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.

Υποπρογράμματα – Πακέτα

(Peter Ashenden, The Students Guide to VHDL)

Procedures

- Μία διαδικασία (procedure) δηλώνεται και κατόπιν καλείται όσες φορές θέλουμε.

  procedure identifier [(parameter_interface_list)] **is**
  {subprogram_declarative_part}
  begin
  {sequential statement}
  end [**procedure**] [identifier]

- Οποιαδήποτε δήλωση γίνει στην διαδικασία είναι ορατή μόνο εσωτερικά (τοπικά).
- Η κλήση μίας ρουτίνας γίνεται με το όνομά της ακολουθούμενο από την λίστα των παραμέτρων της.
- Η επιστροφή από την διαδικασία γίνεται στο τέλος της ή με το **return**.

Παράδειγμα

```
architecture rtl of control_processor is
  type func_code is (add, subtract);
  signal op1, op2, dest : integer;
  signal Z_flag : boolean;
  signal func : func_code;
begin
  alu : process is
    procedure do_arith_op is
      variable result : integer;
    begin
      case func is
        when add => result := op1 + op2;
        when subtract => result := op1 - op2;
      end case;
      dest  <=  result after Tpd;
      Z_flag  <=  result = 0 after Tpd;
    end procedure do_arith_op;
begin
        do_arith_op;
end process alu;
end architecture rtl;
```

Παράμετροι - Μεταβλητές

- Μία διαδικασία (procedure) μπορεί να παίρνει παραμέτρους ώστε να προσαρμόζει την λειτουργία της.
- Η δήλωση παραμέτρων ακολουθεί το συντακτικό της δήλωσης θυρών.

```
procedure do_arith_op ( op : in func_code ) is
    variable result : integer;
  begin
    case op is
      when add => result := op1 + op2;
      when subtract =>  result := op1 - op2;
    end case;
    dest  <=  result after Tpd;
    Z_flag  <=  result = 0 after Tpd;
  end procedure do_arith_op;
```

- Η κλήση γίνεται ως: do_arith_op(add);

Παράμετροι - Μεταβλητές

- Οι παράμετροι που δηλώνονται ως **in** θεωρούνται από την procedure ως σταθερές αφού δεν μπορεί να μεταβληθεί η τιμή τους. Στην procedure περνάει μόνο η τιμή τους.
- Η επιστροφή τιμών γίνεται με παραμέτρους "**out**".

```
procedure addu ( a,b: in word32; result: out word32;  overflow: out boolean ) is
   variable sum : word32;
   variable carry : bit := '0';
 begin
   for index in sum'reverse_range loop
     sum(index) := a(index) xor b(index) xor carry;
     carry := ( a(index) and b(index) ) or ( carry and ( a(index) xor b(index) ) );
   end loop;
   result := sum;
   overflow := carry = '1';
 end procedure addu;
```

Παράμετροι - Μεταβλητές

- Οι παράμετροι που δηλώνονται ως **out** δεν μπορούν να περάσουν τιμή στην procedure.
- Το ταυτόχρονο πέρασμα και η επιστροφή τιμών γίνεται με παραμέτρους "in**out**".

```
procedure negate ( a : inout word32 ) is
   variable carry_in : bit := '1';
   variable carry_out : bit;
  begin
   a := not a;
   for index in a'reverse_range loop
     carry_out :=  a(index) and carry_in;
     a(index) := a(index) xor carry_in;
     carry_in := carry_out;
   end loop;
  end procedure negate;
```

Παράμετροι – Σήματα

- Ένα **σήμα** που περνάει σαν παράμετρος μπορεί να είναι **in, out, inout**.
- **Διαφοροποίηση από μεταβλητές**: Το σήμα που περνάει σαν παράμετρος **in** περνάει σαν αντικείμενο και όχι σαν τιμή. Έτσι μπορεί να εκτελεστεί ένα wait statement στην procedure αφού η τιμή του σήματος όταν αλλάξει είναι ορατή εσωτερικά στην procedure.

```
procedure receive_packet ( signal rx_data : in bit; signal rx_clock : in bit;
                           data_buffer : out packet_array ) is
 begin
   for index in packet_index_range loop
     wait until rx_clock = '1';
     data_buffer(index) := rx_data;
   end loop;
 end procedure receive_packet
```

