dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

4ο Εργαστήριο Space Data Systems

VHDL

Τύποι Σημάτων – Δομή Process - Εντολές υπό συνθήκη

Βασιλόπουλος Διονύσης

Ε.ΔΙ.Π. Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Τύποι σημάτων

Signed-Unsigned αριθμοί

Για να χρησιμοποιήσουμε αριθμούς (προσημασμένους ή μη) πρέπει να δηλώσουμε το πακέτο/βιβλιοθήκη numeric_std με τη δήλωση

use IEEE.NUMERIC_STD.ALL;

Εάν θέλουμε να δηλώσουμε ένα σήμα ως αριθμό τότε η δήλωση

signal x: **signed**(7 downto 0);
Αναφέρεται σε **προσημασμένο αριθμό των 8 bit συμπληρώματος ως προς 2**

ενώ η δήλωση

signal x: **unsigned**(7 downto 0);
Αναφέρεται σε **μη προσημασμένο αριθμό των 8 bit**

Δηλώνονται ως διανύσματα όπως και ο τύπος std_logic_vector
Επιτρέπουν την **εκτέλεση αριθμητικών πράξεων** σε αντίθεση με τα διανύσματα std_logic_vector

VHDL – Unsigned

Πράξεις

Έστω όλα τα σήματα unsigned(3 down to 0), 4 bits
a<=“1000”; -- 8
b<=“0001”; -- 1

Οι τιμές δίνονται όπως και σε std_logic_vector

```
add<=a+b;                                          add=“1001” = 9
sub<=a-b;                                          sub=“0111” = 7
multiply<=a*b;        -- ERROR: a*b = 8 bits!
full_multiply<=a*b; -- full_multiply must have 8 bits   full_multiply=“00001000” = 8
divide<=a/b;                                       divide=“1000” = 8
modulus<=a mod b;                                  modulus=“0000” = 0
```

VHDL – Unsigned

Πράξεις

Έστω όλα τα σήματα unsigned(3 down to 0), 4 bits
a<=“0001”; -- 1
b<=“1000”; -- 8

Οι τιμές δίνονται όπως και σε std_logic_vector

```
add<=a+b;
sub<=a-b;
multiply<=a*b;        -- ERROR: a*b = 8 bits!
full_multiply<=a*b; -- full_multiply must have 8 bits
divide<=a/b;
modulus<=a mod b;
```

add=“1001” = 9
sub=“1001” = **9 (Error), a-b<0**

full_multiply=“00001000” --8
divide=“0000” -- 0 (πηλίκο)
modulus=“0001” -- 1

VHDL – Unsigned

Πρόσθεση

```
library IEEE; use IEEE.NUMERIC_STD.ALL;
….
signal a, b, s: unsigned (7 downto 0);
s<=a+b;
```

+: Παράγει αποτέλεσμα ίδιου μήκους.
Δεν ανιχνεύει υπερχείλιση
Θα μπορούσαμε με προφανώς να έχουμε και - .

Λύση
Χρειαζόμαστε ένα επιπλέον bit για το κρατούμενο (**C**arry)

```
library IEEE; use IEEE.NUMERIC_STD.ALL;
….
signal a, b, s: unsigned (7 downto 0);
signal temp_result: unsigned (7 downto 0);
signal c: std_logic;

temp_result<=('0'&a)+('0'&b);
c<=temp_result(8)
s<=temp_result(7 downto 0);
```

VHDL – Unsigned

Γινόμενο

- Μεγαλύτερο αποτέλεσμα για τελεστέους των n bit :

$$(2^n-1)(2^n-1)=2^{2n}-2^n-2^n+1=2^{2n}-\left(2^{n+1}-1\right)$$

  - Απαιτεί 2^{2n} bit για αποφυγή υπερχείλισης
  - Γινόμενο τελεστέων των n bit και m bit
    - Απαιτεί $n+m$ bit

```
signal x : unsigned(7 downto 0);
signal y : unsigned(13 downto 0);
signal p : unsigned(21 downto 0);
…
p <= x * y;
```

VHDL – Unsigned

Increments

- Απλά, πρόσθεση ή αφαίρεση του 1

```
signal x, s: unsigned(15 downto 0);
...

s <= x + 1;   -- increment x

s <= x - 1;   -- decrement x
```

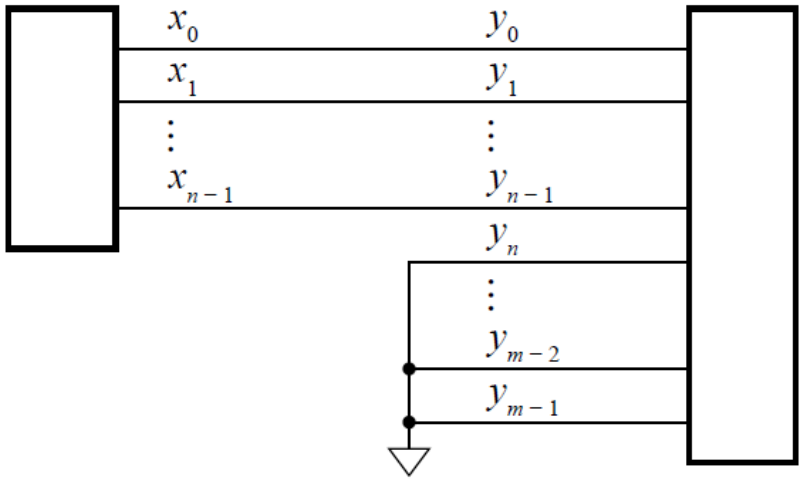
- Σημείωση: 1 (ακέραιος), όχι '1' (bit)

