Συστήματα Μικροϋπολογιστών 2023-24 — 1η Ομάδα Ασκήσεων

Κόρδας Νικόλαος - Α.Μ.: 03121032
Κριθαρίδης Κωνσταντίνος - Α.Μ.: 03121045

15 Απριλίου 2024

1η ΑΣΚΗΣΗ

Δίνεται ο ακόλουθος κώδικας σε δεκαεξαδική μορφή όπου τα bold ειναι opcodes:

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

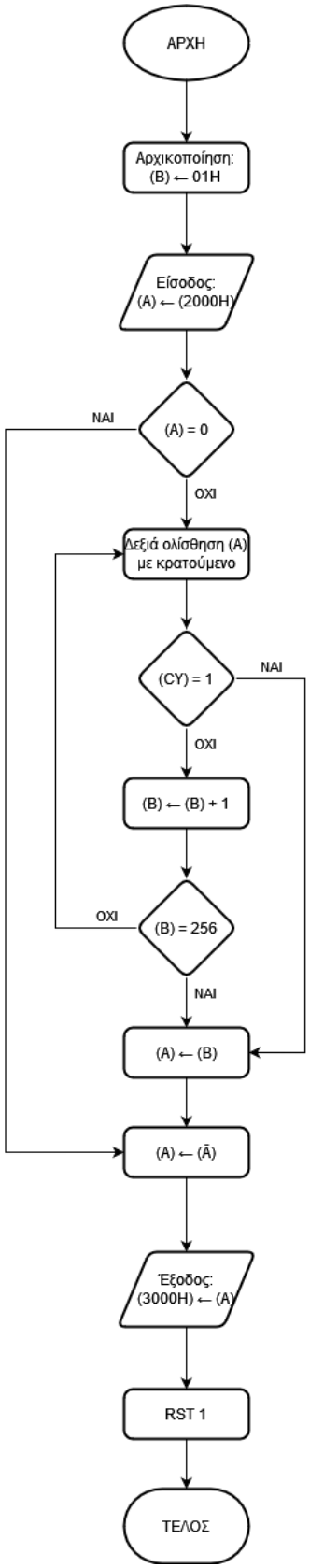
Με τη βοήθεια του πίνακα 2 του παραρτήματος 2 του αρχείου "Εισαγωγή στο Εκπαιδευτικό Σύστημα mLAB" κάναμε disassemle τον παραπάνω κώδικα λαμβάνοντας τις εντολές που αντιστοιχούν στα opcodes. Προσέχουμε ότι όπου χρησιμοποιούνται 16-bit αριθμοί, αυτοί γράφονται παραπάνω σε μορφή LO HI. Ο κώδικας σε Assembly φαίνεται παρακάτω:

```
    MVI B,01H
    LDA 2000H
    CPI 00H
    JZ JUMP1
JUMP3: RAR
    JC JUMP2
    INR B
    JNZ JUMP3
JUMP2: MOV A,B
JUMP1: CMA
    STA 3000H
    RST 1
END
```

Για να κατανοήσουμε καλύτερα τη λειτουργία του κώδικα, δημιουργούμε το διάγραμμα ροής του που απεικονίζεται στην επόμενη σελίδα.

Πρώτα αρχικοποιούμε τον καταχωρητή A στην τιμή 1. Στη συνέχεια διαβάζουμε από την είσοδο (θέση 2000H) με τους διακόπτες και αποθηκεύουμε την τιμή που μας δίνεται στον καταχωρητή A. Στην έξοδο (θέση 3000H) θέλουμε να τυπώσουμε στα λεντάκια την θέση του least significant set bit της εισόδου. Τα λεντάκια λειτουργούν με αρνητική λογική, και θέλουμε το αναμμένο να αναπαριστά το 1, ενώ το σβηστό να αναπαριστά το 0.

