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
ΕΛΛΑΔΑ 2008
Ανάπτυξη παντού. Ανάπτυξη για όλους.

Ακολουθιακές εντολές

(Peter Ashenden, The Students Guide to VHDL)

Εντολή If

Τα βασικά χαρακτηριστικά της είναι τα εξής:

- Μπορεί να χρησιμοποιηθεί για τον έλεγχο μίας ή περισσοτέρων συνθηκών.
- Η πρώτη συνθήκη ελέγχεται με το **if** και οι επόμενες με το **elsif**.
- Μπορεί να συμπεριληφθεί και μία διακλάδωση χωρίς συνθήκη **else**.
- Εκτελείται πάντα η πρώτη διακλάδωση που είναι αληθής, χωρίς να ενδιαφέρουν οι επόμενες συνθήκες.
- Σε κάθε διακλάδωση μπορεί να υπάρχουν περισσότερες της μίας ακολουθιακές εντολές.

Σύνταξη:

```
if condition then
      sequential statements
   [elsif condition then
      sequential statements ]
   [else
      sequential statements ]
end if;
```

Εντολή If

```
entity MUX_4_1a is
   port (S1, S0, A, B, C, D: in std_logic;
            Z: out std_logic);
   end MUX_4_1a;
architecture behav_MUX41a of MUX_4_1a is
begin
   P1: process (S1, S0, A, B, C, D)
   begin
     if (( not S1 and not S0 )='1') then
                Z <= A;
          elsif (( not S1 and S0) = '1') then
                Z<=B;
          elsif ((S1 and not S0) ='1') then
                Z <=C;
          else
                Z<=D;
      end if;
   end process P1;
end behav_MUX41a;
```

Εντολή Πολλαπλής Επιλογής Case

Χρησιμοποιείται όταν έχουμε πολλές διακλαδώσεις που εξαρτώνται από μία λογική έκφραση.

```
case expression is
      when choices =>
            sequential statements
      when choices =>
            sequential statements
            -- branches are allowed
       [ when others => sequential statements ]
end case;
```

Οι βασικοί κανόνες που ακολουθούμε είναι:

- Η έκφραση πρέπει να είναι ακέραιου τύπου ή τύπου απαρίθμησης
- Δεν πρέπει να υπάρχει επικάλυψη επιλογών
- Εάν δεν υπάρχει η επιλογή “when others” τότε πρέπει να καλύπτονται όλες οι πιθανές τιμές της έκφρασης.

Εντολή Case

```
entity MUX_4_1 is
   port ( SEL: in std_logic_vector(2 downto 1);
          A, B, C, D: in std_logic;
          Z: out std_logic);
   end MUX_4_1;
architecture behav_MUX41 of MUX_4_1 is
begin
   PR_MUX: process (SEL, A, B, C, D)
   begin
     case SEL is
          when "00" => Z <= A;
          when "01" => Z <= B;
          when "10" => Z <= C;
          when "11" => Z <= D;
          when others => Z <= 'X';
     end case;
   end process PR_MUX;
end behav_MUX41;
```

Εντολή Case

- Θα πρέπει να χρησιμοποιούνται σαφώς καθορισμένοι τύποι για την έκφραση επιλογής.

```
subtype int_4 is integer range 0 to 3;
variable choise is int_4;
case choise is
        when 0 => …
        …
        when 3 => …
end case;
```

- Σε κάθε εναλλακτική μπορεί να αντιστοιχούν περισσότερες επιλογές:

```
when 0 | 1 | 2 => …
```

- Μπορεί να χρησιμοποιηθεί και εύρος σε μία εναλλακτική:

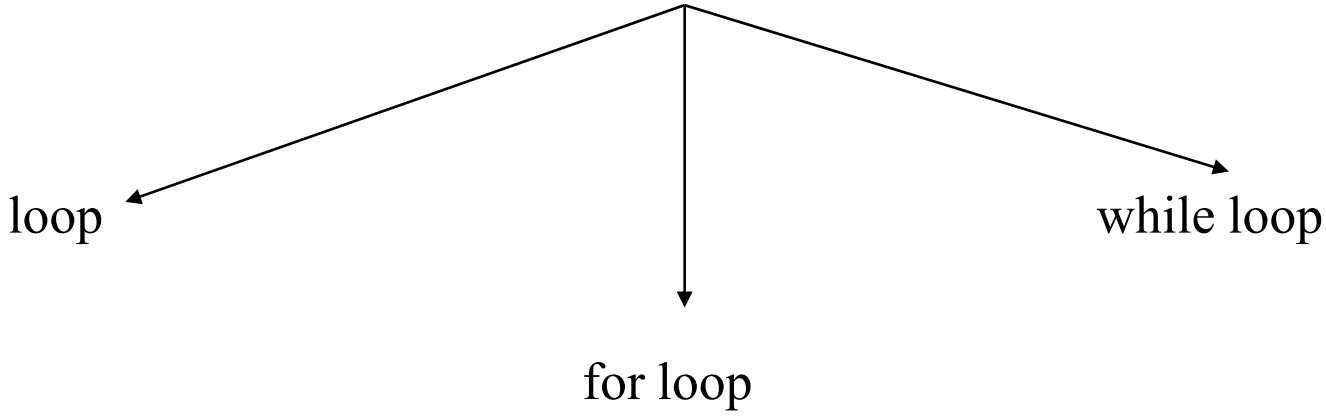
```
when 0 to 2 => …
```

- Η συνθήκη κάθε επιλογής πρέπει να είναι στατικά ορισμένη.
- Σε μία διακλάδωση που δεν χρησιμοποιείται μπαίνει η εντολή null;

