Εθνικο Μετσοβιο Πολυτεχνειο

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ

1η ΑΣΚΗΣΗ

Αχλάτης Στέφανος-Σταμάτης (03116149)

<el16149@central.ntua.gr>

Καπερώνη Φρειδερίκη (03116685)

<el16685@central.ntua.gr>

Απρίλιος 2020

Περιεχόμενα

1η Άσκηση

Στην πρώτη άσκηση μας δίνονται εντολές μηχανής και καλούμαστε να τις μετατρέψουμε σε εντολές Assembly. Παραθέτουμε τον τελικό κώδικα, όπως αυτός εκτελέστηκε στον MicroLab Simulator 8085. (εκτενής ανάλυση και σχολιασμός άσκηση στο αντίστοιχο παράρτημα παρακάτω)

```
BEGIN:
       MVI B,01H
       LDA 2000H
       CPI 00H
       JZ GOTOEND
MYLOOP:
       RAR
       JC BEFOREND
       INR B
       JNZ MYLOOP
BEFOREND:
       MOV A,B
GOTOEND:
       CMA
       STA 3000H
       RST 1
       END
```

Παρακάτω παραθέτουμε και το μεταφρασμένο πρόγραμμα, όπως προέκυψε από τον MicroLab Simulator 8085 όπου φαίνεται η αντιστοιχία των εντολών με τον κώδικα της μηχανής που δόθηκε και στην εκφώνηση.

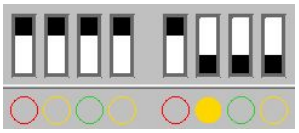
```
BEGIN:
0800   06     MVI B,01H
0801   01
0802   3A     LDA 2000H
0803   00
0804   20
0805   FE     CPI 00H
0806   00
0807   CA     JZ GOTOEND
0808   13
0809   08

MYLOOP:
080A   1F     RAR
080B   DA     JC BEFOREND
080C   12
080D   08
080E   04     INR B
080F   C2     JNZ MYLOOP
0810   0A
0811   08

BEFOREND:
0812   78     MOV A,B

GOTOEND:
0813   2F     CMA
0814   32     STA 3000H
0815   00
0816   30
0817   CF     RST 1
```

ΑΝΑΛΥΤΙΚΑ ΣΧΟΛΙΑ:

- Εκτελώντας το πρόγραμμα στον προσομοιωτή, διαπιστώνουμε ότι ο συσσωρευτής A (accumulator) περιέχει την τιμή που υποδεικνύουν οι διακόπτες (8-bit) και ο καταχωρητής B αρχικοποιείται στην τιμή του δεκαεξαδικού 1. Επίσης, παρατηρούμε ότι για να εκτυπωθεί το περιεχόμενο του καταχωρητή A σε δυαδική μορφή στα leds του μικροϋπολογιστή, πρέπει πρώτα να αντιστρέψουμε το περιεχόμενο του, δηλαδή να κάνουμε το compliment του, και μετά να το τοποθετήσουμε στην θέση μνήμης του μικροϋπολογιστή με τιμή το δεκαεξαδικό 3000.
- Αρχικά γίνεται ο έλεγχος αν η τιμη που έχει δοθεί στον A είναι το μηδέν.Αν είναι ίση με το μηδέν τότε προχωράμε στην εκτύπωση της τιμής του A πάνω στα leds του μικροϋπολογιστή, με την διαδικασία που περιγράψαμε προηγμένος.
- Αν δεν είναι ίση με το μηδέν, κάνουμε δεξιά ολίσθηση του A λαμβάνοντας υπόψην και το κρατούμενο,RAR, και τότε αν CY=1 φορτώνουμε στον A το περιεχόμενο του B και εκτυπώνουμε τον A, αν όμως CY=0 τότε αυξάνουμε την τιμή του B κατα ένα και επαναλαμβάνουμε την διαδικασία κάνοντας δεξια ολίσθηση κλπ.
- Με αυτόν τον τρόπο ο B κρατάει την θέση του πρώτου μη μηδενικού bit του A, δηλαδή για το lsb, αδιαφορόντας για όλα τα υπόλοιπα, και αυτή η θέση εκτυπώνεται στα leds εκτός από την περίπτωση που το A είναι ίσο με το μηδεν όπου τότε εκτυπώνουμε απευθείας το μηδεν στα leds.
- Έτσι, λοιπόν αν περιγράψουμε το πρόγραμμα μας ως ένα σύστημα μπορούμε να πούμε ότι δέχεται έναν 8bit αριθμό μέσω των διακοπτών, όπου ο δεξιότερος αντιστοιχεί στο lsb και ο αριστερότερος στο msb και εκτυπώνει την θέση του lsb δηλαδή αναβει τα led που αντιστοιχούν στην θέση του lsb με βάση δυαδικό σύστημα αρίθμησης.

Ας δείξουμε δύο παραδείγματα!

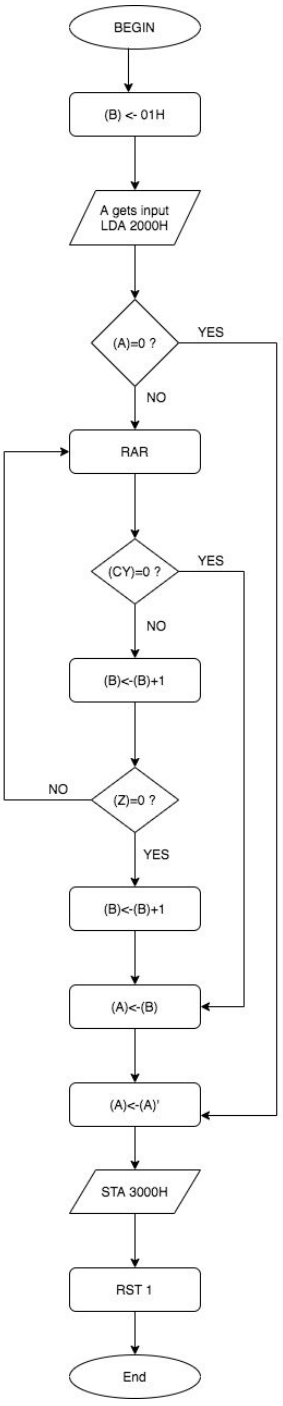
Στο πρώτο βάζουμε την δεκαεξαδική τιμή F8 ή σε δυαδικό 11111000. Η θέση του lsb είναι η 4η θέση, σε δυαδικό σύστημα το 100 και αυτό βλέπουμε να εκτυπώνεται στα leds όπου δηλαδή ανάβει το τρίτο led και όλα τα άλλα είναι κλειστά. Αυτό το παράδειγμα το επιλέξαμε για να τονίσουμε ότι τα περισσότερα σημαντικά ψηφία του lsb μας είναι αδιάφορα.