Αν ο A είναι 0, τότε η είσοδος δεν έχει κανένα 1 στην δυαδική της αναπαράσταση, οπότε δεν ανάβουμε κανένα λαμπάκι. Αφού έχουμε αρνητική λογική, αυτό επιτυγχάνεται συμπληρώνοντας ως προς 1 το A, οπότε γίνεται 1111 1111, και άρα όλα τα λεντάκια παραμένουν σβηστά. Αν ο A δεν είναι 0, τότε τον ολισθάμε προς τα δεξιά με κρατούμενο, και προσθέτουμε 1 στον B μέχρι το κρατούμενο να γίνει 1. Σε αυτήν την περίπτωση, βρήκαμε το least significant set bit της εισόδου και ο B έχει αποθηκευμένη τη θέση του. Αν ο B κάνει υπερχείλιση και γίνει 0 (αντιστοιχεί στο 256) επίσης φεύγει από την επανάληψη. Αφού βγούμε από την επανάληψη και ο B περιέχει την θέση που θέλουμε, την μεταφέρουμε στον A, συμπληρώνουμε ως προς 1 τον A και τον τυπώνουμε στην έξοδο στα λεντάκια με αρνητική λογική, λαμβάνοντας το επιθυμητό αποτέλεσμα. Ύστερα, καλούμε την ρουτίνα εξυπηρέτησης διακοπής 1. Η διακοπή αυτή (η οποία ισοδυναμεί με CALL 0008H) αποθηκεύει τα περιεχόμενα των καταχωρητών στη RAM και επιστρέφει στο monitor πρόγραμμα, με αποτέλεσμα τη δυνατότητα παρατήρης των αποτελεσμάτων του κώδικα χωρίς να τον τρέξουμε βηματικά.


ΑΡΧΗ
Αρχικοποίηση:
(B) ← 01H
Είσοδος:
(A) ← (2000H)
(A) = 0
ΝΑΙ
ΟΧΙ
Δεξιά ολίσθηση (A)
με κρατούμενο
(CY) = 1
ΝΑΙ
ΟΧΙ
(B) ← (B) + 1
(B) = 256
ΟΧΙ
ΝΑΙ
(A) ← (B)
(A) ← (Ā)
Έξοδος:
(3000H) ← (A)
RST 1
ΤΕΛΟΣ

2η ΑΣΚΗΣΗ

Πρώτα από όλα παραθέτουμε τον κώδικα assembly για τον 8085 που αφορά την λύση της άσκησης.

```
IN 10H
MVI D,01H ; Initial state of lights
MVI E,01H ; Flag to go to the left
LXI B,03E8H ; 1s for the delay

SWITCH_LIGHTS:
    MOV A,D
    CMA
    STA 3000H ; Move state to output port
    CALL DELB ; Delay between the results

INPUT:
    LDA 2000H ; Read dip switch
    ANI 03H ; Keep two LSBs
    CPI 02H ; Check if 2nd LSB is on
    JNC SWITCH_LIGHTS ; if it is, stay as you are
    CPI 01H ; Is the LSB on?
    JZ MOVE_LEFT ; If it is move left
    JNZ CIRCLE ; If not circle

MOVE_LEFT:
    MOV A,E
    CPI 01H
    JNZ STEER_RIGHT
STEER_LEFT: MOV A,D
        RLC ; Rotate left
        MOV D,A
        JNC SWITCH_LIGHTS
        MVI E,00H ; Flag right
        MVI D,80H ; D ready
        JMP SWITCH_LIGHTS

STEER_RIGHT: MOV A,D
         RRC ; Rotate right
         MOV D,A
         JNC SWITCH_LIGHTS
         MVI E,01H ; Flag left
         MVI D,01H
         JMP SWITCH_LIGHTS


CIRCLE:
    MOV A,D
    RLC ; Rotate left
    MOV D,A
    JNC SWITCH_LIGHTS ; If no overflow, output
    MVI D,01H ; If overflow, begin again
    JMP SWITCH_LIGHTS

END
```

