Ψηφιακή Σχεδίαση

Ενότητα 13: Εισαγωγή στην VHDL

Δρ. Αλέξανδρος Λαζαρίδης
alazaridis@uowm.gr

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοπός της ενότητας

- Να γίνει εισαγωγή στην VHDL και να αναλυθούν έννοιες της VHDL.

Εισαγωγή

- Η VHDL είναι μια γλώσσα που χρησιμοποιείται για την περιγραφή και μοντελοποίηση ψηφιακών κυκλωμάτων.
- VHDL: V_{HSIC} Hardware Description Language (VHSIC: Very High Speed Integrated Circuit = Πολύ μεγάλης ταχύτητας ολοκληρωμένο κύκλωμα).
- Αρχικοποιήθηκε από το DoD (Department of Defense-USA (Υπουργείο Αμύνης – ΗΠΑ)) στις αρχές του 1980.

Πεδία εφαρμογής της VHDL

- Εξομοίωση ορθής λειτουργίας (Simulation).
- Σύνθεση ψηφιακών κυκλωμάτων (Synthesis).
- Επιβεβαίωση ορθού σχεδιασμού (Design Verification).
- Μοντέλα προδιαγραφών (Specification Models).

Πλεονεκτήματα της γλώσσας VHDL (1)

- Παγκόσμιο πρότυπο (IEEE 1076-1987, 1076-1993).
  - Υποστήριξη από πληθώρα αναπτυξιακών εμπορικών εργαλείων σχεδιασμού (CAD tools).
  - Εύκολη μεταφορά κυκλωματικών περιγραφών σε διαφορετικά αναπτυξιακά περιβάλλοντα.
- Δυνατότητα περιγραφής κυκλώματος /συστήματος σε διαφορετικά ιεραρχικά επίπεδα.
  - Από επίπεδο πύλης μέχρι επίπεδο συστήματος.
- Υποστήριξη εναλλακτικών σχεδιαστικών μεθοδολογιών (Top-down, Bottom-up, Mixed).

Πλεονεκτήματα της γλώσσας VHDL (2)

- Ιεραρχική σχεδίαση (Block Diagrams, Components)
- Επαναχρησιμοποίηση σχεδιασθέντων υπομονάδων (reusable components).
  - Χρήση βιβλιοθηκών με σχεδιασθέντα κυκλώματα
  - Μικρότερος χρόνος ανάπτυξης βελτιωμένων εκδόσεων του κυκλώματος / συστήματος.
- Περιγραφή κυκλώματος / συστήματος ανεξάρτητα από την τεχνολογία υλοποίησης.
  - Επαναχρησιμοποίηση υπάρχουσας κυκλωματικής περιγραφής σε διαφορετικές τεχνολογίες.
  - Μικρότερος χρόνος ανάπτυξης βελτιωμένων εκδόσεων του κυκλώματος / συστήματος.

Πλεονεκτήματα της γλώσσας VHDL (3)

- Υποστήριξη ταυτόχρονων και ακολουθιακών δομών (concurrent and sequential constructions).
  - Οι περισσότερες γλώσσες (π.χ. C) υποστηρίζουν μόνο ακολουθιακές δομές.
  - Οι ταυτόχρονες δομές είναι απαραίτητες για την περιγραφή της λειτουργίας του υλικού.
- Εύκολη διαχείριση λαθών και επιβεβαίωση ορθής λειτουργίας.
  - Προσομοίωση, διαχείριση σφαλμάτων, επαλήθευση σχεδιασμού.

Επίπεδο Συστήματος

- Περιγραφή των προδιαγραφών του συστήματος.
- Δεν απαιτείται πληροφορία χρονισμών.
- Δεν απαιτείται ακριβής καθορισμός της αρχιτεκτονικής του κυκλώματος / συστήματος.
- Δυνατότητα εξομοίωσης και επιβεβαίωσης ορθής λειτουργίας.

Επίπεδο Συμπεριφοράς

- Αναλυτικότερη περιγραφή της συμπεριφοράς / λειτουργίας του κυκλώματος.
  - Καθορισμός των απαιτούμενων αλγορίθμων για την ικανοποίηση των προδιαγραφών του συστήματος.
  - Εμπεριέχει πληροφορίες χρονισμού.
- Μη αναλυτικός καθορισμός της αρχιτεκτονικής του κυκλώματος (καταχωρητές, μνήμες, συνδυαστικά κυκλώματα κλπ.).
  - Ένα μοντέλο περιγραφής σε επίπεδο συμπεριφοράς αποτελειται από τα λειτουργικά στοιχεία και τη διασύνδεση αυτών.
  - Κάθε λειτουργικό στοιχείο μπορεί να εμπεριέχει περισσότερα του ενός στοιχεία και πληροφορία χρονισμού.

Επίπεδα καταχωρητή και πύλης

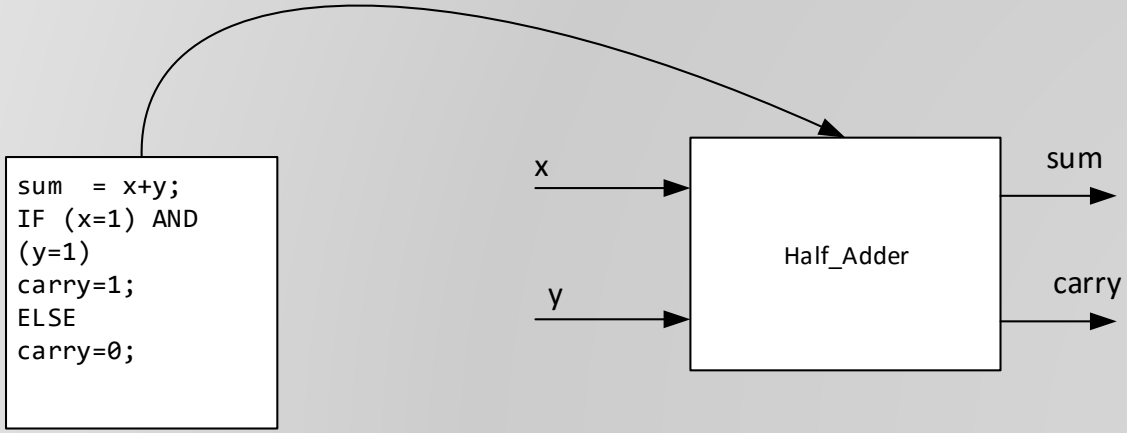
- Επίπεδο καταχωρητή (Register Transfer Level – RTL) .
  - Περιγραφή του κυκλώματος με χρήση συνδυαστικών κυκλωμάτων, καταχωρητών, μνημών, σύγχρονων και ασύγχρονων μηχανών πεπερασμένων καταστάσεων.
- Επίπεδο πύλης (Gate / Logic-Level)
  - Περιραφή του κυκλώματος σε επίπεδο πύλη.
  - Χρήση λογικών εξισώσεων (Boolean Functions).
  - Χρησιμοποιείται κυρίως για το σχεδιασμό βασικών συνδυαστικών κυκλωμάτων (αθροιστές. Πολ/στες κλπ.).
  - Υψηλοι χρόνοι σύνθεσης και εξομοίωσης.

Τρόποι περιγραφής κυκλωμάτων

- Συμπεριφοράς (Behavioral VHDL).
- Ροής Δεδομένων (Data flow VHDL).
- Δομής (Structural VHDL).
- Και οι τρεις παραπάνω μεθόδοι περιγραφής μπορούν να χρησιμοποιηθούν σε κάθε επίπεδο ροής σχεδιασμού.
- Καθώς μετακινούμαστε από το επίπεδο συμπεριφοράς στο επίπεδο δομής:
  - Αναλυτικότερη κυκλωματική περιγραφή.
  - Καλύτερο έλεγχο της σύνθεσης του κυκλώματος.
  - Μεγαλύτερος κώδικας.
  - Υψηλότεροι χρόνοι εξομοίωσης.

Συμπεριφορική VHDL (Behavioral) (1)

- Χρησιμοποιείται για την μοντελοποίηση της συμπεριφοράς του κυκλώματος σε υψηλό και αφηρημένο επίπεδο.
  - Αλγοριθμική περιγραφή της λειτουργίας του κυκλώματος.
  - Δεν περιγράφεται αναλυτικά η κυκλωματική δομή του κυκλώματος.
  - Δεν απαιτούνται αναλυτικές λογικές εξισώσεις.

