dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

7ο Εργαστήριο Σχεδίασης Ψηφιακών Συστημάτων

5η Διάλεξη

Εντολές υπό συνθήκη – Πολυπλέκτες – Τελεστές Structural Αρχιτεκτονική

Βασιλόπουλος Διονύσης

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

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Selected Assignment

Example 1

Ένα σύστημα δέχεται στην είσοδο το σήμα a των 2 bit και στην έξοδο υπάρχει το σήμα b των 3 bit που αντιστοιχεί στη μετάδοση του σήματος a με επιπλέον bit άρτιας ισοτιμίας (b=parity_bit&a).

VHDL – Selected assignment (Dataflow)

Example 1: With Select

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;USE ieee.numeric_std.ALL;



entity selected is
port (a  :   in   std_logic_vector(1 downto 0);
      b  :  out std_logic_vector(2 downto 0)
);

end entity selected;
```

```
architecture Dataflow of selected is

begin

with a select
b<="000" when "00",
     "101" when "01",
     "110" when "10",
     "011" when "11",
     "000" when others;

end architecture Dataflow ;
```

ΔΕΝ ΥΠΑΡΧΕΙ ΠΡΟΤΕΡΑΙΟΤΗΤΑ

Λογική Συνθήκη

Ανάθεση με επιλογή
(διακριτές τιμές συγκεκριμένου σήματος)

ΜΙΑ εντολή: Ανάθεση **ΜΙΑΣ τιμής** σε ένα σήμα

Πρέπει να ελέγχω ΟΛΕΣ τις τιμές

VHDL – Selected assignment (Behavioral)

Example 1: Case

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;USE ieee.numeric_std.ALL;



entity selected is
port (a  :   in   std_logic_vector(1 downto 0);
        b  :  out std_logic_vector(2 downto 0)
);

end entity selected;
```

```
architecture Behavioral of selected is

begin

test: process (a) is

begin

case a is
when “00” => b<=“000";
when “01” => b<=“101";
when “10” => b<=“110";
when “11” => b<=“011";
when others => b<=“000";
end case;
end process test;
end architecture Behavioral;
```

Μπορεί να υπάρχουν πολλές εντολές ανά περίπτωση (when) που χωρίζονται με ;

case
Διακριτές τιμές

ΔΕΝ ΥΠΑΡΧΕΙ ΠΡΟΤΕΡΑΙΟΤΗΤΑ

Πρέπει να ελέγχω ΟΛΕΣ τις τιμές

Το when μπορεί να ελέγχει πάνω από μία τιμή π.χ. when 0|1=>…

VHDL –Conditional assignment

Example 2

Σε ένα Computer Room υπάρχουν 3 κλιματιστικά και ένας αισθητήρας θερμοκρασίας (θερμόμετρο). Εάν το θερμόμετρο δείξει θερμοκρασία κάτω των 25 βαθμών τότε δεν λειτουργεί το κλιματιστικό. Αν η θερμοκρασία είναι μέχρι 30 βαθμούς λειτουργεί το 1ο. Αν η θερμοκρασία είναι μέχρι 45 βαθμούς λειτουργεί και το 2ο. Εάν είναι πάνω από 45 λειτουργεί και το 3ο. Θεωρείστε ότι η σειρά λειτουργίας των κλιματιστικών είναι σαφώς ορισμένη και δεν μας απασχολεί για το πρόβλημά μας. Είσοδος του συστήματος το σήμα temperature και έξοδος το σήμα action (έχει 4 δυνατές τιμές).

VHDL – Conditional assignment (Dataflow)

Example 2: When

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;USE ieee.numeric_std.ALL;

entity conditional is
port (temperature:  in std_logic_vector(5 downto 0);
        action          :  out std_logic_vector(1 downto 0)
);
end entity conditional;

architecture Dataflow of conditional is
signal temp :unsigned(5 downto 0);
begin

temp<= unsigned(temperature);
action<="00" when temp<25 else
      "01" when (temp>=25 and temp<30) else
      "10"  when (temp>=30 and temp<45) else
      "11";


end architecture Dataflow ;
```

max=2^5-1=63

action:
00: Δεν δουλεύει τίποτα
01: Δουλεύει 1 AC
10: Δουλεύουν 2 AC
11: Δουλεύουν και τα 3 AC

ΥΠΑΡΧΕΙ ΠΡΟΤΕΡΑΙΟΤΗΤΑ

Οποιαδήποτε Λογική Συνθήκη

Μία εντολή: Ανάθεση τιμής σε ένα σήμα

VHDL – Conditional assignment (Behavioral)

Example 2: If

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;USE ieee.numeric_std.ALL;

entity conditional is
port (temperature:  in std_logic_vector(5 downto 0);
        action          :   out std_logic_vector(1 downto 0)
);
end entity conditional;

architecture Behavioral of conditional is
signal temp :unsigned(5 downto 0);
```

ΥΠΑΡΧΕΙ ΠΡΟΤΕΡΑΙΟΤΗΤΑ

```
begin
temp<= unsigned(temperature);

action_temp: process(temp) is
begin
if (temp<25) then
   action<="00";
elsif (temp<30) then
   action<="01";
elsif (temp<45) then
   action<="10";
else
   action<="11";
end if;
end process action_temp;

end architecture Behavioral;
```