Τιμές Default

- Μπορούμε να δώσουμε τιμές **default** στις παραμέτρους. Οι τιμές αυτές περνούν στις αντίστοιχες παραμέτρους εάν δεν οριστούν άλλες από τον χρήστη κατά την κλήση της procedure.

```
procedure increment ( a : inout word32;  by : in word32 := X"0000_0001" ) is
   variable sum : word32;
   variable carry : bit := '0';
begin
   for index in a'reverse_range loop
     sum(index) := a(index) xor by(index) xor carry;
     carry := ( a(index) and by(index) ) or ( carry and ( a(index) xor by(index) ) );
   end loop;
   a := sum;
 end procedure increment;
```

- Η κλήση γίνεται ως: incrementer(count, X“0000_0004”) ή ως

  incrementer(counter); ή ως

  incrementer(counter, by => open);

Παράμετροι – Πίνακες χωρίς όρια

- Ένας μη οριοθετημένος πίνακας μπορεί να χρησιμοποιηθεί σαν παράμετρος σε διαδικασία (procedure).
- Κατά την κλήση της procedure πρέπει να περάσουμε σαν παράμετρο έναν οριοθετημένο πίνακα.

```
procedure find_set(v:in bit_vector; found:out boolean; first_set_index:out natural ) is
begin
   for index in v'range loop
     if v(index) = '1' then
              found := true;
              first_set_index := index;
              return;
     end if;
   end loop;
   found := false;
 end procedure find_first_set;
```

Συναρτήσεις

- Οι συναρτήσεις χρησιμοποιούνται ως γενίκευση των εκφράσεων.
- Μία συνάρτηση επιστρέφει τιμή οπότε μπορεί να χρησιμοποιηθεί σε οποιαδήποτε έκφραση.

Χρήση συναρτήσεων

- Μετατροπές τύπων δεδομένων και συστημάτων.
- Χρησιμοποιούνται σε αρχιτεκτονικές αντί δημιουργίας στιγμιοτύπων.
- Επιλύουν διαμάχες πολλαπλών οδηγών σημάτων (resolution functions).
- Δημιουργία τελεστών υπερφόρτωσης.

Συναρτήσεις

```
subprogram_body <=
    [pure | impure]
    function identifier [(parameter_interface_list)] return type_mark is
    {subprogram_declarative_item}
    begin
        {sequential_statement}
    end [function] [identifier]
```

Κανόνες συναρτήσεων

- Οι παράμετροι μίας συνάρτησης πρέπει να είναι σταθερές (σε μία οντότητα μόνο οι θύρες τύπου **in** μπορούν να συμμετάσχουν).
- Η συνάρτηση επιστρέφει τιμή με την κλήση **return** expression; Κάθε συνάρτηση πρέπει να περιλαμβάνει τουλάχιστον μία έκφραση return.
- Οι συναρτήσεις περιέχουν μόνο ακολουθιακές προτάσεις.
- Απαγορεύεται η χρήση εντολής wait σε μία συνάρτηση

Παράδειγμα: αντιστροφή byte

```
entity byte_inv is
  port ( x : in bit_vector (7 downto 0); y : out bit_vector (7 downto 0) );
end byte_inv;

architecture behave of byte_inv is
function invert (input: bit_vector (7 downto 0) ) return bit_vector is
  variable x_out : bit_vector (7 downto 0);
begin
        for i in 7 downto 0 loop
                x_out(7-i) := input(i);
        end loop;
        return x_out;
end invert;
begin
        y<invert(x);
end;
```

Παράδειγμα: μετατροπή τύπου

```
entity bool_conv is
  port ( x : in bit; y : out bit );
end bool_conv;

architecture behave of bool_conv is
function boolean_to_bit (z: boolean)
return bit is
begin
      if z then return '1';
      else return '0';
      end if;
end boolean_to_bit;
begin
   p0: process (x)
   variable b: boolean;
   begin
      b:=true;
      y<=x xor boolean_to_bit(b);
   end process;
end behave;
```

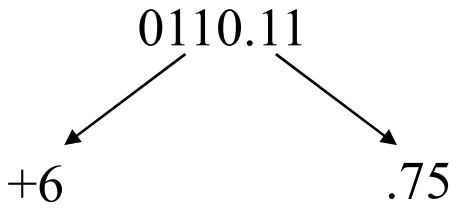
Παράδειγμα: μετατροπή ακεραίου σε δυαδικό

```
function int_to_bit_vector (value: integer)
return bit_vector is
variable z: bit_vector (N-1 downto 0);
variable a: integer:=value;
begin
        for i in 0 to N-1 loop
                if ( (a mod 2)=1 ) then z(i):='1';
                else z(i):='0';
                end if;
                a:=a/2;
        end loop;
        return z;
end int_to_bit_vector;
```