Το αντίστοιχο αρχείο προσομοίωση θα βρίσκεται στο αρχείο ex2.8085 που θα παραδοθεί συμπιεσμένο. Ουσιαστικά, στις πρώτες 4 εντολές καλούμε την εντολή που προτείνεται για να άρουμε την προστασία της μνήμης, φορτώνουμε στον καταχωρητή D την κατάσταση που θέλουμε να ξεκινήσουν τα λαμπάκια (σε θετική λογική), στον καταχωρητή E μια σημαία που θα μας βοηθήσει την κίνηση αριστερά-δεξιά, αλλά και 1 δευτερόλεπτο καθυστέρηση (έχει λίγο καλύτερη οπτικοποίηση από το μισό) με το οποίο θα καλούμε την συνάρτηση DELB (από το documentation είδαμε ότι χρησιμοποιεί ως όρισμα το περιεχόμενο του διπλού καταχωρητή B-C). Στην συνέχεια, προκειμένου να ανάψουμε τα λαμπάκια συμπληρώνουμε το περιεχόμενο του καταχωρητή D (τα λαμπάκια λειτουργούν με αρνητική λογική) και μεταφέρουμε το αποτέλεσμα στην έξοδο, εισάγοντας και κάποια καθυστέρηση για να το δούμε καθαρά. Κατόπιν, διαβάζουμε την κατάσταση του dip switch (input) στον καταχωρητή A και κρατάμε τα 2 LSB, αυτά που μας ενδιαφέρουν. Πρώτα, κοιτάμε αν το δεύτερο LSB είναι αναμμένο και αν ναι, μένουμε ως έχουμε συνεχώς. Αν όχι, ελέγχουμε το LSB και πηγαίνουμε στο κομμάτι κώδικα που εκτελεί την αντίστοιχη ενέργεια. Για να υλοποιήσουμε την κυκλική εναλλαγή ουσιαστικά κάτουμε συνεχώς shift left προσέχοντας τους πλήρεις κύκλους με κατάλληλο χειρισμό του κρατούμενου που προκύπτει. Η "ταλάντωση" υλοποιείται και αυτή με λογικά shifts και την χρήση μιας σημαίας στον καταχωρητή E που μας ενημερώνει για την κατεύθυνση την οποία κινούμαστε.

3η ΑΣΚΗΣΗ

Το πρόβλημα που καλούμαστε να λύσουμε είναι το εξής: Μας δίνονται επαναληπτικά αριθμοί από τα dip switches εισόδου (θέση 2000H). Για κάθε αριθμό που μας δίνεται, αν είναι μεγαλύτερος ή ίσος του 200, θέλουμε να ανάψουμε όλα τα λεντάκια εξόδου (θέση 3000H). Αν είναι μικρότερος του 100, θέλουμε να ανάψουμε τα λεντάκια εξόδου ώστε να αναπαριστούν σε BCD τα δεκαδικά ψηφία του αριθμού. Αλλιώς, θέλουμε να ανάψουμε τα λεντάκια εξόδου ώστε να αναπαριστούν σε BCD τα 2 λιγότερο σημαντικά δεκαδικά ψηφία του αριθμού. Η αναπάσταση με τα λεντάκια θέλουμε να γίνεται με θετική λογική (ενώ λειτουργούν με αρνητική). Παραθέτουμε παρακάτω τον κώδικα που το επιτυγχάνει.

```
START: LDA 2000H  ; Get number from input dip switches
       CPI C7H    ; Is greater than 199
       JNC OVER200  ; If yes, light all LEDs and get next number
       CPI 63H    ; Is greater than 99
       JC UNDER100  ; If not, continue to output its digits
       SUI 64H    ; If yes, subtract 100 to keep its last 2 digits
UNDER100: MVI B,FFH ; FFH stands for -1 in signed two's complement arithmetic
DECA:
       INR B    ; Increase the tens counter
       SUI 0AH    ; Subtract 10
       JNC DECA   ; If A is positive, continue subtracting 10s
       ADI 0AH    ; Else, fix the negative remainder
       MOV C,A    ; Store A's value (units) for later
       MOV A,B    ; Store the number of tens in A
       RLC    ; Shift the tens to the 4MSBs
       RLC
       RLC
       RLC
       ORA C  ; Put the units in the 4 LSBs
       CMA    ; The LEDs have negative logic and we want positive
       STA 3000H  ; Print the answer to the LEDs output
       JMP START  ; Get the next number
OVER200:
       XRA A  ; Clear A so that all LEDs will light up
       STA 3000H  ; Print the answer to the LEDs output
       JMP START  ; Get the next number
END
```

