ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ

ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

Η γλώσσα περιγραφής κυκλωμάτων VHDL

Καβουσιανός Χρ.– Τσιατούχας Γ.
Λέκτορες Πανεπιστημίου
Ιωαννίνων

(Peter Ashenden, The Students Guide to VHDL)

Γενικά

VHDL = VHSIC (Very High Speed Integrated Circuits)
Hardware Description Language

- 1980 Υπουργείο Άμυνας ΗΠΑ-IEEE – Verilog ABEL
- Το 1987 έγινε πρότυπη γλώσσα από το IEEE ως *IEEE Standard 1076*.
- Αναθεωρήθηκε το 1993 και το 2001 και η πρόσφατη έκδοση κυκλοφόρησε ως VHDL-2001.
- Η μάθηση της VHDL απαιτεί γνώση της θεωρίας *Ψηφιακής Σχεδίασης*.

Γενικά

Τα σύγχρονα ψηφιακά συστήματα είναι αρκετά περίπλοκα. Απαιτείται αντιμετώπιση της πολυπλοκότητας για σχεδίαση με σιγουριά τήρησης των προδιαγραφών.

Ένας τρόπος είναι ο συστηματικός σχεδιασμός:

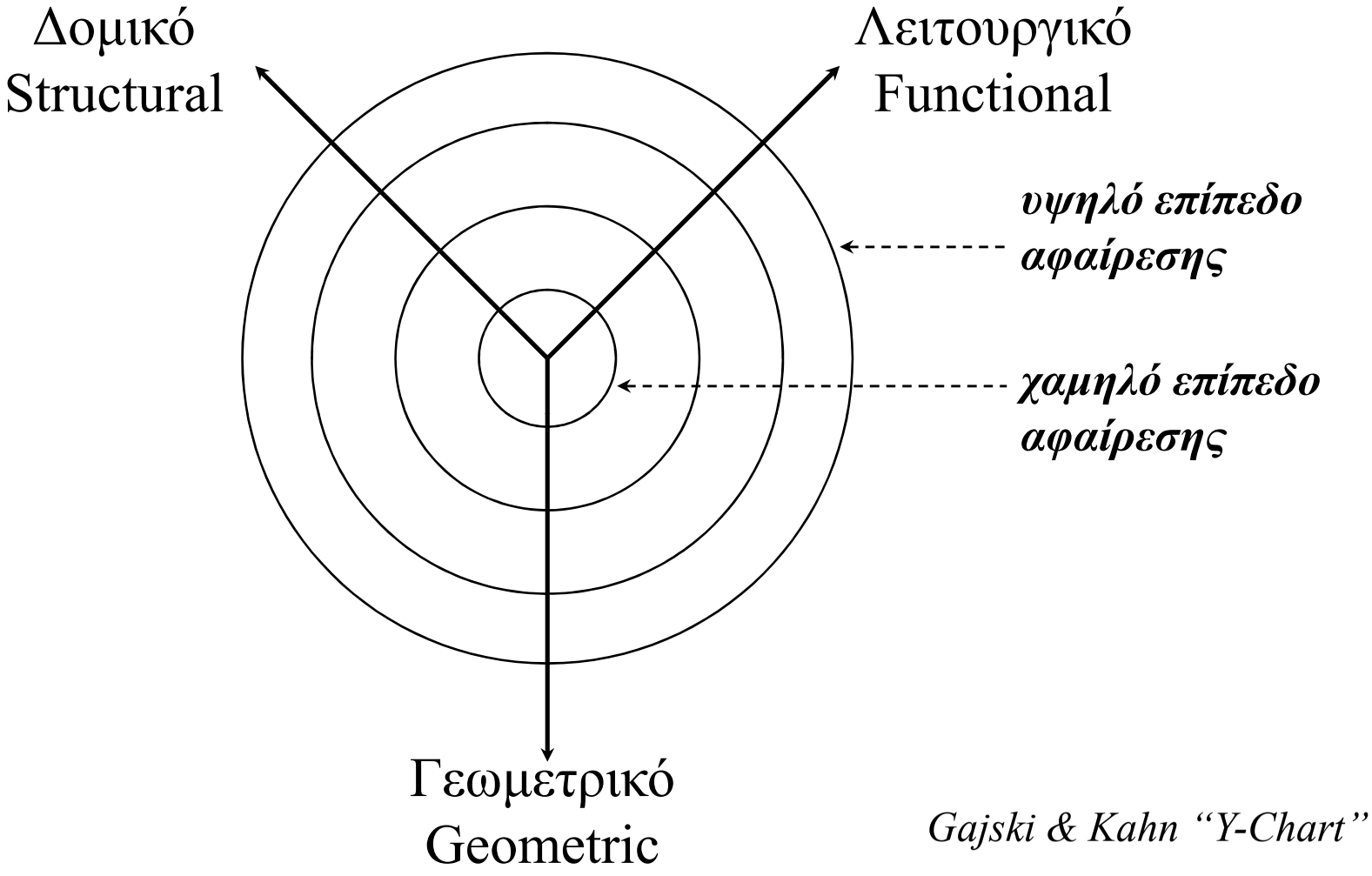
- χρήση ιεραρχικής δομής (από αφηρημένα μοντέλα σε στοιχεία βιβλιοθήκης)
- Κάθε υποσύστημα μπορεί να αντιμετωπιστεί ως ανεξάρτητη αφηρημένη οντότητα (χωρίς λεπτομέρειες υλοποίησης).

Μοντέλο: Τρόπος αναπαράστασης πληροφοριών και επίπεδο αφαίρεσης.