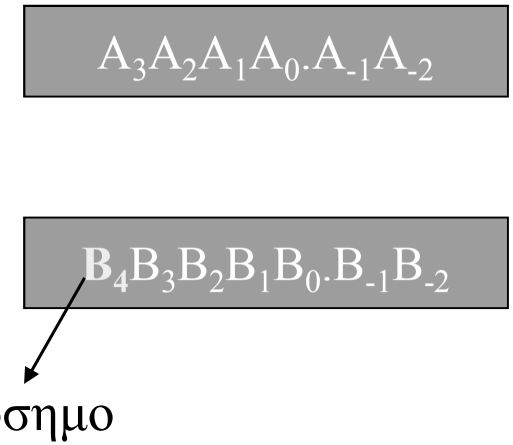
Το a είναι απαραίτητο καθώς η μεταβλητή value δεν μπορεί να μεταβληθεί

Αναπαράσταση σταθερής υποδιαστολής

Η θέση της υποδιαστολής δηλώνεται ρητά

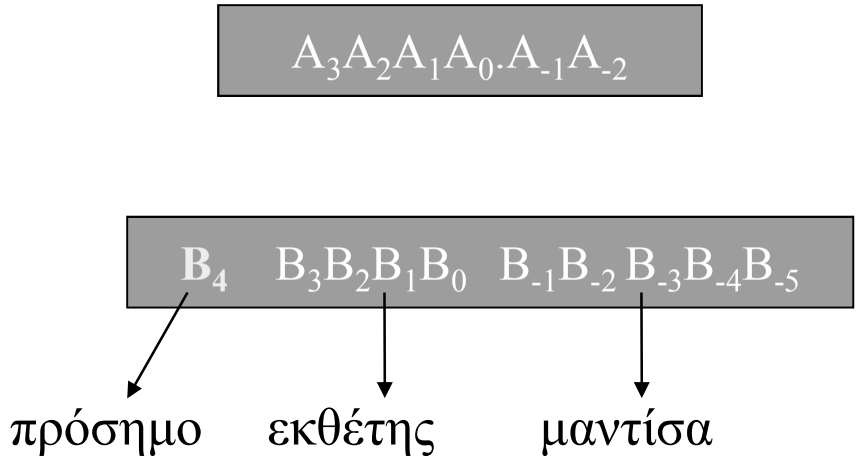



```
library ieee;
use ieee.fixed_pkg.all;
entity fixed_converter is
  port ( A : in ufixed (3 downto -2);
         B : out sfixed (4 downto -2)
  );
```



Αναπαράσταση κινητής υποδιαστολής


```
library ieee;
use ieee.float_pkg.all;
entity float_converter is
  port ( B : in float (4 downto -5);
  );
```



Συναρτήσεις επίλυσης διαμαχών

Είναι χρήσιμες όταν ένα σήμα έχει περισσότερους του ενός οδηγούς

```
entity unresolved is
  port ( x, y : in bit; f : out bit );
end unresolved;

architecture behave of unresolved is
signal z: bit;
begin
        z<=x;
        z<=y;
        f<=z;
end;
```

vhd(8): Error, multiple sources on unresolved signal z;

Συναρτήσεις επίλυσης διαμαχών

```
entity resolved is
  port ( x, y : in bit; f : out bit );
end resolved;
```

Απεριόριστο – δεν γνωρίζουμε πόσοι είναι οι οδηγοί

```
architecture behave of resolved is
type input_signals is array (natural range <>) of bit;
function resolution (inputs: input signals)
return bit is
variable result: bit:='1';
begin
        for i in inputs'range loop
                result:=result and inputs(i);
                exit when result='0';
        end loop
        return result;
end resolution;
```

```
signal z: resolution bit;
begin
        z<=x;
        z<=y;
        f<=z;
end resolution;
```

Συναρτήσεις υπερφόρτωσης τελεστών

```
entity add_bit_vectors is
  generic (size: integer:=8);
  port ( x, y : in bit_vector(size-1 downto 0);
         z : out bit_vector(size-1 downto 0) );
end add_bit_vectors;

architecture behave of add_bit_vectors is
function "+" (a, b : bit_vector(size-1 downto 0))
return bit_vector is
variable result: bit_vector (a'range);
variable carry: bit;
begin
      carry:='0';
      for i in a'low to a'high loop
            result(i):=a(i) xor b(i) xor carry;
            carry:=((a(i) or (b(i)) and carry) or (a(i) and b(i));
      end loop
            return result;
      end;
      begin
            z<=x+y;
      end behave;
```

Δηλώσεις Πακέτων

- Ένα πακέτο στην VHDL είναι ένας τρόπος ομαδοποίησης μίας συλλογής από σχετικές δηλώσεις που εξυπηρετούν ένα κοινό σκοπό.
- Ένα πακέτο μπορεί να αποτελείται από ένα σύνολο δηλώσεων που μοντελοποιούν έναν σχεδιασμό ή ένα σύνολο υποπρογραμμάτων.