VHDL – Unsigned

Επέκταση μηδενός

- Για επέκταση αριθμού από *n* bit σε *m* bit
  - Πρόσθεση αρχικών bit 0
  - π.χ., 72_{10} = 1001000 = 000001001000

Σε όλες τις εντολές όσα bit είναι στο δεξί μέλος τόσο ακριβώς πρέπει να είναι και στο αριστερό




```
signal x : unsigned(3 downto 0);
signal y : unsigned(7 downto 0);

y <= "0000" & x;
```

```
y <= resize(x, 8);
```

Συνένωση - concatenation

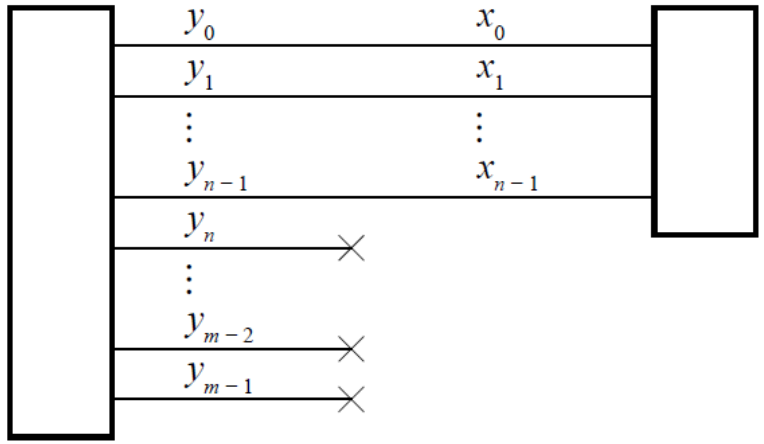
VHDL – Unisigned

Αποκοπή

Για αποκοπή από *m* bit σε *n* bit

- Απορρίπτουμε τα αριστερότερα bit
- Η τιμή διατηρείται εφόσον τα bit που απορρίπτονται είναι 0
- Το αποτέλεσμα είναι το *x* mod 2^n

Έστω: signal y : unsigned(7 downto 0);
signal x : unsigned(3 downto 0);




```
x <= y(3 downto 0);
```

```
x <= resize(y, 4);
```

Ποια είναι πιο ευέλικτη;

x <= resize(y, x'length)

Και για επέκταση και για αποκοπή.

VHDL – Unsigned

Αφαίρεση

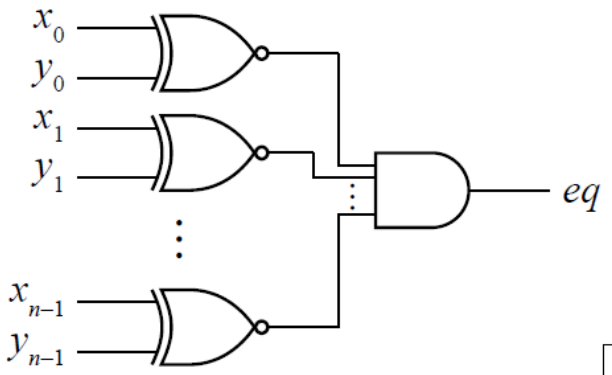
```
library ieee; use ieee.std_logic_1164.all, ieee.numeric_std.all;
entity adder_subtracter is
  port ( x, y          : in unsigned(11 downto 0);
         s             : out unsigned(11 downto 0);
         mode          : in std_logic;
         ovf_unf       : out std_logic );
end entity adder_subtracter;

architecture behavior of adder_subtracter is
signal s_tmp : unsigned(12 downto 0);
begin
  s_tmp <= ('0' & x) + ('0' & y) when mode = '0' else
           ('0' & x) - ('0' & y);
  s <= s_tmp(11 downto 0);
  ovf_unf <= s_tmp(12);
end architecture behavior;
```

Υπό συνθήκη

VHDL – Unsigned

Σύγκριση

- Πύλη XNOR: Ισότητα δύο bit
  - Εφαρμογή σε κάθε bit δύο απρόσημων αριθμών



- Στην VHDL, το x = y δίνει boolean αποτέλεσμα
  - Ψευδές ή αληθές
  - Δεν μπορεί να γίνει ανάθεση σε σήμα std_logic

boolean

τελεστής

```
eq <= '1' when x = y else '0';
```

VHDL – Signed

Πράξεις

Έστω όλα τα σήματα signed(3 down to 0), 4 bits
a<=“1000”; = -8
b<=“0001”; = 1

Οι τιμές δίνονται όπως και σε std_logic_vector

```
add<=a+b;                                   add=“1001” = -7
sub<=a-b;                                   sub=“0111” =  7 (Error a-b<-2^(n-1) = -8)
multiply<=a*b;        -- ERROR: a*b = 8 bits!
full_multiply<=a*b; -- full_multiply must have 8 bits   full_multiply=“00001000” = -8
divide<=a/b;                                divide=“1000” = -8
modulus<=a mod b;                           modulus=“0000” = 0
absolute_value<=abs(a);                     absolute_value=“1000” =-8 ((Error abs(a)>2^(n-1)-1= 7)
```

