# Πανεπιστήμιο Πατρών

## Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

### Εργαστήριο Σχεδίασης Ολοκληρωμένων Κυκλωμάτων

## Σχεδιασμός Ολοκληρωμένων Κυκλωμάτων (VLSI) II

## *Εισαγωγή στη VHDL και το Εργαλείο Modelsim*

**ΕΚΠΟΝΗΣΗ: Γ. Σ. Αθανασίου - Α. Εμερετλής - Γ. Θεοδωρίδης**

## Περιεχόμενα:

# 1. Εισαγωγή στο Modelsim

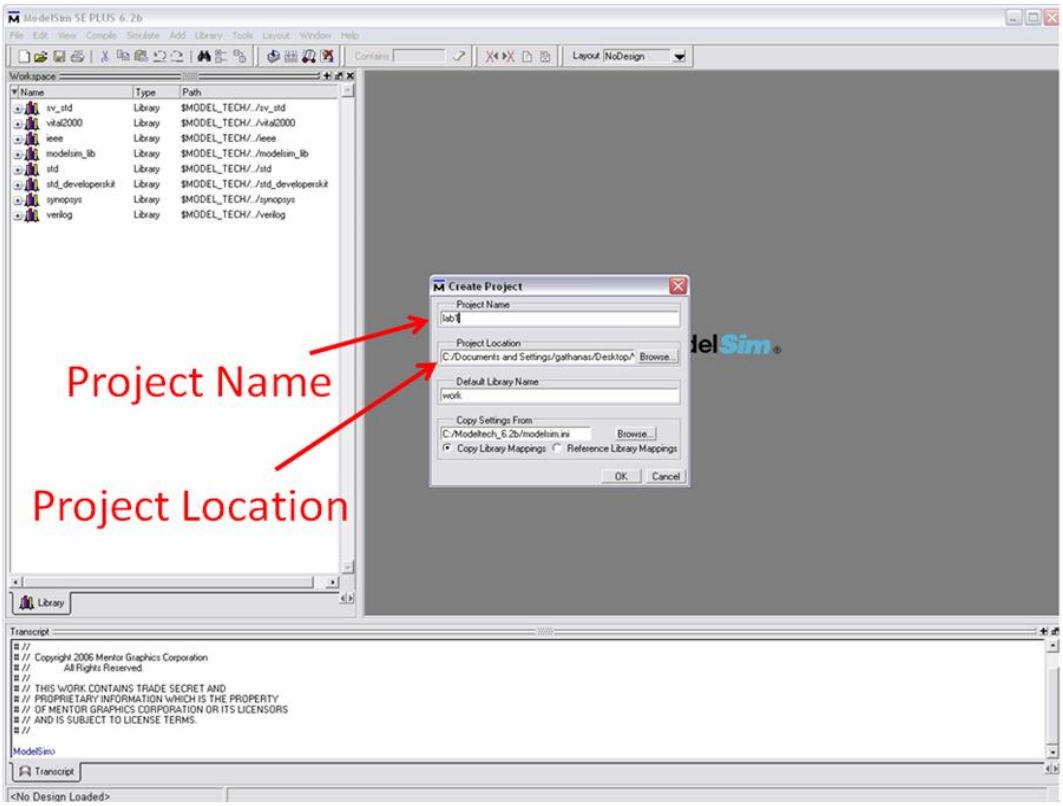
## 1.1 Τι είναι το Modelsim ?

***Modelsim**: a comprehensive simulation and debug environment for complex ASIC and FPGA designs.*

Δηλαδή, **<u>είναι</u>** ένα περιβάλλον για λογική προσομοίωση (simulation) κυκλωμάτων που έχουν περιγραφεί σε γλώσσα *Verilog*, *VHDL* ή *SystemC*.
**<u>Δεν είναι</u>** εργαλείο σύνθεσης, όπως για παράδειγμα το ***XST του Xilinx ISE Design Suite***.
Επομένως, στο **Modelsim** ελέγχουμε τη λογική ορθότητα του κυκλώματος που σχεδιάζουμε.

## 1.2 Δημιουργία Project και συγγραφή κώδικα

File → New → Project



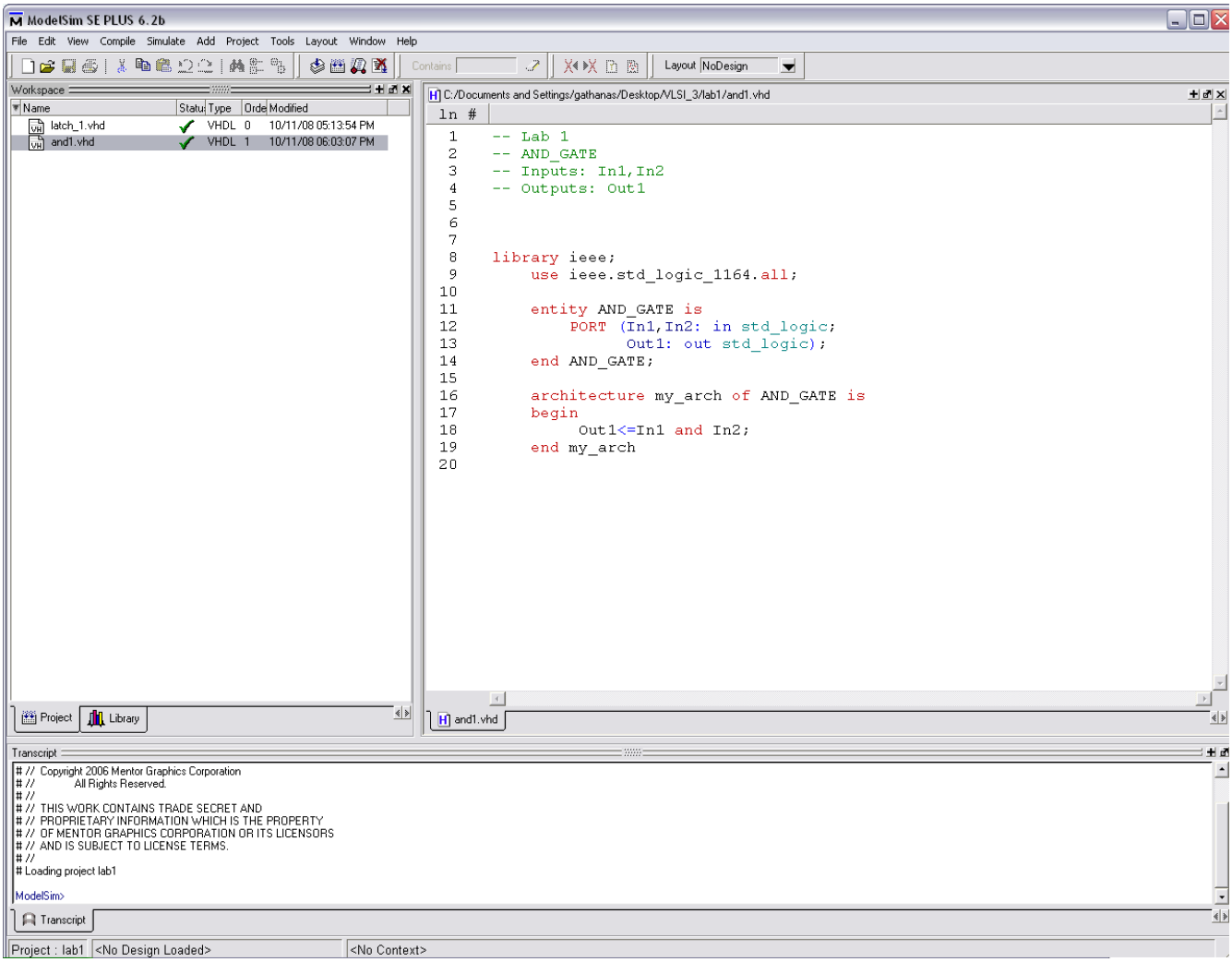

File → New → Source → VHDL

Στη συνέχεια, υπάρχει επιλογή για εισαγωγή ενός υπάρχοντος αρχείου VHDL (.vhd) ή η δημιουργία ενός καινούριου. Επιλέγουμε τη δημιουργία ενός νέου αρχείου με όνομα ***and1*.**

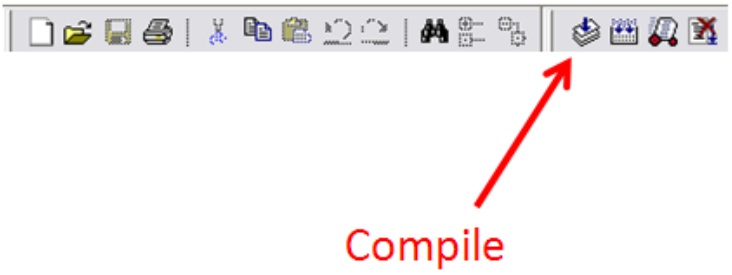
Κατόπιν, συντάσσουμε τον κώδικα που περιγράφει μία απλή πύλη AND, όπως φαίνεται παρακάτω.