Οποιαδήποτε Λογική Συνθήκη

Μπορεί και πολλαπλές εντολές ανά περίπτωση (if/elsif/else)

if, **elsif**, else
Μόνο μέσα
σε process

Πρέπει να
ελέγχω ΟΛΕΣ
τις τιμές

VHDL – Selected & Conditional assignment

Σύνοψη

1.
signal <= value when condition else …

2.
with signal_1 select
signal_2<=value when (discrete signal_1 value),
…

3.
If condition then action;elsif … else end if

4.
case signal is
when value => assignment (discrete signal value),
…
end case;

ΠΡΟΣΟΧΗ: Εάν δεν ελέγξετε όλες τις περιπτώσεις τότε δημιουργούμε latches που είναι ανεπιθύμητο

Και στις 2 περιπτώσεις είμαστε απευθείας στο σώμα της αρχιτεκτονικής και η εντολή αφορά απόδοσης τιμής σε ένα συγκεκριμένο σήμα

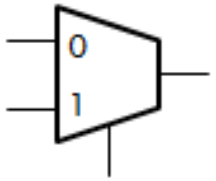
Και στις 2 περιπτώσεις είμαστε στο σώμα process (ή procedure) και αφορά την εκτέλεση μίας η περισσότερων εντολών ανάλογα τη συνθήκη, και μπορεί να αφορά πάνω από ένα σήματα.

http://www.pldworld.com/_hdl/2/_ref/acc-eda/language_overview/concurrent_statements/conditional_vs__selected_assignment.htm
https://insights.sigasi.com/tech/signal-assignments-vhdl-withselect-whenelse-and-case/
https://nandland.com/how-to-avoid-creating-a-latch/

VHDL – Multiplexer

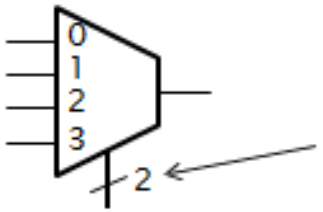
- Επιλέγει μεταξύ εισόδων δεδομένων με βάση μια είσοδο επιλογής

2-σε-1 πολυπλέκτης



sel	z
0	a_0
1	a_1

4-σε-1 πολυπλέκτης



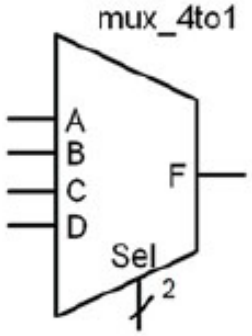
δύο bit επιλογής

sel	z
00	a_0
01	a_1
10	a_2
11	a_3

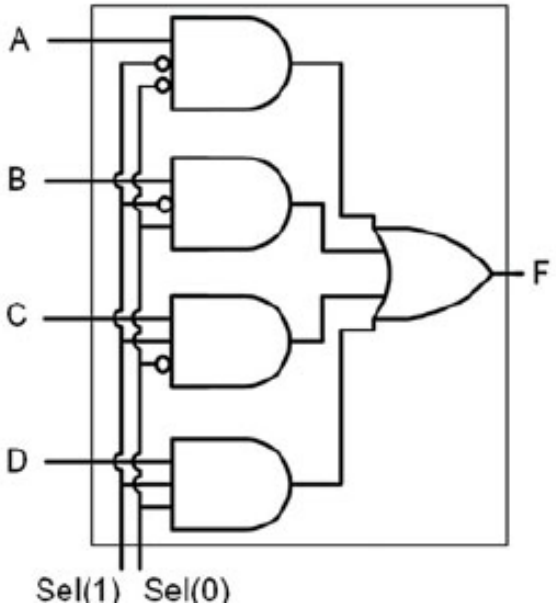
- Ένας N-σε-1 πολυπλέκτης απαιτεί $\lceil \log_2 N \rceil$ bit επιλογής

VHDL – Multiplexer

The symbol and truth table for a 4-to-1 multiplexer are shown below. This can be implemented using a simple sum of products form based on the identity theorem and the appropriate inversions of the select line.



Sel	F
"00"	A
"01"	B
"10"	C
"11"	D

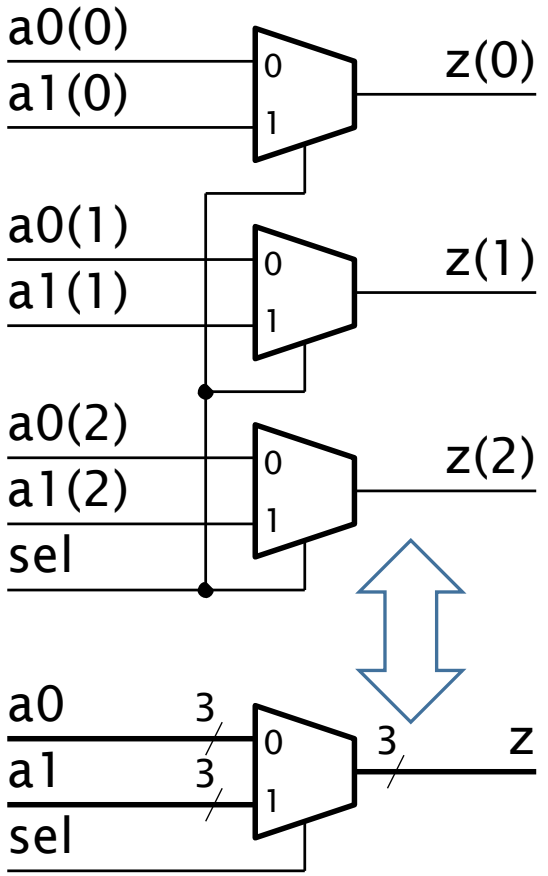


The following shows how to model the behavior of the mux using concurrent signal assignments with logical operators.

```
architecture mux_4to1_arch of mux_4to1 is

 begin

  F <= (A and not Sel(0) and not Sel(1)) or
       (B and not Sel(0) and     Sel(1)) or
       (C and     Sel(0) and not Sel(1)) or
       (D and     Sel(0) and     Sel(1));

end architecture;
```

