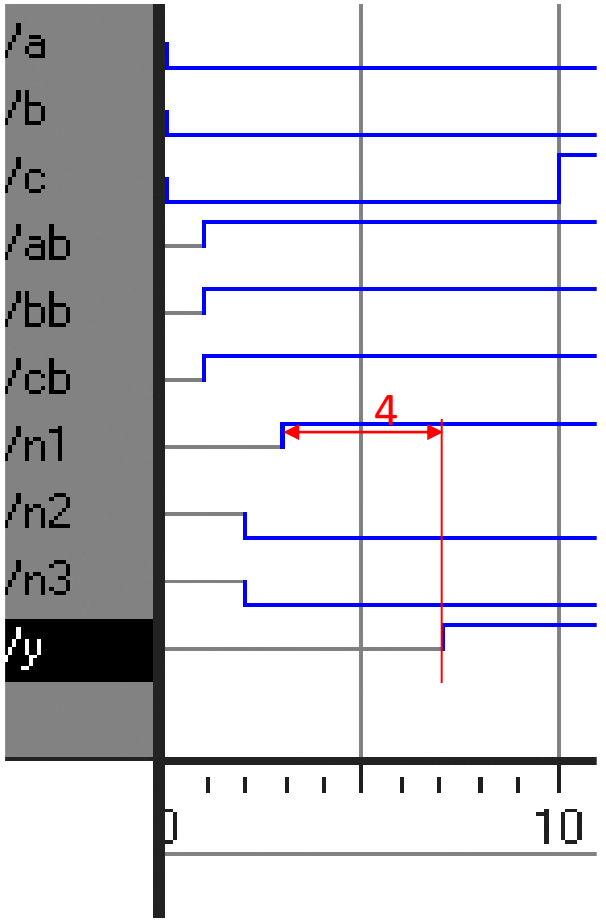


Καθυστερήσεις

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity example is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture synth of example is
  signal ab, bb, cb, n1, n2, n3: STD_LOGIC;
begin
  ab <= not a after 1 ns;
  bb <= not b after 1 ns;
  cb <= not c after 1 ns;
  n1 <= ab and bb and cb after 2 ns;
  n2 <= a and bb and cb after 2 ns;
  n3 <= a and bb and c after 2 ns;
  y  <= n1 or n2 or n3 after 4 ns;
end;
```

Simulation



Ακολουθιακή λογική

Ακολουθιακή λογική

- Τα εργαλεία σύνθεσης αναγνωρίζουν **ιδιοματισμούς (idioms)** της VHDL και τους μετατρέπουν σε latches, flip-flops και FSMs
- Προσοχή: Άλλα στυλ συγγραφής κώδικα ενδέχεται να προσομοιώνονται σωστά, αλλά να συνθέτονται σε λάθος κυκλώματα

process (διεργασία)

Γενική δομή διεργασίας (process):

```
process(sensitivity list) begin
   statement;
end process;
```

Η εντολή **`statement`** στο σώμα μιας εντολής `process` εκτελείται μόνο όταν αλλάζει οποιαδήποτε από τις μεταβλητές που περιέχονται στη **`λίστα ευαισθησίας (sensitivity list)`**

D Flip-Flop

VHDL:

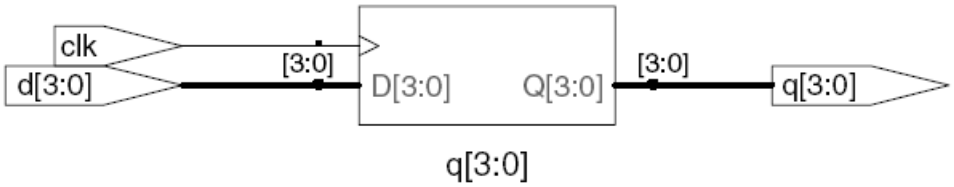
```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flop is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC_VECTOR(3 downto 0);
       q:   out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of flop is
begin
  process(clk) begin
    if rising_edge(clk) then
      q <= d;
    end if;
  end process;
end;
```

- Εδώ, η εντολή `if` ελέγχει αν η αλλαγή είναι μια ανερχόμενη ακμή (`rising_edge`) του ρολογιού (clk)
- Αν όντως ισχύει κάτι τέτοιο, τότε

  `q <= d`

  (διαβάζουμε «το q παίρνει την τιμή d»)
- Επομένως, το flip-flop αντιγράφει (καλύτερα «φωτογραφίζει») το d στο q κατά τη θετική ακμή του ρολογιού, ειδάλλως θυμάται την προηγούμενη κατάσταση του q
  - Προσοχή στους χρόνους setup-hold….

Synthesis:

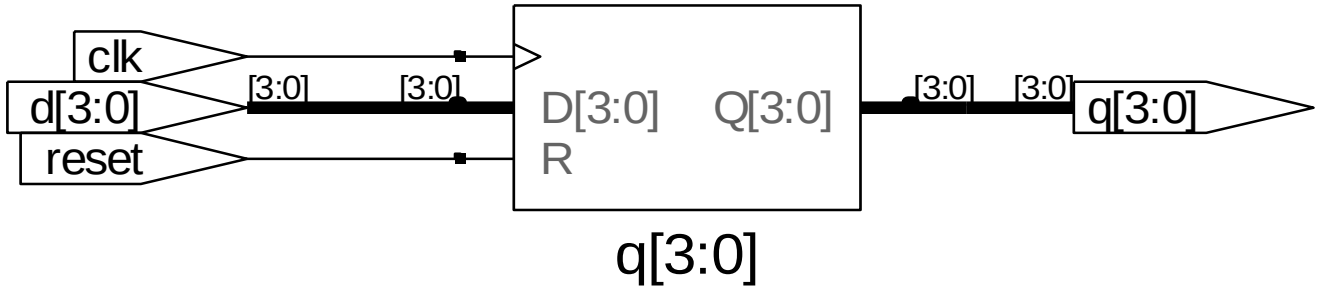


D Flip-Flop με σύγχρονο reset

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flopr is
  port(clk, reset: in  STD_LOGIC;
       d:          in  STD_LOGIC_VECTOR(3 downto 0);
       q:          out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synchronous of flopr is
begin
  process(clk) begin
    if rising_edge(clk) then
      if reset then q <= "0000";
      else q <= d;
      end if;
    end if;
  end process;
end;
```

Synthesis:



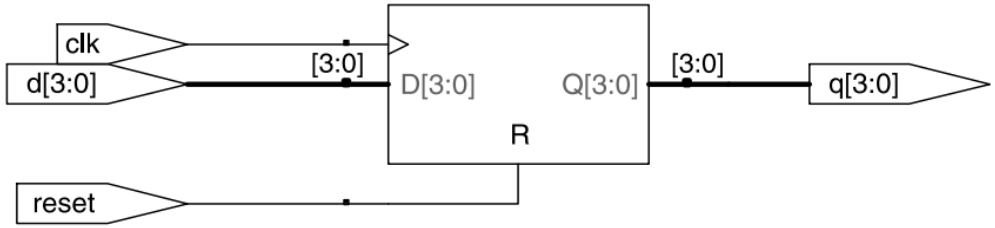
D Flip-Flop με ασύγχρονο reset

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flopr is
  port(clk, reset: in  STD_LOGIC;
       d:          in  STD_LOGIC_VECTOR(3 downto 0);
       q:          out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture asynchronous of flopr is
begin
  process(clk, reset) begin
    if reset then
      q <= "0000";
    elsif rising_edge(clk) then
      q <= d;
    end if;
  end process;
end;
```

- Εδώ, το `reset` βρίσκεται στο sensitivity list, έχει προτεραιότητα (στο εξωτερικό `if`) έναντι του clk και συνεπώς οδηγεί σε ασύγχρονο reset

Synthesis:

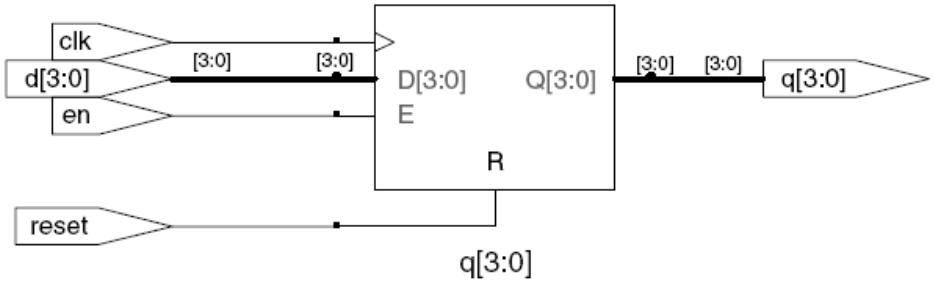


D Flip-Flop με Enable

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity flopenr is
  port(clk,
       reset,
       en: in  STD LOGIC;
       d:  in  STD LOGIC VECTOR(3 downto 0);
       q:  out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture asynchronous of flopenr is
-- ασύγχρονη επαναφορά στο 0
begin
  process(clk, reset) begin
    if reset then
      q <= "0000";
    elsif rising edge(clk) then
      if en then
        q <= d;
      end if;
    end if;
  end process;
end;
```

Synthesis:

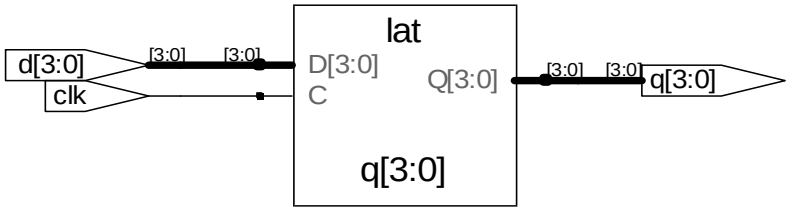


Latch

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity latch is
port(clk: in  STD LOGIC;
     d:   in  STD LOGIC VECTOR(3 downto 0);
     q:   out STD LOGIC VECTOR(3 downto 0));
end;

architecture synth of latch is
begin
  process(clk, d) begin
    if clk = '1' then
      q <= d;
    end if;
  end process;
end;
```

Synthesis:



- Η λίστα ευαισθησίας περιέχει και το clk **και το d**, οπότε η τιμή του process υπολογίζεται κάθε φορά που μεταβάλλεται το clk **ή το d**. Δηλαδή, όσο το clk έχει την τιμή HIGH, το d **ρέει** μέσω του κυκλώματος προς το q

Προσοχή: Δεν χρησιμοποιούμε latches συχνά…
Αλλά μπορεί να γράψετε κώδικα που να οδηγεί σε latch!!
Ελέγξτε το synthesized hardware – εάν έχει latches που δεν σκοπεύατε να παράγεται οφείλεται σε (πολύ συχνό σε αρχάριους) **λάθος**

Σύνοψη

Γενική δομή διεργασίας:

```
process(sensitivity list) begin
   statement;
end process;
```

- **Flip-flop:** `process(clk)`
- **Latch:** `process(clk, d)` **(don't use)**

Περισσότερη συνδυαστική λογική (μέσα σε process)

Συνδυαστική λογική σε process

- Οι εντολές `process` της VHDL χρησιμοποιούνται για την **περιγραφή ακολουθιακών κυκλωμάτων**, επειδή οι έξοδοι στο σώμα αυτών των εντολών θυμούνται την προηγούμενη τιμή τους όταν δεν καθορίζεται νέα τιμή

- Επιπλέον, μπορούν επίσης να χρησιμοποιηθούν εντολές `process` για τη **μοντελοποίηση της συμπεριφοράς της συνδυαστικής λογικής** αν:
  - η λίστα ευαισθησίας είναι ορισμένη έτσι ώστε να αποκρίνεται σε αλλαγές σε όλες τις εισόδους
  - το σώμα αυτών των εντολών καθορίζει την τιμή εξόδου για κάθε πιθανό συνδυασμό τιμών των εισόδων

Συνδυαστική λογική σε process

VHDL:

```
library IEEE; use IEEE.STD LOGIC 1164.all;

entity inv is
  port(a: in  STD_LOGIC_VECTOR(3 downto 0);
       y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture proc of inv is
begin
  process(all) begin
    y <= not a;
  end process;
end;
```

- Το `process(all)` υπολογίζει εκ νέου την τιμή των εντολών που περιέχονται στο process κάθε φορά που μεταβάλλεται οποιοδήποτε από τα σήματα στο σώμα του process
- Είναι ισοδύναμο με το `process(a)`, αλλά είναι καλύτερο επειδή αποφεύγει τα σφάλματα στη περίπτωση που μετονομαστούν ή προστεθούν σήματα μέσα στο process
- Οι εντολές `begin` και `end process` απαιτούνται στη VHDL παρά το γεγονός ότι η εντολή process περιέχει μόνο μία εντολή ανάθεσης τιμής

Εντολές **if/else** και **case**

- Οι εντολές `case` και `if` είναι βολικές για τη μοντελοποίηση πιο περίπλοκης συνδυαστικής λογικής

- Στις επόμενες διαφάνειες θα εξετάσουμε τις εντολές `case` και `if`, οι οποίες πρέπει να εμφανίζονται αποκλειστικά μέσα στο σώμα των εντολών process

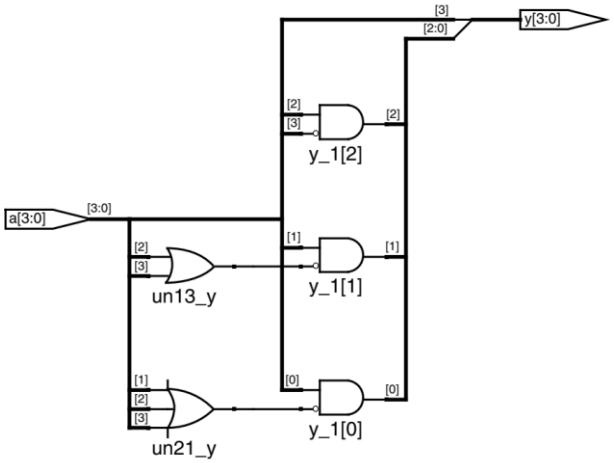
Συνδυαστική λογική με χρήση `if`

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity priorityckt is
  port(a: in  STD_LOGIC_VECTOR(3 downto 0);
       y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of priorityckt is
begin
  process(all) begin
    if    a(3) then y <= "1000";
    elsif a(2) then y <= "0100";
    elsif a(1) then y <= "0010";
    elsif a(0) then y <= "0001";
    else            y <= "0000";
    end if;
  end process;
end;
```

ΚΥΚΛΩΜΑ ΠΡΟΤΕΡΑΙΟΤΗΤΑΣ

Synthesis:



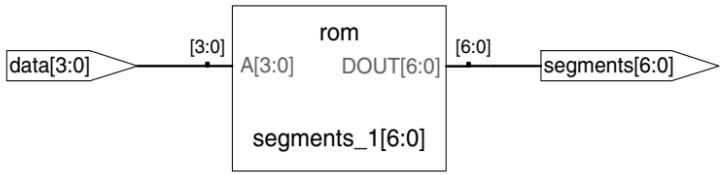
Συνδυαστική λογική με χρήση `case`

VHDL:

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity seven_seg_decoder is
  port(data:     in  STD_LOGIC_VECTOR(3 downto 0);
       segments: out STD_LOGIC_VECTOR(6 downto 0));
end;

architecture synth of seven_seg_decoder is
begin
  process(all) begin
    case data is
      --                          abcdefg
      when X"0" =>   segments <= "1111110";
      when X"1" =>   segments <= "0110000";
      when X"2" =>   segments <= "1101101";
      when X"3" =>   segments <= "1111001";
      when X"4" =>   segments <= "0110011";
      when X"5" =>   segments <= "1011011";
      when X"6" =>   segments <= "1011111";
      when X"7" =>   segments <= "1110000";
      when X"8" =>   segments <= "1111111";
      when X"9" =>   segments <= "1110011";
      when others => segments <= "0000000";
    end case;
  end process;
```

- Η εντολή `case` υπονοεί συνδυαστική λογική combinational logic **μόνο εάν** περιγράφονται όλοι οι συνδυασμοί εισόδου
- Προσοχή στη χρήση του **others!!!**
  Η πρόταση `others` αποτελεί έναν βολικό τρόπο ορισμού της εξόδου για όλες τις περιπτώσεις που δεν παρατίθενται ρητά, και αυτό αποτελεί εγγύηση ότι η λογική είναι συνδυαστική

Synthesis



Συνδυαστική λογική με χρήση `case?`

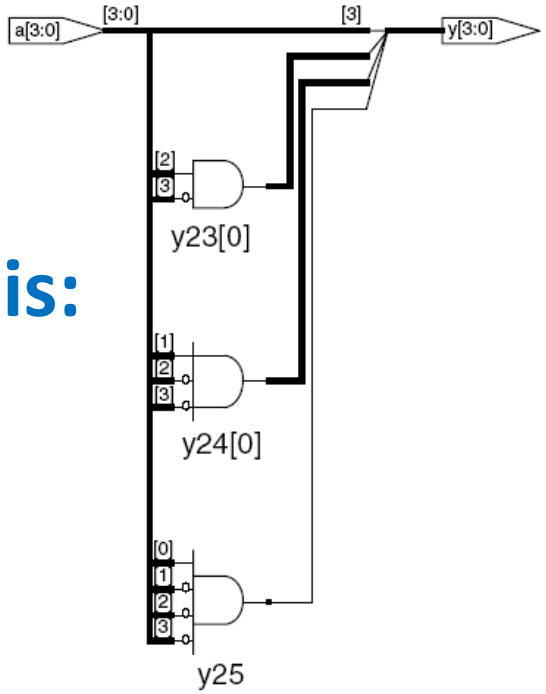
VHDL:

```
library IEEE; use IEEE.STD LOGIC 1164.all;

entity priority_casez is
  port(a: in  STD_LOGIC_VECTOR(3 downto 0);
       y: out STD LOGIC VECTOR(3 downto 0));
end;