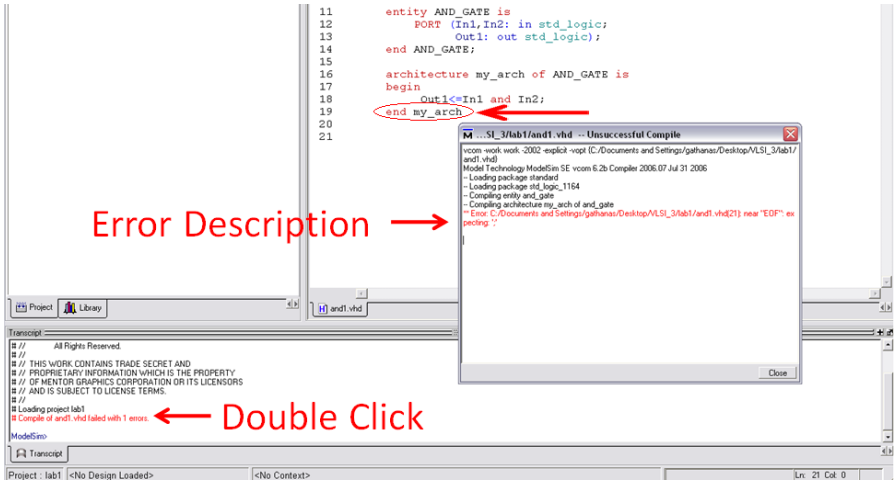
Συνεχίζουμε με μεταγλώττιση του κώδικα, αποσφαλμάτωση και προσομοίωση. Τα βήματα αυτά παρουσιάζονται και αναλύονται παρακάτω.

```
1    -- Lab 1
2    -- AND_GATE
3    -- Inputs: In1,In2
4    -- Outputs: Out1
5
6
7
8    library ieee;
9        use ieee.std_logic_1164.all;
10
11       entity AND_GATE is
12           PORT (In1,In2: in std_logic;
13                 Out1: out std_logic);
14       end AND_GATE;
15
16       architecture my_arch of AND_GATE is
17       begin
18            Out1<=In1 and In2;
19       end my_arch
20
```

