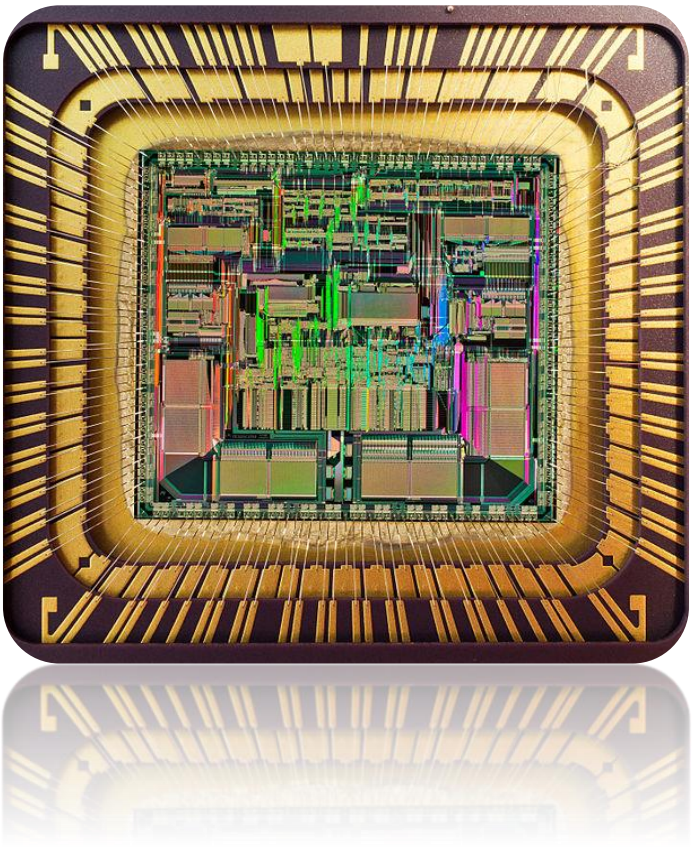
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ

1η Ομάδα Ασκήσεων

Βαβουλιώτης Γεώργιος
ΑΜ : 03112083
6ο Εξάμηνο

Γιαννόπουλος Αναστάσιος
ΑΜ : 03112176
6ο Εξάμηνο



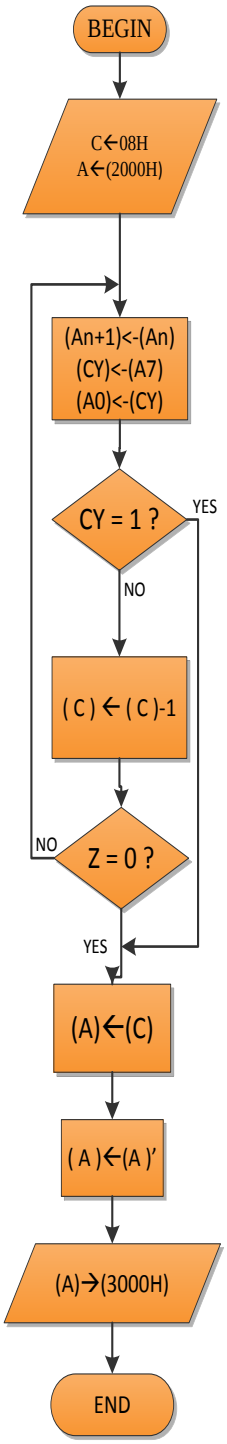
ΑΣΚΗΣΗ 1

Το πρόγραμμα το οποίο δόθηκε σε γλώσσα μηχανής μεταφράστηκε με τη βοήθεια του Παραρτήματος 2 σε συμβολική γλώσσα (assembly), όπως φαίνεται παρακάτω :

```
0800 0E   MVI C,08H
0801 08
0802 3A   LDA 2000H
0803 00
0804 20

TAG2:
0805 17   RAL
0806 DA   JC TAG1
0807 0D
0808 08
0809 0D   DCR C
080A C2   JNZ TAG2
080B 05
080C 08

TAG1:
080D 79   MOV A,C
080E 2F   CMA
080F 32   STA 3000H
0810 00
0811 30
0812 CF   RST 1
```

Το παραπάνω πρόγραμμα εκτελεί την εξής λειτουργία : Ανάλογα με την είσοδο που θα του δώσω από τους διακόπτες (dip switches) μου δίνει μέσω των Led την θέση του πιο σημαντικού άσσου που υπάρχει στην είσοδο (διακόπτες). Η θέση αυτή απεικονίζεται στο δυαδικό σύστημα μέσω των Led, άρα για να βρω την θέση του πιο σημαντικού άσσου κάνω την μετατροπή από δυαδικό σε δεκαδικό σύστημα. Για παράδειγμα αν έχω σαν είσοδο το 0001 1111 στα Led θα εμφανίζεται ο αριθμός 0000 0101 (5 στο δεκαδικό σύστημα) , ο οποίος προφανώς δηλώνει τη θέση του πλέον σημαντικού άσσου της εισόδου.

