Σχεδίαση Υπολογιστικών Συστημάτων

Μοντελοποίηση και προσομοίωση στην VHDL

Μιχάλης Ψαράκης

Αντικείμενα στη VHDL

- Σταθερές (constants)
- Μεταβλητές (variables)
- Σήματα (signals)
- Θύρες (ports)

Δηλώσεις σταθερών

- Οι σταθερές αποδίδουν όνομα και ρητά καθορισμένο τύπο σε μια τιμή
- Η χρήση σταθερών αποτελεί σωστή σχεδιαστική πρακτική
- Οι σταθερές μπορούν να δηλωθούν σε διαφορετικά μέρη ενός VHDL μοντέλου
- Παραδείγματα:

```
constant number_of_bytes : integer := 4;
constant number_of_bits : integer := 8 * number_of_bytes;
```


Δηλώσεις μεταβλητών

- Οι μεταβλητές δεν μπορεί να δηλωθούν σε σημείο του VHDL μοντέλου ώστε να είναι προσπελάσιμες από περισσότερες από μια διεργασίες
  - Εξαίρεση: κοινόχρηστες μεταβλητές (shared variables).
    - Δεν θα ασχοληθούμε με κοινόχρηστες μεταβλητές
  - Θα δηλώνουμε μεταβλητές μόνο στο τμήμα δήλωσης των διεργασιών
    - Ορατές μόνο εντός της διεργασίας
- Η αρχικοποίηση της μεταβλητής είναι προαιρετική
  - Εάν παραληφθεί, η εξ'ορισμού (default) αρχική τιμή είναι η αριστερότερη τιμή του τύπου

```
variable index : integer := 0;
```

Ανάθεση μεταβλητής

- Παραδείγματα:

```
program_counter := 0;
index := index + 1;
```

- Προσοχή! Υπάρχει διαφορά μεταξύ της ανάθεσης μεταβλητής (**:=**) και της ανάθεσης σήματος (**<=**)
- Ανάθεση μεταβλητής (variable assignment):
  - ενημερώνει αμέσως τη μεταβλητή με τη νέα τιμή
- Ανάθεση σήματος (signal assignment):
  - χρονοπρογραμματίζει την εφαρμογή της νέας τιμής στο σήμα σε κάποια μελλοντική χρονική στιγμή

Ανάθεση σήματος (signal assignment)

- Καθορισμός χρόνου καθυστέρησης (προαιρετικός)

```
y <= not or_a_b after 5 ns;
```

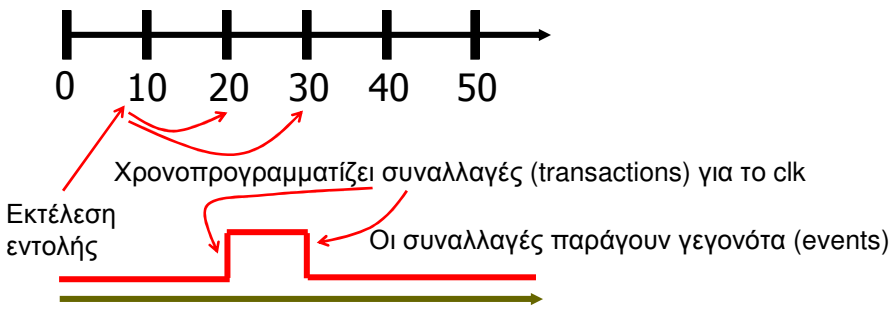
- Μοντελοποιεί ζητήματα χρονισμού (timing issues)
  - Προδιαγράφει την καθυστέρηση διάδοσης (propagation delay) της λειτουργικής μονάδας
  - Στην προσομοίωση, προδιαγράφει πότε θα ανατεθεί η νέα τιμή στο σήμα
- Εάν παραληφθεί ο χρόνος καθυστέρησης, ο προκαθορισμένος χρόνος είναι 0 fs (delta delay)

Ανάθεση σήματος

- Παράδειγμα: παλμός πλάτους T_pw

```
clk <= '1' after T_pw, '0' after 2*T_pw;
```