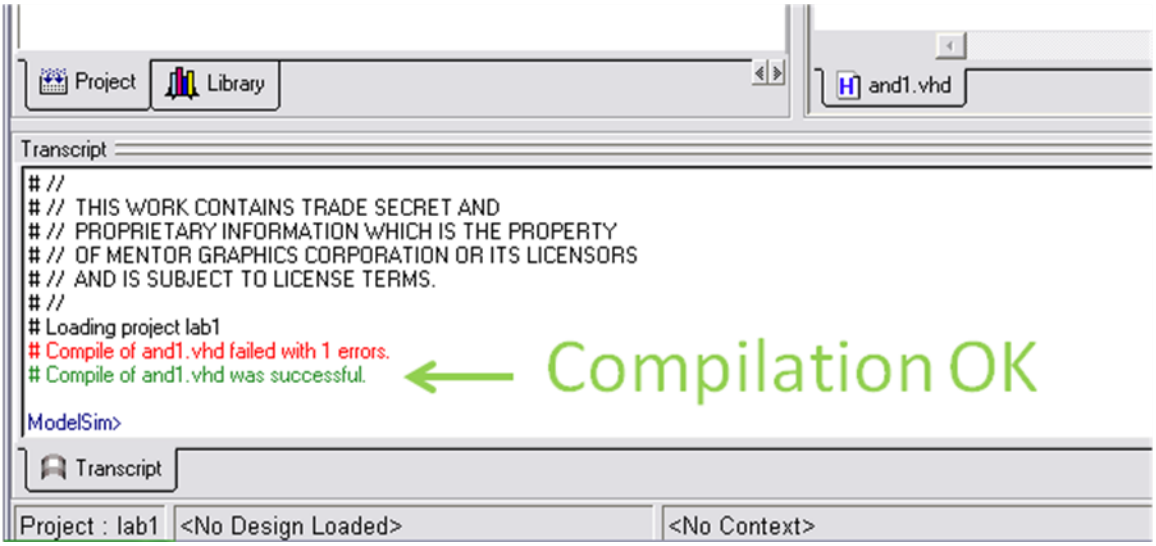


## 1.3 Compilation και Error Correction

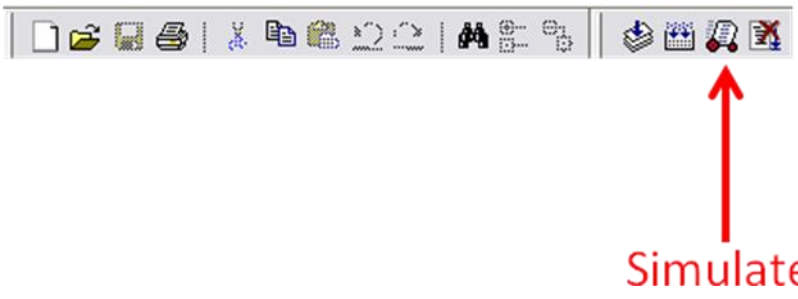


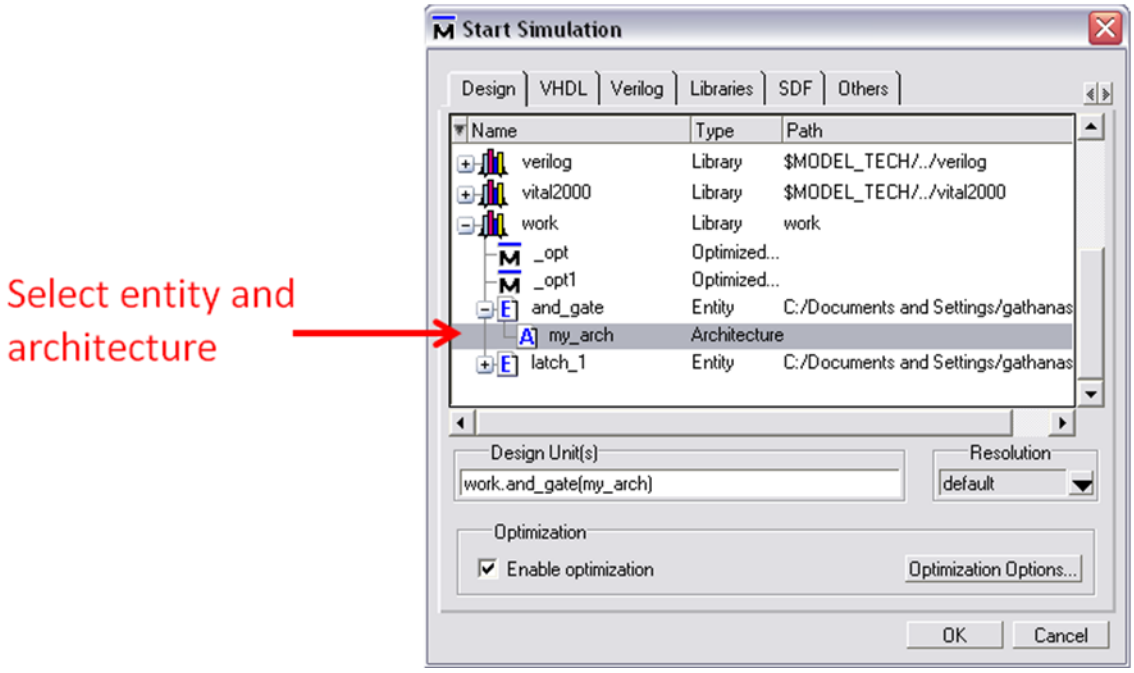



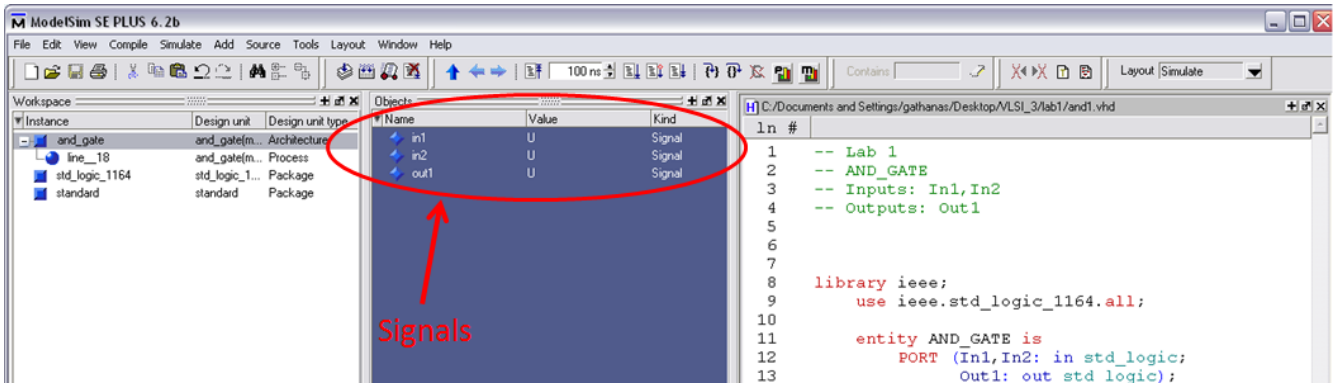

**<u>Λάθος:</u>**
Λείπει το ερωτηματικό στο τέλος της γραμμής ολοκλήρωσης της αρχιτεκτονικής, **end my_arch**;



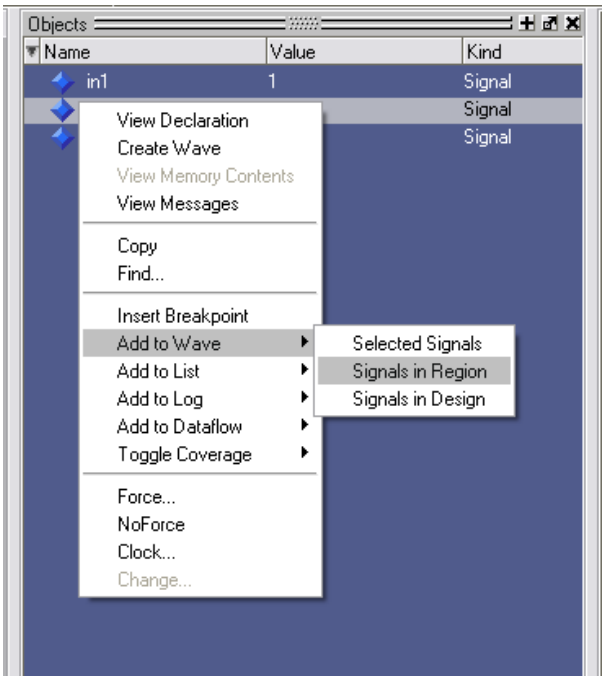
## 1.4 Simulation and Waveforms

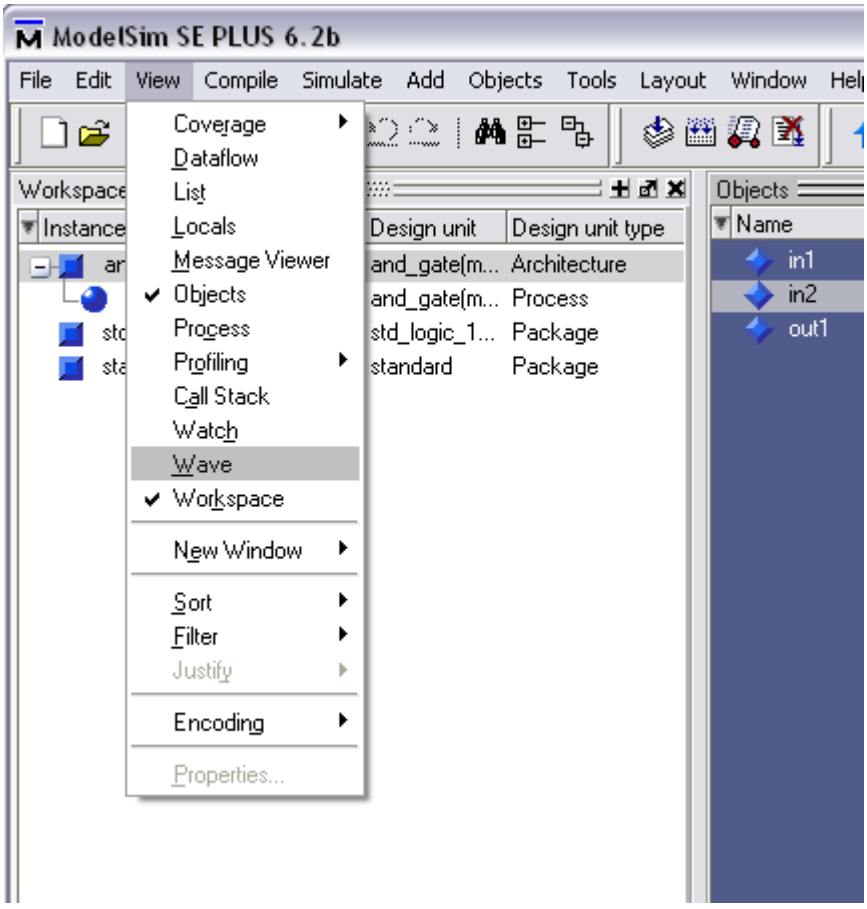


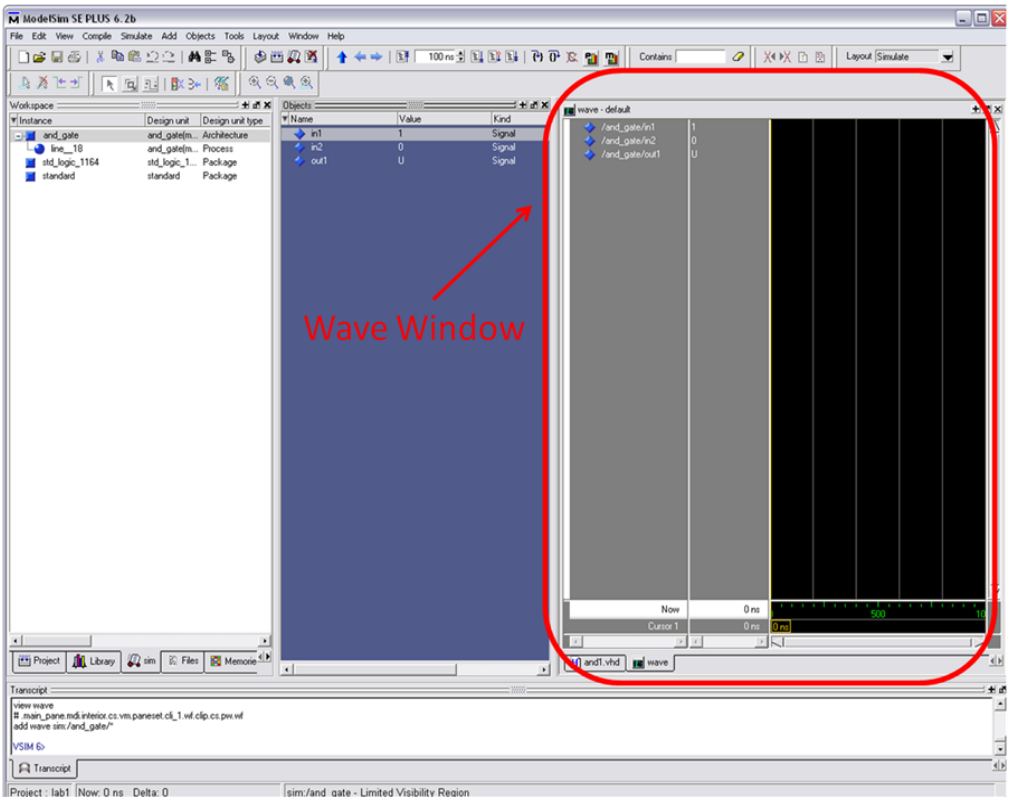



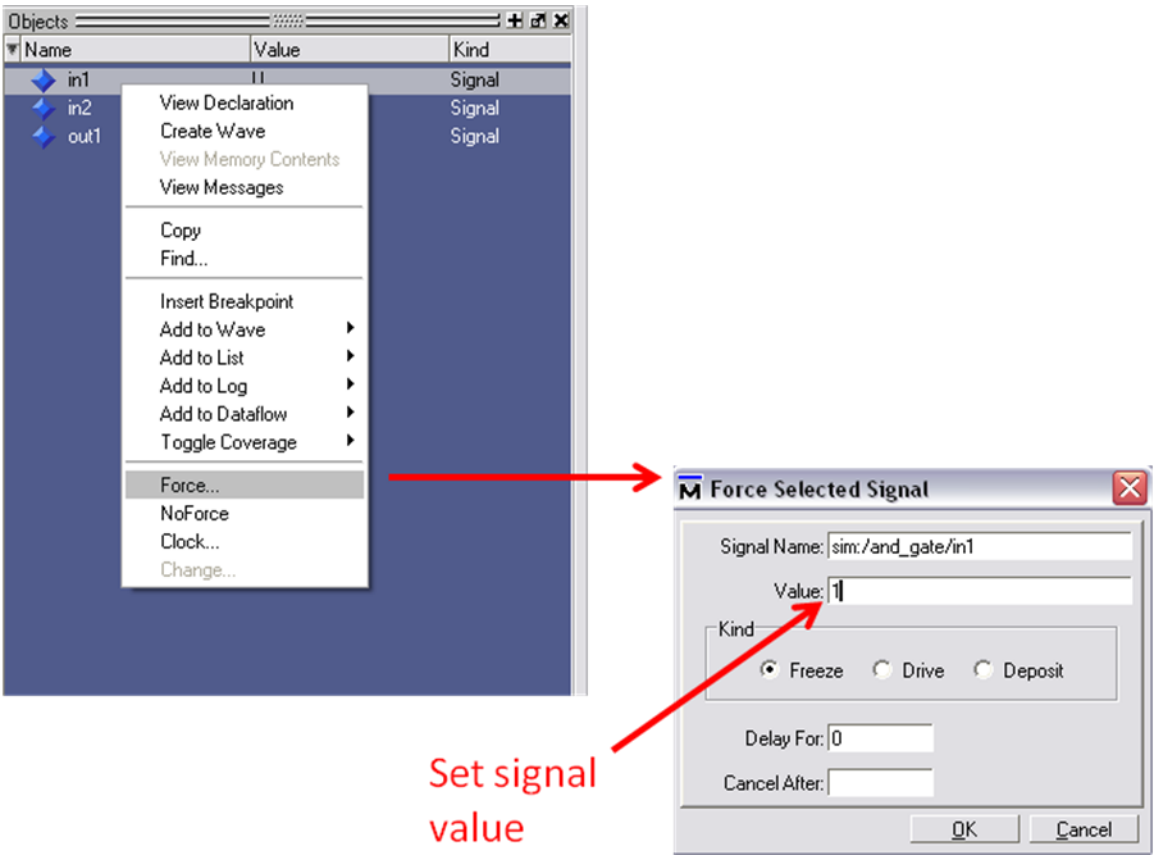

Δεξί click και επιλογή όλων των σημάτων για να εμφανιστούν στο waveform.


