ΑΝΑΠΤΥΞΗ ΕΡΓΑΛΕΙΩΝ CAD ΓΙΑ ΣΧΕΔΙΑΣΗ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ – ΗΡΥ 608 / 419
Εαρινό Εξάμηνο 2023

Προθεσμία: Παρασκευή 28/4/21 έως τα μεσάνυχτα, υποβολή στο eClass

ΑΣΚΗΣΗ 4η (Έκδοση 1.0 της εκφώνησης, μπορεί να υπάρχουν διορθώσεις)

SYNTHESIS: ΑΠΟ ΠΕΡΙΓΡΑΦΗ (BEHAVIORAL, STRUCTURAL) ΣΕ NETLIST (ΟΛΟΚΛΗΡΩΣΗ ΚΥΚΛΟΥ ΜΕ I/O)

1.0 ΕΙΣΑΓΩΓΗ

Όπως έχει εξηγηθεί στο μάθημα, η άσκηση αυτή είναι συνέχεια της 3ης Άσκησης, και αφορά τον τρόπο καθορισμού εισόδου και αριθμών Bits στον αθροιστή, ενώ βγάζει όπως και η άσκηση 3 ένα (επίπεδο) netlist πυλών για τον αθροιστή που μελετάμε, χρησιμοποιώντας αυτούσιο τον κώδικά σας από την άσκηση 3. Επομένως, η έξοδος θα είναι ίδια με εκείνη της άσκησης 3, αλλά με διαφοροποίηση σημάτων που πλέον ορίζονται από τον χρήστη, όπως και ο αριθμός των Bits του αθροιστή μας, και αυτό θα γίνει αλλάζοντας τον κώδικα της άσκησης 2.

2.0 SYNTHESIS – ΣΥΝΘΕΣΗ ΟΠΩΣ ΓΙΝΕΤΑΙ ΠΡΑΓΜΑΤΙΚΑ ΣΤΗΝ VHDL

Ας δούμε τον παρακάτω κώδικα VHDL:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY adder IS
       GENERIC ( N : INTEGER := 16 ) ; --N>1
       PORT (       A      : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                    B      : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                    Cin    : IN STD_LOGIC ;
                    Result  OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0);
                    Cout          : OUT         STD_LOGIC
                    ) ;
END adder

ARCHITECTURE structural OF adder IS
SIGNAL temp_Res: STD_LOGIC_VECTOR (0 to N-1);
SIGNAL temp_Carry: STD_LOGIC_VECTOR (0 to N-2);

COMPONENT full_adder is
       port (x, y, z :  in std_logic;
             s, c    :  out std_logic);
end full_adder;

BEGIN
U0: full_adder PORT MAP (  X => A(0),
             Y => B(0),
             Z  => Cin,
             S => temp_res(0),
             C => temp_Carry(0)
             );
For i in 1 to n-2 generate
U1: full_adder PORT MAP (  X => A(i),
             Y => B(i),
             Z  => temp_Carry(i-1),
             S => temp_res(i),
             C => temp_Carry(i)
             );
end generate;
U1: full_adder PORT MAP (  X => A(n-1),
             Y => B(n-1),
             Z  => temp_Carry(n-2),
             S => temp_res(n-1),
             C => Cout
             );
END structural;
```

Παρατηρούμε τις εξής ενότητες, κάποιες των οποίων έπουμε ήδη δει (έστω, σε περιορισμένη μορφή) στις ασκήσεις μας:

- LIBRARY (κλπ) – μας λέει τι αναγνωρίζει σαν δομικά στοιχεία (βλ. π.χ. την δική μας βιβλιοθήκη πυλών και υποσυστημάτων, αν και είναι πολύ πιο εκτεταμένη η βιβλιοθήκη VHDL)
- ENTITY adder – μας λέει ότι φτιάχνουμε ένα «κουτί» που το λένε ADDER, N Bits, με διανύσματα εισόδου A,B και σήμα εισόδου Cin και διάνυσμα εξόδου το Result, καθώς και σήμα εξόδου το Cout. Είναι σημαντικό ότι σε συνάρτηση με το Architecture, παρακάτω, καθορίζονται επακριβώς το MSB και το LSB.
- ARCHITECTURE structural OF adder IS – έδώ έχουμε ένα φορμαλισμό για τον αθροιστή του ενός Bit (COMPONENT full_adder IS) και για την διασύνδεση τέτοιων COMPONENTS ώστε να έχουμε τον N-Bit adder (από το BEGIN έως το END). Εδώ παρατηρούμε ότι για τον full_adder ΔΕΝ μας δίνει κάποιο netlist, και ο λόγος είναι ότι το βρίσκει στην βιβλιοθήκη που διάβασε προηγουμένως – ξέρει επομένως τι είναι ένας full_adder του ενός Bit. Η δομή όμως (structural) από το BEGIN έως το END είναι σαν γλώσσα προγραμματισμού, γίνεται parsing κανονικά, και η έξοδος, αντί για κώδικας εκτέλεσης π.χ. του βρόχου, είναι ένα netlist του N-Bit adder.

Τι κάνουμε εμείς, και τι αφήνουμε: Ο κώδικας που γράψατε στην Άσκηση 2 είναι ουσιαστικά το back end του μέρους BEGIN … END. Δηλαδή, με βασικές γνώσεις από formal languages και compilers, θα διαβάζατε το συντακτικό του αντίστοιχου κώδικα, και θα δημιουργούσατε «κώδικα» εξόδου το netlist που ήδη κάνατε. Αυτό το βήμα το παρακάμπτουμε γιατί εκτός από το ότι είναι χρονοβόρο, με lex/yacc (ή bison) κλπ. απαιτεί και γνώσεις που ενδεχόμενα δεν έχετε (θεωρώ πιθανό να μην κάνετε back end από compiler σε υποχρεωτικό μάθημα της Σχολής). Θεωρούμε πάντως ότι ο δικός σας κώδικας της Άσκησης 2 ξέρουμε πλέον από που προέρχεται και είναι θέμα εμβάθυνσης σε τεχνολογία compilers για να υλοποιηθεί.

Μένει το ENTITY adder που μας δίνει τον ορισμό ολόκληρου του υποσυστήματος, μαζί με τα σήματα εισόδου/εξόδου. Αυτό θα το δημιουργήσουμε στην Άσκηση 4 αλλάζοντας τον κώδικα της Άσκησης 2. Στην Άσκηση 2 ζητούσαμε ένα αριθμό n και βγάζαμε το netlist σε διασυνδεδεμένους full adders, αλλά με προκαθορισμένα σήματα εισόδου, κλπ. Αυτό το n θα αντικατασταθεί πλέον από ένα αρχείο ASCII που θα έχει την εξής δομή:

```
ENTITY adder IS
VAR N = {1 UP to 8}  %% Number of Bits in the Adder
PORT ( IN A     : {A_MSB,…,A_LSB}        %% A
       IN B     : {B_MSB,…,B_LSB}        %% B
       IN Cin   : {Cin}                  %% Cin
       OUT S    : {S_MSB;…;S_LSB}        %% Sum
       OUT Cout : {Cout}                 %% Cout
END adder
```

Όλες οι μεταβλητές είναι έως δέκα χαρακτήρες, όχι απαραίτητα ίδιο μέγεθος σε χαρακτήρες η καθεμία, και το N μπορεί να πάρει τιμές από 1 έως 8. Το παραπάνω format διατηρεί κάποια από τα χαρακτηριστικά των προηγούμενων ασκήσεων (π.χ. σχόλια με %%), και μας επιτρέπει να ορίσουμε τα σήματα εισόδου όπως θέλουμε, διατηρώντας και τον αριθμό Bits μεταβλητό. Για τα σήματα εξόδου θα ισχύουν τα αντίστοιχα. Επίσης, το format αυτό παρακάμπτει την απαίτηση να χρειάζεται να διαβάζουμε και επεξεργαζόμαστε διανύσματα, αντικαθιστώντας την από την απαίτηση να ορίσουμε όλα τα σήματα ένα-ένα. Ότι βλέπετε σε αγκύλες είναι ορίσματα που πρέπει να οριστούν ένα-ένα στο αρχείο εισόδου, που είναι TXT.

Ένα (πραγματικό) ενδεικτικό αρχείο εισόδου είναι:

```
ENTITY adder IS
VAR N = 3  %% Number of Bits in the Adder
PORT ( IN A     : SO , YOU , THINK       %% A, MSB TO LSB
       IN B     : CAN , TELL , HEAVEN    %% B, MSB TO LSB
       IN Cin   : {FROM}                 %% Cin
       OUT S    : {HELL , BLUE , SKIES } %% Sum, MSB TO LSB
       OUT Cout : {PAIN}                 %% Cout
END adder
```

Επίτηδες επιλέχθηκαν λέξεις από στίχους εξαιρετικού τραγουδιού (δεν γράφω ποιο γιατί μάλλον το ξέρετε όλοι/όλες), για να δείξω ότι η αντιστοίχιση ναι μεν συνήθως θα είναι τα (βαρετά αλλά χρήσιμα) A2, A1, A0, κλπ. όμως αυτό δεν είναι απαραίτητο.

Για το παραπάνω αρχείο εισόδου, το αντίστοιχο αρχείο εξόδου είναι:

```
U1 FULL_ADDER THINK, HEAVEN, FROM
U2 FULL_ADDER YOU, TELL, U1_COUT
U3 FULL_ADDER SO, CAN, U2_COUT
SKIES = U1_S
BLUE = U2_S
HELL = U3_S
PAIN  = U3_COUT
```

Η έξοδος της άσκησης αυτής είναι απόλυτα ισοδύναμη με την έξοδο του εργαστηρίου 2. Μάλιστα, μπορείτε να δοκιμάσετε ότι η Άσκηση 3 είναι OK με τα νέα ονόματα μεταβλητών αλλάζοντας τα ονόματα των σημάτων του αρχείου εισόδου και βεβαιώνοντας ότι η έξοδος παραμένει σωστή, ενώ για την Άσκηση 4 διαβάζετε ένα αρχείο TXT και δημιουργείτε ένα νέο (αν στην άσκηση 3 δεν διαβάζετε ξεχωριστό αρχείο αλλά έχετε ενιαίο κώδικα μπορείτε να τον χρησιμοποιήσετε ως έχει).

3.0 ΣΗΜΑΝΤΙΚΕΣ «ΛΕΠΤΟΜΕΡΕΙΕΣ»

Έχουμε δώσει format για διάφορα πράγματα. Μπορεί να υπάρχει τυχόν λαθάκι ή ασυνέπεια π.χ. σε χαρακτήρες που χωρίζουν πεδία (delimiters), είτε στην ίδια την εκφώνηση είτε στον δικό σας τρόπο υλοποίησής της άσκησης. Σε αυτή την περίπτωση παρακαλώ στείλετέ μου e-mail στο adollas@tuc.gr ώστε να το διευθετήσουμε, αλλά θεωρήσετε ότι ακόμη και σε μία τέοια περίπτωση δεν χρειάζεται να αλλάξετε τους κώδικες των ασκήσεων 2 και 3, αλλά σημειώσετε στην αναφορά τυχόν μικροαλλαγές στο δικό σας format από εκείνο της εκφώνησης.

ΚΑΛΗ ΕΠΙΤΥΧΙΑ!!!