Μοντελοποίηση συστημάτων

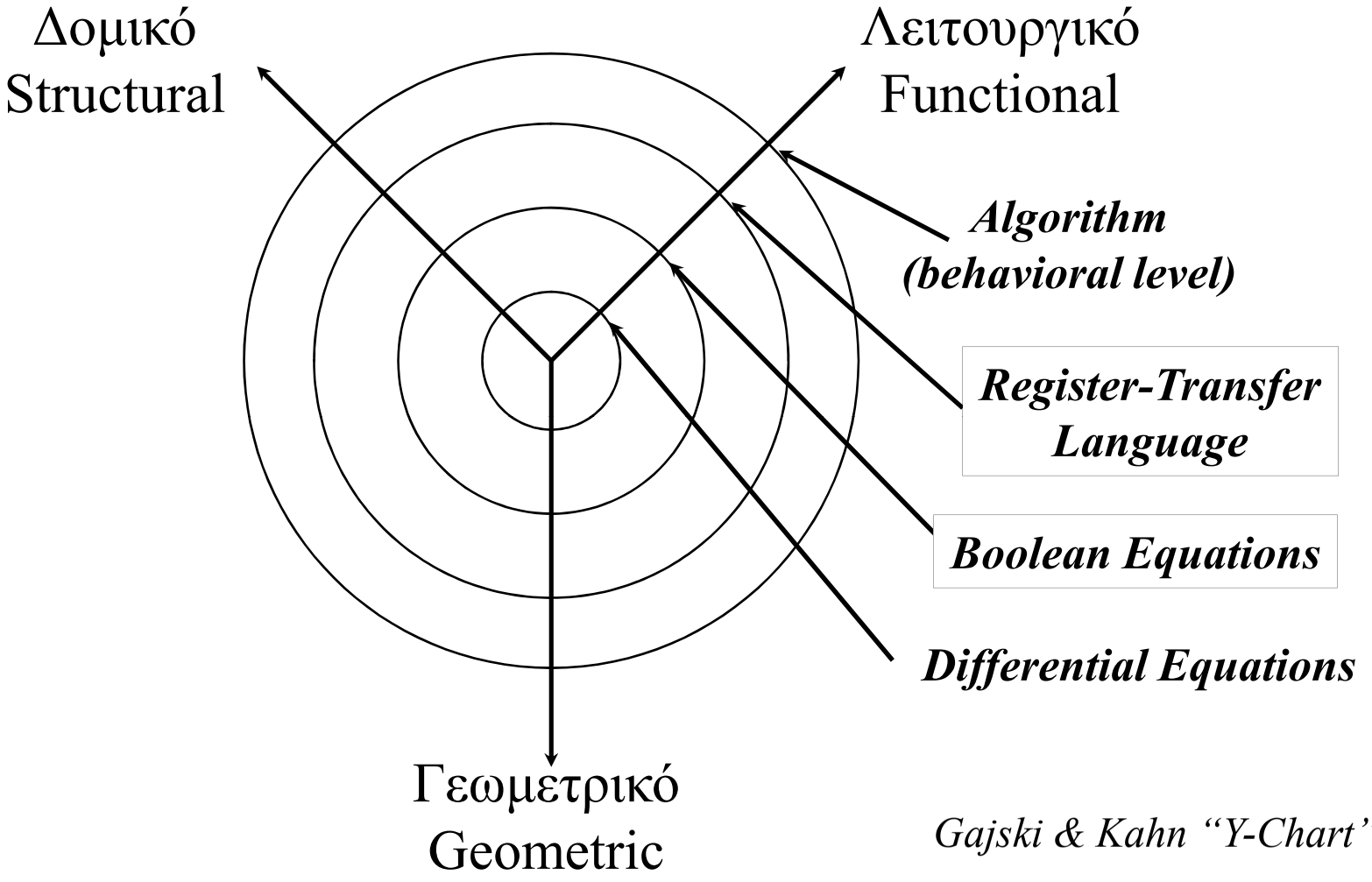
- Η VHDL χρησιμοποιείται για την ανάπτυξη μοντέλων ενός συστήματος
- Ανάγκες χρήσης μοντέλων:
  - *καταγραφή προδιαγραφών*
  - *τεκμηρίωση*
  - *επαλήθευση με τη χρήση προσομοίωσης*
  - *τυπική επαλήθευση (formal verification)*
  - *σύνθεση*
- Στόχος
  - *αξιόπιστη διαδικασία σχεδίασης, με ταυτόχρονη ελαχιστοποίηση του κόστους και του απαιτούμενου χρόνου*
  - *αποφυγή σχεδιαστικών λαθών*

Περιοχές & Επίπεδα Μοντελοποίησης I



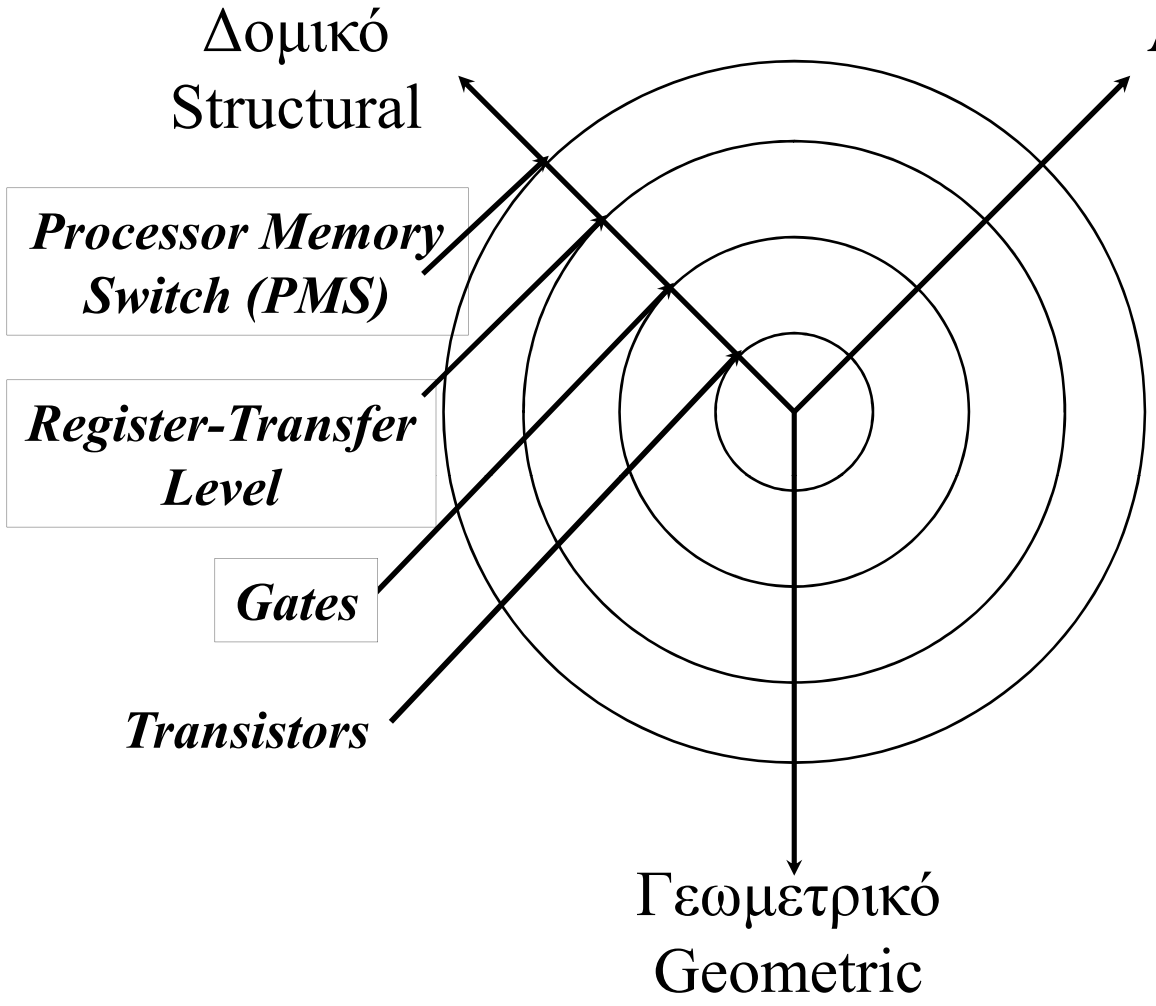
Gajski & Kahn "Y-Chart"

