1η ΟΜΑΔΑ ΑΣΚΗΣΕΩΝ
"Συστήματα Μικροϋπολογιστών"

Αϊβαλής Θεόδωρος 03117099

Σαλιαράκης Παύλος 03117135 28/4/2020

1η ΑΣΚΗΣΗ: Μας δίνεται σε γλώσσα μηχανής ο παρακάτω κώδικας:

06 01 3A 00 20 FE 00 CA 13 08 1F DA 12 08 04 C2 0A 08 78 2F 32 00 30 CF

Το πρόγραμμα υποθέτουμε ότι είναι φορτωμένο στη μνήμη με αρχή τη διεύθυνση 0800.

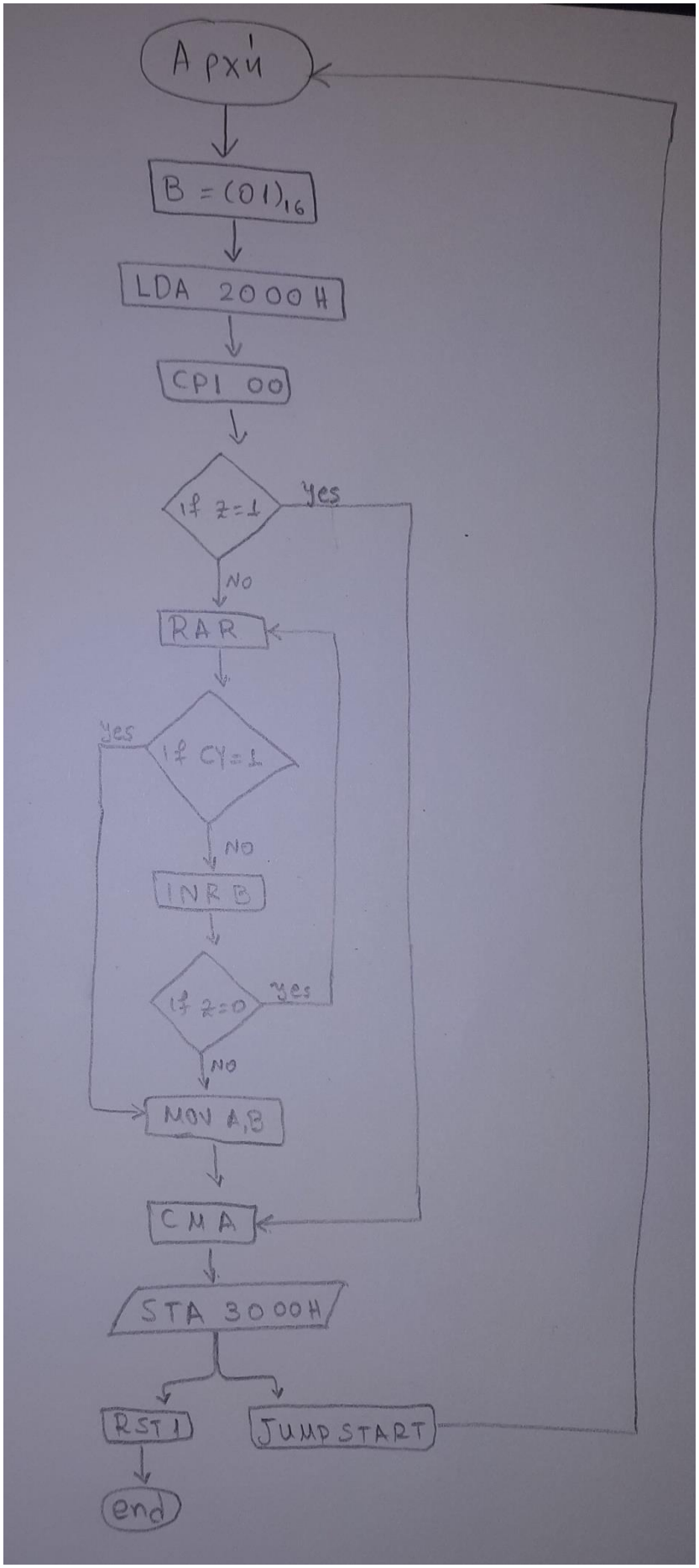
Για τη διαδικασία της αποκωδικοποίησης χρησιμοποιήσαμε τον πίνακα 2 του παραρτήματος 2 των σημειώσεων Εισαγωγή στο Εκπαιδευτικό Σύστημα mLAB.

```
START:
MVI B,01H ; ΜΕΤΑΦΕΡΩ ΣΤΟΝ ΚΑΤΑΧΩΡΗΤΗ Β ΤΟ 01
LDA 2000H ; ΜΕΤΑΦΕΡΩ ΣΤΟΝ ΚΑΤΑΧΩΡΗΤΗ Α ΤΗΝ ΕΙΣΟΔΟ
CPI 00H   ; ΑΦΑΙΡΩ ΑΠΟ ΤΟΝ ΚΑΤΑΧΩΡΗΤΗ Α ΤΟ 00 ΧΩΡΙΣ ΝΑ ΑΛΛΑΞΕΙ ΤΟ ΠΕΡΙΧΟΜΕΝΟ
ΚΑΙ ΕΝΗΜΕΡΩΝΟΝΤΑΙ ΤΑ FLAGS
JZ L1     ; ΑΝ ΤΟ FLAG ΕΙΝΑΙ 1 ΠΑΕΙ ΣΤΟ L1
L3:
RAR       ; ΤΟ ΠΕΡΙΕΧΟΜΕΝΟ ΤΟΥ Α ΠΕΡΙΣΤΡΕΦΕΤΑΙ ΔΕΞΙΑ ΚΑΤΑ ΜΙΑ ΘΕΣΗ ΜΕΣΩ ΤΗΣ
ΣΗΜΑΙΑΣ ΚΡΑΤΟΥΜΕΝΟΥ
JC L2     ; ΑΝ ΤΟ ΚΡΑΤΟΥΜΕΝΟ ΕΙΝΑΙ 1 ΠΑΕΙ ΣΤΟ L2
INR B     ; ΑΥΞΑΝΕΙ ΤΟ ΠΕΡΙΕΧΟΜΕΝΟ ΤΟΥ Β ΚΑΤΑ 1 ΜΟΝΑΔΑ
JNZ L3    ; ΑΝ ΤΟ FLAG ΕΙΝΑΙ 0 ΠΑΕΙ ΣΤΟ L3
L2:
MOV A,B   ; ΜΕΤΑΦΕΡΕΙ ΣΤΟ Α ΤΟ ΠΕΡΙΕΧΟΜΕΝΟ ΤΟΥ Β
L1:
CMA       ; ΒΡΙΣΚΕΙ ΤΟ ΣΥΜΠΛΗΡΩΜΑ ΤΟΥ Α ΚΑΙ ΤΟ ΑΠΟΘΗΚΕΥΕΙ
STA 3000H ; ΤΟ ΠΕΡΙΕΧΟΜΕΝΟ ΤΟΥ Α ΜΕΤΑΦΕΡΕΤΑΙ ΣΤΗΝ ΘΕΣΗ ΜΝΗΜΗΣ 3000
RST 1     ; ΔΙΑΚΟΠΤΕΙ ΤΟΝ ΚΩΔΙΚΑ ΚΑΙ ΠΗΓΑΙΝΕΙ ΣΤΟ 0800(ΣΤΗΝ ΑΡΧΗ)
END
```

Για να τρέχει συνέχεια το πρόγραμμα αντικαθιστώ την εντολή RST 1 με την εντολή JMP START .

Διαπιστώνουμε ότι το παραπάνω πρόγραμμα βρίσκει το 1^{ο} κατά σειρά άσσο (από το LSB) της εισόδου.