VHDL – Signed

Πράξεις

Έστω όλα τα σήματα signed(3 down to 0), 4 bits
a<=“0001”; = 1
b<=“1000”; = -8

Οι τιμές δίνονται ΜΟΝΟ ως άνυσμα (vector) όπως και σε std_logic_vector

```
add<=a+b;                                   add=“1001” = -7
sub<=a-b;                                   sub=“1001” =  -7 (Error a-b>2^(n-1)-1= 7)
multiply<=a*b;        -- ERROR: a*b = 8 bits!
full_multiply<=a*b; -- full_multiply must have 8 bits   full_multiply=“00001000” = -8
divide<=a/b;                                divide=“1000” = 0
modulus<=a mod b;                           modulus=“1001” = -7
absolute_value<=abs(a);                     absolute_value=“0001” =1
```

$$a \ rem \ b \quad = a - (a/b) * b \ \textit{(division truncates toward 0)}$$
$$a \ mod \ b \quad = a - b * floor(a/b) \ \textit{(division floors toward } -\infty)$$

VHDL – Signed

- Τύπος signed από το numeric_std

Απαραίτητη η χρήση της βιβλιοθήκης numeric_std

```
library ieee; use ieee.numeric_std.all;
...
s : signed(15 downto 0);
```

- Οι τύποι signed και unsigned είναι ξεχωριστοί

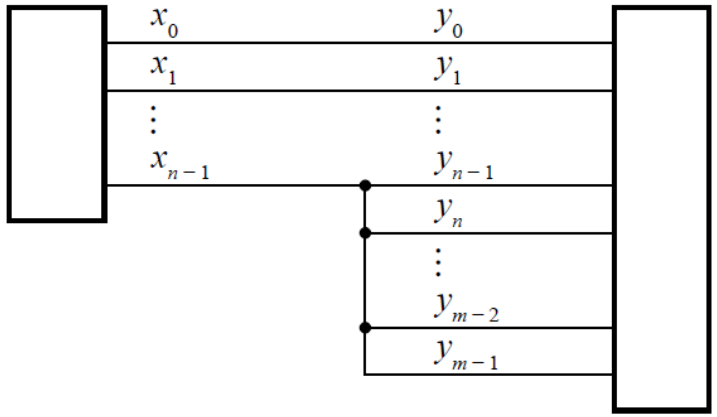
```
signal s1 : unsigned(11 downto 0);
signal s2 : signed(11 downto 0);
...
s1 <= s2; -- illegal
```

```
s1 <= unsigned(s2); -- s2 is known to be non-negative
...
s2 <= signed(s1); -- s1 is known to be less than 2**11
```

VHDL – Signed

Αλλαγή μεγέθους

- Γενικά, για ακεραίους σε συμπλήρωμα ως προς 2
  - Επέκταση με επανάληψη του bit προσήμου
    - *Επέκταση προσήμου*
  - Αποκοπή με απόρριψη αρχικών bit
    - Τα bit που απορρίπτονται πρέπει όλα να είναι τα ίδια, και ίδια με το bit προσήμου του αποτελέσματος

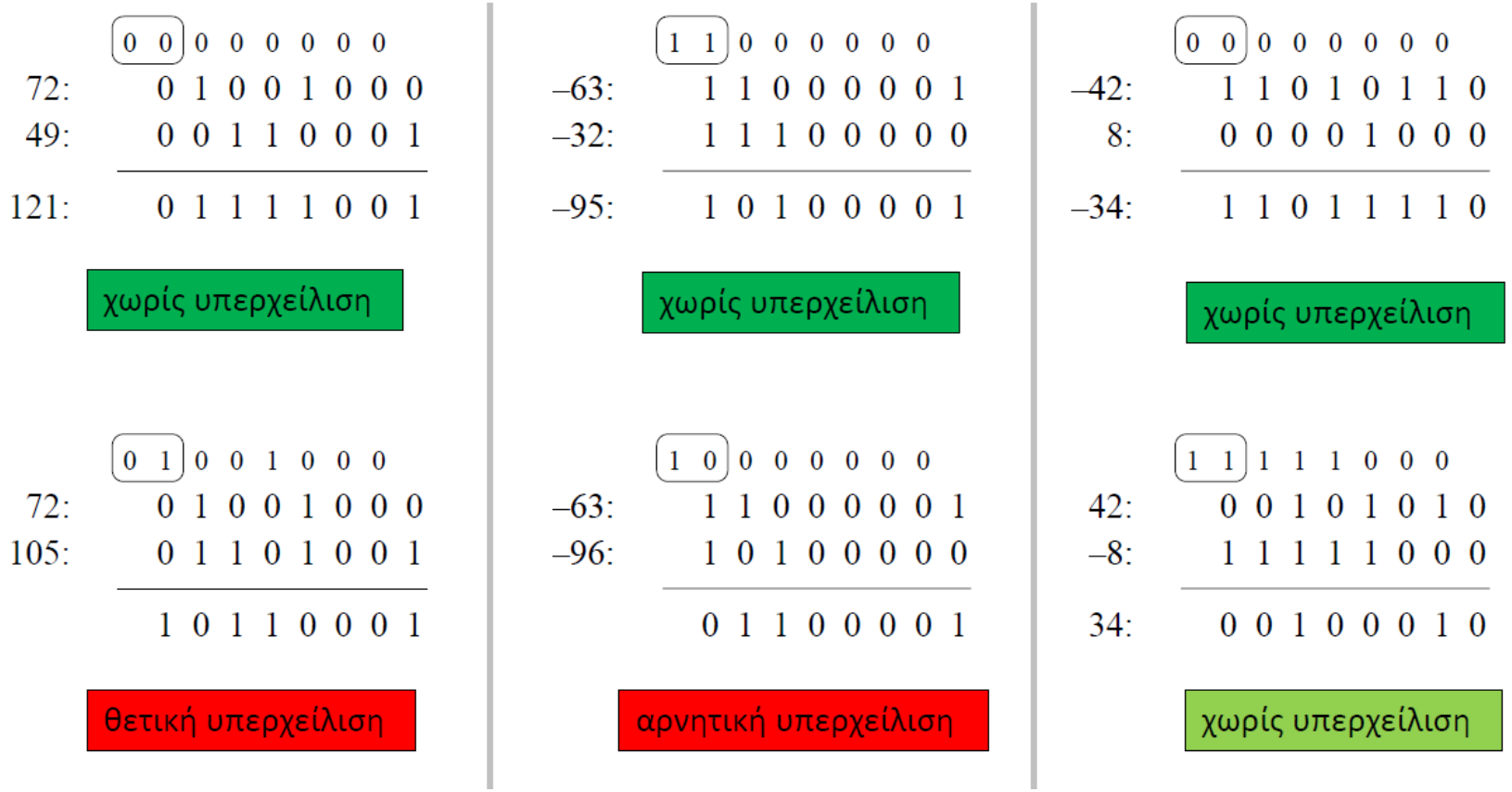



```
signal x : signed (7 downto 0);
signal y : signed (15 downto 0);
...
y <= resize(x, y'length);
...
x <= resize(y, x'length);
```