Στο δεύτερο παράδειγμα βάζουμε τιμή δεκαεξαδική τιμή 10 ή σε δυαδικό 00010000. Η θέση του lsb είναι η 5η θέση, σε δυαδικό σύστημα το 101 και αυτό βλέπουμε να εκτυπώνεται στα leds όπου

δηλαδή ανάβει το πρώτο και το τρίτο led και όλα τα άλλα είναι κλειστά.

Πιο κατανοητή η λειτουργία του προγράμματος θα γίνει αν δώσουμε το διάγραμμα ροής το οποίο παρατίθεται εδώ:



Για να πετύχουμε συνεχής λειτουργία αντικαθιστούμε την εντολή RST 1 με την εντολή JMP BEGIN. Παρατίθεται ο κώδικάς που βάλαμε στον tsik εδώ:

```
BEGIN:
       MVI B,01H
       LDA 2000H
       CPI 00H
       JZ GOTOEND
MYLOOP:
       RAR
       JC BEFOREND
       INR B
       JNZ MYLOOP
BEFOREND:
       MOV A,B
GOTOEND:
       CMA
       STA 3000H
       JMP BEGIN
       END
```

2η Άσκηση

Ζητείται να υλοποιηθεί πρόγραμμα το οποίο όταν το lsb απο τα dip switch είναι on να ανάβει διαδοχικά τα led από δεξιά προς τα αριστερά και όταν φτάσουν στο αριστερότερο led τότε να ανάβουν διαδοχικά από αριστερά στα δεξιά. Οποιαδήποτε στιγμή το lsb των dip switch κλείσει τα led να ανάβουν με κυκλική κίνηση. Οποτεδήποτε το 2ο lsb των dip switch ενεργοποιηθεί θα πρέπει να ανοίξει το πρώτο led και να παραμείνει ανοιχτό μέχρι να κλείσει το dip switch και όταν κλείσει να συνεχίσει την κίνηση του όπως το υποδεικνύει το 1ο lsb των dip switch.
Παραθέτουμε τον τελικό κώδικα, όπως αυτός εκτελέστηκε στον MicroLab Simulator 8085:

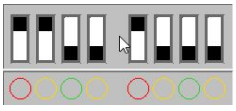
```
START:
      IN 10H
      LXI B,01FAH   ;stall=500ms
      MVI D,01H     ;to remeber state
      MVI E,01H     ;to open led 1 when halt
      MVI H,00H     ;to rem substate when led1 on
      MOV A,D
      JMP PRINT
CONTINUE:
      LDA 2000H     ;load the content of dipSwitch
      RAR           ;2 right rotations
      RAR           ;(CY) = 2 LSB
      JC HALT       ;if CY=1 GOTO HALT
      RAL           ;(CY)=1 LSB
      JC LSB_ON     ;If CY =1 GOTO LSB_ON
                    ;ELSE GOTO LSB_OF
LSB_OF:
      MOV A,D       ;rem state of led
      RRC           ;turn on the right led
      MOV D,A       ;store the new led state
      JMP PRINT     ;print the new led state
LSB_ON:
      MOV A,H       ;load substate when led1 on
      RAR           ;(CY)= 1 lsb of H
      JC GORIGHT    ;IF CY=1 GOTO GORIGHT
                    ;ELSE GOTO GOLEFT
GOLEFT:
      MVI H,00H     ;GOLEFT state is 0
      MOV A,D       ;rem state of led
      RLC           ;turn on the left led
      JC GORIGHT    ;if this isn't >=1 gorigth
      MOV D,A       ;store the new led state
      JMP PRINT     ;print the new led state
GORIGHT:
      MVI H,01H     ;GORIGHT state is 1
      MOV A,D       ;rem state of led
      RRC           ;turn on the rigth led
      JC GOLEFT     ;if this isn't <= 8 gotoleft
      MOV D,A       ;store the new led state
      JMP PRINT     ;print the new led state
HALT:
      MOV A,E       ;rem halt state =>led 1 open
      JMP PRINT     ;print halt state

PRINT:
      CMA           ;compilment of A
      STA 3000H     ;store A/print leds
      CALL DELB     ;wait 500ms
      JMP CONTINUE
END
```

3η Άσκηση

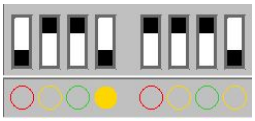
Οι αριθμοί δίνονται ως είσοδοι στα dip switch σε δυαδικη μορφη πχ ο αριθμός 2 είναι ο αριθμός : 00000010, αρά το δευτερο απο τα δεξιά switch είναι ανοιχτό και τα υπόλοιπα κλειστά. Θεωρούμε ότι οι αριθμοί είναι μέχρι και 299 και ότι ο χρήστης δεν προκειται να δώσει αριθμό μεγαλύτερο αυτού και γι αυτο δεν διαχειριζόμαστε τέτοιες καταστάσεις.
Το αποτέλεσμα να εμφανιστεί στην πόρτα εξόδου των LED ως εξής: οι μονάδες στα 4 LSB και οι δεκάδες 4 MSB.
Ο κώδικας όπως έτρεξε στον προσομοιωτή είναι ο εξής:

```
START:
      LXI B,01FAH   ;delay 500ms
      LDA 2000H     ;inputs
HERE:
      CPI 63H       ;load 99 and compare with A
      JNC EKATO     ;If A is bigger than 99 goto
                    ;EKATO
      MVI B,FFH     ;B=-1
DECA:
      INR B         ;B++
      SUI 0AH       ;(A)=(A)-10(10)
      JNC DECA      ;oso einai 8etiko epanalipsi
      ADI 0AH       ;(A)=(A)+10(10) gia na to kanw
                    ;pali 8etiko kai exw sto A tis
                    ;dekades kai sto b monades
      MOV C,A       ;C=A sta lsb tou C oi monades
      MOV A,B       ;A=B sta lsb toy A oi dekades
      RLC
      RLC
      RLC           ;oi dekades erxontai sta msb
      RLC           ;tou A
      ADD C         ;A=A+C, to A exei 10ades sta
                    ;msb kai monades sta lsb
      CMA           ;sumpilroma gt oi eksodoi
                    ;exoun arnitiki logiki
      STA 3000H     ;i dieu8nsi 3000H pairnei timi
                    ;tou A, gia ta led
      CALL DELB     ;wait 500ms
      JMP START     ;ksana
EKATO:
      SUI 64H       ;afairw 100 apo ton reg
      JMP HERE      ;elegxw ksana an einai
                    ;megaliteros tou 100
END
```

