- Συστήματα Μικροϋπολογιστών - Ροή Υ
- 1η Ομάδα Ασκήσεων
- Ναυσικά Αμπατζή (el17198)
- Χρήστος Σιαφαρίκας (el17097)
- Ακαδημαϊκό Έτος 2019-2020

1η ΄Ασκηση

Δίνεται το παρακάτω πρόγραμμα σε γλώσσα μηχανής και υποθέτουμε ότι είναι φορτωμένο στη μνήμη με αρχή τη διεύθυνση 0800.

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

Θα το μετατρέψουμε σε assembly του μΥ-Σ 8085 με τη χρήση των πινάκων των σημειώσεων. Στις παρενθέσεις στα σχόλια δίνονται τα δεδομένα της άσκησης.

06 : MVI B,byte // Φόρτωση απ'ευθείας του δεδομένου(01H) στον καταχωρητή B.

3A : LDA, addr // Φόστωση στον καταχωρητή A το περιεχόμενο που βρίσκεται στην διεύθυνση addr(2000H)

FE : CPI, byte // Σύγκριση του περιεχομένου του καταχωρητή A με άμεσο δεδομένο(00H)

CA : JZ addr // Εάν το αποτέλεσμα της παραπάνω σύγκρισης είναι 0 (Z = 1) jump στο label με διεύθυνση addr(0813H)

1F : RAR // Περιστροφή προς τα δεξιά των bits του καταχωρητή A και εκχώρηση στο CY το A0

DA : JC addr // Εάν το κρατούμενο CY = 1 jump στο label με διεύθυνση addr(0812H)

04 : INR B // Άυξηση του περιεχομένου του καταχωρητή B κατά 1

C2 : JNZ addr //Εάν το αποτέλεσμα είναι μη μηδενικό (Z = 0) jump στο label με διεύθυνση addr(080AH)

78 : MOV A,B // Αντιγραφή του περιεχομένου του B στον καταχωρητή A

2F : CMA // Συμπλήρωση ως προς 1 του περιεχομένου του καταχωρητή A

32 : STA addr // Το περιεχόμενο του A αποθηκεύεται στη διεύθυνση addr(3000H)

CF : RST 1 // Διακοπή του κώδικα και μεταφορά του PC στη διεύθυνση 0800H(αρχή του προγράμματος)

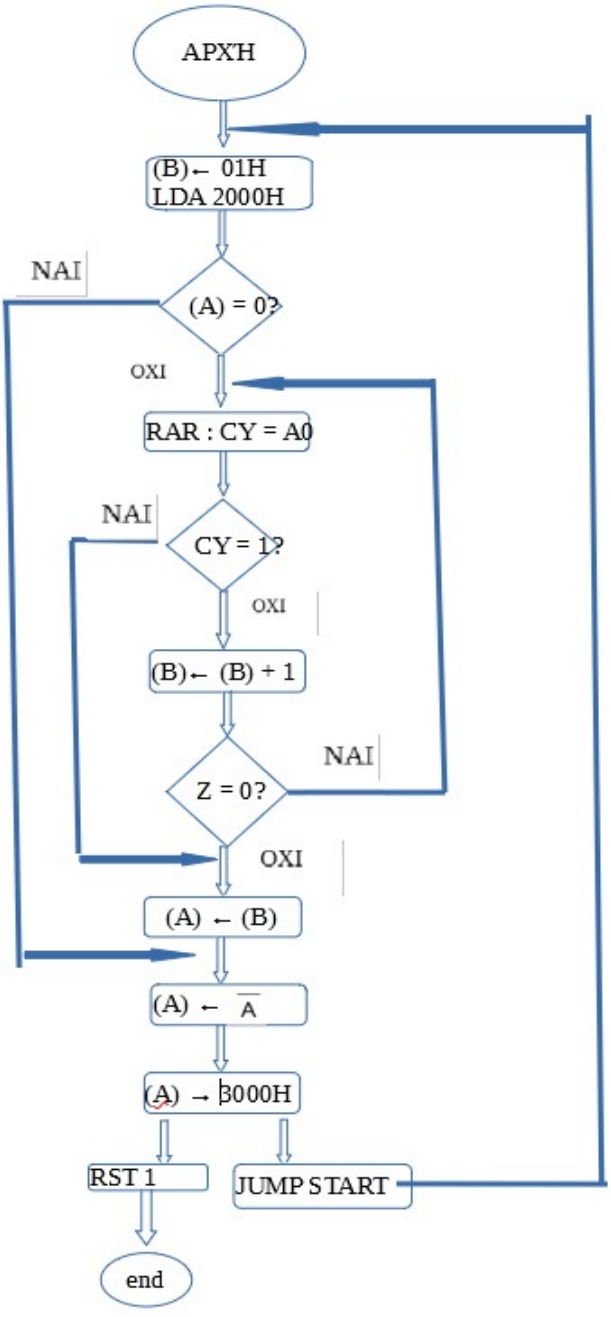
Σύμφωνα με τα νούμερα που έχουν δωθεί για τις διευθύνσεις και τα περιεχόμενα των καταχωρητών προκύπτει το εξής πρόγραμμα σε γλώσσα assembly:

MVI B,01H
LDA 2000H
CPI 00H
JZ L3
L1:
RAR
JC L2
INR B
JNZ L1
L2:
MOV A,B
L3:
CMA
STA 3000H
RST 1

Για να επαναλαμβάνεται χωρίς τέλος ο παραπάνω κώδικας θα πρέπει στην αρχή του προγράμματος να ορίσουμε μία ετικέτα START και η εντολή **RST** 1 να αντικατασταθεί με την εντολή **JUMP** START.

Μετά από τρέξιμο στον προσομοιωτή βρέθηκε ότι το πρόγραμμα αυτό ανάλογα με ποιον διακόπτη ανοίξουμε, ανάβει το αντίστοιχο led το οποίο αντιστοιχεί στη δυαδική αναπαράσταση της θέσης του διακόπτη μετρώντας από δεξιά προς τα αριστερά. Για παράδειγμα εάν ανοίξουμε τον τρίτο διακόπτη από δεξιά προς τα αριστερά το 3 αντιστοιχεί στο δεκαδικό 11. Άρα θα ανοίξουν τα δύο πρώτα leds. Εάν τη στιγμή που δίνουμε την
εντολή RUN ήταν παραπάνω από ένα dip switches ανοικτά, τότε θα ανοίξουν τα leds που αντιστοιχούν στο δεξιότερο dip switch σε κατάσταση ON.

Διάγραμμα Ροής



2η Άσκηση

Ακολουθεί το ζητούμενο πρόγραμμα:

```
IN 10H
MVI B,01H
MVI C,F4H          //Θα χρησιμοποιηθεί στην κλήση της DELB
FLASH:
LDA 2000H          //Εκκίνηση με φόρτωση του A στην είσοδο
RAR                // Περιστροφή προς τα δεξιά ώστε CY = a0 , δηλαδή η κατάσταση του
                    LSB της θύρας των dip switch
RAR                //Περιστροφή ξανά ώστε CY = a1 , δηλαδή η κατάσταση του 2ου LSB
JC STABLE          // Εάν  το 2ο LSB είναι ON jump στην STABLE
RAL                //Μία περιστροφή προς τα αριστερά ώστε να επανέλθουμε στην
                   κατάσταση του του  LSB της θύρας των dip switch
JNC KIKL           // Εάν το LSB είναι OFF jump στην KIKL

LEFT:              //Ρουτίνα κίνησης των leds προς τα αριστερά
MOV A,B
CMA                //Συμπληρώνεται το περιεχόμενο του A διότι τα leds είναι αρνητικής
                    λογικής και άρα θέλουν 0 στην είδοσο για να ανάψουν
STA 3000H          // Το A οδηγείται στην έξοδο ώστε να ανάψει το κατάλληλο led
CMA                // Επαναφέρουμε το περιεχόμενο του A
RLC
JC RIGHTT          // Όταν το led φτάσει στο 8ο MSB jump στην RIGHTT
MOV B,A
JMP FLASH

RIGHTT:
CALL DELB
MVI A,40H          //Αρχικοποίηση του A στο 68 και jump στην RIGHT
JMP RIGHT

KIKL:              // Όταν το LSB της θύρας των dip switch  είναι στο OFF το led κάνει
                   κυκλική κίνηση
CALL DELB
MOV A,B

CMA
STA 3000H
CMA
```

```
RLC
MOV B,A
JMP FLASH


STABLE:          // Εάν το 2ο LSB της θύρας των dip switch είναι στο ON ανάβει
                 χωρίς να κινείται το led της θέσης 0
MVI A,01H
CMA
STA 3000H
JMP FLASH


RIGHT:           //Ρουτίνα κίνησης των leds προς τα δεξιά
CALL DELB
CMA
STA 3000H
CMA
MOV B,A
LDA 2000H        // Φόρτωση του της εισόδου στον A
RAR
JNC KIKL         //Έλεγχος εάν το LSB είναι στο OFF

MOV A,B
RRC              // Έλεγχος για το εάν έφτασε το led ξανά στο 1
JNC RIGHT
MVI A,01H        //Όταν το led φτάνει στο 1 jump στην FLASH ώστε να ξεκινήσει να
                  κινείται αριστερά
JMP FLASH

END
```

3η Άσκηση

Ακολουθεί το ζητούμενο πρόγραμμα:

START:

```
LDA 2000H    //Φόρτωση της εισόδου
CPI C8H      //Σύγκριση του A  με το 200
JC BY //Εάν είναι μεγαλύτερο jump στην BY
SUI 64H      //Αλλιώς αφαίρεσε 100
```

