Περίγραμμα Διάλεξης (Ι)

- Τι είναι η VHDL;
- Πλεονεκτήματα της VHDL στη σχεδίαση κυκλωμάτων VLSI.
- Βασική δομή κώδικα VHDL.
- Τύποι δεδομένων της VHDL.
- Τύποι σημάτων της VHDL.
- Καταστάσεις σημάτων θύρας της VHDL.
- Τελεστές της VHDL.
- Περιγραφή ροής δεδομένων στη VHDL.

Περίγραμμα Διάλεξης (2)

- Υποκυκλώματα.
- Περιγραφή δομής στη VHDL.
- Βιβλιοθήκες και Πακέτα.
- Σύγχρονες ή παράλληλες εντολές.
- Επαναληπτική ανάθεση και δημιουργία στιγμιότυπων.
- Παραμετρική περιγραφή.
- Η δομή της διαδικασίας.
- Ακολουθιακές εντολές.
- Περιγραφή συμπεριφοράς στη VHDL.

Περίγραμμα Διάλεξης (3)

- Προκαθορισμένες τιμές σημάτων στην περιγραφή συμπεριφοράς.
- Το φαινόμενο της μνήμης.
- Περιγραφή βασικών ακολουθιακών στοιχείων.
- Αλλαγή στην ακμή του ρολογιού.
- Ασύγχρονες είσοδοι.
- Περιγραφή καταχωρητών.
- Περιγραφή μετρητών.
- Μηχανές πεπερασμένων καταστάσεων.
- Αυτοματοποίηση προσομοίωσης με testbench.

Τι είναι η VHDL;

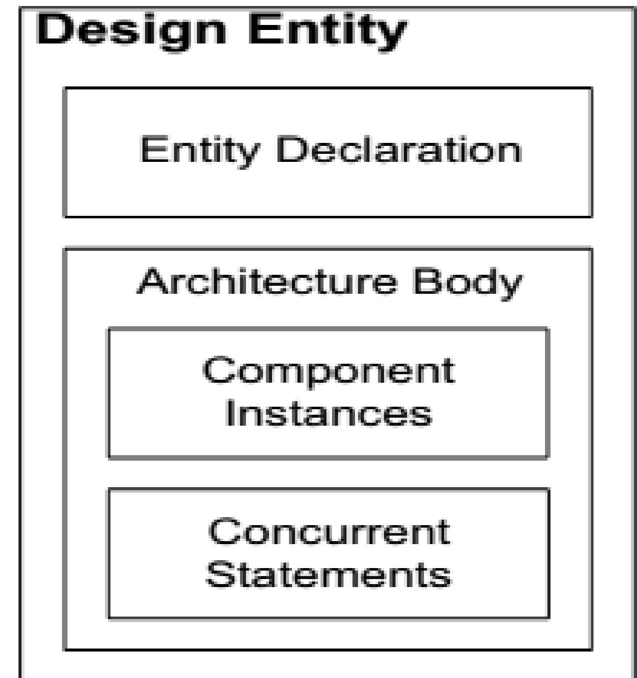
- Ακρωνύμιο: **VHSIC** (Very High Speed Integrated Circuit) **Hardware Description Language**.
- Γλώσσα περιγραφής κυκλωμάτων ή υλικού.
- Περιγραφή κυκλωμάτων ως προς τη δομή τους (**structural**), τη ροή δεδομένων (**dataflow**) ή τη συμπεριφορά (**behavioral**).
- Δυνατότητα περιγραφής του χρονισμού του κυκλώματος (**timing**) για εκτέλεση προσομοίωσης.
- Χρησιμοποιείται σε συνδυασμό με έναν μεταφραστή για την σύνθεση κυκλωμάτων σε ολοκληρωμένα ειδικής σχεδίασης.

Πλεονεκτήματα της VHDL στη σχεδίαση κυκλωμάτων VLSI

- Αποτελεί βιομηχανικό πρότυπο και ως εκ τούτου είναι ανεξάρτητη του εργαλείου σχεδίασης CAD και του τρόπου υλοποίησης του κυκλώματος.
- Η χρήση κώδικα αντί σχηματικών είναι αποτελεσματικότερη για την σχεδίαση μεγάλων και πολύπλοκων κυκλωμάτων κυρίως λόγω της ευκολότερης διαχείρισης και τροποποίησης της σχεδίασης.
- Δυνατότητα επαναχρησιμοποίησης σχεδιάσεων και πρόσβαση σε έτοιμες βιβλιοθήκες τρίτων.
- Η δυνατότητα περιγραφής ενός κυκλώματος σε επίπεδο συμπεριφοράς επιτρέπει τη σχεδίαση σε υψηλότερα επίπεδα αφαίρεσης, όπου ο μεταφραστής δημιουργεί αυτόματα το κύκλωμα με χρήση έτοιμων βιβλιοθηκών και τεχνικών σύνθεσης.
- Γνωστές δομές γλωσσών προγραμματισμού (αν και με διαφορετική φιλοσοφία καθώς όλες οι έννοιες που ενσωματώνει αφορούν το υλικό).
- Όλα τα βασικά πλεονεκτήματα των υψηλού επιπέδου γλωσσών προγραμματισμού
  - Δόμηση
  - Παραμετροποίηση
  - Βρόχοι επανάληψης
  - Εντολές συνθήκης
  - Ιεράρχηση
- Πλέον στην πράξη η ανάπτυξη ψηφιακών κυκλωμάτων γίνεται σχεδόν αποκλειστικά σε γλώσσες περιγραφής υλικού.

Βασική δομή κώδικα VHDL

```
ENTITY circuit_name IS
    PORT (signal_name: {mode} {type};
              signal_name: {mode} {type});
END circuit_name;

ARCHITECTURE architecture_name OF circuit_name IS
    [SIGNAL declarations;]
    [CONSTANT declarations;]
    [TYPE declarations;]
    [COMPONENT declarations;]

    BEGIN
          [COMPONENT instantiation statements;]
          [concurrent assignment statements;]
          [PROCESS statements;]

END architecture_name;
```



Παρατηρήσεις

- Η VHDL δεν είναι case sensitive.
- Στο σώμα της αρχιτεκτονικής γίνεται η περιγραφή του κυκλώματος ανάλογα με τον επιθυμητό τρόπο (δομή, ροή δεδομένων, συμπεριφορά).
- Μία οντότητα σχεδίασης μπορεί να έχει παραπάνω από μία αρχιτεκτονικές οι οποίες θα περιγράφουν με διαφορετικό τρόπο το ίδιο κύκλωμα.
- Η αρχιτεκτονική μπορεί να έχει οποιοδήποτε όνομα, αλλά συνήθως δίνουμε κάποιο που να αντιπροσωπεύει τον τρόπο περιγραφής του κυκλώματος (π.χ. structural, behavioral, dataflow).
- Όλες οι εντολές που βρίσκονται στο κυρίως σώμα της αρχιτεκτονικής, αναφέρονται ως σύγχρονες ή concurrent και εκτελούνται παράλληλα από τον μεταφραστή.
- Οι μοναδικές ακολουθιακές ή sequential εντολές οι οποίες εκτελούνται σειριακά από τον μεταφραστή είναι αυτές που βρίσκονται εντός ενός PROCESS, η οποία όμως με τη σειρά της αποτελεί μια σύγχρονη εντολή καθώς το τελικό της αποτέλεσμα εκτελείται παράλληλα με τις υπόλοιπες σύγχρονες εντολές από τον μεταφραστή.
- Το τελικό αποτέλεσμα είναι μοναδικό και δημιουργείται μετά το τέλος της επεξεργασίας όλων των εντολών, με τον τρόπο που αναφέραμε παραπάνω, από τον μεταφραστή.