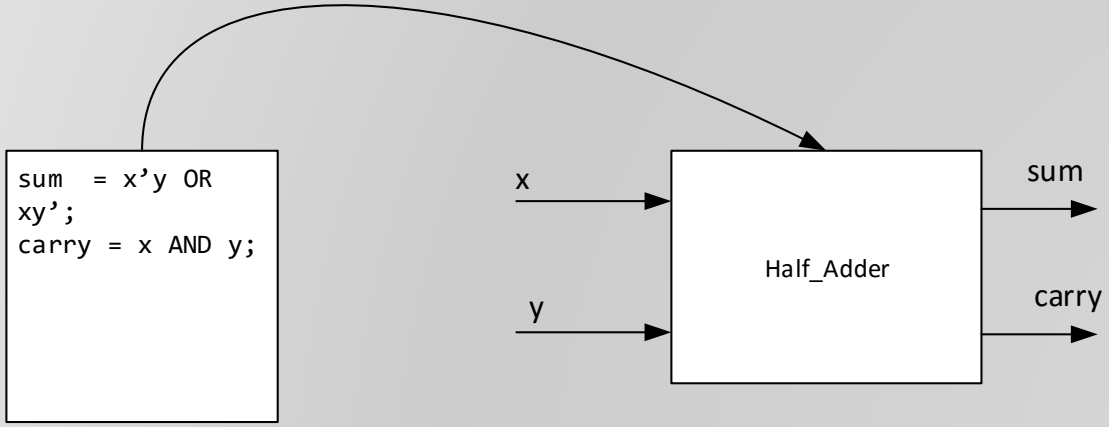


Συμπεριφορική VHDL (Behavioral) (2)

- Δυνατότητα εξομοίωσης για την επιβεβαίωση ορθής λειτουργίας του κυκλώματος.
- Επαλήθευση μέσω back-annotated πληροφορίας, επιτρέπει επιπλέον να γίνει επιβεβαίωση ορθής λειτουγίας με στοιχεία της τεχνολογίας ολοκλήρωσης.

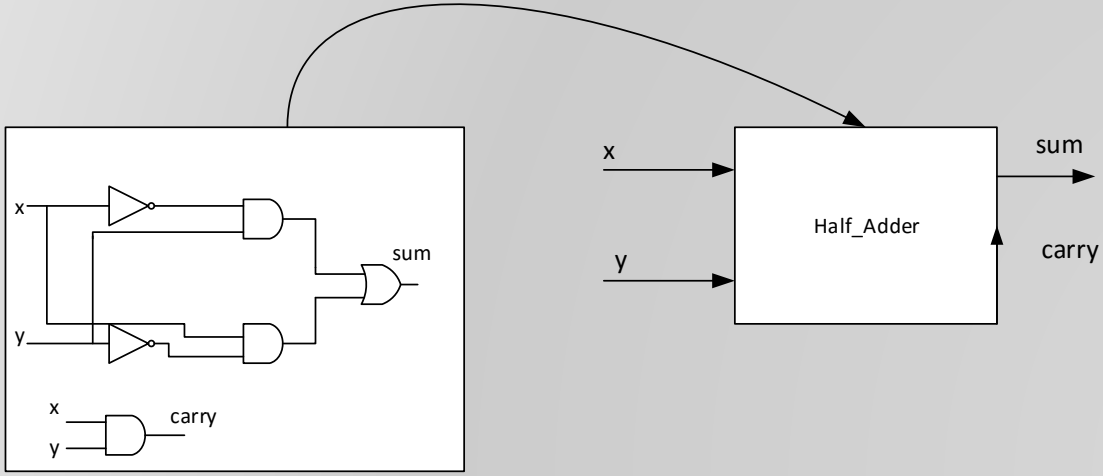
VHDL ροής δεδομένων (Data – Flow)

- Αναλυτικότερη περιγραφή της λειτουργίας του κυκλώματος.
- Χρήση λογικών εξισώσεων για την μοντελοποίηση της ροής δεδομένων.
- Δυνατότητα εξομοίωσης και επιβεβαίωσης ορθής λειτουργίας.



VHDL δομής (Structural)

- Χρησιμοποιείται για την περιγραφή της διασύνδεσης των δομικών μονάδων του κυκλώματος.
  - Οι δομικές μονάδες ποικίλουν από απλές πύλες μέχι σύνθετα κυκλώματα.
  - Προυποθέτει την ύπαρξη βιβλιοθήκης από σχεδιασθέντα δομικά στοιχεία και την δυνατότητα χρήσης αυτών.



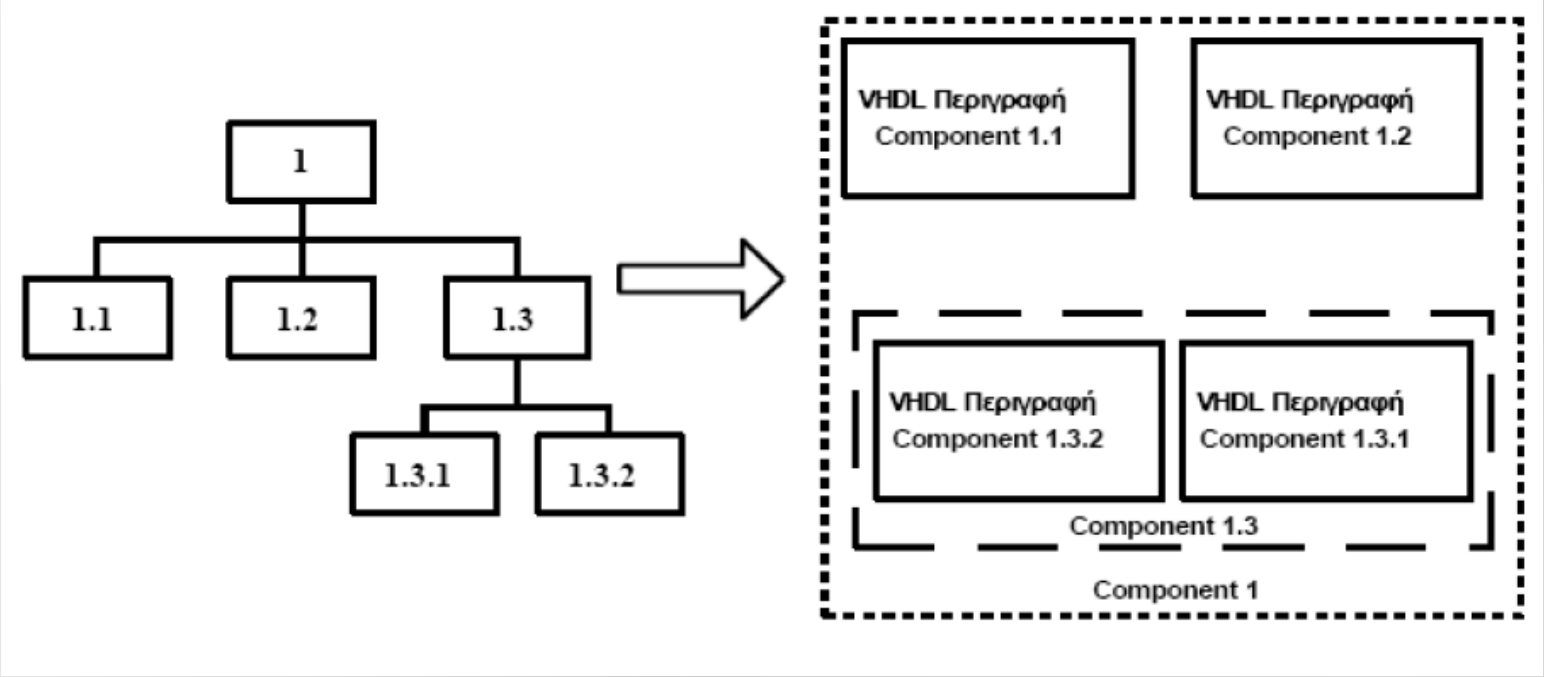
Ιεραρχική σχεδίαση (1)

- Περιγραφή του κυκλώματος σε διαφορετικά ιεραρχικά επίπεδα.
  - Ένας ιεραρχικός σχεδιασμός αποτελείται από υπομοβάδες που εμπεριέχουν άλλες υπομονάδες, VHDL κώδικες ή συνδυασμούς αυτών.
- Στόχος: Η ευκολότερη κατανόηση και διαχείριση του σχεδιασμού.
  - Η πολυπλοκότητα δεν μειώνεται πάντοτε!
- Αποτέλεσμα: Η γρήγορη και αποτελεσματικότερη σχεδίαση πολύπλοκων κυκλωμάτων.