BY:

```
CPI 64H      //Σύγκριση με το 100
JC START     //Εάν είναι μεγαλύτερο jump στην αρχή

SUI 64H      //Αλλιώς αφαίρεσε 100
MVI B,FFH    // B-<100
```

DECA:

```
INR B //Άυξησε το B
SUI 0AH       //Αλλεπάλληλες αφαιρέσεις του 10
JNC DECA     //Αν είναι θετικός συνέχισε
ADI 0AH      //Διόρθωση του αρνητικού υπολοίπου
MOV C,A      //Αποθήκευση του LSB
SUB A
ADD B
RLC          //4 περιστροφές για να φορτωθεί το MSB στην έξοδο
RLC
RLC
RLC
ADD C
CMA          //Αντιστροφή της εισόδου γιατί  τα leds είναι αρνητικής λογικής
STA 3000H
JMP START

END
```

4η Άσκηση

Έχουμε τρεις διαφορετικές τεχνολογίες και αρχικά θέλουμε να υπολογίσουμε το κόστος της κάθε μιας :

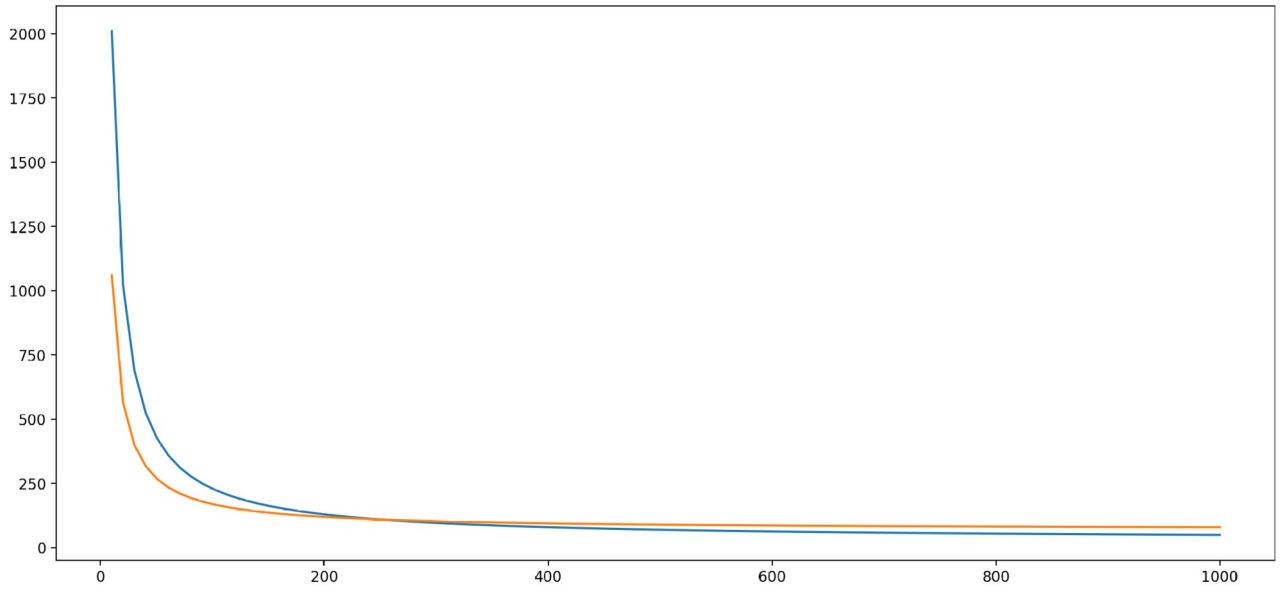
Η 1η είναι η χρήση διακριτών στοιχείων και ολοκληρωμένων μονάδων (I.C.) τα οποία συναρμολογούνται σε μια σε μια σχετικά μεγάλη πλακέτα. Η σχεδίαση κοστίζει 20.000 ενώ η κάθε πλακέτα κοστίζει 15 και η συναρμολόγηση της ανά τεμάχιο 15. Άρα το συνολικό κόστος υπολογίζεται και είναι : Κόστος = 20.000 + (15 +15) * χ , όπου χ είναι το σύνολο των τεμαχίων.