architecture dontcare of priority casez is
begin
  process(all) begin
    case? a is
      when "1---" => y <= "1000";
      when "01--" => y <= "0100";
      when "001-" => y <= "0010";
      when "0001" => y <= "0001";
      when others => y <= "0000";
    end case?;
  end process;
end;
```

Κύκλωμα για πίνακες αληθείας με «αδιάφορες» τιμές

Η εντολή `case?` συμπεριφέρεται όπως η `case`, με την εξαίρεση ότι αναγνωρίζει και το – ως «αδιάφορη» τιμή στην είσοδο

Synthesis:



Αναθέσεις τιμής με άμεση και μη άμεση εφαρμογή

Συνδυαστική λογική σε process

- Η VHDL υποστηρίζει εντολές ανάθεσης τιμής στο σώμα μιας εντολής process:
  - με άμεση εφαρμογή
    - Πραγματοποιείται με βάση τη σειρά εμφάνισης των εντολών
    - Ακριβώς όπως σε μια τυπική γλώσσα προγραμματισμού
  - χωρίς άμεση εφαρμογή
    - Πραγματοποιείται ταυτόχρονα (**concurrent statements**)
    - όλες οι εντολές αποτιμώνται πριν από την ενημέρωση οποιουδήποτε από τα σήματα που υπάρχουν στις αριστερές πλευρές των εντολών

Συνδυαστική λογική σε process

- Σε μια εντολή process της VHDL:
  - το **:=** υποδηλώνει ανάθεση τιμής με άμεση εφαρμογή
  - το **<=** υποδηλώνει ανάθεση τιμής χωρίς άμεση εφαρμογή (γνωστή και ως ταυτόχρονη ανάθεση τιμής)
- Αναθέσεις τιμών χωρίς άμεση εφαρμογή γίνονται αποκλειστικά σε εξόδους και σε εσωτερικά σήματα που δηλώνονται ως **signal** πριν από το `begin` στην αρχιτεκτονική (architecture)
- Αναθέσεις τιμών με άμεση εφαρμογή γίνονται αποκλειστικά σε μεταβλητές που δηλώνονται ως **variable** πριν από το `begin` στη process όπου χρησιμοποιείται η μεταβλητή
- Εκτός από τις εντολές process, το **<=** μπορεί να εμφανίζεται και σε εντολές ταυτόχρονης ανάθεσης τιμής σήματος, όπου επίσης αποτιμάται ταυτόχρονα

Εντολές ανάθεσης τιμής με άμεση εφαρμογή

VHDL:

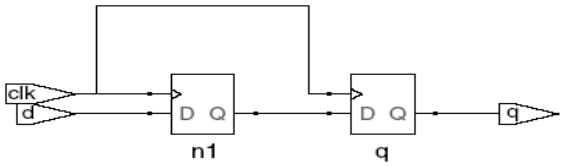
```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fulladder is
  port(a, b, cin: in  STD LOGIC;
       s, cout:   out STD LOGIC);
end;

architecture synth of fulladder is
begin
  process(all)
    variable p, g: STD LOGIC;
  begin
    p := a xor b; -- με άμεση εφαρμογή
    g := a and b; -- με άμεση εφαρμογή
    s <= p xor cin;
    cout <= g or (p and cin);
  end process;
end;
```

- Καλύτερα να χρησιμοποιούνται εντολές ανάθεσης τιμής με άμεση εφαρμογή για ενδιάμεσες **μεταβλητές (variables)** στη συνδυαστική λογική
- Στο παράδειγμα γίνεται χρήση τέτοιων εντολών για τον υπολογισμό των μεταβλητών p και g, έτσι ώστε να παίρνουν τις νέες τιμές τους προτού χρησιμοποιηθούν για τον υπολογισμό των s και cout, που εξαρτώνται από αυτές
- Επειδή τα p και g βρίσκονται στην αριστερή πλευρά μιας εντολής ανάθεσης τιμής με άμεση εφαρμογή (:=) μέσα στο σώμα μιας εντολής process, πρέπει να δηλώνονται ως **variable**, και όχι ως signal
- Η δήλωση των μεταβλητών εμφανίζεται πριν από το `begin` στη process όπου χρησιμοποιείται η μεταβλητή

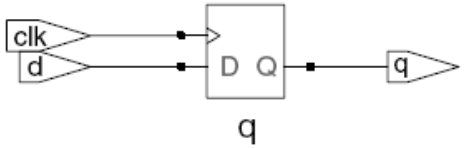
Ανάθεση τιμής με άμεση και χωρίς άμεση εφαρμογή

- **<=** ανάθεση τιμής **χωρίς άμεση** εφαρμογή
  - Συμβαίνει ταυτόχρονα με άλλες
- **:=** ανάθεση τιμής **με άμεση** εφαρμογή
  - Συμβαίνει τη στιγμή μου εμφανίζεται στον κώδικα

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity sync is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC;
       q:   out STD_LOGIC);
end;

architecture good of sync is
  signal n1: STD_LOGIC;
begin
  process(clk) begin
    if rising_edge(clk) then
      n1 <= d;
      q <= n1;
    end if;
  end process;
end;
```




```
-- κακή υλοποίηση συγχρονιστή μέσω της χρήσης
-- αναθέσεων τιμών με άμεση εφαρμογή

library IEEE; use IEEE.STD_LOGIC_1164.all;

entity syncbad is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC;
       q:   out STD_LOGIC);
end;

architecture bad of syncbad is
begin
  process(clk)
    variable n1: STD_LOGIC;
  begin
    if rising_edge(clk) then
      n1 := d; -- με άμεση εφαρμογή
      q <= n1;
    end if;
  end process;
end;
```



Κανόνες για τη ανάθεση σήματος

- **Σύγχρονη ακολουθιακή λογική:** χρήση **`process(clk)`** και αναθέσεις τιμών χωρίς άμεση εφαρμογή (**<=**)

```
process(clk) begin
   if rising_edge(clk) then
      n1 <= d;  -- χωρίς άμεση εφαρμογή
      q  <= n1; -- χωρίς άμεση εφαρμογή
  end if;
end process;
```

- **Απλή συνδυαστική λογική:** χρήση εντολών ταυτόχρονης ανάθεσης έξω από το process

```
y <= d0 when s = '0' else d1;
```

- **Περισσότερο σύνθετη συνδυαστική λογική:** χρήση **`process(all)`** και αναθέσεις τιμών με άμεση εφαρμογή (**:=**)

```
process(all)
   variable p, g: STD_LOGIC;
begin
   p := a xor b; -- με άμεση εφαρμογή
   g := a and b; -- με άμεση εφαρμογή
   s <= p xor cin;
   cout <= g or (p and cin);
end process;
```

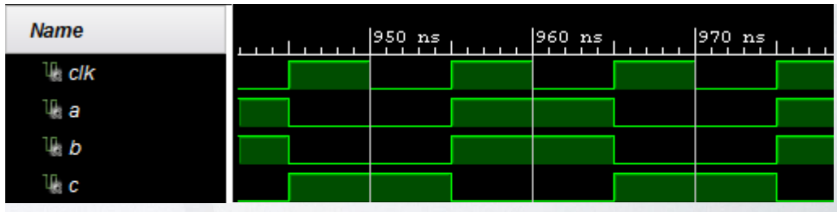
- Ανάθεση σήματος σε **μια μόνο** process ή σε εντολές ταυτόχρονης ανάθεσης σήματος
  - **Σε πολλαπλή ανάθεσης τιμής στο ίδιο σήμα, κατά την προσομοίωση και τη σύνθεση λαμβάνεται υπόψη η τελευταία ανάθεση τιμής**

Κλασσική «παγίδα» στη VHDL

- **Οι εντολές ανάθεσης σήματος μέσα σε process δεν εκτελούνται όπως σε μια γλώσσα προγραμματισμού υψηλού επιπέδου: οι εντολές, στη σειρά, εκτελούνται στο τέλος του process**

- Σε αυτό το process, η 1η ανάθεση δεν συμβαίνει ποτέ!
- Η 3η "c" και η 4η ανάθεση "a" λαμβάνουν την «παλαιές» (πριν την ακμή του clk) τιμές των "a" και "c", αντίστοιχα!

```
process (clk) is
begin
    if rising_edge(clk) then
        a <= b;
        b <= c;
        c <= a;
        a <= c;
    end if;
end process;
```

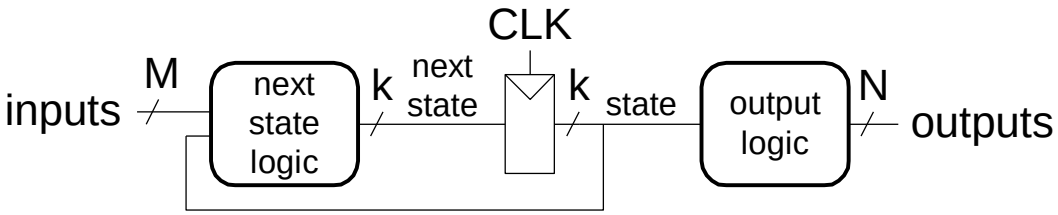


Μηχανές Πεπερασμένων Καταστάσεων

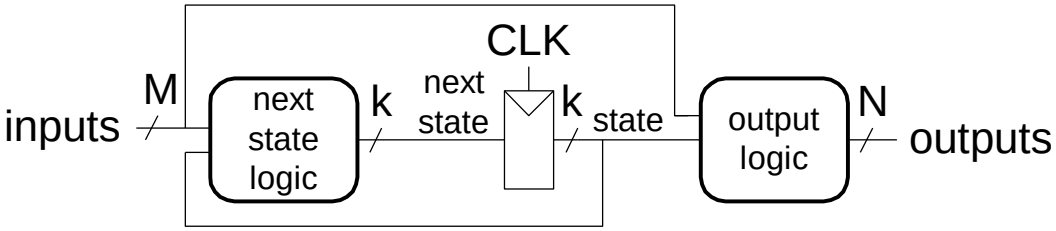
Finite State Machines (FSMs)

- Τρία blocks:
  - Λογική επόμενης κατάστασης (next state logic)
  - Καταχωρητής κατάστασης (state register)
  - Λογική εξόδου (output logic)