Ιεραρχική σχεδίαση (2)

- Βασική ιδέα: Απόκρυψη μη χρήσιμης πληροφορίας σε κάθε ιεραρχικό επίπεδο.
- Αρχή του μαύρου κουτιού: Μόνο οι είσοδοι / έξοδοι και η λειτουργία του κάθε component (συμπεριφορά των εισόδων εξόδων) είναι ορατές.

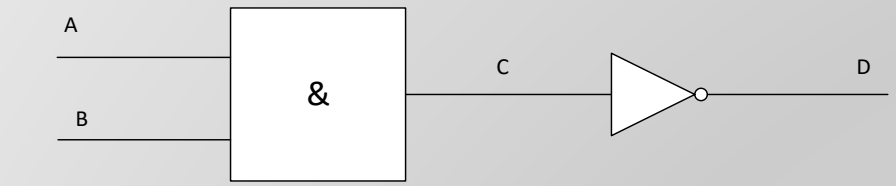
Ιεραρχική σχεδίαση (3)



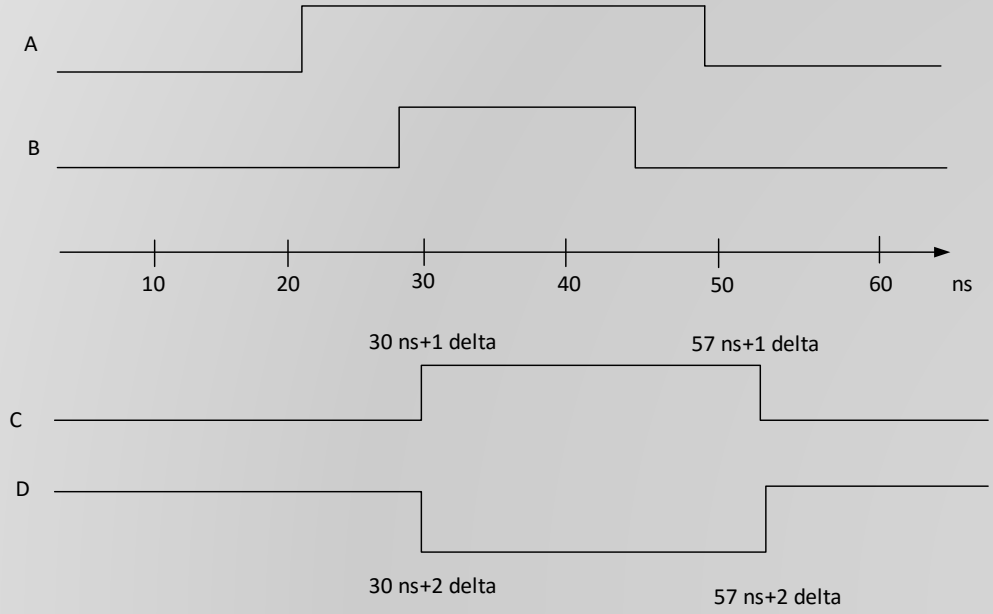
Delta Time (1)

- Χρησιμοποιείται από τον εξομοιωτή για την χρονοδρομολόγηση συμβάντων.
- Delta delay χρησιμοποιείται όταν δεν ορίζεται άλλος τύπος καθυστέρησης.
- Είναι απειροελάχιστος (infinitisimal) χρόνος μη ορατός από τον χρήστη.
  - Δεν εισάγει καμία πραγματική καθυστέρηση.
- B <= a; -- Το σήμα b παίρνει την τιμή του a μετά από delta delay.

Delta Time (2)



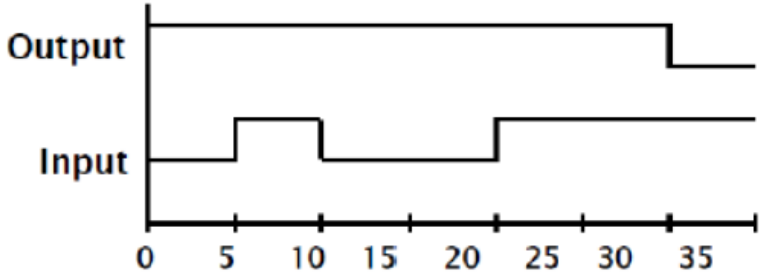

```
c <= a AND b;
d <= NOT c;
```



Αδρανειακή Καθυστέρηση (Inertial Delay)

- Default delay type.
- Επιτρέπει τον ορισμό καθυστέρησης από τον χρήστη.
- Απορροφά παλμούς μικρότερης διάρκειας απο την προκαθορισμένη καθυστέρηση.

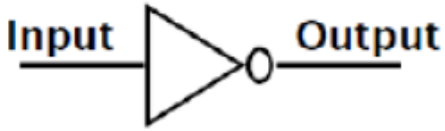
```
-- Inertial is the default
Output <= NOT Input AFTER 10 ns;
```

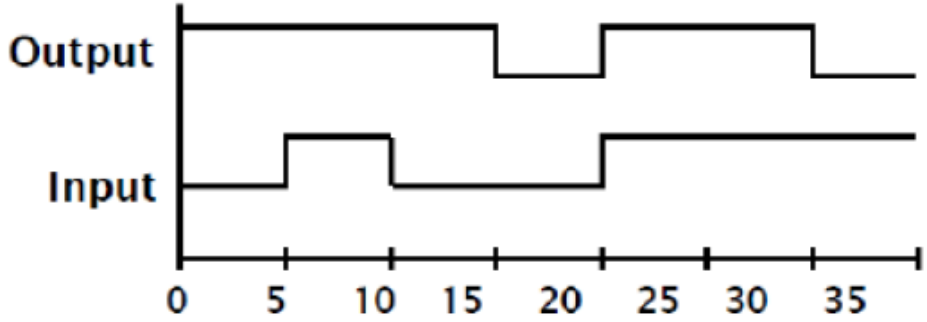


Μεταφορά καθυστέρησης (Transport Delay)

- Πρέπει να καθορίζονται ρητά από τον χρήστη.
- Επιτρέπει στον χρήστη που καθορίζεται η καθυστέρηση.
- Περνάει όλες τις μεταβάσεις εισόδου με καθυστέρηση.

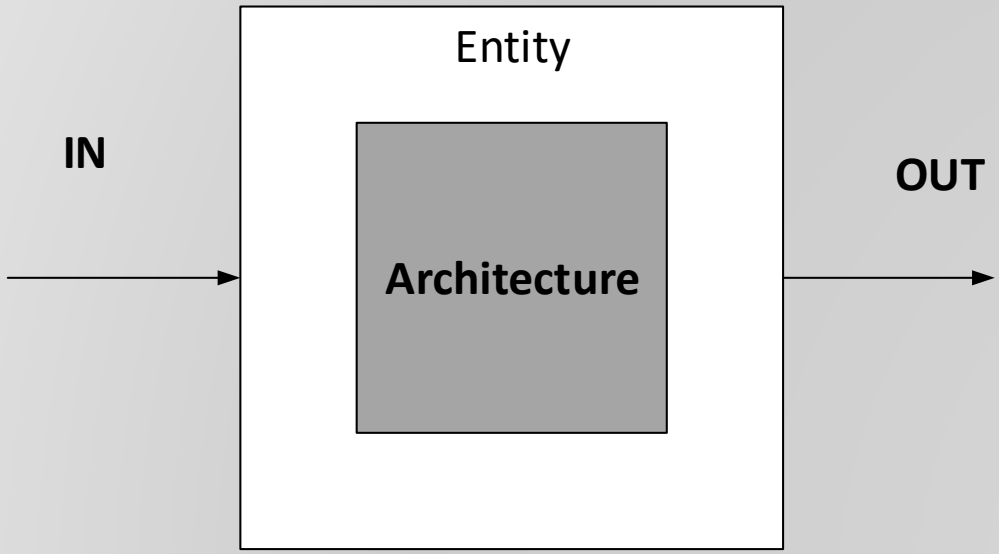
```
-- TRANSPORT must be specified
Output <= TRANSPORT NOT Input AFTER 10 ns;
```





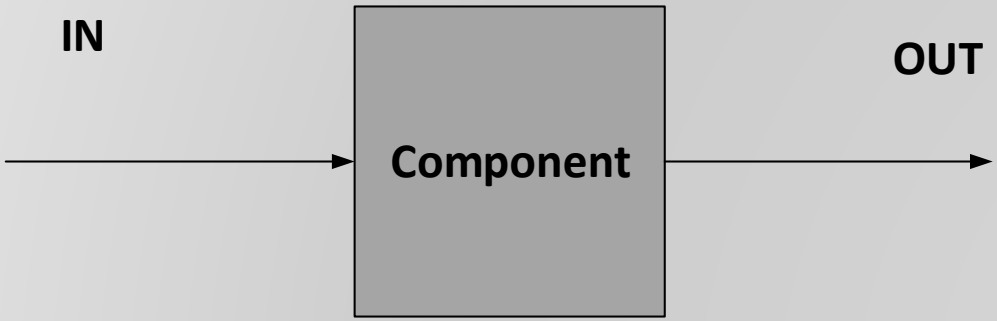
Υπομονάδες (Components) (1)

- Κάθε ομάδα αποτελειται από δυο μέρη:
  - Entity – Σηματα εισόδου / εξόδου (in / out ports).
  - Architecture – Περιγραφή της λειτουργίας του είτε σε επίπεδο συμπεριφοράς είτε σε επίπεδο κυκλώματος.