Παρακάτω έχουμε το διάγραμμα ροής:


Αρχή
B = (01)16
LDA 2000H
CPI 00
If Z=1
Yes
NO
RAR
yes
If CY=1
NO
INR B
If Z=0
yes
NO
MOV A,B
CMA
STA 3000H
RST 1
JUMP START
end

2η ΑΣΚΗΣΗ: Θέλουμε να γράψουμε σε assembly πρόγραμμα που να απεικονίζει, στη θύρα εξόδου (3000 Hex), ένα αναμμένο led το οποίο να κινείται αριστερά (από το LSB προς το MSB) και όταν φτάνει στο όγδοο (MSB) να κινείται δεξιά (προς το LSB) όταν το LSB της θύρας των dip switch (θύρα εισόδου 2000 Hex) είναι ON. Αλλιώς, οποτεδήποτε είτε στην αρχή είτε ενδιάμεσα, γίνεται OFF το LSB των dip switch, το led να κάνει κυκλική κίνηση. Τέλος, το 2ο LSB της θύρας των dip switch όταν γίνεται ON να ανάβει χωρίς να κινείται το led της θέσης 0. Στη συνέχεια, όταν ξαναγίνει OFF να συνεχίζεται η κίνησή του από το σημείο που είχε μείνει, σύμφωνα με το LSB των dip switch.

```
START:
IN 10H
LXI B,01F4H   ;Εκχωρώ στους καταχωρητές B και C την τιμη 01F4H=500ms των 16
bit για να καλέσω αργότερα την υπορουτίνα DELB για καθυστέρηση 500ms
MVI E,01H     ;Εκχωρώ στον E την τιμή 01H για να ορίσω ότι πρώτα ανάβει το
1ο Led


MAIN:
LDA 2000H     ;Εκχωρώ στον καταχωρητή A την είσοδο από τους διακόπτες
RAR
RAR           ;Κάνω δύο δεξιές περιστροφές στον A προκειμένου να ελέγξω το
δεύτερο LSB
JC LAMP       ;Αν το δεύτερο LSB είναι ίσο με 1 τότε πρέπει να ανάβει συνεχώς
το 10 LED
RAL
RAL           ;Δύο αριστερές περιστροφές για να επαναφέρω το A
CALL DELB     ;Καλώ την ρουτίνα DELB για την καθυστέρηση
RAR           ;Δεξιά περιστροφή του A για να ελέγξω το LSB
JNC LEFT      ;Αν είναι ίσο με 0 κάνω συνεχόμενα άλματα προς τα αριστερά από
εκεί που άναψε το τελευταίο LED
MOV A,E       ;Μεταφέρω στον A τον αριθμό που δείχνει ποιό LED πρέπει να
ανάψει
CPI 01H       ;Συγκρίνω το A με το 01H (πρώτο LED)
JZ LEFT       ;Αν είναι ίσα κάνω άλμα προς τα αριστερά
CPI 80H       ;Συγκρίνω το A με το 80H (τελευταίο LED)
JZ RIGHT      ;Αν είναι ίσα κάνω άλμα προς τα δεξιά
MOV A,D       ;Χρησιμοποιώ τον καταχωρητή ως FLAG και τον μεταφέρω στον A
CPI 00H
JZ LEFT       ;Αν το FLAG είναι ίσο με 0 τότε κάνω άλμα προς τα αριστερά
JMP RIGHT     ;Αλλιώς (αν είναι ίσο με 1) κάνω άλμα προς τα δεξιά

LAMP:
MVI A,01H     ;Εκχωρώ στον A το 01H για να ανάψει το πρώτο LED
CMA           ;Παίρνω το συμπλήρωμα του A διότι τα LED είναι αρνητικής
λογικής
STA 3000H     ;Αποθηκεύω στη θύρα εξόδου την τιμή του A
CMA           ;Επαναφέρω την αρχική τιμή του A
JMP MAIN      ;Κάνω αλμα πίσω στην MAIN προκειμένου να ελέγξω αν έχει αλλάξει
η τιμή της εισόδου των διακοπτών

LEFT:
MOV A,E       ;Μεταφέρω στον A την τιμή που δείχνει ποιό LED πρέπει να ανάψει
CMA           ;Παίρνω το συμπλήρωμα του A
STA 3000H     ;Αποθηκεύω στη θύρα εξόδου το A που δείχνει ποιό LED θα ανάψει
CMA           ;Επαναφέρω την αρχική τιμή του A
RLC           ;Ολισθαίνω μία θέση αριστερά για να οριστεί το επόμενο LED που
θα ανάψει
MOV E,A       ;Μεταφέρω στον E το αποτέλεσμα
MVI D,00H     ;Ενημερώνω τη σημαία D
JMP MAIN      ;Κάνω αλμα πίσω στην MAIN προκειμένου να ελέγξω αν έχει αλλάξει
η τιμή της εισόδου των διακοπτών

RIGHT:
MOV A,E      ;Μεταφέρω στον A την τιμή που δείχνει ποιό LED πρέπει να ανάψει
CMA          ;Παίρνω το συμπλήρωμα του A
```

```
STA 3000H   ;Αποθηκεύω στη θύρα εξόδου το Α που δείχνει ποιό LED θα ανάψει
CMA         ;Επαναφέρω την αρχική τιμή του Α
RRC         ;Ολισθαίνω μία θέση δεξιά για να οριστεί το επόμενο LED που θα
ανάψει
MOV E,A     ;Μεταφέρω στον Ε το αποτέλεσμα
MVI D,01H   ;Ενημερώνω τη σημαία D
JMP MAIN    ;Κάνω άλμα πίσω στην MAIN προκειμένου να ελέγξω αν έχει αλλάξει
η τιμή της εισόδου των διακοπτών
END
```

3η ΑΣΚΗΣΗ:

Θέλουμε να επεκταθεί το 4ο παράδειγμα που αφορά στη μετατροπή δυαδικού αριθμού των 8 bits σε δεκαδική μορφή 2 ψηφίων (σελ. 84 του βιβλίου) χωρίς τον περιορισμό να είναι μικρότεροι του 100(δυαδικό). Τα 8 bit του δυαδικού αριθμού x υποθέτουμε ότι δίνονται από τα dip switches της πόρτας εισόδου (θέση μνήμης 2000 Hex). Το αποτέλεσμα να εμφανισθεί στην πόρτα εξόδου των LED (που αντιστοιχεί στη θέση 3000 Hex) ως εξής: οι μονάδες στα 4 LSB και οι δεκάδες 4 MSB. Στην περίπτωση που ο αριθμός είναι μεγαλύτερος του 99, το αποτέλεσμα να εμφανίζεται σε modulo 100 δηλαδή να εμφανίζεται ο αριθμός x-100 ή x-200. Το πρόγραμμα να είναι συνεχούς λειτουργίας.

```
START:
LDA 2000H ; ΦΟΡΤΩΝΩ ΤΗΝ ΕΙΣΟΔΟ ΣΤΟ Α
LABEL:
SUI 64H   ; ΑΦΑΙΡΩ ΤΟ 100 ΑΠΟ ΤΟΝ Α
JNC LABEL ; ΑΝ ΕΙΝΑΙ ΜΕΓΑΛΥΤΕΡΟ ΤΟΥ 100 ΠΑΕΙ ΣΤΟ LABEL
          ; ΑΝ ΕΙΝΑΙ ΜΙΚΡΟΤΕΡΟ ΤΟΥ 100 ΠΡΟΧΩΡΑΕΙ
MVI D,FFH ; ΒΡΙΣΚΕΙ ΤΟ ΣΥΜΠΛΗΡΩΜΑ ΩΣ ΠΡΟΣ 1
DECA:
INR D     ; ΑΥΞΑΝΩ ΚΑΤΑ 1 ΤΟ D
SUI 0AH   ; ΜΕΙΩΝΩ ΚΑΤΑ 10 ΤΗΝ ΕΙΣΟΔΟ Α
JNC DECA  ; ΑΝ ΕΙΝΑΙ ΘΕΤΙΚΟΣ ΣΥΝΕΧΙΖΕΙ ΝΑ ΑΦΑΙΡΕΙ ΑΛΛΙΩΣ ΠΡΟΧΩΡΑΕΙ
ADI 0AH   ; ΑΝ ΕΙΝΑΙ ΑΡΝΗΤΙΚΟ ΔΙΟΡΘΩΣΕ ΤΟ ΑΡΝΗΤΙΚΟ ΥΠΟΛΟΙΠΟ
MOV E,A   ; ΠΗΓΑΙΝΕΙ ΤΟ Α ΣΤΟ Ε
MOV A,D   ; ΠΗΓΑΙΝΕΙ ΤΟ D ΣΤΟ Α
RLC       ; ΠΑΩ ΤΙΣ 10αδες ΣΤΑ 4 MSB
RLC
RLC
RLC
ADD E     ; ΠΡΟΣΘΕΤΩ ΤΟ Ε ΣΤΟΝ Α ΚΑΙ ΠΑΝΕ ΟΙ ΜΟΝΑΔΕΣ ΣΤΑ 4 MSB
CMA       ; ΒΡΙΣΚΩ ΤΟ ΣΥΜΠΛΗΡΩΜΑ ΤΗΣ Α ΓΙΑΤΙ Η ΕΞΟΔΟΣ ΕΙΝΑΙ ΑΡΝΗΤΙΚΗΣ
ΛΟΓΙΚΗΣ
STA 3000H ; ΑΠΟΘΗΚΕΥΣΗ ΣΤΗΝ ΘΕΣΗ ΜΝΗΜΗΣ ΕΞΟΔΟΥ
JMP START ; ΠΑΕΙ ΠΙΣΩ ΣΤΗΝ ΕΙΣΟΔΟ
END
```