Moore FSM

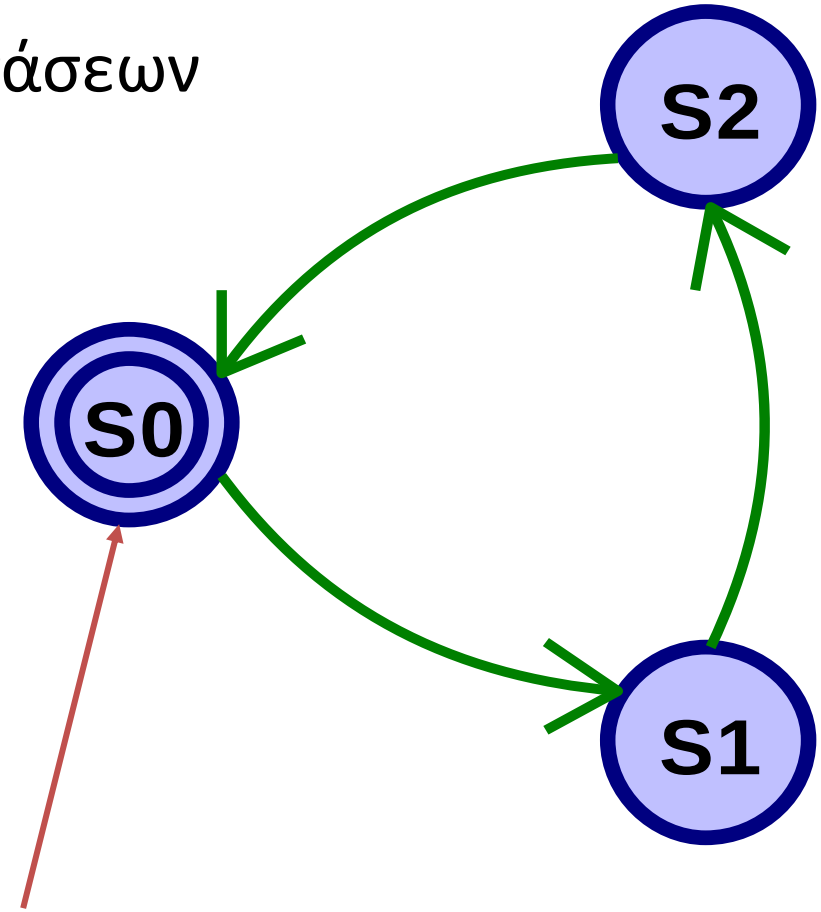


Mealy FSM



Μετρητής Modulo-3

Διάγραμμα καταστάσεων



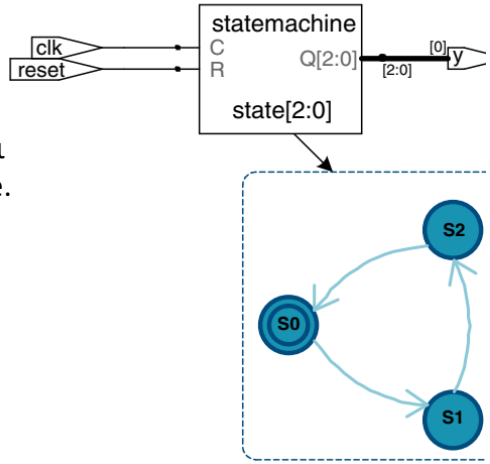
Ο διπλός κύκλος δείχνει το reset state

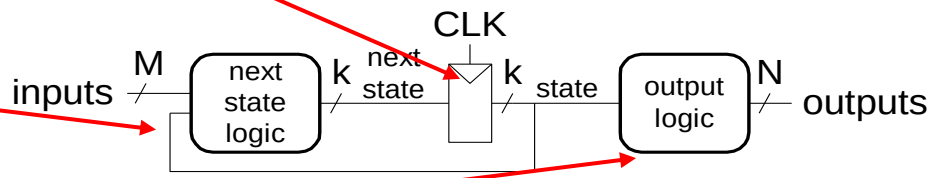
FSM μετρητή Modulo-3 σε VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity divideby3FSM is
  port(clk, reset: in  STD_LOGIC;
       y:          out STD_LOGIC);
end;

architecture synth of divideby3FSM is
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
  -- καταχωρητής κατάστασης
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

  -- λογική επόμενης κατάστασης
  nextstate <= S1 when state = S0 else
               S2 when state = S1 else
               S0;

  -- λογική εξόδου
  y <= '1' when state = S0 else '0';
end;
```

Δήλωση καταστάσεων: ορίζεται ένας νέος τύπος δεδομένων απαρίθμησης (enumeration), το statetype, με τρεις πιθανές τιμές: S0, S1 και S2.
Η τρέχουσα κατάσταση state και η επόμενη κατάσταση nextstate είναι εσωτερικά σήματα τύπου statetype.
Με τον τύπο δεδομένων απαρίθμησης αντί για τη ρητή επιλογή της κωδικοποίησης καταστάσεων, η VHDL απελευθερώνει το εργαλείο σύνθεσης, το οποίο μπορεί να διερευνήσει διάφορες κωδικοποιήσεις καταστάσεων προκειμένου να επιλέξει τη βέλτιστη από αυτές

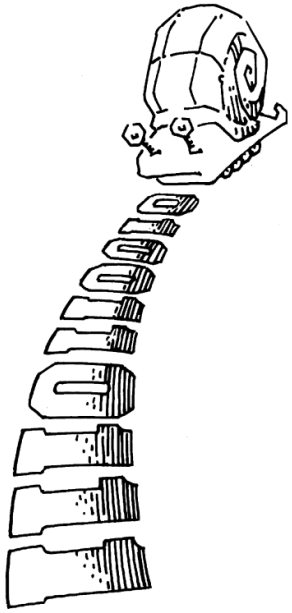
Synthesis:





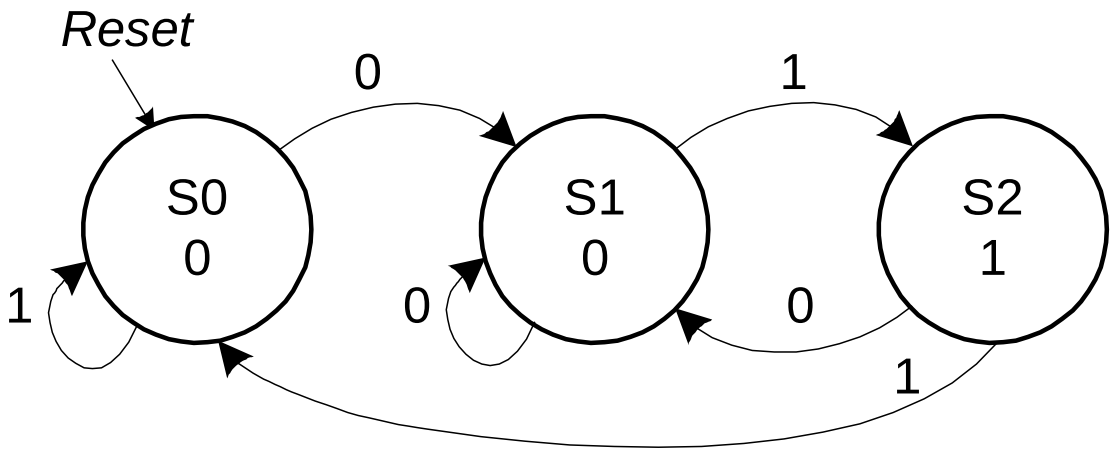
“01” Sequence Detector

Το «χαμογελαστό σαλιγκάρι» (sequence detector):
*Το σαλιγκάρι χαμογελάει όταν τα δύο τελευταία διαδοχικά bit πάνω από τα οποία έχει περάσει είναι **01**. Σχεδιάστε τη μηχανή FSM έτσι ώστε να υπολογίζει πότε το σαλιγκάρι πρέπει να χαμογελάει. Η έξοδος Y έχει την τιμή TRUE όταν το σαλιγκάρι χαμογελάει.*



Διάγραμμα καταστάσεων

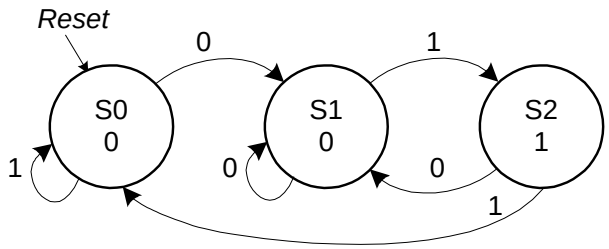
Moore FSM

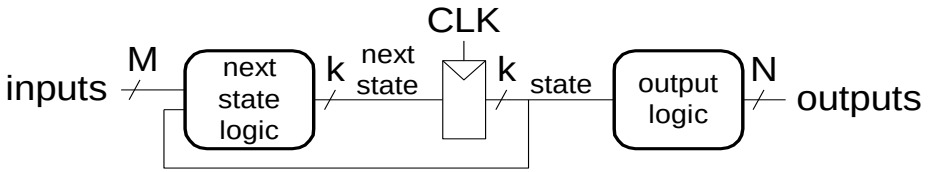


“01” Sequence Detector FSM: Moore

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity patternMoore is
  port(clk, reset: in STD_LOGIC;
       a: in STD_LOGIC;
       y: out STD_LOGIC);
end;

architecture synth of patternMoore is
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
-- καταχωρητής κατάστασης
  process(clk, reset) begin
    if reset then
       state <= S0;
    elsif rising_edge(clk) then
       state <= nextstate;
     end if;
  end process;
-- λογική επόμενης κατάστασης και λογική εξόδου
   process(all) begin
     y <= '0';
     nextstate <= S0;
     case state is
       when S0 =>
          if a then
             nextstate <= S0;
           else
             nextstate <= S1;
          end if;
       when S1 =>
          if a then
             nextstate <= S2;
          else
             nextstate <= S1;
          end if;
       when S2 =>
           y <= '1';
           if a then
             nextstate <= S0;
           else
              nextstate <= S1;
           end if;
       when others =>
           nextstate <= S0;
     end case;
   end process;
end;
```

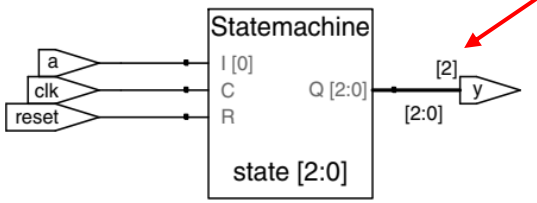
Moore FSM





Παρατηρήστε ότι το εργαλείο σύνθεσης χρησιμοποιεί κωδικοποίηση καταστάσεων μεγέθους 3-bit (Q[2:0])