Όχι πλέον σταθερή τιμή

VHDL – Signed

Πρόσθεση



VHDL – Signed

Πρόσθεση

- Το αποτέλεσμα του + έχει ίδιο μέγεθος με τους τελεστέους

ισχύει και για unsigned

```
signal v1, v2 : signed(11 downto 0);
signal sum : signed(12 downto 0);
...
sum <= resize(v1, sum'length) + resize(v2 sum'length);
```

- Για έλεγχο υπερχείλισης, σύγκριση προσήμων

```
signal x, y, z: signed(7 downto 0);
signal ovf : std_logic;
...
z <= x + y;
ovf <= (not x(7) and not y(7) and z(7))
       or (x(7) and y(7) and not z(7));
```

VHDL – Signed

Signed: Πολλαπλασιασμός/Διαίρεση

- Πολλαπλασιασμός με 2^k
  - Αριστερή λογική ολίσθηση (όπως για απρόσημους)

Το πρόσημο πρέπει να παραμείνει

- Διαίρεση με 2^k
  - *Δεξιά αριθμητική ολίσθηση*
  - Απόρριψη των k λιγότερο σημαντικών bit, και εισαγωγή k αντιγράφων του bit προσήμου στο περισσότερο σημαντικό άκρο
  - π.χ., s = "11110011" -- –13
    shift_right(s, 2) = "11111100" -- –13 / 2^2

VHDL – Package:numeric_std

Τύποι - Συναρτήσεις

Τύποι
signed – unsigned

Συναρτήσεις – Πράξεις
+, -, *, /, mod, rem, abs

Συναρτήσεις – Σύγκριση
>, <, <=, >=, /= (όλες επιστέφουν Boolean)

Συναρτήσεις – Edge detection
rising_edge(), falling_edge()

VHDL – Package:numeric_std

Τύποι - Συναρτήσεις

Συναρτήσεις – Μετατροπές (conversion functions)
(Το αρχικό σήμα αλλάζει σε νέο)

Name	Input type	Return type
To_integer()	unsigned	integer
To_integer()	signed	integer
To_unsigned()	integer, <size>	unsigned (size-1 downto 0)
To_signed()	integer, <size>	signed (size-1 downto 0)