Το διάγραμμα ροής (το οποίο σχεδιάστηκε με τη βοήθεια του προγράμματος Visio 2010) του παραπάνω προγράμματος φαίνεται παρακάτω :



Παρατήρηση : Στο παραπάνω διάγραμμα ροής ο συμβολισμός (A)' δηλώνει το συμπλήρωμα ως προς 1 του περιεχομένου του καταχωρητή Α.

Για να έχει το παραπάνω πρόγραμμα μια συνεχόμενη μορφή λειτουργίας θα έπρεπε να προστεθεί μια εντολή *JMP START*, όπου *START* είναι μια ετικέτα στην αρχή του προγράμματος. Συγκεκριμένα το πρόγραμμα θα έπαιρνε την εξής μορφή:

```
START:
0800 0E   MVI C,08H
0801 08
0802 3A   LDA 2000H
0803 00
0804 20

TAG2:
0805 17   RAL
0806 DA   JC TAG1
0807 0D
0808 08
0809 0D   DCR C
080A C2   JNZ TAG2
080B 05
080C 08

TAG1:
080D 79   MOV A,C
080E 2F   CMA
080F 32   STA 3000H
0810 00
0811 30
0812 C3   JMP START
0813 00
0814 08
```

ΑΣΚΗΣΗ 2

Ο κώδικας σε assembly της άσκηση αυτής φαίνεται παρακάτω :

```
MVI C,F4H     ; insert delay to see led's movement
MVI D,01H     ; register D shows which led have to open
STC           ; make curry flag 1
CMC           ; curry flag=0
MOV A,D
CMA
STA 3000H     ; anavw to led 1 me xrhsh antistrofhs logikhs


BEGIN:
LDA 2000H     ; fortwnw ston A tis 8eseis twn dip switches
RAR           ; Curry= LSB
JC  STABLE    ; an LSB=1 paramenw sthn katastash pou eimai
RAL
RAL           ; Curry_Flag= MSB
JC  MTOL      ; an MSB=1 tote paw apo 8-->1
JMP LTOM      ; alliws paw apo 1-->8


STABLE:
MOV A,D       ; (A)<--(D) gia na parameinw sto idio LED
JMP LEDOPEN


LEDOPEN:
MOV D,A       ; (A)--> (D)
CMA
STA 3000H     ; anavw to antistoixo LED
CALL DELB     ; eisagw ka8usterhsh
JMP BEGIN

LTOM:
MOV A,D       ; (A)<--(D)
RAL           ; paw sto epomeno LED(sto epomeno apo aristera)
JC GOTOLED1   ; an eimai sto LED 8 paw sto 1
JMP LEDOPEN

GOTOLED1:
SUB A         ; (A)=0
ADI 01H       ; (A)=1
JMP LEDOPEN


MTOL:
MOV A,D       ; (A)<--(D)
RRC           ; paw sto epomeno LED(sto epomeno apo deksia)
JMP LEDOPEN

END
```

<u>Παρατηρήσεις σχετικά με τον κώδικα και τον τρόπο υλοποίησης</u> :

- Αρχικά στο πρόγραμμα ανάβω αυθαίρετα το πρώτο Led αφού θεωρώ ότι από το Led αυτό ξεκινάει η κίνηση που επιθυμώ. Στη συνέχεια ανάλογα με τις εισόδους των διακοπτών επιλέγω την φορά που θέλω να έχει η διαδρομή των Led ή αν επιθυμώ να σταματήσει η κίνηση τους. Ο πίνακας λειτουργίας του παραπάνω προγράμματος είναι ο παρακάτω :