Περιοχές & Επίπεδα Μοντελοποίησης ΙΙ



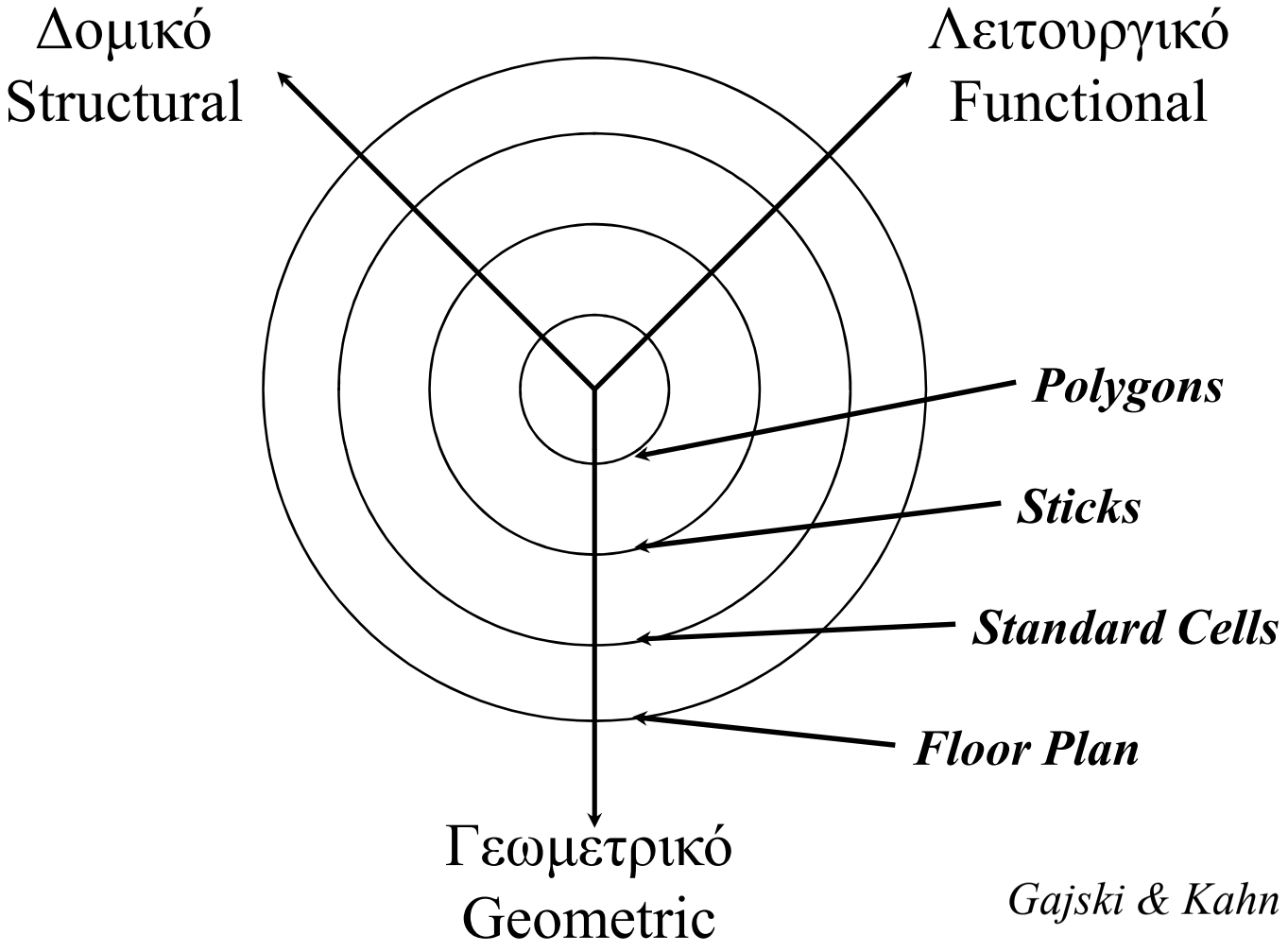
Gajski & Kahn "Y-Chart"