  package_declaration <=

  package identifier **is**

  { package_declarative_item}

  end [**package**] [identifier];

- Για να χρησιμοποιήσουμε μία δήλωση ενός πακέτου χρησιμοποιούμε τελείες:

  Όνομα_Βιβλιοθήκης . Όνομα_Πακέτου . Αντικείμενο

- Ένα πακέτο όταν αναλυθεί τοποθετείται στην βιβλιοθήκη εργασίας.
- Γενικά σήματα (ρολόϊ) είναι προτιμότερο να τοποθετούνται σε πακέτα.

Παράδειγμα

```
package cpu_types is
  constant word_size : positive := 16;
  constant address_size : positive := 24;
  subtype word is bit_vector(word_size - 1 downto 0);
  subtype address is bit_vector(address_size - 1 downto 0);
  type status_value is ( halted, idle, fetch, mem_read, mem_write,
                         io_read, io_write, int_ack );
end package cpu_types;
```

```
entity address_decoder is
  port ( addr : in work.cpu_types.address;  status : in work.cpu_types.status_value;
        mem_sel, int_sel, io_sel : out bit );
end entity address_decoder;
architecture functional of address_decoder is
  constant mem_low : work.cpu_types.address := X"000000";
  constant mem_high : work.cpu_types.address := X"EFFFFF";
  constant io_low : work.cpu_types.address := X"F00000";
  constant io_high : work.cpu_types.address := X"FFFFFF";
```

Παράδειγμα

```
begin
  mem_decoder :
    mem_sel <=      '1' when ( work.cpu_types."="(status, work.cpu_types.fetch)
                    or work.cpu_types."="(status, work.cpu_types.mem_read)
                    or work.cpu_types."="(status, work.cpu_types.mem_write) )
                    and addr >= mem_low and addr <= mem_high
                    else  '0';

  int_decoder :
    int_sel <= '1' when work.cpu_types."="(status, work.cpu_types.int_ack) else  '0';

  io_decoder :
  io_sel <= '1' when ( work.cpu_types."="(status, work.cpu_types.io_read)
                    or work.cpu_types."="(status, work.cpu_types.io_write) )
                    and addr >= io_low and addr <= io_high
              else '0';

end architecture functional;
```

Πακέτα

- Ένα πακέτο μπορεί να περιλαμβάνει και δηλώσεις υποπρογραμμάτων. Οποιοδήποτε πρόγραμμα μπορεί να χρησιμοποιήσει το πακέτο και να καλέσει τα υποπρογράμματα γνωρίζοντας μόνο την διασύνδεση.
- Ιδιαίτερα χρήσιμες είναι οι δηλώσεις σταθερών.

Σώμα Πακέτου

- Υπάρχει μόνο όταν στην δήλωση του πακέτου υπάρχουν υποπρογράμματα.

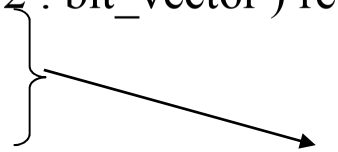
package body identifier **is**
{ package_body_declarative_item}
end [**package body**] [identifier];

- Το σώμα πακέτου περιλαμβάνει πάλι τις δηλώσεις υποπρογραμμάτων.
- Ένα σώμα μπορεί να περιλαμβάνει και επιπλέον υποπρογράμματα – δηλώσεις.

Πακέτα

```
package bit_vector_signed_arithmetic is
  function "+" ( bv1, bv2 : bit_vector ) return bit_vector;
  function "-" ( bv : bit_vector ) return bit_vector;
  function "*" ( bv1, bv2 : bit_vector ) return bit_vector;
  -- . . .
end package bit_vector_signed_arithmetic;
```

```
package body bit_vector_signed_arithmetic is
  function "+" ( bv1, bv2 : bit_vector ) return bit_vector is -- . . .
  function "-" ( bv : bit_vector ) return bit_vector is -- . . .
  function mult_unsigned ( bv1, bv2 : bit_vector ) return bit_vector is -- . . .
    -- . . .
  end function mult_unsigned;
```