INPUT		OUTPUT
MSB	LSB	LEDS
0	0	LEFT
1	0	RIGHT
X	1	STOP

- Εισάγω καθυστέρηση στο πρόγραμμα με χρήση της συνάρτησης DELB η οποία εισάγει καθυστέρηση ίση με (1ms)*(BC) .
- Στον κώδικα έχω κάνει **bold** τις ετικέτες του προγράμματος στις οποίες γίνονται τα διάφορα jumps για να είναι εμφανή τα διάφορα μέρη του κώδικα.
- Ένας δεύτερος τρόπος για να υλοποιηθεί η ετικέτα LTOM είναι ο παρακάτω :

```
LTOM: MOV A,D
      RLC
      JMP LEDOPEN
```

ΑΣΚΗΣΗ 3

Λόγο ελλιπούς κατανόησης της εκφώνησης της άσκησης έχουν υλοποιηθεί 2 διαφορετικά προγράμματα σε assembly, των οποίων η λειτουργία φαίνεται παρακάτω :

1η εκδοχή : Εδώ αν δοθεί αριθμός μεγαλύτερος του 99 τότε δεν με ενδιαφέρει ποιος είναι ο αριθμός αυτός και απλά αναβοσβήνω με ικανοποιητικό ρυθμό τα MSB και LSB των Led. Αν στην συνέχεια δοθεί ένας αριθμός μικρότερος του 99 τότε προφανώς απεικονίζονται οι δεκάδες και οι μονάδες του στα Led όπως ζητείται. Ο κώδικας φαίνεται παρακάτω :

```
START:
LDA 2000H           ;pairnw eisodo apo tous diakoptes
CPI 63H
JNC MORETHAN99      ;tsekarw an einai megaluteros tou 99
MVI D,FFH           ;(D)=-1 se sumplhrwma ws pros 2

LESSTHAN99:
INR D               ;(D)<--(D)+1
SUI 0AH             ;afairw 10 ka8e fora
JNC LESSTHAN99      ;oso einai 8etikos afairw ksana to 10
ADI 0AH             ;dior8wnw to arnhtiko upoloipo
MOV H,A             ;o H exei pleon tis monades
SUB A               ;mhdenizw ton A
ADD D               ;(A)=(D)
RAL                ;kanw 4 RAL giati 8elw tis dekades sta MSB twn LED
RAL
RAL
RAL
ADD H   ;pros8etw kai tis monades gia na emfanistoun sta LSB twn LED
CMA
STA 3000H
JMP START           ;exw sunexomenh roh programmatos

MORETHAN99:
MVI E,F0H           ;(E)=11110000
MOV A,E             ;(A)<--(E)
CMA
STA 3000H
LXI B,0F47H        ;fortwnw ston BC enan ari8mo gia swsth ka8usterhsh
CALL DELB    ;h sunarthsh DELB eisagei ka8usterhsh ish me (1ms)*(BC))
MVI E,0FH           ;(E)=00001111
MOV A,E             ;(A)<--(E)
CMA
STA 3000H
LXI B,0A64H        ;fortwnw ston BC enan ari8mo gia swsth ka8usterhsh
CALL DELB           ;h DELB eisagei ka8usterhsh ish me (1ms)*(BC))
JMP START           ;exw sunexomenh roh programmatos

END
```

2η εκδοχή : Εδώ αν δοθεί αριθμός μεγαλύτερος του 99 τότε βρίσκω τις δεκάδες(στην ουσία οι δεκάδες για να μπορούν να απεικονιστούν στα Led θα πρέπει να είναι το πολύ 15 αφού μέχρι των αριθμό αυτό μπορούν να απεικονίσουν τα Led) και τις μονάδες του, τις εμφανίζω στα Led και μετά τις κάνω να αναβοσβήνουν με ικανοποιητικό ρυθμό μέχρι να δοθεί ένας άλλος αριθμός από τους διακόπτες. Ο κώδικας φαίνεται παρακάτω :

```
START:
LDA 2000H       ;pairnw eisodo apo tous diakoptes
MVI D,FFH       ;(D)=-1 se sumplhrwma ws pros 2
MOV L,A

FINDFORALL:
INR D           ;(D)<--(D)+1
SUI 0AH         ;afairw 10 ka8e fora
JNC FINDFORALL ;oso einai 8etikos afairw ksana to 10
ADI 0AH         ;dior8wnw to arnhtiko upoloipo
MOV H,A         ;o H exei pleon tis monades
SUB A           ;mhdenizw ton A
ADD D           ;(A)=(D), o D exei tis dekades
RAL             ;kanw 4 RAL giati 8elw tis dekades sta MSB twn LED
RAL
RAL
RAL
ADD H          ;pros8etw kai tis monades gia na fanoun sta LSB twn LED
MOV H,A         ;krataw to periexomeno toy A(dekades+monades)
CMA
STA 3000H
MOV A,L  ;o A exei pleon thn eisodo twn diakoptwn gia na kanw sugkrih
CPI 63H
JNC MORETHAN99  ;tsekarw an einai megaluteros tou 99
JMP START       ;exw sunexomenh roh programmatos

MORETHAN99:
MOV A,H         ;epanaferw ston A dekades kai monades
MVI E,F0H       ;(E)=11110000
ANA E           ;krataw mono ta msb toy A dhladh tis dekades
CMA
STA 3000H
LXI B,0F47H     ;fortwnw ston BC enan ari8mo gia swsth ka8usterhsh
CALL DELB
MVI E,0FH       ;(E)=00001111
MOV A,H         ;epanafora periexomenou A
ANA E           ;krataw mono ta lsb toy A dldadh tis monades
CMA
STA 3000H
LXI B,0A65H     ;fortwnw ston BC enan ari8mo gia swsth ka8usterhsh
CALL DELB       ;h DELB eisagei ka8usterhsh ish me (1ms)*(BC)
LDA 2000H       ;diabazw ksana thn eisodo apo tous diakoptes
CMP H           ;tsekarw an h twrinh me thn palia eisodo einai idies
JZ MORETHAN99   ;an einai idies tote ta led menoun idia
JMP START       ;alliws exei dw8ei kainourgios ari8mos

END
```