Η 2η τεχνολογία είναι η χρήση FPGAs και μικρού αριθμού περιφερειακών τοποθετημένα σε μια μικρή πλακέτα. Το αρχικό κόστος σχεδίασης είναι 10.000 και το κάθε τεμάχιο χρειάζεται 10 για συναρμολόγηση ενώ κοστίζει από μόνο του 60. Τελικά το συνολικό κόστος είναι : Κόστος = 10.000 + (60 + 10) * χ

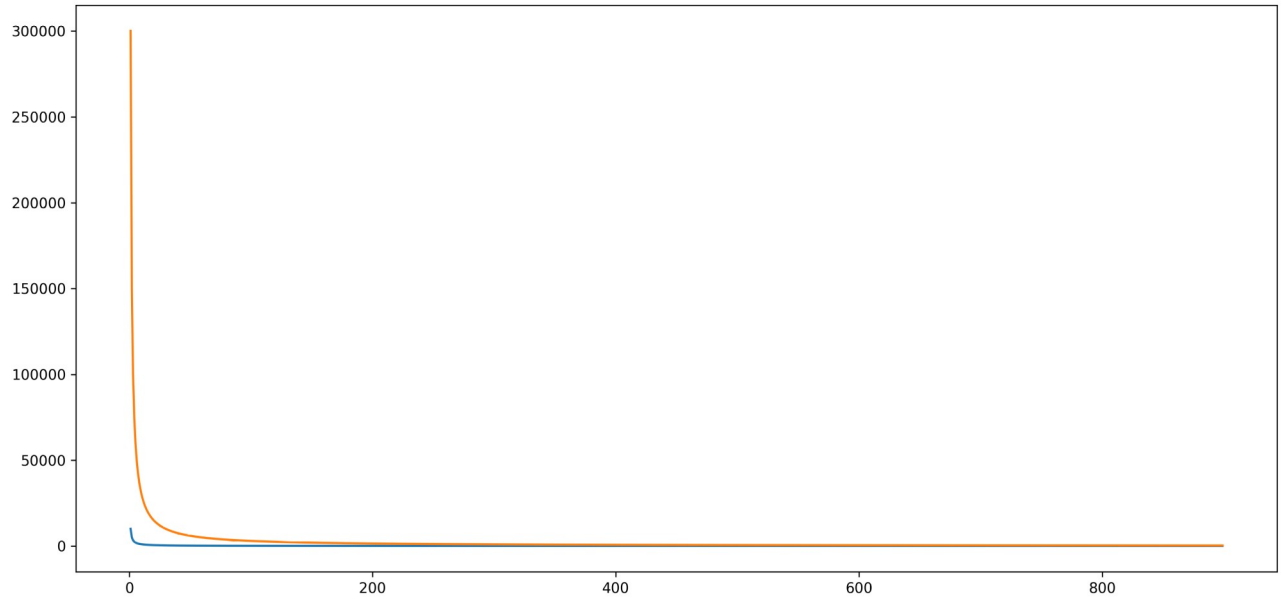
Η 3η τεχνολογία είναι η σχεδίαση ειδικού SoC με μια πολύ μικρή πλακέτα. Όπως και πριν το κόστος σχεδίασης είναι 300.000 αλλά η τιμή του τεμαχίου και της συναρμολόγησης είναι από 1 ευρώ. Έτσι το τελικό κόστος είναι: Κόστος = 300.000 + (1+1)*χ

Οι γραφικές παραστάσεις είναι:

- Για την 1η και τη 2η



- και για την 3η με τη 2η



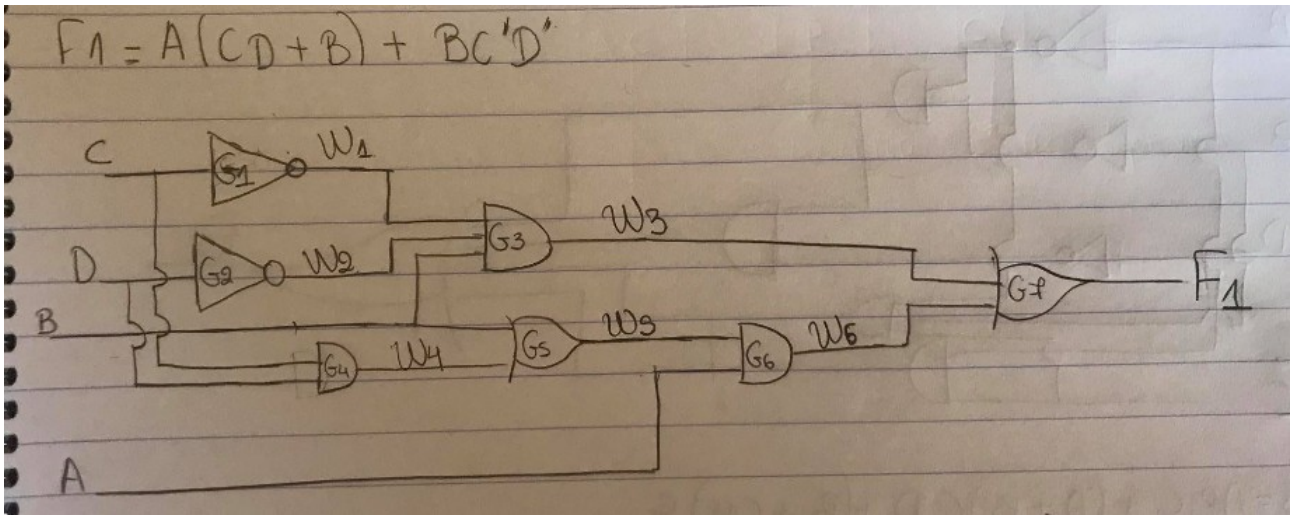
Για να βρούμε την περιοχή όπου είναι συμφερότερη η κάθε τεχνολογία παρατηρούμε σε ποιο σημείο έχουν το ίδιο κόστος άρα λύνουμε την εξίσωση :
Κόστος της 1ης = κόστος της 2ης
20.000 + (15 +15) * χ = 10.000 + (60 + 10) * χ
χ= 250
Επομένως αφού η 2η έχει μεγαλύτερη κλίση αλλά μικρότερη αρχική τιμή την επιλέγουμε για το διάστημα όπου χ<250 ενώ την 1η για χ>250
Τώρα λύνουμε την εξίσωση της 1ης με τη 3η
20.000 + (15 +15) * χ= 300.000 + (1+1)*χ
χ= 1000
Άρα με παρόμοια λογική θα επιλέξουμε την 1η για το διάστημα 250<χ<1000 και για χ>1000 επιλέγουμε την 3η αφού έχει μικρότερη κλίση (2<30)
Για να σταματήσει να συμφέρει η 1η για κάθε χ πρέπει από χ = 250 ακόμα και για χ=1000 να έχει μικρότερη τιμή η 2η τεχνολογία δηλαδή θέλουμε
20.000 + 30*χ > 10.000 +(ψ+10)*χ, όπου ψ το νέο κόστος των I.C.
Η λύση που θέλουμε είναι για μέγιστο χ = 1000 άρα θα πρέπει το ψ<= 30
Άρα η μέγιστη τιμή είναι ψ=30 ανά τεμάχιο .

