Πανεπιστήμιο Πατρών

Τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών

Εργαστήριο Σχεδιασμού Ολοκληρωμένων Κυκλωμάτων

Σχεδιασμός Ολοκληρωμένων Κυκλωμάτων (VLSI) II

Εαρινό Εξάμηνο 2026

3η Εργαστηριακή Άσκηση

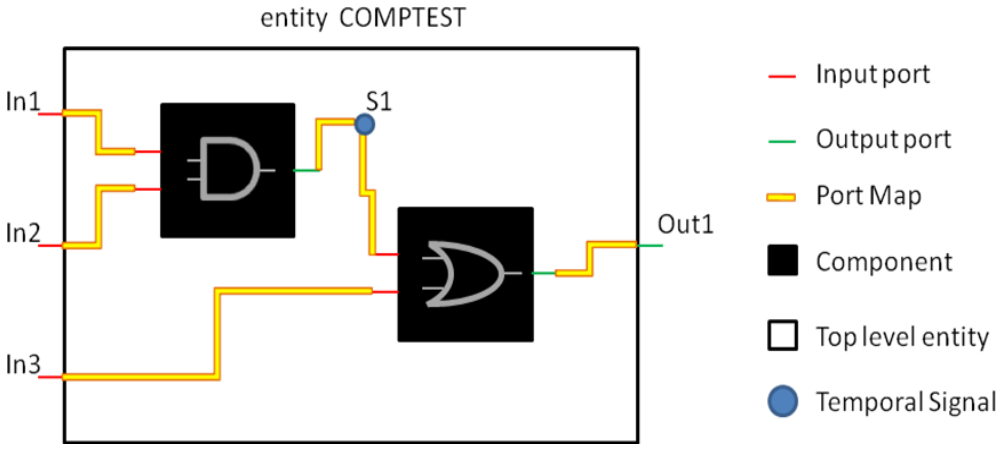
Structural VHDL

Κατά το δομικό (structural) τρόπο σχεδίασης, γίνεται περιγραφή των διασυνδέσεων των υπομονάδων του κυκλώματος. Δεν απαιτείται περιγραφή της λειτουργία της κάθε υπομονάδας συνεχώς, με την προϋπόθεση ότι: α) τα επιμέρους υποκυκλώματα (components) έχουν μεταγλωττιστεί και β) οι βιβλιοθήκες, στις οποίες έχουν είναι αποθηκευμένα τα αρχεία των μεταγλωττίσεων, είναι ορατές. Για παράδειγμα τα παραπάνω μπορούν να επιτευχθούν με δύο τρόπους.

Ο πρώτος τρόπος είναι να υπάρχουν σε ένα αρχείο όλοι οι σχεδιασμοί (entities, architectures) για όλα τα components. Αυτό όμως έχει ως αποτέλεσμα ο μοναδικός κώδικας να είναι μεγάλος και δύσχρηστος. Επίσης, μειώνει σημαντικά την ευελιξία σε περίπτωση αλλαγών. Τέλος, δεν υποστηρίζει την περίπτωση όπου διαφορετικές ομάδες αναλαμβάνουν την επιμέρους ανάπτυξη των υποκυκλωμάτων και μετά τα επιμέρους υποκυκλώματα επαναχρησιμοποιούνται για την υλοποίηση ενός πολύπλοκου σχεδιασμού.

Ένα δεύτερος τρόπος είναι να αναπτυχθούν ξεχωριστά τα υποκυκλωμάτα σε ξεχωριστούς σχεδιασμούς και στη συνέχεια να χρησιμοποιηθούν σε ένα ανώτερο ιεραρχικά επίπεδο του σχεδιασμού. Έτσι, θα μπορούσαν όλα τα χαμηλότερου επιπέδου υποκυκλώματα να μεταγλωττίζονται στη βιβλιοθήκη work του ανώτερου επίπεδου. Ο μόνος περιορισμός είναι ότι τα υποκυκλώματα μεταγλωττίζονται από το χαμηλότερο στο υψηλότερο ιεραρχικό επίπεδο. Έτσι, όταν ένα υψηλότερο επίπεδο της ιεραρχίας πρόκειται να χρησιμοποιήσει ένα υποκύκλωμα (component) ενός χαμηλότερου επιπέδου, τότε το δεύτερο θα έχει ήδη μεταγλωττιστεί, θα είναι ορατό και θα μπορεί να χρησιμοποιηθεί.