Παρατηρήσεις σχετικά με τον κώδικα και τον τρόπο υλοποίησης και των 2 εκδοχών :

- **Προσοχή** : Στην δεύτερη εκδοχή αν στην είσοδο μπει αριθμός μεγαλύτερος του 159 το πρόγραμμα βγάζει λάθος δεκάδες αφού έχω στη διάθεση μου για την απεικόνιση των δεκάδων 4 Led, άρα μπορώ να απεικονίσω μέχρι το 15 (από το 0 έως το 15). Αν δοθεί αριθμός μεγαλύτερος του 159 έχω λάθος αποτέλεσμα όπως είναι και λογικό.
- Όπως και στην 1η άσκηση εισάγω καθυστέρηση στο πρόγραμμα με χρήση της συνάρτησης DELB η οποία εισάγει καθυστέρηση ίση με (1ms)*(BC) .
- Λόγω της συνεχόμενης ροής του προγράμματος, για να απεικονίσουμε ένα νέο αριθμό αρκεί να αλλάξουμε τις θέσεις των διακοπτών και θα δούμε απ' ευθείας το αποτέλεσμα στα Led.
- Στον κώδικα έχω κάνει **bold** τις ετικέτες του προγράμματος στις οποίες γίνονται τα διάφορα jumps για να είναι εμφανή τα διάφορα μέρη του κώδικα.