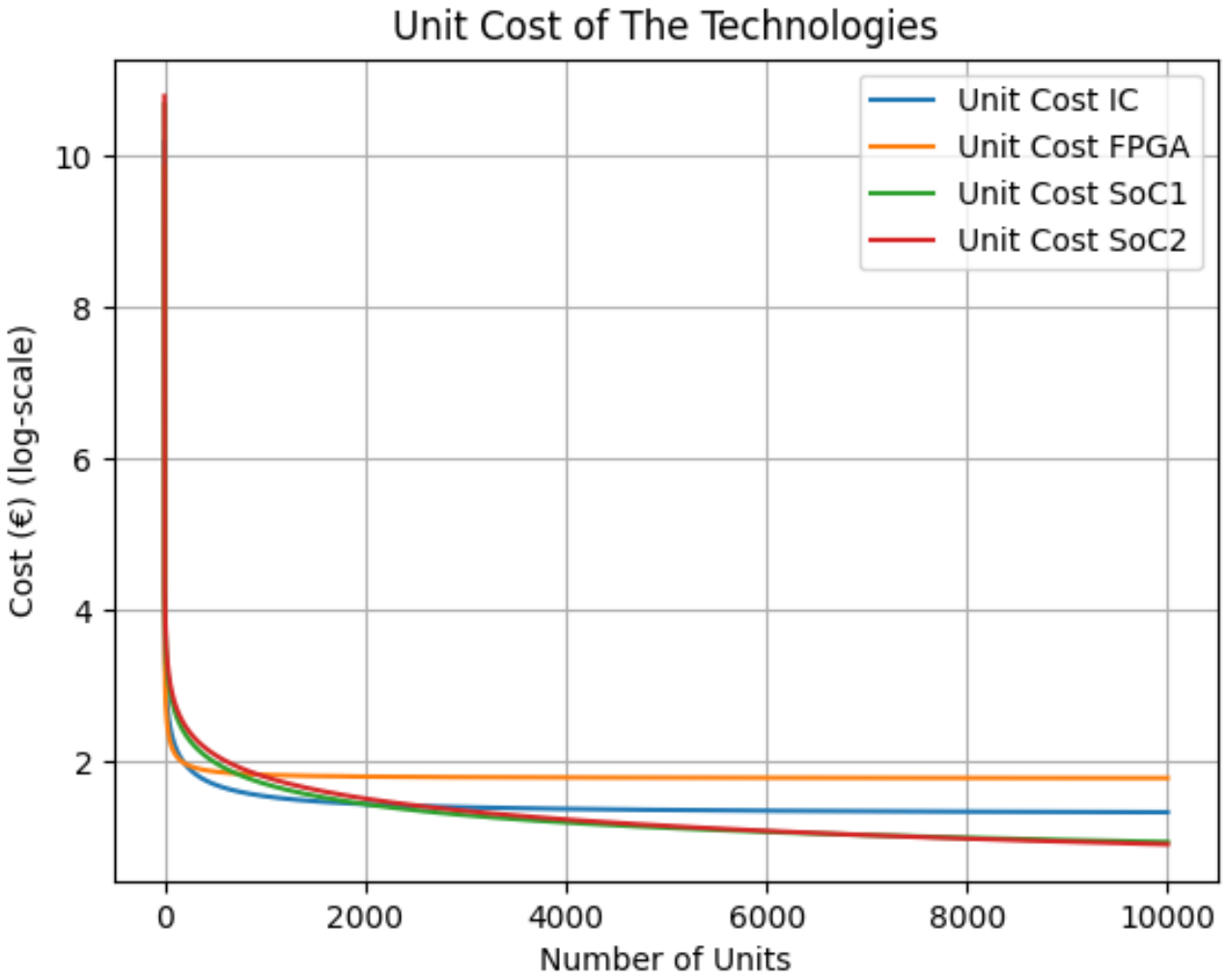
Λαμβάνουμε επαναληπτικά από τα dip switches την είσοδο και την αποθηκεύουμε στον A. Ελέγχουμε σε ποιό διάστημα ανήκει ο αριθμός και κάνουμε jump σε αντίστοιχες θέσεις για να τον χειριστούμε ανάλογα. Η γενική δομή του κώδικα λήφθηκε από το παράδειγμα 4 στην σελίδα 84 του βιβλίου. Οι δεκάδες μετρώνται αυξάνοντας την τιμή ενός μετρητή σε διαδοχικές αφαιρέσεις του 10 μέχρι ο αριθμός να γίνει αρνητικός, οπότε του ξαναπροσθέτουμε 10 και λαμβάνουμε τις μονάδες. Η έξοδος γίνεται τοποθετώντας στον A τις δεκάδες και κάνοντας 4 left shifts ώστε αυτές να μεταφερθούν στα 4 MSBs του. Στη συνέχεια κάνουμε OR με τις μονάδες, οπότε αυτές τοποθετούνται στα LSBs (χωρίς να επηρεάσουν τις δεκάδες αφού πιάνουν το πολύ 4 bits). Για να χειριστούμε τα LEDs με θετικη λογική πρέπει να αντιστρέψουμε τον A πριν τον αποθηκεύσουμε στη θέση 3000H όπου αυτά βρίσκονται. Στο τέλος, επιστρέφουμε στην αρχή του κώδικα για να διαβάσουμε τον επόμενο αριθμό.