Τύποι δεδομένων της VHDL

- Ο βασικότερος τύπος δεδομένων της VHDL είναι ο τύπος σήματος ή SIGNAL, ο οποίος στην περιγραφή υλικού παίζει το ρόλο που έχει ο τύπος της μεταβλητής στις συμβατικές γλώσσες προγραμματισμού.
- Στην VHDL υπάρχουν επίσης και οι γνωστοί σας τύποι CONSTANT και VARIABLE οι οποίοι δρουν βοηθητικά στη σύνταξη προγραμμάτων και δεν αντιστοιχούν σε κάποια έννοια του υλικού.
- Εάν ο χρήστης επιθυμεί να ορίσει κάποιο δικό του τύπο, χρησιμοποιεί την έκφραση TYPE (όπως θα δούμε και παρακάτω για τον ορισμό πινάκων (arrays) και σημάτων απαρίθμησης (enumeration) για μηχανές πεπερασμένων καταστάσεων).

Τύποι σημάτων της VHDL (1)

- Ο πιο συνηθισμένος τύπος σήματος είναι ο STD_LOGIC (και STD_LOGIC_VECTOR εάν πρόκειται για διάνυσμα), του οποίου οι δυνατές τιμές είναι '0', 'I', 'Z' (υψηλή σύνθετη αντίσταση), '-' (αδιάφορη), 'L' (ασθενές 0), 'H' (ασθενές 1), 'U' (μη αρχικοποιημένη), 'X' (άγνωστη), 'W' (ασθενής άγνωστη).

- Αποτελεί επέκταση του παλαιότερου τύπου BIT (και BIT_VECTOR αντίστοιχα) ο οποίος μπορούσε να λάβει μόνο τις τιμές '0' και 'I'.

- Απαιτεί τις δηλώσεις βιβλιοθήκης:
  LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  πριν από τη δήλωση της αρχικής οντότητας

Τύποι σημάτων της VHDL (2)

- Ορισμός διανυσμάτων:
  STD_LOGIC_VECTOR(low_index TO high_index)
  STD_LOGIC_VECTOR(high_index DOWNTO low_index)

- Σε ένα διάνυσμα σημάτων μπορούμε να χρησιμοποιήσουμε οποιοδήποτε υποσύνολο bits ως ένα ξεχωριστό διάνυσμα (slice). Για παράδειγμα εάν ορίσουμε
  SIGNAL C: STD_LOGIC_VECTOR(7 DOWNTO 0)
  Κάθε μία από τις παρακάτω εκφράσεις είναι ορθή:
  C(7 DOWNTO 4)
  C(0)

- Η τιμή για δεδομένα 1 bit αναγράφεται πάντοτε μέσα σε αποστρόφους (π.χ. ‘0’), ενώ για τα δεδομένα πολλών bit χρησιμοποιούμε εισαγωγικά “0101”.

- Ένας ακόμη χρήσιμος τύπος, κυρίως για την σχεδίαση μετρητών, είναι ο τύπος INTEGER τη χρήση του οποίου θα δούμε στη συνέχεια της διάλεξης.

Καταστάσεις σημάτων θύρας (port) στη VHDL

- IN: Σήμα εισόδου σε μια οντότητα
- OUT: Σήμα εξόδου σε μια οντότητα
- BUFFER: Σήμα εξόδου από μια οντότητα το οποίο όμως ταυτόχρονα χρησιμοποιείται και στην παραγωγή άλλου εσωτερικού σήματος μέσα στην ίδια οντότητα
- INOUT: Σήμα εισόδου και εξόδου ταυτόχρονα (bidirectional)

Τελεστές της VHDL

- Αριθμητικοί: +, -, *, /, ** (δύναμη), & (συνένωση ή concatenation), ABS (απόλυτη τιμή), MOD (υπόλοιπο)
- Σχεσιακοί: =, /=, <, <=, >, >=
- Λογικοί: NOT, AND, OR, NAND, NOR, XOR, XNOR
- Οι αριθμητικοί τελεστές έχουν την υψηλότερη σχετική προτεραιότητα και οι λογικοί(εκτός του NOT) τη χαμηλότερη
- **Προσοχή**: Οι λογικοί τελεστές δεν έχουν κάποια προτεραιτότητα μεταξύ τους (όπως συμβαίνει με τις λογικές εκφράσεις όπου συνήθως η πράξη AND προηγείται της πράξης OR) και θα πρέπει να χρησιμοποιούνται παρενθέσεις για την περιγραφή της επιθυμητής λειτουργίας

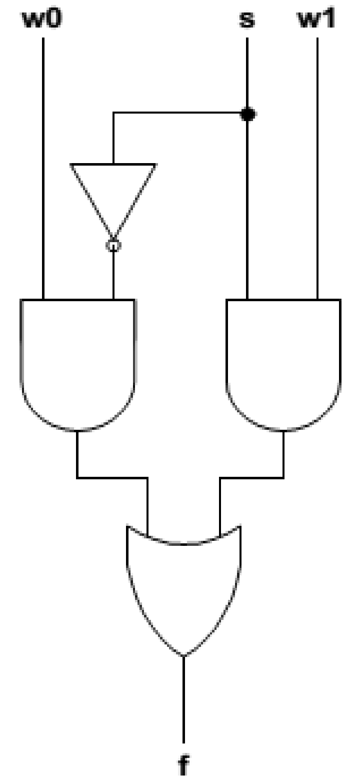
  π.χ. $A \cdot B + C' \cdot D \Rightarrow$ **A AND B OR NOT C AND D (ΛΑΘΟΣ)**

  (A AND B) OR (NOT C AND D) (ΣΩΣΤΟ)

Περιγραφή ροής δεδομένων στη VHDL (1)

- Η περιγραφή της ροής δεδομένων στη VHDL γίνεται με χρήση των λογικών τελεστών και ενός τελεστή ανάθεσης σήματος (**<=**).
- Παράδειγμα: πολυπλέκτης 2-σε-1

```
ENTITY mux2to1 IS
        PORT ( w0, w1, s: IN STD_LOGIC;
                   f: OUT STD_LOGIC);
END mux2to1;

ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
        f<=(NOT s AND w0) OR (s AND w1);
END dataflow;
```



Περιγραφή ροής δεδομένων στη VHDL (2)

- Τα σήματα τύπου STD_LOGIC και STD_LOGIC_VECTOR μπορούν να χρησιμοποιηθούν σε συνδυασμό με αριθμητικούς και συγκριτικούς τελεστές εάν περιληφθεί στο πρόγραμμα κάποια από τις ακόλουθες δηλώσεις:

```
USE ieee.std_logic_signed.all;
USE ieee.std_logic_unsigned.all;
```

  Οι οποίες επιτρέπουν να χρησιμοποιηθούν προσημασμένοι ή μη προσημασμένοι δυαδικοί αριθμοί.