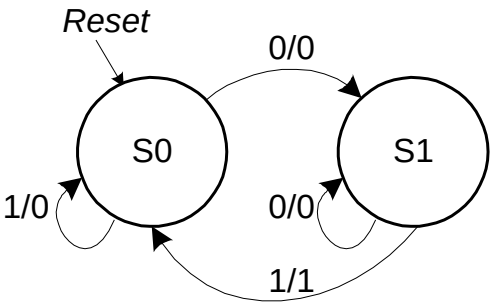
Synthesis:



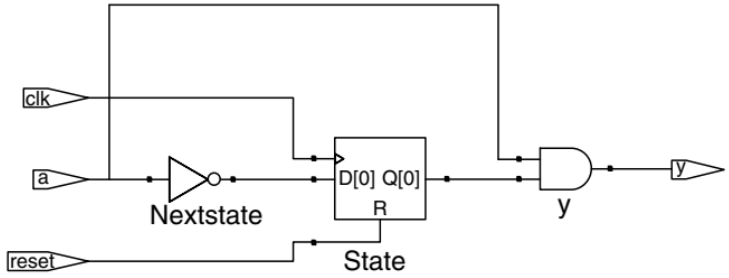
“01” Sequence Detector FSM: Mealy

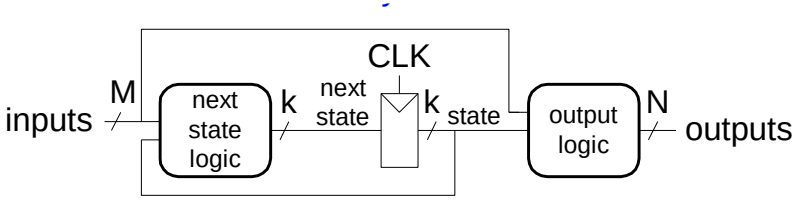
```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity patternMealy is
   port(clk, reset: in STD_LOGIC;
        a: in STD_LOGIC;
        y: out STD_LOGIC);
end;
architecture synth of patternMealy is
   type statetype is (S0, S1);
   signal state, nextstate: statetype;
begin
-- καταχωρητής κατάστασης
   process(clk, reset)
   begin
      if reset then
         state <= S0;
      elsif rising_edge(clk) then
         state <= nextstate;
      end if;
   end process;
-- λογική επόμενης κατάστασης
   process(all) begin
     y <= '0';
     nextstate <= S0;
     case state is
       when S0 =>
         if a then
            nextstate <= S0;
         else
            nextstate <= S1;
         end if;
       when S1 =>
         if a then
           y <= '1';
           nextstate <= S0;
         else
           nextstate <= S1;
         end if;
       when others =>
          nextstate <= S0;
     end case;
   end process;
end;
```

Mealy FSM



Synthesis:





Κλασσική «παγίδα» στη VHDL

- **Ελλιπής ανάθεση τιμής σε εντολές if και case**
- Εάν δεν βάλουμε στο κώδικα αρχικές αναθέσεις τιμών, το εργαλείο CAD δεν γνωρίζει τι να κάνει με τις εξόδους που δεν έχουν προσδιοριστεί για αυτά τα if/cases
- Το εργαλείο θα δημιουργήσει ένα στοιχείο κατάστασης για να κρατήσουν την παλαιά κατάσταση
  - Άλλωστε, έτσι φτιάχνουμε FF/Latches!
- Σε μια συνδυαστική process όμως, αυτό θα δημιουργήσει ανεπιθύμητα latches (κακή πρακτική σε FPGA design!!)

```
architecture EXAMPLE_BEH of EXAMPLE is
begin
  process (EN, A, B)
  begin
    X <= '0';  -- αρχικές τιμές
    Y <= '0';
    if (EN = '1') then
      X <= A;
      Y <= B;
    else
      X <= '1';
    end if;
  end process;
end EXAMPLE_BEH;
```

```
ASYNC: process (current_state, X_in)
  begin
    next_state <= S0;
    Y <= '0';
    case current_state is
      when S0 =>
        if (X_in = '0') then next_state <= S1;
        else next_state <= S0; end if;
      when S1 =>
        if (X_in = '1') then next_state <= S2;
        else next_state <= S1; end if;
      when S2 => Y <= '1';
        if (X_in = '0') then next_state <= S1;
        else next_state <= S0; end if;
        -- fail-safe behavior
      when others => next_state <= S0;
    end case;
  end process;
end BEHAVIORAL;
```

Μνήμες

Μνήμες στη VHDL

- Single-port RAM με ασύγχρονη ανάγνωση
- Υλοποίηση ως distributed RAM
  - Χρήση μόνο για μικρές μνήμες

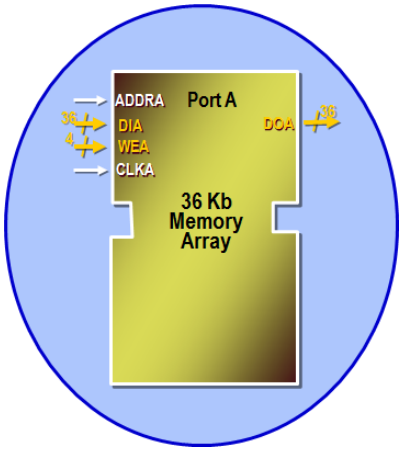
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity single_port_ram_asynch is
    port (CLK : in std_logic;
          WE  : in std_logic;
          A   : in std_logic_vector(5 downto 0);
          DI  : in std_logic_vector(15 downto 0);
          DO  : out std_logic_vector(15 downto 0));
end single_port_ram_asynch ;
architecture syn of single_port_ram_asynch is
    type ram_type is array (63 downto 0) of std_logic_vector (15 downto 0);
    signal RAM : ram_type;
begin
    process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            if (WE = '1') then
                RAM(to_integer(unsigned(A))) <= DI;
            end if;
        end if;
    end process;
    DO <= RAM(to_integer(unsigned(A)));
end syn;
```

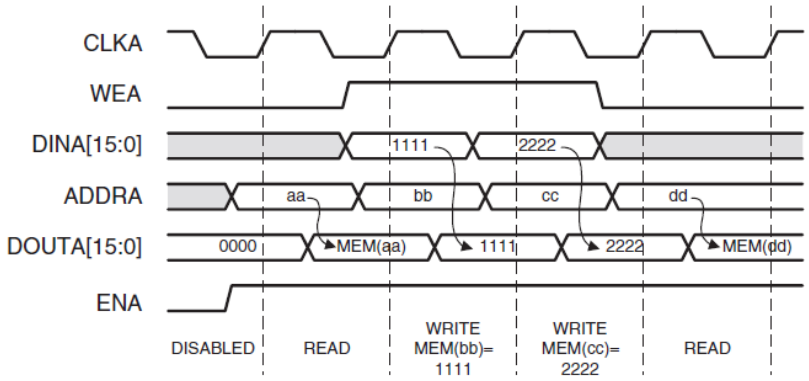
Η αποθήκευση της RAM αναπαρίσταται από ένα σήμα array

Μνήμες στη VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity single_port_ram_write_first is
    port (CLK : in std_logic;
          WE  : in std_logic;
          A   : in std_logic_vector(9 downto 0);
          DI  : in std_logic_vector(15 downto 0);
          DO  : out std_logic_vector(15 downto 0));
end single_port_ram_write_first ;
architecture syn of single_port_ram_write_first is
    type ram_type is array (1023 downto 0) of std_logic_vector (15 downto 0);
    signal RAM : ram_type;
    signal ram_data: std_logic_vector(15 downto 0);
begin
    process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            if (WE = '1') then
                RAM(to_integer(unsigned(A))) <= DI;
                 ram_data <=di;
             else
                 ram_data <=RAM(to_integer(unsigned(A)));
            end if;
        end if;
    end process;
    DO <= ram_data;
end syn;
```