Περιοχές & Επίπεδα Μοντελοποίησης ΙΙΙ



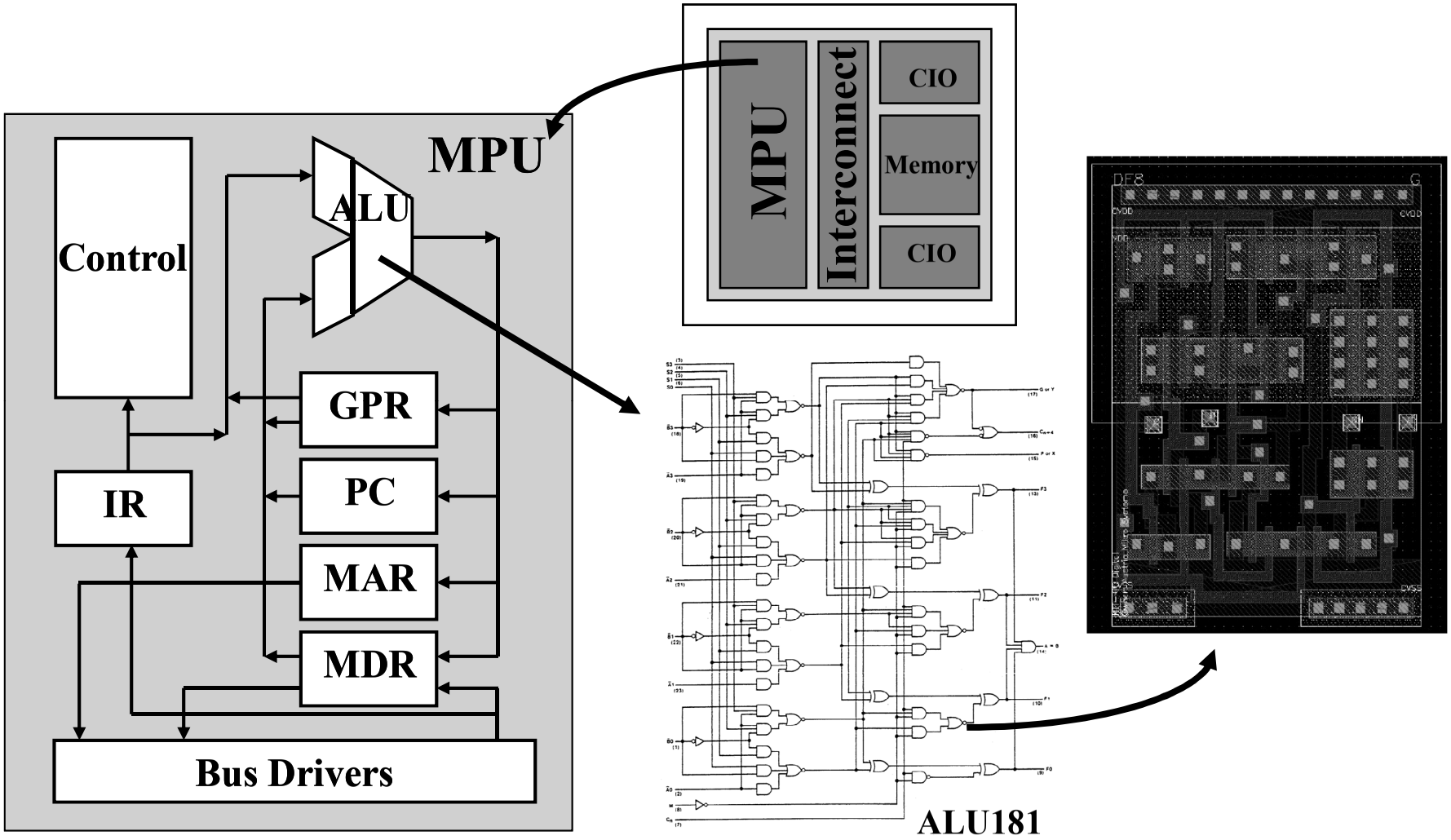
Gajski & Kahn "Y-Chart"

Περιοχές & Επίπεδα Μοντελοποίησης IV



Gajski & Kahn "Y-Chart"

Παράδειγμα


MPU
Control
ALU
GPR
PC
IR
MAR
MDR
Bus Drivers
MPU
Interconnect
CIO
Memory
CIO
ALU181

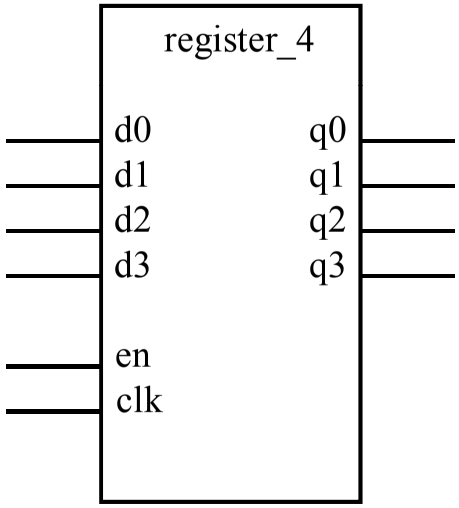
Βασικές Αρχές

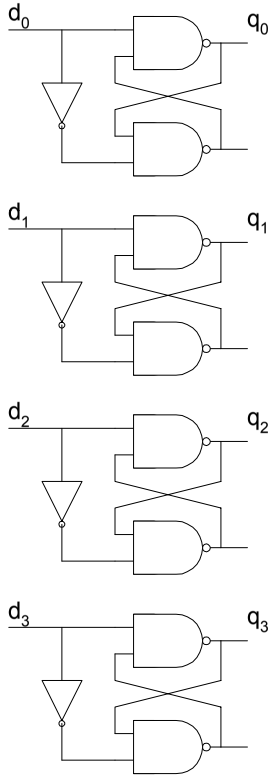
- *Μοντέλα Διασυνδέσεων (Interface Models)*
- *Μοντέλα Συμπεριφοράς (Behaviour Models)*
- *Μοντέλα Δομής (Structure Models)*
- *Μοντέλα Ελέγχου-Επαλήθευσης (Test Bench Models)*
- *Προσομοίωση (Simulation)*
- *Σύνθεση (Synthesis)*

Η δομή μιας VHDL σχεδίασης μοιάζει με τη δομή μίας σύγχρονης σχεδίασης λογισμικού: η VHDL περιγράφει από κοινού ένα τρόπο διασύνδεσης/επικοινωνίας του σχεδιασμού με το εξωτερικό περιβάλλον καθώς και μία εσωτερική υλοποίηση.

Βασικές Αρχές

Η VHDL περιγραφή μιας σχεδίασης αποτελείται από *οντότητες (entities)* και *αρχιτεκτονικές (architectures)*. Σε κάθε οντότητα αντιστοιχεί τουλάχιστον μία αρχιτεκτονική





Δήλωση Οντότητας

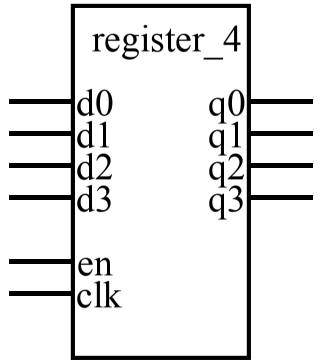
- *Δήλωση Οντότητας (Entity)*
  - ονοματίζει μια κυκλωματική οντότητα (module) και περιγράφει τον τρόπο διασύνδεσής της με το περιβάλλον παραθέτοντας τις θύρες εισόδου/εξόδου σημάτων

όνομα οντότητας ***ονόματα θυρών*** ***συμπεριφορά θύρας (φορά)***

```
entity register_4 is
   port (d0, d1, d2, d3, en, clk : in bit;
         q0, q1, q2, q3          : out bit);
end entity register_4;
```

χρήση μπλε χρώματος για τις δεσμευμένες λέξεις

τύπος θύρας



{Συμπεριφορά θύρας: in, out, inout, buffer }

Αρχιτεκτονική

Αρχιτεκτονικό Σώμα (Architecture Body)

- περιγράφει μία πιθανή υλοποίηση μιας οντότητας
- ενδεχομένως να υπάρχουν περισσότερες της μίας περιγραφές

Ένα αρχιτεκτονικό σώμα μπορεί να αποδίδει μία *αρχιτεκτονική συμπεριφοράς (behavioural architecture)* ή μία *αρχιτεκτονική δομής (structural architecture)*

- *Αρχιτεκτονική Συμπεριφοράς (Behavioural Architecture):* περιγράφει τον αλγόριθμο που επιτελεί η οντότητα και περιλαμβάνει καταχωρήσεις διαδικασιών.