- Με τον τρόπο αυτό περιγράφονται κυκλώματα όπως αθροιστές και συγκριτές που βρίσκονται σε υψηλότερα επίπεδα ιεραρχίας από το επίπεδο λογικής

Περιγραφή ροής δεδομένων στη VHDL (3)

- Παράδειγμα: αθροιστής 4-bit

```
ENTITY adder4 IS
    PORT (ci : IN STD_LOGIC;
            x, y : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
            s : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
            co  : OUT STD_LOGIC);
END adder4;

ARCHITECTURE dataflow OF adder4 IS
    SIGNAL sum : STD_LOGIC_VECTOR(4 DOWNTO 0);
    BEGIN
         sum <= ('0' & x) + y + ci;
         s <= sum(3 DOWNTO 0);
         co <= sum(4);
END dataflow;
```

Υποκυκλώματα

- Κάθε οντότητα στη VHDL μπορεί να λειτουργήσει ως υποκύκλωμα (component) σε μια οντότητα υψηλότερου επιπέδου.

- Ο τρόπος με τον οποίο δηλώνουμε ένα **component** είναι ο ακόλουθος:

  COMPONENT component_name

  PORT(local port list)

  END COMPONENT;

  Ενώ η δημιουργία στιγμιοτύπου υποκυκλώματος (component instantiation) πραγματοποιείται ως εξής:

  Label: component_name **PORT MAP** (port association list);

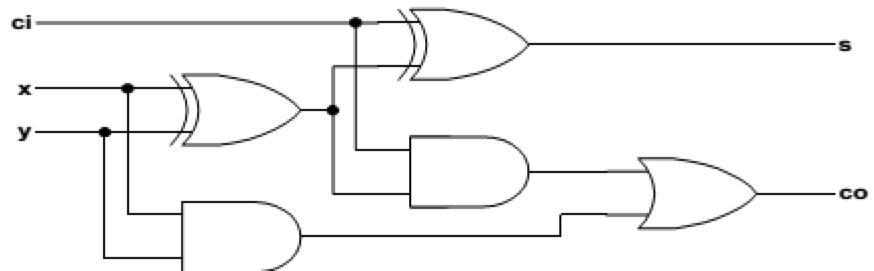
- **Προσοχή: Απαγορεύεται η απευθείας αντιστοίχισημιας σταθερής τιμής στη θύρα ενός υποκυκλώματος, και αυτό θα πρέπει να γίνει με τον ορισμό ενός κατάλληλου βοηθητικού σήματος**
  π.χ. PORT MAP (‘1’, x, y, s)
  high <= ‘1’; … PORT MAP (high, x, y, s)

Περιγραφή δομής στη VHDL (1)

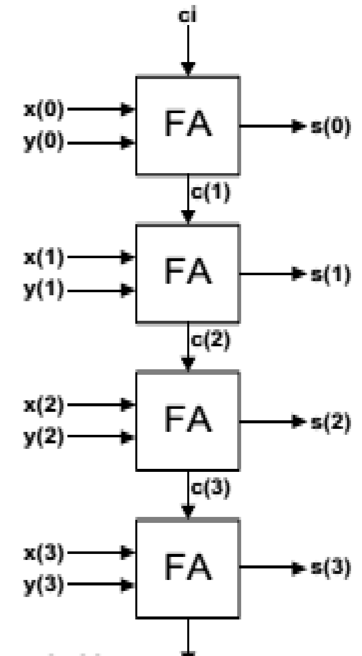
- Βασίζεται στον ιεραρχικό ορισμό υποκυκλωμάτων και τη μεταξύ τους διασύνδεση στο ανώτερο επίπεδο

- Παράδειγμα: αθροιστής 4-bit
  1st step: full adder 1 bit

```
ARCHITECTURE dataflow OF full_adder IS
    BEGIN
        s <= x XOR y XOR ci;
        co <= (x AND y) OR (ci AND x) OR (ci AND y);
END dataflow;
```



Περιγραφή δομής στη VHDL (2)




```
ARCHITECTURE structure OF adder4 IS
    SIGNAL c : STD_LOGIC_VECTOR(1 TO 3);
    COMPONENT fulladd
            PORT (ci, x, y : IN STD_LOGIC;
            s, co : OUT STD_LOGIC);
    END COMPONENT;

BEGIN
    stage0 :fulladd PORT MAP (ci, x(0), y(0), s(0), c(1));
    stage1: fulladd PORT MAP (c(1), x(1), y(1), s(1), c(2));
    stage2: fulladd PORT MAP (c(2), x(2), y(2), s(2), c(3));
    stage3: fulladd PORT MAP (ci => c(3), co => co, x => x(3), y => y(3), s =>
    s(3));
END structure
```

Βιβλιοθήκες και πακέτα

- Οι δηλώσεις των υποκυκλωμάτων μπορούν να γίνουν συγκεντρωτικά σε ένα πακέτο (package) της VHDL. Ένα πακέτο αποτελεί ένα αρχείο κώδικα στο οποίο ορίζονται συγκεντρωτικά διάφορες δομές της γλώσσας (π.χ. τύποι δεδομένων) και το οποίο μπορεί στη συνέχεια να χρησιμοποιηθεί από οποιοδήποτε άλλο αρχείο κώδικα.

- Ορισμός πακέτου:

```
PACKAGE package_name IS
[TYPE declarations;]
[SIGNAL declarations;]
[COMPONENT declarations;]
END package_name;
```

- Η πρόσβαση σε ένα υπάρχον πακέτο γίνεται με τη δήλωση:

```
LIBRARY library_name;
USE library_name.package_name.all;
```

όπου το όνομα βιβλιοθήκης αντιπροσωπεύει τη φυσική τοποθεσία στους καταλόγους αρχείων όπου βρίσκεται το επιθυμητό πακέτο.