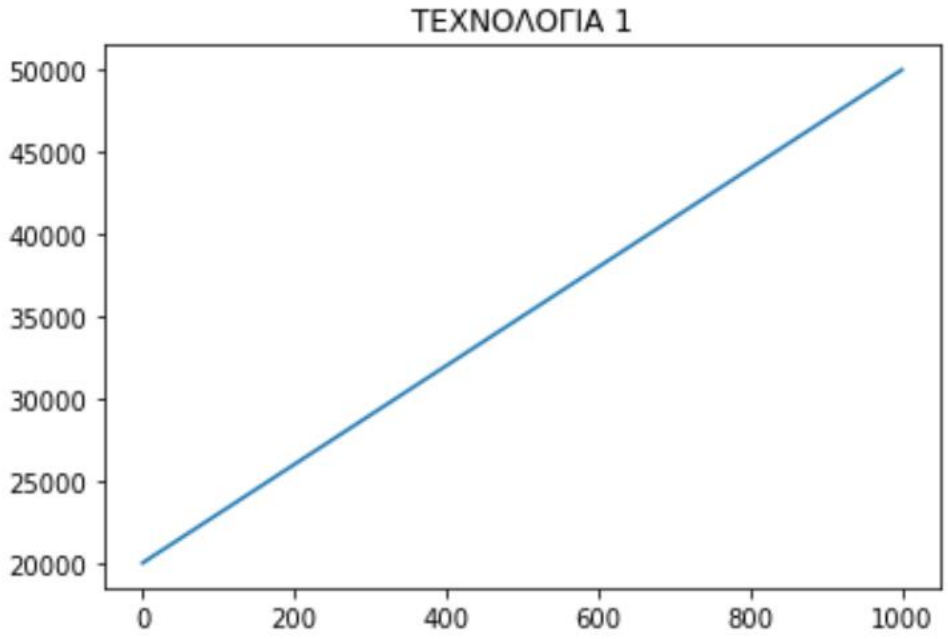
4η ΑΣΚΗΣΗ: Θέλουμε να μελετηθεί από τεχνικοοικονομικής άποψης η κατασκευή μιας φορητής ηλεκτρονικής συσκευής με τη χρήση 3ων διαφορετικών τεχνολογιών:

1.Χρήση διακριτών στοιχείων και ολοκληρωμένων μονάδων (I.C.) όπως μικροελεγκτών, περιφερειακών, μνημών κ.λπ. τα οποία συναρμολογούνται σε μια σε μια σχετικά μεγάλη πλακέτα. Το αρχικό κόστος σχεδίασης του συστήματος υποθέτουμε ότι είναι 20.000€. Το κόστος των I.C. ανά πλακέτα υποθέτουμε ότι είναι 15€ και η κατασκευή της πλακέτας με την συναρμολόγησή της επίσης 15€ ανά πλακέτα.

Η συνάρτηση του κόστους ανά τεμάχιο είναι :

Κόστος =20.000+30x , όπου x: το πλήθος των τεμαχίων

Η καμπύλη κόστους ανά τεμάχιο είναι :

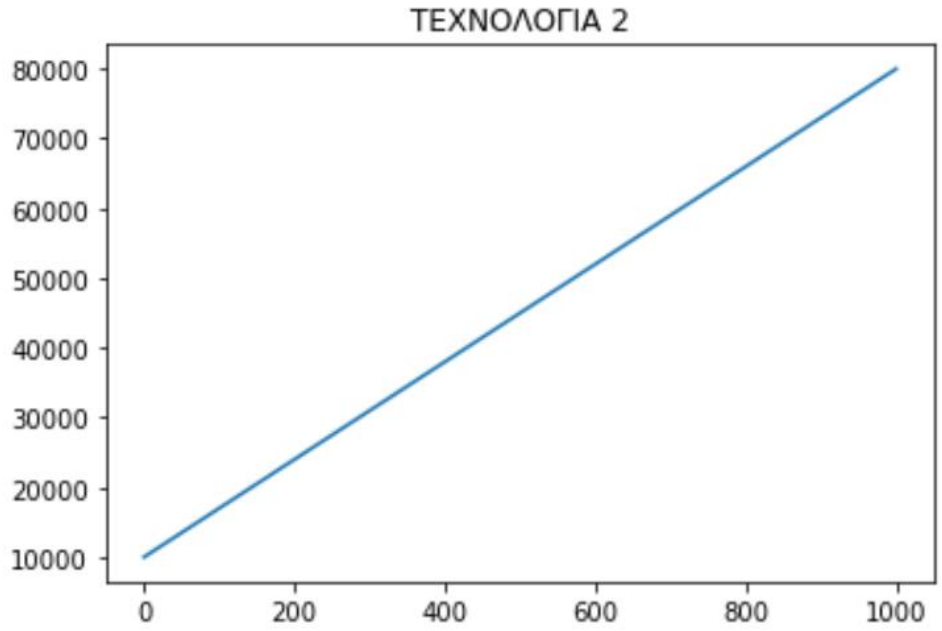


2.Χρήση FPGAs και μικρού αριθμού περιφερειακών τοποθετημένα σε μια μικρή πλακέτα. Αρχικό κόστος σχεδίασης του συστήματος: 10.000€, κόστος ανά πλακέτα των I.C.: 60€, κόστος κατασκευής της πλακέτας και συναρμολόγησης ανά τεμάχιο: 10€.