Πολλαπλασιασμός μη προσημασμένων

Πακέτα

```
function "*" ( bv1, bv2 : bit_vector ) return bit_vector is
  begin
    if bv1(bv1'left) = '0' and bv2(bv2'left) = '0' then
      return mult_unsigned(bv1, bv2);
    elsif bv1(bv1'left) = '0' and bv2(bv2'left) = '1' then
      return -mult_unsigned(bv1, -bv2);
    elsif bv1(bv1'left) = '1' and bv2(bv2'left) = '0' then
      return -mult_unsigned(-bv1, bv2);
    else  return mult_unsigned(-bv1, -bv2);
    end if;
  end function "*";
  -- . . .
end package body bit_vector_signed_arithmetic;
```

Πολλαπλασιασμός προσημασμένων

Πακέτα

- Η χρήση ενός αντικειμένου δηλωμένου σε ένα πακέτο απαιτεί την χρήση πολλών ονομάτων (βιβλιοθήκης, πακέτου, αντικειμένου).
- Η πολλαπλή χρήση αντικειμένων μίας βιβλιοθήκης οδηγεί σε μακρόσυρτα ονόματα.
- Με την χρήση της έκφρασης "**use βιβλιοθήκη.πακέτο**" μπορούν να γίνουν απευθείας ορατά τα αντικείμενα ενός πακέτου στο πρόγραμμα.
- Ένα θέλουμε να έχουμε απευθείας πρόσβαση σε όλα τα αντικείμενα ενός πακέτου είναι προτιμότερη η χρήση της έκφρασης

"**use βιβλιοθήκη.πακέτο.all**"

- Κάθε πακέτο που χρησιμοποιείται από μία οντότητα θα πρέπει απαραίτητα να δηλώνεται πριν την οντότητα (και κάθε οντότητα).
- Παράδειγμα: για την χρήση του IEEE standard logic package έχουμε

use ieee.std_logic_1164.all

Βιβλιοθήκες

- Ένα μοντέλο μετά την μεταγλώττιση αποθηκεύεται σε μία βιβλιοθήκη.
- Επιτρέπουν την χρήση των μονάδων σχεδίασης και μελλοντικά.
- Η βιβλιοθήκη work είναι default και αφορά τον τρέχων σχεδιασμό κάθε φορά.
- Για την δημιουργία μιας βιβλιοθήκης είναι απαραίτητη η χρήση πακέτων.

Παράδειγμα:

δημιουργία της βιβλιοθήκης sync η οποία περιλαμβάνει ένα πακέτο καταχωρητών

Βιβλιοθήκη sync: βήμα 1ο

1. Δημιουργούμε 4 χωριστά αρχεία με 4 περιγραφές των στοιχείων αποθήκευσης

```
library ieee;
use ieee.std_logic_1164.all
entity dff1 is
  port ( d, clk : in std_logic; q : out std_logic );
end;

architecture behave of dff1 is
begin
        process (d, clk)
        begin
                if (clk = '1') and clk'event then q<=d; end if;
        end process;
end behave;
```

Βιβλιοθήκη sync: βήμα 1ο

```
library ieee;
use ieee.std_logic_1164.all
entity dff1_rst is
        port ( d, clk, rst : in std_logic; q : out std_logic );
end dff1_rst;

architecture behave of dff1_rst is
begin
        process (d, clk, rst)
        begin
                if (rst = '1') then q<='0';
                elsif (clk = '1') and clk'event then q<=d;
                end if;
        end process;
end behave;
```

Βιβλιοθήκη sync: βήμα 1ο

```
library ieee;
use ieee.std_logic_1164.all
entity dff8 is
  generic (size: integer:=8);
  port (    d : in std_logic_vector(size-1 downto 0);
            clk : in std_logic;
            q : out std_logic_vector(size-1 downto 0) ); end;

architecture behave of dff8 is
begin
      process (d, clk)
      begin
            if (clk = '1') and clk'event then q<=d; end if;
      end process;
end behave;
```

Βιβλιοθήκη sync: βήμα 1ο

```
library ieee;
use ieee.std_logic_1164.all
entity dff8_rst is
  generic (size: integer:=8);
  port (    d : in std_logic_vector(size-1 downto 0);
            clk, rst : in std_logic;
            q : out std_logic_vector(size-1 downto 0) ); end;

architecture behave of dff8_rst is
begin
        process (d, clk)
        begin
        if (rst = '1') then q <= (q'range =>'0');
        elsif (clk = '1') and clk'event then q<=d; end if;
        end process;
end behave;
```

Βιβλιοθήκη sync: βήμα 2ο

2. Δημιουργούμε το αρχείο που θα περιέχει το βασικό πακέτο

```
library ieee;
use ieee.std_logic_1164.all
package registers is
component dff1 port ( d, clk : in std_logic; q : out std_logic ); end component;
component dff1_rst port ( d, clk, rst: in std_logic; q: out std_logic ); end component;
component dff8  generic (size: integer:=8); port (d: in std_logic_vector(size-1
      downto 0); clk : in std_logic; q : out std_logic_vector(size-1 downto 0) );
      end component;
component dff8_rst  generic (size: integer:=8); port (d: in std_logic_vector(size-1
      downto 0); clk, rst : in std_logic; q: out std_logic_vector(size-1 downto 0) );
      end component;
end registers;
```