Αρχιτεκτονική

Διαδικασίες (Process Statements): *συλλογή ενεργειών που εκτελούνται ακολουθιακά.*

- Εκτίμηση εκφράσεων
- Ανάθεση τιμών σε μεταβλητές
- Εκτέλεση υπό συνθήκη
- Επαναλαμβανόμενη εκτέλεση
- Κλήσεις υποπρογραμμάτων
- Ανάθεση σημάτων (διαφοροποίηση: προγραμματισμός ανάθεσης σε μέλλοντα χρόνο).
- Αναμονή γεγονότων (wait statements).

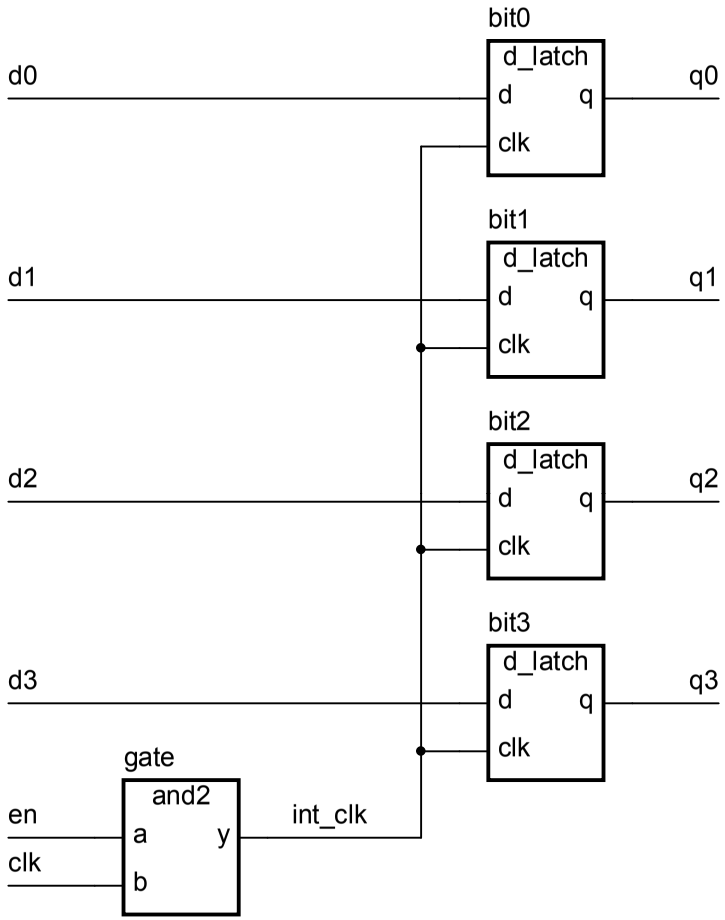
Παράδειγμα Αρχιτεκτονικής

```
architecture behav of register_4 is
begin
   storage : process is
      variable stored_d0,stored_d1, stored_d2, stored_d3 : bit;
   begin
      if en = '1' and clk = '1' then
         stored_d0 := d0;
         stored_d1 := d1;
         stored_d2 := d2;
         stored_d3 := d3;
      end if;
      q0 <= stored_d0 after 5 ns;
      q1 <= stored_d1 after 5 ns;
      q2 <= stored_d2 after 5 ns;
      q3 <= stored_d3 after 5 ns;
      wait on d0, d1, d2, d3, en, clk;
   end process storage;
end architecture behav;
```

Μοντελοποίηση Δομής

- *Αρχιτεκτονική Δομής (Structural Architecture*)
  - υλοποιεί μία οντότητα ως μία σύνθεση υποσυστημάτων ή κυκλωματικών στοιχείων
  - περιλαμβάνει
    - *δηλώσεις σημάτων (signal declarations)*, για εσωτερικές διασυνδέσεις
      - οι θύρες μιας οντότητας (entity) πραγματεύονται επίσης ως σήματα
    - *στιγμιότυπα κυκλωματικών στοιχείων (component instances)*
      - στιγμιότυπα ζευγών entity/architecture που έχουν δηλωθεί νωρίτερα
    - *χαρτογράφηση θυρών (port maps)* των στιγμιότυπων των κυκλωματικών στοιχείων
      - σύνδεση σημάτων στις θύρες των κυκλωματικών στοιχείων
    - *δηλώσεις αναμονής (wait statements)*

Αρχιτεκτονική Δομής I



Δομική σύνθεση του καταχωρητή 4-bit με τη χρήση D-Latch και AND κυκλωματικών στοιχείων