ΑΣΚΗΣΗ 4

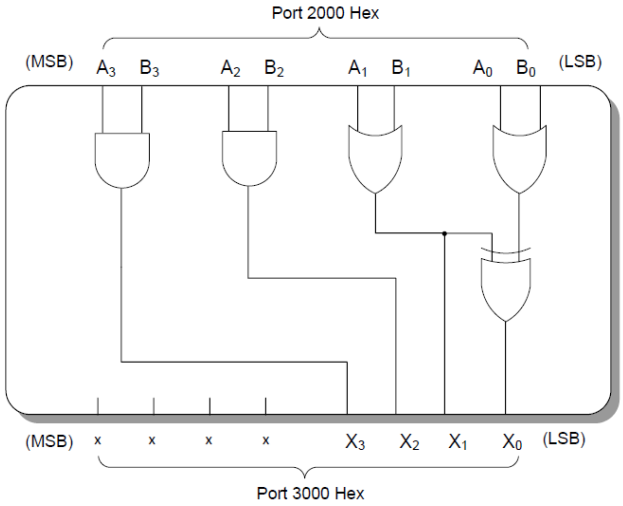
Ο κώδικας σε assembly της άσκηση αυτής φαίνεται παρακάτω :

```
BEGIN:
LDA 2000H              ;READ INPUT
MVI B,00H              ;ARXIKOPOIHSH B
STC                    ;CY=1
CMC                    ;TELIKA CY=0
MVI C,00H              ;ARXIKOPOIHSH C
MVI D,00H              ;ARXIKOPOIHSH D
RAR                    ;CY = LSB
JC COMP1               ;IF CY = 1 JUMP COMP1
RAR                    ;NEXT INPUT-BIT
JC FIRST_OR_RESULT_1   ;IF CY = 1 THEN RESULT = 1
JNC FIRST_OR_RESULT_0 ;ELSE RESULT = 0

COMP1:
RAR                    ; NEXT INPUT - BIT

FIRST_OR_RESULT_1:
INR B                  ;B=1

FIRST_OR_RESULT_0:
MOV D,B                ;D=RESULT
MVI B,00H              ;CLEAR B
RAR                    ;NEXT INPUT-BIT
JC COMP2               ;IF CY=1 JUMP COMP2
RAR                    ;NEXT INPU -BIT
JC SECOND_OR_RESULT_1  ;IF CY=1 THEN RESULT=1
JNC SECOND_OR_RESULT_0 ;ELSE RESULT=0

COMP2:
RAR                    ;NEXT INPUT-BIT

SECOND_OR_RESULT_1:
INR B                  ;B=1

SECOND_OR_RESULT_0:
MOV C,B                ;C=RESULT
MVI B,00H              ;CLEAR B
MOV E,A                ;SAVE A
MOV A,C                ;A=RESULT
CMP D                  ;C=D?
STC                    ;CY=1
CMC                    ;TELIKA CY=0
RAL                    ;SHIFT LEFT GIA NA EUTHUGRAMISW THN EKSODO
MOV C,A                ;C=RESULT TOU DEYTEROY LSB
JZ CONTINUE            ;IF RESULTS ARE SAME THEN XOR_RESULT=0
INR C             ;ELSE XOR RESULT = 1 (O EXEI TO RESULT TWN 2 LSB)

CONTINUE:
MOV A,E                ;EPANAFORA TOU A
RAR                    ;NEXT INPUT-BIT
JNC COMP3              ;IF CY=0 JUMP COMP3
RAR                    ;NEXT BIT
JNC FIRST_AND_RESULT_0 ;IF CY=0 THEN AND_RESULT=0
INR B                   ;ELSE RESULT=1
```

```
JC FIRST_AND_RESULT_0  ;JUMP  KSEPERNONTAS COMP3

COMP3:
RAR                    ;NEXT BIT

FIRST_AND_RESULT_0:
MOV E,A                ;SAVE A
MOV A,B                ;A=AND_RESULT
STC                    ;CY=1
CMC                    ;TELIKA CY=0
RAL                ;2 SHIFT LEFT GIA NA PAEI TO AND_RESULT STH THESI 3
RAL
ADD C                  ;TWRA O A PERIEXEI [0 0 0 0 0 X2 X1 X0]
MOV C,A                ;C=RESULT EWS TWRA
MOV A,E                ;EPANAFORA TOU A
MVI B,00H              ;CLEAR B
RAR                    ;NEXT INPUT-BIT
JNC COMP4              ;IF CY=0 JUMP COMP4
RAR                    ;NEXT BIT
JNC SECOND_AND_RESULT_0 ;IF CY=0 THEN AND_RESULT=0
INR B                   ;ELSE AND_RESULT=1
JC SECOND_AND_RESULT_0  ;JUMP KSEPERNONTAS TO COMP4

COMP4:
RAR                    ;NEXT BIT

SECOND_AND_RESULT_0:
MOV E,A                ;SAVE A
MOV A,B                ;A=AND_RESULT
STC                    ;CY=1
CMC                    ;TELIKA CY=0
RAL                ;3 SHIFT LEFT GIA NA PAEI TO AND_RESULT STH THESI 4
RAL
RAL
ADD C                  ;TWRA O A PERIEXEI [0 0 0 0 X3 X2 X1 X0]
MOV C,A                ;C=TELIKO RESULT

FINISH:
MOV A,C                ;TO TELIKO RESULT PAEI STON A
CMA                    ;TA LED EINAI ANTISTROFHS LOGIKIS
STA 3000H              ;OUTPUT
RST 1

END
```

<u>Παρατηρήσεις σχετικά με τον κώδικα και τον τρόπο υλοποίησης</u> :

- Το παραπάνω πρόγραμμα σε assembly εξομοιώνει την λειτουργία του I.C. που φαίνεται παρακάτω :



- Η ιδέα πάνω στην οποία στηρίζεται ο παραπάνω κώδικας είναι η υλοποίηση κάθε πύλης ξεχωριστά από δεξιά προς τα αριστερά όπως αυτό είναι εμφανές και από τα σχόλια που υπάρχουν στο πρόγραμμα. Θα πρέπει να τονιστεί πως αν στην είσοδο μιας πύλης OR το πρώτο bit εισόδου είναι 1 δεν χρειάζεται αν τσεκάρουμε και το επόμενο αφού το αποτέλεσμα είναι σίγουρα 1 (δυικά αν στην είσοδο μιας πύλης AND το πρώτο bit είναι 0 τότε σίγουρα η έξοδος είναι 0).
- Στον κώδικα έχω κάνει **bold** τις ετικέτες του προγράμματος στις οποίες γίνονται τα διάφορα jumps για να είναι εμφανή τα διάφορα μέρη του κώδικα.