- Single-port WRITE_FIRST
- Υλοποίηση ως Block RAM

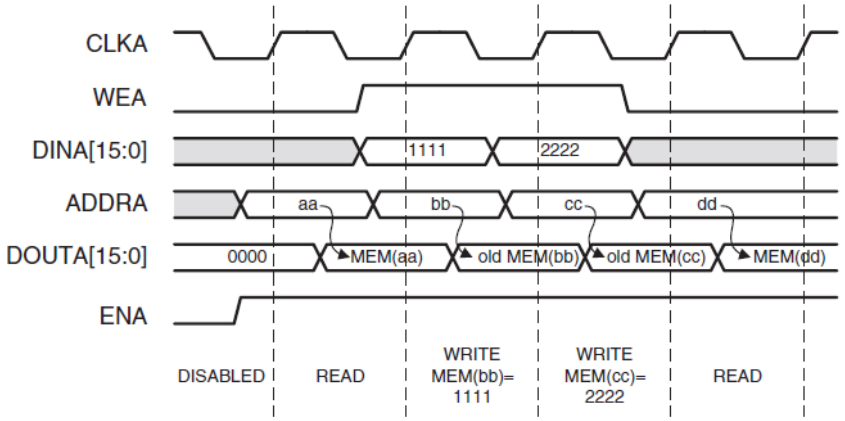




Μνήμες στη VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity single_port_ram_read_first is
    port (CLK : in std_logic;
          WE  : in std_logic;
          A   : in std_logic_vector(9 downto 0);
          DI  : in std_logic_vector(15 downto 0);
          DO  : out std_logic_vector(15 downto 0));
end single_port_ram_read_first ;
architecture syn of single_port_ram_read_first is
    type ram_type is array (1023 downto 0) of std_logic_vector (15
downto 0);
    signal RAM : ram_type;
    signal ram_data: std_logic_vector(15 downto 0);
begin
    process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            if (WE = '1') then
                RAM(to_integer(unsigned(A))) <= DI;
           end if;
           ram_data <=RAM(to_integer(unsigned(A)));
        end if;
    end process;
    DO <= ram_data;
end syn;
```

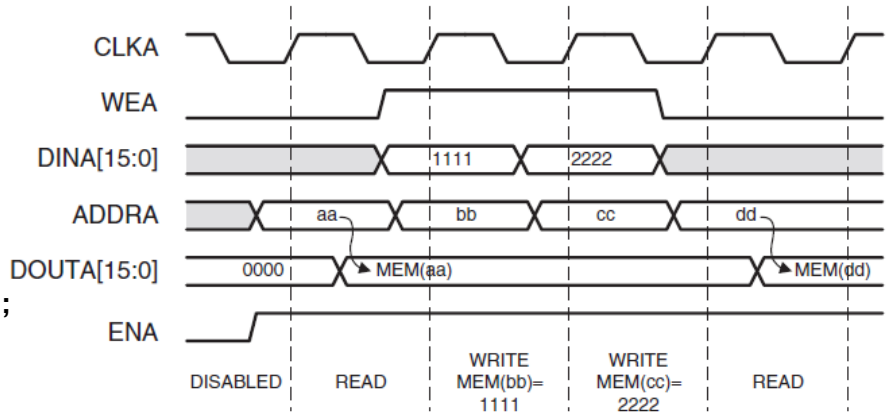
- Single-port READ_FIRST
- Υλοποίηση ως Block RAM



Μνήμες στη VHDL

- Single-port NO_CHANGE
- Υλοποίηση ως Block RAM

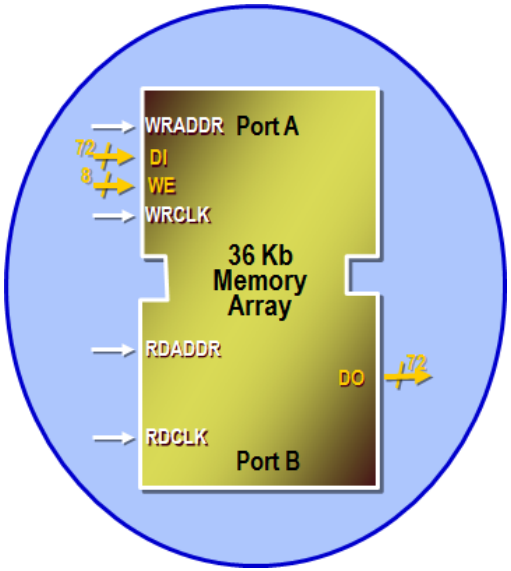
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity single_port_ram_no_change is
    port (CLK : in std_logic;
          WE  : in std_logic;
          A   : in std_logic_vector(9 downto 0);
          DI  : in std_logic_vector(15 downto 0);
          DO  : out std_logic_vector(15 downto 0));
end single_port_ram_no_change ;
architecture syn of single_port_ram_no_change is
    type ram_type is array (1023 downto 0) of std_logic_vector (15 downto
0);
    signal RAM : ram_type;
    signal ram_data: std_logic_vector(15 downto 0);
begin
    process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            if (WE = '1') then
                RAM(to_integer(unsigned(A))) <= DI;
            else
                ram_data <=RAM(to_integer(unsigned(A)));
            end if;
        end if;
    end process;
    DO <= ram_data;
end syn;
```



Μνήμες στη VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity simple_dual_port is
    port (CLK     : in std_logic;
          wea     : in std_logic;
          web     : in std_logic;
          addra   : in std_logic_vector(9 downto 0);
          addrb   : in std_logic_vector(9 downto 0);
          DI      : in std_logic_vector(15 downto 0);
          DO      : out std_logic_vector(15 downto 0));
end simple_dual_port ;
architecture syn of simple_dual_port is
    type ram_type is array (1023 downto 0) of std_logic_vector (15 downto
0);
    signal RAM : ram_type;
    signal ram_data: std_logic_vector(15 downto 0);
begin
    process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            if (wea = '1') then
                RAM(to_integer(unsigned(addra))) <= DI;
            end if;
            ram_data <=RAM(to_integer(unsigned(addrb)));
        end if;
    end process;
    DO <= ram_data;
end syn;
```

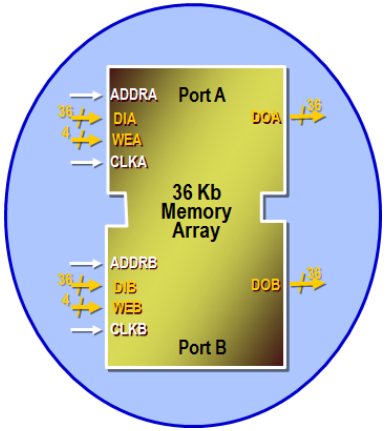
- Simple Dual-port
- Single clock
- Read-first για το port B
- Υλοποίηση ως Block RAM



Μνήμες στη VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity true_dual_port is
    port (clka    : in std_logic;
          clkb    : in std_logic;
          wea     : in std_logic;
          web     : in std_logic;
          addra   : in std_logic_vector(9 downto 0);
          addrb   : in std_logic_vector(9 downto 0);
          dina    : in std_logic_vector(15 downto 0);
          dinb    : in std_logic_vector(15 downto 0);
          douta   : out std_logic_vector(15 downto 0);
          doutb   : out std_logic_vector(15 downto 0)
          );
end true_dual_port;
```

- True Dual-port
- Dual clock
- Read-first για κάθε θύρα
- Υλοποίηση ως Block RAM




```
architecture syn of true_dual_port is
    type ram_type is array (1023 downto 0) of std_logic_vector
(15 downto 0);
    signal RAM : ram_type;
    signal ram_data_a: std_logic_vector(15 downto 0);
    signal ram_data_b: std_logic_vector(15 downto 0);
begin
    process (clka)
    begin
        if (clka'event and clka = '1') then
            if (wea = '1') then
                RAM(to_integer(unsigned(addra))) <= dina;
            end if;
            ram_data_a <=RAM(to_integer(unsigned(addra)));
        end if;
    end process;
    process (clkb)
    begin
        if (clkb'event and clkb = '1') then
            if (web = '1') then
                RAM(to_integer(unsigned(addrb))) <= dinb;
             end if;
             ram_data_b <=RAM(to_integer(unsigned(addrb)));
        end if;
    end process;
   douta <= ram_data_a;
   doutb <= ram_data_b;
end syn;
```

Μνήμες στη VHDL

```
library ieee; use ieee.numeric_std.all;
architecture ROM_based of seven_seg_decoder is
  type ROM_array is
      array (0 to 31) of std_logic_vector ( 7 downto 1 );
  constant ROM_content : ROM_array
    := ( 0 => "0111111", 1 => "0000110",
         2 => "1011011", 3 => "1001111",
         4 => "1100110", 5 => "1101101",
         6 => "1111101", 7 => "0000111",
         8 => "1111111", 9 => "1101111"
         10 to 15 => "1000000",
         16 to 31 => "0000000" );
begin
  seg <= ROM_content(to_integer(unsigned(blank & bcd)));
end architecture ROM_based;
```

Τύποι δεδομένων

Τύποι δεδομένων

- Η VHDL είναι μια “strongly typed” γλώσσα
  - Κάθε αντικείμενο (π.χ. σήμα ή μεταβλητή) μπορεί να λάβει τιμές μόνο από τον δηλωμένο τύπο
- Η VHDL περιέχει προκαθορισμένους τύπους
  - Δηλώνονται σε πακέτα (packages)
  - Πακέτο standard της βιβλιοθήκης ieee:
    - Τύποι: BIT, BOOLEAN, INTEGER και REAL
  - Πακέτο std_logic_1164 της βιβλιοθήκης ieee:
    - Τύποι: STD_LOGIC και STD_ULOGIC
  - Πακέτο numeric_std της βιβλιοθήκης ieee:
    - Τύποι: UNSIGNED και SIGNED
    - Ορίζει αριθμητικές πράξεις, συγκρίσεις και λογικούς τελεστές
- Ο χρήστης μπορεί να ορίσει δικούς του τύπους