- Ειδική περίπτωση βιβλιοθήκης είναι η βιβλιοθήκη "work" που αντιπροσωπεύει τον κατάλογο εργασίας(working directory), και η πρόσβαση σε οποιοδήποτε πακέτο που έχει δημιουργηθεί στο χώρο αυτό γίνεται απλά με τη δήλωση:

```
USE work.package_name.all;
```

Σύγχρονες ή παράλληλες εντολές (1)

- Σύγχρονες εντολές είναι όλες οι εντολές ανάθεσης σήματος (signal assignment).

- Δεν έχει καμιά σημασία η σειρά αναγραφής τους, πράγμα που σημαίνει ότι τα ακόλουθα τμήματα κώδικα είναι ισοδύναμα:

  x <= a + b;   z <= x + c;

  z <= x + c;   x <= a + b;

- Προσοχή: Απαγορεύεται η απευθείας ανάθεση σημάτων με διαφορετικούς τύπους και θα πρέπει να χρησιμοποιηθεί κατάλληλη συνάρτηση μετατροπής για κάτι τέτοιο π.χ. εάν x είναι σήμα τύπου STD_LOGIC_VECTOR(7 DOWNTO 0) και y σήμα τύπου INTEGER RANGE 0 TO 255, τότε συνάρτηση μετατροπής x <= CONV_STD_LOGIC_VECTOR (y,8) (απαιτείται αρχικά η δήλωσηUSE ieee.std_logic_arith.all;)

Σύγχρονες ή παράλληλες εντολές (2)

- Ανάθεση σήματος υπό συνθήκη (conditional signal assignment):

  signal_name<= assignment_expr WHEN conditional_expr ELSE
  assignment_expr

- Ανάθεση σήματος με επιλογή (selected signal assignment):

  WITH selection_signal SELECT

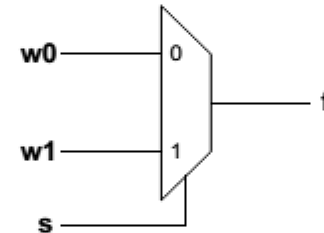
  signal_name<=assignment_expr WHEN selection_value

- Η διαφορά μεταξύ των δύο παραπάνω αναθέσεων είναι ότι με την πρώτη μπορεί κανείς να περιγράψει προτεραιότητα (καθώς κάθε μια συνθήκη έχει μεγαλύτερη προτεραιότητα από τις υποκείμενες σε αυτή) ενώ στη δεύτερη οι συνθήκες είναι (και θα πρέπει να είναι) αμοιβαία αποκλειόμενες.

Σύγχρονες ή παράλληλες εντολές (3)

- Παράδειγμα: πολυπλέκτης 2-σε-1:

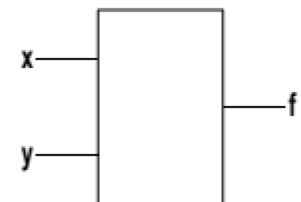
```
ARCHITECTURE dataflow OF mux2to1 IS
BEGIN
    f<=w0 WHEN s='0' ELSE w1;
END dataflow;
```



- Παράδειγμα: περιγραφή κυκλώματος με βάση τον πίνακα αληθείας του.

```
ARCHITECTURE dataflow OF circuit IS
BEGIN
        s<=x&y;

        WITH s select
                f<='0' WHEN "00",
                   '1' WHEN "01",
                   '1' WHEN "10",
                   '0' WHEN OTHERS;
END dataflow;
```

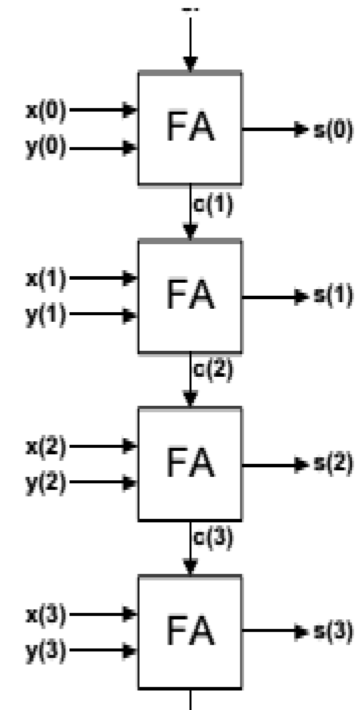
x	y	f
0	0	0
0	1	1
1	0	1
1	1	0



Επαναληπτική ανάθεση και δημιουργία στιγμιότυπων

- Η επαναληπτική ανάθεση σήματος ή δημιουργία στιγμιότυπου υποκυκλώματος μπορεί να γίνει με μία δομή βρόχου όπως η ακόλουθη:

```
generate_label:
FOR index_variable IN start_value TO end_value GENERATE
[COMPONENT instantiation statements;]
[concurrent assignment statements;]
END GENERATE;
```



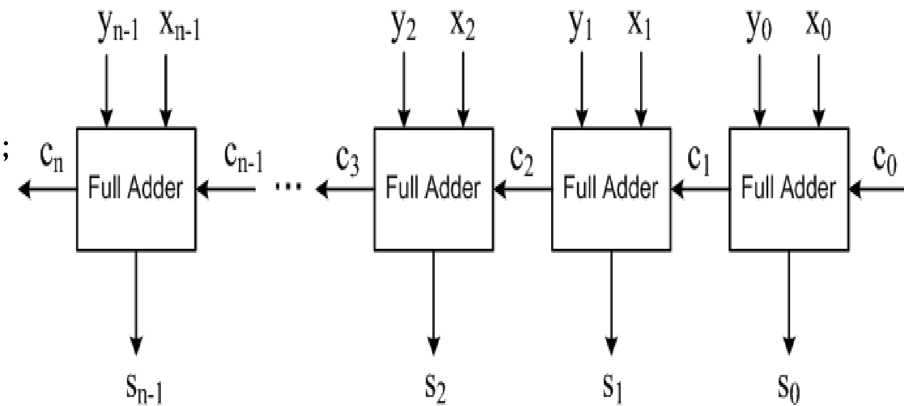
- Παράδειγμα: αθροιστής 4-bit

```
ARCHITECTURE structure OF adder 4 IS
        SIGNAL c: STD_LOGIC_VECTOR(0 TO 4);
        BEGIN
                c(0)<=ci;
                co<=c(4);
          gen: FOR i IN 0 TO 3 GENERATE
          stg: full_adder PORT MAP (c(i), x(i), y(i), s(i), c(i+1));
          END GENERATE;
END structure;
```

Παραμετρική περιγραφή (1)

- Αθροιστής n-bit:

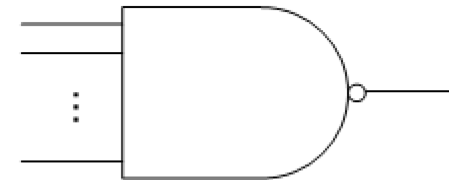
```
ENTITY addern IS
GENERIC (n: INTEGER:= 4);
PORT(ci: IN STD_LOGIC;
          x, y: IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
          s: OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0);
           co: OUT STD_LOGIC);
END addern;

ARCHITECTURE structural OF addern IS
SIGNAL c: STD_LOGIC_VECTOR(0 TO n);

          BEGIN
          c(0)<=ci;
          co<=<c(n);

          gen: FOR I IN 0 TO n-1 GENERATE
          stg: full_adder PORT MAP(c(i), x(i), y(i), s(i), c(i+1));
          END GENERATE;
END structural;
```