VHDL – Package:numeric_std

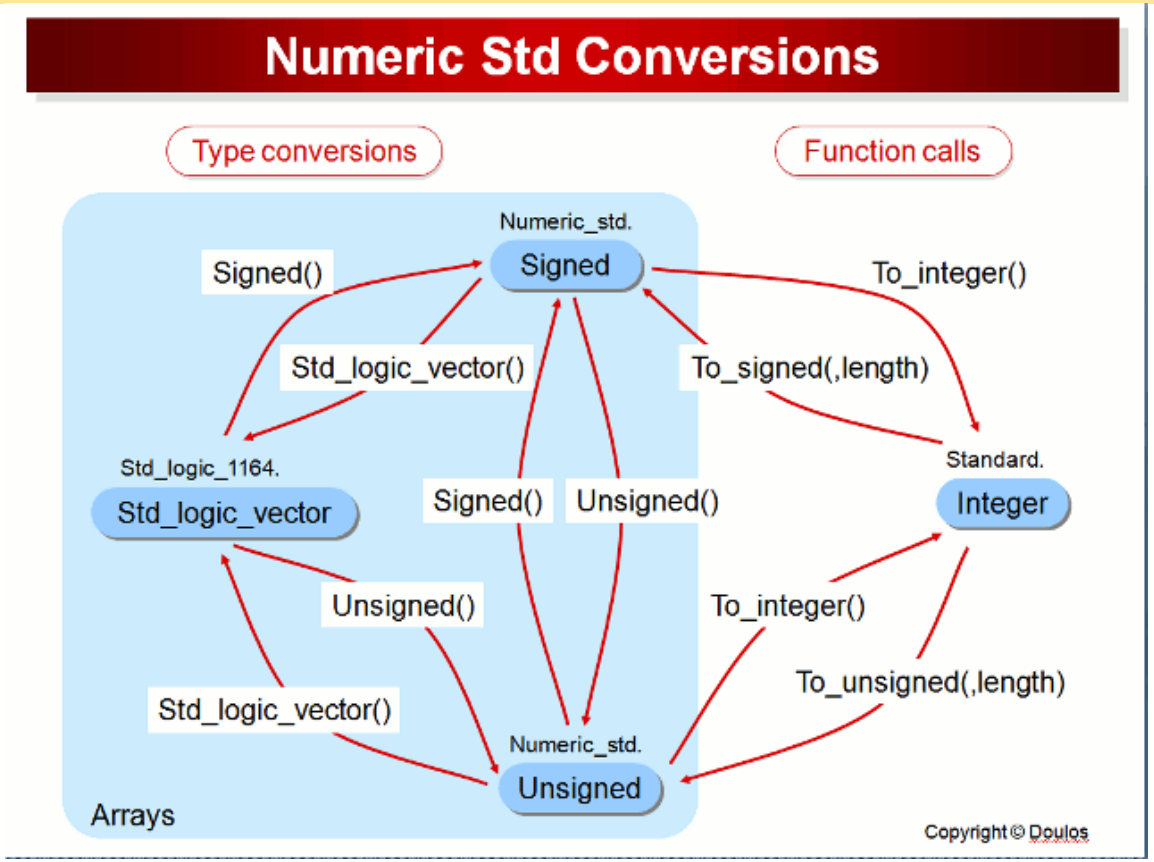
Τύποι - Συναρτήσεις

Συναρτήσεις – Μετατροπές (casting functions)
Το αρχικό σήμα δεν αλλάζει, αλλάζει μόνο ο τύπος του

Name	Input type	Return type
std_logic_vector()	unsigned	std_logic_vector
std_logic_vector()	signed	std_logic_vector
unsigned()	std_logic_vector	unsigned
signed()	std_logic_vector	signed

VHDL – Τύποι σημάτων

Μετατροπές



VHDL – Τύποι σημάτων

Μετατροπές

Έστω οι δηλώσεις:
signal x: std_logic_vector(3 downto 0);
signal y: unsigned(3 downto 0);
signal z: signed(3 downto 0);

Για να μετατρέψουμε τον ένα τύπο στον άλλο:
x<=std_logic_vector(y);
ή
y<=unsigned(x);
ή
z<=signed(x);
ή
z<=signed(y);

Έστω οι δηλώσεις:
signal x: std_logic_vector(5 downto 0);
signal y: unsigned(3 downto 0);

Για να μετατρέψουμε τον ένα τύπο στον άλλο:
x<=std_logic_vector(resize(y,x'length));
ή
y<=unsigned(x(3 downto 0));

σωστό π.χ. και το y<=unsigned(x(4 downto 1));

Όσα bit είναι αριστερά μιας εντολής, τόσα πρέπει να είναι και δεξιά

VHDL – Τύποι σημάτων

Μετατροπές

Έστω οι δηλώσεις:
signal x: std_logic_vector(3 downto 0);
signal y: unsigned(3 downto 0);
signal z: signed(3 downto 0);

Για να μετατρέψουμε τον ένα τύπο στον άλλο:
x<=std_logic_vector(y);
ή
y<=unsigned(x);
ή
z<=signed(x);
ή
z<=signed(y);

Έστω οι δηλώσεις:
signal x: std_logic_vector(5 downto 0);
signal y: unsigned(3 downto 0);

Για να μετατρέψουμε τον ένα τύπο στον άλλο:

x<=std_logic_vector(resize(y,x'length));
ή
y<=unsigned(x(3 downto 0));

σωστό π.χ. και το y<=unsigned(x(4 downto 1));

Όσα bit είναι αριστερά μιας εντολής, τόσα πρέπει να είναι και δεξιά

VHDL – Τύποι σημάτων

Περίπτωση τιμής ‘X’

Έστω η δήλωση:
signal x: std_logic_vector(3 downto 0);

και στην αρχιτεκτονική υπάρχουν ταυτόχρονα και οι δύο εντολές:

x<=“0101”;
x<=“0100”;

Τελικά το x έχει αποτέλεσμα

x=“010**X**”

VHDL – Τύποι σημάτων (1/3)

Οι ακόλουθοι τύποι υποστηρίζονται εξ' ορισμού στην VHDL.

- bit : τιμές MONO '0' και '1' {0,1}
- Boolean: τιμές Αληθής/Ψευδής – 'true'/'false' {true, false}
- Character: ASCII character (είναι 256 σε πλήθος}

Enumerated type: Συγκεκριμένες τιμές

- bit_vector: κατ' αναλογία με std_logic_vector
- String: array of characters

Vector type: Συγκεκριμένες τιμές

Υπάρχουν και άλλα Type/Subtypes είτε στη standard βιβλιοθήκη είτε σε άλλες (ieee.numeric_std.all: signed/unsigned, ieee.std_logic_1164: std_logic, std_logic_vector)

VHDL – Packages

Πακέτα της VHDL

<u>ΔΕΝ</u> τα χρησιμοποιούμε στο μάθημα

numeric_std_unsigned
numeric_bit
numeric_bit_unsigned

std_logic_arith/unsigned/signed ← Legacy packages

Τα χρησιμοποιούμε μόνο σε testbenches.
ΔΕΝ είναι synthesizable

math_real
math_complex

VHDL – Τύποι σημάτων (2/3)