Βιβλιοθήκη sync: βήμα 3ο

3. Εκτελούμε μεταγλώττιση στο όνομα sync (ανάλογα με τον μεταγλωττιστή)
4. Η χρήση γίνεται ως εξής:

library sync;

use sync.registers.**all**

Σταθερές Generic

- Οι σταθερές generic παραμετροποιούν τις οντότητες.
- Η δήλωση τους γίνεται ως: **generic**(generic_interface_list); και τοποθετείται πριν την δήλωση των θυρών.
- Μόνο σταθερές μπορούν να δηλωθούν με αυτόν τον τρόπο.
- Κάθε generic σταθερά είναι ορατή σε όλη την οντότητα.
- Οι σταθερές generic μπορεί να έχουν default τιμές ή/και να αρχικοποιηθούν κατά το instantiation με την έκφραση **generic map** κατά αναλογία με το **port map**.

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

Σταθερές Generic

```
architecture simple of and2 is

begin
        y<=a and b after Tpd;
end architecture simple;

gate1: entity work.and2(simple)
        generic map(Tpd=>2 ns);
        port map(a=>sig1, b=>sig2, y=>sig_out);

gate2: entity work.and2(simple)
        generic map(Tpd=>3 ns);
        port map(a=>a1, b=>b1, y=>sig1);
```

Components

- Τα components χρησιμοποιούνται για την περιγραφή διασύνδεσης υποσυστημάτων.
- Η δήλωση ενός component καθορίζει το interface με την αντίστοιχη οντότητα σε όρους σταθερών generic και θυρών.

component identifier
generic (generic_interface_list);
port (port_interface_list);
end component [identifier];

- Η δήλωση γίνεται μέσα στο σώμα αρχιτεκτονικής όπου θα χρησιμοποιηθεί το component.
- Τα συντακτικά δήλωσης μίας οντότητας και ενός component είναι όμοια καθώς εξυπηρετούν τον ίδιο σκοπό.
- Το instantiation ενός component γίνεται ως εξής:

Label: [**component**] component_name **generic map** (list) **port map** (list);

Components

```
entity reg4 is
    port ( clk, clr : in bit;  d : in bit_vector(0 to 3); q : out bit_vector(0 to 3) );
end reg4;

architecture struct of reg4 is
component flipflop is
    generic ( Tprop, Tsetup, Thold : delay_length );
    port ( clk : in bit;  clr : in bit;  d : in bit; q : out bit );
end component;

begin
 bit0 : flipflop
    generic map ( Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns )
    port map ( clk => clk, clr => clr, d => d(0), q => q(0) );
```

Components

```
bit1 : flipflop
    generic map ( Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns )
    port map ( clk => clk, clr => clr, d => d(1), q => q(1) );

bit2 : flipflop
    generic map ( Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns )
    port map ( clk => clk, clr => clr, d => d(2), q => q(2) );

 bit3 : flipflop
    generic map ( Tprop => 2 ns, Tsetup => 2 ns, Thold => 1 ns )
    port map ( clk => clk, clr => clr, d => d(3), q => q(3) );

end struct;
```

Component Configuration

- Το component αντιστοιχεί σε κάποια οντότητα της βιβλιοθήκης work.
- Εάν δηλωθεί απλά ως component με το ίδιο όνομα της οντότητας τότε η αντιστοίχιση υπονοείται.
- Ωστόσο πολλές φορές για μία οντότητα έχουμε πολλές αρχιτεκτονικές.

Για να μην χρησιμοποιείται κάποια τυχαία, καλό είναι να ορίζουμε ρητά την επιθυμητή αρχιτεκτονική.

Χρησιμοποιούμε την δήλωση "**Configuration**".