Objects
Name
Value
Kind
in1
1
Signal
Signal
Signal
View Declaration
Create Wave
View Memory Contents
View Messages
Copy
Find...
Insert Breakpoint
Add to Wave
Add to List
Add to Log
Add to Dataflow
Toggle Coverage
Force...
NoForce
Clock...
Change...
Selected Signals
Signals in Region
Signals in Design

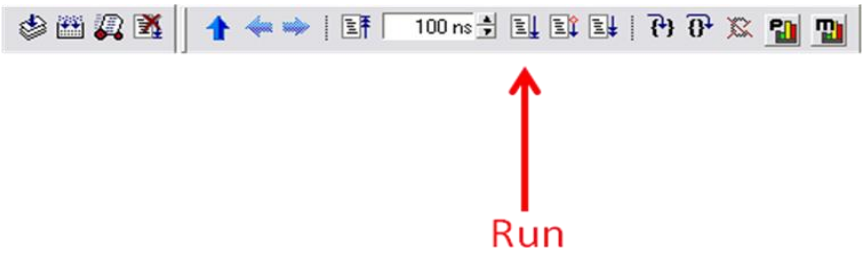

ModelSim SE PLUS 6.2b
File
Edit
View
Compile
Simulate
Add
Objects
Tools
Layout
Window
Coverage
Dataflow
List
Locals
Message Viewer
Objects
Process
Profiling
Call Stack
Watch
Wave
Workspace
New Window
Sort
Filter
Justify
Encoding
Properties...
Workspace
Instance
Design unit
Design unit type
and_gate(m...
Architecture
and_gate(m...
Process
std_logic_1...
Package
standard
Package
Objects
Name
in1
in2
out1



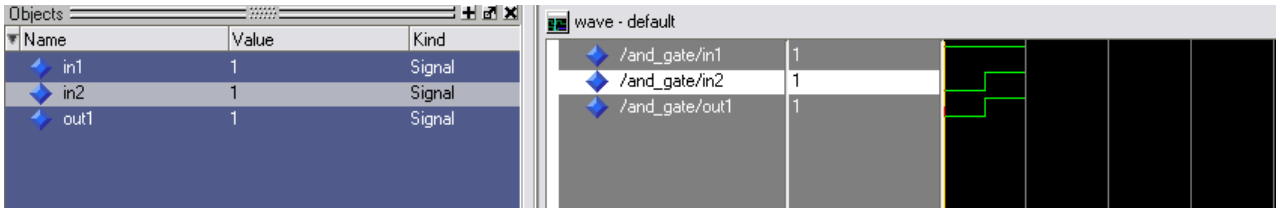

Βάζουμε στο in1 τιμή 1:



Επαναλαμβάνουμε το προηγούμενο βήμα για το in2 βάζοντας την τιμή 0. Έτσι, παρατηρούμε το παρακάτω:

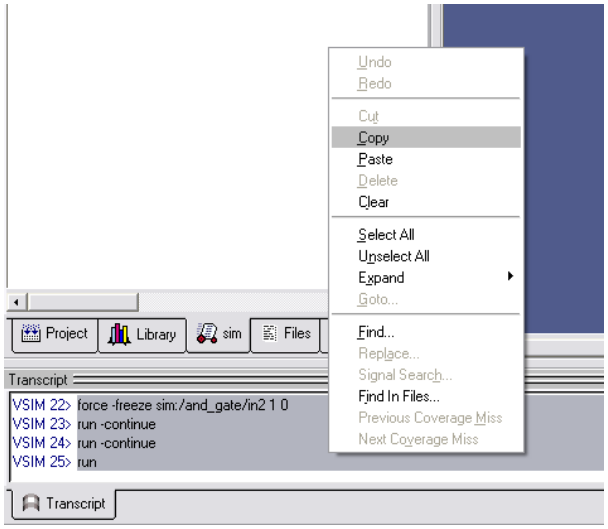




Μετά το πρώτο RUN κάνουμε force το signal In2 σε 1 και με το επόμενο RUN παρατηρούμε το αποτέλεσμα στο out1.

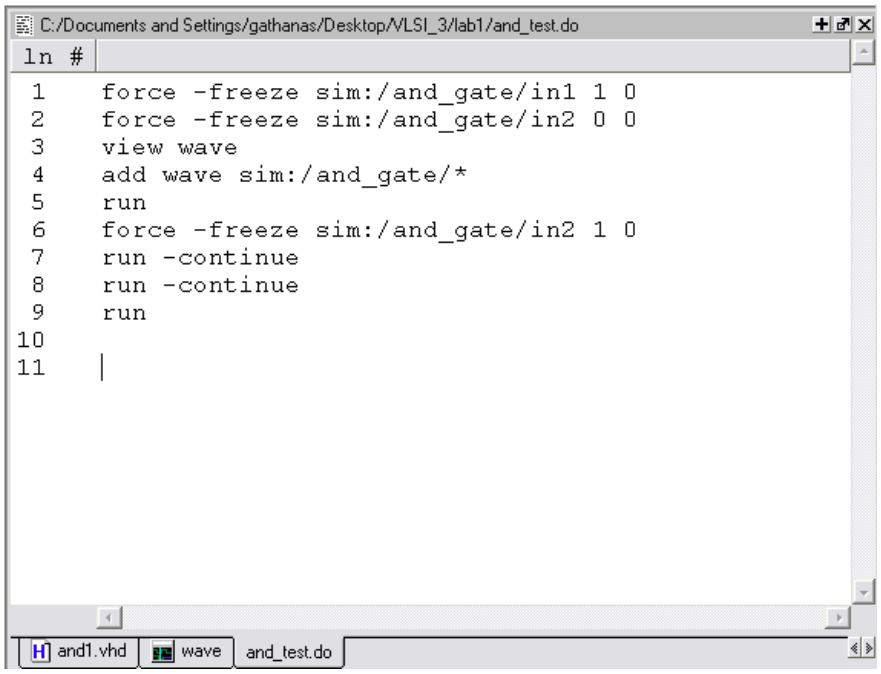



## 1.5 Scripting

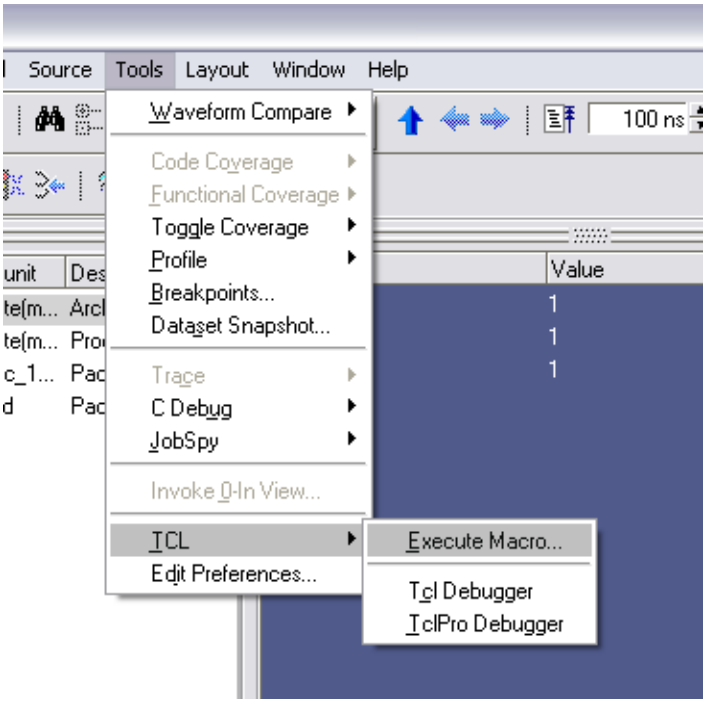
Μέχρι στιγμής, έχουμε παρατηρήσει ότι όλες οι ενέργειες έχουν καταγραφεί στο Transcript Window. Έτσι, αν θέλουμε να αυτοματοποιήσουμε τη διαδικασία αντιγράφουμε τις εντολές σε ένα **.do** αρχείο.



Στη συνέχεια ακολουθούμε την ροή:
File → New → Source → Do
Και στο αρχείο που δημιουργείται κάνουμε paste τις εντολές.