Η συνάρτηση του κόστους ανά τεμάχιο είναι :

Κόστος =10.000+70x , όπου x: το πλήθος των τεμαχίων

Η καμπύλη κόστους ανά τεμάχιο είναι :

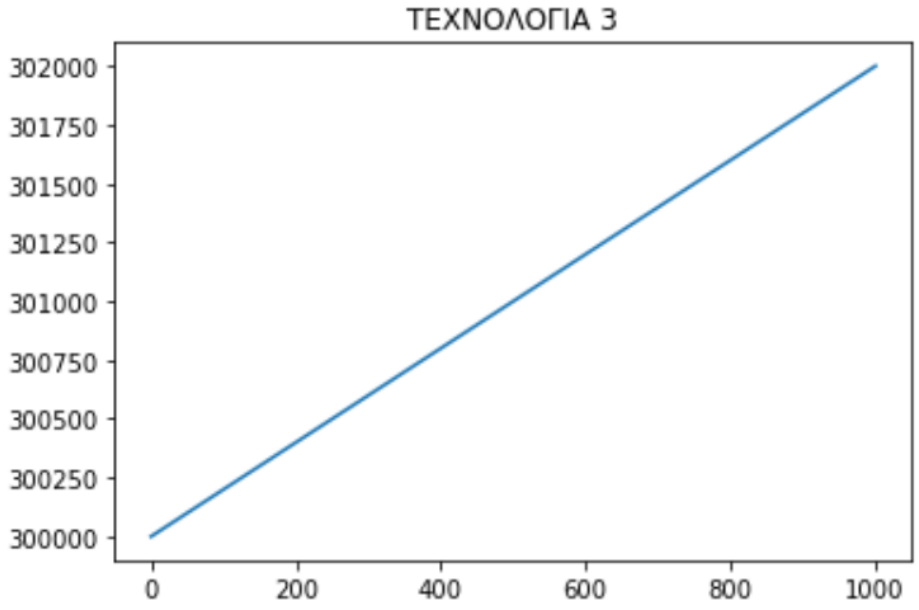


3.Σχεδίαση ειδικού SoC με μια πολύ μικρή πλακέτα. Αρχικό κόστος σχεδίασης: 300.000€, κόστος παραγωγής ανά τεμάχιο των I.C.: 1€, κόστος κατασκευής πλακέτας και συναρμολόγησης ανά τεμάχιο: 1€.

Η συνάρτηση του κόστους ανά τεμάχιο είναι :

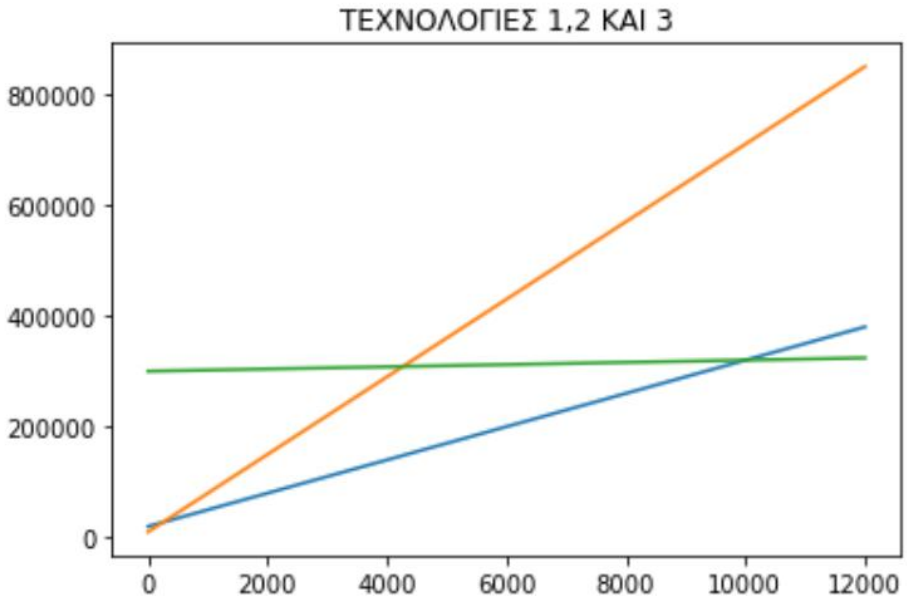
Κόστος =300.000+2x , όπου x: το πλήθος των τεμαχίων

Η καμπύλη κόστους ανά τεμάχιο είναι :



Στη συνέχεια, υπολογίζονται οι περιοχές του πλήθους των τεμαχίων για τις οποίες κάθε τεχνολογία θεωρείται καταλληλότερη. Παρατηρείται ότι από τη μία τεχνολογία στην επόμενη, όσο αυξάνεται το αρχικό κόστος της σχεδίασης, τόσο προτιμότερη είναι η τεχνολογία για μεγαλύτερο πλήθος τεμαχίων.

Αρχικά εδώ βλέπουμε και τις 3 τεχνολογίες μαζί.



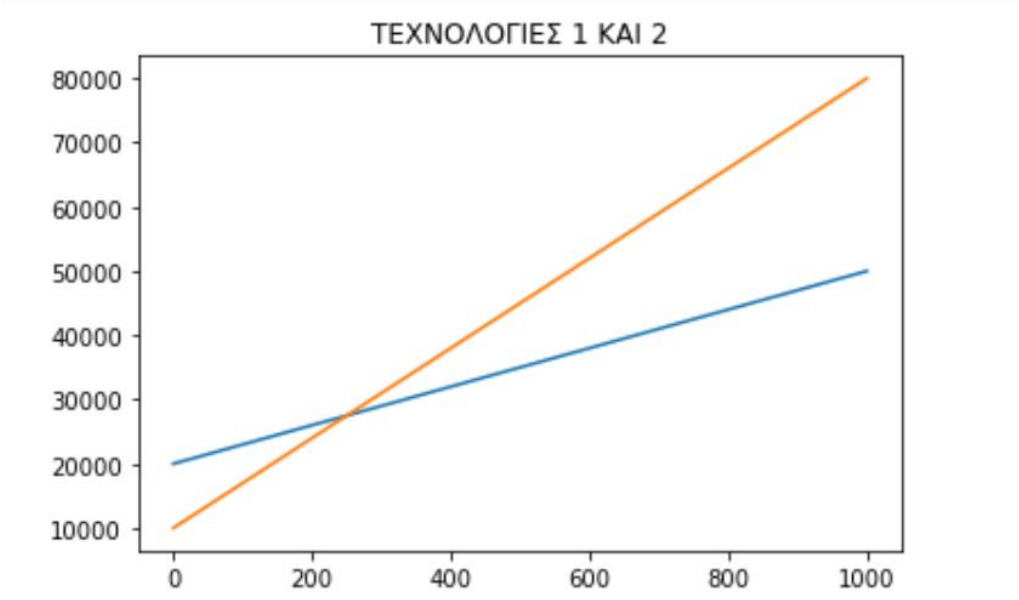
(Μπλε : τεχνολογία 1, Πορτοκαλί : τεχνολογία 2, Πράσινη : τεχνολογία 3)

Οι τεχνολογίες 1 και 2 καταλήγουν να εμφανίζουν ίδιο κόστος για πλήθος τεμαχίων

x1, όπου:

$$20.000+30x_1=10.000+70x_1 \Rightarrow$$
$$\Rightarrow x_1=250$$

Κάτι ανάλογο φαίνεται και στο διάγραμμα.