Παραμετρική περιγραφή (2)

- Παράδειγμα: NAND n-inputs

```
ENTITY nandn IS
	GENERIC(n: INTEGER:= 4);
	PORT(x: IN STD_LOGIC_VECTOR(1 TO n);
		f: OUT_STD_LOGIC);
END nandn;

ARCHITECTURE dataflow OF nandn IS
SIGNAL temp: STD_LOGIC_VECTOR(1 TO n);

	BEGIN
	temp<=(OTHERS=>'1');
	f<='0' WHEN x=temp ELSE '1';
END dataflow;
```



Η δομή της διαδικασίας

- Η δομή διαδικασίας (PROCESS) χρησιμοποιείται για να στεγάσει τις ακολουθιακές εντολές που εκτελούνται σειριακά κατά τη μετάφραση, αλλά από μόνη της αποτελεί μια σύγχρονη εντολή.

- PROCESS[signal_name, signal_name, ….]
  [VARIABLE declarations;]
  BEGIN
  [sequential statements;]
  [VARIABLE assignment statements;]
  END PROCESS;

- Λίστα ευαισθησίας (sensitivity list) της διαδικασίας είναι κάθε σήμα που χρησιμοποιεί η διαδικασία και του οποίου η αλλαγή τιμής μεταβάλλει την έξοδο του κυκλώματος (για συνδυαστικά κυκλώματα η λίστα ευαισθησίας περιλαμβάνει όλες τις εισόδους που χρησιμοποιούνται μέσα στη διαδικασία).

Παρατηρήσεις

- Οι εντολές μιας διαδικασίας εκτελούνται σειριακά από το μεταφραστή αλλά τα αποτελέσματά τους δεν είναι ορατά έξω από την διαδικασία έως ότου περατωθεί η εκτέλεση.

- Συνέπεια αυτού είναι ότι εάν σε ένα σήμα γίνουν περισσότερες της μιας αναθέσεις μέσα στο σώμα της διαδικασίας θα υπερισχύσει η τελευταία (στις σύγχρονες εντολές δεν επιτρέπονται περισσότερες αναθέσεις στο ίδιο σήμα).

- Τα δεδομένα τύπου VARIABLE (μεταβλητές) εμφανίζονται μόνο μέσα σε διαδικασίες (η εμβέλειά τους είναι τοπική) και δεν αντιστοιχούν σε σήματα αλλά έχουν βοηθητικό ρόλο στην αποθήκευση κάποιων τιμών (η ανάθεσή τους γίνεται με τον τελεστή := και υλοποιείται άμεσα κατά την εκτέλεση της διαδικασίας και όχι στο τέλος της).

Ακολουθιακές εντολές (1)

- Ακολουθιακή δομή συνθήκης:

```
IF conditional_expression THEN
[sequential statements;]
[ELSIF conditional_expression THEN]
[sequential statements;]
[ELSE]
[sequential statements;]
END IF;
```

- Ακολουθιακή δομή επιλογής:

```
CASE selection_signal IS
WHEN selection_values =>
[sequential statements;]
[WHEN selection_values =>]
[sequential statements;]
END CASE;
```

Ακολουθιακές εντολές (2)

- Όπως και με τις αντίστοιχες σύγχρονες εντολές, η διαφορά μεταξύ των δύο παραπάνω είναι ότι η πρώτη διαθέτει ενδογενή την έννοια της προτεραιότητας ενώ για τη δεύτερη οι συνθήκες πρέπει να είναι αμοιβαία αποκλειόμενες

- Οι παραπάνω εντολές, όμως, είναι πιο ισχυρές από τις αντίστοιχες σύγχρονες καθώς για κάθε συνθήκη μπορούν να εκτελεστούν οποιοδήποτε είδος και οποιοσδήποτε αριθμός ακολουθιακών εντολών και αναθέσεων, ενώ στις πρώτες αντιστοιχεί μόνομία εντολή ανάθεσης σε κάθε συνθήκη

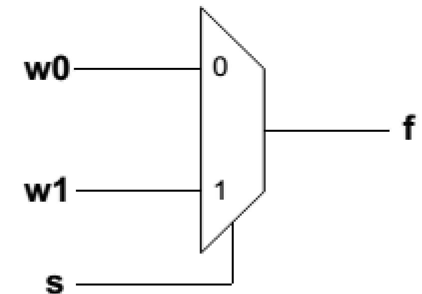
- Ακολουθιακή δομή βρόχου:

```
FOR index_variable IN start_value TO end_value LOOP
[sequential statements;]
END LOOP;
```

Περιγραφή της συμπεριφοράς στη VHDL (1)

- Βασίζεται στη χρήση ακολουθιακών εντολών και τη δομή PROCESS
- Περιγραφή συμπεριφοράς του πολυπλέκτη2-σε-1:

```
ARCHITECTURE behavior OF mux2to1 IS
	BEGIN
	PROCESS (w0, w1, s)
		BEGIN
		IF s = '0' THEN f <= w0;
		ELSE  f <= w1;
		END IF;
	END PROCESS;
END behavior;
```



Περιγραφή της συμπεριφοράς στη VHDL (2)

- Εναλλακτική περιγραφή συμπεριφοράς του πολυπλέκτη2-σε-1:

```
ARCHITECTURE behavior OF mux2to1 IS
BEGIN
	PROCESS (w0, w1, s)
	BEGIN
		f <= w0;
		IF s = '1' THEN f <= w1;
		END IF;
	END PROCESS;
END behavior;
```

```
IF s = '1' THEN f <= w1;
ENDIF;
f <= w0;
```

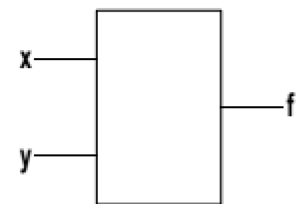
- Προσοχή: Η σειρά αναγραφής των ακολουθιακών εντολών επηρεάζει το τελικό αποτέλεσμα καθώς αυτές εκτελούνται σειριακά

Περιγραφή της συμπεριφοράς στη VHDL (3)

- Περιγραφή συμπεριφοράς κυκλώματοσμε βάση τον πίνακα αλήθειας:

```
ARCHITECTURE behavior OF tr_table IS
SIGNAL s: STD+LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
        PROCESS (x, y)
        BEGIN
                s <= x & y;
                CASE s IS
                        WHEN “00” => f <= ‘0’;
                        WHEN “01” => f <= ‘1’;
                        WHEN “10” => f <= ‘1’;
                        WHEN OTHERS => f <= ‘0’;
                END CASE;
        END PROCESS;
END behavior;
```

x	y	f
0	0	0
0	1	1
1	0	1
1	1	0



Προκαθορισμένες τιμές σημάτων στην περιγραφή συμπεριφοράς

- Για την σωστή λειτουργία των συνδυαστικών κυκλωμάτων τα οποία περιγράφονται από τις ακολουθιακές εντολές συνθήκης (τύπου IF) και επιλογής (τύπου CASE) θα πρέπει να υπάρχουν προκαθορισμένες τιμές σημάτων όταν αυτά δεν εμφανίζονται σε όλες τις δυνατές περιπτώσεις (καθώς και όταν δεν υπάρχει η εντολή ELSE στη δομή της συνθήκης).