5η Άσκηση

(I) Δομική περιγραφή των παρακάτω λογικών συναρτήσεων:

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

```
module Simple_Circuit(A,B,C,D);
        output F1;
        input A,B,C,D;
        wire w1, w2 , w3, w4, w5, w6;
        not G1(w1,C);
        not  G2(w2,D);
        and G3(w3,w1,w2,D);
        and  G4(w4,C,D);
        or G5(w5,w4,B);
        and  G6(w6,w5,A);
        or G7(F1,w3,w6);
endmodule
```

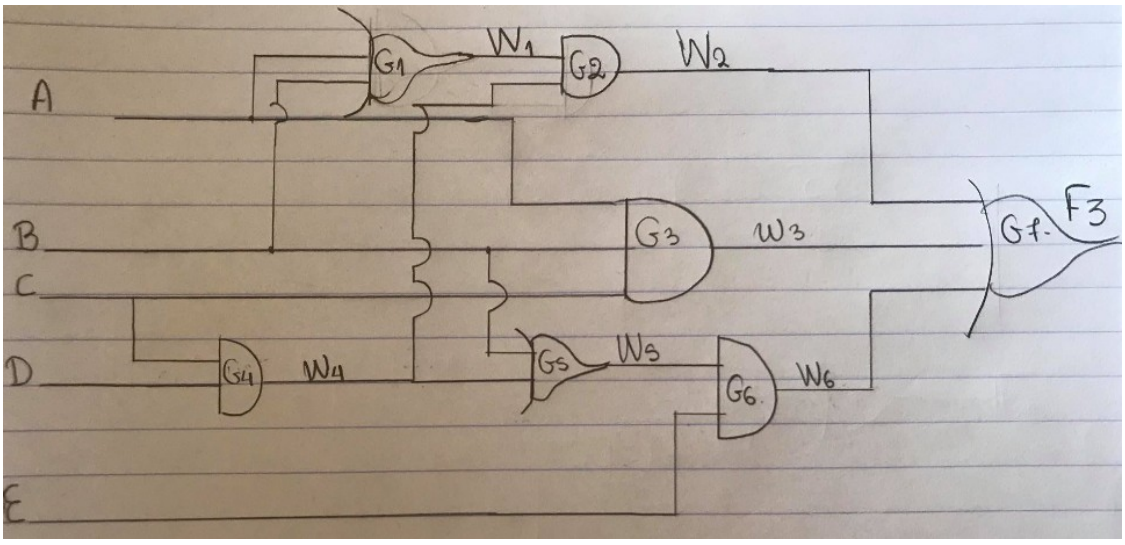


- F2(A,B,C,D) = Σ (0, 2, 3, 5, 7, 8, 10, 11, 14, 15)

```
primitive UDP_5(F2,A,B,C,D) // Verilog model: User-defined Primitive
output F2;
input A,B,C,D;
        table
        A B C D : F2
        0  0  0  0  :   1
        0  0  0  1 :  0
        0  0  1  0  :  1
        0  0  1  1  :  1
        0  1  0  0  :  0
        0  1  0  1  :   1
        0  1  1  0  :   0
        0  1  1  1  :   1
        1  0  0  0  :   1
        1  0  0  1  :   0
        1  0  1  0  :   1
        1  0  1  1  :   1
        1  1  0  0  :   0
        1  1  0  1  :  0
        1  1  1  1  :   1
        1  1  1  1  :   1
        endtable
endprimitive
```