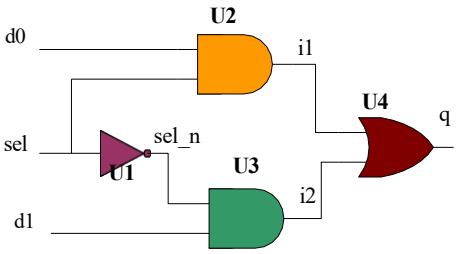
Στη γενική μορφή ο ιεραρχικός σχεδιασμός παρουσιάζεται με ακόλουθο απλό παράδειγμα. Τα χαμηλότερου επιπέδου υποκυκλώματα είναι οι πύλες AND, OR ενώ το υποκύλωμα COMPTEST είναι ανώτερο ιεραρχικά.



Στη συνέχεια ακολουθεί ένα παράδειγμα όπου περιγράφεται με δομικό τρόπο το κύκλωμα του παρακάτω σχήματος. Στόχος είναι να περιγράφει ένας πολυπλέκτης (component mux) υλοποιούμενος από πύλες. Σε αυτό το παράδειγμα, υποθέτουμε ότι οι κώδικες των υποκυκλωμάτων AND, OR, NOT έχουν αναπτυχθεί, επιβεβαιωθεί η ορθή λειτουργίας τους και μεταγλωττιστεί στη βιβλιοθήκη work του υποκυκλώματος mux.

Ο κώδικας VHDL με χρήση δομικής σχεδίασης αποτελείται από τα ακόλουθα τμήματα: βιβλιοθήκες, **ENTITY** (top level component), **ARCHITECTURE** (top level component). Εντός της ενότητας ARCHITECTURE περιλαμβάνονται οι ακόλουθες ενότητες: α) *δηλώσεις των υποκυκλωμάτων που θα χρησιμοποιηθούν (Components Declaration), β) προσδιορισμός της αρχιτεκτονικής που θα χρησιμοποιηθεί για το κάθε υποκύκλωμα* (*Components Specification*) *και γ) καθορισμός της συνδεσμολογίας μεταξύ των υποκυκλωμάτων (Components instantiation)*. Έτσι, ο κώδικας έχει ως εξής:

```
---- Entity (top level interface) of mux component-----
Entity mux is
port ( d0, d1, sel : in  std_logic;
                  q : out std_logic );
end;

---- Architecture of mux component-----

architecture str_mux of mux is

-- Components Declaration. Δήλωση των χαμηλότερου επιπέδου υποκυκλωμάτων από τα οποία αποτελείται η αρχιτεκτονική (AND, OR, NOT). Κατά τις δηλώσεις χρησιμοποιείται ακριβώς το ίδιο interface (ENTITY), που χρησιμοποιήθηκε στους κώδικες περιγραφής αυτών-----

component and_comp

port (  a,b : in   std_logic;
        c   : out  std_logic);
end component;

component inv_comp
port (  a   : in    std_logic;
        b   : out  std_logic);
end component;

component or_comp
port (  a,b : in   std_logic;
        c   : out  std_logic);
end component;

---- Δήλωση εσωτερικών σημάτων της ARCHITECTURE-----
signal i1, i2, sel_n: std_logic

-- Comp Specification. Προσδιορισμός της αρχιτεκτονικής που θα χρησιμοποιηθεί για κάθε υποκυκλώμα-----
for U1 : inv_comp use entity work.inv_comp(rtl);
for U2, U3: and_comp use entity
work.and_comp(rtl);
for  U4: or_comp use entity work.or_comp(rtl);
```