Τέλος, εκτελούμε το script με τον παρακάτω τρόπο:

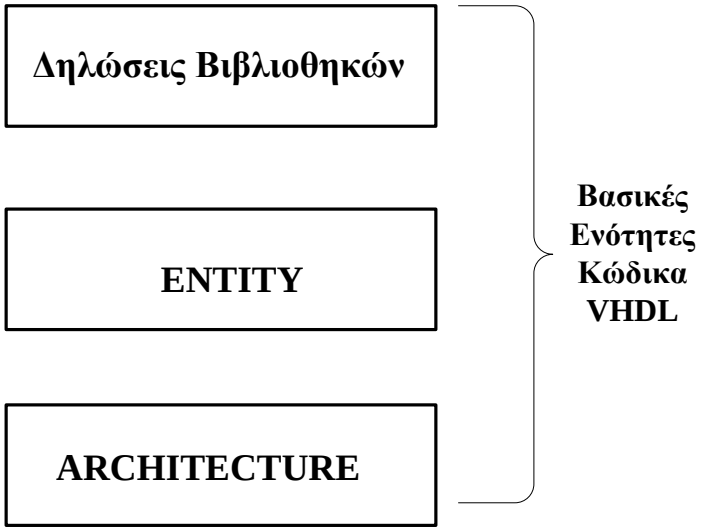


# 2. VHDL - Βασικά χαρακτηριστικά

## 2.1 Ιεραρχική Ροή Σχεδίασης – Δομή κώδικα

Η περιγραφή του κυκλώματος που σχεδιάζεται γίνεται σε διαφορετικά ιεραρχικά επίπεδα. Έτσι, ένας ιεραρχικός σχεδιασμός αποτελείται από υπομονάδες, που εμπεριέχουν άλλες υπομονάδες, VHDL κώδικες ή συνδυασμούς αυτών. Με αυτόν τον τρόπο επιτυγχάνεται η μείωση της πολυπλοκότητας και η ευκολότερη διαχείριση του σχεδιασμού. Ως εκ τούτου, έχουμε γρηγορότερη, οργανωμένη και αποτελεσματικότερη σχεδίαση πολύπλοκων κυκλωμάτων.

Ένας κώδικας VHDL αποτελείται από 3 βασικές ενότητες: τις Βιβλιοθήκες, το Entity και την Architecture.




### 2.1.1 Βιβλιοθήκες

Η δήλωση των βιβλιοθηκών γίνεται με τον ακόλουθο τρόπο:

- **LIBRARY** <library name >;
- **USE** <library name >. <package name>. <package parts>;

Συνήθως απαιτούνται τουλάχιστον τρία πακέτα από τρείς βιβλιοθήκες: *ieee.std_logic_1164(lib.ieee), standard(lib.std), work.*

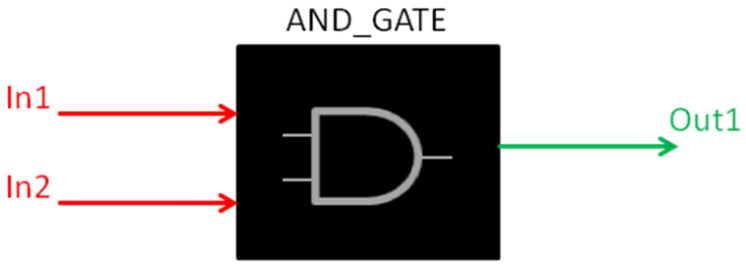
Η βιβλιοθήκη ieee πρέπει ΠΑΝΤΑ να δηλώνεται. Οι άλλες δύο είναι πάντα ορατές και δεν χρειάζεται. Παράδειγμα σύνταξης της ieee:

- **LIBRARY** ieee;
- **USE** ieee.std_logic_1164.all;

### 2.1.2 Entity

Στην entity προσδιορίζουμε τα i/o του κυκλώματος. Όπως φαίνεται παρακάτω, θεωρούμε το κύκλωμα σαν μαύρο κουτί και δεν μας ενδιαφέρει το εσωτερικό του.

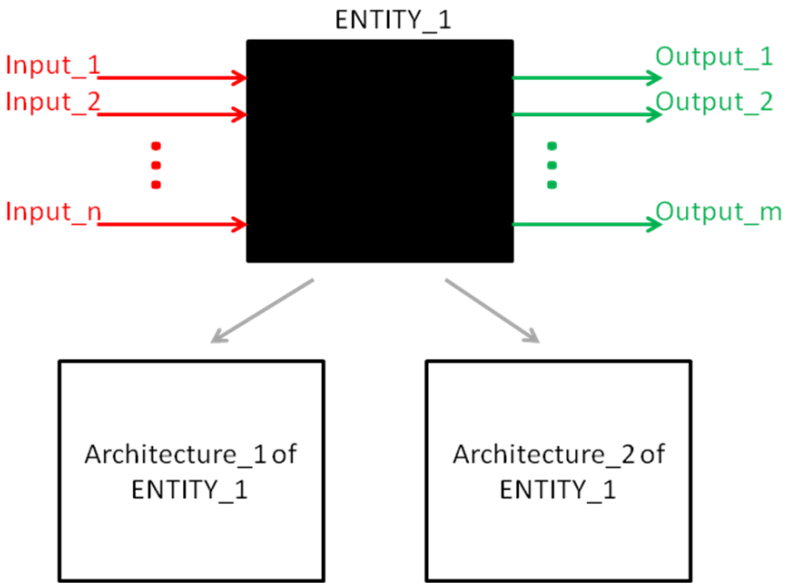

Για το παράδειγμα της πύλης AND που έχει παρουσιαστέι παραπάνω, τα i/o φαίνεται στο σχήμα:




```
entity AND_GATE is
    PORT (In1,In2: in std_logic;
          Out1: out std_logic);
end AND_GATE;
```

### 2.1.3 Architecture

Σε αντίθεση με την entity, στην architecture περιγράφουμε τη λειτουργικότητα (εσωτερικό) του κυκλώματος. Μπορούμε να έχουμε παραπάνω από μία αρχιτεκτονικές για μία entity.



## 2.2 Κλάσεις – Αντικείμενα – Τύποι Δεδομένων

Κάθε αντικείμενο ανήκει σε μία κλάση και έχει έναν τύπο δεδομένων. Οι κλάσεις είναι: SIGNALS, VARIABLES και CONSTANTS. Στο εργαστήριο μας θα ασχοληθούμε μόνο με SIGNALS.

Οι προκαθορισμένοι τύποι δεδομένων που θα μας απασχολήσουν είναι:

- Bit και bit_vector (package: *standard* – lib*: std*)
- Std_logic και Std_logic_vector (pack. *std_logic_1164* – lib.*ieee)*
- Boolean (package: *standard* – lib*: std*)

Ο τύπος bit και bit_vector λαμβάνουν δύο δυνατές τιμές: '**0**' και '**1**'. Παραδείγματα:

```
signal x : bit
signal y : bit_vector (3 downto 0);
signal w : bit_vector (0 to 3);
…
x <= '1';
y <= "0111";        -- MSB=0  Προσοχή στη φορά δήλωσης του διανύσματος
w <= "1110";        -- MSB=0
```

Ο τύπος std_logic και std_logic_vector λαμβάνουν τις εξής δυνατές τιμές:

```
'X'     -- Forcing unknown (synthesizable unknown)
'0'     -- Forcing  0 (synthesizable '0')
'1'     -- Forcing 1 (synthesizable '1')
'Z'     -- High impendance (synthesizable tri-state)
'W'     -- Weak unknown
'L'     -- Weak 0
'H'     -- Weak 1
'-'     -- don't care
```

Οι 4 τελευταίες είναι μη συνθέσιμες και καλό είναι να μην χρησιμοποιούνται. Για τις ανάγκες του παρόντος εργαστηρίου θα χρησιμοποιούμε σχεδόν πάντα τις τιμές 0 και 1.