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

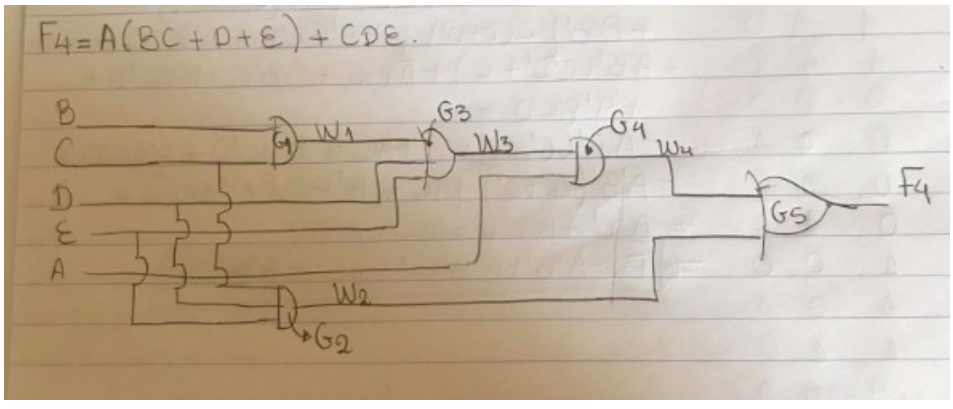
```
module Simple_Circuit(A,B,C,D,E);
        output F3;
        input A,B,C,D,E;
        wire w1, w2 , w3, w4, w5, w6;
        and G1(w1,A,B);
        and G4(w4,C,D);
        and G3(w3,A,B,C);
        or G5(w5,B,w4);
        and G2(w2,w1,w4);
        and G6(w6,w5,E);
        or G7(F3,w2,3,6);
endmodule
```



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

```
module Simple_Circuit(A,B,C,D,E);
        output F3;
        input A,B,C,D,E;
        wire w1, w2 , w3, w4;
        and G1(w1,B,C);
        and G2(w2,C,D,E);
        or G3(w1,D,E);
        and G4(w4,w3,A);
        or G5(F4,w4,w2);

endmodule
```



(II) Κώδικας Verilog για τις ίδιες συναρτήσεις σε Μοντελοποίηση ροής δεδομένων (με εντολές συνεχούς ανάθεσης):

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

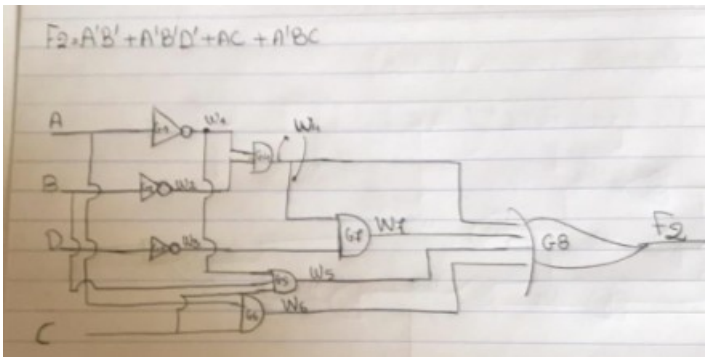
```
module Circuit(A,B,C,D);
        output F1;
        input A,B,C,D;
        assign F1 = A & ((C&D) | B) | (B & ~C & ~D);
endmodule
```

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

moduleCircuit(A,B,C,D);

output F2;
input A,B,C,D;
assign F2 = (~A & ~B) | (~A & ~B & ~D) | (A&C) | (~A & B & C);
endmodule

Το F2 υλοποιήθηκε σύμφωνα με το εξής κύκλωμα το οποίο είναι το αποτέλεσμα της ελαχιστοποίησης της F2 που προκύπτει από τον πίνακα αληθείας.