- **ΠΑΡΑΔΕΙΓΜΑ:**

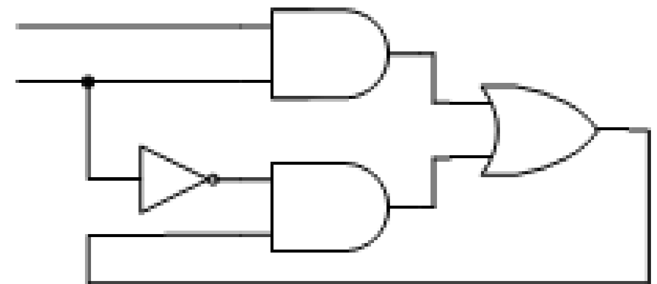
```
A<='0';
B<='0';

CASE y IS
        WHEN '0' => A<='1';
        WHEN OTHERS => B<='1';
END CASE;
```

Το φαινόμενο μνήμης

- Εάν το πρόγραμμα δεν καθορίζει την τιμή ενός σήματος, τότε θεωρείται ότι αυτό διατηρεί την τρέχουσα τιμή του (**φαινόμενο μνήμης**).

- ΠΑΡΑΔΕΙΓΜΑ: Στην εναλλακτική περιγραφή που έχουμε δει, ενός πολυπλέκτη 2-σε-1 σε VHDL έχουμε την ακόλουθη αρχιτεκτονική:

```
ARCHITECTURE behavioral OF mux2to1 IS
BEGIN
        PROCESS(w0, w1, s)
        BEGIN
                f<=w0;
                IF s='1' THEN f<=w1;
                END IF;
        END PROCESS;
END behavioral;
```



- Η εμφάνιση μνημονικής συμπεριφοράς (ουσιαστικά ενός βρόχου ανάδρασης) είναι ανεπιθύμητη για τα συνδυαστικά κυκλώματα, αλλά αποτελεί τη βάση για την ανάπτυξη των ακολουθιακών κυκλωμάτων.

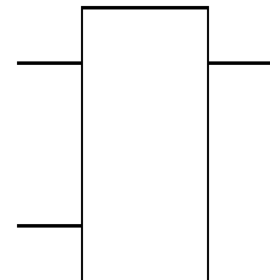
Περιγραφή βασικών ακολουθιακών στοιχείων

- Ακολουθιακό κύκλωμα είναι αυτό που η επόμενη κατάστασή του (η έξοδός του) εξαρτάται από τις εισόδους του και την παρούσα κατάστασή του.

- Η περιγραφή βασικών ακολουθιακών στοιχείων επιτυγχάνεται με τη χρήση αποκλειστικά ακολουθιακών εντολών και τη δομή της διαδικασίας.

- Μανδαλωτής τύπου D (D-Latch):

```
ENTITY latch IS
PORT(D, Clock: IN STD_LOGIC;
        Q: OUT STD_LOGIC);
END latch;

ARCHITECTURE behavioral OF latch IS
BEGIN
        PROCESS(D, Clock)
        BEGIN
                IF Clock='1' THEN Q<=D;
                END IF;
        END PROCESS;
END behavioral;
```



Αλλαγή στην ακμή του ρολογιού (1)

- Flip-Flop τύπου D (DFF)

```
ARCHITECTURE behavioral OF dff IS
BEGIN
        PROCESS(Clock)
        BEGIN
                IF Clock'EVENT AND Clock='1' THEN Q<=D;
                END IF;
        END PROCESS;
END behavioral;
```

- Στα ακμοπυροδότητα στοιχεία μόνο το σήμα ρολογιού εισέρχεται ως παράμετρος στη λίστα ευαισθησίας (μαζί με τυχόν ασύγχρονα σήματα) καθώς η έξοδος του κυκλώματος μεταβάλλεται αποκλειστικά και μόνο στην ακμή του ρολογιού.

- Clock'EVENT: Μια ιδιότητα του σήματος ρολογιού η οποία αναφέρεται στην αλλαγή κατάστασης του.

Αλλαγή στην ακμή του ρολογιού (2)

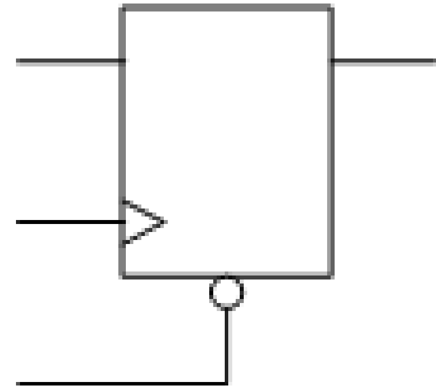
- Εναλλακτική περιγραφή DFF:

```
ARCHITECTURE behavioral OF dff IS
BEGIN
	PROCESS
		BEGIN
		WAIT UNTIL Clock'EVENT AND Clock='1';
		Q<=D;
	END PROCESS;
END behavioral;
```

- Η δομή WATI UNTIL εμφανίζεται πάντοτε ως πρώτη εντολή και καλύπτει ολόκληρη την διαδικασία (αντίθετα με τη δομή συνθήκης τύπου IF), ενώ επιπλέον η χρήση της δεν περιλαμβάνει ποτέ λίστα ευαισθησίας καθώς υπονοεί ότι συνίσταται αποκλειστικά και μόνο από το σήμα ρολογιού (δηλαδή ότι δεν υπάρχουν ασύγχρονα σήματα στη συγκεκριμένη διαδικασία).