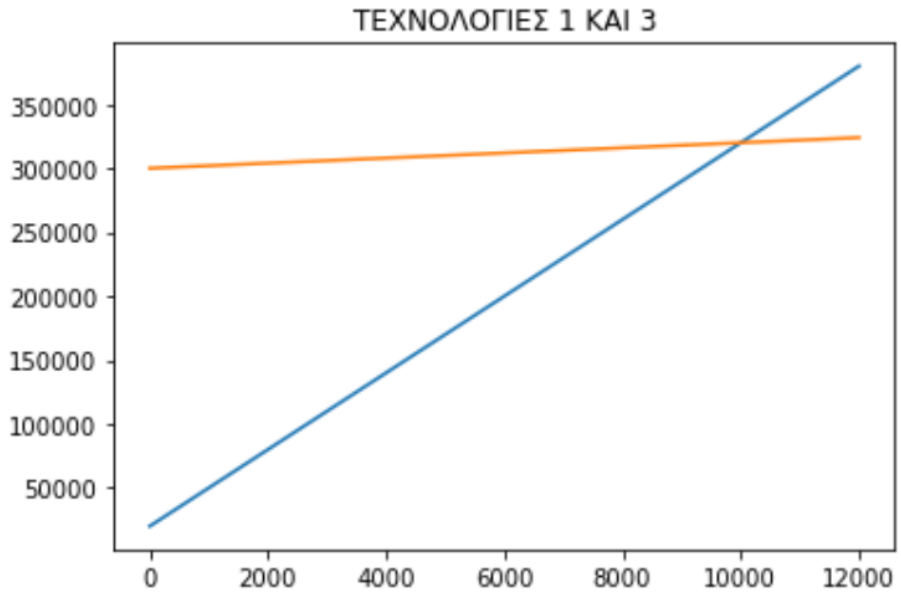


(Μπλε : τεχνολογία 1, Πορτοκαλί : τεχνολογία 2)

Οι τεχνολογίες 1 και 3 καταλήγουν να εμφανίζουν ίδιο κόστος για πλήθος τεμαχίων x2, όπου:

$$20.000+30x_2=300.000+2x_2 \Rightarrow$$
$$\Rightarrow x_2=10.000$$

Κάτι ανάλογο φαίνεται και στο διάγραμμα.



(Μπλε : τεχνολογία 1, Πορτοκαλί : τεχνολογία 3)

Συνολικά και για τις 3 τεχνολογίες βγάζουμε τα εξής συμπεράσματα:

->Η δεύτερη τεχνολογία είναι καλύτερη σε σχέση με την πρώτη για τεμάχια λιγότερα των 250.

->Όσο αυξάνω το x τόσο περισσότερο συμφέρει η πρώτη τεχνολογία.

->Για x μεγαλύτερο από 10.000 και πάνω συμφέρει καλύτερα η τεχνολογία τρία.

Άρα συνοπτικά:

$2^{η}$: 0 έως 250 τεμάχια

$1^{η}$: 250 έως 10.000 τεμάχια

$3^{η}$: πάνω από 10.000 τεμάχια

Τώρα για να μπορέσουμε να εξαφανίσουμε την $1^{η}$ τεχνολογία έχουμε :

Κόστος $1^{ης}$: (20.000/x)+30 ανά τεμάχιο

Κόστος $2^{ης}$: (10.000/x)+70 ανά τεμάχιο

Θέλουμε ουσιαστικά στα 10.000 τεμάχια το κόστος ανά τεμάχιο της $1^{ης}$ τεχνολογίας να είναι μεγαλύτερη από της $2^{ης}$.

Τελικά προκύπτει ότι πρέπει η νέα τιμή για τα I.C.να είναι μικρότερη του 31.

<u>5η ΑΣΚΗΣΗ:</u>

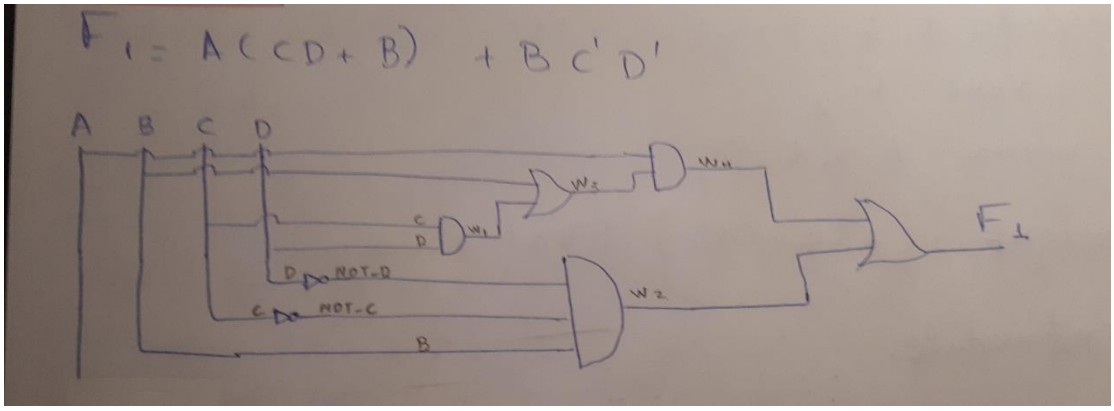
(i),(ii) Δίνεται παρακάτω ο κώδικας σε Verilog για κάθε περίπτωση καθώς και η ανάλυση σε επίπεδο πυλών.

F1=A(CD + B) + BC'D'

```
module F_1(A,B,C,D,F1):
  input A,B,C,D;
  output F1;
  wire NOT_C,NOT_D,w1,w2,w3,w4;
  not
    G1=(NOT_C,C);
    G2=(NOT_D,D);
   and
    G3=(w1,C,D);
    G4=(w2,B,NOT_C,NOT_D);
```

```
    or G5=(w3,B,w1);
    and G6=(w4,A,w3);
    or  G7=(F1,w2,w4);
end module;
```

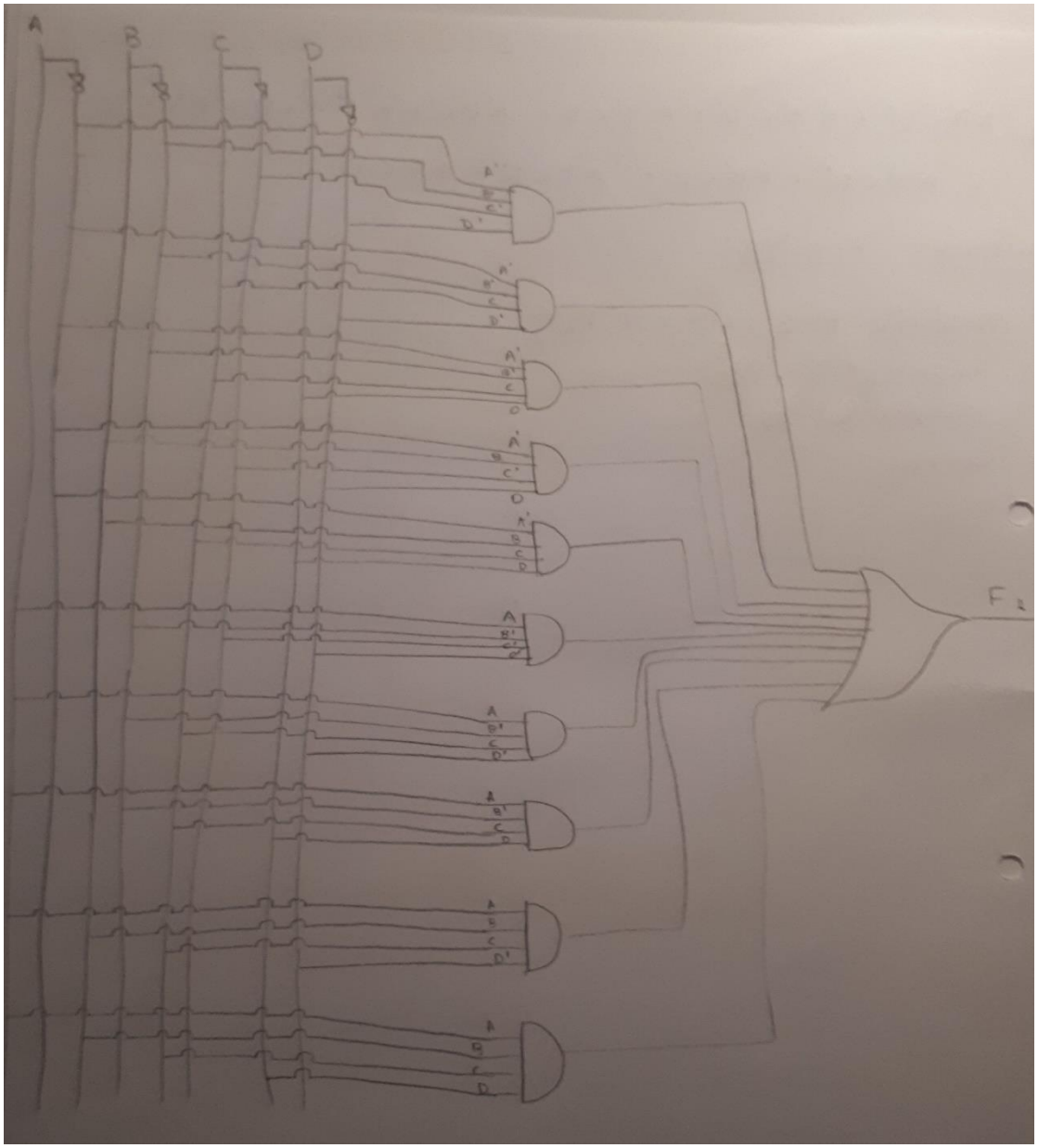
```
module F_1(A,B,C,D,F1):
  input A,B,C,D;
  output F1;
assign F1=(A&((C&D)|B))  |  (B&(~C)&(~D));
end module;
```