Ο τύπος Boolean έχει δύο δυνατές τιμές: **TRUE** και **FALSE.**

Η ανάθεση τιμής σε ένα διάνυσμα τύπου bit ή std_logic γίνεται με τους εξής τρόπους:
Έστω το διάνυσμα a, τ.ω. να ισχύει: signal a : std_logic_vector (2 **downto** 0);
Τότε, η ανάθεση μπορεί να γίνει:

```
a <= "10100000";

a <= (7=>'1', 6=>'0', 0=>'0', 1=>'0', 5=>'1', 4=>'0', 3=>'0', 2=>'1');

a <= (7|5=>'1', 6|4|3|2|1|0=>'0');

a <= (7|5=>'1', others=>'0');

a <= "00000000"

a <= (others=>'0');
```

Γενικά, η VHDL βάζει περιορισμούς στην εμβέλεια του κάθε αντικειμένου, η οποία καθορίζεται από το τμήμα του κώδικα που αυτό δηλώνεται. Με άλλα λόγια, όσον αφορά την εμβέλεια ενός αντικειμένου ισχύουν τα εξής:

- Αντικείμενα που δηλώνονται σε πακέτο (package) είναι ορατά από όλες τις οντότητες που χρησιμοποιούνται το πακέτο
- Αντικείμενα που δηλώνονται σε ENTITY είναι ορατά από όλες τις αρχιτεκτονικές που ανήκουν στην ENTITY
- Αντικείμενα που δηλώνονται σε μια ARCHITECTURE είναι ορατά μόνο εντός της ARCHITECTURE
- Αντικείμενα που δηλώνονται σε μια PROCESS είναι ορατά μόνο εντός της PROCESS

## 2.3 Τελεστές και Ιδιότητες

Η VHDL περιέχει διάφορους τύπους προκαθορισμένων τελεστών, που είναι:

- Τελεστές ανάθεσης
- Λογικοί τελεστές
- Αριθμητικοί τελεστές
- Τελεστές ολίσθησης
- Τελεστές συνένωσης

Πιο αναλυτικά:

- **Τελεστές ανάθεσης:** Χρησιμοποιούνται για την ανάθεση (εκχώρηση) τιμής σε SIGNALS, VARIABLES και CONSTANTS
  - '<=' για την ανάθεση τιμής σε SIGNALS
  - ':=' για την ανάθεση τιμής σε VARIABLE και CONSTANTS

```
signal x : std_logic;

variable y : std_logic_vector (3 downto 0);

…

x <='1';

y:= "0010";
```

- **Λογικοί τελεστές:** Είναι οι
  - NOT
  - AND
  - OR
  - NAND
  - NOR
  - XOR
  - XNOR
  - (η παραπάνω σειρά καθορίζει τις προτεραιότητες)

Τα δεδομένα πρέπει να είναι bit ή std_logic (και οι vector επεκτάσεις τους)

```
y<= not A nand b;
x<= a or b;
```

- Αριθμητικοί τελεστές: +, -, *, /, **, mod, rem, abs
  Τα δεδομένα πρέπει να είναι τύπου integer, signed, unsigned ή real
  - Αν χρησιμοποιηθεί το πακέτο *std_logic_unsigned* τότε επιτρέπεται και ο τύπος δεδομένων std_logic_vector

  Οι τελεστές +, - και * είναι πάντα συνθέσιμοι. Για τους υπόλοιπους εξαρτάται από το εργαλείο σύνθεσης.

- Τελεστές σύγκρισης: =, /=, <, >, <=, >=
  - Οι τελεστέοι πρέπει να είναι ίδιου (οποιουδήποτε) τύπου
  - Επιστέφουν Boolean (true, false) τιμή

- Τελετές ολίσθησης: sll, slr, sla, sra, rol ,ror

  sll -- *Λογική ολίσθηση αριστερά οι θέσεις στα δεξιά συμπληρώνονται με '0'*

  srl -- *Λογική ολίσθηση δεξιά οι θέσεις στα αριστερά συμπληρώνονται με '0'*

  sla -- *Αριθμητική ολίσθηση αριστερά το δεξ. bit επαναλαμβάνεται στα δεξιά*

  sra - *-Αριθμητική ολίσθηση δεξιά – το αριστ.bit επαναλαμβάνεται στα αριστερά*

  rol -- *Λογική αριστερή περιστροφή*

  ror -- *Λογική δεξιά περιστροφή*

- Το σύμβολο **&** χρησιμοποιείται για ενοποίηση διανυσμάτων

## 2.4 Concurrent VHDL

Η λειτουργία του υλικού είναι «*παράλληλη*» από τη φύση της. Έτσι, κάθε φορά που αλλάζει το δυναμικό των ηλεκτρικών σημάτων αλλάζουν **ταυτόχρονα** και οι λογικές τιμές αυτών. Συνεπώς, υπάρχει ανάγκη για κατάλληλες δομές που να μοντελοποιούν την παράλληλη αυτή λειτουργία του υλικού. Τέτοιες δομές είναι τα concurrent statements και τα concurrent objects (signals).

### 2.4.1 Signals

Τα signals στη VHDL μοντελοποιούν ηλεκτρικές συνδέσεις (καλώδια). Ο τρόπος σύνταξης στον κώδικα είναι ο εξής:

**Signal Assignment**:

<target_identifier> <= <expression> ;

**Signal Assignment with delay**:

<target_identifier> <= <expression> after 10 ns;

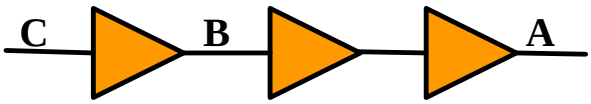
**Οι αναθέσεις τιμών με χρονική καθυστέρηση (after) δεν είναι συνθέσιμες και ΔΕΝ ΘΑ ΜΑΣ ΑΠΑΣΧΟΛΗΣΟΥΝ ΣΤΟ VLSI II**

Οι συντρέχουσες (concurrent) δηλώσεις ενεργοποιούνται από συμβάντα (event-driven) και εκτελούνται ταυτόχρονα, ανεξάρτητα από τη σειρά εμφάνισης στον κώδικα VHDL. Για παράδειγμα οι κώδικες:

```
Architecture example of ex is
begin
    a <= b;
    b <= c;
end example;
```

```
Architecture example of ex is
begin
    b <= c;
    a <= b;
end example;
```

παράγουν το ίδιο κύκλωμα:




### 2.4.2 Concurrent Statements

- Η εντολή ***WHEN***

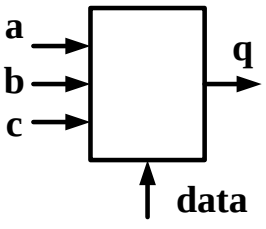
  **Σύνταξη:**

  <target> <= <expres.> [**after** <expres.> ] **when** <expres.> **else**
  <expres.> [**after** <expres.> ] … ;

  - Επιτρέπεται η χρήση περισσότερων του ενός SIGNALS στην συνθήκη
    - Προσδίδει μεγαλύτερη ευελιξία και χρησιμότητα
  - ΠΡΟΣΟΧΗ!!! Η σειρά με την οποία εμφανίζονται οι συνθήκες στον κώδικα είναι σημαντική
    - Ο κώδικας εντός της WHEN είναι ακολουθιακός => Η σειρά εμφάνισης των συνθηκών είναι σημαντική
    - Οι εκφράσεις εξετάζονται ακολουθιακά από πάνω προς τα κάτω και μόλις μία είναι TRUE η εκτέλεση της εντολής τερματίζεται.
    - Έχει ως συνέπεια τη δημιουργία ενός δένδρου από πολυπλέκτες για την τήρηση των προτεραιοτήτων
    - Τα ίδια ισχύουν και για την εντολή WITH

  Παράδειγμα:

```
Architecture rtl of ex is
begin
q <= a when data =“00” else
       b when data =“11” else
       c;
end;
```



- Η εντολή ***WITH***

**Σύνταξη:**

<**with**> <expression> **select**
<target> <= <expression> **when** <chose>;

- Όλες οι καταστάσεις του σήματος που οδηγείται πρέπει να απαριθμούνται
  - Χρήση when others για τις υπόλοιπες περιπτώσεις
- Λιγότερο ευέλικτη σε σύγκριση με τη δήλωση when
  - Η δήλωση with επιτρέπει μόνο μια έκφραση (expression)

Παράδειγμα:

```
entity example is
port ( a,b,c : in  std_logic;
  data  : in  std_logic_vector (1 downto 0);
  q     : out std_logic);
end example;
```

```
architecture rtl of example is
begin
with data select
q <= a when "00",
     b when "11",
     c when others;
end;
```

Ένας πολυπλέκτης μπορεί να υλοποιηθεί με τη βοήθεια τελεστών ή με χρήση των παραπάνω εντολών:

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
    port ( a, b, c, d: in std_logic;
           sel: in std_logic_vector (1 downto 0);
           y: out std_logic);
end mux;
```

```
architecture operators of mux is
begin
  y <= (a and not sel(1) and not sel(0)) or
    (b and not sel(1) and sel(0)) or
    (c and sel(1) and not sel(0)) or
    (d and sel(1) and sel(0));
end operators ;
```

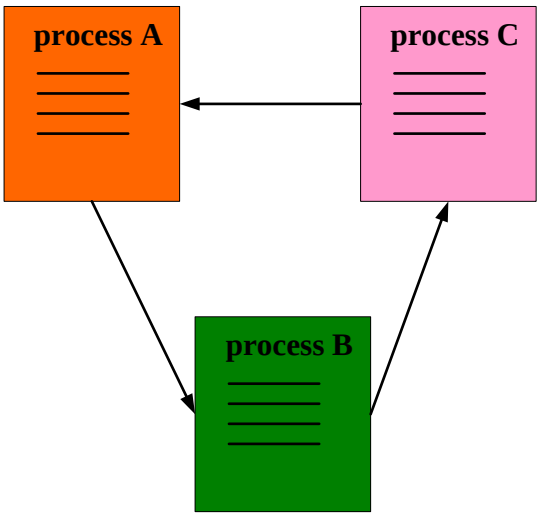
```
architecture mux1 of mux is
begin
  y <= a when sel="00" else
       b when sel="01" else
       c when sel="10" else
       d;
end mux1;
```

```
architecture mux2 of mux is
begin
  with sel select
    y <= a when "00",
         b when "01",
         c when "10",
         d when others;