- Εάν T_pw = 10ns, αρχική τιμή του clk = '0' και η πρόταση εκτελεστεί σε χρόνο 10 ns



Σήματα (signals) & θύρες (ports)

- Τα σήματα μεταφέρουν δεδομένα εντός της αρχιτεκτονικής
- Οι θύρες αποτελούν τα σήματα επικοινωνίας της οντότητας με τον «έξω κόσμο»

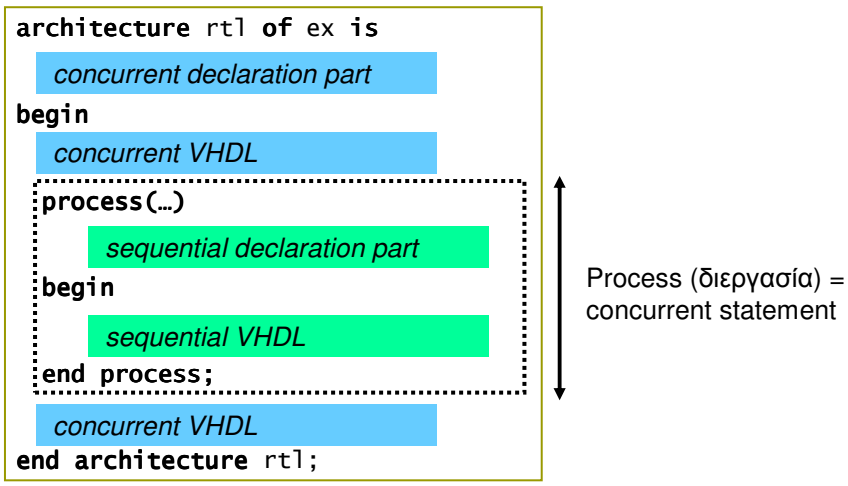
```
entity circuit is
  port (port specification);
end entity circuit;

architecture arch of circuit is
signal declaration
begin
...
end architecture arch;
```

Παράλληλο πεδίο (concurrent domain) και ακολουθιακό πεδίο (sequential domain)

- Οι εντολές της γλώσσας που βρίσκονται στο παράλληλο πεδίο εκτελούνται παράλληλα
- Οι εντολές της γλώσσας που βρίσκονται στο ακολουθιακό πεδίο εκτελούνται με την σειρά
  - όπως στις γλώσσες προγραμματισμού
  - το πιο ισχυρό κομμάτι της γλώσσας
- Οι όροι παράλληλη και ακολουθιακή εκτέλεση προτάσεων αφορούν την εξομοίωση
  - το υλικό εκτελεί παράλληλα

Παράλληλη & ακολουθιακή VHDL



Παράλληλη & ακολουθιακή VHDL : δηλώσεις

- Παράλληλη VHDL
  - δήλωση σημάτων
- Ακολουθιακή VHDL
  - δήλωση μεταβλητών
- Παράλληλη/Ακολουθιακή VHDL
  - δήλωση τύπων, σταθερών

Παράλληλη & ακολουθιακή VHDL : εντολές

- Παράλληλη VHDL
  - process
  - component
  - when-else, with-select
- Ακολουθιακή VHDL
  - variable assignment
  - if-then-else
  - case
  - wait
  - loop
- Παράλληλη/Ακολουθιακή VHDL
  - signal assignment
  - assertion, report

Διεργασία (process)

```
process sensitivity list
    declaration part
begin
    statement part
end process;
```