Range Types

- Real: -1.7e38 έως +1.7e38
- integer: -2^{31} έως $2^{31}-1$ (signed binary number range: $[-2^{n-1}, 2^{n-1}-1]$, 32bit word arch.)
  - integer range 0 to 1000
- natural: 0 έως $2^{31}-1$
- positive: 1 έως $2^{31}-1$

Subtypes: Υποτύποι του integer

Physical Types

time: range $[-2^{n-1}, 2^{n-1}-1]$ (units: fs = 10^{-15} (base unit), ps=10^{-12}, ns=10^{-9}, μs=10^{-6}, ms=10^{-3}, s, min, h)

VHDL – Τύποι σημάτων (3/3)

User Defined

type name is (value1, value2, value3, …)

example

type traffic_light is (red, orange, green) -- Μπορούμε να επιλέξουμε κωδικοποίηση (binary, onehot, …)

signal traffic: traffic_light;

Array type

type name is array(<range>) of <element_type> -- range (max .. min/downto .. to). Το array μπορεί να χρησιμοποιηθεί, μόνο σε συνάρτηση με το type.

example

type stdvector is array (0 to 7) of std_logic_vector(3 downto 0);

signal my_stdvector_array: stdvector;

Subtypes

subtype name is <type> range <min> to <max>

example

subtype small_int is integer range 0 to 15;

signal count : small_int := 0;

VHDL – Χαρακτηριστικά (attributes) σημάτων

attributes: Παρέχουν επιπλέον πληροφορίες για το σήμα, πέραν της τιμής του.
Με τη χρήση **signal'attribute_name** έχουμε πρόσβαση στην αντίστοιχη τιμή
Κάθε attribute επιστρέφει μια τιμή ενός συγκεκριμένου τύπου.

Attribute	Information returned	Type returned
A'event	True when signal A changes, false otherwise	boolean
A'active	True when an assignment is made to A, false otherwise	boolean
A'last_event	Time when signal A last changed	time
A'last_active	Time when signal A was last assigned to	time
A'last_value	The previous value of A	same type as A

std_logic

Attribute	Information returned	Type returned
B'length	Size of the vector (e.g., 8)	integer
B'left	Left bound of the vector (e.g., 7)	integer
B'right	Right bound of the vector (e.g., 0)	integer
B'range	Range of the vector "(7 downto 0)"	string

std_logic_vector

```
for i in b_tb'range loop
   -- Do something with b_tb(i)
end loop;
```

VHDL – Process

- **process**: η διεργασία είναι μία ομάδα από εντολές που εκτελούνται ακολουθιακά μετά από κάποιο γεγονός (event: αλλαγή τιμής σήματος)

Sensitivity list

```
architecture arch_name of entity_name is
begin
  label: process (signal_name, ..., signal_name)
    variable variable_name: variable_type;
  begin
    sequential_statement;
    ...
    sequential_statement;
  end process;
end arch_name;
```

VHDL – Process

Variables (Μεταβλητές)

variable_name: το όνομα της μεταβλητής
(εάν είναι πολλές μεταβλητές χωρίζονται με κόμμα)

- *μέσα στις διεργασίες ορίζονται τοπικές μεταβλητές και ΟΧΙ εσωτερικά σήματα*
- *στις δηλώσεις μεταβλητών (μετά το variable) προσδιορίζονται μεταβλητές που μπορεί να μην έχουν τη φυσική σημασία του σήματος*

variable_type: ο τύπος της μεταβλητής (STD_LOGIC)

VHDL – Process

Variables (Μεταβλητές)

```
variable my_number: natural;

variable my_logic: std_logic:='1';

variable my_vector :  std_logic_vector(3 downto 0):= "1110";
```

Αρχικοποίηση τιμών. Μπορεί να γίνει και σε ΟΛΑ τα σήματα.

Σε process, function, procedure …

https://nandland.com/variables-vs-signals/

VHDL – Process

Variables (Μεταβλητές)

```
variable_name := expression;
```

ΠΡΟΣΟΧΗ
Είναι **:=** και όχι **<=**

- **variable_name**: το όνομα της μεταβλητής
- **expression**: έκφραση με σήματα, μεταβλητές και τελεστές
  - στις ακολουθιακές (sequential) εντολές ανάθεσης μεταβλητής :
    - στην έκφραση προσδιορίζονται **σήματα**, που ανήκουν ή δεν ανήκουν στη λίστα ευαισθησίας, και **μεταβλητές** που δηλώνονται κατά τη δήλωση μεταβλητών
    - στο αριστερό μέρος της εντολής προσδιορίζεται **μεταβλητή** που δηλώνεται κατά τη δήλωση των μεταβλητών

VHDL – Process

Variables (Μεταβλητές)

- **Διαφορά μεταβλητής και σήματος μέσα σε μία διεργασία**
  - η μεταβλητή παίρνει νέα τιμή **στιγμιαία** με τον τελεστή ανάθεσης **:=**, αμέσως μόλις εκτελεστεί η αντίστοιχη εντολή μέσα στη διεργασία
  - σε αντίθεση, το σήμα παίρνει νέα τιμή **με καθυστέρηση δέλτα δ_{delay}**, με τον τελεστή ανάθεσης **<=**, στο **τέλος** της εκτέλεσης της διεργασίας

Η χρήση των μεταβλητών μειώνει σημαντικά το χρόνο της προσομοίωσης

https://nandland.com/variables-vs-signals/

VHDL – Process

signal_name: το όνομα του σήματος
(εάν είναι πολλά σήματα χωρίζονται με κόμμα)