ΑΣΚΗΣΗ 5

Η συνάρτηση του κόστους για κάθε τεχνολογία είναι:

Διακριτά Στοιχεία: $K_{\Delta\Sigma}(x) = 20000 + (10+10)x = 20000 + 20x$

FPGAs: $K_{FPGAs}(x) = 10000 + (30+10)x = 10000 + 40x$

SoC-1: $K_{SoC-1}(x) = M + (2+2)x = M + 4x$

SoC-2: $K_{SoC-2}(x) = 200000 + (1+1)x = 200000 + 2x$

Οι συναρτήσεις κόστους ανά τεμάχιο είναι :

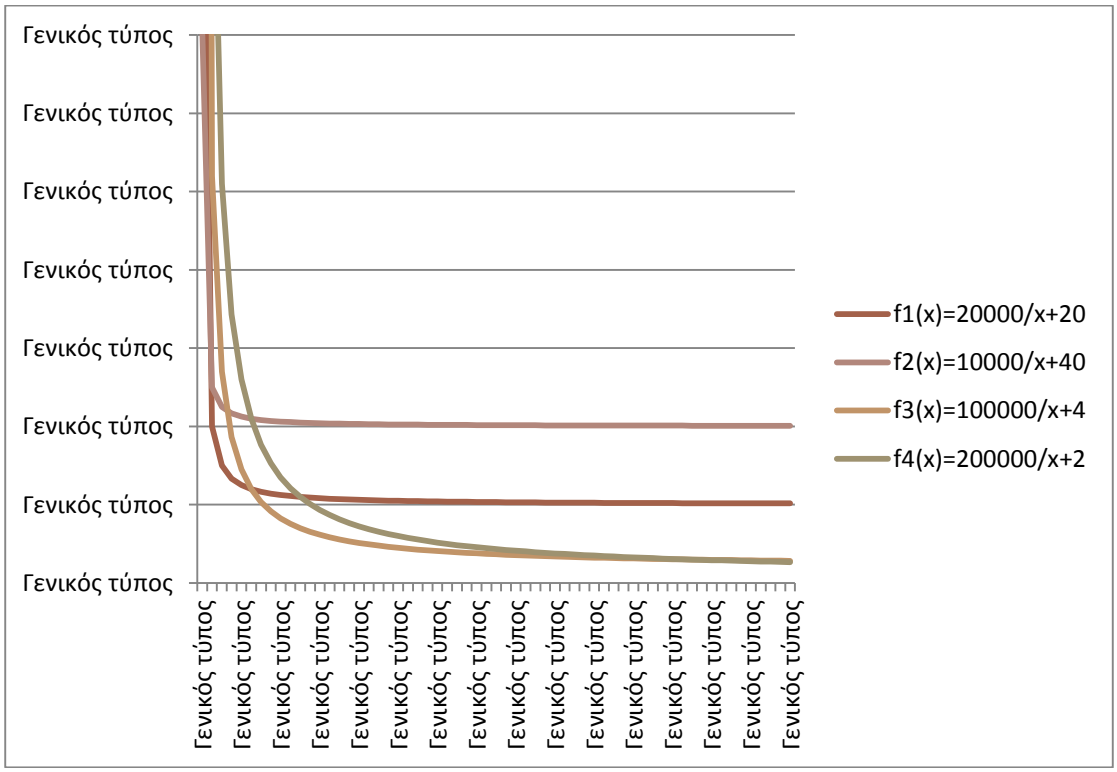
Διακριτά Στοιχεία: $K_{\Delta\Sigma}(x) = \frac{20000}{x} + (10+10) = \frac{20000}{x} + 20$