Αρχιτεκτονική Δομής II

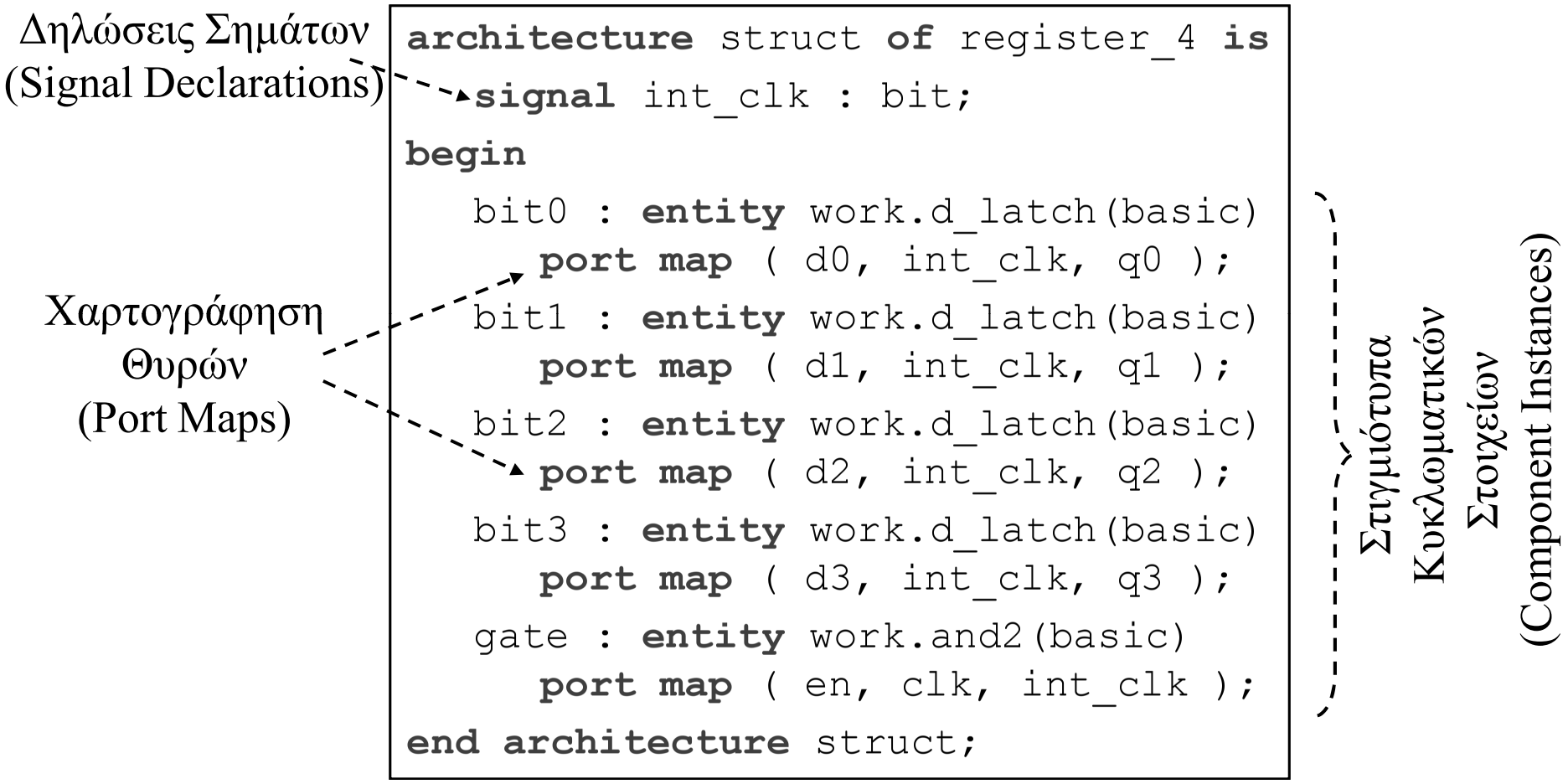
- Αρχικά η δήλωση των entities/architectures των κυκλωματικών στοιχείων D-Latch και AND.

```
entity d_latch is
   port ( d, clk   : in bit;
                q   : out bit );
end entity d_latch;

architecture basic of d_latch is
begin
   latch_behavior : process is
   begin
      if clk = '1' then
         q <= d after 2 ns;
      end if;
      wait on clk, d;
   end process latch_behavior;
end architecture basic;
```

```
entity and2 is
   port ( a, b  : in bit;
             y   : out bit );
end entity and2;

architecture basic of and2 is
begin
   and2_behavior : process is
   begin
      y <= a and b after 2 ns;
      wait on a, b;
   end process and2_behavior;
end architecture basic;
```

Αρχιτεκτονική Δομής III

- Χρήση των κυκλωματικών στοιχείων στην υλοποίηση του καταχωρητή 4-bit .




```
architecture struct of register_4 is
   signal int_clk : bit;
begin
   bit0 : entity work.d_latch(basic)
      port map ( d0, int_clk, q0 );
   bit1 : entity work.d_latch(basic)
      port map ( d1, int_clk, q1 );
   bit2 : entity work.d_latch(basic)
      port map ( d2, int_clk, q2 );
   bit3 : entity work.d_latch(basic)
      port map ( d3, int_clk, q3 );
   gate : entity work.and2(basic)
      port map ( en, clk, int_clk );
end architecture struct;
```

Μεικτή Περιγραφή Συμπεριφοράς - Δομής

Μία αρχιτεκτονική μπορεί να ενσωματώνει ταυτόχρονα τμήματα περιγραφής συμπεριφοράς και δομής:

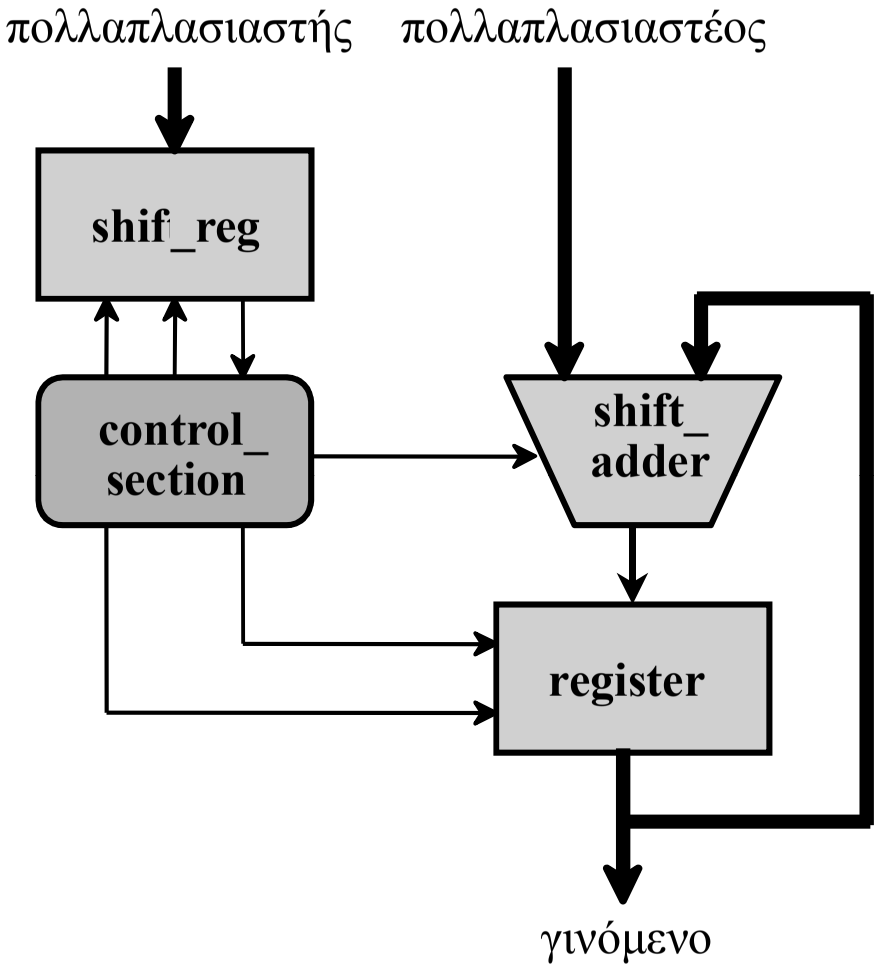
- συνύπαρξη διαδικασιών (process statements) και δομών (component instances)
- οι διαδικασίες και οι δομές επικοινωνούν με σήματα στα οποία μπορούν να διαβάζουν και να αποδίδουν τιμές.
- Οι διαδικασίες και οι δομές ονομάζονται *concurrent statements* καθώς εκτελούνται ταυτόχρονα.