Ας τρέξουμε ενδεικτικά κάποια παραδείγματα:
π.χ. ο αριθμός 200:



π.χ. ο αριθμός 110:



4η Άσκηση

Για την τεχνικο-οικονομική σύγκριση των 3 διαφορετικών τεχνολογιών υλοποίησης θα κατασκευάσουμε έναν **πίνακα** όπου θα κατατάξουμε στις **αντίστοιχες κατηγορίες** (διακριτών στοιχείων και ολοκληρωμένων μονάδων (I.C.), FPGAs, SoC) τα **αντίστοιχα κόστη** (Αρχικό κόστος σχεδίασης, Κόστος των I.C. ανά τεμάχιο, Κόστος κατασκευής πλακέτας και συναρμολόγησης των ICs). Ύστερα θα χαραχθούν οι αντίστοιχες **καμπύλες κόστους ανά τεμάχιο** συσκευής για τις 3 τεχνολογίες.

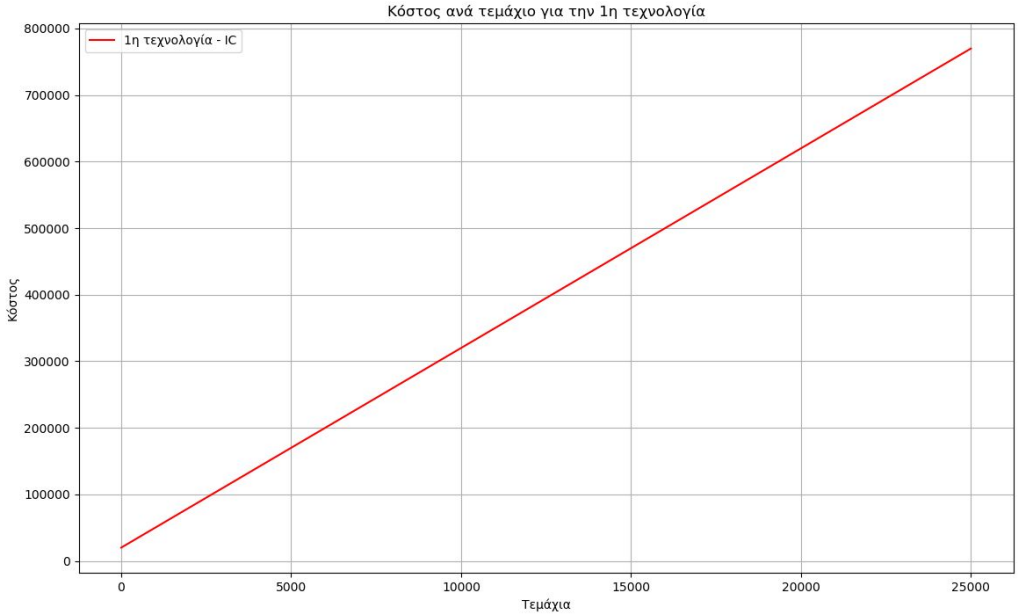
	Microprocessor Based	FPGAs	ASIC (SoC)
Αρχικό κόστος σχεδίασης	20.000€	10.000€	300.000€
Κόστος των I.C. ανά τεμάχιο	15€	60€	1€
Κόστος κατασκευής πλακέτας και συναρμολόγησης των ICs ανά τεμάχιο	15€	10€	1€

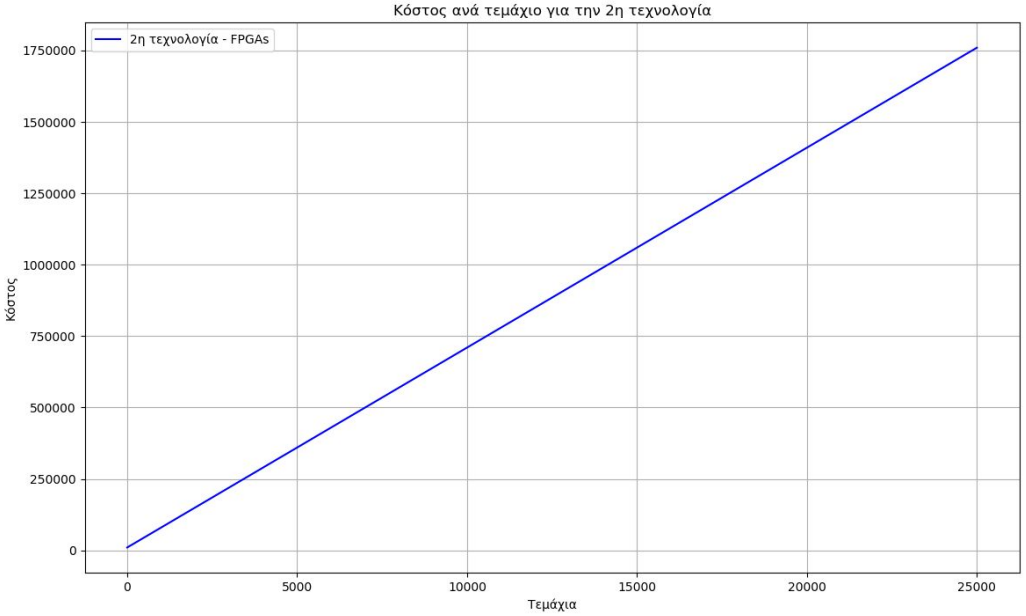
Έστω **χ το πλήθος των τεμαχίων.** Τότε σύμφωνα με τον παρακάτω τύπο - συνάρτηση κόστους ανά τεμάχιο, έχουμε για κάθε μία κατηγορία τεχνολογίας:

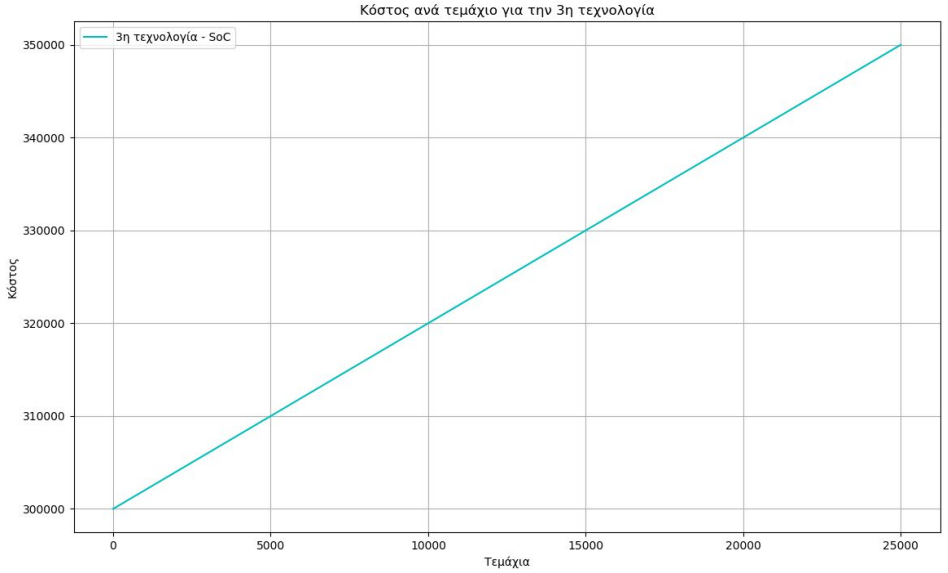
Κόστος = Αρχικό + (Κόστος-ICs + Κόστος-κατασκευής)*Πλήθος

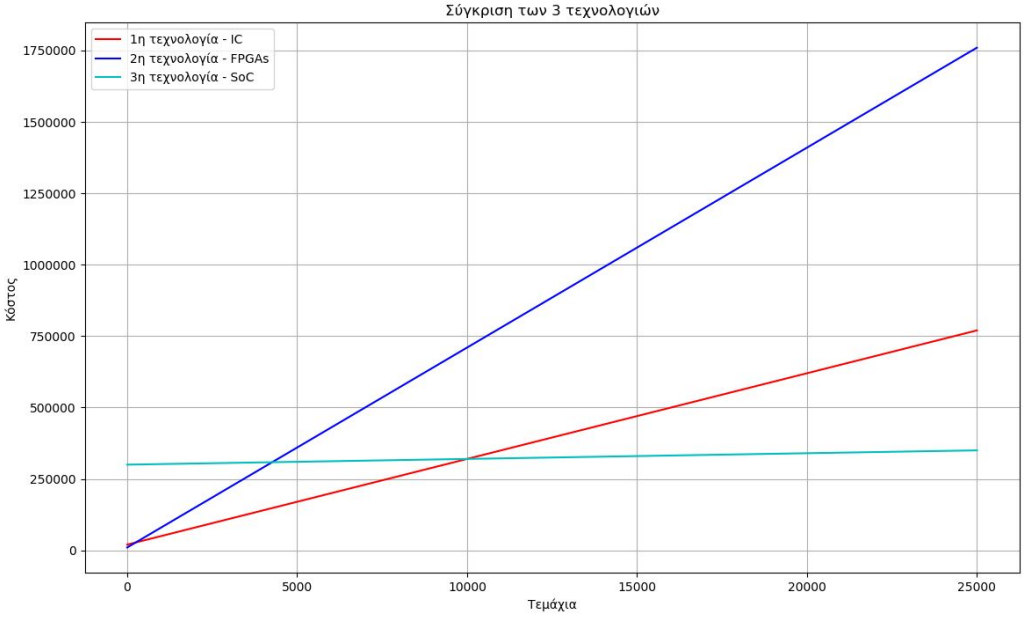
1. **Κ1 = 20.000 + (15 + 15)χ → Κ1 = 20.000 + 30χ**
2. **Κ2 = 10.000 + (60 + 10)χ → Κ2 = 10.000 + 70χ**
3. **Κ3 = 300.000 + (1 + 1)χ → Κ3 = 300.000 + 2χ**

★ **20.000 + 30χ = 10.000 + 70χ → χ = 250 τεμάχια**
★ **10.000 + 70χ = 300.000 + 2χ → χ = 4.264 τεμάχια**
★ **300.000 + 2χ = 20.000 + 30χ → χ = 10.000 τεμάχια**


Κόστος ανά τεμάχιο για την 1η τεχνολογία
1η τεχνολογία - IC
Κόστος
Τεμάχια
0
100000
200000
300000
400000
500000
600000
700000
800000
0
5000
10000
15000
20000
25000


Κόστος ανά τεμάχιο για την 2η τεχνολογία
2η τεχνολογία - FPGAs
Κόστος
Τεμάχια
0
250000
500000
750000
1000000
1250000
1500000
1750000
0
5000
10000
15000
20000
25000





Παρατηρώντας την γραφική παράσταση με τις 3 τεχνολογίες συγκεντρωμένες εξάγουμε τα εξής συμπεράσματα:

- Για την παραγωγή έως και **250 τεμαχίων** προτιμητέα είναι η χρήση της **τεχνολογίας FPGAs** και μικρού αριθμού περιφερειακών.
- Για την παραγωγή από **250 έως και 10.000 τεμαχίων** προτιμητέα είναι η χρήση της **τεχνολογίας διακριτών στοιχείων και ολοκληρωμένων μονάδων (I.C.).**
- Για την παραγωγή **10.000 τεμαχίων και άνω** προτιμητέα είναι η χρήση της τεχνολογίας σχεδίασης **ειδικού SoC με μια πολύ μικρή πλακέτα**

Η τεχνολογία 1, συμφέρει όπως αποδείξαμε προηγουμένως για πλήθος τεμαχίων μεταξύ 250 και 10.000, όπου η 2η καλύτερη είναι η τεχνολογία FPGA, δηλαδή η 2η τεχνολογία. Άρα, πρέπει στη μεταβολή του κόστους ανά τεμάχιο των IC στην τεχνολογία των FPGA και έστω y το νέο κόστος:

$20.000 + 30x > 10.000 + (y+10)x$

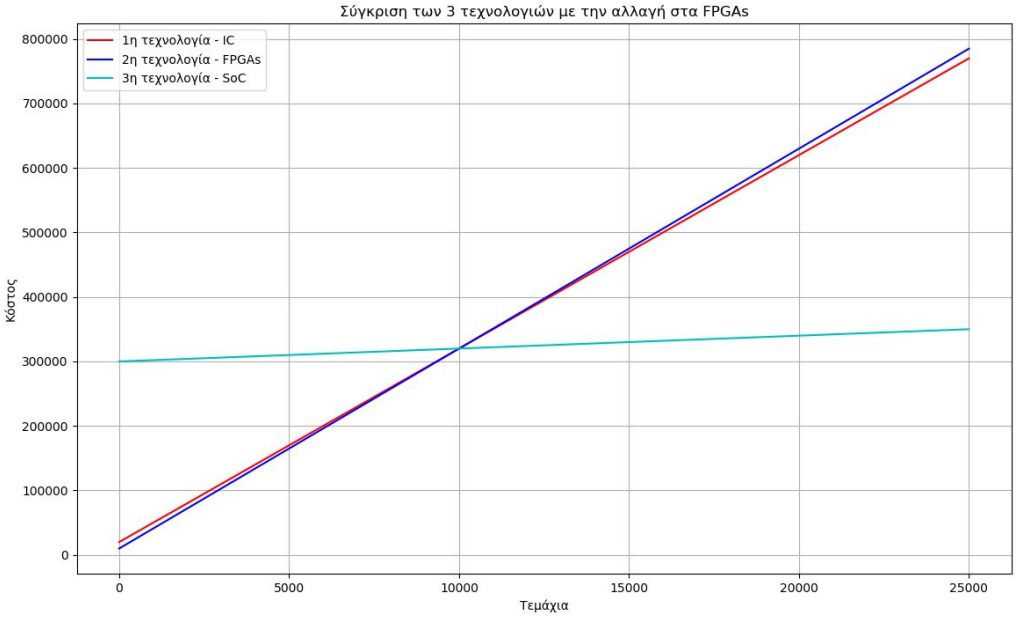
$10.000 > (y-20)x$

Επειδή η παραπάνω σχέση πρέπει να ισχύει για: $250 < \chi < 10.000$, αντικαθιστώ στην ανισοτική σχέση όπου $\chi = 10.000$ και έχω

$y > 20.$

Συνεπώς για **y = 21** για τα IC των FPGAs θα μπορούσε να εξαφανιστεί η επιλογή της 1ης τεχνολογίας.

Παρακάτω φαίνεται η αντίστοιχη γραφική παράσταση με την αλλαγή στα IC των FPGAs.



Να σημειωθεί ότι στις επόμενες ασκήσεις (5, 6, 7) η παρουσίαση των λογικών κυκλωμάτων πραγματοποιήθηκε με τη βοήθεια του προγράμματος "Logisim 2.7"

5η Άσκηση

i) Περιγραφή σε επίπεδο πυλών

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

Σε επίπεδο πυλών η περιγραφή της F1 είναι η εξής:

```
module 5i1(A, B, C, D, F)
    output F;
    input A, B, C, D;
    wire w1, w2, w3, w4, Cnot, Dnot;

    and G1(w1, C, D);
    or G2(w2, w1, B);
    and G3(w3, A, w2);
    not G4(Cnot, C);
    not G5(Dnot, D);
    and G6(w4, B, Cnot, Dnot);
    or G7(F, w3, w4);

endmodule
```

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

Με χρήση του χάρτη Karnaugh προκύπτει η βέλτιστη F2 σε άθροισμα γινομένων:

AB/CD	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	0	1	1
10	1	0	1	1

AB/CD	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	0	1	1
10	1	0	1	1

F2(A, B, C, D)=Σ (0, 2, 3, 5, 7, 8, 10, 11, 14, 15) → F2 = A'BD + D' + CD + AC

Σε επίπεδο πυλών η περιγραφή της F2 είναι η εξής:

```
module 5i2(A, B, C, D, F)
    output F;
    input A, B, C, D;
    wire w1, w2, w3, Cnot, Dnot;

    not
        G1(Anot, A);
        G2(Dnot, D);

    and
        G3(w1, Anot, B, D);
        G4(w2, C, D);
        G5(w3, A, C);

    or G6(F, w1, Dnot, w2, w3);

endmodule
```

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

Σε επίπεδο πυλών η περιγραφή της F3 είναι η εξής:

```
module 5i3(A, B, C, D, E, F);
    output F;
    input A, B, C, D, E;
    wire w1, w2, w3, w4, w5, w6;

    and G1(w1, A, B, C);
    or G2(w2, A, B);
    and G3(w3, w2, C, D);
    and G4(w4, C, D);
    or G5(w5, B, w4);
    and G6(w6, w5, E);
    or G7(F, w1, w3, w6);
endmodule
```

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

Σε επίπεδο πυλών η περιγραφή της F4 είναι η εξής:

```
module 5i4(A, B, C, D, E, F);
    output F;
    input A, B, C, D, E;
    wire w1, w2, w3, w4;

    and G1(w1, B, C);
    or G2(w2, w1, D, E);
    and G3(w3, C, D, E);
    and G4(w4, A, w2);
    or G5(F, w4, w3);

endmodule
```

ii) Περιγραφή σε Μοντελοποίηση ροής δεδομένων (data flow)

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

```
module 5ii1 (F, A, B, C, D);
    output F;
    input A, B, C, D;
    assign F = (A && ((C && D) || B)) || (B && (!C) && (!D));
endmodule
```

2. F2(A, B, C, D)=Σ (0, 2, 3, 5, 7, 8, 10, 11, 14, 15) → F2 = A'BD + D' + CD + AC όπως βρήκαμε πριν.

```
module 5ii2 (F, A, B, C, D);
    output F;
    input A, B, C, D;
    assign F = ((!A) && B && D) || (!D) || (C && D) || (A && C);
endmodule
```

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

```
module 5ii3 (F, A, B, C, D, E);
    output F;
    input A, B, C, D, E;
    assign F = (A && B &&C) || ((A || B) && C && D) || ((B || (C && D)) && E);
endmodule
```