FPGAs: $K_{FPGAs}(x) = \frac{10000}{x} + (30+10) = \frac{10000}{x} + 40$

SoC-1: $K_{\Delta\Sigma}(x) = \frac{100000}{x} + (2+2) = \frac{100000}{x} + 4$

SoC-2: $K_{\Delta\Sigma}(x) = \frac{200000}{x} + (1+1) = \frac{200000}{x} + 2$

Στο παρακάτω διάγραμμα συνυπάρχουν οι γραφικές παραστάσεις των συναρτήσεων κόστους ανά τεμάχιο:

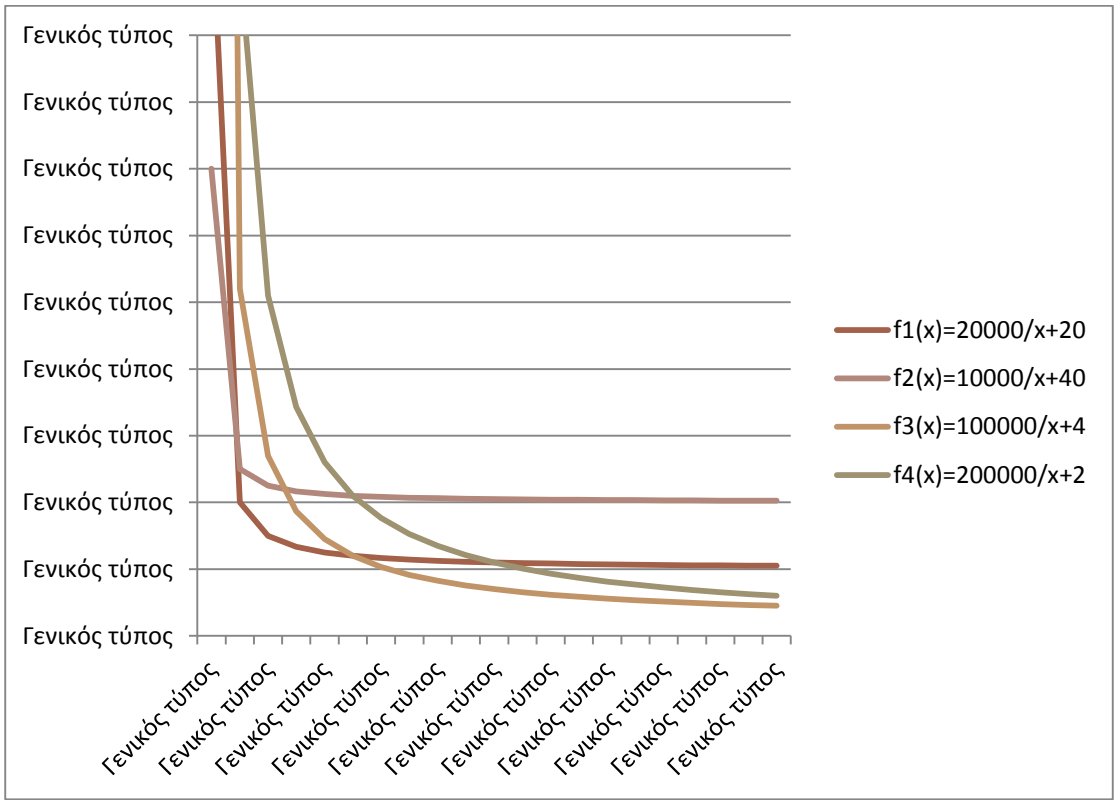


Αντιστοιχία χρωμάτων:

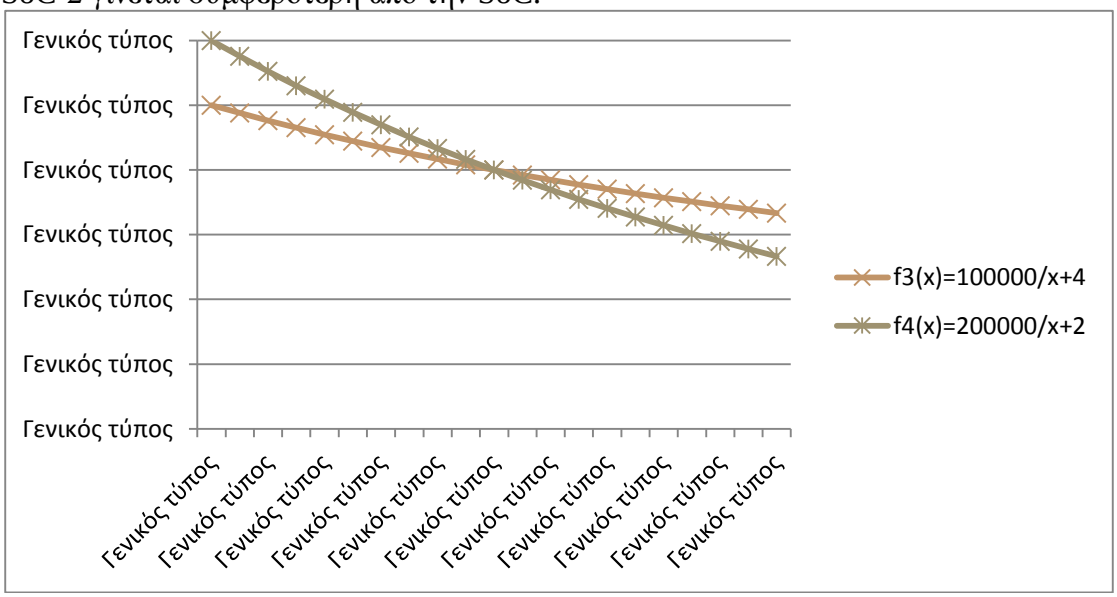
- Μπορντώ: Διακριτά στοιχεία
- Πράσινο: FPGAs
- Μωβ: SoC-1
- Μπλε: SoC-2