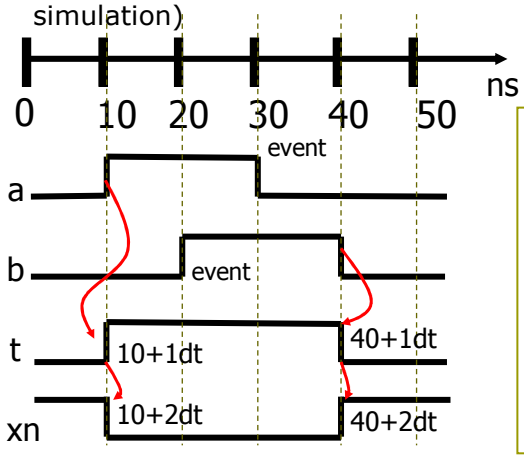
- Λίστα ευαισθησίας (sensitivity list)
  - λίστα σημάτων στα οποία η διεργασία είναι ευαίσθητη
- Τμήμα δηλώσεων (declaration part)
  - τοπικές μεταβλητές, δεν είναι ορατές έξω από τη διεργασία
- Τμήμα προτάσεων (statement part)
  - περιέχει τις ακολουθιακές εντολές

Λίστα ευαισθησίας (sensitivity list)

- Οι διεργασίες εκτελούνται σαν ένας ατέρμονος βρόχος
  - όταν τελειώσει η εκτέλεση των εντολών της διεργασίας, η εξομοίωση αρχίζει από την αρχή
- Η εκτέλεση των εντολών της διεργασίας ξεκινάει όταν συμβεί ένα γεγονός (event) σε ένα από τα σήματα της λίστας ευαισθησίας
  - γεγονός = αλλαγή τιμής
  - όταν τελειώσει η εκτέλεση των εντολών της διεργασίας, η διεργασία «αναστέλλεται» μέχρι να συμβεί ένα γεγονός στην λίστα ευαισθησίας

Παράδειγμα

- Ανάθεση σημάτων (signal assignment)
- Προσομοίωση βασισμένη σε γεγονότα (event-driven VHDL simulation)




```
entity nor2 is
  port (a,b : in bit;
        xn  : out bit);
end entity nor2;
architecture beh of nor2 is
signal t : bit;
begin
  t <= a or b;
  xn <= not t;
end architecture beh;
```


Παράδειγμα

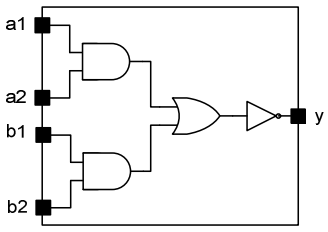
- Διεργασίες (process)
- Ανάθεση σημάτων (signal assignment)

```
entity and_or_inv is
   port ( a1,a2,b1,b2 : in bit := '1';
          y : out bit );
end entity and_or_inv;

architecture prim of and_or_inv is
signal and_a,and_b : bit;
signal or_a_b : bit;

begin
and_gate_a : process (a1, a2) is
begin
and_a <= a1 and a2;
end process and_gate_a;

and_gate_b : process (b1, b2) is
begin
and_b <= b1 and b2;
end process and_gate_b;
-- …cont.
```




```
or_gate :process (and_a,and_b) is
begin
or_a_b <= and_a or and_b;
end process or_gate;

inv : process (or_a_b) is
begin
y <= not or_a_b;
end process inv;

end architecture prim;
```

Παράδειγμα

- Μοντελοποίηση ενός μανταλωτή (latch) SR

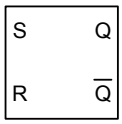
```
entity latch_SR is
  port ( S, R : in bit;
       Q, Qn : out bit );
end entity latch_SR;

architecture prim of latch_SR is

begin
P1: process (R,Qn)
begin
   Q <= R nor Qn;
end process;

P2: process (S,Q)
begin
   Qn <= S nor Q;
end process;

end architecture prim;
```



S	R	Q	Qn
0	0	last Q	last Qn
0	1	0	1
1	0	1	0
1	1	0	0

Σήματα εναντίον μεταβλητών

```
signal sum1, sum2:
integer;
…
p1: process (din)
begin
       sum1 <= din + 1;
       sum2 <= sum1 + 1;
end process;
```

Time	din	Sum1	Sum2
…	0	0	0
t1	1	0	0
t1+1dt	1	2	1
t2	2	2	1
t2+1dt	2	3	3

```
p2: process (din)
variable sum1,sum2:
integer;

begin
       sum1 := din + 1;
       sum2 := sum1 + 1;
end process;
```

