

**συστηματα μικρουπολογιστων**

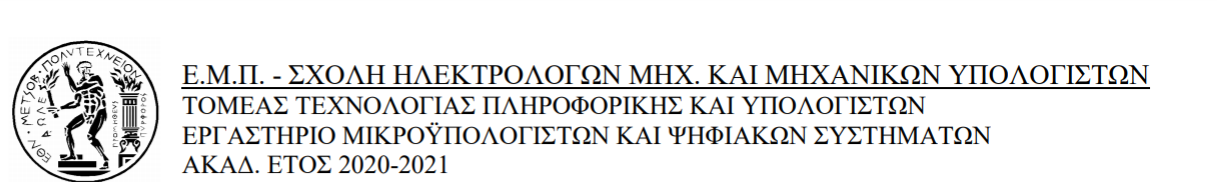
**1η σειρα ασκησεων**



April 14, 2021

ΘΟΔΩΡΗΣ ΑΡΑΠΗΣ – EL18028

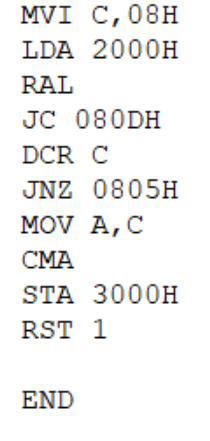
ΚΡΙΣ ΚΟΥΤΣΗ – EL18905

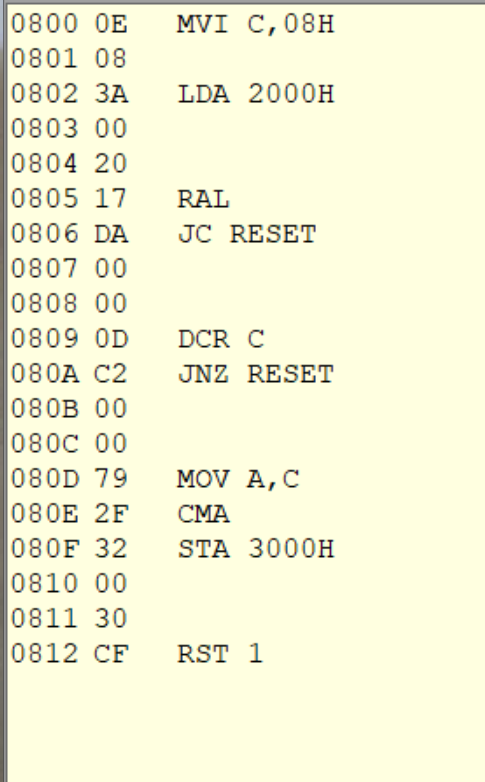


**ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ**

**ΑΣΚΗΣΕΙΣ ΠΡΟΣΟΜΟΙΩΣΗΣ**

**Άσκηση 1**

Μεταφράζοντας το πρόγραμμα της άσκησης από γλώσσα μηχανής σε γλώσσα Assembly λαμβάνουμε το παρακάτω αριστερά:

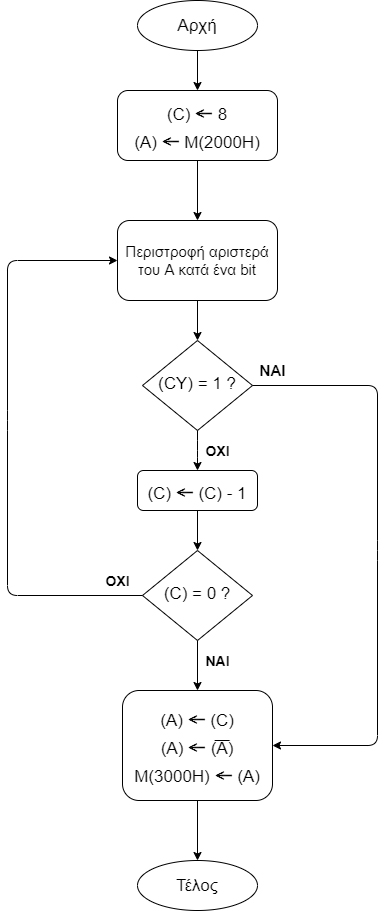


Περιέχεται στο αρχείο *Translated\_in\_Assembly.8085.* Προκειμένου, όμως, να μεταφραστεί και να εκτελεστεί θα πρέπει να αντικαταστήσουμε τις διευθύνσεις των συνθηκών άλματος με ετικέτες, όπως φαίνεται στο τελικό πρόγραμμα, γιατί αλλιώς θα αποτύχει η μετάφραση.

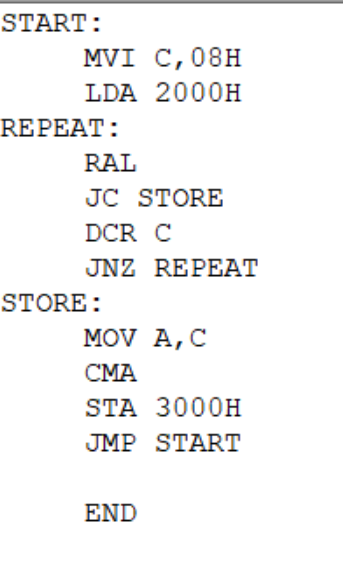
Η λειτουργία του προγράμματος αυτού είναι να εμφανίζει σε δυαδική αναπαράσταση στα φωτάκια LED (αριθμημένα από δεξιά προς τα αριστερά κατά 1, 2,… ,8) τον αριθμό της θέσης του πιο αριστερού dip switch που είναι ON ή, από άλλη οπτική, επιστρέφει το συμπλήρωμα ως προς 1 (αριθμημένα από αριστερά προς τα δεξιά κατά 0, 1,… ,7) του αριθμού της θέσης του πιο αριστερού dip switch που είναι ON. Η τιμή αυτή αναπαριστάται δυαδικά από τα αναμένα LEDs (Αντίστροφη λογική, .)