F2(A, B, C, D)=Σ (0, 2, 3, 5, 7, 8, 10, 11, 14, 15)

```
module F_2 (A,B,C,D,F2);
 input A,B,C,D;
 output F2;
 wire D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10;
not
   GA=(A_NOT ,A);
   GB=(B_NOT ,B);
```

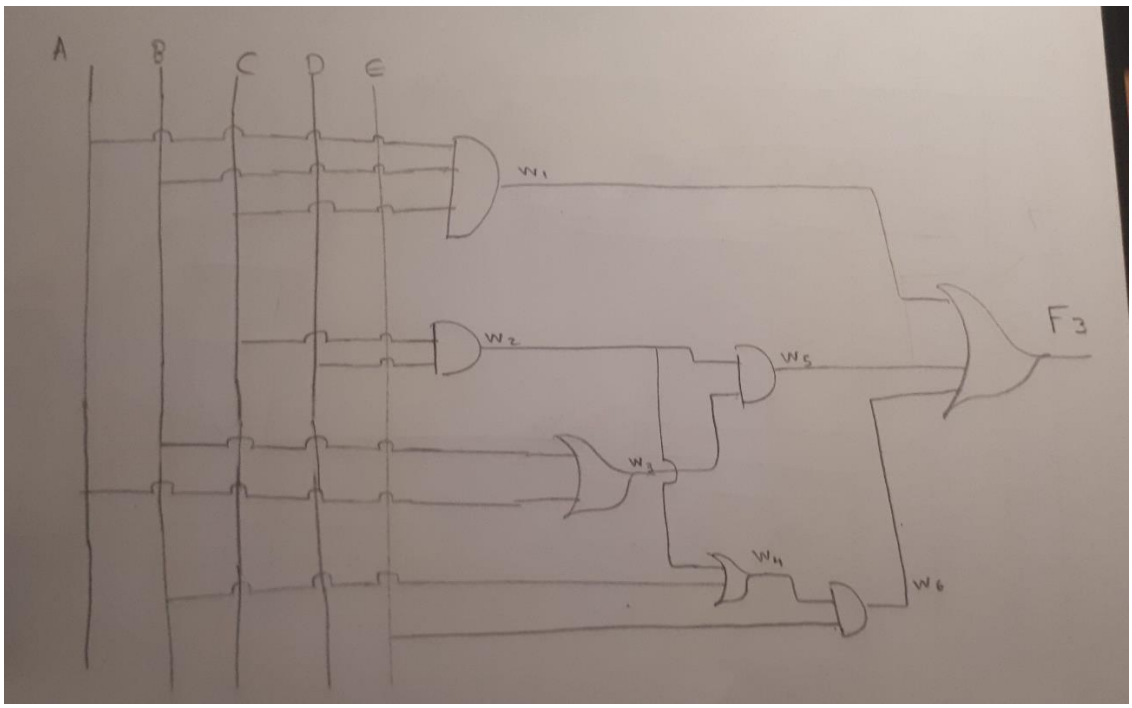
```
   GC=(C_NOT ,C);
   GD=(D_NOT ,D);
and
   G0 = (D0, A_NOT ,B_NOT ,C_NOT ,D_NOT);
   G2 = (D2, A_NOT ,B_NOT ,C ,D_NOT);
   G3 = (D3 ,A_NOT ,B_NOT ,C ,D);
   G5 = (D5, A_NOT ,B ,C_NOT ,D);
   G7 = (D7, A_NOT ,B ,C ,D);
   G8 = (D8, A ,B_NOT ,C_NOT ,D_NOT);
   G10 = (D10 ,A ,B_NOT ,C ,D_NOT);
   G11 = (D11, A, B_NOT ,C, D);
   G14 = (D14, A ,B ,C, D_NOT);
   G15 = (D15, A ,B ,C ,D);
or (F2,D0, D2, D3,D5 , D7, D8, D10, D11 ,D14 ,D15 );
end module;


module F_2 (A,B,C,D,F2);
 input A,B,C,D;
 output F2;
assign F2=((~A)&(~B)&(~C)&(~D)) | ((~A)&(~B)&(C)&(~D)) | ((~A)&(~B)&(C)&(D))
|((~A)&(B)&(~C)&(D)) | ((~A)&(B)&(C)&(D)) | ((A)&(~B)&(~C)&(~D)) |
((A)&((~B)&(C)&(~D)) | ((A)&(~B)&(C)&(D)) | ((A)&(B)&(C)&(~D)) | ((A)&(B)&(C)&(D));
end module;
```



F3=ABC + (A + B)CD +(B+ CD)E

```
module F_3(A,B,C,D,E,F3);
  input A, B, C, D, E;
  output F3;
  wire w1,w2,w3,w4,w5,w6;
  and
    G1=(w1,A,B,C);
    G2(w2,C,D);
```

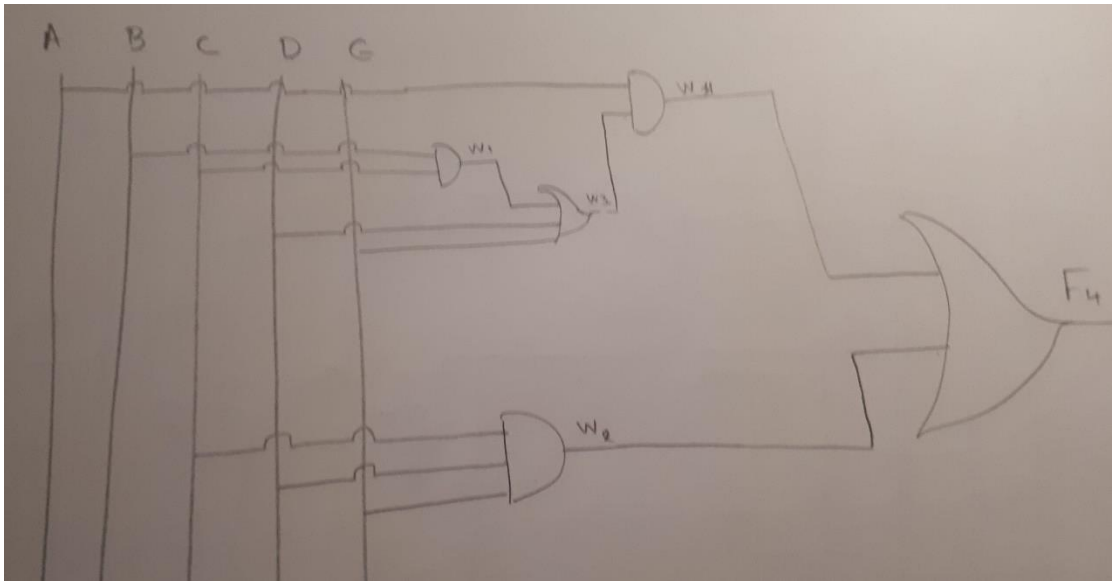
```
 or
    G3=(w3,A,B);
    G4=(w4,w2,B);
 and
    G5=(w5,w2,w3);
    G6=(w6,w4,E);
 or G7=(F3,w1,w5,w6);
end module;

module F_3(A,B,C,D,E,F3);
 input A, B, C, D, E;
 output F3;
 assign F3 =    (A&B&C)   |    (A|B)&(C&D);
```