- *στις δηλώσεις των σημάτων (μετά το process) που απαρτίζουν τη λίστα ευαισθησίας (sensitivity list) το σήμα είναι είσοδος της υπομονάδας που δηλώνεται κατά τη δήλωση των διαύλων της οντότητας ή εσωτερική διασύνδεση της υπομονάδας*
- *κάθε αλλαγή τιμής σήματος εισόδου που ανήκει στη λίστα ευαισθησίας οδηγεί στην ακολουθιακή εκτέλεση των εντολών της διεργασίας μία φορά*
- *εάν περισσότερες από μία εντολές αναθέτουν τιμή σε κάποιο σήμα λαμβάνεται υπόψη μόνο η τελευταία ακολουθιακή εντολή*
- *Προσοχή στη δήλωση των σημάτων στη λίστα ευαισθησίας*
  - μία ελλιπής δήλωση σημάτων στη λίστα ευαισθησίας θα οδηγήσει είτε σε μη σωστή σύνθεση, είτε σε ασυμφωνία της προσομοίωσης πριν και μετά τη σύνθεση και την υλοποίηση

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Dataflow)

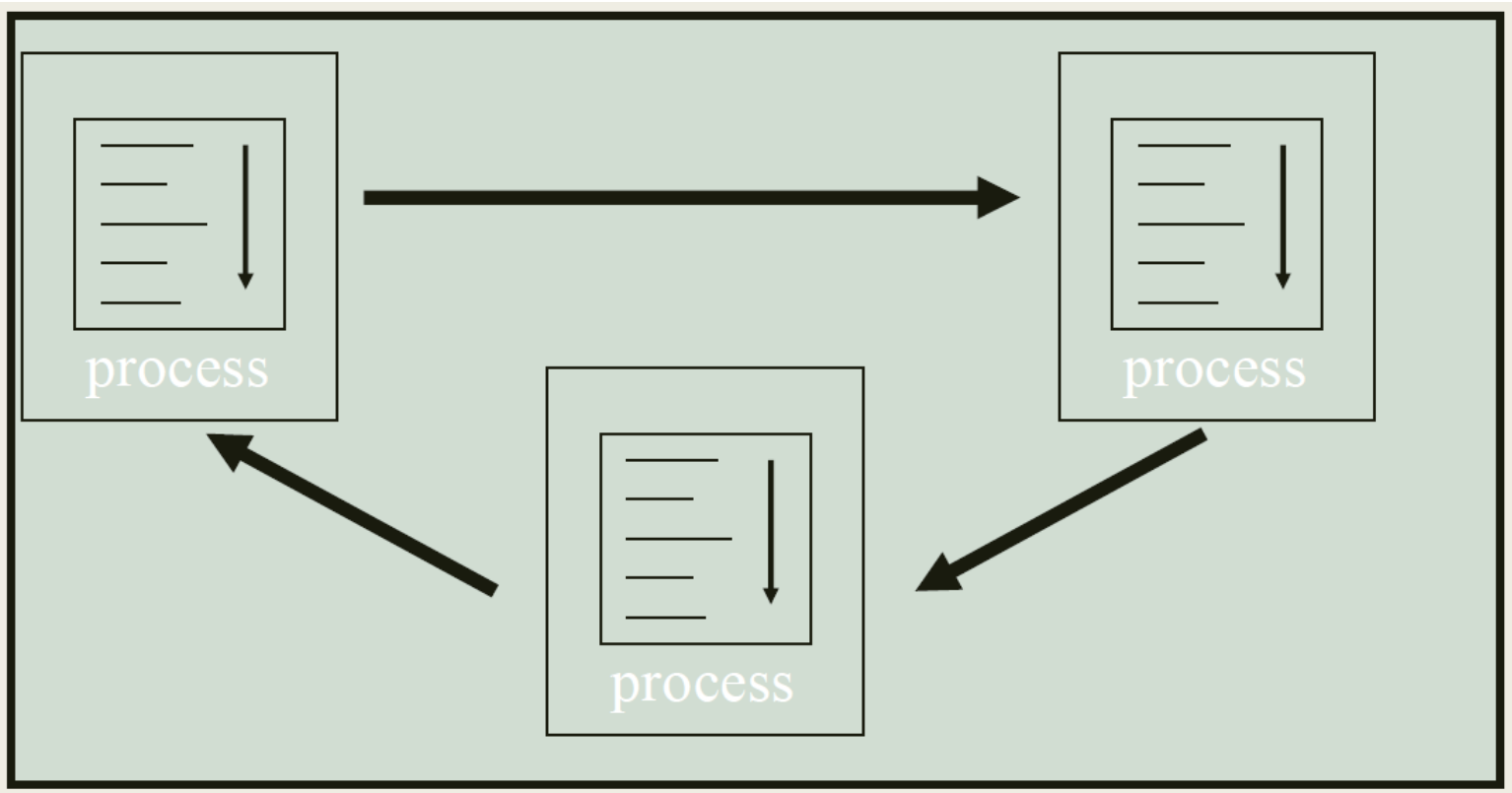
```
architecture Dataflow of CR_AC is
begin

        AirCond_1<=Sensor_1 or Sensor_2;
        AirCond_2<=Sensor_1 and Sensor_2;

end architecture Dataflow;
```

VHDL - Παράδειγμα

Υλοποίηση Αρχιτεκτονικής (Behavioral)

```
architecture behavioral of CR_AC is
begin

room:  process (Sensor_1, Sensor_2) is
          begin
               AirCond_1<=Sensor_1 or Sensor_2;
               AirCond_2<=Sensor_1 and Sensor_2;
          end process room;

end architecture Behavioral;
```

VHDL – Process



Κάθε διεργασία (process) εκτελεί τις εντολές της ακολουθιακά, ενώ πολλές διεργασίες μαζί αλληλεπιδρούν ταυτόχρονα. Επίσης, ταυτόχρονα αλληλεπιδρούν εντολές ταυτόχρονης ανάθεσης και διεργασίες.