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

module Circuit(A,B,C,D,E);

output F3;
input A,B,C,D,E;
assign F3 = (A & B & C) | ((A|B) & C & D) | (B | (C & D))%E;
endmodule

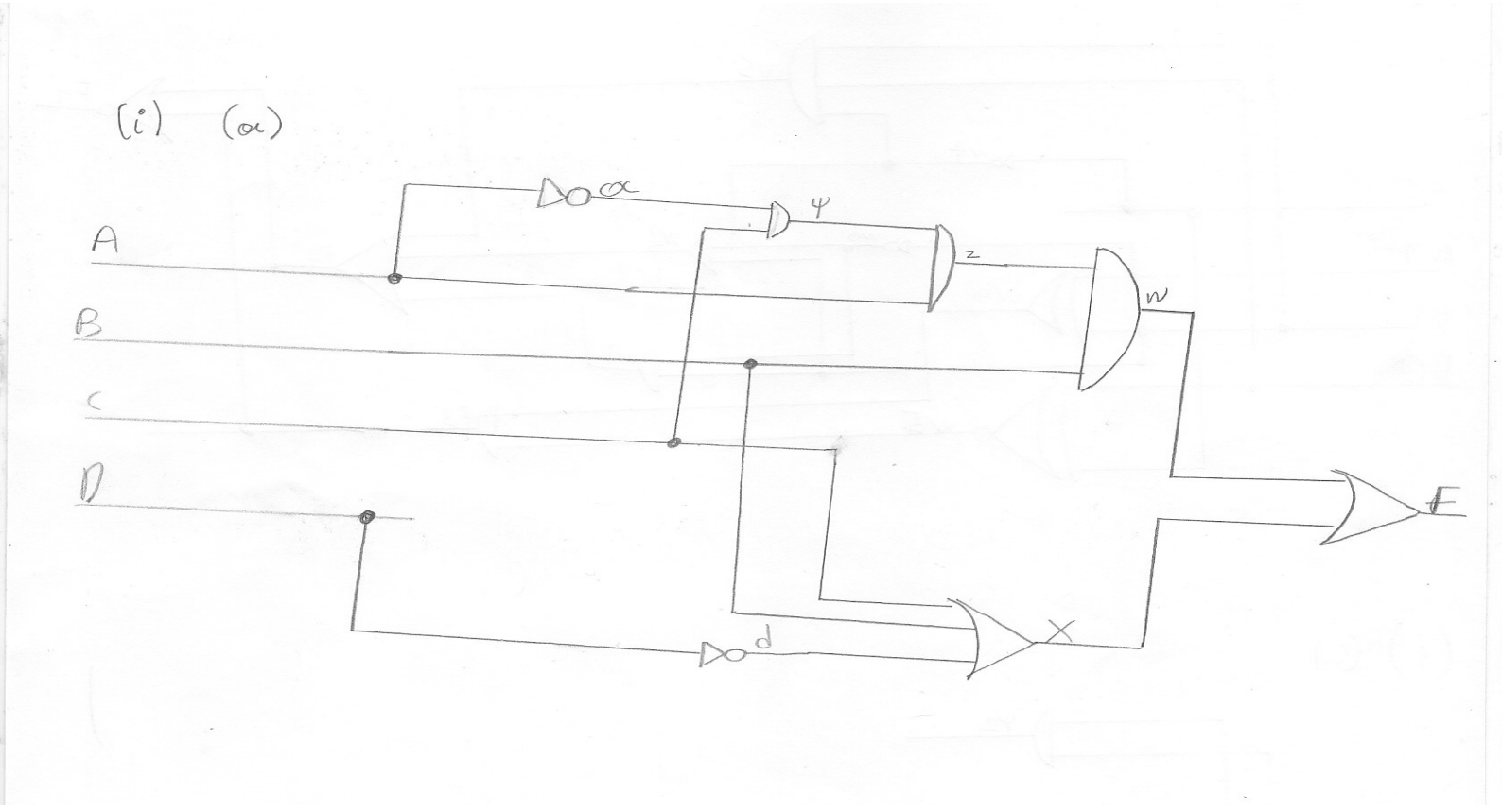
- F4=A(BC + D + E) + CDE

module Circuit(A,B,C,D,E);

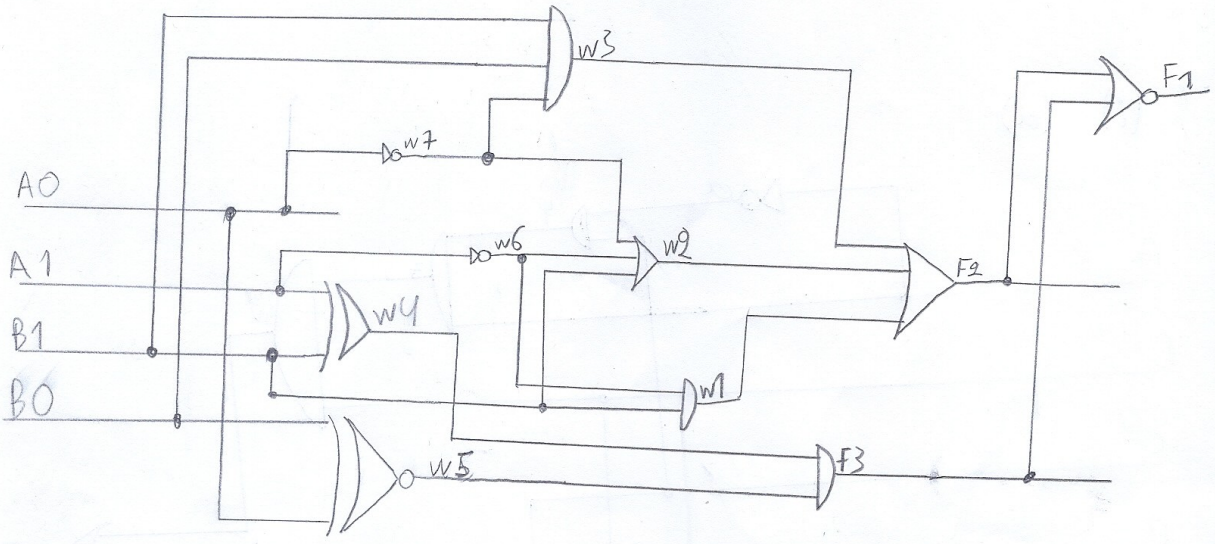
output F4;
input A,B,C,D,E;
assign F4 = (A & ((B & C) | D |E)) | (C & D & E);
endmodule