F4=A(BC + D + E) + CDE

```
module F_4(A,B,C,D,E,F4);
  input A,B,C,D,E;
  output F4;
  wire w1,w2,w3,w4;
  and
    G1=(w1,B,C);
    G2=(w2,C,D,E);
   or G3=(w3,w1,D,E);
   and G4=(w4,A,w3);
   or    G5=(F4,w2,w4);
end module;
```

```
 module F_4(A,B,C,D,E,F4);
  input A,B,C,D,E;
  output F4;
  assign F4=   A&((B&C) | D | E)  |  (C&D&E);
```



6η ΑΣΚΗΣΗ:

(i) Θέλουμε να σχεδιαστεί το λογικό διάγραμμα των ψηφιακών κυκλωμάτων που ορίζονται από τις παρακάτω περιγραφές Verilog:

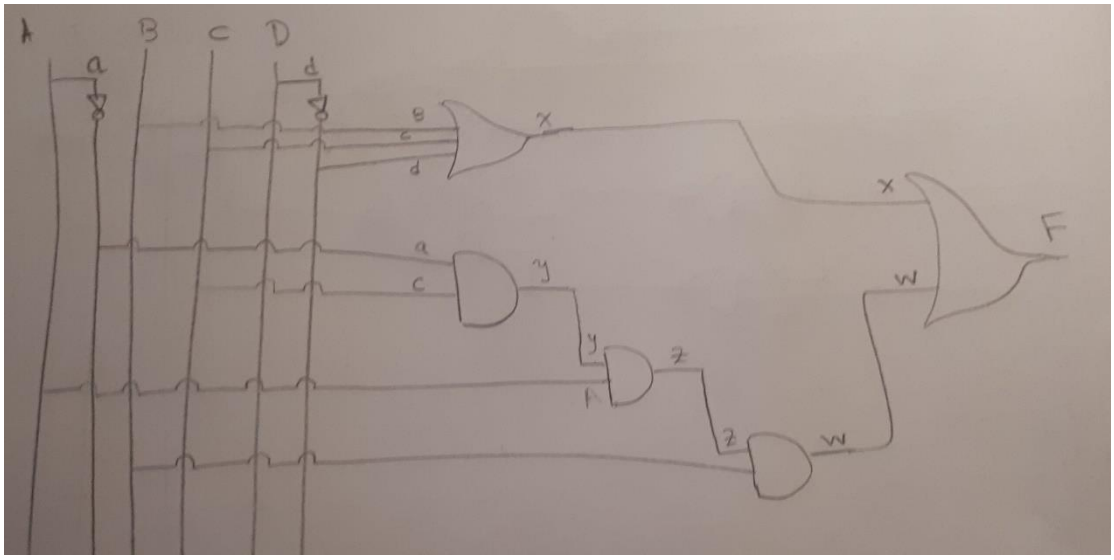
a) module Circuit_A (A, B, C, D, F); input A, B, C, D;
output F;
wire w, x, y, z, a, d;
or (x, B, C, d);
and (y, a ,C);
and (w, z ,B);

and (z, y, A);

or (F, x, w);

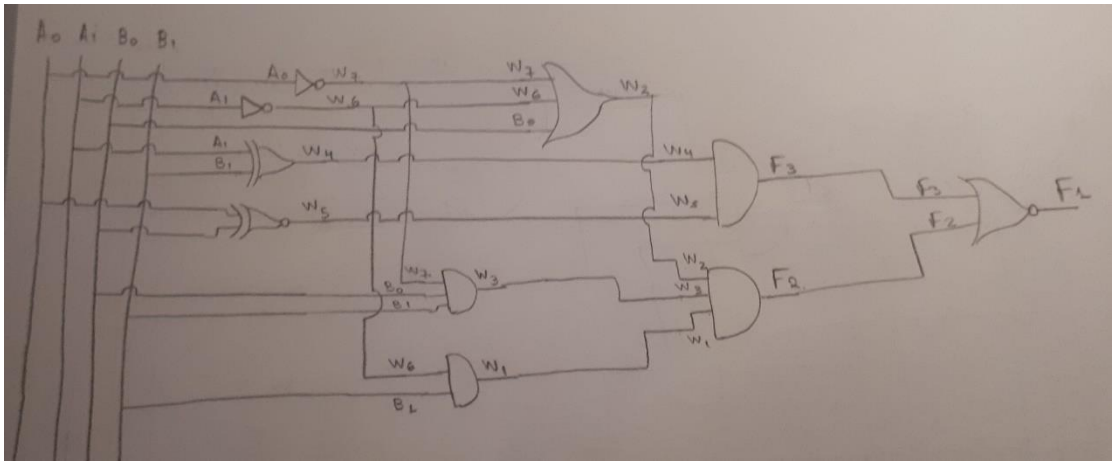
not (a, A); not (d, D);

endmodule



(b) module Circuit_B (F1, F2, F3, A0, A1, B0, B1); output F1, F2, F3;
input A0, A1, B0, B1;
nor (F1, F2, F3);
or (F2, w1, w2, w3);
and (F3, w4, w5);
and (w1, w6, B1);
or (w2, w6, w7, B0);
and (w3, w7, B0, B1);
not (w6, A1);
not (w7, A0);
xor (w4, A1, B1);
xnor (w5, A0, B0);

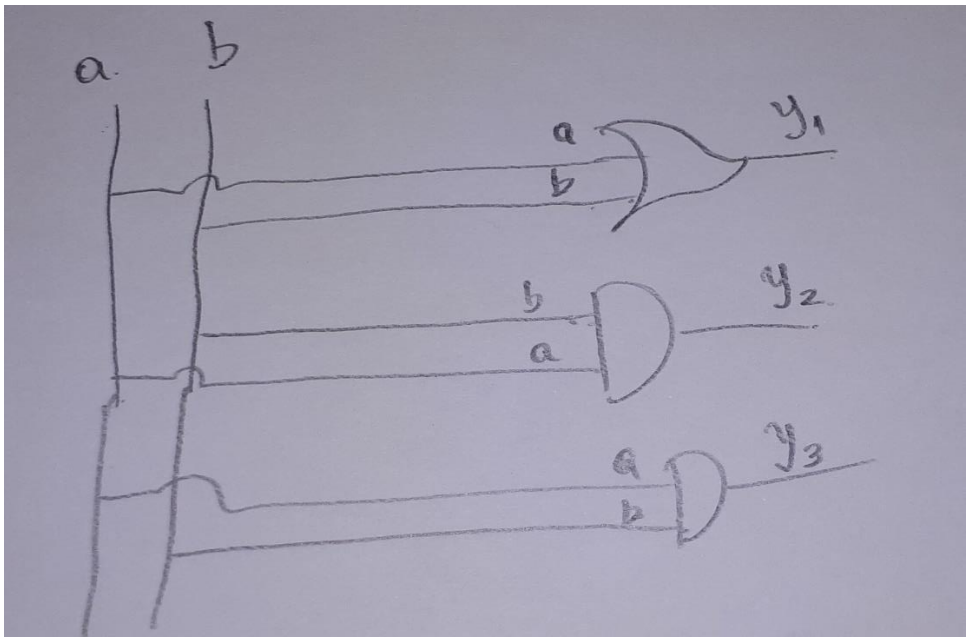
endmodule



(c) module Circuit_C (y1, y2, y3, a, b);

output y1, y2, y3;
input a, b;
assign y1 = a || b;
and (y2, a, b);
assign y3 = a && b;

endmodule



(ii) Βρίσκουμε την ιεραρχική περιγραφή HDL σε επίπεδο πύλης για έναν αθροιστή-αφαιρέτη τεσσάρων bit για μη προσημασμένους δυαδικούς αριθμούς.