Configuration μέσα στην αρχιτεκτονική

Έστω ότι για το component “flipflop” έχουμε δημιουργήσει 2 διαφορετικές αρχιτεκτονικές sync, async:

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

entity flipflop is
        port ( clk, clr, d: in std_logic; q : out std_logic );
end flipflop;

architecture sync of flipflop is
begin
  process(CLK)
  begin
        if (CLK'event and CLK='1') then
            if (CLR='1') then q<='0'; else q<=d; end if;
        end if;
  end process;
end sync;
```

Configuration μέσα στην αρχιτεκτονική

```
architecture async of flipflop is
begin
  process(CLK,CLR)
  begin
        if (CLR='1') then q<='0';
              elsif (CLK'event and CLK='1') then q<=d; end if;
        end process;
end async;
```

- Κατά την δομική περιγραφή ενός καταχωρητή 4-bits (αρχιτεκτονική του) καθορίζουμε ρητά ποιο flip flop θα χρησιμοποιήσουμε:

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

entity reg4 is
        port (    clk, clr : in std_logic;  d : in std_logic_vector(0 to 3);
                  q : out std_logic_vector(0 to 3) );
end reg4;
```

Configuration μέσα στην αρχιτεκτονική

```
architecture struct of reg4 is
component flipflop
        port ( clk, clr, d : in std_logic; q : out std_logic );
end component;

for ff0, ff1 : flipflop use entity work.flipflop(sync);
for ff2, ff3 : flipflop use entity work.flipflop(async);

begin
  ff0 : flipflop port map ( clk => clk, clr => clr, d => d(0), q => q(0) );
  ff1 : flipflop port map ( clk => clk, clr => clr, d => d(1), q => q(1) );

  ff2 : flipflop port map ( clk => clk, clr => clr, d => d(2), q => q(2) );
  ff3 : flipflop port map ( clk => clk, clr => clr, d => d(3), q => q(3) );
end struct;
```

Sync

Async

Configuration μέσα στην αρχιτεκτονική

- Αν θέλουμε όλα να έχουν την ίδια αρχιτεκτονική χρησιμοποιούμε την λέξη ALL:

```
architecture struct of reg4 is
component flipflop
        port ( clk, clr, d : in std_logic; q : out std_logic );
end component;

for all : flipflop use entity work.flipflop(sync);

begin
  ff0 : flipflop port map ( clk => clk, clr => clr, d => d(0), q => q(0) );
  ff1 : flipflop port map ( clk => clk, clr => clr, d => d(1), q => q(1) );
  ff2 : flipflop port map ( clk => clk, clr => clr, d => d(2), q => q(2) );
  ff3 : flipflop port map ( clk => clk, clr => clr, d => d(3), q => q(3) );
end struct;
```

Sync

Configuration αυτόνομο

- Πολλές φορές θέλουμε να δημιουργούμε πολλά διαφορετικά configurations για ελέγχους παραμέτρων.
- Τότε δημιουργούμε αυτόνομα configurations (η δήλωση δεν μπαίνει μέσα σε καμία αρχιτεκτονική).
- Οι οντότητες σχεδιάζονται ανεξάρτητα από υλοποιήσεις και παρέχουν την μέγιστη δυνατή ευελιξία.
- Το επιθυμητό configuration καθορίζεται κατά την δημιουργία του test bench αρχείου όποτε και πρέπει να είναι γνωστή η διαμόρφωση που θα χρησιμοποιηθεί.

Configuration αυτόνομο

Παράδειγμα: στον καταχωρητή reg4 η αρχιτεκτονική είναι όπως πριν αλλά χωρίς τις δηλώσεις configurations

```
architecture struct of reg4 is

component flipflop
        port ( clk, clr, d : in std_logic; q : out std_logic );
end component;

begin
  ff0 : flipflop port map ( clk => clk, clr => clr, d => d(0), q => q(0) );
  ff1 : flipflop port map ( clk => clk, clr => clr, d => d(1), q => q(1) );
  ff2 : flipflop port map ( clk => clk, clr => clr, d => d(2), q => q(2) );
  ff3 : flipflop port map ( clk => clk, clr => clr, d => d(3), q => q(3) );
end struct;
```

Configuration αυτόνομο

Το επιθυμητό configuration καθορίζεται χωριστά:

```
use work.all;
configuration mixed of reg4 is
   for struct
            for ff0, ff1 : flipflop use entity work.flipflop(sync); end for;
            for ff2, ff3 : flipflop use entity work.flipflop(async); end for;
   end for;
end mixed;
```

Η χρήση του αναβάλλεται για την δημιουργία του Test bench που θα ελέγξει τον σχεδιασμό:

Configuration αυτόνομο

```
entity test_bench is
end test_bench;

architecture test of test_bench is

component reg4
        port (    clk, clr : in std_logic;  d : in std_logic_vector(0 to 3);
                  q : out std_logic_vector(0 to 3) );
end component;

for reg0: reg4 use configuration work.mixed;

signal clk, clr : std_logic;
signal d, q : std_logic_vector(0 to 3);

begin
  reg0 : reg4 port map (clk,clr,d,q);
end test;
```

Configuration αναδρομικό

Ένα configuration μπορεί να χρησιμοποιεί αναδρομικά και άλλα configurations για components.