Ασύγχρονες Είσοδοι

- DFF με ασύγχρονο μηδενισμό:



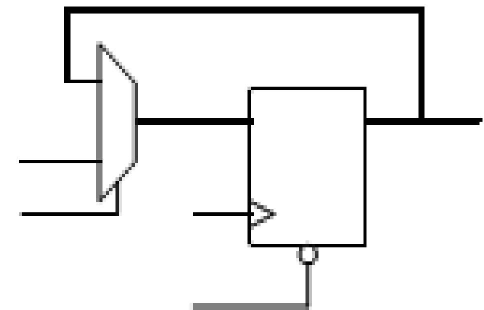
```
ARCHITECTURE behavioral OF dff IS
BEGIN
   PROCESS(Clock, Resetn)
   BEGIN
        IF Resetn=‘0’ THEN Q<=‘0’;
        ELSIF Clock’EVENT AND Clock=‘1’ THEN Q<=D;
       END IF;
   END PROCESS;
END behavioral;
```

Περιγραφή Καταχωρητών (1)

- Η περιγραφή των απλών καταχωρητών είναι ίδια με αυτή των DFFs με τη διαφορά ότι τα σήματα εισόδου (D) και εξόδου (Q) ορίζονται ως διανύσματα τύπου STD_LOGIC_VECTOR με το επιθυμητό μήκος.

- ΠΑΡΑΔΕΙΓΜΑ: Παραμετρικός καταχωρητής **n-bit** με είσοδο ενεργοποίησης (**enable**) και ασύγχρονο μηδενισμό.

```
…
GENERIC (n : INTEGER := 8);
PORT (Resetn, E, Clock : IN STD_LOGIC;
          D  : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
          Q  : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0));
…
        IF Resetn = '0' THEN Q <= (OTHERS => '0');
        ELSIF Clock'EVENT AND Clock = '1' THEN
                IF E = '1' THEN Q <= D;
                END IF;
        END IF;
…
```



Περιγραφή Καταχωρητών (2)

- Καταχωρητής ολίσθησης (shift-register) 4-bi με είσοδο παράλληλης φόρτωσης: φόρτωσης:

```
…
PORT (R  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      w, L, Clock : IN STD_LOGIC;
      Q  : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0));
…
      IF Clock'EVENT AND Clock = '1' THEN
            IF L = '1' THEN Q <= R;
            ELSE
                  Q(0) <= Q(1);        Q(3) <= w;
                  Q(1) <= Q(2);        Q(2) <= Q(3);
                  Q(2) <= Q(3);        Q(1) <= Q(2);
                  Q(3) <= w;           Q(0) <= Q(1);
            END IF;
      END IF;
…
```

Περιγραφή Μετρητών (1)

- Αύξοντας μετρητής **4-bit** με εισόδους ενεργοποίησης και ασύγχρονου μηδενισμού:

```
USE ieee.std_logic_unsigned.all;
….
PORT(Resetn, E: IN STD_LOGIC;
    Clock: IN STD_LOGIC;
    Q: BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0));
….
   IF Resetn='0' THEN Q<="0000";
   ELSIF Clock'EVENT AND Clock='1' THEN
         IF E='1' THEN Q<=Q+1;
         END IF;
   END IF;
```

Περιγραφή Μετρητών (2)

- Εναλλακτική περιγραφή αύξοντα μετρητή 4-bit (πολύ συνηθισμένη στη βιβλιογραφία και τη βιομηχανία):

```
….
PORT(Resetn, E: IN STD_LOGIC;
        Clock: IN STD_LOGIC;
        Q: BUFFER INTEGER RANGE 0 TO 15);
….
IF Resetn=‘0’ THEN Q<=‘0’;
ELSIF Clock’EVENT AND Clock=‘1’ THEN
    IF E=‘1’ THEN Q<=Q+1;
    END IF;
END IF;
```

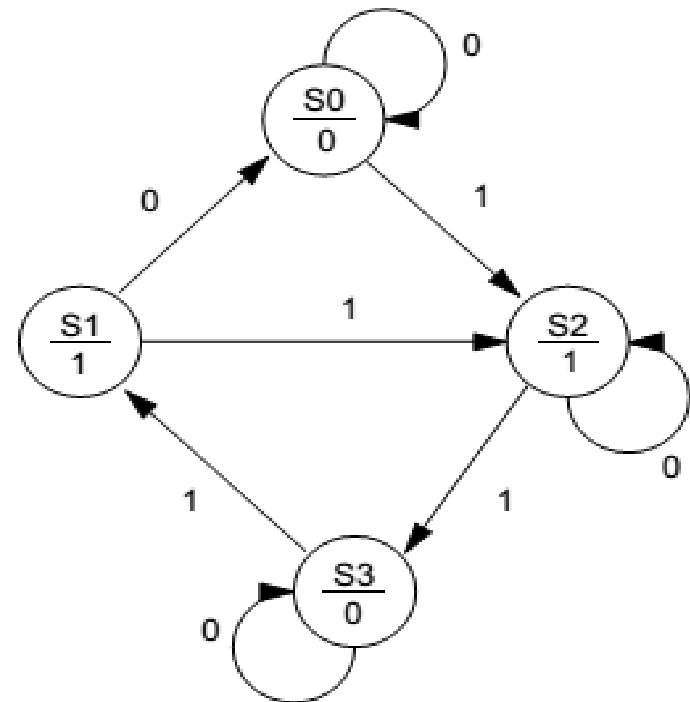
Περιγραφή Μετρητών (3)

- Παραμετρικός φθίνοντας μετρητής με εισόδους ενεργοποίησης και σύγχρονης (παράλληλης) φόρτωσης:

```
….
GENERIC(counter_size: INTEGER:= 8);
PORT(L, E: IN STD_LOGIC;
     Clock: IN STD_LOGIC;
     Q: BUFFER INTEGER RANGE 0 TO counter_size-1);
….
IF Clock'EVENT AND Clock='1' THEN
   IF E='1' THEN
        IF L='1' THEN Q<=counter_size-1;
        ELSE Q<=Q-1;
        END IF;
   END IF;
END IF;
….
```

Μηχανές Πεπερασμένων Καταστάσεων (1)

- **Moore Machine:** Οι έξοδοι μιας μηχανής Moore εξαρτώνται αποκλειστικά από την παρούσα κατάσταση.
- Οι μεταβάσεις των εξόδων είναι σύγχρονες με το ρολόι του συστήματος.