VHDL – Multiplexer

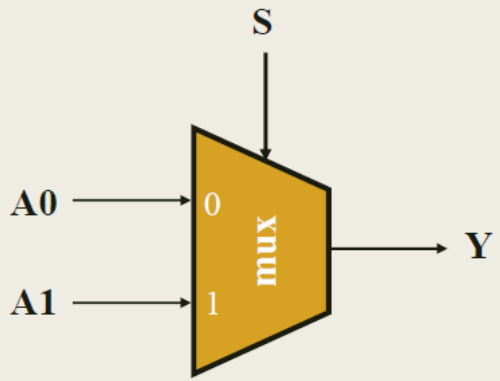
- Για να επιλέξουμε μεταξύ *N* κωδικών λέξεων των *m* bit
  - Συνδέουμε παράλληλα *m* πολυπλέκτες των *N* εισόδων



VHDL – Multiplexer

Entity (2to1)

```
entity MUX_2_to_1 is
    port (
      A0: in STD_LOGIC;
      A1: in STD_LOGIC;
      S: in STD_LOGIC;
      Y: out STD_LOGIC);
end MUX_2_to_1;
```



VHDL – Multiplexer

Architecture (2to1 - behavioral)

Η αρχιτεκτονική του πολυπλέκτη 2 σε 1 στη VHDL
Περιγραφή συμπεριφοράς

```
architecture BEHAVIORAL of MUX_2_to_1 is
begin
  process (A0, A1, S)
  begin
    if (S = '0') then
      Y <= A0;
    else
      Y <= A1;
    end if;
  end process;
end BEHAVIORAL;
```

Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες
οι είσοδοι του συνδυαστικού κυκλώματος

y<=A0 when s='0' else A1;

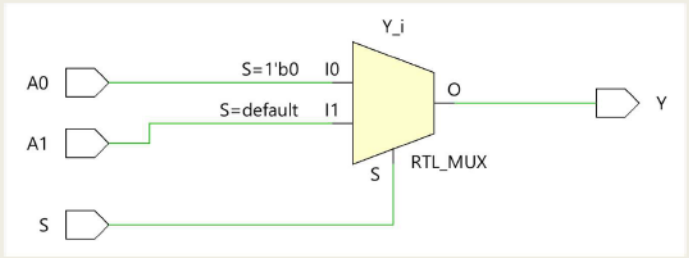
VHDL – Multiplexer

Architecture (2to1 - behavioral)

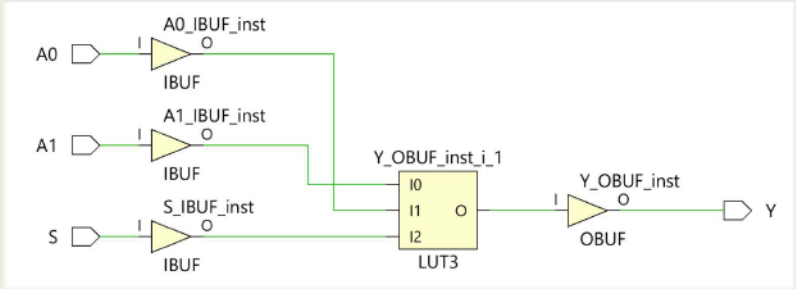
Η αρχιτεκτονική του πολυπλέκτη 2 σε 1 στη VHDL

Περιγραφή συμπεριφοράς

- Σχηματικό διάγραμμα RTL



- Σχηματικό διάγραμμα σε τεχνολογία FPGA

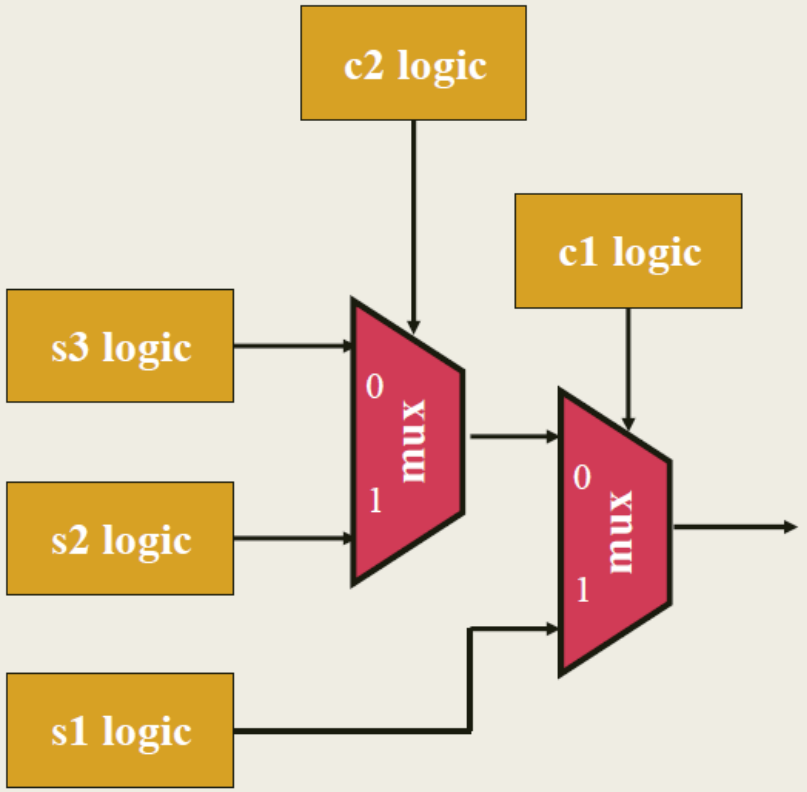


VHDL – Multiplexer

Υλοποίηση If

- Υλοποίηση εντολής IF με τη χρήση πολυπλεκτών 2 σε 1

```
if c1 then
    s1;
elsif c2 then
    s2;
else
    s3;
end if;
```