Τύποι δεδομένων του IEEE Standard 1164

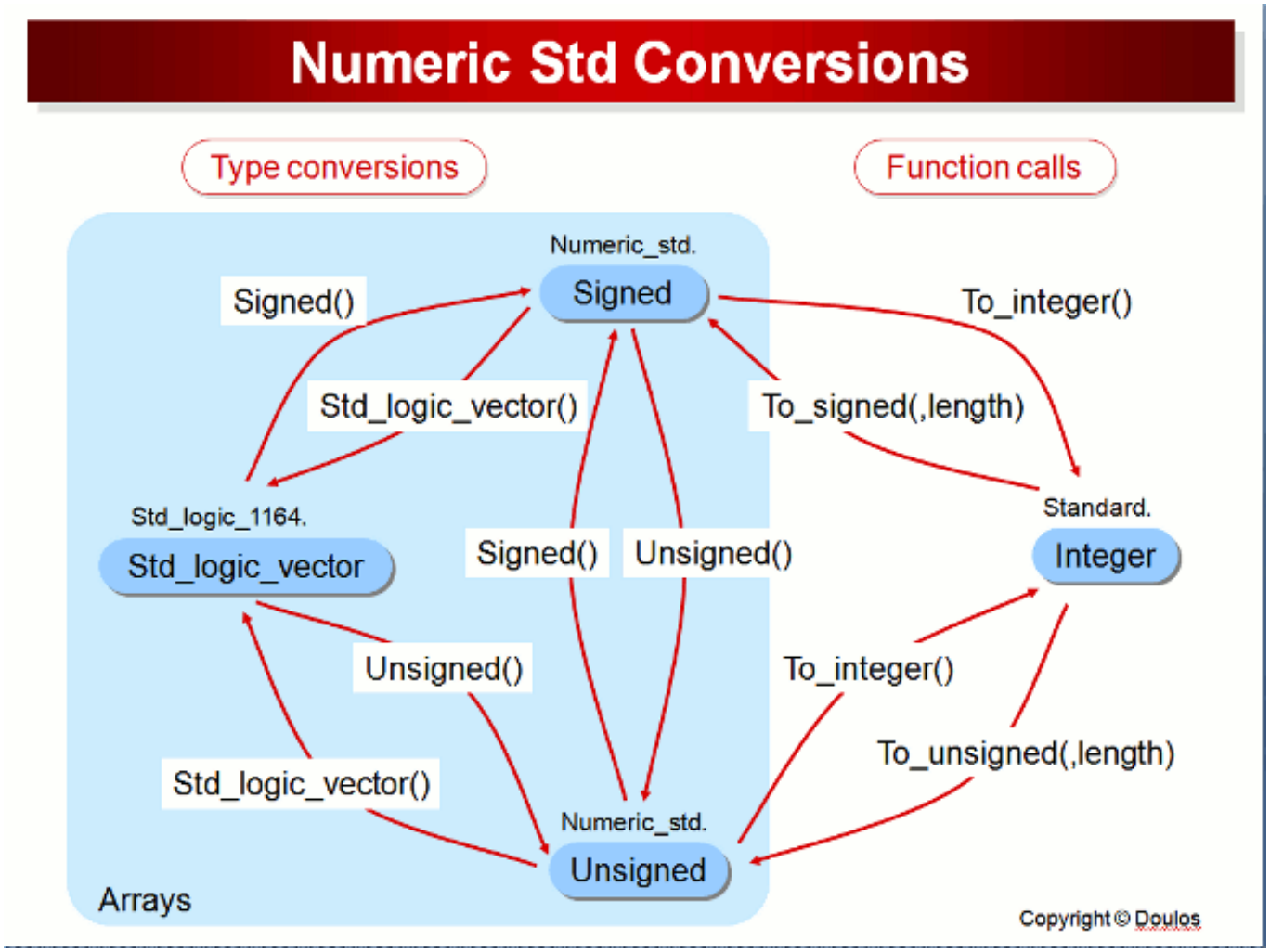
- Εννέα τύποι δεδομένων std_logic :
  - ‘U’, ‘X’ – uninitialized/unknown
  - ‘0’, ‘1’ – strongly-driven 0/1
  - ‘L’, ‘H’ – weakly-driven 0/1
  - ‘Z’, ‘W’ – High impedance/weakly driven
  - ‘-’ - don’t care
- Ο τύπος std_logic_vector είναι πίνακας από std_logic
- Πώς συμπεριλαμβάνουμε το πακέτο:

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

Τύποι signed και unsigned

- 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 bit1
- `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

Μετατροπή δεδομένων στο numeric_std



Μη-προσημασμένη πρόσθεση

- Χρήση λειτουργιών από το numeric_std

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

```
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);
```

Επέκταση με μηδενικά στα a και b κατά 1-bit

Το C είναι το MSB του αποτελέσματος

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;
```

Επέκταση μηδενός «με το χέρι»

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;
```

Αλλαγή μεγέθους

- Συνάρτηση RESIZE στο numeric_std

```
-- Id: R.1
function RESIZE (ARG: SIGNED; NEW_SIZE: NATURAL) return SIGNED;
-- Result subtype: SIGNED(NEW_SIZE-1 downto 0)
-- Result: Resizes the SIGNED vector ARG to the specified size.
--         To create a larger vector, the new [leftmost] bit positions
--         are filled with the sign bit (ARG'LEFT). When truncating,
--         the sign bit is retained along with the rightmost part.

-- Id: R.2
function RESIZE (ARG: UNSIGNED; NEW_SIZE: NATURAL) return UNSIGNED;
-- Result subtype: UNSIGNED(NEW_SIZE-1 downto 0)
-- Result: Resizes the SIGNED vector ARG to the specified size.
--         To create a larger vector, the new [leftmost] bit positions
--         are filled with '0'. When truncating, the leftmost bits
--         are dropped.
```

Προσημασμένη πρόσθεση

- Το αποτέλεσμα του + έχει ίδιο μέγεθος με τους τελεστέους
  - Εάν θέλουμε να συμπεριλάβουμε την υπερχείλιση επεκτείνουμε κατά 1

```
signal v1, v2 : signed(11 downto 0);
signal sum : signed(12 downto 0);
...
sum <= resize(v1, sum'length) + resize(v2, sum'length);
```

Επέκταση προσήμου αυτόματα με το resize function

- Για έλεγχο υπερχείλισης, σύγκριση προσήμων

```
signal x, y, z: signed(7 downto 0);
signal ovf : std_logic;
...
z <= x + y;
ovf <= (not x(7) and not y(7) and z(7))
       or (x(7) and y(7) and not z(7));
```

Παραμετροποιημένες Υπομονάδες

Παραμετροποιημένες υπομονάδες

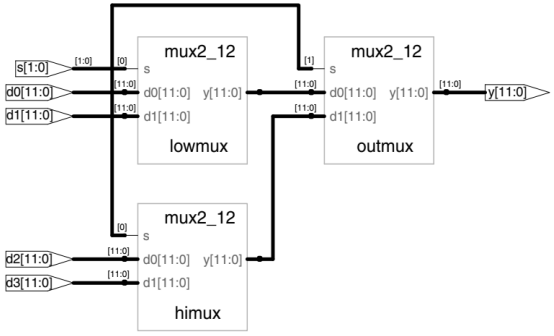
```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2 is
  generic(width: integer : = 8);
  port(d0,
    d1: in  STD_LOGIC_VECTOR(width-1 downto 0);
    s:  in  STD_LOGIC;
    y:  out STD_LOGIC_VECTOR(width-1 downto 0));
end;

architecture synth of mux2 is
begin
  y <= d1 when s else d0;
end;
```

Η εντολή `generic` περιλαμβάνει μια προεπιλεγμένη ακέραια τιμή (8) για το εύρος bit (width)

Παράδειγμα:MUX 4:1 των 12 bit (mux4_12): πρέπει να παρακάμψει το προεπιλεγμένο εύρος bit χρησιμοποιώντας τον τύπο **`generic map`** κατά τη χρήση του στοιχείου, όπως φαίνεται παρακάτω.

```
lowmux: mux2 generic map(12) port map(d0, d1, s(0), low);
himux:  mux2 generic map(12) port map(d2, d3, s(0), hi);
outmux: mux2 generic map(12) port map(low, hi, s(1), y);
```

Synthesis (mux4_12):




```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4_8 is
  port(d0, d1, d2,
       d3: in  STD_LOGIC_VECTOR(7 downto 0);
       s:  in  STD_LOGIC_VECTOR(1 downto 0);
       y:  out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture struct of mux4_8 is
  component mux2
   generic(width: integer := 8);
   port(d0,
        d1: in  STD_LOGIC_VECTOR(width-1 downto 0);
         s: in  STD_LOGIC;
         y: out STD_LOGIC_VECTOR(width-1 downto 0));
  end component;
  signal low, hi: STD_LOGIC_VECTOR(7 downto 0);
begin
  lowmux: mux2 port map(d0, d1, s(0), low);
  himux:  mux2 port map(d2, d3, s(0), hi);
  outmux: mux2 port map(low, hi, s(1), y);
end;
```

MUX 4:1 των 8 bit (mux4_8): χρησιμοποιεί 3 φορές το ίδιο component του mux2 με το προεπιλεγμένο εύρος bit

Παραμετροποιημένες υπομονάδες

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD_UNSIGNED.all;

entity decoder is
  generic(N: integer := 3);
  port(a: in  STD_LOGIC_VECTOR(N-1 downto 0);
       y: out STD_LOGIC_VECTOR(2**N-1 downto 0));
end;

architecture synth of decoder is
begin
  process(all)
  begin
    y <= (OTHERS => '0');
    y(TO_INTEGER(a)) <= '1';
  end process;
end;
```

Αποκωδικοποιητής

Ένας μεγάλος αποκωδικοποιητής $N{:}2^N$ είναι δύσκολο να οριστεί με εντολές `case`, αλλά εύκολο με χρήση παραμετροποιημένου κώδικα ο οποίος απλώς θέτει την τιμή του κατάλληλου bit εξόδου ίση με 1.

Χρησιμοποιεί αναθέσεις τιμών με άμεση εφαρμογή για να μηδενίσει όλα τα bit, και μετά αλλάζει την τιμή του κατάλληλου bit σε 1

Προσημασμένος πολλαπλασιασμός

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity MUL32 is
  generic (WIDTH : positive := 32);
  port (
    A:   in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    B:   in STD_LOGIC_VECTOR (WIDTH-1 downto 0);
    P:  out STD_LOGIC_VECTOR (2*WIDTH-1 downto 0));
end MUL32;
architecture BEHAVIORAL of MUL32 is
begin
  MUL32: process (A, B)
  variable A_s: SIGNED (WIDTH-1 downto 0);
  variable B_s: SIGNED (WIDTH-1 downto 0);
  variable P_s: SIGNED (2*WIDTH-1 downto 0);
  begin
    A_s := signed(A);
    B_s := signed(B);
    P_s := A_s * B_s;
    P <= std_logic_vector(P_s);
end process;
end BEHAVIORAL;
```

Προγράμματα δοκιμών (Testbenches)

Προγράμματα δοκιμών (Testbenches)