Υπομονάδες (Components) (2)

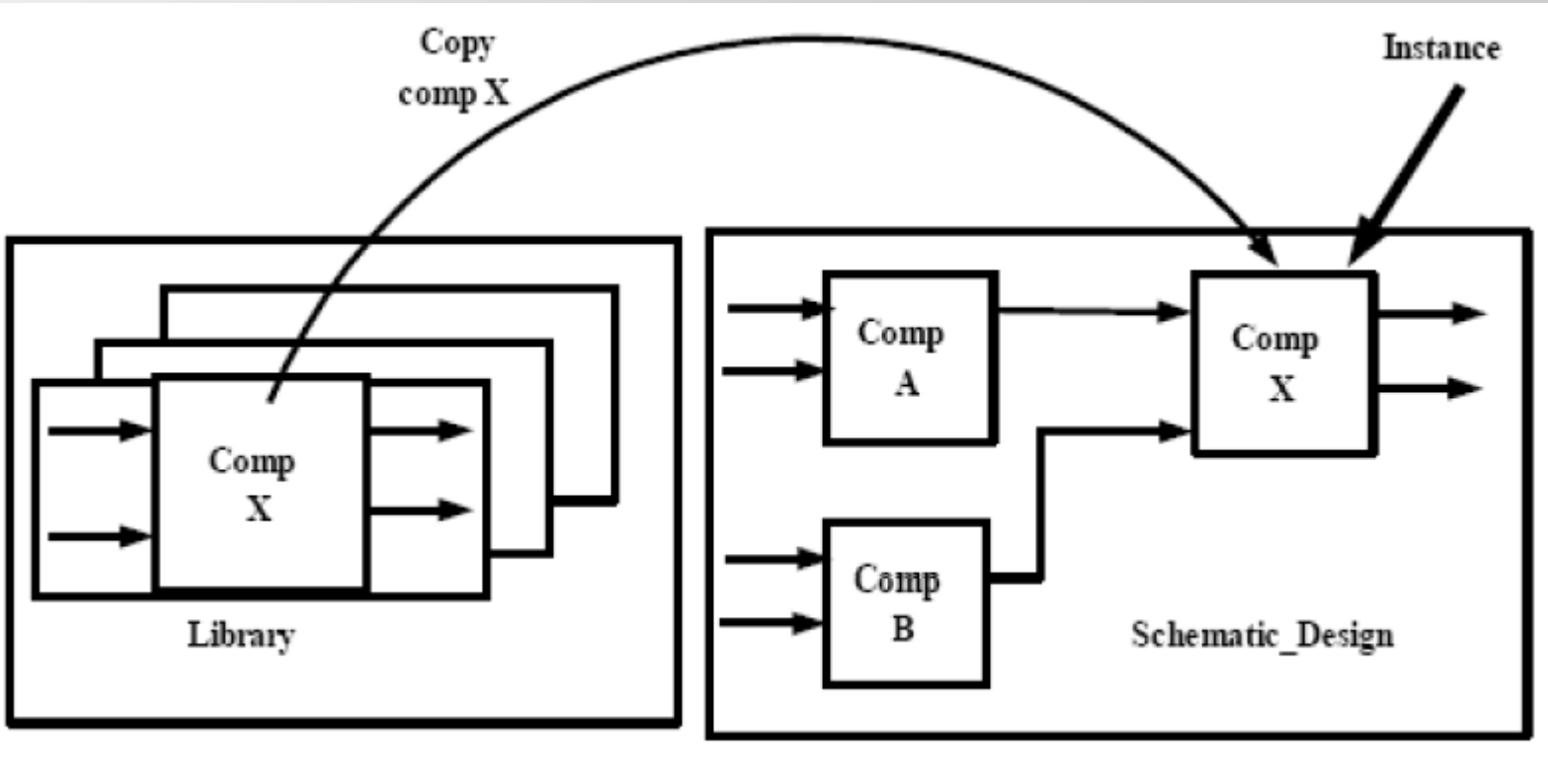
- Αρχή του μάυρου κουτιού
  - Μόνο η γνώση των σημάτων εισόδου / εξόδου και της συμπεριφοράς αυτών απαιτούνται για την χρήση ενός component.
  - Δε μας ενδιαφέρει η υλοποίηση του.



Υπομονάδες (Components) (3)

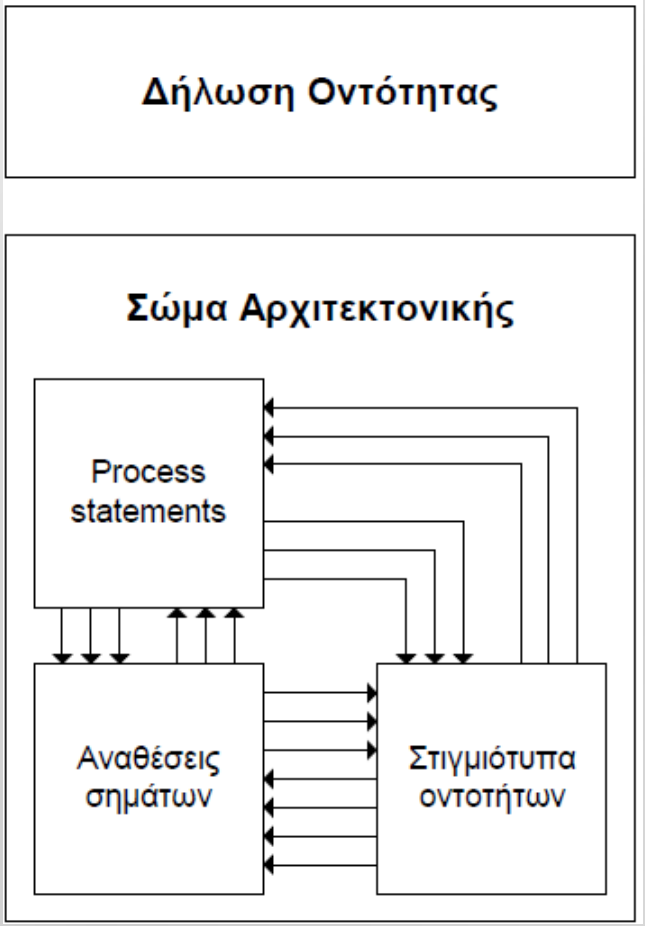
- Θεμελιώδης έννοια της VHDL. Χρησιμοποιούνται για την περιγραφή της λειτουργίας απλών πυλών μέχρι συστημάτων.
- Αποθήκευση σχεδιασθέντων υπομονάδων σε βιβλιοθήκη και επαναχρησιμοποίηση αυτών.

Υπομονάδες (Components) (4)



Βασική δομή ενός VHDL αρχείου

- **Δήλωση οντότητας (entity declaration):** δήλωση σημάτων εξόδου.
- **Σώμα Αρχιτεκτονικής (architectural body):** περιγραφή της οντότητας, μπορεί να αποτελείται από διασυνδεδεμένες δομικές μονάδες (components), διαδικασίες (processes) και αναθέσεις σημάτων, τα οποία λειτουργούν παράλληλα.



Δήλωση οντότητας (1)

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

Δήλωση οντότητας (2)

- **Τύποι:**
  - bit/ bit_vector std_logic/std_logic_vector Boolean/integer/real/character time
- **Mode:**
  - **in**: είσοδος
  - **out**: έξοδος που δεν διαβάζεται εσωτερικά
  - **buffer:** έξοδος που διαβάζεται και εσωτερικά
  - **input**: είσοδος/ έξοδος
- **std_logic**: ορίζεται στο package std_logic_1164 της IEEE βιβλιοθήκης, προτιμάται από το bit type, περιγράφει ένα ψηφιακό σύστημα με ακρίβεια αφού παίρνει τιμές “0”, “1”, “-” (don’t care), “Z” (high impedance – μία γραμμή δεν οδηγείται), κ.α.