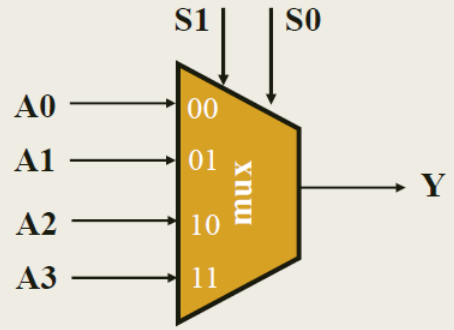


VHDL – Multiplexer

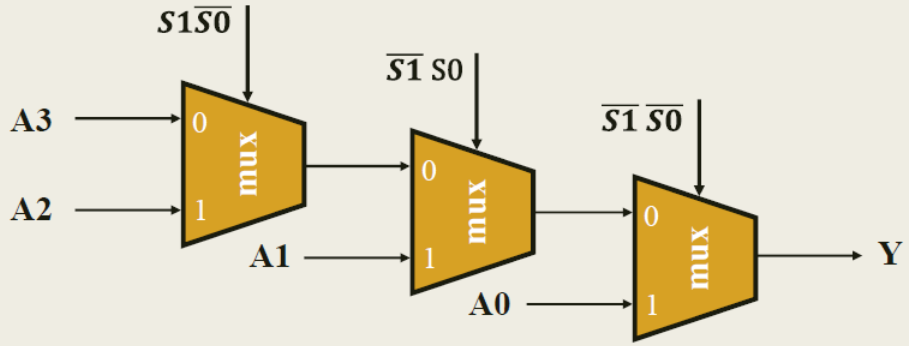
Entity (4to1)

Η οντότητα του πολυπλέκτη 4 σε 1 στη VHDL

```
entity MUX_4_to_1 is
    port (
      A0: in STD_LOGIC;
      A1: in STD_LOGIC;
      A2: in STD_LOGIC;
      A3: in STD_LOGIC;
      S0: in STD_LOGIC;
      S1: in STD_LOGIC;
      Y: out STD_LOGIC);
end MUX_4_to_1;
```



Λύση 1: Υλοποίηση με πολυπλέκτες 2 σε 1 σε δομή αλυσίδας



VHDL – Multiplexer

Architecture (4to1)

Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς – Λύση 1

```
architecture BEHAVIORAL of MUX_4_to_1 is
begin
  process (A0, A1, A2, A3, S0, S1)
  begin
    if    (S1 = '0' and S0 = '0') then Y <= A0;
    elsif (S1 = '0' and S0 = '1') then Y <= A1;
    elsif (S1 = '1' and S0 = '0') then Y <= A2;
    else                               Y <= A3;
    end if;
  end process;
end BEHAVIORAL;
```

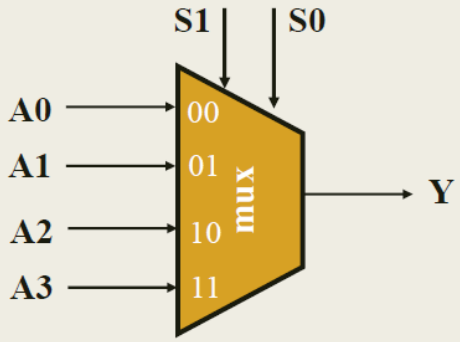
Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες
οι είσοδοι του συνδυαστικού κυκλώματος

VHDL – Multiplexer

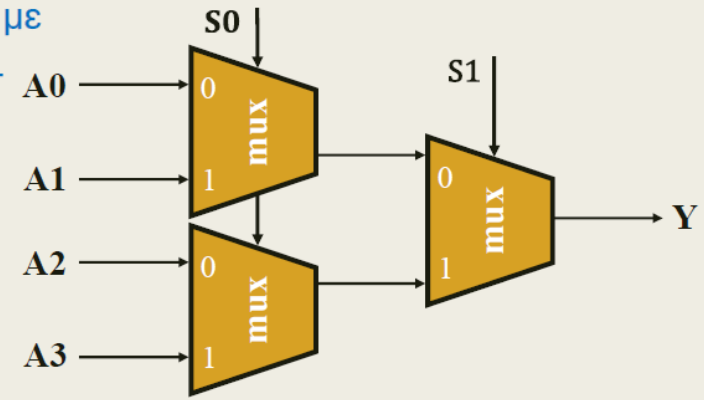
Entity (4to1)

Η οντότητα του πολυπλέκτη 4 σε 1 στη VHDL

```
entity MUX_4_to_1 is
    port (
      A0: in STD_LOGIC;
      A1: in STD_LOGIC;
      A2: in STD_LOGIC;
      A3: in STD_LOGIC;
      S0: in STD_LOGIC;
      S1: in STD_LOGIC;
      Y: out STD_LOGIC);
end MUX_4_to_1;
```



