ΑΝΑΠΤΥΞΗ ΕΡΓΑΛΕΙΩΝ CAD ΓΙΑ ΣΧΕΔΙΑΣΗ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ – ΗΡΥ 608 / 419

Εαρινό Εξάμηνο 2023
Προθεσμία: Παρασκευή 24/3/21 έως τα μεσάνυχτα, υποβολή στο eClass

ΑΣΚΗΣΗ 2η (Έκδοση 1.0 της εκφώνησης, μπορεί να υπάρχουν διορθώσεις)

SYNTHESIS: ΑΠΟ ΠΕΡΙΓΡΑΦΗ (BEHAVIORAL, STRUCTURAL) ΣΕ NETLIST

1.0 ΕΙΣΑΓΩΓΗ

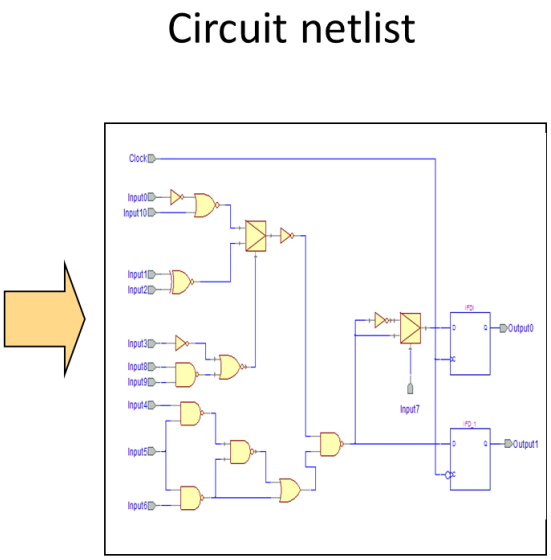
Έχουμε διδαχθεί στο μάθημα για την έννοια της σύνθεσης, όπου κάποια περιγραφή είτε δομική (structural) είτε συμπεριφερική (behavioral) μετατρέπεται σε κύκλωμα, δηλαδή σε ένα netlist διασυνδεδεμένων εξαρτημάτων (π.χ. λογικές πύλες, standard cells για VLSI, τρανζίστορ, κλπ.). Στις επόμενες ασκήσεις θα εμβαθύνουμε στο σημαντικό αυτό μέρος της σχεδιαστικής ροής. Οι ασκήσεις θα επικαλύπτονται ώστε να προλάβουμε τον χρόνο για την εβδομάδα που δεν κάναμε μάθημα, και θα δοθούν σε μία αλληλουχία από βήματα ούτως ώστε να μην είναι υπερβολικός ο φόρτος ανά άσκηση.

Σημειώνουμε ότι κανονικά χρειάζεται να γραφτεί κανονικός compiler, με λεξικογραφική ανάλυση της εισόδου (lex/yacc ή bison) και παραγωγή (με την τεχνική έννοια του όρου) της εξόδου. Αυτό το στοιχείο θα το παρακάμψουμε γιατί μπορεί να μην έχετε εμπειρία με τα σχετικά εργαλεία. Για όποιον/όποιαν ενδιαφέρεται και για αυτήν την διάσταση, μπορεί να δοθεί αργότερα έξτρα άσκηση με μορφή bonus. Για τον σκοπό της άσκησης θα κρατήσουμε τα πράγματα απλά: θα ασχοληθούμε για τώρα μόνο με αθροιστές με διάδοση κρατουμενου (ripple carry adders) που είναι φτιαγμένοι από λογικές πύλες (θα δώσουμε «βιβλιοθήκη» επιλογών) , και θα δούμε τόσο structural προσέγγιση όσο και behavioral προσέγγιση. Θα δούμε επίσης πως το PORT της VHDL επηρρεάζει την διασύνδεση των σημάτων.