Έχουμε την υλοποίηση του ημιαθροιστή και του πλήρους αθροιστή αρχικά.

```
module half_adder(output, s, c, input, x, y);
  xor(s, x, y);
  and(c, x, y);
endmodule

module full_adder (output, s, c, input, x, y, z);
   wire s1,c1,c2;
   half_adder HA1(s1, c1, x, y);
                HA2(s, c2, s1, z);
    or G1(c, c2, c1);
 endmodule


module 4_bit_adder_substracter(output[3:0] sum, output c4, input[3:0] A, B, input M);
wire c1,c2,c3,xo1,xo2,xo3,xo4;
xor   G0(xo1,M,B[0]);
      G1(xo2,M,B[1]);
      G2(xo3,M,B[2]);
      G3(xo4,M,B[3]);
full_adder FA0(sum[0],c1,A[0],xo1,M);
           FA1(sum[1],c2,A[1],xo2,M);
           FA2(sum[2],c3,A[2],xo2,M);
           FA3(sum[3],c4,A[3],xo3,M);
endmodule
```

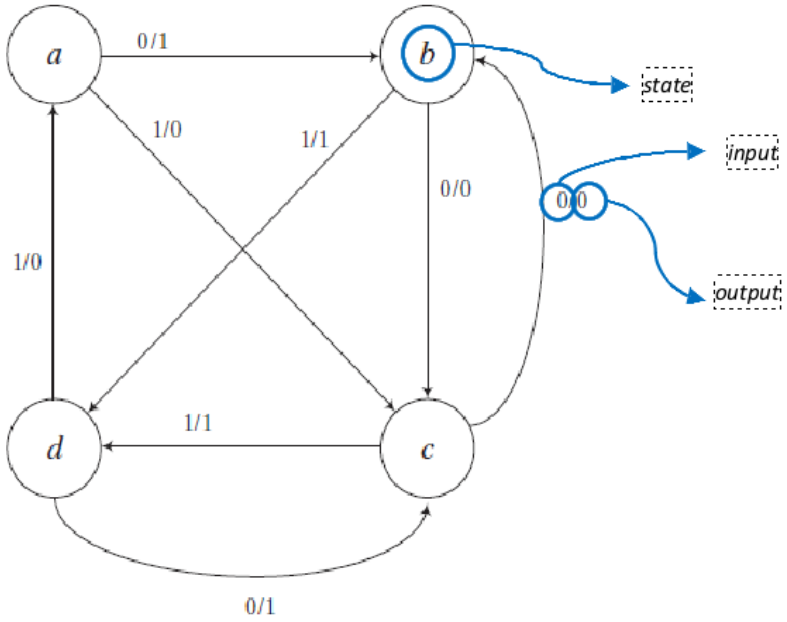
(iii) Χρησιμοποιούμε τον τελεστή υπό συνθήκη (? :) και γράφουμε την περιγραφή ροής δεδομένων HDL ενός αθροιστή-αφαιρέτη τεσσάρων bits μη προσημασμένων αριθμών.

```
module binary_adder_substracter (sum, cout, A, B, M);
 output[3:0]sum;
output cout;
```

```
input [3:0] A, B;
input M;
assign {cout,sum}=(M==0?A+B:A-B);
endmodule
```

7η ΑΣΚΗΣΗ:

(α)




```
module Moore_Module_1(y,x,clock,reset);
  output y;
  input x,clock,reset;
  reg state;
  parameter  S0=a ,S1=b ,S2=c ,S3=d
  always @(posedge clock, negedge reset);
  if (reset==0) state <=S0;
  else case (state)
    S0: if(~x)state<=S1, assign y=1;
        else state<=S2,  assign y=0;
```

S1: if(~x)state<=S2, assign y=0;

else state<=S3, assign y=1;

S2: if(~x)state<=S1, assign y=0;

else state<=S2, assign y=1;

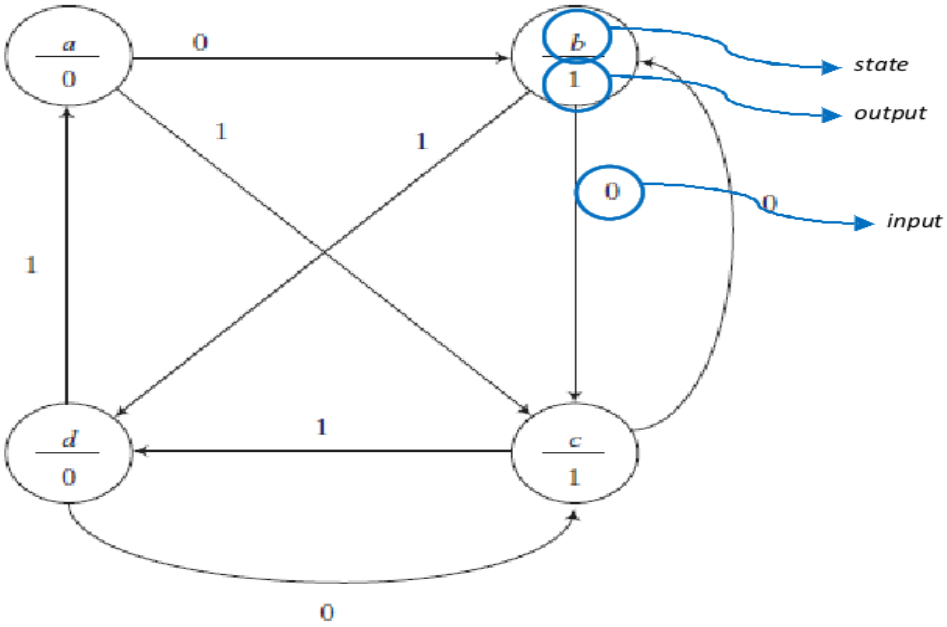
S3: if(~x)state<=S2, assign y=1;

else state<=S0, assign y=0;

endcase

endmodule;

(ii)




```
module Moore_Module_2(y,x,clock,reset);
  output y;
  input x,clock,reset;
  reg state;
  parameter  S0=a ,S1=b ,S2=c ,S3=d
  always @(posedge clock, negedge reset);
  if (reset==0) state <=S0;
  else case (state)
    S0:  assign y=0;
         if(~x)state<=S1   else state<=S2;
```

```
    S1: assign y=1;
        if(~x)state<=S2  else state<=S3;
    S2: assign y=1;
        if(~x)state<=S1  else state<=S2;
    S3:  assign y=0;
        if(~x)state<=S2   else state<=S0;
 endcase
endmodule;
```

(iii) Βρίσκουμε μια περιγραφή συμπεριφοράς Verilog ενός up-down counter των 4 bit που διαθέτει 2 εισόδους (Up/Down και Clear). Χρησιμοποιούμε D-flip flop.

Και του up-down counter

```
module Binary_counter_up_down_4(output reg[3:0]A, input up, down, clock, clear);
always @(posedge clock, negedge clear);
if(~clear)  A<=4'b0000;
else if(up) A<=A+1'b1;
else if(down) A<=A-1'b1;
else A<=A;
endmodule;
```