Time	din	Sum1	Sum2
…	0	0	0
t1	1	2	3
t1+1dt	1	2	3
t2	2	3	4
t2+1dt	2	3	4

- Τί θα συμβεί αν αλλάξουμε τη σειρά των εντολών μέσα στις διεργασίες p1 και p2;

Ακολουθιακές εντολές

- Εντολές if
- Εντολές case
- Εντολές loop
- Εντολές assert
- Εντολές wait

Εντολές if

Απλή εντολή if

```
if en = '1' then
 stored_value :=
          data_in;
end if;
```

Φράση else

```
if sel = 0 then
  result <= input_0;
else
  result <= input_1;
end if;
```

Πολλαπλές ακολουθ. εντολές

```
if opcode = halt_opcode then
  PC := effective_address;
  executing := false;
  halt_indicator <= true;
end if;
```

Πολλαπλές φράσεις elsif

```
if mode = immediate then
  operand := immed_operand;
elsif opcode = load or
         opcode = add or
           opcode = subtract then
  operand := memory_operand;
else
  operand := address_operand;
end if;
```

Ένθετες (nested) εντολές if

```
if phase = wash then
  if cycle_select = delicate_cycle then
    agitator_speed <= slow;
  else
    agitator_speed <= fast;
  end if;
  agitator_on <= true;
end if;
```

Εντολές case

Μοντέλο ALU (Arithmetic & Logic Unit)

```
-- οι λειτουργίες της ALU
type alu_func is (pass1, pass2, add, subtract);
…
-- το σήμα ελέγχου της ALU
signal func : alu_func;
…
case func is
   when pass1 =>
      result := operand1;
   when pass2 =>
      result := operand2;
   when add =>
      result := operand1 + operand2;
   when subtract =>
      result := operand1 – operand2;
end case;
```


While loop

Μοντέλο της συνάρτησης συνημιτόνου (cosine function) με χρήση της σειράς

```
entity cos is
  port (theta: in real; result: out real;);
end entity;

architecture series of cos is
begin

  P1: process (theta) is
  variable sum, term, n: real;
  begin
    sum := 1.0; term := 1.0; n := 0.0;
    while abs term > abs (sum/1.0E6) loop
      n := n + 2.0;
      term := (-term)*(theta**2)/((n-1)*n);
      sum := sum + term;
      end loop;
    result <= sum;
  end process;

end architecture;
```

$$\cos\theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \ldots$$

Άθροιση διαδοχικών όρων της σειράς έως ότου οι όροι να γίνουν μικρότεροι από το ένα 1/1.0E6 του αποτελέσματος

For loop

Μοντέλο της συνάρτησης συνημιτόνου (cosine function) με χρήση της σειράς

```
architecture fixed_length_series of cos is
begin

  summation : process (theta) is
  variable sum, term : real;
  begin
    sum := 1.0;
    term := 1.0;
    for n in 1 to 9 loop
      term := (-term)*theta**2/real(((2*n-1)*2*n));
      sum := sum + term;
    end loop;
    result <= sum;
  end process summation;

end architecture fixed_length_series;
```

$$\cos\theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \ldots$$

Άθροιση των 10 πρώτων όρων της σειράς

Εντολές assert

- Οι εντολές ισχυρισμού (assert statement) χρησιμοποιούνται για την επαλήθευση του μοντέλου
- Παράδειγμα: **assert** *condition*;
  - Ισχυριζόμαστε ότι η *condition* είναι πάντα αληθής (όποτε εκτελείται η assertion)
  - Εάν δεν είναι αληθής ⇒ παραβίαση ισχυρισμού
    - Ο προσομοιωτής αναφέρει το γεγονός
- Οι εντολές assert χρησιμοποιούνται από:
  - Εργαλεία σύνθεσης (synthesizer)
    - Για βελτιστοποίηση του αποτελέσματος
  - Εργαλεία τυπικής επαλήθευσης (formal verifier)
    - Αποδεικνύουν ότι ισχύει για όλα τα πιθανά ερεθίσματα εισόδου (input stimuli)