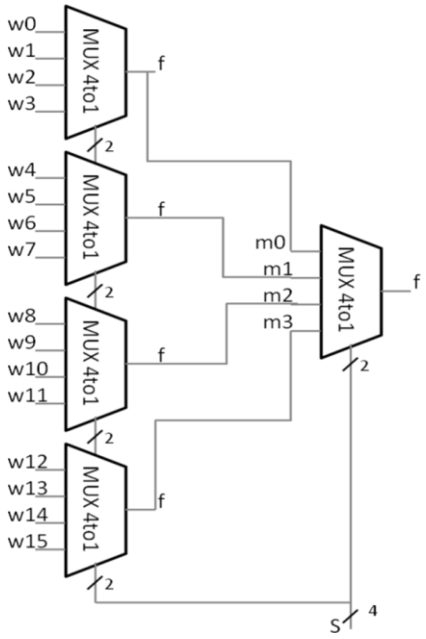
← Αν έχουν περιγραφεί τα components σε βιβλιοθήκες, τότε είναι απαραίτητες.

```
---- Body of ARCHITECTURE-----
begin
-- Component Instantiation. Καθορισμός της συνδεσμολογίας μεταξύ των υποκυκλωμάτων-----
U1: inv_comp port map (sel, sel_n);
U2: AND_comp port map (d0, sel, i1);
U3: AND_comp port map (sel_n, d1,i2);
U4: OR_comp port map (i1,i2,q);
end;
```

Αλλιώς, αν έχουν περιγραφεί πιο πάνω στον κώδικα ή σε άλλο αρχείο vhdl εντός του project τότε δεν χρειάζονται.

Σημείωση: **<u>Η port list πρέπει να είναι ίδια με αυτή της ενότητας ENTITY της υπομονάδας.</u> Για αποφυγή λαθών προτείνεται η αντιγραφή της οντότητας της υπομονάδας κατά τη δήλωση αυτής.**

Για τη δημιουργία εύκολη διαχείριση μεγάλων κυκλωμάτων και συστημάτων, μαζί με τις παραπάνω δομικές αρχές σχεδιασμού, γίνεται χρήση και των παρακάτω δύο εντολών (η πρώτη για παραμετροποίηση και η δεύτερη για εύκολη δημιουργία πολλαπλών στιγμιότυπων ενός component). Για παράδειγμα, το παρακάτω κύκλωμα πολυπλεκτών:



περιγράφεται δομικά, ως εξής:

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

    entity mux16to1 is
        port(w: in std_logic_vector(15 downto 0);
             s: in std_logic_vector(3 downto 0);
             f: out std_logic);
    end mux16to1;

    architecture my_arch of mux16to1 is

        component mux4to1
        port(w0, w1, w2, w3: in std_logic;
             s: in std_logic_vector(1 downto 0);
             f: out std_logic);
        end component;

        signal m: std_logic_vector(3 downto 0);

        begin
             mux_1: mux4to1 port map (w(0), w(1), w(2),
                                      w(3), s(1 downto 0), m(0));
             mux_2: mux4to1 port map (w(4), w(5), w(6),
                                      w(7), s(1 downto 0), m(1));
             mux_3: mux4to1 port map (w(8), w(9), w(10),
                                      w(11), s(1 downto 0), m(2));
             mux_4: mux4to1 port map (w(12), w(13), w(14),
                                      w(15), s(1 downto 0), m(3));
            mux5: mux4to1 port map (m(0), m(1), m(2), m(3), s(3 downto 2), f);
    end my_arch;
```

Το component mux4to1 μπορεί να περιγραφεί με κώδικα VHDL είτε χρησιμοποιώντας την εντολή WITH, είτε την εντολή WHEN. Η περιγραφή με εντολή WITH παρουσιάζεται παρακάτω:

```
1  -- Lab 2
2  -- MUX4to1
3  -- Inputs: w0, w1, w2, w3, s
4  -- Outputs: f
5
6  library ieee;
7      use ieee.std_logic_1164.all;
8
9      entity mux4to1 is
10         port(w0, w1, w2, w3: in std_logic;
11              s: in std_logic_vector(1 downto 0);
12              f: out std_logic);
13     end mux4to1;
14
15     architecture my_mux4to1 of mux4to1 is
16         begin
17             with s select
18                 f <=  w0 when "00",
19                       w1 when "01",
20                       w2 when "10",
21                       w3 when OTHERS;
22     end my_mux4to1;
23
```

Η παραπάνω περιγραφή μπορεί να γίνει σε ξεχωριστό αρχείο vhd, το οποίο θα βρίσκεται στον ίδιο φάκελο του project που βρίσκεται και το αρχείο του πολυπλέκτη 16 σε 1. Εναλλακτικά, μπορεί να προστεθεί στο ίδιο αρχείο vhd με τον πολυπλέκτη 16 σε 1.

- Η εντολή ***GENERIC***

Με την εντολή GENERIC μπορούμε να δηλώσουμε «σταθερές», ώστε να παραμετροποιείται εύκολα το κύκλωμα, δηλαδή, να αλλάζει τα μεγέθη εντός του εύκολα. Τα GENERICS μοιάζουν πολύ με τα ports. Ορίζονται εντός της entity και μπορούν να γίνουν map με την εντολή GENERIC MAP. Παράδειγμα σύνταξης:

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

entity latch_n is
    generic (n :integer :=8);
    port (clock, reset: in std_logic;
        input1: in std_logic_vector(n-1 downto 0);
        output1: out std_logic_vector(n-1 downto 0));
    end latch_n;

architecture my_latch_n of latch_n is
    signal temp: std_logic_vector(n-1 downto 0);
    begin
        process (clock, reset, input1)
            begin
                if reset='1' then
                    temp<=(others=>'0');
                else
                   if clock='1' and clock'event then
                       temp<=input1;
                   end if;
                end if;
        end process;
        output1<=temp;
end my_latch_n;
```

Default value, αν δεν πάρει άλλη τιμή από generic map υψηλότερου επιπέδου

Υπάρχουν περιπτώσεις χρήσης των Generics σε ιεραρχία με περισσότερα του ενός components. Για παράδειγμα, στον παρακάτω κώδικα, υπάρχουν δύο generics. Το *m* με default value 32 και το *n* στο component με default value 8. Με τη χρήση του generic map, εξασφαλίζουμε ότι το *n* θα παίρνει πάντα την τιμή του *m*. Δηλαδή, αν από υψηλότερο επίπεδο δεν προσδιοριστεί το *m* (πράγμα που ισχύει και στο παρακάτω παράδειγμα) θα πάρει και το *m* και το *n* την τιμή 32.

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

entity gen_map_test is
    generic (m :integer :=32);
    port (clock, reset: in std_logic;
        in1: in std_logic_vector(m-1 downto 0);
        out1: out std_logic_vector(m-1 downto 0));
    end gen_map_test;

architecture my_arch of gen_map_test is

    component latch_n
        generic (n :integer :=8);
        port(clock, reset: in std_logic;
        input1: in std_logic_vector(n-1 downto 0);
        output1: out std_logic_vector(n-1 downto 0));
    end component;

    begin
        L1: latch_n generic map(m) port map(clock, reset, in1, out1);