- Κώδικας HDL που δοκιμάζει μια άλλη μονάδα: *device under test* (DUT)
- **Δεν είναι συνθέσιμος κώδικας**
- **Τύποι:**
  - Απλό
  - Αυτοελεγχόμενο (Self-checking)
  - Αυτοελεγχόμενο με διανύσματα δοκιμής (test vectors)

Testbenches

- Γράψτε το testbench για τη δοκιμή της υπομονάδας syllyfunction

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture synth of sillyfunction is
begin
y <= (not a and not b and not c) or
     (a and not b and not c) or
     (a and not b and c);
end;
```

Testbench 1: Απλό Testbench

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity testbench1 is -- χωρίς εισόδους ή εξόδους
end;

architecture sim of testbench1 is
  component sillyfunction
    port(a, b, c: in  STD_LOGIC;
         y:       out STD_LOGIC);
  end component;
  signal a, b, c, y: STD_LOGIC;
begin
  -- δημιουργία στοιχείου της διάταξης υπό δοκιμή
  dut: sillyfunction port map(a, b, c, y);

  -- εφαρμογή εισόδων, μία κάθε φορά
  process begin
    a <= '0'; b <= '0'; c <= '0'; wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    b <= '1'; c <= '0';           wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    a <= '1'; b <= '0'; c <= '0'; wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    b <= '1'; c <= '0';           wait for 10 ns;
    c <= '1';                     wait for 10 ns;
    wait; -- αναμονή επ' αόριστον
  end process;
end;
```

```
entity sillyfunction is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;
```

Testbench 2: Self-Checking Testbench

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity testbench2 is -- χωρίς εισόδους ή εξόδους
end;

architecture sim of testbench2 is
  component sillyfunction
    port(a, b, c: in  STD_LOGIC;
         y:       out STD_LOGIC);
  end component;
  signal a, b, c, y: STD_LOGIC;
begin
  -- δημιουργία στοιχείου της διάταξης
  -- υπό δοκιμή
  dut: sillyfunction port map(a, b, c, y);

  /* εφαρμογή εισόδων, μία κάθε φορά,
   για τον έλεγχο των αποτελεσμάτων */
  process begin
   a <= '0'; b <= '0'; c <= '0'; wait for 10 ns;
     assert y = '1' report "000 failed.";
   c <= '1';                     wait for 10 ns;
     assert y = '0' report "001 failed.";
   b <= '1'; c <= '0';           wait for 10 ns;
     assert y = '0' report "010 failed.";
   c <= '1';                     wait for 10 ns;
     assert y = '0' report "011 failed.";
   a <= '1'; b <= '0'; c <= '0'; wait for 10 ns;
     assert y = '1' report "100 failed.";
   c <= '1';                     wait for 10 ns;
     assert y = '1' report "101 failed.";
   b <= '1'; c <= '0';           wait for 10 ns;
     assert y = '0' report "110 failed.";
   c <= '1';                     wait for 10 ns;
     assert y = '0' report "111 failed.";
   wait; -- αναμονή επ' αόριστον
 end process;
end;
```

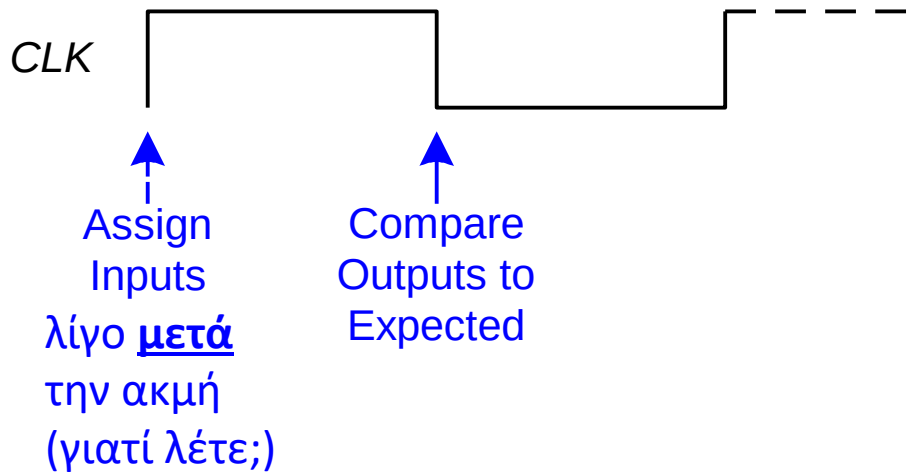
- Η εντολή `assert` ελέγχει μια συνθήκη και τυπώνει το μήνυμα που προσδιορίζεται στην πρόταση `report` αν δεν ικανοποιείται η συνθήκη
- Οι εντολές `assert` έχουν νόημα <u>μόνο στην προσομοίωση και όχι στη σύνθεση</u>

Testbench 3: Testbench w/ Testvectors

- Αρχείο testvectors: είσοδοι και αναμενόμενες έξοδοι
- Testbench:
  1. Δημιουργία clock για την ανάθεση τιμών στις εισόδους και διάβασμα των εξόδων
     - Πιθανά και αρχικοποίηση με reset
  2. Ανάγνωση αρχείου testvectors σε array και εκτέλεση δοκιμής
  3. Σύγκριση αποτελεσμάτων με τις αναμενόμενες τιμές και άμεση αναφορά πιθανού σφάλματος
  4. Σύνοψη αποτελεσμάτων και πιθανών σφαλμάτων

Testbench 3: Testbench w/ Testvectors

- **Testbench clock:**
  - Ανάθεση **inputs** (στο **rising edge**)
  - Σύγκριση **outputs** με τις αναμενόμενες εξόδους (στο **falling edge)**.



- Χρήση του clock που ορίζεται στο testbench clock **και ως clock** για τα σύγχρονα ακολουθιακά κυκλώματα

Testbench 3: Αρχείο testvectors

- File: `example.txt`
- Περιέχει τα διανύσματα δοκιμής abc_yexpected

```
//abc_yexpected
  000_1
  001_0
  010_0
  011_0
  100_1
  101_1
  110_0
  111_0
```

1. Δημιουργία Clock (και reset)

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_TEXTIO.ALL; use STD.TEXTIO.all;

entity testbench3 is -- χωρίς εισόδους ή εξόδους
end;

architecture sim of testbench3 is
  component sillyfunction
    port(a, b, c: in  STD_LOGIC;
         y:       out STD_LOGIC);
  end component;
  signal a, b, c, y: STD_LOGIC;
  signal y_expected: STD_LOGIC;
  signal clk, reset: STD_LOGIC;
begin
  -- δημιουργία στοιχείου της διάταξης υπό δοκιμή
  dut: sillyfunction port map(a, b, c, y);

  -- παραγωγή ρολογιού με περίοδο 10 ns
  process begin
    clk <= '1'; wait for 5 ns;
    clk <= '0'; wait for 5 ns;
  end process;

  -- κατά την έναρξη της δοκιμής, ενεργοποιούμε
  -- το reset για 27 ns
  process begin
    reset <= '1'; wait for 27 ns; reset <= '0';
    wait;
  end process;
```

2. Ανάγνωση testvectors σε array & εκτέλεση δοκιμής

```
-- φορτώνουμε τα διανύσματα δοκιμής και
-- εκτελούμε τις δοκιμές
process is
  file tv: text;
  variable L: line;
  variable vector_in: std_logic_vector
                      (2 downto 0);
  variable dummy: character;
  variable vector_out: std_logic;
  variable vectornum: integer := 0;
  variable errors: integer := 0;
begin
  FILE_OPEN(tv, "example.tv", READ_MODE);
  while not endfile(tv) loop
    -- αλλαγή διανυσμάτων κατά την
       ανερχόμενη ακμή
    wait until rising_edge(clk);
    /* ανάγνωση της επόμενης γραμμής των
     διανυσμάτων δοκιμής και διαχωρισμός
     σε δύο κομμάτια */
    readline(tv, L);
    read(L, vector_in);
    read(L, dummy); -- ο χαρακτήρας
                       υπογράμμισης αγνοείται
    read(L, vector_out);
    (a, b, c) <= vector_in(2 downto 0)
                 after 1 ns;
    y_expected <= vector_out after 1 ns;
```

Ανάγνωση εισόδων δοκιμής

Εφαρμογή εισόδων δοκιμής **1ns μετά το rising_edge (clk)**

Ανάγνωση αναμενόμενων αποτελεσμάτων 1ns μετά την εφαρμογή των εισόδων

3. Σύγκριση αποτελεσμάτων δοκιμής με τις αναμενόμενες τιμές & άμεση αναφορά σφάλματος

```
  -- έλεγχος αποτελεσμάτων κατά την
  -- κατερχόμενη ακμή
  wait until falling_edge(clk);

  if y /= y_expected then
    report "Error: y = " &
                  std_logic'image(y);
    errors := errors + 1;
  end if;

  vectornum := vectornum + 1;
end loop;
```

Σύγκριση εξόδου με την αναμενόμενη στο falling_edge (clk)

Τα πιθανά σφάλματα αναφέρονται ΑΜΕΣΩΣ μόλις συμβούν

4. Σύνοψη αποτελεσμάτων & αναφορά πιθανών σφαλμάτων

```
-- σύνοψη των αποτελεσμάτων στο τέλος
-- της προσομοίωσης
if (errors = 0) then
  report "NO ERRORS -- " &
         integer'image(vectornum) &
         " tests completed successfully."
         severity failure;
else
  report integer'image(vectornum) &
  " tests completed, errors =" &
  integer'image(errors)
  severity failure;
end if;
```

Το testbench του παραδείγματος είναι μάλλον υπερβολικό για ένα τόσο απλό κύκλωμα.
Όμως, μπορείτε να το τροποποιήσετε εύκολα για να γράψετε δοκιμές πιο πολύπλοκων κυκλωμάτων!