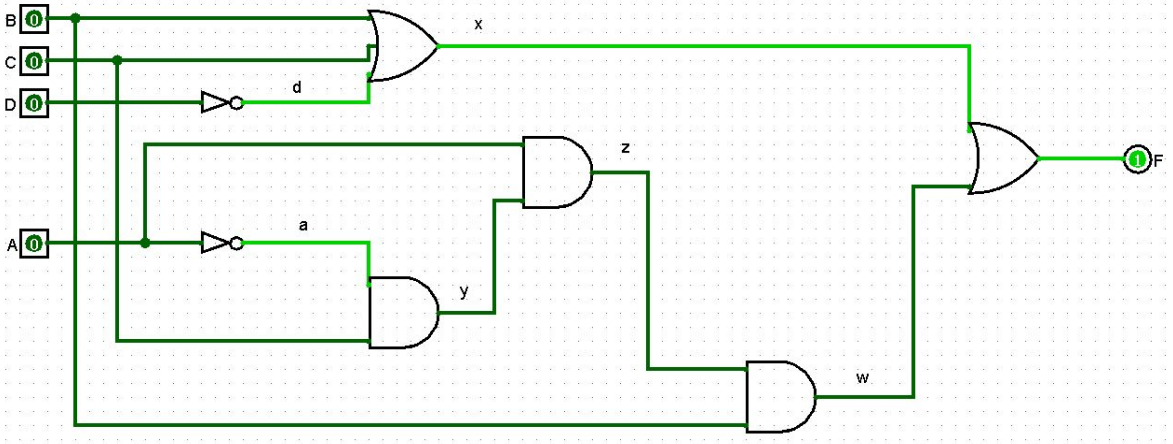
4. F4=A(BC + D + E) + CDE

```
module 5ii4 (F, A, B, C, D, E);
    output F;
    input A, B, C, D, E;
    assign F = (A && ((B && C) || D || E)) || (C && D && E);
endmodule
```

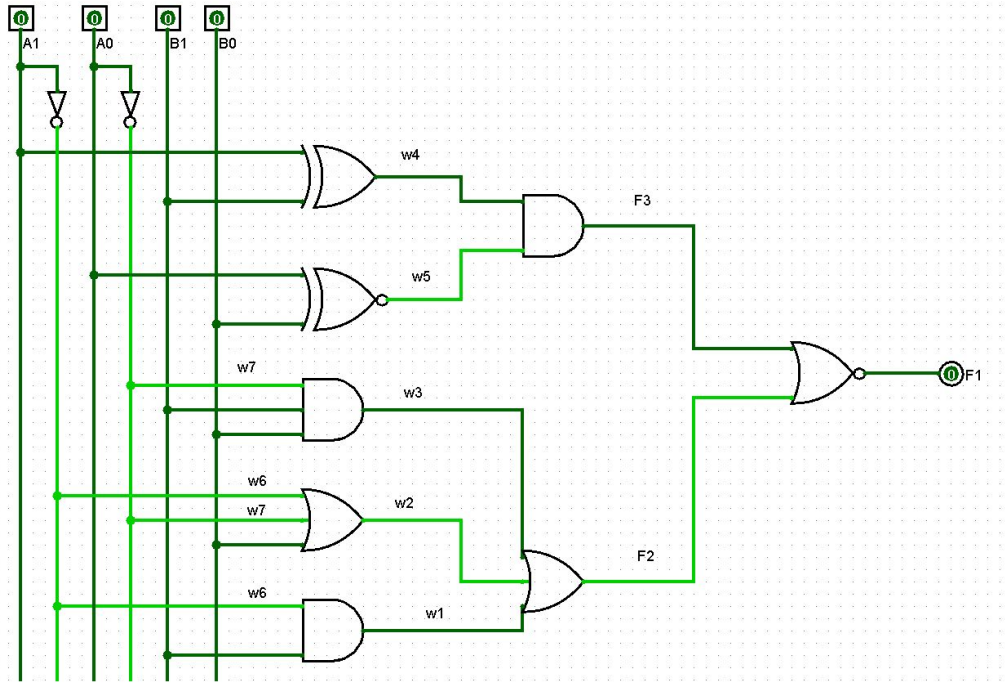
Άσκηση 6

i) Λογικό διάγραμμα

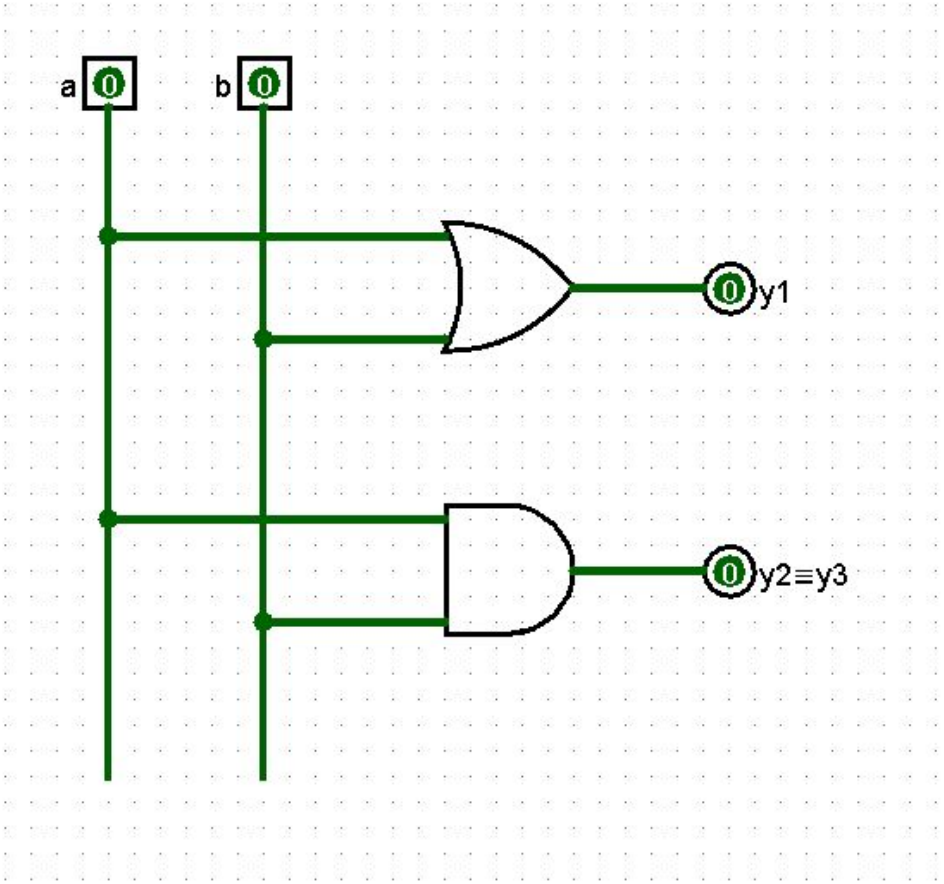
a)



b)



c)