Δεξιά παρατίθεται το αντίστοιχο πρόγραμμα με τις συμβολικές διευθύνσεις.

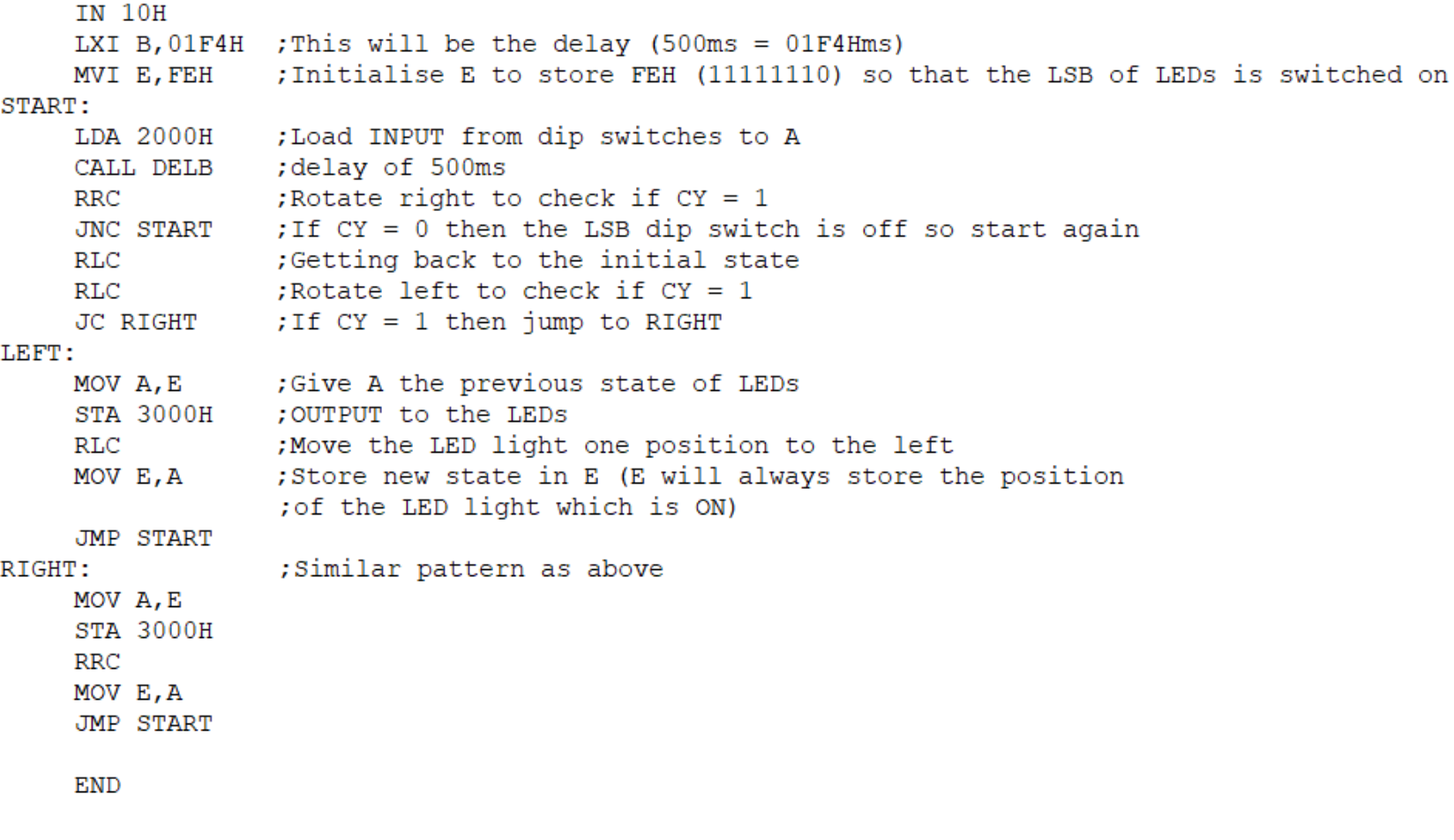
Το διάγραμμα ροής του προγράμματος είναι το ακόλουθο:



Προκειμένου να αποκτήσει συνεχή λειτουργία το πρόγραμμα, αρχικά μπορούμε βγάλουμε την εντολή RST 1 (η οποία καταλαμβάνει και ένα byte μνήμης και η οποία θέτει το ) και στην θέση της να βάλουμε μία συνθήκη άλματος JMP START, όπου START είναι η ετικέτα της διεύθυνσης 0800, δηλαδη της αρχής του προγράμματος. Ως εκτούτου, θα έχουμε το εξής πρόγραμμα που περιέχεται στο αρχείο *Modified\_to\_run\_forever.8085*:

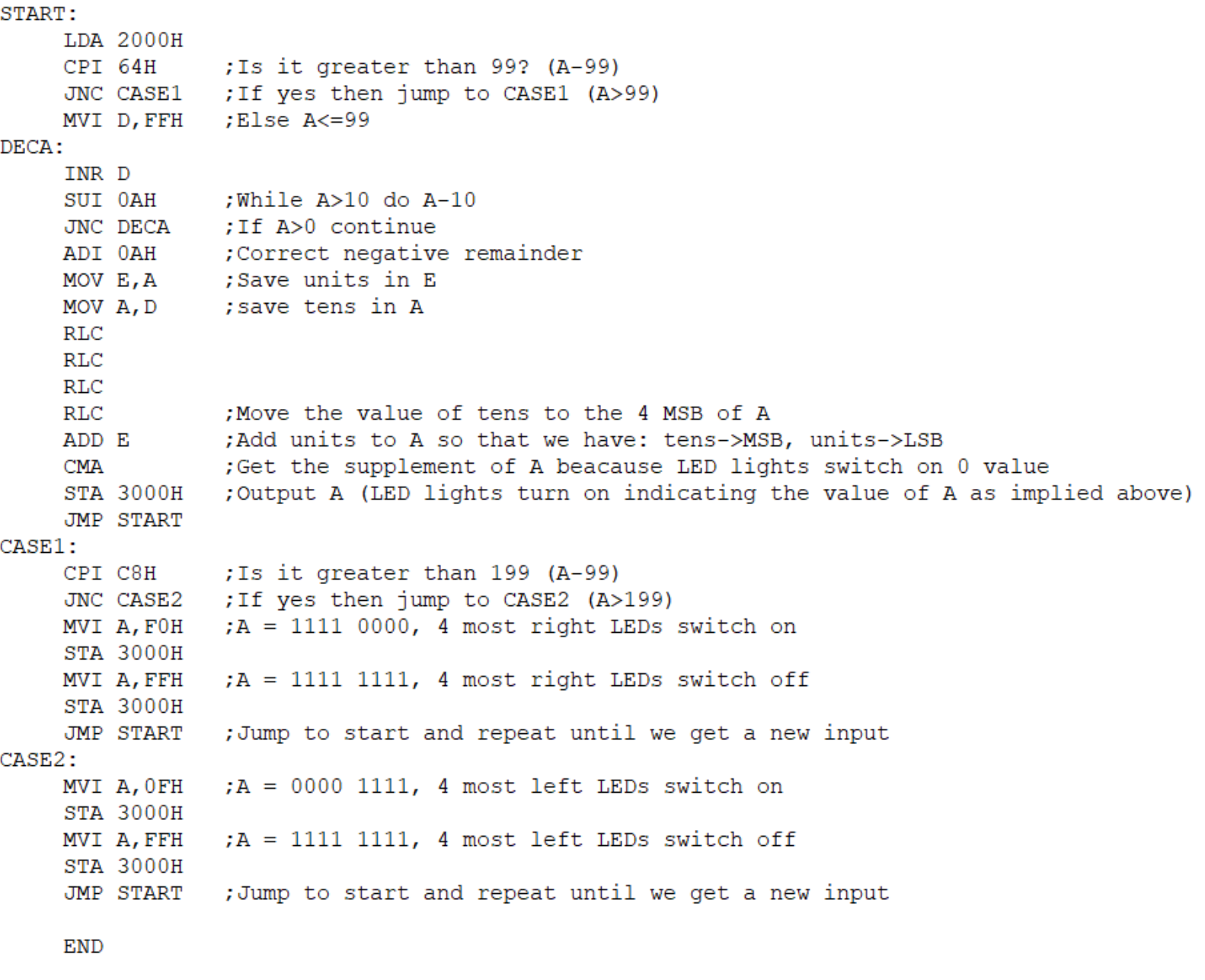


**Άσκηση 2**



Το πρόγραμμα βρίσκεται στο αρχείο *Άσκηση\_2.8085.*

**Άσκηση 3**



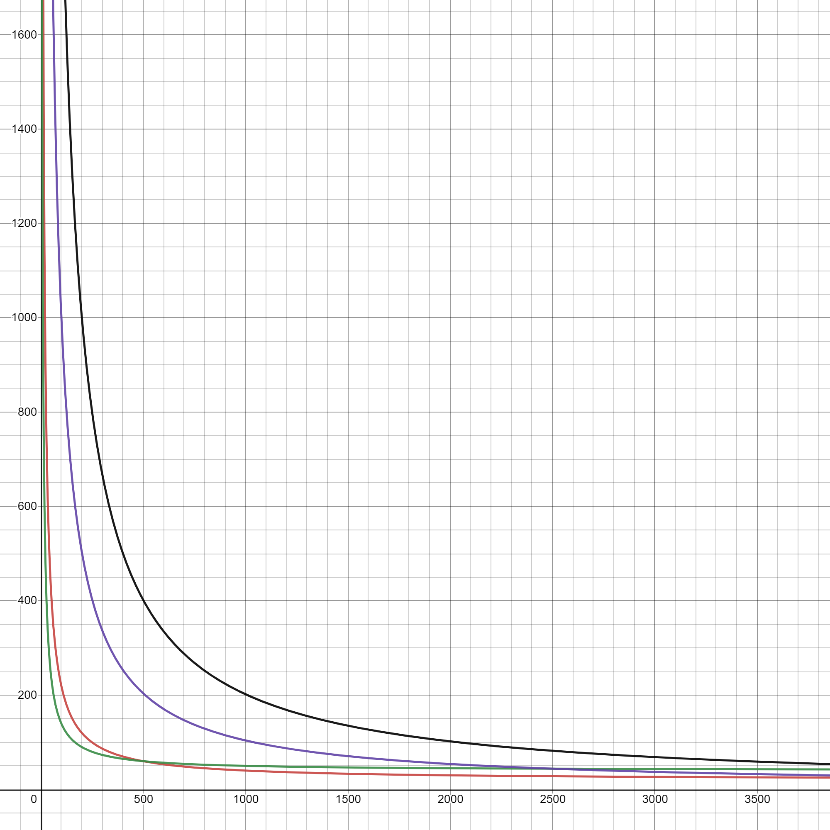
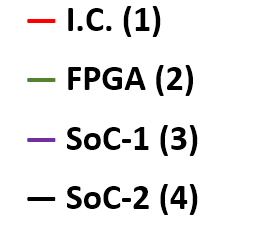
Το πρόγραμμα βρίσκεται στο αρχείο *Άσκηση\_3.8085.*

**Άσκηση 4**

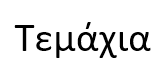
Αρχικά, με βάση τα δεδομένα, εξάγουμε τις σχέσεις κόστους ανά τεμάχιο και έχουμε τις παρακάτω συναρτήσεις κόστους ανά τεμάχιο :



Οι αντίστοιχες καμπύλες φαίνονται παρακάτω:







Στη συνέχεια εξισώνουμε κάθε συνάρτηση κόστους σε ζεύγη και λαμβάνουμε τα εξής σημεία τομής των καμπυλών:

**1-2:**

**1-3:**

**1-4:**

**2-3:**

**2-4:**

**3-4:**

Σύμφωνα με τα παραπάνω και σε συνδυασμό με το διάγραμμα των γραφικών παραστάσεων, συμπεραίνουμε πως οι συμφερότερες περιοχές τεμαχίων για κάθε τεχνολογία είναι:

**(1) I.C.:**

**(2) FPGA:**

**(3) SoC-1:**

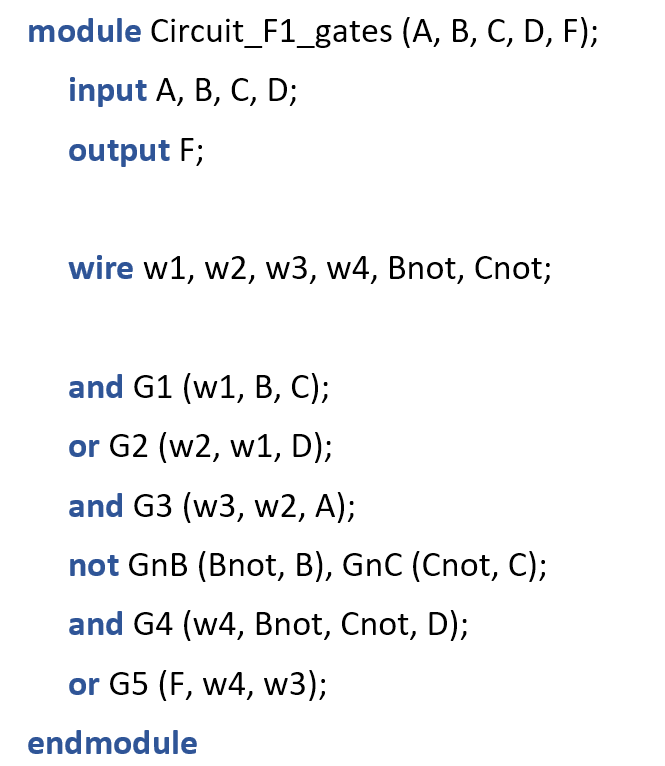
**(4) SoC-2.:**

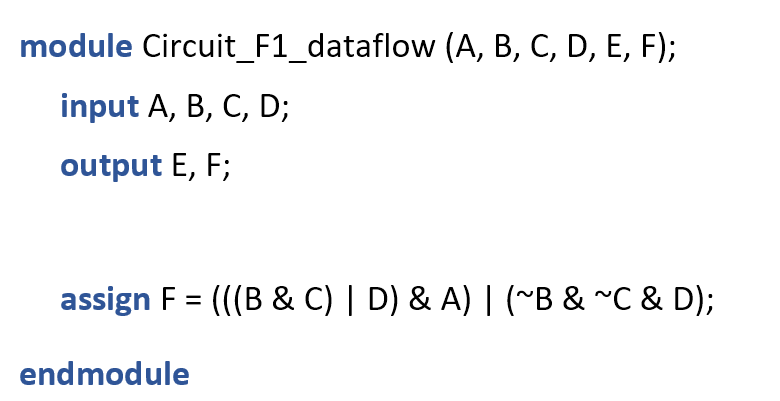
Εύκολα παρατηρούμε πως οι τεχνολογίες με το μεγαλύτερο κόστος σχεδίασης (SoC-1, SoC-2) είναι πιο κερδοφόρες για υψηλότερους αριθμούς τεμαχίων.

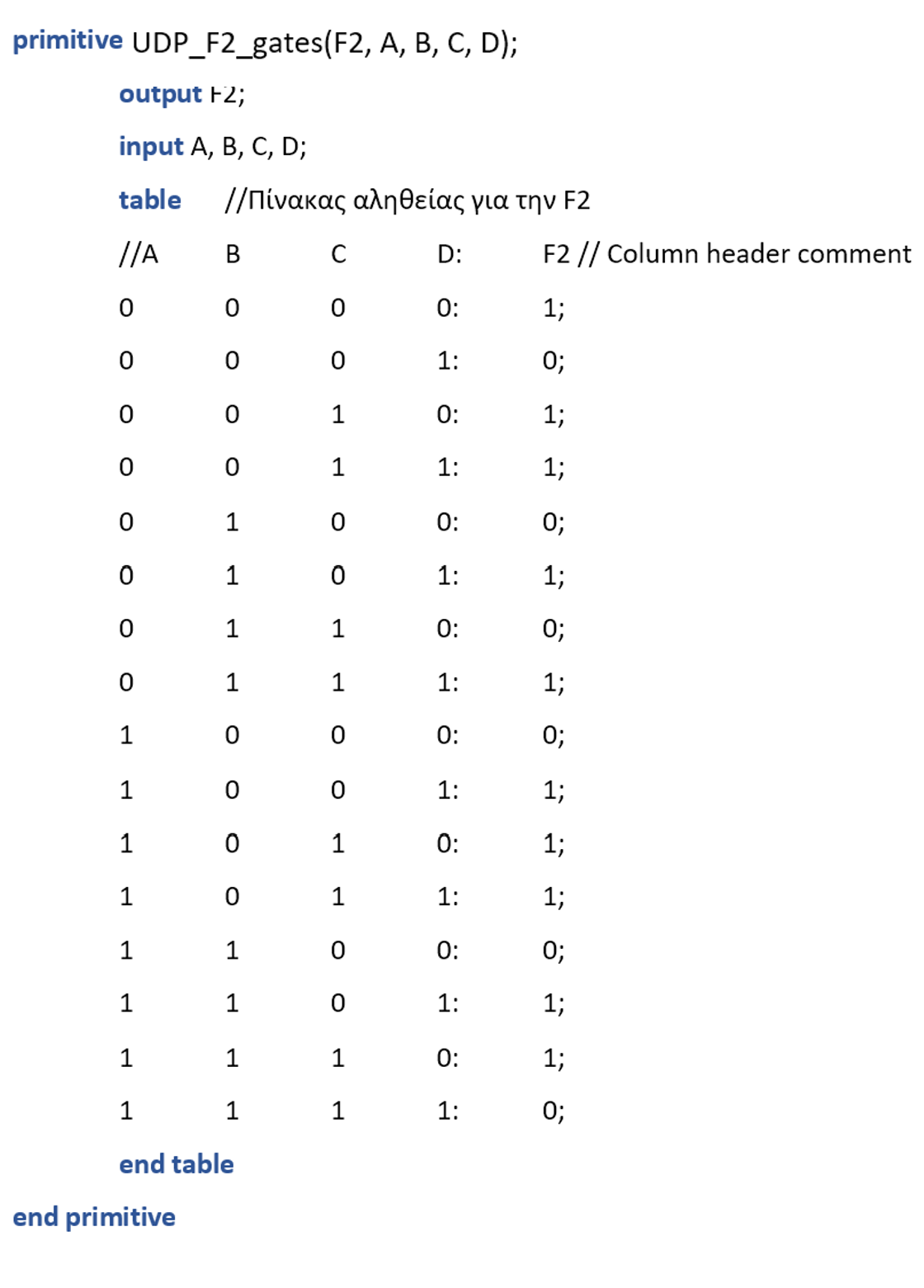
Προκειμένουν να εξαφανιστεί η επιλογή της πρώτης τεχνολογίας θα πρέπει, για το νέο κόστος (έστω ) ανά I.C. της τεχνολογίας των FPGAs, να ισχύει:

Για θα πρέπει να ισχύει , αλλά επειδή προφανώς δεν μπορούν να σχεδιαστούν άπειρα τεμάχια αρκεί να ισχύει . Άρα, οποιαδήποτε τιμή από 0 εώς 10€ για το νέο κόστος αρκεί για να αποκλειστεί η πρώτη τεχνολογία.

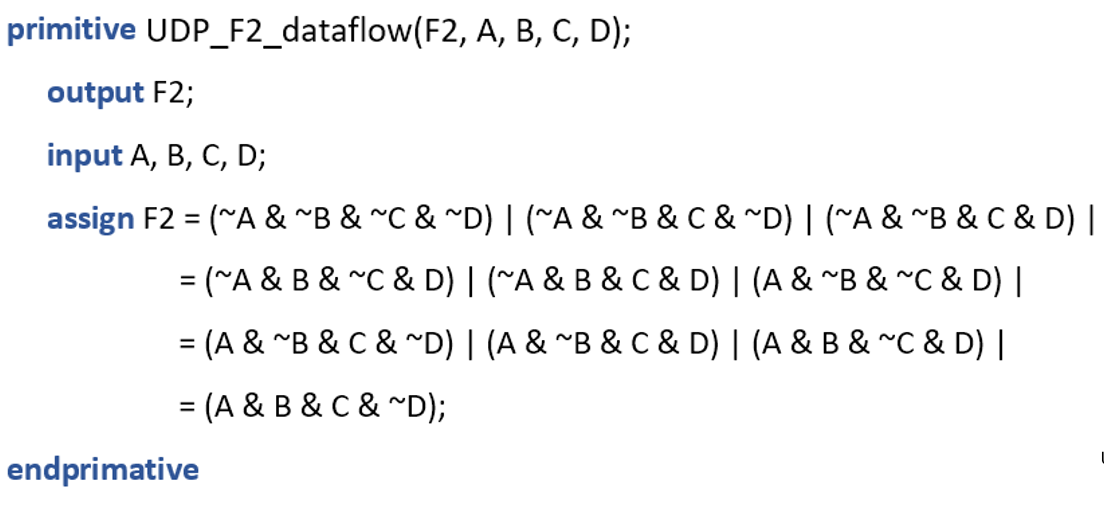
**Άσκηση 5**

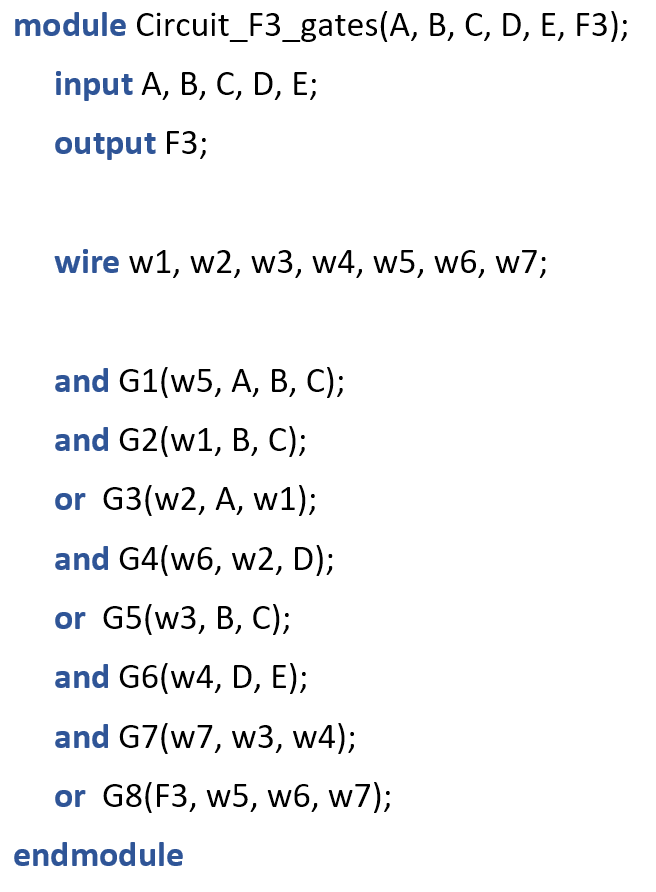
**(i) (ii)**

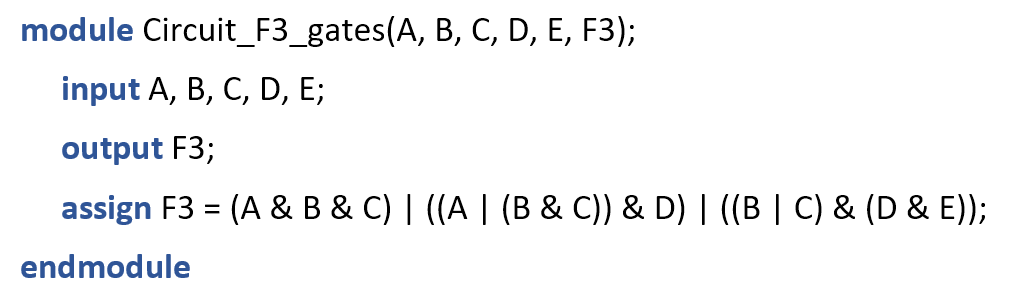


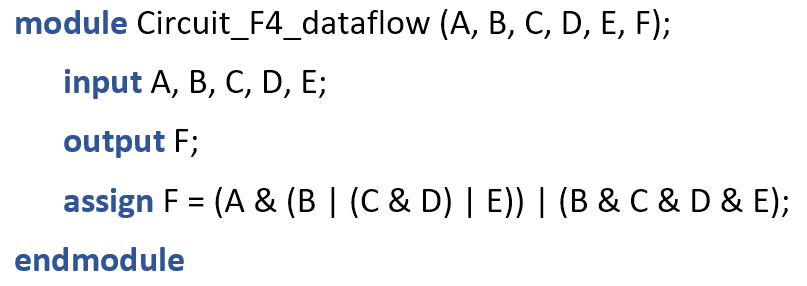
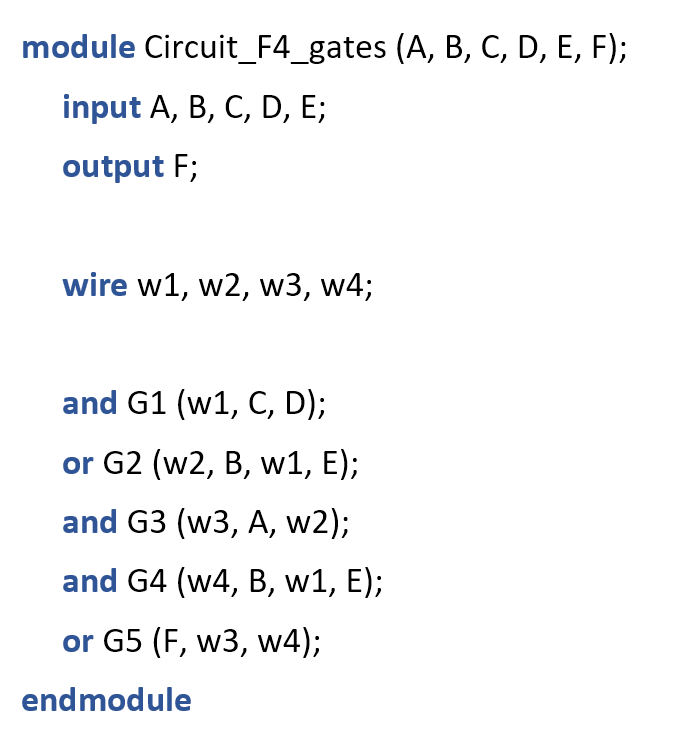