2.0 SYNTHESIS – ΣΥΝΘΕΣΗ

Στο παρακάτω διάγραμμα έχουμε μία βασική ιδέα της λογικής σύνθεσης:

```
architecture MLU_DATAFLOW of MLU is

signal A1:STD_LOGIC;
signal B1:STD_LOGIC;
signal Y1:STD_LOGIC;
signal MUX_0, MUX_1, MUX_2, MUX_3: STD_LOGIC;

begin
        A1<=A when (NEG_A='0') else
                not A;
        B1<=B when (NEG_B='0') else
                not B;
        Y<=Y1 when (NEG_Y='0') else
                not Y1;

        MUX_0<=A1 and B1;
        MUX_1<=A1 or B1;
        MUX_2<=A1 xor B1;
        MUX_3<=A1 xnor B1;

        with (L1 & L0) select
                Y1<=MUX_0 when "00",
                        MUX_1 when "01",
                        MUX_2 when "10",
                        MUX_3 when others;

end MLU_DATAFLOW;
```



Για την άσκησή μας θα δημιουργήσουμε σε μία σειρά ασκήσεων λογικά κυκλώματα για ένα πλήρη αθροιστή μεταβλητού αριθμού Bits και με εξωτερικά σήματα που ορίζει ο χρήστης. Για τώρα δεν θα βάλουμε τον χρήστη στην εικόνα, και η βιβλιοθήκη μας θα είναι με ένα και μοναδικό στοιχείο (έναν full adder), και μάλιστα θα είναι σε αυτήν την άσκηση ενσωματωμένη στο παρόν πρόγραμμα. Θα δώσουμε όμως τον τρόπο σε netlist για την περιγραφή λογικών κυκλωμάτων και υποσυστημάτων, αφού κάτι τέτοιο θα χρειαστεί σε μελλοντικές ασκήσεις.

2.0 NETLIST ΓΙΑ ΤΗΝ ΑΣΚΗΣΗ

Θα δώσουμε ένα απλό τρόπο περιγραφής netlist (δικής μας επινόησης, αλλά και σε πιο ρεαλιστικές περιπτώσεις δεν είναι πολύ διαφορετικά τα nettlist). Όλα τα αρχεία είναι ASCII, και στην αρχή της γραμμής δύο αστεράκια υποδηλώνουν κάποιο keyword ή πεδίο, ενώ δύο "%" υποδηλώνουν σχόλιο και αγνοείται το υπόλοιπο της γραμμής. Μπορεί σε μελλοντικές ασκήσεις να προσθέσουμε στοιχεία στο netlist. Επίσης, στα netlist του μαθήματος τα κενά (blank, space) αγνοούνται, ενώ η νέα γραμμή σημαίνει νέα καταχώρηση (π.χ. τα σχόλια είναι μόνο μέχρι το τέλος της τρέχουσας γραμμής).

Για τώρα δεν δίνουμε την λειτουργικότητα – σπάμε δηλαδή το πρόβλημα της λειτουργικότητας από εκείνο της διασύνδεσης εξαρτημάτων. Δημιουργούμε (αλλά αυτό θα γίνει σε επόμενες ασκήσεις) δύο είδη netlist – εκείνο από το οποίο «τραβάμε» πύλες και υποσυστήματα, και εκείνο το οποίο δημιουργούμε για την έξοδο του προγράμματός μας. Ενδιάμεσα υπάρχει η είσοδος του χρήστη, που δεν είναι netlist αλλά μία συμπεριφερική ή δομική περιγραφή του τι απαιτείται. Το format του netlist εισόδου είναι το παρακάτω, και για τώρα ας θεωρηθεί ότι μόνο οι παρακάτω πύλες είναι διαθέσιμες (στις επόμενες ασκήσεις, για την παρούσα άσκηση έχουμε μόνο full adders), ενώ μέρος της άσκησης είναι να χρησιμοποιήσετε ένα **`COMP full_adder`** που σε μελλοντικές ασκήσεις θα είναι στο **`subsystem library`** αλλά για τώρα είναι μέσα στο πρόγραμμα.