Εδώ, στο τρίτο παράδειγμα συμπεραίνουμε ότι οι δύο εντολές "**and (y2, a, b);**" και "**assign y3 = a && b;**" είναι ισοδύναμες, αφού παράγεται το ίδιο αποτέλεσμα (δημιουργείται μία πύλη AND με εισόδους A, B και παράγει προφανώς το άθροισμά τους AB)

ii)αθροιστής-αφαιρέτης τεσσάρων bit για μη προσημασμένους δυαδικούς αριθμούς

Παρατίθεται τμήμα από το βιβλίο "DIGITAL DESIGN FOURTH EDITION M. MORRIS MANO California State University, Los Angeles MICHAEL D. CILETTI University of Colorado, Colorado Springs", το σχήμα του αθροιστή-αφαιρέτη τεσσάρων bit για λόγους ευκολίας.

Η ιεραρχική περιγραφή HDL σε επίπεδο πύλης είναι η ακόλουθη. Να σημειωθεί ότι δεν προστέθηκε στην περιγραφή η αναγνώριση υπερχείλισης με έξοδο V (πύλη xor με εισόδους τα κρατούμενα C3, C3).

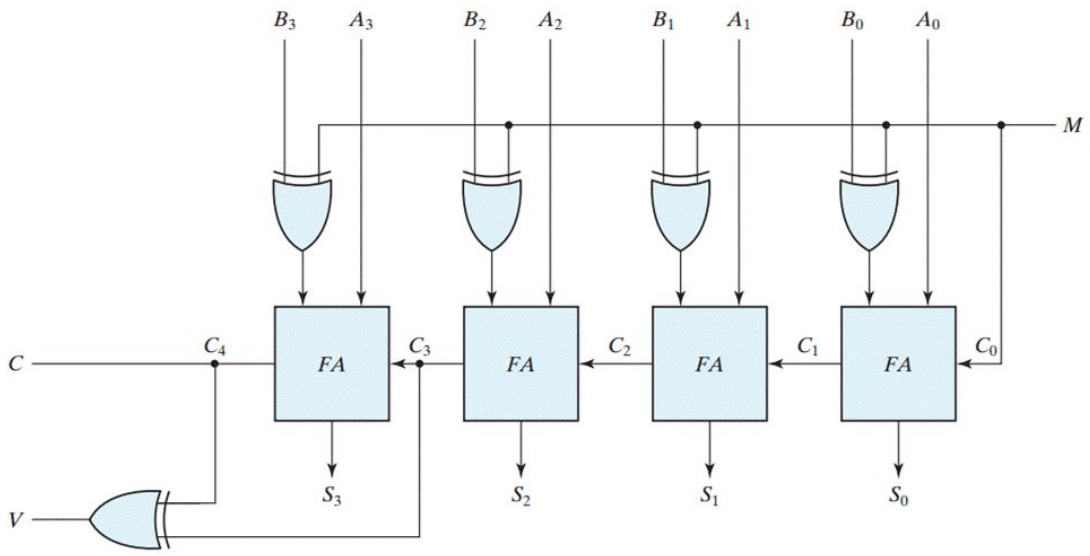


FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

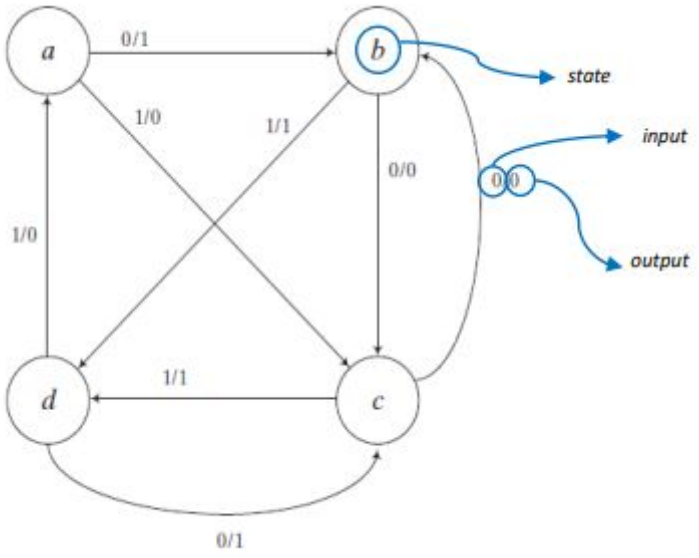
```
module half_adder (output S, C, input x, y);
    xor (S, x, y);
    and (C, x, y);
endmodule


module full_adder (output S, C, input x, y, z);
    wire S1, C1, C2;
    half_adder HA1 (S1, C1, x, y); // Instantiate HAs
    half_adder HA2 (S, C2, S1, z);
    or G1 (C, C2, C1);
endmodule


// Description of four-bit adder - substractor
module 4_bit_adder_subtractor ( output [3: 0] Sum,
    output C4, input [3: 0] A, B, input M);
    wire C1, C2, C3; // Intermediate carries
    wire [3: 0] B_xor_M; // Intermediate carries
    xor (B_xor_M[0], B[0], M);
    xor (B_xor_M[1], B[1], M);
    xor (B_xor_M[2], B[2], M);
    xor (B_xor_M[3], B[3], M);
// Instantiate chain of full adders
full_adder FA0 (Sum[0], C1, A[0], B_xor_M[0], M),
           FA1 (Sum[1], C2, A[1], B_xor_M[1], C1),
           FA2 (Sum[2], C3, A[2], B_xor_M[2], C2),
           FA3 (Sum[3], C4, A[3], B_xor_M[3], C3);
endmodule
```

iii)Περιγραφή ροής δεδομένων HDL ενός αθροιστή-αφαιρέτη τεσσάρων bits μη προσημασμένων αριθμών με χρήση του τελεστή υπό συνθήκη (? :)

```
// Dataflow description of four-bit adder-subtractor
module binary_adder_subtractor (Sum, Cout, A, B, sel); //sel = selector
    output [3: 0] Sum;
    output Cout;
    input [3: 0] A, B;
    input sel;
    assign {Cout, Sum} = sel ? A-B : A+B;  //MUX 2:1 with 2 inputs (A-B, A+B) select input: selector
endmodule
```