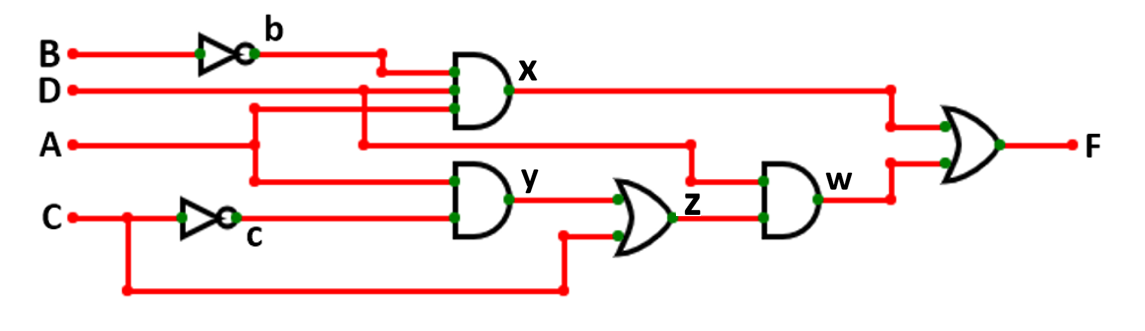


 **module** Circuit\_F3\_dataflow (A, B, C, D, E, F3)

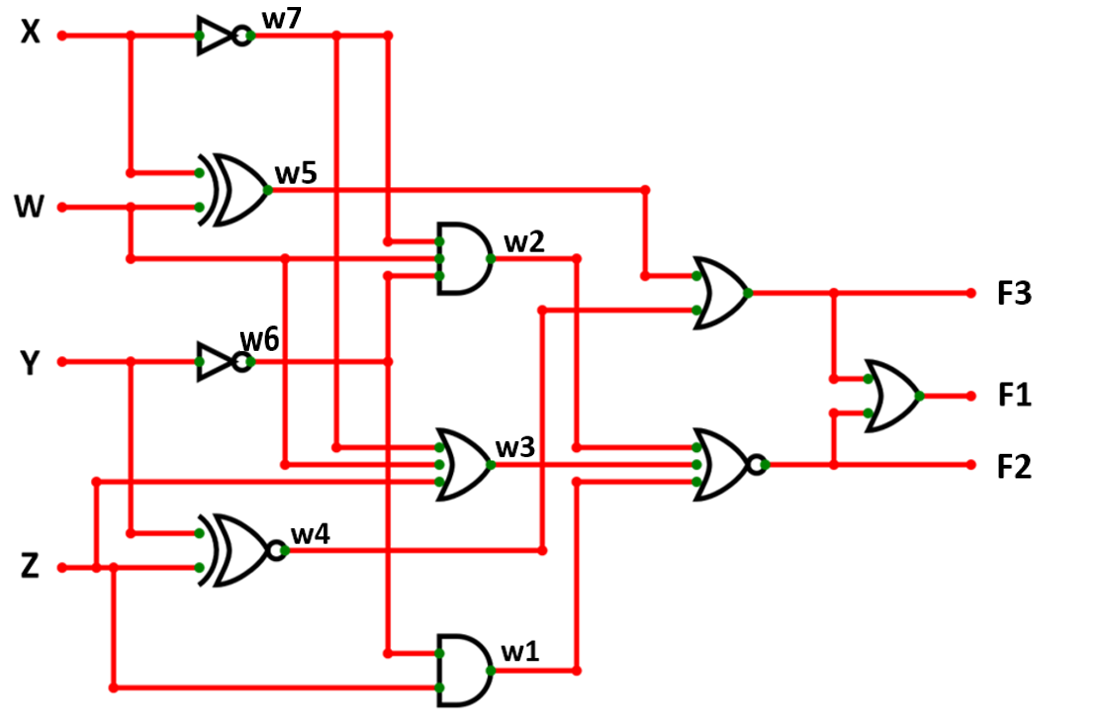
 

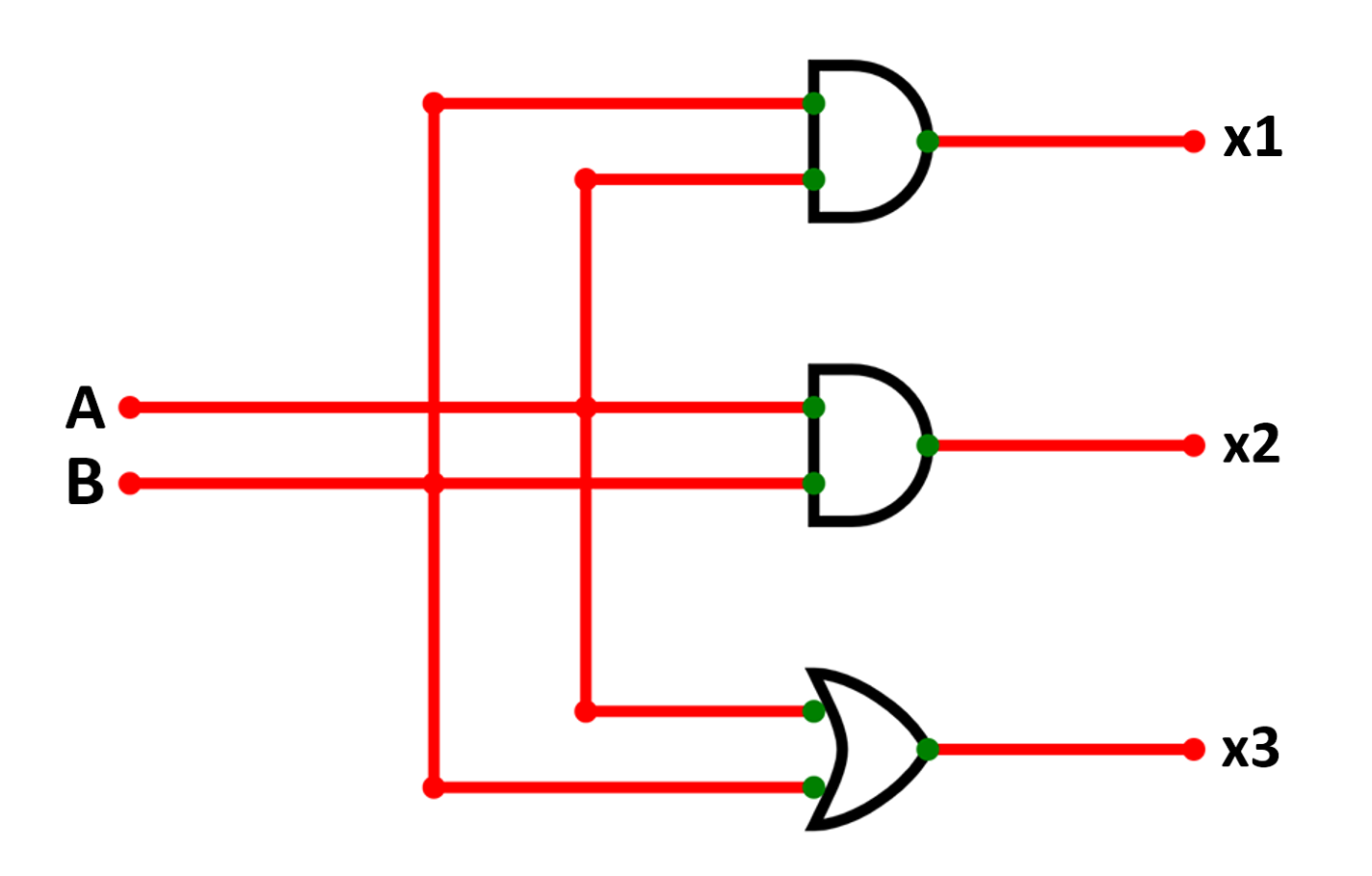
**Άσκηση 6**

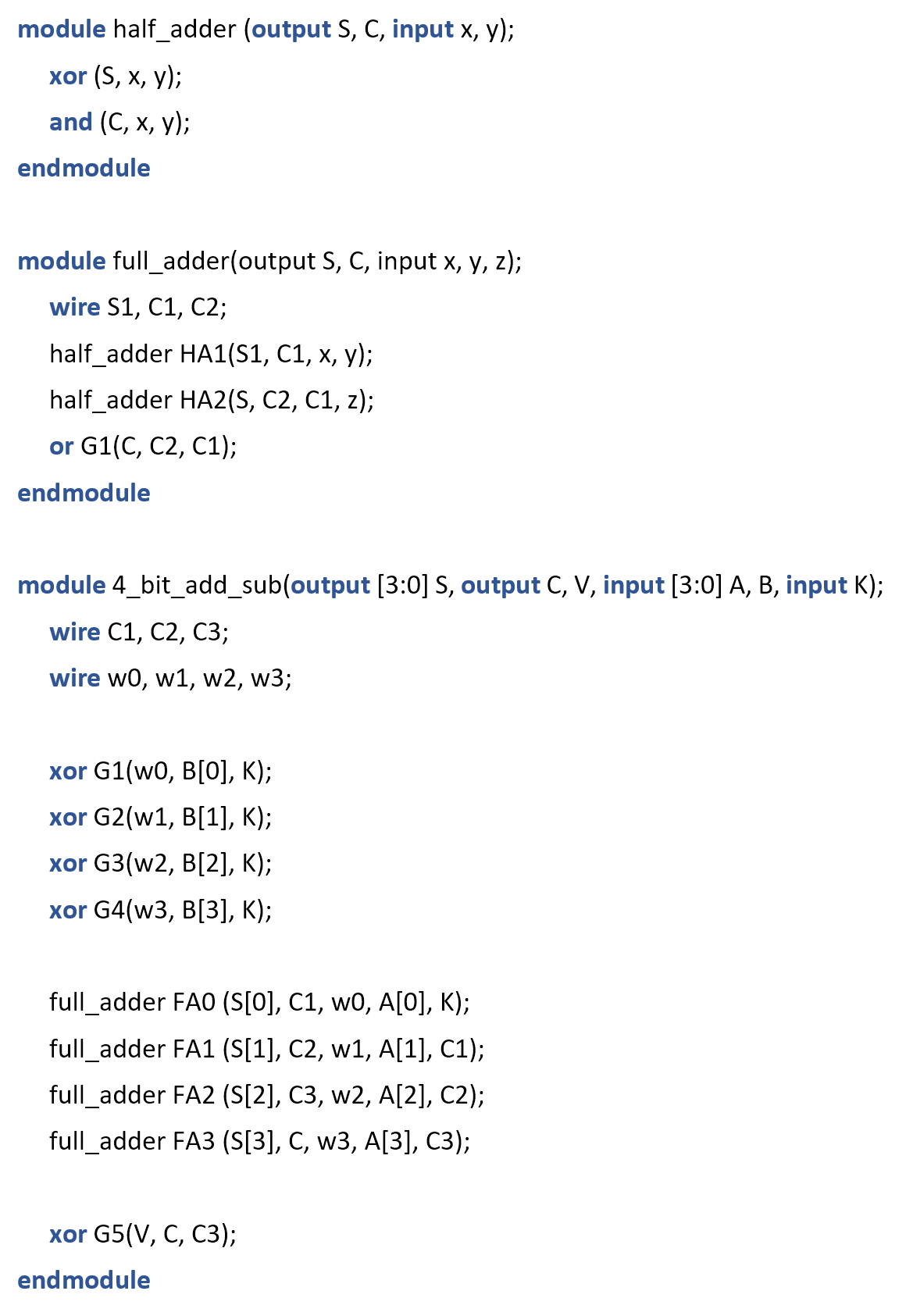
**(i)**

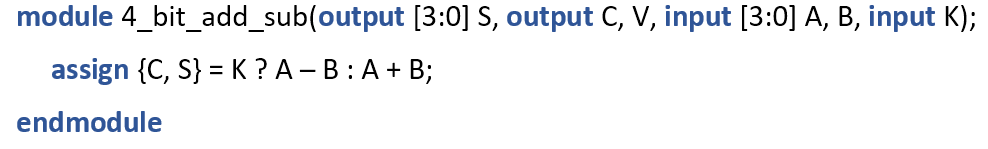
**(a)**

**(b)**



**(c)**

**(ii)**

**(iii)**

**Άσκηση 7**

**(i)**

**module** Mealy\_Machine (**output** reg out, **input** in, clock, reset);

**reg** [1:0] state, next\_state;

**parameter** a = 2’b00,

b = 2’b11,

c = 2’b10,

d = 2’b01;