Φράσεις report, severity

Φράση αναφοράς (report clause)

```
assert initial_value <= max_value
  report "initial value too large";
```

Για να ξέρουμε ποιος ισχυρισμός παραβιάστηκε

Φράση αυστηρότητας (severity clause)

```
assert initial_value <= max_value
   report "initial value too large"
     severity warning;
```

Πόσο επηρεάζει η παραβίαση του ισχυρισμού το μοντέλο μας;

Οι προσομοιωτές μας επιτρέπουν να καθορίσουμε ένα severity threshold

Πάνω από το threshold σταματούν την προσομοίωση

Default threshold = error

Προκαθορισμένος τύπος απαρίθμησης

```
type severity_level is (note, warning, error, failure);
```


Παράδειγμα: SR latch

Πρόταση ισχυρισμού: Οι είσοδοι S και R δεν πρέπει να είναι και οι δύο 1

```
entity SR_flipflop is
  port (S,R : in bit; Q : out bit );
end entity SR_flipflop;
architecture checking of SR_flipflop is
begin
  set_reset : process (S, R) is
  begin
    assert S = '1' nand R = '1';
    if S = '1' then
      Q <= '1';
    end if;
    if R = '1' then
      Q <= '0';
    end if;
  end process set_reset;
end architecture checking;
```

Η πρόταση assert θα ανιχνεύσει παραβίαση ισχυρισμού εάν το μοντέλο χρησιμοποιηθεί λανθασμένα

Παράδειγμα

```
entity max3 is
   port (a,b,c : in integer; z : out integer );
end entity max3;
architecture check_error of max3 is
begin
  maximizer : process (a, b, c)
  variable result : integer;
  begin
    if a > b then
      if a > c then
        result := a;
      else
        result := a; -- Ουπς! Θα έπρεπε να είναι:
      end if;    --  result := c;
    elsif b > c then
      result := b;
    else
      result := c;
    end if;
    assert result >= a and result >= b and result >= c
      report "inconsistent result for maximum"
        severity failure;
    z <= result;
  end process maximizer;
end architecture check_error;
```

Κύκλωμα επιλογής της μέγιστη τιμής μεταξύ των 3 εισόδων του

Πρόταση ισχυρισμού: Η έξοδος πρέπει να είναι μεγαλύτερη ή ίση και από τις 3 εισόδους

Η πρόταση assert θα ανιχνεύσει παραβίαση ισχυρισμού εάν το μοντέλο δεν έχει υλοποιηθεί σωστά

Εντολές wait

- Ακολουθιακή πρόταση: αναστέλλει την εκτέλεση της διεργασίας
- Μια πρόταση wait συντάσσεται με:
  - Φράση ευαισθησίας (sensitivity clause): **wait on**
  - Φράση συνθήκης (condition clause): **wait until**
  - Φράση χρονικής υπέρβασης (timeout clause): **wait for**

Εντολή wait on

- Αναστέλλει τη διεργασία έως ότου ένα γεγονός συμβεί σε ένα από τα σήματα της λίστας
- Η λίστα των σημάτων ονομάζεται λίστα ευαισθησίας (sensitivity list)
- Χρήσιμη σε διεργασίες που μοντελοποιούν μπλοκ συνδυαστικής λογικής

Ισοδύναμες διεργασίες

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

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


Εντολή wait until

- Αναστέλλει τη διεργασία έως ότου ικανοποιηθεί η συνθήκη
- Η συνθήκη ελέγχεται κάθε φορά που συμβαίνει γεγονός σε οποιοδήποτε σήμα της συνθήκης

```
clock_gen : process is
begin
  clk <= '1' after T_pw, '0' after 2*T_pw;
  wait until clk = '0';
end process clock_gen;
```

Εντολή wait for

- Αναστέλλει τη διεργασία για το χρονικό διάστημα που ορίζεται στην πρόταση

```
clock_gen : process is
begin
  clk <= '1' after T_pw, '0' after 2*T_pw;
  wait for 2*T_pw;
end process clock_gen;
```