Εντολές Επανάληψης

Γενική Δομή Επανάληψης

```
[ loop_label :]iteration_scheme loop
      sequential statements
      [next  [label] [when condition];
      [exit  [label] [when condition];
end loop [loop_label];
```



Εντολές Επανάληψης

Δομή Βασικής Επανάληψης

```
[ loop_label :] loop
      sequential statements
      [next  [label] [when condition];
      [exit  [label] [when condition];
end loop [ loop_label];
```

Στις κοινές γλώσσες προγραμματισμού η άπειρη επανάληψη δεν είναι χρήσιμη αλλά όχι και στην μοντελοποίηση κυκλωμάτων.

loop

count:=count+1;

exit when count=2;

end loop;

Εντολές Επανάληψης

Example of a basic **loop** to implement a counter that counts from 0 to 31

```
entity COUNT31 is
   port ( CLK: in std_logic;
          COUNT: out integer);
   end COUNT31;
architecture behav_COUNT of COUNT31 is
begin
   P_COUNT: process
     variable intern_value: integer :=0;
   begin
     COUNT <= intern_value;
     loop
        wait until CLK='1';
        intern_value:=(intern_value + 1) mod 32;
        COUNT <= intern_value;
     end loop;
     end process P_COUNT;
end behav_COUNT;
```

Η εντολή **wait** προκαλεί διακοπή της επανάληψης και είναι απαραίτητη στην μοντελοποίηση συστημάτων. Χωρίς διακοπή δεν προχωρά ο χρόνος κατά την εξομοίωση

Εντολή Exit

```
entity counter is  port ( clk, reset : in bit;  count : out natural ); end entity counter;
architecture behavior of counter is
begin
  incrementer : process is
    variable count_value : natural := 0;
  begin
    count <= count_value;
    loop
      loop
            wait until clk = '1' or reset = '1';
            exit when reset = '1';
            count_value := (count_value + 1) mod 16;
            count <= count_value;
      end loop;
      count_value := 0;
       count <= count_value;
       wait until reset = '0';
    end loop;
  end process incrementer;
end architecture behavior;
```

- Χρησιμοποιείται για τερματισμό μίας επανάληψης και μεταφέρει τον έλεγχο στην πρώτη εντολή μετά τον βρόγχο: **exit;**
- Μπορεί να εκτελεστεί υπό συνθήκη: **exit when** *condition*;

Εντολή Exit

Σε περιπτώσεις που πρέπει να προκληθεί αυτόματη έξοδος από εξωτερικό βρόγχο πρέπει να χρησιμοποιηθεί και η ετικέτα του βρόγχου.

```
outer : loop
    …
    inner : loop
            …
            exit outer when condition-1
            …
            exit when condition-2
            …
    end loop inner;
    …
    exit when condition-3
    …
end loop outer
```

Εντολή Next

- Χρησιμοποιείται όπως και η εντολή exit.
- Προκαλεί τερματισμό της τρέχουσας επανάληψης και μετάβαση στην επόμενη
- Η χρήση της εντολής next σε εμφωλευμένα loop επιτρέπεται αλλά δεν χρησιμοποιείται.

```
loop
    …
    next when condition;
    …
end loop
```

Εντολή Null

- Μηδενική πρόταση (καμία ενέργεια).
- Χρησιμοποιείται σε προτάσεις πολλαπλής επιλογής όταν δεν απαιτείται να γίνει κάποια ενέργεια

```
case inputs is
    when "00" => null;
    …
end case
```

Εντολές Επανάληψης While

- Ο έλεγχος της συνθήκης γίνεται πριν την εκτέλεση κάθε επανάληψης.
- Μπορεί να μην εκτελεστεί καμία επανάληψη αν αρχικά η συνθήκη είναι ψευδής.
- Πρέπει να εξασφαλίζεται η έξοδος είτε με την συνθήκη, είτε με εντολή exit.