   end my_arch;
```

- Η εντολή ***FOR GENERATE***

Η εντολή αυτή παρέχει έναν εύκολο τρόπο επανάληψης μίας λογικής εξίσωσης ενός port map ή της δημιουργίας του στιγμιότυπου ενός component. Με την εντολή αυτή μπορούμε να δημιουργήσουμε πολλά port maps με εύκολο τρόπο. Παρόλα αυτά, χρειάζεται ιδιαίτερη προσοχή στην χρήσης της καθώς προϋποθέτει την ύπαρξη μεταβλητών για την εισαγωγή του επιθυμητού αριθμού components.

Για παράδειγμα, το κύκλωμα του πολυπλέκτη 16 σε 1 που παρουσιάστηκε παραπάνω, με χρήση FOR GENERATE μπορεί να περιγραφεί ως εξής:

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

    entity mux16to1 is
        port(w: in std_logic_vector(15 downto 0);
             s: in std_logic_vector(3 downto 0);
             f: out std_logic);
    end mux16to1;

    architecture my_arch of mux16to1 is

        component mux4to1
        port(w0, w1, w2, w3: in std_logic;
             s: in std_logic_vector(1 downto 0);
             f: out std_logic);
        end component;

        signal m: std_logic_vector(3 downto 0);

        begin
            generate_label:
            for i in 0 to 3 generate
                 mux_i: mux4to1 port map (w(4*i), w(4*i+1), w(4*i+2),
                              w(4*i+3), s(1 downto 0), m(i));
            end generate;

            mux5: mux4to1 port map (m(0), m(1), m(2), m(3), s(3 downto 2), f);
    end my_arch;
```

Ασκήσεις για το σπίτι

Άσκηση 1: Σχεδίαση και Υλοποίηση μίας ALU

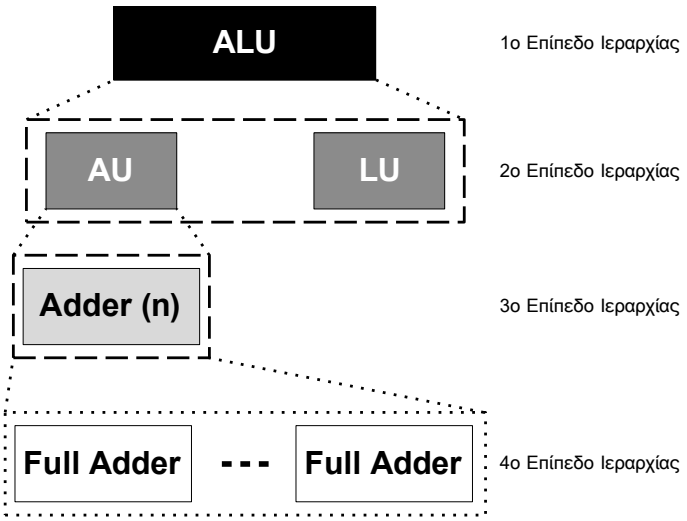
Στόχος της άσκησης είναι αρχικά να σχεδιαστεί μια πλήρως λειτουργική μονάδα αριθμητικών και λογικών πράξεων **σε δομική σχεδιαστική μορφή**. Η μονάδα θα πρέπει να έχει τις εξής εισόδους και εξόδους:

A: Είσοδος (8-bit) – Πρώτος τελεστέος σε συμπλήρωμα ως προς 2
B: Είσοδος (8-bit) – Δεύτερος τελεστέος σε συμπλήρωμα ως προς 2
Op: Είσοδος (3-bit) – Κωδικός πράξης
Out: Έξοδος (8-bit) – Αποτέλεσμα σε συμπλήρωμα ως προς 2.
Zero: Έξοδος (1-bit) – Ενεργοποιημένη αν το αποτέλεσμα είναι μηδέν
Cout: Έξοδος (1-bit) – Ενεργοποιημένη αν υπήρξε κρατούμενο (Carry)

Η «συμπεριφορά» της ALU είναι η εξής:

Κωδικός	**Πράξη**	**Αποτέλεσμα**
Op = 000	Πρόσθεση	Out = A + B
Op = 001	Αφαίρεση	Out = A - B
Op = 100	Λογικό «ΚΑΙ»	Out = A & B
Op = 101	Αντιστροφή του Α	Out = ! A
Op = 110	Λογικό «Η»	Out = A | B
Op = 111	Λογικό «XOR»	Out = A ⊕ B