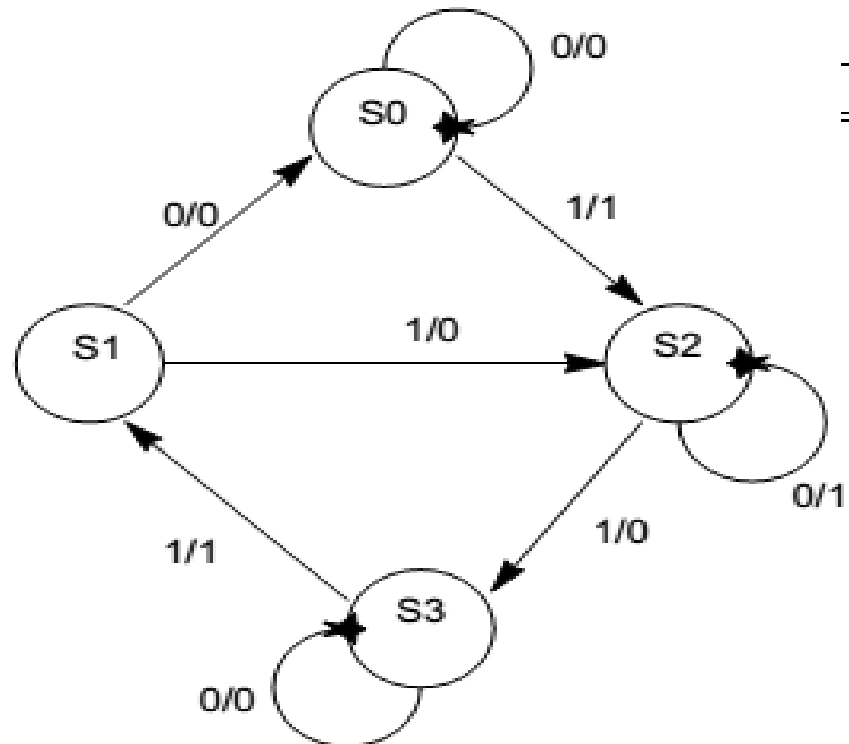
Μηχανές Πεπερασμένων Καταστάσεων (2)

```
ENTITY moore IS
PORT(x, clock: IN STD_LOGIC; Z: OUT STD_LOGIC);
END moore;

ARCHITECTURE behavioral OF moore IS
TYPE state_type IS (S0, S1, S2, S3);
SIGNAL current_state, next_state: state_type;
BEGIN
    --process to hold combinational logic
    PROCESS(current_state, x)
    BEGIN
      CASE current_state IS
        WHEN S0=>Z<='0';
          IF x='0' THEN next_state<=S0;
          ELSE next_state<=S2;
          END IF;
        WHEN S1=>Z<='1';
          IF x='0' THEN next_state<=S0;
          ELSE next_state<=S2;
          END IF;
```

```
   WHEN S2=>Z<='1';
      IF x='0' THEN next_state<=S2;
      ELSE next_state<=S3;
      END IF;
   WHEN S3=>Z<='0';
      IF x='0' THEN next_state<=S3;
      ELSE next_state<=S1;
      END IF;
  END CASE;
END PROCESS;

--process to hold synchronous elements (flip-flops)
PROCESS
BEGIN
   WAIT UNTIL clock'EVENT AND clock='1';
   current_state<=next_state;
END PROCESS;

END behavioral;
```

Μηχανές Πεπερασμένων Καταστάσεων (3)

- Mealy Machine: Οι έξοδοι μιας μηχανής Mealy εξαρτώνται τόσο από την παρούσα κατάσταση όσο και από την τιμή της εισόδου.
- Οι μεταβάσεις των εξόδων είναι σύγχρονες με το ρολόι του συστήματος (οι αλλαγές πραγματοποιούνται με το που αλλάζει η τιμή της εισόδου).

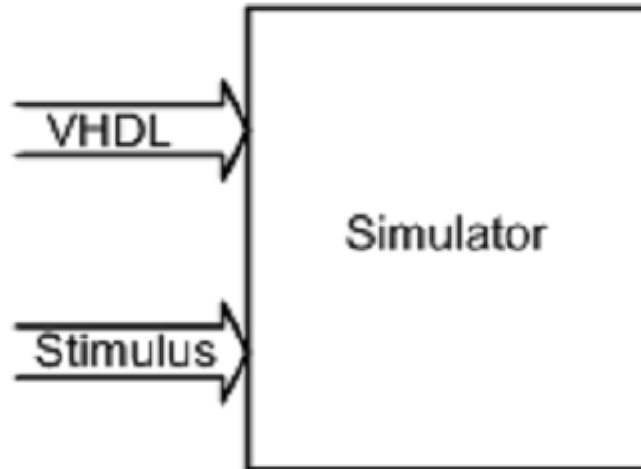


Μηχανές Πεπερασμένων Καταστάσεων (4)

```
ENTITY mealy IS
PORT(x, clock: IN STD_LOGIC; z: OUT STD_LOGIC);
END mealy;
ARCHITECTURE behavioral OF mealy IS
TYPE state_type IS (S0, S1, S2, S3);
SIGNAL current_state, next_state: state_type;
BEGIN
  PROCESS(current_state, x)
  BEGIN
  CASE current_state IS
    WHEN S0=>
      IF x='0' THEN
            Z<='0';
            next_state<=S0;
      ELSE
            z<='1';
            next_state<=S2;
    WHEN S1=>
      IF x='0' THEN
            z<='0';
            next_state<=S0;
       ELSE
            z<='0';
            next_state<=S2;
      END IF;
    WHEN S2=>
      IF x='0' THEN
            Z<='1';
            next_state<=S2;
      ELSE
            z<='0';
            next_state<=S3;
       END IF;
    WHEN S3=>
       IF x='0' THEN
            z<='0';
            next_state<=S3;
       ELSE
            z<='1';
            next_state<=S1;
       END IF;
   END CASE;
  END PROCESS;

  --process to hold asynchronous elements (flip-flops)
  PROCESS
  BEGIN
    WAIT UNTIL clock'EVENT AND clock='1';
    current_state<=next_state;
  END PROCESS;
END behavioral;
```

Testbench (1)

- Προσομοίωση σε **RTL** επίπεδο (**RTL Simulation**)
  - **Testbench**
    - Επιβεβαίωση ορθής λειτουργίας (**Verify the functionality of a design**)



Testbench (2)

```
ENTITY testbench IS
END testbench;
ARCHITECTURE behavior OF testbench IS
constant PERIOD:time:= 6.66 ns;
            COMPONENT c17
            PORT(       S1 : IN std_logic;
                        ...S23 : OUT std_logic );
            END COMPONENT;
            SIGNAL vector_cnt: integer := 1;
            SIGNAL S1   : std_logic;
            ...SIGNAL S23 :std_logic;
type test_record is record
        S1: std_logic; S2: std_logic;
        S3: std_logic; S6: std_logic; S7: std_logic;
end record;

type test_array is array(positive range <>) of test_record;

constant test_vectors : test_array := (
            ('1','1','0','0','1'),
            ('1','1','1','0','1'),
            ('1','0','0','1','0'),
            ('1','1','1','0','1'));
```

Declare a record type

Create an Array

Test vectors

```
BEGIN
   dut: c17 PORT MAP(
            S1  => S1,  S2  => S2 ,
            S3  => S3,  S6  => S6 ,
            S7  => S7,  S22 => S22,
            S23 => S23);
   testrun: process
    variable vector : test_record;
    begin
      for index in test_vectors'range loop
            vector_cnt <= index;
            vector := test_vectors(index);
             S1  <= vector.S1  ;
             S2  <= vector.S2  ;
             S3  <= vector.S3  ;
             S6  <= vector.S6  ;
             S7  <= vector.S7  ;
            wait for PERIOD;
      end loop;
      wait;
    end process;
END behavior;
```

Apply Stimulus