Λύση 2: Υλοποίηση με πολυπλέκτες 2 σε 1 σε δομή δένδρου



VHDL – Multiplexer

Architecture (4to1)

Η αρχιτεκτονική του πολυπλέκτη 4 σε 1 στη VHDL
Περιγραφή συμπεριφοράς – Λύση 2

```
architecture BEHAVIORAL of MUX_4_to_1 is
begin
  process (A0, A1, A2, A3, S0, S1)
  begin
    if (S1 = '0') then
      if (S0 = '0') then Y <= A0;
      else Y <= A1;
      end if;
    else
      if (S0 = '0') then Y <= A2;
      else Y <= A3;
      end if;
    end if;
  end process;
end BEHAVIORAL;
```

Στη λίστα ευαισθησίας συμπεριλαμβάνονται όλες οι είσοδοι του συνδυαστικού κυκλώματος

VHDL – Τελεστές (operators)

Λογικοί Τελεστές

Operator	Operation
not	Logical negation
and	Logical AND
nand	Logical NAND
or	Logical OR
nor	Logical NOR
xor	Logical Exclusive-OR
xnor	Logical Exclusive-NOR

Εφαρμόζονται σε/ανά bit

VHDL – Τελεστές (operators)

Αριθμητικοί Τελεστές

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
mod	Modulus
rem	Remainder
abs	Absolute value
**	Exponential

Εφαρμόζονται σε σήματα τύπου integer/signed/unsigned

VHDL – Τελεστές (operators)

Τελεστές σύγκρισης (Relational Operators)

Operator	Returns true if the comparison is:
=	Equal
/=	Not equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal

VHDL – Τελεστές (operators)

Τελεστές Ολίσθησης (Shift/Rotate Operators)

Operator	Operation
sll	Shift left logical
srl	Shift right logical
sla	Shift left arithmetic
sra	Shift right arithmetic
rol	Rotate left
ror	Rotate right

Εφαρμόζονται σε vector

VHDL – Τελεστές (operators)

Συναρτήσεις Ολίσθησης (Πακέτο numeric_std)

shift_left()
shift_right()
-

Αριθμητική Ολίσθηση

rotate_left()
rotate_right()

Εφαρμόζονται σε signed/unsigned

VHDL – Τελεστές (operators)

VHDL'93 Vivado 2019.2, **2022.2**

signed/unsigned

Για το std_logic_vector
ΔΕΝ υπάρχει κάποια συνάρτηση.

Εάν όμως ενεργοποιήσουμε τη VHDL 2008 τότε μπορούμε να χρησιμοποιήσουμε τα sll, srl, rol, ror.

```
x<=a sll 2;
y<=a srl 2;
-----------------
r<=shift_left(a,2);
t<=shift_right(a,2);
-----------------
q<=a rol 2;
u<=a ror 2;
i<=rotate_left(a,2);
o<=rotate_right(a,2)
```

Εάν όμως ενεργοποιήσουμε τη VHDL 2008 τότε μπορούμε να χρησιμοποιήσουμε και τα sla, sra.

VHDL – Τελεστές (operators)

VHDL'93 Vivado 2022.1

std_logic_vector

```
x<=a sll 2;
y<=a srl 2;
-----------------
q<=a rol 2;
u<=a ror 2;
```

signed/unsigned

```
x<=a sll 2;
y<=a srl 2;
-----------------
z<=a sla 2;
w<=a sra 2;
r<=shift_left(a,2);
t<=shift_right(a,2);
-----------------
q<=a rol 2;
u<=a ror 2;
i<=rotate_left(a,2);
o<=rotate_right(a,2)
```

Παρόμοια προβλήματα υπάρχουν και στα άλλα vector (string, bit_vector)

VHDL – Τελεστές (operators)

Προτεραιότητα

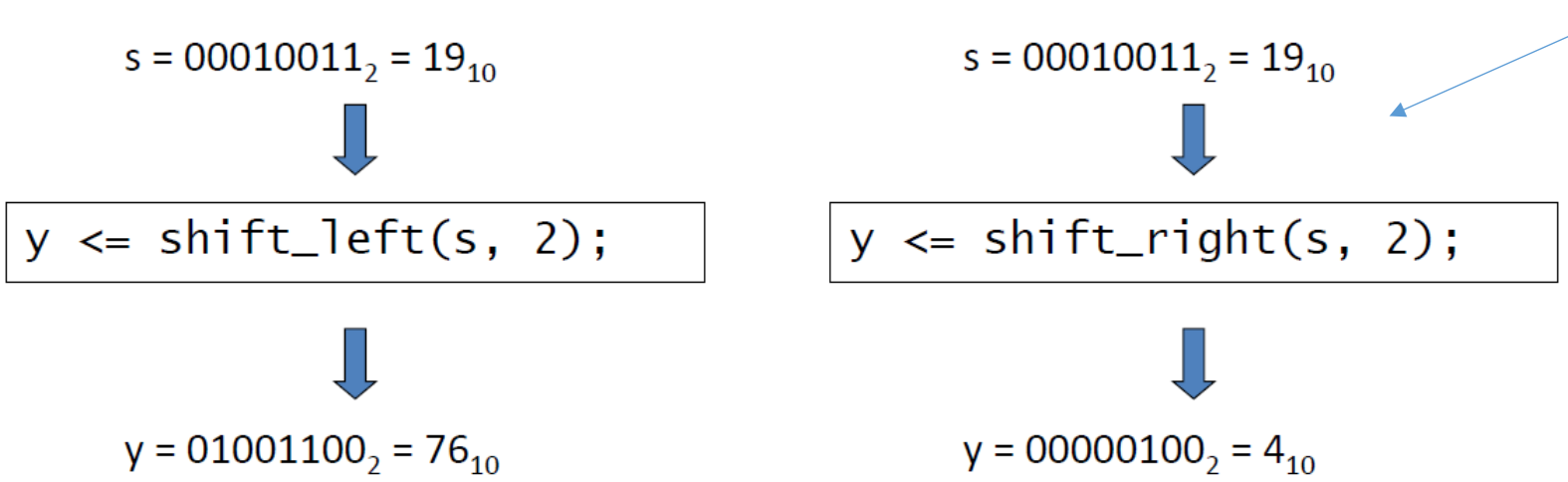
	Τελεστής	Σημασία
Υψηλότερη	`not`	NOT
	`*` `/` `mod` `rem`	MUL, DIV, MOD, REM
	`+` `-`	PLUS, MINUS
	`rol` `ror` `srl` `sll`	Περιστροφή, λογική ολίσθηση
	`<` `<=` `>` `>=`	Σχετική σύγκριση
	`=` `/=`	Σύγκριση ισότητας
Χαμηλότερη	`and` `or` `nand` `nor` `xor` `xnor`	Λογικές πράξεις (εκτελούνται από αριστερά προς τα δεξιά)