Ιεραρχία της σχεδίασης:

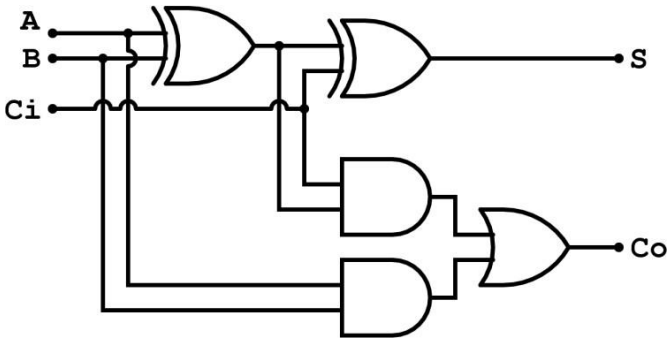


Στην σχεδίασή σας πρέπει να χρησιμοποιήσετε τουλάχιστον τέσσερα επίπεδα ιεραρχίας. Στο υψηλότερο επίπεδο βρίσκεται η μονάδα ALU, η οποία αποτελείται από δύο εξαρτήματα (components) ένα για αριθμητικές πράξεις (AU) και ένα για λογικές πράξεις (LU), και ό,τι λογική είναι απαραίτητη για την διασύνδεσή τους (2ο επίπεδο).

Το εξάρτημα Αριθμητικών Πράξεων (AU) πρέπει οπωσδήποτε να υλοποιηθεί σε δομική μορφή (π.χ. με χρήση της λειτουργικότητας GENERATE) και χρησιμοποιώντας ένα εξάρτημα πλήρους αθροιστή (σε δομική μορφή επίσης – 3ο επίπεδο). Το εξάρτημα Λογικών πράξεων θα πρέπει να υλοποιηθεί επίσης με δομική μορφή (κάθε 8-bit πύλη ξεχωριστά).

Παρατηρήσεις:

Δώστε προσοχή στις εξόδους Zero και Cout. Ποιες είναι οι συνθήκες που ορίζουν αυτές τις εξόδους? Πως πετυχαίνω την αφαίρεση χρησιμοποιώντας έναν αθροιστή όταν η κωδικοποίηση των τελεσταίων είναι σε συμπλήρωμα ως προς δύο; Ένας πλήρης αθροιστής υλοποιείται δομικά με χρήση πυλών XOR, AND και OR:



Να υλοποιηθεί σύνθεση στα 7 nm και 45 nm, με την προσθήκη καταχωρητών, όπως και στην προηγούμενη εργαστηριακή άσκηση.

Επιπλέον, στους φακέλους σας υπάρχει ο φάκελος *XCELIUM*, ο οποίος περιέχει το Genus script που πρέπει να τρέξετε για να παραχθεί το αρχείο *.v*, το οποίο θα χρησιμοποιηθεί για την προσομοίωση (simulation) με το XCELIUM. Απαιτείται η χρήση των εργαλείων XCELIUM και LEC για την ALU.

Άσκηση 2: a) Σχεδιάστε έναν Carry-Select Adder των 16-bit με ομάδες block των 4-bit.
b) Χρησιμοποιώντας την εντολή GENERATE, σχεδιάστε τον ίδιο αθροιστή με block των 2, 4 και 8-bit. Συμβουλευτείτε την εικόνα 11.24 του CMOS VLSI Design: A Circuits and Systems Perspective των E.Weste και D.Harris

Οι ασκήσεις να υλοποιηθούν να υλοποιηθούν ΑΥΣΤΗΡΑ με ΔΟΜΙΚΗ μορφή κώδικα (structural VHDL) χρησιμοποιώντας συντρέχουσες εντολές