4η ΑΣΚΗΣΗ

Οι σχέσεις για το κόστος ανά τεμάχιο (σε €) για την κάθε τεχνολογία είναι:

- IC: $K_u(x) = \frac{15000}{x} + 20$
- FPGA: $K_u(x) = \frac{7000}{x} + 60$
- SoC1: $K_u(x) = \frac{47000}{x} + 4$
- SoC2: $K_u(x) = \frac{61000}{x} + 2$

Θα παρουσιάσουμε τις καμπύλες του κόστους ανά τεμάχιο για την κάθε τεχνολογία σχεδίασης. Η απεικόνιση γίνεται σε λογαριθμική κλίμακα για να είναι πιο ευπαρουσίαστη:

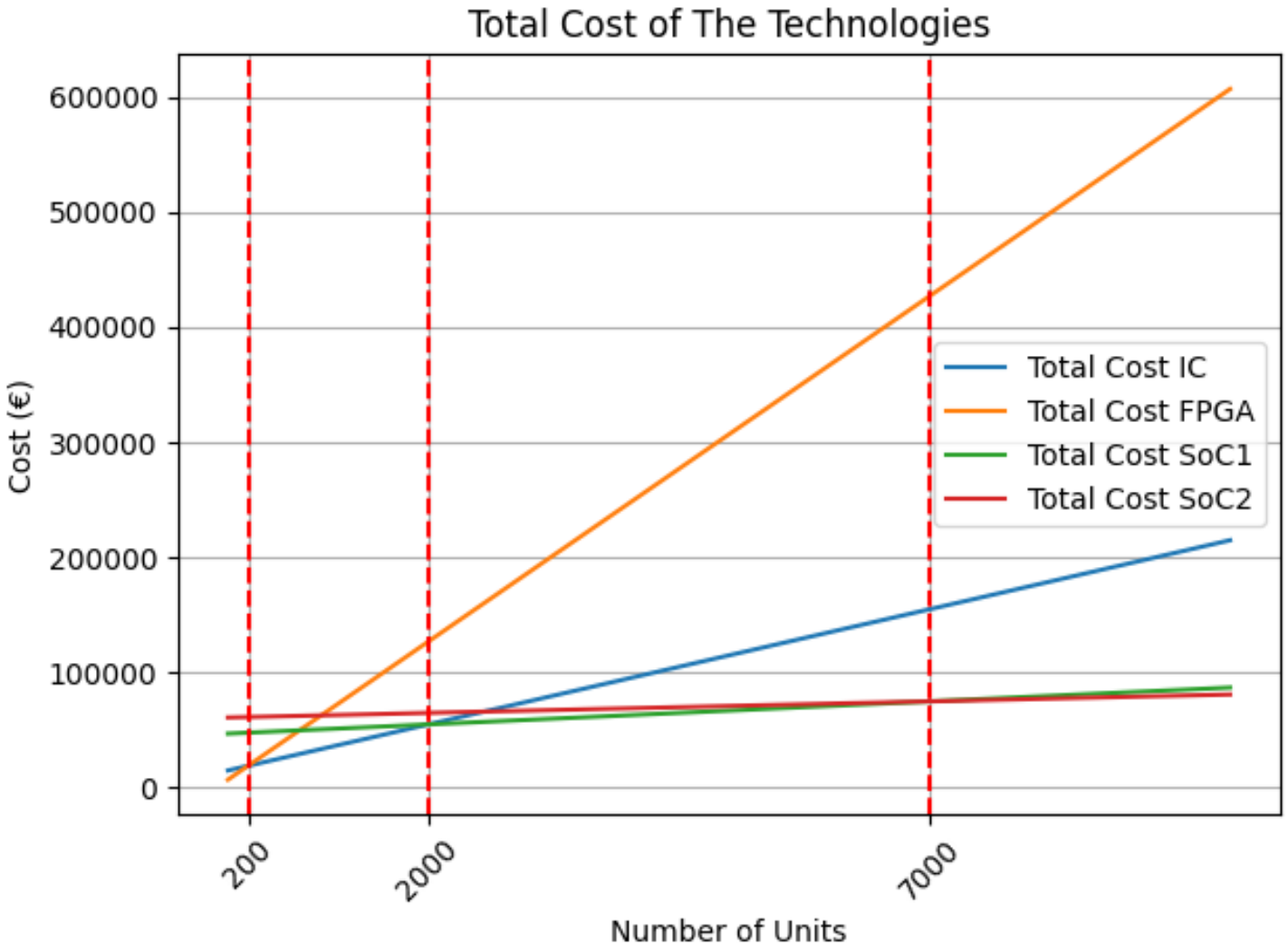


Παρατηρούμε πως το κόστος ανά τεμάχιο για τις τεχνολογίες SoC1 και SoC2 είναι σχεδόν ίδιο για εύρος τεμαχίων από 0 έως 10000 (η πράσινη και η κόκκινη καμπύλη σχεδόν ταυτίζονται). Αξίζει να σημειώσουμε