VHDL

Υλοποίηση Αρχιτεκτονικής (Behavioral)

1. Όμοιο RTL Design με το Dataflow
2. ΔΕΝ χρειάζεται να αλλάξουμε τίποτα στην προσομοίωση
3. Εισαγωγή της δομής process (με λίστα ευαισθησίας)
4. Σε περίπτωση που υπάρχει μόνο η εντολή process, τότε ουσιαστικά οι εντολές μας εκτελούνται σειριακά
5. Σε περίπτωση που υπάρχουν και απλές εντολές ανάθεσης στην αρχιτεκτονική, τότε μπορείτε να θεωρήσετε όλη τη δομή process σαν μία εντολή που εκτελείται παράλληλα με τις εντολές ανάθεσης (που είναι εκτός process)

VHDL – Process

Διαφορά στην εφαρμογή της τιμής μίας sequential εντολής ανάθεσης μεταβλητής ή σήματος μέσα σε μία διεργασία :

– η **μεταβλητή παίρνει νέα τιμή άμεσα** με τον τελεστή ανάθεσης :=, αμέσως μόλις εκτελεστεί η αντίστοιχη εντολή μέσα στη διεργασία

– σε αντίθεση, **το σήμα παίρνει νέα τιμή με καθυστέρηση** δέλτα δ_{delay}, με τον τελεστή ανάθεσης <=, στο τέλος της εκτέλεσης της διεργασίας

– το σήμα θυμάται την τιμή του μέχρι να φτάσει το τέλος της εκτέλεσης της διεργασίας και να λάβει μία νέα τιμή

VHDL – Process

- Οι τιμές αναθέτονται στα σήματα είτε κάθε φορά που συναντάται η εντολή **wait** μέσα στο process είτε όταν φτάνουμε στο **end** του process.
- Ανάθεση **νέας τιμής** σε ένα **σήμα** μπορεί να γίνει μόνο μία φορά σε κάθε εκτέλεση του process.
- Οι εντολές ανάθεσης τιμών στα σήματα εκτελούνται με τη σειρά που εμφανίζονται μέσα στο process

Οπότε εάν υπάρχουν πολλές αναθέσεις τιμών σε ένα σήμα (άρα πολλές εντολές μέσα στο process που δίνουν τιμή στο σήμα), τότε το σήμα παίρνει την τιμή της τελευταίας ανάθεσης και οι ενδιάμεσες εντολές ανάθεσης στο σήμα, αγνοούνται.

Όταν υπάρχει sensitivity list ΔΕΝ μπορώ να έχω και wait στο process

VHDL – Process

- Οι εντολές μέσα στο process **εκτελούνται τουλάχιστον 1 φορά**, ακόμα και εάν δεν υπάρχει sensitivity list.

- Όταν **δεν υπάρχει sensitivity list** οι εντολές μέσα στο process **εκτελούνται συνέχεια**. Θα πρέπει να υπάρχει **εντολή wait** ώστε να ανατεθούν τιμές στα σήματα (simulation).

- Εάν υπάρχει wait σε σήμα που δεν αλλάζει τιμή (εντός του process) τότε η εκτέλεση σταματάει στο συγκεκριμένο wait.

- Όλο το **process θεωρείται ΜΙΑ εντολή μέσα σε μια αρχιτεκτονική** και εκτελείται παράλληλα με πιθανές άλλες εντολές της.

VHDL – Process – Wait statement

```
test:process (a,b) is
begin
....
end process test;
```

```
test:process  is
begin
....
wait on a,b;
end process test;
```

Δεν το γράφουμε αλλά ισχύει

```
test:process (a,b) is
begin
....
end process test;
```

```
test:process (a,b) is
begin
....
wait on c;
end process test;
```

Το process «κολλάει» σε αυτή την εντολή

VHDL – Process – Wait statement

```
test:process  is
begin
....
wait until (condition);
…….
end process test;
```

Το process εκτελείται άμεσα και συνεχώς αλλά για να περάσει από το wait πρέπει να ικανοποιείται η συνθήκη (δηλαδή condition=true)

Το process μπορεί να έχει και λίστα ευαισθησίας

Περίληψη

- Τύποι σημάτων Unsigned/Signed
- Μετατροπές τύπων
- Attributes
- Process
- Variable
- Διαβάζετε τις παραγράφους 2.4, 3.1 - 3.2, 4.5.4 (θεωρία και VHDL) από Ashenden
- Διαβάζετε τις παραγράφους 1.4, 1.5, 2.2-2.5, 4.2.2 - 4.2.3, 4.2.7, 4.5.4 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.
- Διαβάζετε τις παραγράφους 3.5.4, 4.6.6, 4.6.8, 12.2.6 - 12.2.8, 12.2.11 - 12.2.12, 12.9.1, 12.9.5 - 12.9.7, 12.10.3 από το βιβλίο των Brown-Vranesic.
- Συνοπτικός οδηγός VHDL: https://redirect.cs.umbc.edu/portal/help/VHDL/summary_one.html