Μικτές προτάσεις wait

- **wait on** *sensitivity_list* **until** *condition*;
  - Αναστέλλει τη διεργασία έως ότου συμβεί ένα γεγονός
    και η συνθήκη είναι αληθής
- **wait on** *sensitivity_list* **for** *time_period*;
  - Αναστέλλει τη διεργασία έως ότου συμβεί ένα γεγονός
    ή λήξει το χρονικό διάστημα
- **wait until** *condition* **for** *time_period*;
  - Αναστέλλει τη διεργασία έως ότου ικανοποιηθεί η συνθήκη ή λήξει το χρονικό διάστημα

Σχεδίαση συνδυαστικών κυκλωμάτων

- **Προσοχή:** όταν χρησιμοποιούμε διεργασίες για την υλοποίηση ενός συνδυαστικού κυκλώματος
- Όλες οι είσοδοι πρέπει να είναι στη λίστα ευαισθησίας
  - Σε διαφορετική περίπτωση υπάρχει διαφορά μεταξύ του μοντέλου προσομοίωσης και του μοντέλου που προκύπτει από τη σύνθεση
- Σε όλα τα σήματα εξόδου πρέπει να ανατεθεί μία τιμή κάθε φορά που εκτελείται η διεργασία
  - Σε διαφορετική περίπτωση το μοντέλο που προκύπτει από τη σύνθεση θα περιέχει στοιχεία μνήμης (latches) – επομένως θα είναι ακολουθιακό κύκλωμα

Παράδειγμα

- Τι θα συμβεί στο μοντέλο προσομοίωσης του παρακάτω αθροιστή;

```
entity adder is
  port ( a,b : in integer range 0 to 15;
         x   : out integer range 0 to 31 );
end entity adder;
architecture beh of adder is
begin
  p: process (a)
  begin
    x <= a + b;
  end process;
end architecture beh;
```

Παράδειγμα

- Πολύ συχνό σχεδιαστικό λάθος
  - Ελλιπής πρόταση if: δεν αναθέτει τιμή για κάποιο σήμα σε κάποια διακλάδωση της πρότασης if
- Δεν περιγράφει συνδυαστική λογική
  - Υπάρχει ανάδραση στο κύκλωμα
  - Περιγράφει ακολουθιακή λογική

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

architecture beh of inc_if is
begin
  p: process (a,b,en)
  begin
    if en = '1' then
       z <= a;
    else
       y <= b;
    end if;
  end process;
end architecture beh;
```


Παράλληλες αναθέσεις σημάτων

- Παράλληλες προτάσεις ανάθεσης σήματος (concurrent signal assignment statement)
  - Χρησιμοποιούνται στο παράλληλο πεδίο της γλώσσας
- Χρησιμοποιούνται για τη μοντελοποίηση συνδυαστικών κυκλωμάτων
  - Συνδυαστικός μετασχηματισμός των εισόδων στις εξόδους
- Μπορούν να αντικαταστήσουν διεργασίες κάνοντας το μοντέλο πιο ευανάγνωστο
- Δύο μορφές παράλληλων αναθέσεων σημάτων
  - Ανάθεση σήματος υπό συνθήκη (conditional signal assignment)
  - Ανάθεση σήματος με επιλογή (selected signal assignment)

Ανάθεση σήματος υπό συνθήκη

- Μοντέλο πολυπλέκτη 4-σε-1

```
zmux : z <= d0 when sel1 = '0' and sel0 = '0' else
           d1 when sel1 = '0' and sel0 = '1' else
           d2 when sel1 = '1' and sel0 = '0' else
           d3;
```

```
zmux : process is
   begin
     if sel1 = '0' and sel0 = '0' then
        z <= d0;
     elsif sel1 = '0' and sel0 = '1' then
        z <= d1;
     elsif sel1 = '1' and sel0 = '0' then
        z <= d2;
     else
        z <= d3;
     end if;
     wait on d0, d1, d2, d3, sel0, sel1;
   end process zmux;
```

Ισοδύναμη διεργασία

Παραδείγματα

- Ανάθεση σήματος χωρίς συνθήκη