επίσης πως η υλοποίηση με FPGA είναι, σχεδόν σταθερά, η ακριβότερη επιλογή ανά τεμάχιο. Τέλος, βλέπουμε πως για αριθμό τεμαχίων μικρότερο περίπου των 3000, το κόστος ανά τεμάχιο των συσκευών κατασκευασμένων με διακριτά IC είναι μικρότερο από των δύο SoC. Αλλά, μετά από αυτόν τον αριθμό, το κόστος των τεχνολογιών SoC ανά τεμάχιο είναι συμφερότερο.

Οι σχέσεις του συνολικού κόστους (σε €) για τις 4 διαφορετικές τεχνολογίες:

- IC: $K_t(x) = 15000 + 20x$
- FPGA: $K_t(x) = 7000 + 60x$
- SoC1: $K_t(x) = 47000 + 4x$
- SoC2: $K_t(x) = 61000 + 2x$

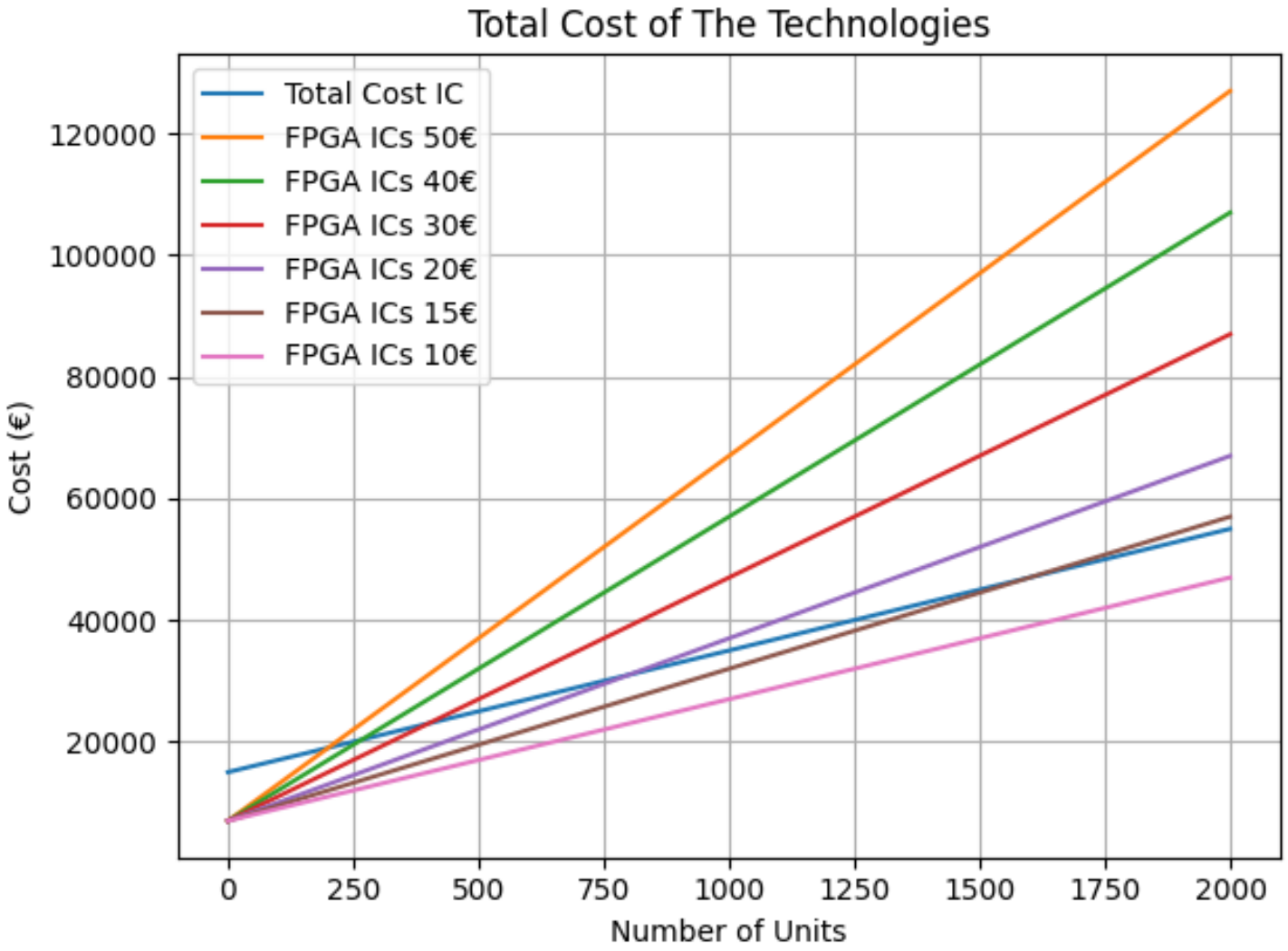
Τώρα παρουσιάζουμε τις γραφικές παραστάσεις των παραπάνω σχέσεων:



Στην οπτικοποίησή μας φαίνονται με κόκκινες διακεκκομμένες γραμμές που υποδεικνύουν τις περιοχές για τις οποίες είναι συμφερότερη κάθε τεχνολογία - δηλαδή, η γραφική παράσταση του κόστους της βρίσκεται χαμηλότερα από τις υπόλοιπες. Πιο συγκεκριμένα, διαπιστώνουμε πως για αριθμό τεμαχίων μικρότερο των 200, η συμφέρουσα επιλογή είναι η κατασκευή με τεχνολογία FPGA. Για αριθμό τεμαχίων από 201 έως 2000, καλύτερη επιλογή είναι η τεχνολογία με διακριτά IC. Για το εύρος τεμαχίων 2001 έως 7000 μας συμφέρει η τεχνολογία SoC1 ενώ για αριθμούς συσκευών από 7001 και πάνω, έστω και για λίγο, φθηνότερη είναι η τεχνολογία SoC2.

Συμπερασματικά, αυτό που προκύπτει και από τα 2 γραφήματα είναι ότι για μικρό αριθμό τεμαχίων συμφέρει η τεχνολογία FPGA, ενώ για μεγάλες ποσότητες η τεχνολογία SoC (και η 1 και η 2).

Τέλος, μας ζητείται να διαπιστώσουμε για ποιο κόστος και κάτω των IC ανά τεμάχιο δεν αξίζει καν να σκεφτούμε την υλοποίηση αποκλειστικά με ICs. Για αυτό, δημιουργούμε τις καμπύλες συνολικού κόστους για το διαφορετικό κόστος τεμαχίων IC για την (γενικά ακριβότερη) υλοποίηση με FPGA, ταυτόχρονα με εκείνη της υλοποίησης αποκλειστικά με IC.



Αυτό που βλέπουμε είναι πως για τιμές IC 15€ ανά τεμάχιο και πάνω, μπορεί αρχικά τα FPGA είναι είναι φθηνότερα, αλλά από έναν αριθμό τεμαχίων και πάνω, αυτό παύει να ισχύει. Αλλά, για τιμές 10€ και κάτω ανά τεμάχιο, βλέπουμε πως η ροζ γραμμή μένει πάντα κάτω από αυτή των IC. Οπότε θεωρητικά, αν μπορούσαμε να έχουμε κόστος ολοκληρωμένων κάτων απο 10€ ανά τεμάχιο για την υλοποίηση με FPGA δεν θα άξιζε καν να εξετάσουμε εκείνη με διακριτά ICs.