Δομή Επανάληψης While

```
[ loop_label :] while condition loop
      sequential statements
      [next  [label] [when condition];
      [exit  [label] [when condition];
end loop[ loop_label ];
```

Παράδειγμα While

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

architecture series of cos is
begin
  summation : process (theta)
    variable sum, term : real;
    variable n : natural;
  begin
    sum := 1.0;
    term := 1.0;
    n := 0;
    while abs term > abs (sum / 1.0E6) loop
      n := n + 2;
      term := (-term) * theta**2 / real(((n-1) * n));
      sum := sum + term;
    end loop;
    result <= sum;
  end process summation;
end architecture series;
```

Εντολή Επανάληψης For

- Προκαθορίζεται ο αριθμός των επαναλήψεων.
- Η δήλωση του identifier που χρησιμοποιείται για την επανάληψη γίνεται αυτόματα από την δομή της επανάληψης.
- Δεν μπορεί να γίνει ανάθεση στον identifier της δομής.
- Ο identifier είναι διακριτού τύπου

Δομή Επανάληψης For

```
[ loop_label :] for identifier in range loop
      sequential statements
      [next  [label] [when condition];
      [exit  [label] [when condition];
end loop[ loop_label ];
```

integer_expr to/downto integer_expr

Εντολή Επανάληψης For

```
entity parity is
  port ( x0, x1, x2, x3 : in bit;  y : out bit );
end parity;

architecture behave of parity is
begin
  p0 : process (x0, x1, x2, x3 )
    variable count : integer range 0 to 4;
    variable s : bit_vector (1 to 4);
  begin
    s := x0 & x1 & x2 & x3;
    count := 0;
    for i in 1 to 4 loop
      if s(i)='1' then
        count:=count+1;
      end if;
  end loop;
    if count mod 2=0 then y <= '1';
    elsif count mod 2=1 then y <= '0';
    end if;
  end process;
  end;
```

Εντολή Επανάληψης For

```
erroneous : process is
    variable i, j: integer;
begin
    i:= loop_param;              --error
    for loop_param in 1 to 10 loop;
            loop_param:=5;       --error
    end loop;
    j:= loop_param;              --error
end process erroneous;
```

```
hiding_example : process is
    variable a, b: integer;
begin
    a := 10;
    for a in 0 to 7 loop;
            b := a;
    end loop;
    --a=10 and b=7
end process hiding_example;
```

Εντολή Wait

Εντολή Wait: σταματά μία διαδικασία έως ότου να συμβεί ένα γεγονός:

wait until condition

wait for time expression

wait on signal

wait

Για παράδειγμα η αναμονή θετικής ακμής ρολογιού γράφεται ως εξής:

wait until CLK'event and CLK='1'

Μία διαδικασία (process) με εντολή wait δεν μπορεί να έχει και sensitivity list.

Όταν χρησιμοποιείται ένα ή περισσότερα wait σε μία process τότε θα χρησιμοποιηθεί ακολουθιακή λογική και τα αποτελέσματα των υπολογισμών θα αποθηκευτούν σε flip flops.

Εντολές Assertion & Report

Η εντολή assert ελέγχει μία συνθήκη και εάν βρεθεί ψευδής τότε εκτυπώνει το κατάλληλο μήνυμα λάθους και ενημερώνει τον εξομοιωτή για τον βαθμό σημαντικότητας του λάθους.

assert boolean_expression
report expression
severity expression

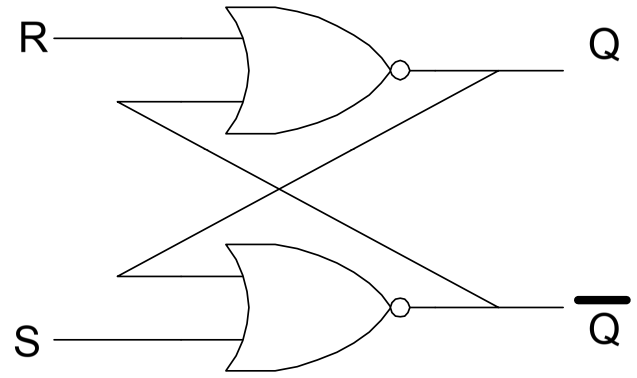
Η σημαντικότητα ενός λάθους (**severity**) παίρνει τις τιμές **note, warning, error, failure**.

assert clock_pulse_width>=min_width
report “Clock less than minimum allowed”
severity error

Παράδειγμα Assertion & Report

Έλεγχος για λανθασμένη τροφοδότηση εισόδων σε flip flop

```
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;
```



Λανθασμένη Κατάσταση: S=R=1 ⇒ SR=1

Έγκυρες Καταστάσεις: $\overline{SR}$=1 (S=0 ή R=0)

Παράδειγμα Assertion & Report

Εάν θέλουμε κάποιες συνθήκες να ελέγχονται σε όλες τις αρχιτεκτονικές για κάποια οντότητα, τότε χρησιμοποιούμε τα assert και report στην δήλωση της οντότητας.

```
entity dff is
    generic (setup, hold : time := 4 ns);
    port (set, rst, clk, d : in std_logic; q, qb: out std_logic);
begin
assert not(set='1' and rst='1') report "invalid state" severity warning;
assert not (clk='1' and clk'event and d'last_event<setup ) report "setup violation" severity warning;
assert not (d'event and clk='1' and clk'last_event<hold) report "hold violation"  severity warning;
end dff;
```