Στην πράξη δεν γίνονται έτσι τα πράγματα, αλλά υπάρχει λόγος που έχουμε αυτή την δομή. Τα παρακάτω αρχεία για τις βιβλιοθήκες πυλών *δεν χρειάζονται για την παρούσα άσκηση*, αλλά θα χρειαστούν για τις επόμενες. Επειδή οι πύλες έχουν ακριβώς μία έξοδο η καθεμία, δεν χρειάζεται να ορίσουμε την έξοδο – ο κόμβος που καθορίζει την χρήση της πύλης **`(UXXX)`** είναι η έξοδος της πύλης. Αυτό φαίνεται καθαρά στο παράδειγμα χρήσης των πυλών σε βιβλιοθήκες υποσυστημάτων. Για την βιβλιοθήκη **`MUX`** που περιλάβαμε σαν παράδειγμα προσέχουμε ότι οι κόμβοι που δείχνουν κάθε πύλη αντιπροσωπεύουν την έξοδο αυτής της πύλης και μπορούν να χρησιμοποιηθούν σαν είσοδοι όταν την χρησιμοποιούμε. Για υποσυστήματα με πολλές εξόδους θα επεκτείνουμε την περιγραφή ως εξής: Στην αρχική περιγραφή **`COMP`**, αν έχουμε παραπάνω από μία έξοδο, τότε βάζουμε το επίθεμα "**`_<signal_name>"`**. Αν π.χ. για ένα full_adder έχουμε έξοδο C και έξοδο S, τότε στην μεν περιγραφή του θα έχουμε κάτι σαν:

```
COMP FULL_ADDER ; IN: A, B, CIN ; OUT: S, COUT
BEGIN …
…
S =
COUT =
END FULL_ADDER NETLIST
```

και στην χρήση του **`full_adder`** τα σχετικά σήματα θα είναι π.χ. **`U33_S, U33_COUT`** αν το **`U33`** είναι κάποιος **`full_adder`**.

Για να έχουμε επίσης ένα εύκολο τρόπο να ορίσουμε σήματα σαν έξοδο πυλών, χρησιμοποιούμε το "=", στον πολυπλέκτη έχουμε την έξοδο **`D`** να είναι η έξοδος της πύλης **`U4`**.

```
%%   THIS IS THE COMPONENT LIBRARY, COMPRISING OF GATES
**      COMPONENT LIBRARY
COMP NOT      ;     IN:P
COMP NAND2    ;     IN:P,Q
COMP NOR2     ;     IN:P,Q
COMP XOR2     ;     IN:P,Q

%%   THIS IS THE SUBSYSTEM LIBRARY, COMPRISING OF SUBSYSTEMS
%%   MADE OF GATES FROM THE COMPONENT LIBRARY
**   SUBSYSTEM LIBRARY
COMP MUX ; IN:A,B,C   ; OUT:D
BEGIN MUX NETLIST
U1 NOT C
U2 NAND2 A, U1
U3 NAND2 B, C
U4 NAND2 U2, U3
D = U4
END MUX NETLIST
```

Παρατηρούμε ότι όλες μας οι πύλες στην μορφή που χρησιμοποιούνται στο τελικό κύκλωμα παίρνουν ένα μοναδικό αριθμό **UXXX** που πλέον τις καθορίζει όχι απλά σαν πύλες κάποιου τύπου αλλά σαν τις συγκεκριμένες πύλες στο συγκεκριμένο σημείο. Θα κρατήσουμε την ίδια δομή και για υποσυστήματα, οπότε μπορεί να έχουμε κάτι όπως **U45 FULL_ADDER** …, αλλά όπως

Η μορφή του netlist για την έξοδο της άσκησης 2 και για τις επόμενες θα είναι η εξής:

U<number> <component type> <list of inputs>

3.0 ΤΙ ΠΡΕΠΕΙ ΝΑ ΚΑΝΕΙ Ο ΚΩΔΙΚΑΣ ΤΗΣ ΠΑΡΟΥΣΑΣ ΑΣΚΗΣΗΣ

Ο κώδικάς σας πρέπει να ζητάει απο τον χρήστη έναν ακέραιο αριθμό, με κάποιο ενδεικτικό prompt “Please enter number of bits” και το οποίο θα έχει τιμές 1 – 8 για τώρα (ώστε να ελέγχουμε τα κυκλώματα εύκολα). Καλό είναι ο κώδικάς σας να ελέγχει ότι ο χρήστης έδωσε τιμές στο πεδίο αυτό. Κατόπιν, το πρόγραμμά σας θα δημιουργεί ένα αρχείο εξόδου (π.χ. full_adder_netlist) με το netlist για τον αθροιστή n bits. Για την άσκηση 2 το δομικό μας στοιχείο είναι ο full_adder (μόνο – δεν τον υλοποιούμε ακόμη με πύλες). Επειδή χρειάζονται εξωτερικά σήματα, αυτά θα είναι σαν είσοδοι τα An-1 – A0, Bn-1 – B0, Cin (A0, B0 είναι τα LSB), και τα υπόλοιπα σήματα θα είναι εσωτερικά. Αντίστοιχα, το netlist σας πρέπει να δημιουργεί τις εξόδους Sn-1 – S0, και Cout, χρησιμοποιώντας την δομή “=” του netlist όπως περιγράψαμε παραπάνω (π.χ. S7 = U45_S). Η αρίθμηση **UXXX** δεν είναι απαραίτητο να είναι διαδοχική και πλήρης αλλά είναι απαραίτητο να μην έχουν τον ίδιο αριθμό **U** δύο διαφορετικοί κόμβοι. Κάλλιστα όμως μπορούμε να έχουμε **U1,U3,U4,U11** σε ένα κύκλωμα και να λείπουν τα υπόλοιπα (θα δούμε γιατί είναι αυτό χρήσιμο).

4.0 Η ΜΕΓΑΛΗ ΕΙΚΟΝΑ

Στις επόμενες ασκήσεις θα κάνουμε τα εξής: (1) Ο full adder σαν υποσύστημα θα αντλείται από μία βιβλιοθήκη, η δε δομή του ιδίου του full adder θα αντλείται επίσης από βιβλιοθήκη πυλών. (2) Σε ξεχωριστή άσκηση θα έχουμε την αντιστοίχιση των σημάτων στο netlist (ουσιαστικά το PORTMAP ή με όρους FPGA από την προχωρημένη λογική σχεδίαση το UCF). Κατόπιν θα δούμε κατά πόσον έχει νόημα να κάνουμε σε ξεχωριστή άσκηση την μετατροπή από behavioral σε structural.

4.0 ΑΝΑΦΟΡΑ

Η αναφορά πρέπει να είναι πλήρης. Οι κώδικες πρέπει να έχουν σχόλια (καλά, πλήρη και στα Αγγλικά, με block comments στην αρχή και στις βασικές ενότητες), και το κείμενο πρέπει επίσης να είναι επαρκές για το τι κάνει η άσκηση και πως.

5.0 ΠΡΟΣΟΧΗ

Υπάρχουν τρόποι να κάνει κανείς την άσκηση εύκολα και γρήγορα, χωρίς όμως να χτίζει για να χρησιμοποιήσει κώδικα σε μελλοντικές ασκήσεις (και να δυσκολευτεί κατόπιν). Προτείνουμε να δημιουργήσετε εσωτερικές δομές δεδομένων για τις βιβλιοθήκες και υποσυστήματα, και αρχικά αυτές οι δομές έχουν δεομένα που βάζετε απ' ευθείας. Όταν όμως διαβάζετε από βιβλιοθήκη τα δεδομένα αυτά, θα έχετε ευχέρεια να συμπληρώνετε τις δομές σας (επεκτείνοντάς τες όπου χρειάζεται). Αντίστοιχα, μπορεί να υπάρχει κάποια δομή που περιλαμβάνει την λίστα με τις πύλες και υποσυστήματα που γνωρίζει το σύστημά σας, μαζί με πληροφορίες για το που είναι το netlist ούτως ώστε να χτίσετε πάνω σε αυτό (σας θύμισε κάτι αυτό το τελευταίο; Θυμόσαστε από τους Ψηφιακούς Υπολογιστές τι κάναμε με τα labels στον Assembler;).

ΚΑΛΗ ΕΠΙΤΥΧΙΑ!!!