6η Άσκηση

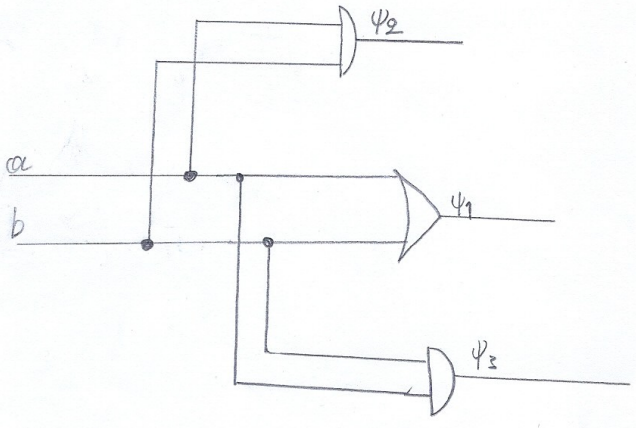
(I)



(i) (b)



(i) (c)



(II)

```
module half_adder(output S, C, input a , b)
xor (S, a, b);
and(C, a, b);
endmodule

module full_adder_substractor ( output S, C, input a , b , x, k)
wire w1, w2, w3;
xor (x1, b, k)
half_adder HA1 (w1 , w2 , a , b);
               HA2 (S , w3 , w1 , x);
or G1 (C , w3 , w2);
endmodule

module 4_bit_adder_substractor (output [3: 0] Sum, output C4, input [3: 0] A, B, input C0 , k)
wire w1 , w2, w3 , x1 , x2 , x3 , x4;
full_adder_substractor  FAS0 (Sum[0], w1, A[0], x1, C0, k),
                        FAS1 (Sum[1], w2, A[1], x2, w1, k),
                        FAS2 (Sum[2], w3, A[2], x3, w2, k),
                        FAS3 (Sum[3], C4, A[3], x4, w3, k);
endmodule
```

(III)

```
module 4_bit_adder_substractor (Sum, C4,  A, B, C0 , k)
output [3: 0] Sum;
output C4;
 input [3: 0] A, B;
 input C0 , k;
assign {C4,Sum}= k?A-B+C0:A+B+C0;
endmodule
```

7η Άσκηση

(I) //FSM Model

```
module FSM_Model(y,x,clock, reset );
        output y ;
        input x,clock, reset ;
        reg    [1:0]      state;
        always @(posedge clock, negedge reset)
                if(reset == 0) state <= a;
                else case(state)
                        a : if(~x) state  <= b ; y = 1; else state <= c; y = 0;
                        b : if(~x) state <= c; y = 0; else state <= d; y = 1;
                        c : if(~x) state <= b; y = 0; else state <= d; y = 1;
                        d : if(~x) state <= c; y = 1; else state <= a; y = 0;
                endcase;
        endmodule
```

(II) //FSM Model

```
module FSM_Model(y,x,clock, reset );
        output y ;
        input x,clock, reset ;
        reg state;
        parameter a= 1'b0, b= 1'b1, c = 1'b1, d = 1'b0;
        always @(posedge clock, negedge reset)
                if(reset == 0) state <= a;
                else case(state)
                        a : if(~x) state  <= b ; else state <= c;
                        b : if(~x) state <= c;  else state <= d;
                        c : if(~x) state <= b;  else state <= d;
                        d : if(~x) state <= c; else state <= a;
                endcase;
        assign y = state;
endmodule
```

(III)

```
module Up_Down_Counter (output reg [3: 0] A, input CLK, Up, Down, reset_b);

        always @ (posedge CLK, negedge reset_b)
                if (reset_b ==0) A <= 4'b0000;
                else case ({Up, Down})
                  2'b10: A <= A + 4'b0001;
                  2'b01: A <= A - 4'b0001;
                  default: A <= A;
                endcase
        endmodule
```