--ΠΡΟΣΟΧΗ: Χρήση «,» αντί για «;». Επίσης δεν μπορεί να είναι « "d when "11"».
```

## 2.5 Processes

Η διεργασία (Process) είναι θεμελιώδης έννοια της VHDL και προέρχεται από το συμβατικό λογισμικό. Αντιστοιχεί σε ακολουθιακή εκτέλεση εντολών, η οποία συντελείται μετά από διέγερση και όταν ολοκληρωθεί επιστρέφει σε κατάσταση αναμονής.

Περισσότερες από μία διεργασίες μπορούν να εκτελούνται ταυτόχρονα και να διεγείρονται από συντρέχουσες δομές (σήματα). Πάντα, όμως, εντός της διεργασίας ο κώδικας εκτελείται ακολουθιακά.




Σύνταξη:

[<process_name> :] **process** [ (sensitivity_list)]

[<process_declarative_part>]

**begin**

<process_statement_part>

**end process** [<process_name>];

Η διεργασία θα διεγείρεται μέσω αλλαγής κάποιου σήματος είτε εντός του sensitivity list είτε μέσω κάποιου wait statement:

```
ff : process (a,b)  -- sensitivity list
begin
    q <= a;
    z <= b;
end ;
```

```
cc : process
begin
    wait on a, b; -- wait statement
    q <= a;
    z <= b;
end;
```

**ΑΠΑΓΟΡΕΥΕΤΑΙ Η ΤΑΥΤΟΧΡΟΝΗ ΧΡΗΣΗ WAIT ΚΑΙ SENSITIVITY LIST!!**

Κάθε process είναι ένα concurrent statement → **έχει ένα σήμα εξόδου!**

Ιδιαίτερη προσοχή πρέπει να δίνεται ώστε να μην δημιουργηθεί ατέρμων κύκλος (infinite loop), όπως για παράδειγμα στην παρακάτω περίπτωση, όπου η διεργασία συνεχίζει να εκτελείται επ' άπειρον, αφού δεν υπάρχει statement που να την «ελέγχει»:

```
process
begin
 a <= '1';
end process;
```

Γενικοί κανόνες:

- **ΟΛΑ ΤΑ ΣΗΜΑΤΑ ΕΙΣΟΔΟΥ ΘΑ ΠΡΕΠΕΙ ΝΑ ΠΕΡΙΛΑΜΒΑΝΟΝΤΑΙ ΣΤΗΝ SENSITIVITY LIST!**
- **ΟΛΟΙ ΟΙ ΣΥΝΔΥΑΣΜΟΙ ΕΙΣΟΔΩΝ/ΕΞΟΔΩΝ ΠΡΕΠΕΙ ΝΑΠΕΡΙΛΑΜΒΑΝΟΝΤΑΙ ΣΤΟΝ ΚΩΔΙΚΑ ΤΗΣ ΔΙΕΡΓΑΣΙΑΣ (Πρέπει να εξάγεται πλήρης πίνακας αληθείας)**

**Σύγχρονα Ακολουθιακά Κυκλώματα:**

Μια βασική χρήση των processes, εκτός των παραπάνω, είναι στην περιγραφή σύγχρονων ακολουθιακών κυκλωμάτων. Σε αυτήν την περίπτωση, για τη διέγερση τους απαιτείται αλλαγή του σήματος του ρολογιού. Έτσι επιτυγχάνεται η αλλαγή κατάστασης του κυκλώματος μόνο στις χρονικές στιγμές αλλαγής ρολογιού. Η σύνταξη της παραπάνω χρήσης είναι:

```
Alt 1:process(clk)
    begin
      if clk'event and clk='1' then
      q<=d;
     end if;
    end process;
```

Κύκλωμα πυροδοτούμενο στην θετική ακμή του ρολογιού και μόνο! (Positive-edge triggered - Χρησιμοποιείται για δημιουργία registers)

```
Alt 2:process(clk)
    begin
      if clk'event then
      q<=d;
     end if;
    end process;
```

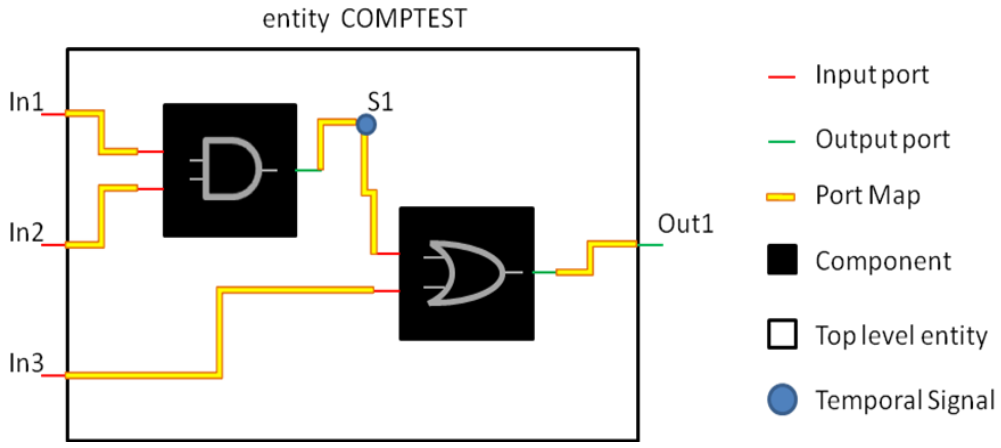
Κύκλωμα πυροδοτούμενο σε ακμή ρολογιού, είτε θετική είτε αρνητική (Edge triggered).

```
Alt 3:process(clk)
    begin
      if clk='1' then – ή '0'
      q<=d;
     end if;
    end process;
```

Κύκλωμα πυροδοτούμενο σε επίπεδο ρολογιού (1 ή 0 ανάλογα την επιλογή – Level triggered)

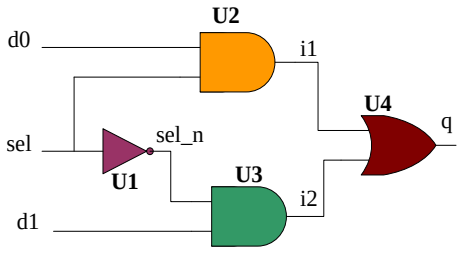
## 2.6 Structural VHDL

Κατά το δομικό (structural) τρόπο σχεδίασης, γίνεται περιγραφή των διασυνδέσεων των υπομονάδων του κυκλώματος. Δεν απαιτείται περιγραφή της λειτουργία της κάθε υπομονάδας συνεχώς, με την προυπόθεση ότι υπάρχει βιβλιοθήκη στην οποία ορίζονται όλες οι χρησιμοποιούμενες υπομονάδες ή έχουν ορισθεί στην αρχή του κώδικα μία φορά. Σχηματικά, ισχύει το εξής:




Ο κώδικας VHDL με χρήση δομικής σχεδίασης έχει την μορφή:

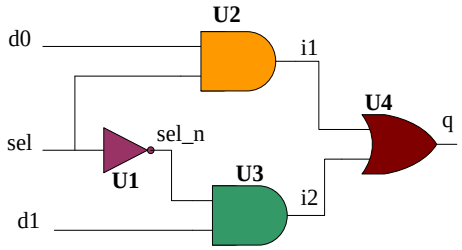
```
Entity mux is
port (d0,d1,sel : in  std_logic;
          q    : out std_logic);
end;
architecture str_mux of mux is

-- Component Declaration
component and_comp

port (  a,b : in   std_logic;
          c    : out  std_logic);
end component;

component inv_comp
port (  a   : in    std_logic;
          b   : out  std_logic);
end component;

component or_comp
port (  a,b : in   std_logic;
          c    : out  std_logic);
end component;

-- Internal signals declaration
signal i1, i2, sel_n: std_logic
```



```
-- Comp Specification
for U1 : inv_comp use entity work.inv_comp(rtl);
for U2, U3: and_comp use entity work.and_comp(rtl);
for  U4: or_comp use entity work.or_comp(rtl);
```

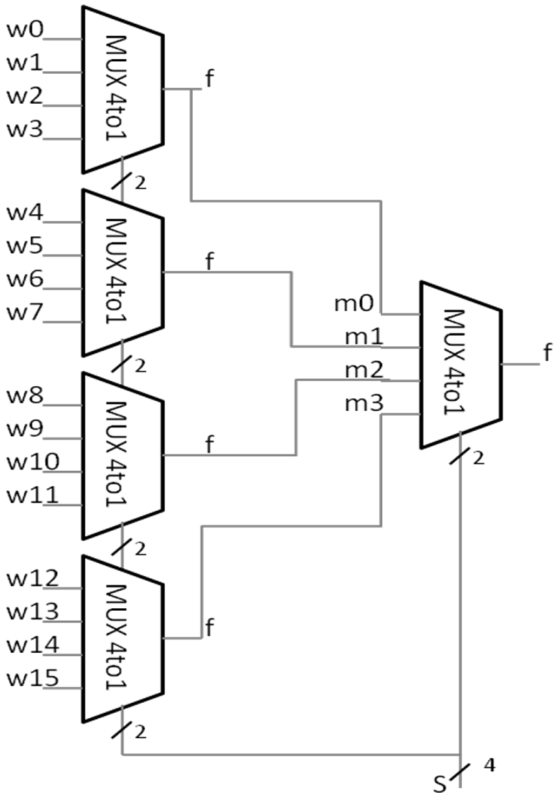
← Αν έχουν περιγραφεί τα components σε βιβλιοθήκες, τότε είναι απαραίτητες. Αλλιώς, αν έχουν περιγραφεί πιο πάνω στον κώδικα ή σε άλλο αρχείο vhdl εντός του project τότε δεν χρειάζονται.