Δίνει μεγάλη ευελιξία στην σχεδίαση και επιτρέπει χρήση προσχεδιασμένων τμημάτων.

Παράδειγμα Μεικτής Περιγραφής

Παράδειγμα: RTL μοντέλο πολλαπλασιαστή

- το data path περιγράφεται δομικά
- το τμήμα ελέγχου (control) περιγράφεται σε μορφή συμπεριφοράς



Παράδειγμα Μεικτής Περιγραφής ΙΙ

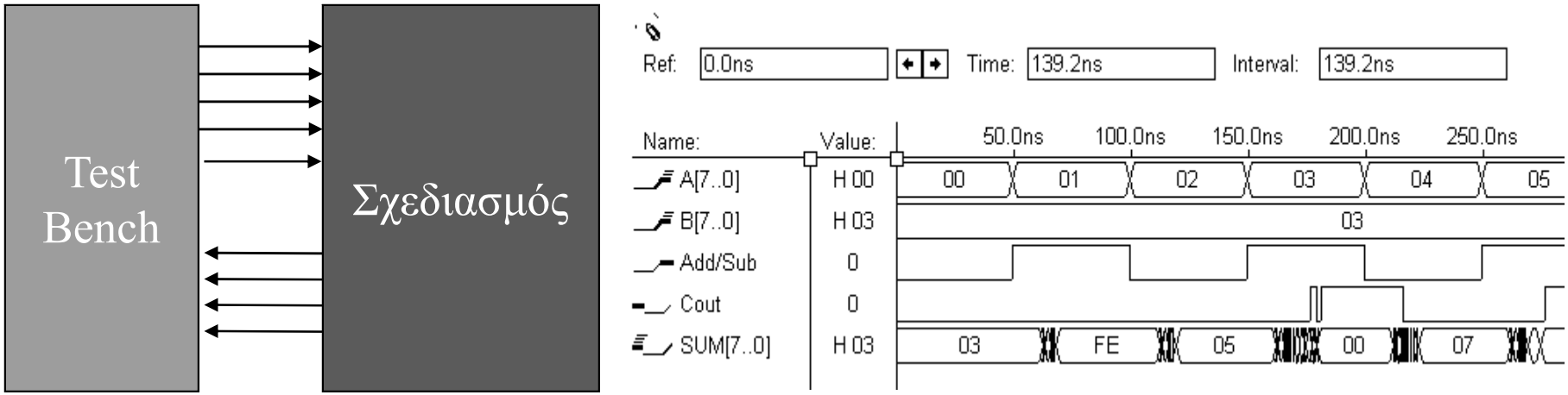
```
entity multiplier is
   port ( clk, reset : in bit;
          multiplicand, multiplier : in integer;
          product : out integer );
end entity multiplier;

architecture mixed of mulitplier is
   signal partial_product, full_product : integer;
   signal arith_control, result_en, mult_bit, mult_load : bit;
begin
   arith_unit : entity work.shift_adder(behavior)
      port map ( addend => multiplicand, augend => full_product,
                 sum => partial_product,
                 add_control => arith_control );
   result : entity work.register(behavior)
      port map ( d => partial_product,  q => full_product,
                 en => result_en,  reset => reset );
   ...
```

Παράδειγμα Μεικτής Περιγραφής III

```
   …
   multiplier_sr : entity work.shift_reg(behavior)
      port map ( d => multiplier,  q => mult_bit,
                 load => mult_load,  clk => clk );
   product <= full_product;

   control_section : process is
      -- variable declarations for control_section
      -- …
   begin
      -- sequential statements to assign values to control

      -- signals
      -- …
      wait on clk, reset;
   end process control_section;
end architecture mixed;
```

Μοντέλα Ελέγχου-Επαλήθευσης

- Ο έλεγχος μιας σχεδίασης γίνεται με μεθόδους προσομοίωσης
- Απαιτείται ο σχεδιασμός καθώς και μία μέθοδος τροφοδότησης ακολουθιών τιμών στις εισόδους του σχεδιασμού
- Η καταγραφή των σημάτων εξόδων καθώς και η σύγκρισή τους με αναμενόμενες τιμές δίνει το αποτέλεσμα του ελέγχου.