```
PC_incr :
  next_PC <= PC + 4 after 5 ns;
```

```
PC_incr : process is
begin
   next_PC <= PC + 4 after 5 ns;
   wait on PC;
end process PC_incr;
```

- Παραγωγή ενός σήματος reset στην αρχή της προσομοίωσης

```
reset_gen :
reset <= '1', '0' after 200 ns
            when extended_reset else
         '1', '0' after 50 ns;
```

```
reset_gen : process is
begin
if extended_reset then
   reset <= '1', '0' after 200 ns;
else
   reset <= '1', '0' after 50 ns;
end if;
wait;
end process reset_gen;
```

Ανάθεση σήματος με επιλογή

- Μοντέλο ALU

```
alu : with alu_function select
       result <= a + b after Tpd   when alu_add | alu_add_unsigned,
                 a - b after Tpd   when alu_sub | alu_sub_unsigned,
                 a and b after Tpd when alu_and,
                 a or b after Tpd  when alu_or,
                 a after Tpd       when alu_pass_a;
```

```
alu : process is
  begin
    case alu_function is
      when alu_add | alu_add_unsigned => result <= a + b after Tpd;
      when alu_sub | alu_sub_unsigned => result <= a - b after Tpd;
      when alu_and        => result <= a and b after Tpd;
      when alu_or         => result <= a or b after Tpd;
      when alu_pass_a     => result <= a after Tpd;
    end case;
    wait on alu_function, a, b;
  end process alu;
```

Ισοδύναμη διεργασία

Περιγραφές δομής (structural descriptions)

- Η ιεραρχική σχεδίαση βασίζεται στην περιγραφή δομής
- Περιγραφή δομής:
  - περιγράφει τα συστατικά (components) του κυκλώματος και πως αυτά συνδέονται μεταξύ τους
- Περιγραφή δομής στη VHDL:
  - Υλοποιείται με χρήση της πρότασης εμφάνισης στιγμιότυπου συστατικού (component instantiation)

Στιγμιότυπο συστατικού (component instance)

- Δήλωση της οντότητας DRAM_controller και της αρχιτεκτονικής fpld

```
entity DRAM_controller is
    port (rd, wr, mem : in bit;
          ras, cas, we, ready : out bit)
end entity DRAM_controller;
```

- Στιγμιότυπο συστατικού

```
main_memory_controller: entity work.DRAM_controller(fpld)
    port map (cpu_rd, cpu_wr, cpu_mem,
              cpu_ras, cpu_cas, cpu_we, cpu_ready);
```


Στιγμιότυπο συστατικού (component instance)

- Συσχέτισης θέσης (positional association)

```
main_memory_controller: entity work.DRAM_controller(fpld)
    port map (cpu_rd, cpu_wr, cpu_mem,
              cpu_ras, cpu_cas, cpu_we, cpu_ready);
```

- Συσχέτιση ονόματος (named association)

```
main_memory_controller: entity work.DRAM_controller(fpld)
    port map (rd => cpu_rd, wr => cpu_wr, mem = >cpu_mem,
              we => cpu_we, ready => cpu_ready,
              ras => cpu_ras, cas => cpu_cas);
```

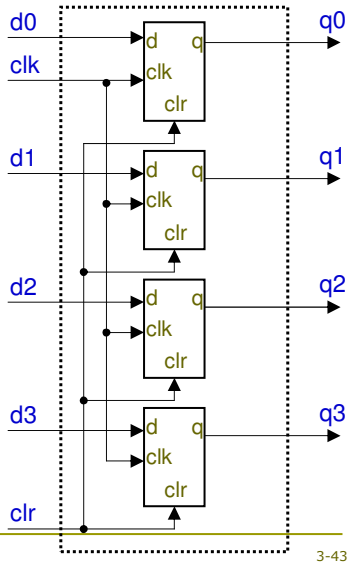
Παράδειγμα: καταχωρητής 4-bit