VHDL – Τελεστές (operators)

Unsigned: Ολίσθηση

- Λειτουργίες shift_left και shift_right
  - Αποτέλεσμα ίδιου μεγέθους με τον τελεστέο

Αποκοπή προς τα κάτω



VHDL – Τελεστές (operators)

Μπορείτε να διαβάσετε και:
https://www.nandland.com/vhdl/examples/example-shifts.html
https://redirect.cs.umbc.edu/portal/help/VHDL/operator.html
https://www.vhdl-online.de/vhdl_reference_93/shift_operators

Μπορείτε να δείτε επίσης και τη συμπεριφορά τους σε bit_vector:
https://hdlworks.com/hdl_corner/vhdl_ref/VHDLContents/BitVector.htm

Συνοπτικός οδηγός VHDL
https://www.ics.uci.edu/~jmoorkan/vhdlref/vhdl.html

Για διαφορές mod/rem:
https://stackoverflow.com/questions/25848879/difference-between-mod-and-rem-operators-in-vhdl

VHDL

Ψηφιακό κύκλωμα – VHDL: Architecture (Structural)

```
architecture arch_name of entity_name is
    signal signal_name: signal_type;
    component comp_name
        port (
            signal_name: mode signal_type;
            ...
            signal_name: mode signal_type);
    end component;
begin
  label_1: comp_name port map (signal_name, ..);
  ...
  label_n: comp_name port map (signal_name, ..);
  concurrent_component_statement;
  ...
  concurrent_component_statement;

end architecture arch_name;
```

Ψηφιακό κύκλωμα – VHDL: Architecture (Structural 1/2)

- **arch_name**: το όνομα της αρχιτεκτονικής
- **entity_name**: το όνομα της οντότητας
- **comp_name**: το όνομα του **στοιχείου (component)** που χρησιμοποιείται στην αρχιτεκτονική της οντότητας.
  - Το **στοιχείο είναι μία ήδη προκαθορισμένη οντότητα**. Ξέρουμε τη λειτουργικότητα που προσφέρει. Όταν επαναχρησιμοποιούμε μια οντότητα (entity) την ονομάζουμε **component**.
- **signal_name**: το όνομα του σήματος (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
  - στις δηλώσεις σημάτων (μετά το **is** και πριν το **begin**) το σήμα είναι μία **εσωτερική διασύνδεση** της αρχιτεκτονική της οντότητας
  - στις δηλώσεις των διαύλων του στοιχείου (component) το σήμα είναι είσοδος, έξοδος του στοιχείου, όπως ακριβώς προκύπτει από τη δήλωση των διαύλων της οντότητας του συγκεκριμένου στοιχείου
- **signal_type**: ο τύπος του σήματος (STD_LOGIC ή άλλος)
- **Ταυτόχρονες εντολές ανάθεσης σήματος** (Όπως σε αρχιτεκτονική Dataflow)

VHDL

Ψηφιακό κύκλωμα – Αναπαράσταση σε VHDL – Αρχιτεκτονική – Components (2/2)

- **Ταυτόχρονες εντολές στοιχείων** (concurrent_component_statements)

```
label: comp_name port map (signal_name, ..);
```

- **label**: οι μοναδικές ετικέτες των στοιχείων. Δημιουργείται ένα στιγμιότυπο (**instance**) του component με το όνομα **label**. Η χρήση ενός ή παραπάνω component στην αρχιτεκτονική του συστήματος, ονομάζεται **instantiation**.
- **comp_name**: το όνομα του στοιχείου (υποκύκλωμα) που χρησιμοποιείται στην αρχιτεκτονική της οντότητας
- **port_map**: περιγράφει τη σύνδεση των σημάτων της οντότητας με τα port του component. ς
- **signal_name**: το όνομα του σήματος που συνδέεται στο υποκύκλωμα (comp_name) (εάν είναι πολλά σήματα χωρίζονται με κόμμα)
  - το σήμα είναι μία διασύνδεση που αφορά τη συγκεκριμένη αρχιτεκτονική της οντότητας που χρησιμοποιεί το στοιχείο
  - αντιστοιχεί **αμφιμονοσήμαντα** στο αντίστοιχο σήμα της δήλωσης των port του υποκυκλώματος (comp_name) (**θέλει προσοχή η σειρά των σημάτων**)