Δήλωση οντότητας (3)

- Προαιρετικές
  - Καθορίζουν τοπικές σταθερές (χρονισμούς – μεγέθη)

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

Δήλωση οντότητας (4)

- Άθροιση δύο 4-bit αριθμών

```
Entity adder is
port ( a , b: in std_logic_vector ( 3 downto 0 );
sum : out std_logic_vector ( 4 downto 0 ) );   end adder;
```

Σώμα αρχιτεκτονικής

- Καθορίζει πώς λειτουργεί το κύκλωμα και πώς υλοποιείται.

Architecture architecture_name **of** NAME_ENTITY **is**
-- Declarations
-- components declarations
-- signal declarations
-- constant declarations
-- function declarations
-- procedure declarations
-- type declarations

:
begin
-- Statements
:
End architecture_name;

Περιγραφή δομής (structural) (1)

- Τα συστήματα περιγράφονται ως διασύνδεση components. Υπονοείται μία ιεραρχική δομή. Για κάθε χρησιμοποιούμενο component πρέπει να γίνει: Δήλωση του component στο σώμα αρχιτεκτονικής. όπου θα χρησιμοποιηθεί Instantiation του component, όπου γίνεται και η διασύνδεσή του με το περιβάλλον σύστημα.
- Η δήλωση του component είναι όμοια με την δήλωση της αντίστοιχης οντότητας .

Περιγραφή δομής (structural) (2)

- Τα συστήματα περιγράφονται ως διασύνδεση components. Υπονοείται μία ιεραρχική δομή.
- Για κάθε χρησιμοποιούμενο component πρέπει να γίνει:
  - Δήλωση του component στο σώμα αρχιτεκτονικής όπου θα χρησιμοποιηθεί .
  - Instantiation του component, όπου γίνεται και η διασύνδεσή του με το περιβάλλον σύστημα.

    instance_name: component name

    port map (port1 => signal1, port2 => signal2, … port3 => signaln);

    Πέρασμα παραμέτρων στις θύρες με βάση το όνομα

    ή

    port map (signal1, signal2,…signaln);
- με βάση τη θέση (το πρώτο port αντιστοιχεί στο πρώτο σήμα, το δεύτερο port αντιστοιχεί στο δεύτερο σήμα, κ.ο.κ.). Η θέση των σημάτων πρέπει να είναι στην ιδία σειρά με τα component’s ports.

Περιγραφή δομής (structural) (3)

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

signal VEH_AT_MAIN_INVERT, INT: std_logic;
begin
            U0: NOT1 port map (out1=>VEH_AT_MAIN_INVERT, in1=>VEH_AT_MAIN);
            U1: AND2 port map (VEH_AT_MAIN, VEH_AT_LOCAL, INT);
            U2: OR2 port map (VEH_AT_MAIN_INVERT, INT, RED);
end Structural;
```

Περιγραφή υπερχείλισης (Data flow)

- Τα συστήματα περιγράφονται με concurrent statements (ταυτόχρονες δηλώσεις).
- Υπάρχουν τρεις βασικές κατηγορίες assignments:
  - Απλά assignments (όπως αυτά που χρησιμοποιούνται μέσα σε processes, με τη διαφορά ότι εδώ εκτελούνται παράλληλα, και για αυτό ονομάζονται concurrent (ταυτόχρονες)).
  - Υπό συνθήκη (conditional) assignments.
  - Assignments επιλεγμένων σημάτων (selected signals).

Απλές αναθέσεις (assignments)

Target_signal <= expression

```
Sum <= ( A xor B ) xor Cin;
Carry <= (A and B);
Z <= ( not X ) or Y after 2 ns;
```

Εξαρτώμενες αναθέσεις (Conditional assignments) (1)

Target_signal <= *expression* **when** Boolean_condition **else**

expression **when** Boolean_condition **else**

.

.

expression;

- Το target signal θα πάρει την τιμή της πρώτης έκφρασης για την οποία η Boolean condition θα είναι αληθής.
- Η δομή when-else είναι χρήσιμη για την περιγραφή λογικών συναρτήσεων που έχουν μορφή πίνακα αληθείας.

Προϋπόθεση ανάθεσης - Conditional assignments (2)

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

- Παράδειγμα (2-to-1 mux):

```
entity MUX_2_1 is
port (A, B: in std_logic;
                SEL: in std_logic;
                Z: out std_logic); end MUX_2_1;
architecture concurr_MUX21 of MUX_2_1 is
begin
        Z <= A when SEL = '0' else
                B;

end concurr_MUX21;
```

Αναθέσεις επιλεγμένων σημάτων - Selected signals assignments (1)

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

- Όπως και η δομή when-else είναι χρήσιμη για την περιγραφή μιας συνάρτησης σαν πίνακα αλήθειας.
- Είναι ισοδύναμη των case δομών μέσα σε processes.

Αναθέσεις επιλεγμένων σημάτων - Selected signals assignments (2)

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

- Παράδειγμα (2-to-1 mux):