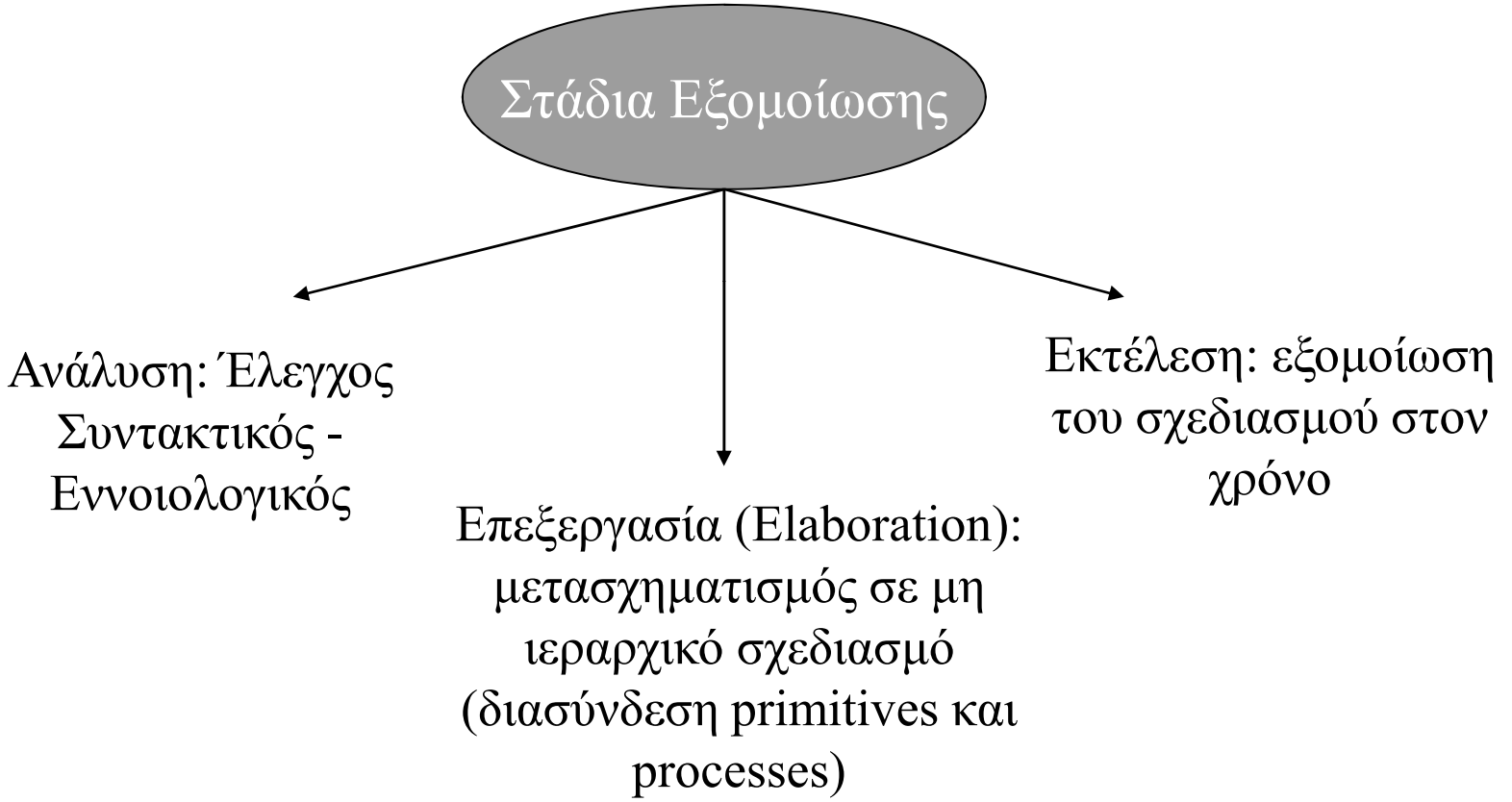


Μοντέλα Ελέγχου-Επαλήθευσης ΙΙ

```
entity test_bench is
end entity test_bench;

architecture test_register_4 of test_bench is
    signal d0, d1, d2, d3, en, clk, q0, q1, q2, q3 : bit;
begin
    dut : entity work.register_4(behav)
        port map ( d0, d1, d2, d3, en, clk, q0, q1, q2, q3 );
    stimulus : process is
    begin
        d0 <= '1';  d1 <= '1';  d2 <= '1';  d3 <= '1';  wait for 20 ns;
        en <= '0';  clk <= '0';  wait for 20 ns;
        en <= '1';  wait for 20 ns;
        clk <= '1';  wait for 20 ns;
        d0 <= '0';  d1 <= '0';  d2 <= '0';  d3 <= '0';  wait for 20 ns;
        en <= '0';  wait for 20 ns;
        …
        wait;
    end process stimulus;
end architecture test_register_4;
```

Διαδικασία Εξομοίωσης Σχεδιασμού

Βασικός λόγος δημιουργίας ενός μοντέλου σχεδιασμού είναι η εξομοίωση.



Εκτέλεση εξομοίωσης

- Εκτέλεση των διαδικασιών

- Προσομοίωση διακριτών γεγονότων (*events*)
  - ο χρόνος εξελίσσεται με διακριτά βήματα (χρονικούς κύκλους)
  - όταν η τιμή ενός σήματος αλλάζει τότε έχουμε την εμφάνιση ενός γεγονότος

- Μια διαδικασία είναι ευαίσθητη σε γεγονότα των σημάτων εισόδου
  - όπως αυτά προσδιορίζονται από τις καταχωρήσεις wait
  - σε αυτά τα γεγονότα η διαδικασία επανακαλείται και προσδιορίζει νέες μελλοντικές τιμές για τα σήματα εξόδου

Αλγόριθμος Εξομοίωσης I

- **Φάση Αρχικοποίησης**
  - σε κάθε σήμα δίδεται η αρχική του τιμή
  - το "χρονόμετρο" της προσομοίωσης τίθεται στην τιμή 0
  - για κάθε διαδικασία
    - ενεργοποίηση
    - εκτέλεση μέχρι να προκύψει καταχώρηση wait οπότε και η διαδικασία οδηγείται σε αναμονή (αναστολή εκτέλεσης)
      - η εκτέλεση συνήθως περιλαμβάνει τη μελλοντική καταχώρηση τιμών σε σήματα (δηλ. προγραμματίζει για κάποιες μελλοντικές στιγμές την ύπαρξη κάποιων δραστηριοτήτων)

Αλγόριθμος Εξομοίωσης ΙΙ

- Κύκλος προσομοίωσης
  - εξέλιξη του χρόνου προσομοίωσης στον επόμενο χρονικό κύκλο όπου υπάρχει κάποια δραστηριότητα
  - στον τρέχοντα χρονικό κύκλο
    - ενημέρωση των τιμών των σημάτων
      - ύπαρξη γεγονότος αν η νέα τιμή διαφέρει από την παλιά
  - Για κάθε διαδικασία ευαίσθητη σε οποιοδήποτε γεγονός (ή της οποίας ο χρόνος αναμονής έχει λήξει)
    - επανάκληση
    - εκτέλεση μέχρι να προκύψει wait και εν συνεχεία αναμονή

- Η προσομοίωση τελειώνει όταν δεν υπάρχουν άλλες προγραμματισμένες δραστηριότητες

Σύνθεση

- Είναι ουσιαστικά ο **απώτερος στόχος** χρήσης μίας γλώσσας περιγραφής κυκλωμάτων
- Μεταφράζει την περιγραφή (μοντέλο) μιας σχεδίασης σε ένα **netlist** από κυκλωματικά στοιχεία μιας δεδομένης βιβλιοθήκης.
- Υπάρχουν **περιορισμοί** στον τρόπο ανάπτυξης της HDL περιγραφής μιας σχεδίασης ώστε να είναι συνθέσιμη.
  - ενώ όλες οι περιγραφές μπορούν να προσομοιωθούν δεν είναι κατά ανάγκη εφικτή η σύνθεση όλων των δυνατών περιγραφών!
- Η σύνθεση εξαρτάται από τα χρησιμοποιούμενα εργαλεία.
- Εκμεταλλεύεται τα πλεονεκτήματα που έχει κάθε αυτοματοποιημένη διαδικασία (*απεικόνιση σε διαφορετικές βιβλιοθήκες, βελτιστοποίηση, παραγωγή διανυσμάτων ελέγχου κλπ*).

Σχεδιαστική Μεθοδολογία