**always @** (**posedge** clock, **negedge** reset) // state update or reset

**if** (reset == 0) state <= a;

**else** state <= next\_state;

**always @** (state, in)

**case** (state)

a: **if** (in) next\_state = a; **else** next\_state = d;

b: **if** (in) next\_state = a; **else** next\_state = c;

c: **if** (in) next\_state = b; **else** next\_state = d;

d: **if** (in) next\_state = d; **else** next\_state = c;

**endcase**

**always @** (state, in) // output formation

**case** (state)

a, c: out = ~in;

b, d: out = in;

**endcase**

**endmodule**

**(ii)**

**module** Moore\_FSM\_ii (**input** in, clock, reset**, output** out);

**Reg** [1: 0] state;

**Parameter** a = 2’b00, b = 2’b01, c = 2’b10, d = 2’b11;

**always @** (**postedge** clock, **negedge** reset)

**if** (reset == 0) state <= a;

**else** **case** (state)

a: **if** (in) state <= a; **else** state <= d;

b: **if** (in) state <= a; **else** state <= c;

c: **if** (in) state <= d; **else** state <= b;

d: **if** (in) state <= d; **else** state <= c;

**endcase**

**always @** (state)

**case** (state)

a, d: out <= 1’b0;

b, c: out <= 1’b1;

**endcase**

**endmodule**