```
entity MUX_2_1 is
port (A, B: in std_logic;
        SEL: in std_logic;
        Z: out std_logic);
end MUX_2_1;
architecture concurr_MUX21b of MUX_2_1 is
 begin
        with SEL select
         Z <= A when '0',
        B when '1';
 end concurr_MUX21b;
```

Συμπεριφορική περιγραφή (Behavioral)

- Η βασική δομή σε behavioral περιγραφές είναι η process
  - Η δομή process δηλώνεται μέσα σε μία αρχιτεκτονική και εκτελείται παράλληλα με άλλες δομές (concurrent – ταυτόχρονα).
  - Οι εντολές εσωτερικά στη δομή εκτελούνται ακολουθιακά.
  - Στη δομή process υπάρχει μία λίστα με σήματα στα οποία είναι «ευαίσθητη» η διαδικασία process.
  - Κάθε αλλαγή στην τιμή αυτών των σημάτων προκαλεί την άμεση εκτέλεση της διαδικασίας.
  - Εναλλακτικά μπορεί να συμπεριληφθεί μία εντολή WAIT.
  - Μεταβλητές/ σταθερές ορίζονται στο τμήμα δηλώσεων της, πριν το begin.

Η δομή διεργασίας (process)

- Μία process χωρίς wait statement εκτελείται για πάντα

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

Δήλωση αναμονής (Wait statement)

- Καθορίζουν πότε αντιδρούν οι processes σε αλλαγές στις τιμές σημάτων.
- Το wait statement αναγκάζει μία process να σταματήσει και ορίζει υπό ποια συνθήκη θα ξεκινήσει πάλι.
- Μπορούμε να συμπεριλάβουμε κάθε συνδυασμό ή και κανέναν από αυτούς σε μια wait statement.

wait_statement <=

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

Wait statements – Λίστα ευαισθησίας (1)

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

Wait statements – Λίστα ευαισθησίας (2)

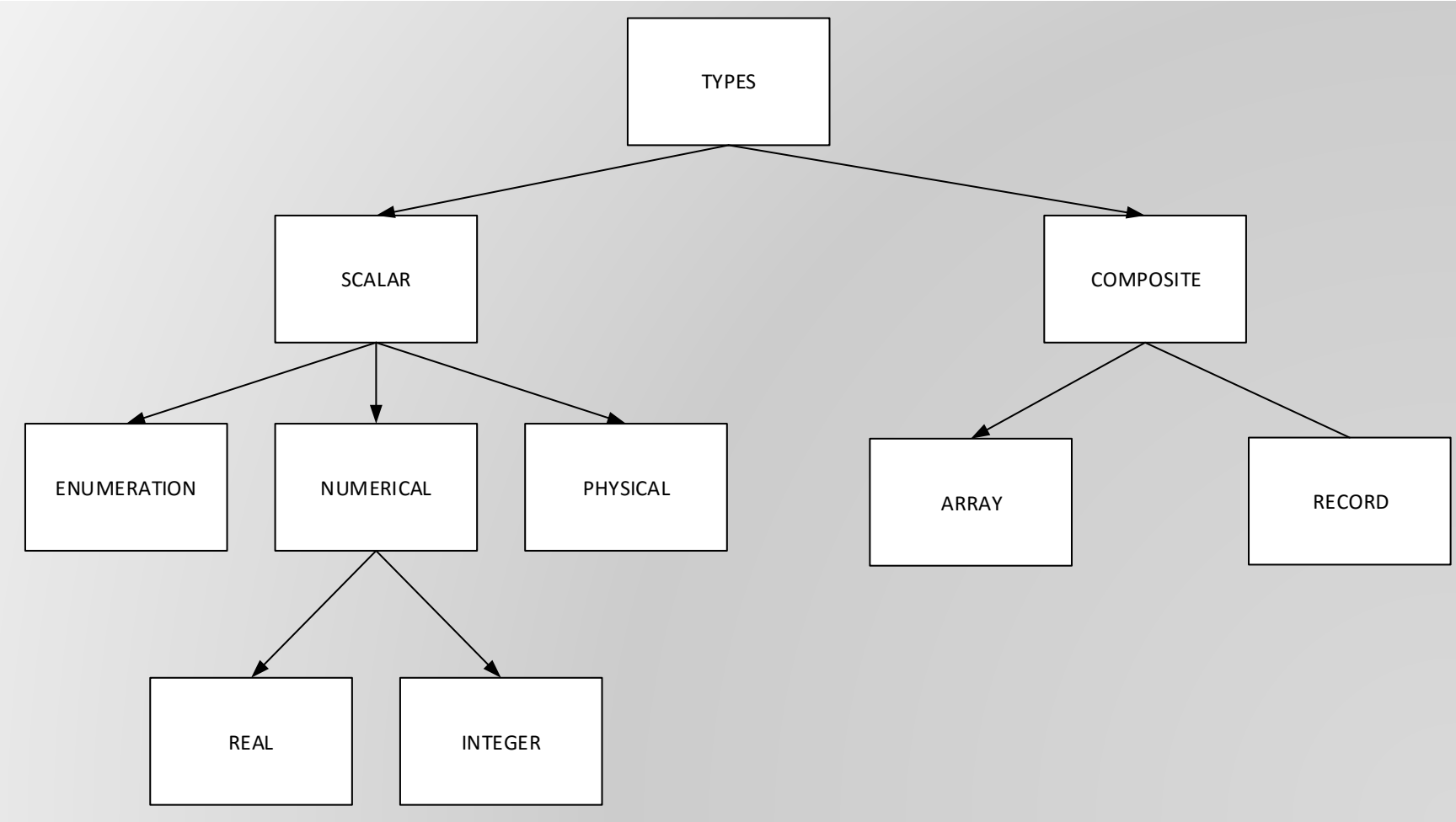
```
half_add : process
begin
        sum<=a xor b;
        carry<= a and b;
        wait on a,b;
end process;
```

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

Wait statements – Λίστα ευαισθησίας (3)

- Ένα wait statement με μία λίστα ευαισθησίας είναι ισοδύναμο με μία process που εμπεριέχει λίστα ευαισθησίας.
- Το παραπάνω παράδειγμα περιγράφει συνδυαστική λογική. Συνήθως, τα processes χρησιμοποιούνται για την περιγραφή ακολουθιακής λογικής.

Τύποι δεδομένων



Τύπος απαρίθμησης (Enumerated Type)

- Καθορίζει ένα σύνολο τιμών που μπορεί να πάρει ένα αντικείμενο.
- Χρησιμοποιούνται σε μεγάλο βαθμό για τη δήλωση των καταστάσεων μιας FSM.
- Παράδειγμα
  - **TYPE** bit **IS** (‘0’, ‘1’);
  - **TYPE** states **IS** (IDLE, RECEIVE, SEND);
- Ένα αντικείμενο τύπου bit (παραπάνω) μπορεί να πάρει μόνο τις τιμές 1 και 0.

Αριθμητικοί τύποι (Numerical types)

- **Ακέραιο:** Καθορίζει ένα σύνολο ακέραιων τιμών.
  - **TYPE** my_int **IS** RANGE 0 **TO** 15
- Η VHDL εμπεριέχει έναν προκαθορισμένο ακέραιο τύπο-integer.
  - **TYPE** integer **IS** RANGE -2147483647 **TO** 2147483647;
  - Είναι χρήσιμο να δηλώνεται το ακριβές εύροστου ακεραίου για λόγους μνήμης.
- **Πραγματικός:** Καθορίζει ένα εύρος πραγματικών αριθμών.
  - **TYPE** real **IS** RANGE -1.0e38 **TO** 1.0e38;
- ΠΡΟΣΟΧΗ!!!! Δεν υποστηρίζεται από κανένα εργαλείο σύνθεσης.

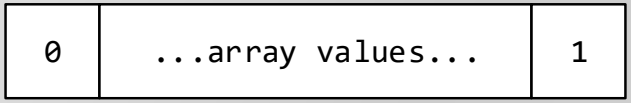
Φυσικοί Τύποι - Physical Types

- Η VHDL υποστηρίζει τύπους για τη μοντελοποίηση φυσικών μεγεθών όπως ο τύπος time.
  - **TYPE** time **IS** RANGE -2147483647 **TO** 2147483647
    units
    fs;
    ps=1000 fs;
    ns=1000 ps;
    …..
    END units;
- Υποστηρίζει επίσης αντιστάσεις και πυκνωτές.
- ΠΡΟΣΟΧΗ!!! Δεν υποστηρίζονται από κανένα εργαλείο σύνθεσης
  - Χρησιμοποιούνται μόνο για εξομοίωση και για καθορισμό των προδιαγραφών (simulation and specification models).

Τύποι Array

```
TYPE data_bus IS ARRAY (0 TO 31) OF BIT;
```

0...element numbers… 31




```
VARIABLE X:     data_bus;
VARIABLE Y:     BIT

Y := X(12);     -- Y gets value of 12th element
```

```
TYPE register IS ARRAY (15 DOWNTO 0) OF BIT;
```

IEEE-1076 Προκαθορισμένοι τύποι

- Το πρότυπο IEEE-1076 περιλαμβάνει τους ακόλουθους τύπους δεδομένων.
  - **TYPE** bit **IS** (‘0’, ‘1’);
  - **TYPE** bit_vector **IS** array (integer RANGE <>) **OF** bit;
  - **TYPE** integer **IS** RANGE minint **TO** maxint;
  - **SUBTYPE** positive **IS** integer RANGE 0 **TO** maxint;
  - **SUBTYPE** natural **IS** integer RANGE 0 **TO** maxint;
  - **TYPE** boolean **IS** (true, false);

Τύποι δεδομένων std_ulogic / std_ logic (1)

- Std_ulogic
  - ‘U’ -- Un-initialized
  - ‘X’ -- Forcing unknown
- Std_logic
  - Ίδιες τιμές με std_ulogic.

  Επίλυση συγκρούσεων (conflicts) όταν ένα σήμα οδηγείται από περισσότερους του ενός σήματα / οδηγούς (drivers).

Τύποι δεδομένων std_ulogic / std_ logic (2)

```
ENTITY ex IS
  PORT  (d, c, en1, en2: IN std_logic;
           dbus: OUT std_logic);
END;
```

- Το παρακάτω είναι σφάλμα:

```
ARCHITECTURE rtl OF ex IS
BEGIN
  dbus <= d WHEN en1 = '1' ELSE 'Z'
  dbus <= c WHEN en2 = '1' ELSE 'Z'
END;
```

Εντολή Record

- Χρησιμοποιείται για τη συλλογή ενός ή περισσότερων στοιχείων διαφορετικου τύπου σε ένα ενιαίο κατασκεύασμα.
- Τα στοιχεία μπορεί να είναι οποιουδήποτε VHDL τύπου.
- Η πρόσβαση στα στοιχεία γίνεται μεσω ονόματος πεδίου.

```
TYPE binary IS ( ON, OFF );
TYPE switch_info IS
        RECORD
                status : binary;
                Idnumber : integer;
END RECORD;

VARIABLE switch : switch_infol

switch.status := on; -- status of the switch
switch.Idnumber := 30; -- number  of the switch
```

Εντολή Subtype

- Επιτρέπει στον χρήστη να ορίζει περιορισμούς σε έναν τύπο δεδομένων.
- Μπορεί να περιέχει ένα ολόκληρο εύρος από βασικούς τύπους.
- Αναθέσεις πού είναι εκτός εύρους της υποκατηγορίας τύπων επιστρέφουν σφάλμα.
- Παράδειγμα Subtype:

SUBTYPE name IS base_type RANGE <user range>;

SUBTYPE first_ten IS INTEGER RANGE 0 to 9

Διανύσματα (Vectors)

- Χρησιμοποιούνται για την μοντελοποίηση διαύλων (busses).
  - Δηλώνονται στις οντότητες ή και στην αρχιτεκτονική.