Παράδειγμα: έστω ότι και το flip flop έχει κάποια **configuration** με τα ονόματα conf_sync και conf_async. Τότε το configuration του καταχωρητή μπορεί να γραφεί ως:

```
use work.all;
configuration mixed of reg4 is
   for struct
         for ff0, ff1 : flipflop use configuration work.conf_sync; end for;
         for ff2, ff3 : flipflop use configuration work.conf_async; end for;
   end for;
end mixed;
```

Αρχεία

- Παρέχουν ένα τρόπο διασύνδεσης μίας σχεδίασης με δεδομένα τα οποία βρίσκονται εξωτερικά της σχεδίασης.
- Οι βασικές δηλώσεις βρίσκονται στο πακέτο textio της βιβλιοθήκης std
- Η δήλωση ενός αρχείου κειμένου χρησιμοποιεί τις ακόλουθες δυο δηλώσεις του πακέτου textio

type line **is access** string

type text **is file of** sting

Δείκτης

Παράδειγμα: υλοποίηση μετρητή με χρήση αρχείου

το αρχείο περιέχει τα ακόλουθα δεδομένα, ένα σε κάθε γραμμή:

000 001 010 011 100 101 110 111

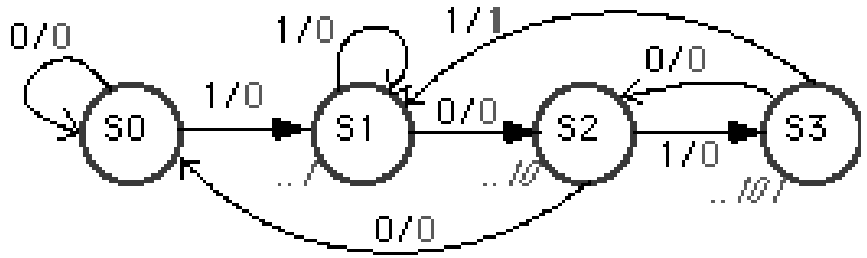
Αρχεία

```
library std;
use std.textio.all;

entity counter is
    generic (N: integer:=3);
    port ( clock: in bit;
           z: out bit_vector(N-1 downto 0));
end counter;

architecture behave of counter is
file myfile : TEXT is "cntr.txt";
begin
  process
  variable myline: Line;
  variable bv: bit_vector (N-1 downto 0);
  begin
      wait on clock until clock='1';
      while not (endfile(myfile)) loop
          readline (myfile, myline);
          read(myline, bv);
          z<=bv;
      end loop;
  end process;
end;
```

Διαβάζει μία γραμμή

Μετατρέπει το αλφαριθμητικό σε μεταβλητή

Παράδειγμα: Μηχανή Καταστάσεων



VHDL file for a sequence detector (1011) implemented as a Mealy Machine

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

entity myvhdl is
   port (CLK, RST, X: in STD_LOGIC;
         Z: out STD_LOGIC);
end;

architecture myvhdl_arch of myvhdl is
-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S1, S2, S3, S4);
signal Sreg0: Sreg0_type;
begin
```

```
begin
--concurrent signal assignments
Sreg0_machine: process (CLK)
begin
if CLK'event and CLK = '1' then
   if RST='1' then
      Sreg0 <= S1;
   else
   case Sreg0 is
      when S1 =>
         if X='0' then
            Sreg0 <= S1;
         elsif X='1' then
            Sreg0 <= S2;
         end if;
      when S2 =>
         if X='1' then
            Sreg0 <= S2;
         elsif X='0' then
            Sreg0 <= S3;
         end if;
      when S3 =>
         if X='1' then
            Sreg0 <= S4;
         elsif X='0' then
            Sreg0 <= S1;
         end if;
      when S4 =>
         if X='0' then
            Sreg0 <= S3;
         elsif X='1' then
            Sreg0 <= S2;
         end if;
      when others =>
         null;
   end case;
   end if;
end if;
end process;
-- signal assignment statements for combinatorial outputs
Z_assignment:
Z <= '0' when (Sreg0 = S1 and X='0') else
      '0' when (Sreg0 = S1 and X='1') else
      '0' when (Sreg0 = S2 and X='1') else
      '0' when (Sreg0 = S2 and X='0') else
      '0' when (Sreg0 = S3 and X='1') else
      '0' when (Sreg0 = S3 and X='0') else
      '0' when (Sreg0 = S4 and X='0') else
      '1' when (Sreg0 = S4 and X='1') else
      '1';
end mywhdl_arch;
```