VHDL

Υλοποίηση Αρχιτεκτονικής (Structural)

1. Για την υλοποίηση της αρχιτεκτονικής χρησιμοποιούμε τη λειτουργικότητα άλλων Οντοτήτων που έχουμε ήδη υλοποιήσει.
2. Τις οντότητες που θα χρησιμοποιήσουμε τις δηλώνουμε (όνομα και port) ανάμεσα στο is και το begin της αρχιτεκτονικής. Όμως πλέον έχουν την έννοια της συνιστώσας (component).
3. Η αρχιτεκτονική πλέον αποτελείται και από εντολές που καλούν(δημιουργούν) τα components, τα οποία μπορεί και να επικοινωνούν μεταξύ τους (συνηθέστερη περίπτωση).
4. Άρα μια οντότητα μπορεί να χρησιμοποιεί άλλες οντότητες (με τη μορφή component), οι οποίες μπορεί να είναι υλοποιημένες με οποιοδήποτε από τις 3 αρχιτεκτονικές. Στη περίπτωση της structural δομής σχηματίζεται ένα δέντρο, τα φύλλα του οποίου είναι άλλες οντότητες. Τα φύλλα που είναι τερματικά έχουν δομή Dataflow ή Behavioral.
5. ΔΕΝ αλλάζει τίποτα στην προσομοίωση

Μπορείτε να δείτε και το: https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

VHDL – Structural architecture

ALU 8bit από 2 ALU των 4bit

Πρόσθεση

4 bit

```
  1001
 +1101
 -------
  0110   Carry='1'
```

8 bit

1=Carry in στη 2η ALU

```
 0010 1001
+0101 1101
---------------
 1000 0110   Carry_out='1' από 1η ALU
```

Carry_in='0' 1η ALU

Carry_out='0' από τη 2η ALU

Διπλασιασμός

4 bit

```
 1001 & '0'
 --------
  0010   Carry='1'
```

8 bit

1=Carry in στη 2η ALU

```
0010 1001 & '0'
---------------
0101 0010     Carry_out='1' από 1η ALU
```

Carry_in='0' 1η ALU

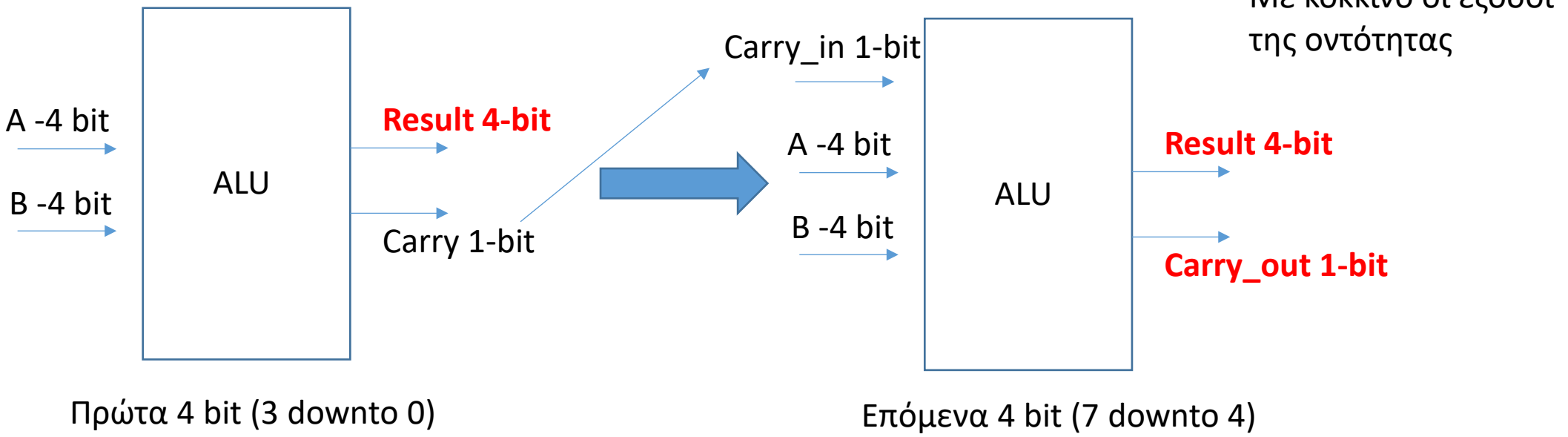
Carry_out='0' από τη 2η ALU

VHDL – Structural architecture

ALU 8bit από 2 ALU των 4bit

Για να μπορέσουμε να συνδυάσουμε 2 alu 4-bit θα πρέπει να μπορέσουμε να χειριστούμε το carry