```
ENTITY ex IS
  PORT ( a: IN std_logic_vector ( 3 DOWNTO 0 );
         b : OUT bit_vector ( 0 TO 4 ) );
END;

ARCHITECTURE rtl OF ex IS
SIGNAL s1 :std_logic_vector ( 3 DOWNTO 0 );
SIGNAL s2 : std_ulogic_vector ( 0 TO 2 );
BEGIN
        ......
END;
```

Ανάθεση τιμής σε διάνυσμα (1)

```
ARCHITECTURE rtl OF ex IS
SIGNAL a, b : std_logic_vector ( 3 DOWNTO 0 );
SIGNAL c,   : std_logic_vector ( 3 DOWNTO 0 );
BEGIN
        a <= “0110”;  -- ( a(3), a(2), a(1), a(0) ) = ( 0110 )
        b <= “1101”; -- ( b(3), b(2), b(1), b(0) ) = ( 1101 )

        c <= a AND b; -- ( c(3), c(2), c(1), c(0) ) = ( 0100 )

END;
```

- Τα διανύσματα στα οποία επιτελείται η λογική πράξη πρέπει να είναι του ίδιου εύρους όπως και το διάνυσμα στο οποίο ανατίθεται το αποτέλεσμα.

- Κατά την ανάθεση τιμής πρέπει να λαμβάνονται υπόψη το έυρος αλλά και η διεύθυνση δήλωσης του διανύσματος.

Ανάθεση τιμής σε διάνυσμα (2)

- Ισοδύναμες αναθέσεις :

```
a(2) <= d(0);  b(2) <= c(2);
a(1) <= d(1);  b(1) <= d(1);
a(0) <= d(2);  b(0) <= d(0);
```

Ανάθεση τιμής σε διάνυσμα (3)

- **ARCHITECTURE** rtl **OF** ex **IS**
  SIGNAL a : std_logic_vector (4 **DOWNTO** 0);
  SIGNAL b : std_ulogic_vector (0 **TO** 4);
  SIGNAL c : std_logic;
  BEGIN
  a <= “0110”;
  b(4) <= c;
  b (0 to 3) <= a (3 **DOWNTO** 0) ;
  END;

- **ARCHITECTURE** rtl **OF** ex **IS**
  SIGNAL a : std_logic_vector (2 **DOWNTO** 0);
  SIGNAL b : std_ulogic_vector (3 **DOWNTO** 0);
  BEGIN
  a <= b; <- **ΛΑΘΟΣ!!!** Δίαυλοι όχι ίσου εύρους
  END;

Ακολουθιακές Δομές (Sequential Structures)

- Η λειτουργία του υλικού είναι «παράλληλη» από τη φύση της.
  - Όλα τα κυκλώματα ενεργοποιούνται ακαριαία σε τυχόν αλλαγή των εισόδων τους.
  - Χρήση ταυτόχρονων δομών (concurrent structures) – Εκτελούνται **ταυτόχρονα** (concurrently) ανεξάρτητα της σειράς εμφάνισης τους στον κώδικα.
  - Χρήση ακολουθιακών δομών για πληρέστερη μοντελοποίηση.
  - Οι ακολουθιακές δομές εκτελούνται **ακολουθιακά** σύμφωνα με τη σειρά εμφάνισης τους στον VHDL κώδικα.
  - Χρησιμοποιούνται για ακολουθιακή επεξεργασία δεδομένων (sequential data processing).

Μεταβλητές

- Target_identifier>=<selected_expression>
- <selected_expression> = <identifier> [(and ! or… ! xor)] <identifier> …. [(and ! or… ! xor)]
- Δηλώνονται και χρησιμοποιούνται σοτν VHDL κώδικα.
- Χρησιμοποιούνται για αποθήκευση προσωρινών αποτελεσμάτων.
- Δεν αποτελούν ηλεκτρικές συνδέσεις.
- **PROCESS**(….)

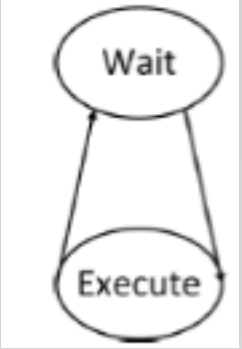
```
VARIABLE a, b, c : std_logic:     --Δήλωση μεταβλητών
c:=a AND b:                       --Χρήση μεταβλητών
```

Σήματα και μεταβλητές

- Η βασική διαφορά σήματος και μεταβλητής συνιστάται στον τρόπο ανάθεσης τιμών.
  - Η μεταβλητή είναι «ακολουθιακή» οντότητα ενώ το σήμα «συντρέχουσα» οντότητα.
  - Η τιμή ανατίθεται στο σήμα μετά το delta time ενώ στην μεταβλητή ακαριαία.

Διεργασία (Process) (1)

- Απαιτέι θεμελιώδη έννοια της VHDL.
  - Προέρχεται από το συμβατικό λογισμικό και αντιστοιχεί σε ένα ακολουθιακό πρόγραμμα.
- Καταστάσεις: Εκτέλεσης – Αναμονής
- Μέσω αλλαγής κατάλληλων σημάτων η διεργασία εκτελειται και επιστρέφει σε κατάσταση αναμονής.
  - Ο χρόνος εκτέλεσης ισούται με 1 delta_time.



Διεργασία (Process) (2)

- Περισσότερες από μια διεργασίες μπορεί να εκτελούνται ταυτόχρονα.
  - Διεγείρονται από συντρέχουσε δομές (σήματα)
  - Εντός της διεργασίας ο κώδικας εκτελείται ακολουθιακά.
- Είναι ισοδύναμο με την ταυτόχρονη (παράλληλη) εκτέλεση περισσότερων του ενός προγραμμάτων συμβατικού λογισμικού.

Διεργασία – Σύνταξη (1)

- [<process_name:] **PROCESS** [(sensitivity_list)]

  [<process_declarative_part>]

  BEGIN

  <process_statement_part>

  END PROCESS [<process_name>];
- Η διεγασία διεγείρεται μέσω αλλαγής κάποιου σήματος.
  - της sensitivity_list
  - του wait statement

Διεργασία – Σύνταξη (2)

- Παράδειγμα:

```
ff: PROCESS (a,b) –sensitivity list
BEGIN
q<=a;
z<=b;
END;
```

```
cc: PROCESS
        BEGIN
    WAIT ON a,b ; --wait statement
        q<=a;
            z<=d;
        END;
```

Διεργασία – Σύνταξη (3)

ff: **PROCESS** (a,b) --sensitivity list

BEGIN

q <= a;

z <=b;

END

cc: **PROCESS**

BEGIN

WAIT ON a, b; -- wait statement

q <= a;

z <= d;

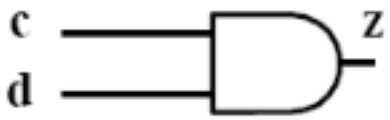
END

- Απαγορεύεται ταυτόχρονη χρήση wait και sensitivity_list!!!

Διεργασία – Παράδειγμα (1)

- Η process είναι ένα concurrent statement (ταυτόχρονη δήλωση) => 1 σήμα εξόδου.

```
ARCHITECTURE sequential OF ex IS
SIGNAL z, a, b, c, d : std_logic;
BEGIN
PROCESS (a, b, c, d)
z <= a AND b;
z <= c AND d;
END process;
END concurrent;
```



Διεργασία – Παράδειγμα (2)

- **SIGNAL** a, b, c, d: **INTEGER** := 0;

```
one: PROCESS (a)
BEGIN
a <= 1;
IF (a=1) THEN b<=1; END IF;
a<= '0'
END PROCESS;

two: PROCESS (a)
VARIABLE c: INTEGER;
BEGIN
c:=1;
IF (c=1) THEN d<= '1'; END IF;
END PROCESS;
```

- Το b <= 1 δε θα γίνει ποτέ. Το a <= 1 γίνεται όταν η διεργασία ενεργοποιείται για πρώτη φορά. Όμως επικαλύπτεται από το a <= 0.
- Το d <= 1 γίνεται κανονικά διότι η μεταβλητή παίρνει ακαριαία τιμή.

Είδη Διεργασιών

- Συνδυαστικές διεργασίες (Combinational processes).
- Διεργασίες χρονοδρομολογούμενες από ρολόι (Clocked Processes).

Συνδυαστικές Διεργασίες

- Χρησιμοποιούνται για την μοντελοποίηση συνδυαστικών κυκλωμάτων.
- Όλα τα σήματα εισόδου πρέπει να περιέχονται στο sensitivity_list ή στο wait statement.
  - Ένα συνδυαστικό κύκλωμα διεγείρεται μέσω αλλαγής της εισόδου.
  - Η διεργασία πρέπει να διεγείρεται από **όλα** τα σήματα εισόδου.
  - Παράλειψη σήματος εισόδου οδήγεί σε μη διέργερση της διεργασίας σε τυχόν αλλαγή αυτού.

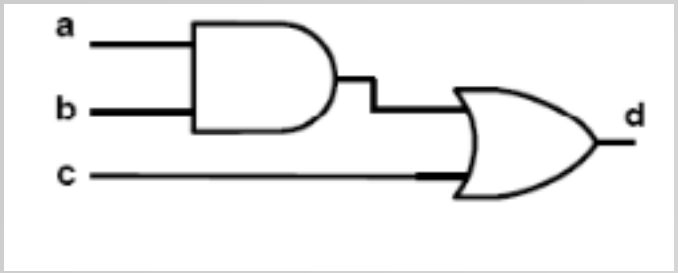
Συνδυαστικές Διεργασίες – Παράδειγμα (1)

- Ένα σύνηθες λάθος είναι να μην περιέχεται ένα απαιτούμενο σήμα στη sensitivity list.