```
entity reg4 is
  port( clk, clr: in bit
        d0, d1, d2, d3 : in bit;
        q0, q1, q2, q3 : out bit)
end entity reg4;

architecture struct of reg4 is
begin

  bit0 : entity work.d_ff(beh)
     port map (d0, clk, clr, q0);
  bit1 : entity work.d_ff(beh)
     port map (d1, clk, clr, q1);
  bit2 : entity work.d_ff(beh)
     port map (d2, clk, clr, q2);
  bit3 : entity work.d_ff(beh)
     port map (d3, clk, clr, q3);

end architecture struct;
```



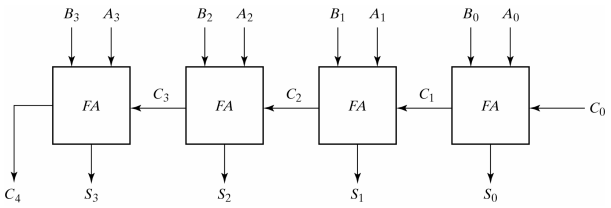
Παράδειγμα: αθροιστής 4-bit

```
entity Full_adder is
  port (A,B,Cin  : in bit;
        Sum,Cout : out bit);
end entity Full_adder;

architecture beh of Full_adder is
 …
end architecture beh;
```

```
entity ripple_adder is
 port (a,b : in bit_vector(3 downto 0);
       cin : in bit;
       sum : out bit_vector(3 downto 0);
       cout : out bit);
end entity ripple_adder;

architecture struct of ripple_adder is
…
end architecture struct;
```



Παράδειγμα: αθροιστής 4-bit

```
architecture struct of ripple_adder is

signal c1,c2,c3 : bit;

begin

  bit0: entity work.full_adder(beh)
    port map (a=>a(0), b=>b(0), cin=>cin,sum=>sum(0), cout=>c1);

  bit1: entity work.full_adder(beh)
    port map (a=>a(1), b=>b(1), cin=>c1, sum=>sum(1), cout=>c2);

  bit2: entity work.full_adder(beh)
    port map (a=>a(2), b=>b(2), cin=>c2, sum=>sum(2), cout=>c3);

  bit3: entity work.full_adder(beh)
    port map (a=>a(3), b=>b(3), cin=>c3, sum=>sum(3), cout=>cout);

end architecture struct;
```


Παραμετροποιημένες μονάδες

- Σχεδίαση παραμετροποιημένων μονάδων με τη χρήση **generic**
- Δήλωση οντότητας
  - Γεννήτρια ρολογιού με μεταβλητή περίοδο και διάρκεια παλμού
- Στιγμιότυπο συστατικού
  - Περίοδος 10 ns και διάρκεια παλμού 5 ns

```
entity clock is
  generic (period, pulse : time);
  port (clk : out bit);
end entity clock;

architecture beh of clock is
begin
  p1: process is
  begin
    clk <= '1';
    wait for pulse;
    clk <= '0';
    wait for (period-pulse);
  end process;
end architecture;
```

```
clock_u1:
  entity work.clock(beh)
  generic map (period => 10ns,
               pulse => 5ns)
  port map (clk => clk);
```

Παραμετροποιημένες μονάδες

- Δήλωση οντότητας
  - Πράξη AND μεταξύ διανυσμάτων bit μήκους width

```
entity and_bv is
  generic (width : integer);
  port(a: in bit_vector(0 to width-1);
       b: in bit_vector(0 to width-1);
       z: out bit_vector(0 to width-1));
end entity and_bv;

architecture beh of and_bv is
begin
  and_op : process (a, b) is
  begin
    for index in 0 to width-1
      z(index) := a(index) and b(index);
    end loop;
   end process and_op;
end architecture beh;
```

- Στιγμιότυπο συστατικού
  - Πράξη AND μεταξύ 8-bit διανυσμάτων

```
and_bv_u1:
   entity work.and_bv
   generic map (width => 8)
   port map (in1, in2, z);
```