Με κόκκινο οι έξοδοι της οντότητας

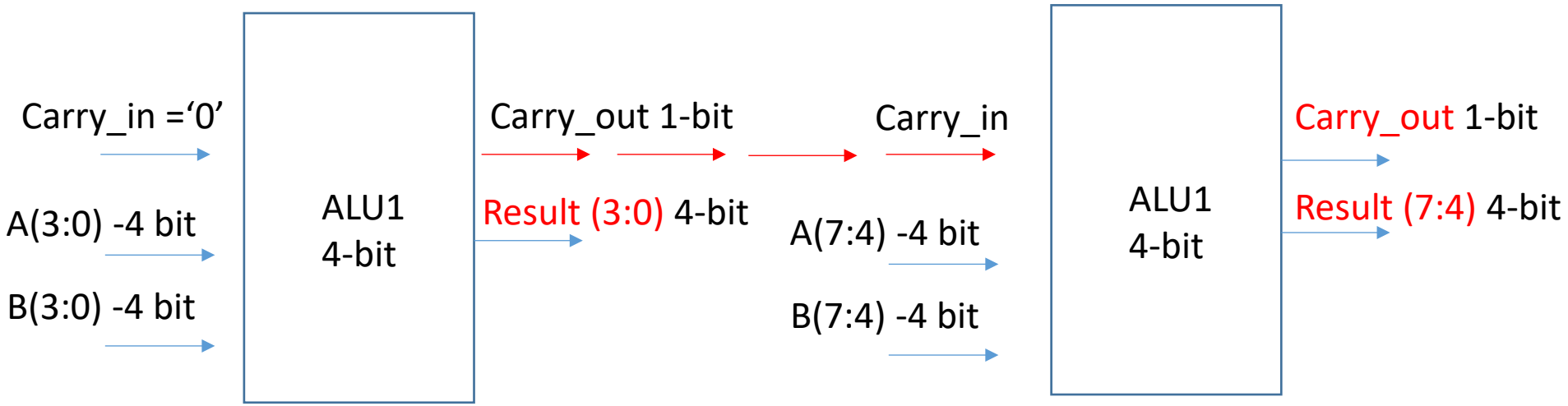


Πρώτα 4 bit (3 downto 0)

Επόμενα 4 bit (7 downto 4)

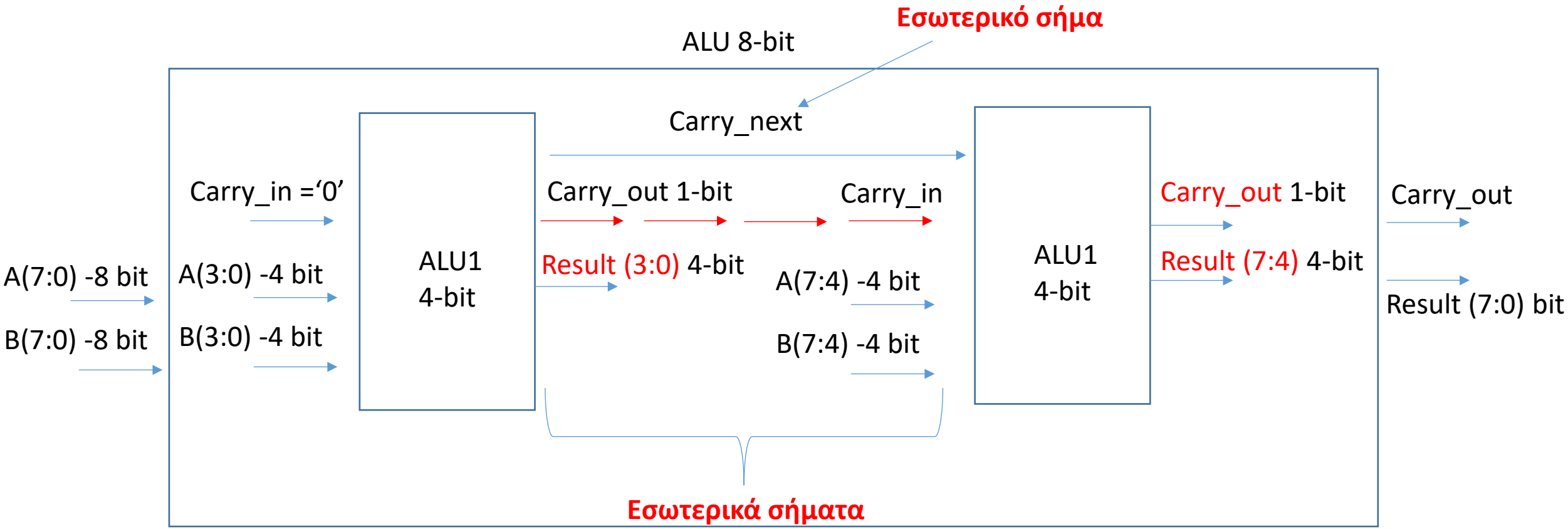
VHDL – Structural architecture

ALU 8bit από 2 ALU των 4bit



VHDL – Structural architecture

Αθροιστής 8bit από 2 αθροιστές των 4bit



VHDL – Structural architecture

Αθροιστής 8bit από 2 αθροιστές των 4bit

first_half: Adder_4bit port map(A_8bit(3 downto 0), B_8bit(3 downto 0), '0', Sum_8bit(3 downto 0), Carry_next);

second_half: Adder_4bit port map(A_8bit(7 downto 4), B_8bit(7 downto 4), Carry_next, Sum_8bit(7 downto 4), Cout_8bit);

Περίληψη

- Selected Assignment
- Conditional Statements
- Πολυπλέκτες
- Τελεστές
- Ολισθήσεις
- Structural Αρχιτεκτονική
- Διαβάζετε τις παραγράφους 2.1.3, 2.3.2 (θεωρία και VHDL) από Ashenden
- Διαβάζετε τις παραγράφους 4.2.1, 4.2.4, 4.2.6, 4.3, 4.5.1, 4.5.2, 2.8.1 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.
- Διαβάζετε τις παραγράφους 4.1.1, 4.6, 4.6.1, 4.6.2, 4.6.3, 4.6.7, 4.6.8, 12.3, 12.6, 12.7.3, 12.7.4, 12.9.1-12.9.3 από το βιβλίο των Brown-Vranesic.