```
PROCESS ( a, s )
BEGIN
IF ( s = '1' ) THEN
        y <= a;
ELSE y <= b;
```

Συνδυαστικές Διεργασίες – Παράδειγμα (2)

```
good:  PROCESS ( a, b, c )
BEGIN
d<= ( a AND b ) OR c;
END PROCESS;
```



Χρονισμένες Διεργασίες (1)

- Χρησιμοποιούνται για τη μοντελοποίηση σύγχρονων ακολουθιακών κυκλωμάτων.
- Για τη διέγερση τους απαιτείται αλλαγή του σήματος ρολογιού.
  - Ένα ακολουθιακό κύκλωμα αλλάζει κατάσταση μόνο στις χρονικές στιγμές αλλαγής του ρολογιού.

Εναλλακτικές περιγραφές του ρολογιού (1)

- Alt 1:

```
PROCESS (CLK)
BEGIN
        IF clk'event AND clk='1' THEN
        q<=d;
        END IF;
END PROCESS
```

- Alt 2:

```
PROCESS (CLK)
BEGIN
        IF clk='1' THEN
        q<=d;
        END IF;
END PROCESS
```

- Alt 4:

```
PROCESS (CLK)
BEGIN
        WAIT UNTIL clk='1'
        q<=d;
END PROCESS
```

- Alt 5:

```
PROCESS (CLK)
BEGIN
        WAIT UNTIL rising_edge(clk);
        q<=d;
END PROCESS
```

Εναλλακτικές περιγραφές του ρολογιού (2)

- Alt 3:

```
PROCESS (CLK)
BEGIN
        IF clk'event AND clk='1' AND clk'last_value  = '0' THEN
        q<=d;
        END IF;
END PROCESS;
```

- Οι 1, 2, 3 και 4 είναι πιο ευρέως χρησιμοποιούμενες και υποστηρίζονται από την πληθώρα των εργαλείων σύνθεσης.
- Η 3 είναι η πιο πλήρης για σύνθεση.

Χρονισμένες Διεργασίες (2)

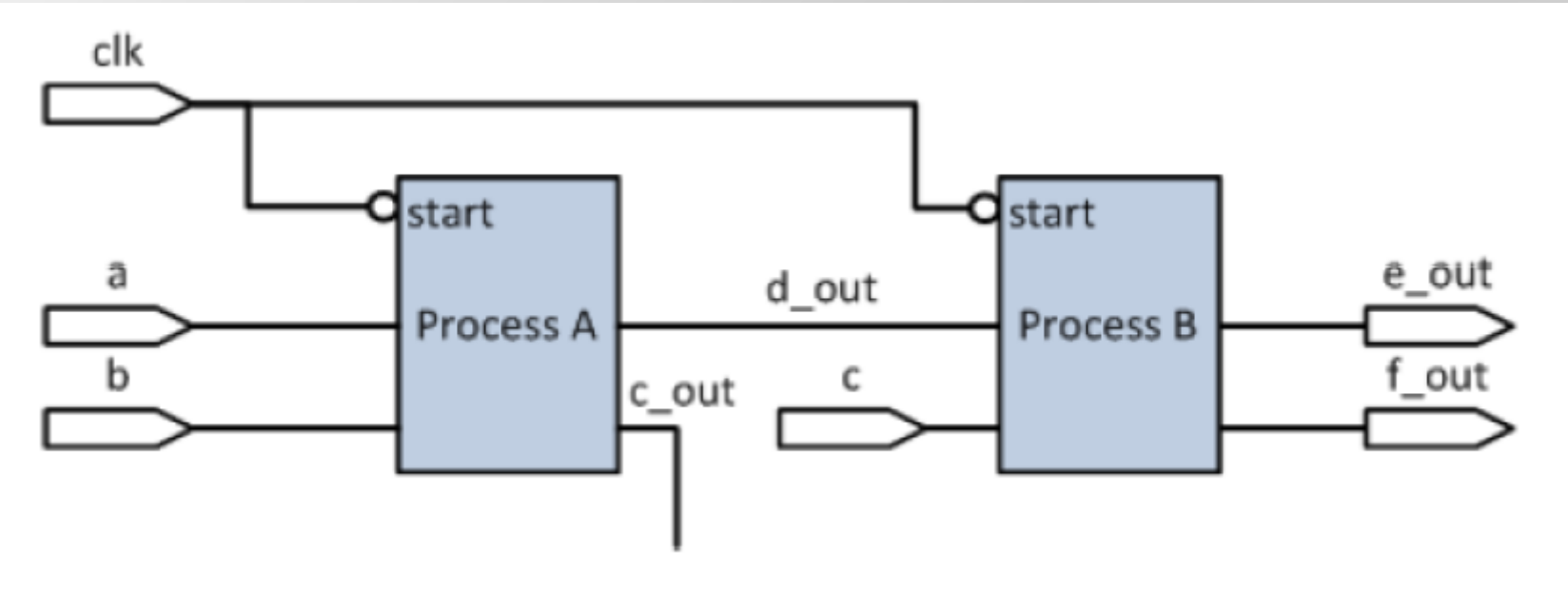
- A:

```
PROCESS
BEGIN
   WAIT UNTIL clk = ‘0’
   c_out<= NOT (a AND b);
   d_out<= NOT b;
END PROCESS a;
```

- B:

```
PROCESS
BEGIN
   WAIT UNTIL clk = ‘0’
   e_out<= NOT (d_out AND c);
   f_out<= NOT c;
END PROCESS b;
```

Χρονισμένες Διεργασίες (3)



Χρονισμένες Διεργασίες (4)

- Σε μια clocked διεργασία όλα τα σήματα στα οποία γίνεται ανάθεσης τιμής οδηγούνται από flip-flop.

Χρονισμένες Διεργασίες (5)

- Παράδειγμα

example : **PROCESS**

BEGIN

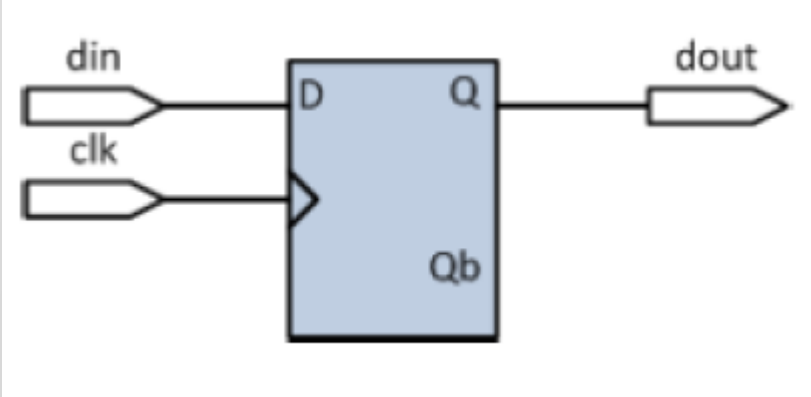
WAIT UNTIL clc = ‘1’;

dout <= din;

END;

Χρονισμένες Διεργασίες (6)

- Παράδειγμα (συνέχεια)



Βιβλιογραφία

Βασισμένες στο υλικό των:

ΑΣΗΜΑΚΗ-ΒΟΥΡΒΟΥΛΑΚΗ-ΚΑΚΑΡΟΥΝΤΑ-ΛΕΛΙΓΚΟΥ

Τέλος Ενότητας