Ο οριζόντιος άξονας εκφράζει τον *αριθμό τεμαχίων*
και ο κατακόρυφος άξονας το *κόστος τεχνολογίας ανά τεμάχιο*.

Τα ακόλουθα δύο διαγράμματα παρατίθενται για να γίνει πιο εμφανής η διαφορά(μέχρι 20000 τεμάχια) :



Εδώ παρουσιάζεται το διάστημα όπου γίνεται εμφανές το σημείο όπου η τεχνολογία SoC-2 γίνεται συμφερότερη από την SoC.



Σχολιασμός:

- Από τα παραπάνω διαγράμματα παρατηρούμε ότι μέχρι τα 500 περίπου τεμάχια υπερτερεί η τεχνολογία των **FPGAs**.
- Από 500 μέχρι 5000 περίπου τεμάχια υπερτερεί όλων η τεχνολογία των **διακριτών στοιχείων**.
- Από 5000 μέχρι 50000 περίπου τεμάχια η πιο συμφέρουσα είναι η τεχνολογία **SoC-1**.
- Από 50000 τεμάχια και πάνω χαμηλότερο κόστος ανά τεμάχιο έχει η τεχνολογία **SoC-2** (σε αντιδιαστολή με ότι το αρχικό κόστος σχεδίασης είναι πολύ μεγαλύτερο από τις προηγούμενες κατηγορίες) .

Σχόλιο:
Για μεγάλες παραγωγές τεμαχίων συμφέρει περισσότερη η SoC-2.

Θεωρώντας τώρα ως άγνωστο (N) την τιμή των I.C. στη τεχνολογία των FPGAs. Η **συνάρτηση κόστους** γίνεται:

$$\mathrm{K}_F(x) = 10000 + (N+10)\cdot x$$

Αντίστοιχα θεωρούμε ως άγνωστο (M) την τιμή αρχικού κόστος SoC-1 οπότε η **συνάρτηση κόστους** γίνεται:

$$K_{SoC-1}(x) = M + (2+2)x = M + 4x$$

Για να εξαφανιστεί η επιλογή της τεχνολογίας των διακριτών στοιχείων θα πρέπει να ισχύει:

$$\mathrm{K}_{\Delta\Sigma}(x) - \mathrm{K}_F(x) > 0 \Leftrightarrow 20000 + 20x - 10000 - Nx - 10x > 0$$

$$\mathrm{K}_{\Delta\Sigma}(x) - \mathrm{K}_{SoC-1}(x) > 0 \Leftrightarrow 20000 + 20x - M - 4x > 0$$

Οι επιτρεπόμενες τιμές των N,M εξαρτώνται από το πλήθος τεμαχίων που παράγουμε. Επιλύοντας τις παραπάνω ανισώσεις έχουμε:

$$N < \frac{10000}{x} + 10$$

$$M < 20000 + 16x$$

Από τις παραπάνω ανισώσεις παίρνουμε τις οριακές τιμές.

Συμπέρασμα:
Επομένως παρατηρούμε ότι ανάλογα με την παραγωγή που θέλουμε μπορούμε να μεταβάλλουμε κατάλληλα τα N,M ώστε οι τεχνολογίες των FPGAs και SoC να μπορούν να εξαφανίσουν την τεχνολογία των διακριτών στοιχείων.