```
begin
-- Component Instantiation
U1: inv_comp port map (sel, sel_n);
U2: AND_comp port map (d0, sel, i1);
U3: AND_comp port map (sel_n, d1,i2);
U4: OR_comp port map (i1,i2,q);
end;
```




- **<u>Η port_list πρέπει να είναι ίδια με αυτή της οντότητας της υπομονάδας</u>**
  - **Για αποφυγή λαθών προτείνεται η αντιγραφή της οντότητας της υπομονάδας κατά τη δήλωση αυτής**

Για τη δημιουργία εύκολη διαχείριση μεγάλων κυκλωμάτων και συστημάτων, μαζί με τις παραπάνω δομικές αρχές σχεδιασμού, γίνεται χρήση και των παρακάτω δύο εντολών (η πρώτη για παραμετροποίηση και η δεύτερη για εύκολη δημιουργία πολλαπλών στιγμιότυπων ενός component):
Για παράδειγμα, το παρακάτω κύκλωμα πολυπλεκτών:



περιγράφεται δομικά, ως εξής:

```
library ieee;
    use ieee.std_logic_1164.all;

    entity mux16to1 is
        port(w: in std_logic_vector(15 downto 0);
             s: in std_logic_vector(3 downto 0);
             f: out std_logic);
    end mux16to1;

    architecture my_arch of mux16to1 is

        component mux4to1
        port(w0, w1, w2, w3: in std_logic;
             s: in std_logic_vector(1 downto 0);
             f: out std_logic);
        end component;

        signal m: std_logic_vector(3 downto 0);

        begin
             mux_1: mux4to1 port map (w(0), w(1), w(2),
                                      w(3), s(1 downto 0), m(0));
             mux_2: mux4to1 port map (w(4), w(5), w(6),
                                      w(7), s(1 downto 0), m(1));
             mux_3: mux4to1 port map (w(8), w(9), w(10),
                                      w(11), s(1 downto 0), m(2));
             mux_4: mux4to1 port map (w(12), w(13), w(14),
                                      w(15), s(1 downto 0), m(3));
            mux5: mux4to1 port map (m(0), m(1), m(2), m(3), s(3 downto 2), f);
    end my_arch;
```

Το component mux4to1 μπορεί να περιγραφεί με κώδικα VHDL είτε χρησιμοποιώντας την εντολή WITH, είτε την εντολή WHEN. Η περιγραφή με εντολή WITH παρουσιάζεται παρακάτω:

```
1  -- Lab 2
2  -- MUX4to1
3  -- Inputs: w0, w1, w2, w3, s
4  -- Outputs: f
5
6  library ieee;
7      use ieee.std_logic_1164.all;
8
9      entity mux4to1 is
10         port(w0, w1, w2, w3: in std_logic;
11              s: in std_logic_vector(1 downto 0);
12              f: out std_logic);
13     end mux4to1;
14
15     architecture my_mux4to1 of mux4to1 is
16         begin
17             with s select
18                 f <=  w0 when "00",
19                       w1 when "01",
20                       w2 when "10",
21                       w3 when OTHERS;
22     end my_mux4to1;
23
```

Η παραπάνω περιγραφή μπορεί να γίνει σε ξεχωριστό αρχείο vhd, το οποίο θα βρίσκεται στον ίδιο φάκελο του project που βρίσκεται και το αρχείο του πολυπλέκτη 16 σε 1. Εναλλακτικά, μπορεί να προστεθεί στο ίδιο αρχείο vhd με τον πολυπλέκτη 16 σε 1.

- Η εντολή ***GENERIC***

Με την εντολή GENERIC μπορούμε να δηλώσουμε «σταθερές», ώστε να παραμετροποιείται εύκολα το κύκλωμα, δηλαδή, να αλλάζει τα μεγέθη εντός του εύκολα. Τα GENERICS μοιάζουν πολύ με τα ports. Ορίζονται εντός της entity και μπορούν να γίνουν map με την εντολή GENERIC MAP. Παράδειγμα σύνταξης:

```
library ieee;
use ieee.std_logic_1164.all;

entity latch_n is
    generic (n :integer :=8);
    port (clock, reset: in std_logic;
        input1: in std_logic_vector(n-1 downto 0);
        output1: out std_logic_vector(n-1 downto 0));
    end latch_n;

architecture my_latch_n of latch_n is
    signal temp: std_logic_vector(n-1 downto 0);
    begin
        process (clock, reset, input1)
            begin
                if reset='1' then
                    temp<=(others=>'0');
                else
                    if clock='1' and clock'event then
                        temp<=input1;
                    end if;
                end if;
        end process;
        output1<=temp;
end my_latch_n;
```

Default value, αν δεν πάρει άλλη τιμή από generic map υψηλότερου επιπέδου

Υπάρχουν περιπτώσεις χρήσης των Generics σε ιεραρχία με περισσότερα του ενός components. Για παράδειγμα, στον παρακάτω κώδικα, υπάρχουν δύο generics. Το *m* με default value 32 και το *n* στο component με default value 8. Με τη χρήση του generic map, εξασφαλίζουμε ότι το *n* θα παίρνει πάντα την τιμή του *m*. Δηλαδή, αν από υψηλότερο επίπεδο δεν προσδιοριστεί το *m* (πράγμα που ισχύει και στο παρακάτω παράδειγμα) θα πάρει και το *m* και το *n* την τιμή 32.

```
library ieee;
use ieee.std_logic_1164.all;

entity gen_map_test is
    generic (m :integer :=32);
    port (clock, reset: in std_logic;
        in1: in std_logic_vector(m-1 downto 0);
        out1: out std_logic_vector(m-1 downto 0));
    end gen_map_test;

architecture my_arch of gen_map_test is

    component latch_n
        generic (n :integer :=8);
        port(clock, reset: in std_logic;
        input1: in std_logic_vector(n-1 downto 0);
        output1: out std_logic_vector(n-1 downto 0));
    end component;

    begin
        L1: latch_n generic map(m) port map(clock, reset, in1, out1);

   end my_arch;
```

- Η εντολή ***FOR GENERATE***

Η εντολή αυτή παρέχει έναν εύκολο τρόπο επανάληψης μίας λογικής εξίσωσης ενός port map ή της δημιουργίας του στιγμιότυπου ενός component. Με την εντολή αυτή μπορούμε να δημιουργήσουμε πολλά port maps με εύκολο τρόπο. Παρόλα αυτά, χρειάζεται ιδιαίτερη προσοχή στην χρήσης της καθώς προυποθέτει την ύπαρξη μεταβλητών για την εισαγωγή του επιθυμητού αριθμού components.

Για παράδειγμα, το κύκλωμα του πολυπλέκτη 16 σε 1 που παρουσιάστηκε παραπάνω, με χρήση FOR GENERATE μπορεί να περιγραφεί ως εξής:

```
library ieee;
    use ieee.std_logic_1164.all;

    entity mux16to1 is
        port(w: in std_logic_vector(15 downto 0);
            s: in std_logic_vector(3 downto 0);
            f: out std_logic);
    end mux16to1;

    architecture my_arch of mux16to1 is

        component mux4to1
        port(w0, w1, w2, w3: in std_logic;
            s: in std_logic_vector(1 downto 0);
            f: out std_logic);
        end component;

        signal m: std_logic_vector(3 downto 0);

        begin
            generate_label:
            for i in 0 to 3 generate
                 mux_i: mux4to1 port map (w(4*i), w(4*i+1), w(4*i+2),
                              w(4*i+3), s(1 downto 0), m(i));
            end generate;

            mux5: mux4to1 port map (m(0), m(1), m(2), m(3), s(3 downto 2), f);
    end my_arch;
```

**Βιβλιογραφία – Χρήσιμα Links για VHDL και Modelsim :**


1. **CMOS VLSI Design, A Circuits and Systems Perspective – Third Edition.** Neil H.E. Weste, David Harris. *Addison Wesley Publications*
2. **The Designer's Guide to VHDL.** Peter J. Asbenden. University of Adelaide. *Morgan Kaufman Publishers, Inc., California*
3. **Circuit Design with VHDL.** Volnei A. Pedroni. *MIT Press. Cambridge, Massachusetts - London, England*
4. **Σχεδίαση Ψηφιακών Συστημάτων με τη Γλώσσα VHDL.** S. Brown, Z. Vranesic. *Εκδόσεις Τζιόλα, Θεσσαλονίκη*
5. **Modelsim_Tutorial:** **http://pages.cs.wisc.edu/~markhill/cs552/Fall2006/handouts/se_tutor.pdf**
6. **VHDL tutorial: http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html**
7. **MIT Course: http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-111Spring2004/LectureNotes/index.htm**