7η άσκηση

i) **Mealy FSM**



Εφαρμόζουμε την κωδικοποίηση των καταστάσεων ως εξής:

a -> 00
b -> 01
c -> 10
d -> 11

Παρούσα κατάσταση F-F		Είσοδος x	Επόμενη Κατάσταση F-F		Έξοδος y
A_n	B_n		A_{n+1}	B_{n+1}	
0	0	0	0	1	1
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	0	0	0

[1]

Η περιγραφή συμπεριφοράς σε Verilog HDL, είναι ο παρακάτω:

```
// Mealy model FSM
module Mealy_Zero_Detector (
    output reg y,
    input x, clock, reset
);

    reg[1:0] state, next_state;
    parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;

    always @ ( posedge clock, negedge reset) // state transition
        if (reset == 0) state <= a; // Αρχικοποίηση στην a
        else state <= next_state;

    always @ (state, x)  // Form the next state
        case(state)
            a: begin y = ~x; if (~x) next_state = b; else next_state = c; end
            b: begin y = x; if (~x) next_state = c; else next_state = d; end
            c: begin y = x; if (~x) next_state = b; else next_state = d; end
            d: begin y = ~x; if (~x) next_state = c; else next_state = a; end
        endcase

endmodule
```

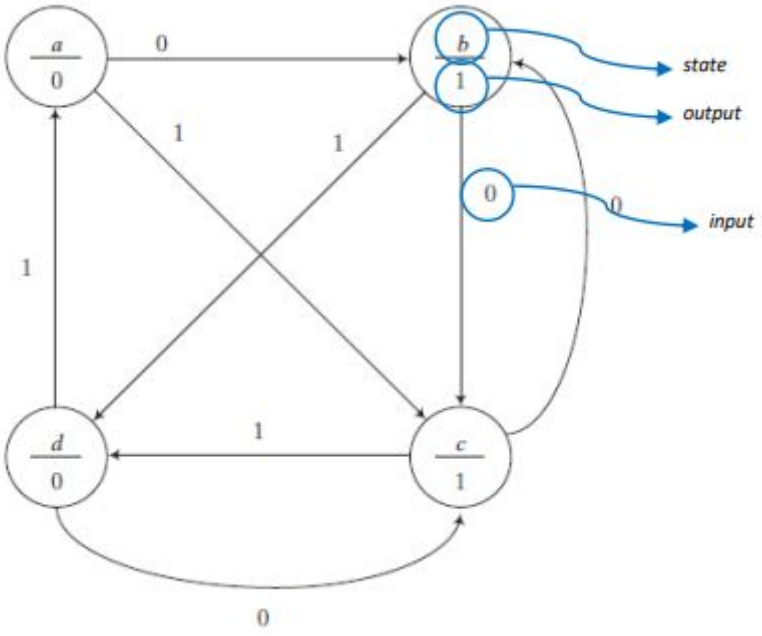
[1] Πίνακας μεταβάσεων (και καταστάσεων) του κυκλώματος του ερωτήματος 7i

Παρατηρούμε ότι

1. στην κατάσταση a: **y=x'**
2. στην κατάσταση b: **y=x**
3. στην κατάσταση c: **y=x**
4. στην κατάσταση d: **y=x'**

Να σημειωθεί ότι συνδυάσαμε τις μεταβάσεις καταστάσεων και την έξοδο σε ένα block **always.**

ii) **Moore FSM**



Εφαρμόζουμε την κωδικοποίηση των καταστάσεων ως εξής:

a -> 00
b -> 01
c -> 10
d -> 11

Παρούσα κατάσταση F-F		**Είσοδος x**	**Επόμενη Κατάσταση F-F**		**Έξοδος y**
A_n	B_n		A_{n+1}	B_{n+1}	
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	0	0	0

[2]

Παρατηρούμε ότι με την αντίστοιχη είσοδο x η επόμενη κατάσταση στην οποία μεταβαίνουμε είναι ίδια όπως στο προηγούμενο παράδειγμα με την μοναδική διαφορά ότι η έξοδος y είναι διαφορετική, είναι σταθερή ανεξάρτητα της εισόδου, διαφοροποιείται ανάλογα με την κατάσταση στην οποία βρίσκεται. Συνεπώς και στο κομμάτι κώδικα της verilog η μόνη αλλαγή θα είναι ο ορισμός της εξόδου και προφανώς **η ιδιότητα του Moore machine να εξαρτάται η έξοδος μόνο από την παρούσα κατάσταση και όχι και από την είσοδο.**

[2] Πίνακας μεταβάσεων (και καταστάσεων) του κυκλώματος του ερωτήματος 7i

```
// Moore model FSM
module Moore_Zero_Detector (
    output reg y,
    input x, clock, reset
);

    reg[1:0] state;
    parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;

    always @ ( posedge clock, negedge reset) // state transition
        if (reset == 0) state <= a; // Αρχικοποίηση στην a
        else case(state);
            a: begin y = 0; if (~x) state = b; else state = c; end
            b: begin y = 1; if (~x) state = c; else state = d; end
            c: begin y = 1; if (~x) state = b; else state = d; end
            d: begin y = 0; if (~x) state = c; else state = a; end
        endcase

endmodule
```

iii) **Up-down counter των 4 bit που διαθέτει 2 εισόδους (Up/Down και Clear), με χρήση D-flip flop και εισόδους σύγχρονες (με το ρολόι).**

Να σημειωθεί ότι το clear το θεωρήσαμε αρνητικά ακμοπυροδότητο. Όταν το clear μηδενίζεται, οι είσοδοι των flip flops είναι D = 0, άρα οι επόμενες καταστάσεις είναι μηδενικές, άρα η μέτρηση ξεκινάει από την αρχή πάλι (είτε πάνω είτε κάτω) από το "0000".

```
// up-down counter των 4 bit
module Up_Down_Counter_4bit (output reg [3: 0] A, input CLK, Up, Down, clear);
    always @ (posedge CLK, negedge clear)
        if (clear ==0) A <= 4'b0000;
        else if (Up) A <= A + 4'b0001;
        else if (Down) A <= A - 4'b0001;
endmodule
```