

Συστήματα Μικροϋπολογιστών

Απαντήσεις των ασκήσεων στην 1η
ομάδα ασκήσεων

3 ασκήσεις που ακολουθούν είναι όλες
ασκήσεις προσομοίωσης και εν συνεχεία,
ακολουθούν θεωρητικές ασκήσεις, Verilog και
προσομοιώσεις/κυκλώματα με την χρήση του
logisim

ΦΟΙΤΗΤΕΣ-ΕΠΙΜΕΛΕΙΑ: Αλκιβιάδης Μιχαλίτσης
και Βόικος Στέφανος
14/4/2021

Στοιχεία φοιτητών

Ονοματεπώνυμο: Αλκιβιάδης Μιχαλίτσης και Στέφανος Βόικος

A.M: el18868 και el18162

Ακαδημαϊκό εξάμηνο: 6^ο

Σχολή: Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
(Η.Μ.Μ.Υ)

ΑΠΑΝΤΗΣΕΙΣ ΤΩΝ ΑΣΚΗΣΕΩΝ ΤΗΣ ΟΜΑΔΑΣ 1 ΣΤΟ ΜΑΘΗΜΑ «ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ»

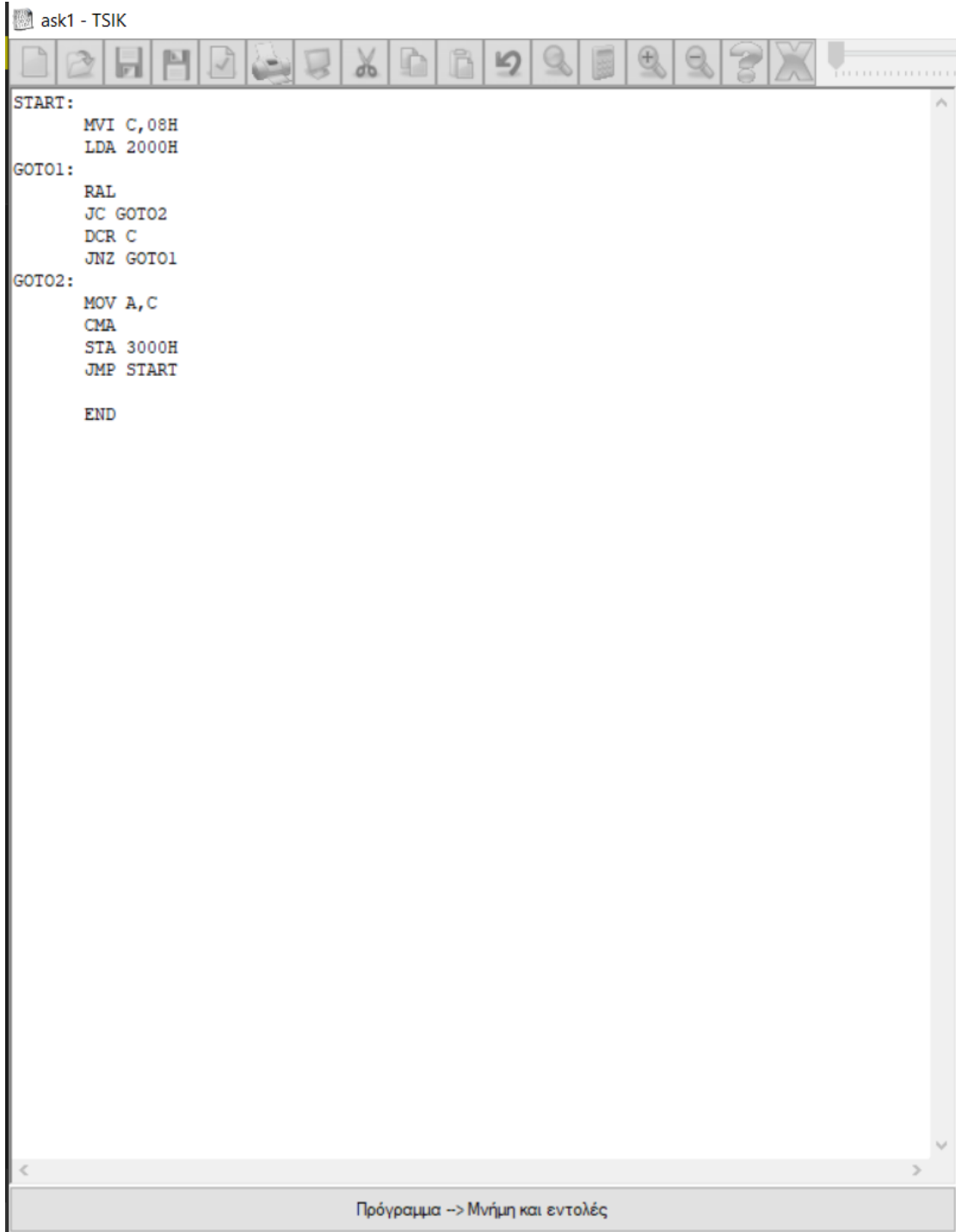
1^η ΑΣΚΗΣΗ:

0E 08 3A 00 20 17 DA 0D 08 0D C2 05 08 79 2F 32 00 30 CF

0E → MVI κ,byte //Μετακινεί τον αριθμό στον καταχωρητή κ={A,B,C,D,E,H,L}
3A → LDA *adr* // Φορτώνει στον καταχωρητή A το περιεχόμενο που βρίσκεται στην διεύθυνση *adr*= (2000)₁₆
17 → RAL // Περιστρέφει τα bits του καταχωρητή A και εκχωρεί στο CY(bit του καταχωρητή της σημαίας) το A0
DA → JC *label* //αν CY=1 μεταβαίνει στο *label* =(080D)₁₆ που δίνεται σε μορφή διεύθυνσης 2 byte.
0D → DCR κ //αφαιρεί 1 από το περιεχόμενο του καταχωρητή κ={A,B,C,D,E,H,L}
C2 → JNZ *label* // αν Z≠0 (το z=1 δίνεται από την σημαία αν η προηγούμενη εντολή είχε ως αποτέλεσμα 0) τότε μεταβαίνει στο *label*=(0805)₁₆
79 → MOV κ1,κ2 // αντιγράφει το περιεχόμενο του κ2 στον κ1 (κ1 ←κ2)

2F → CMA //συμπληρώνει ως προς 1 το περιεχόμενο του καταχωρητή A
32 → STA *adr* //αποθηκεύει το περιεχόμενο του καταχωρητή A στην διεύθυνση *adr* = (3000)₁₆
CF → RST 1 //διακόπτει τον κώδικα και πηγαίνει τον PC στην διεύθυνση (0800)₁₆

ask1 - TSIK

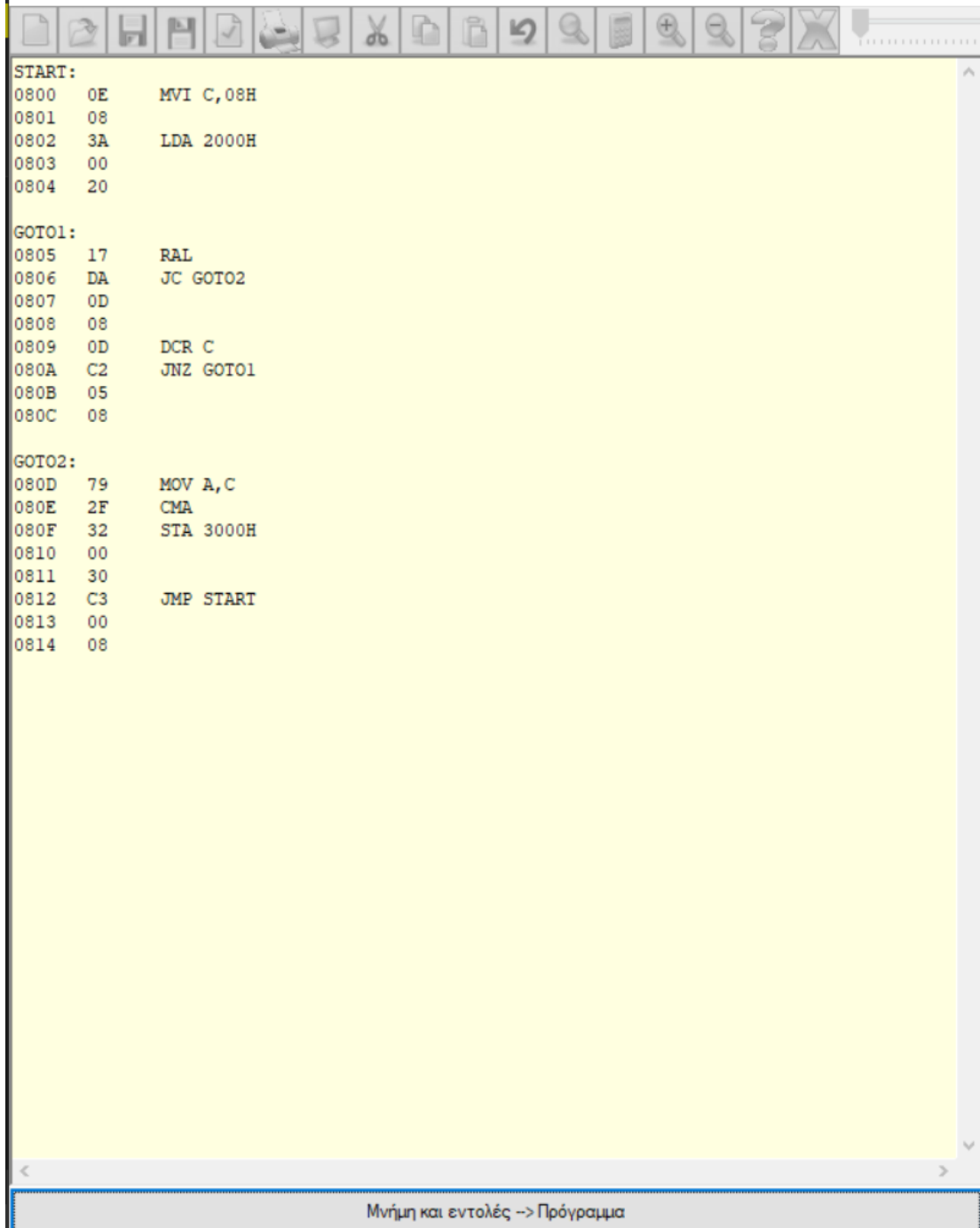


```
START:
    MVI C,08H
    LDA 2000H
GOTO1:
    RAL
    JC GOTO2
    DCR C
    JNZ GOTO1
GOTO2:
    MOV A,C
    CMA
    STA 3000H
    JMP START

    END
```

Πρόγραμμα -> Μνήμη και εντολές

ask1* - TSIK

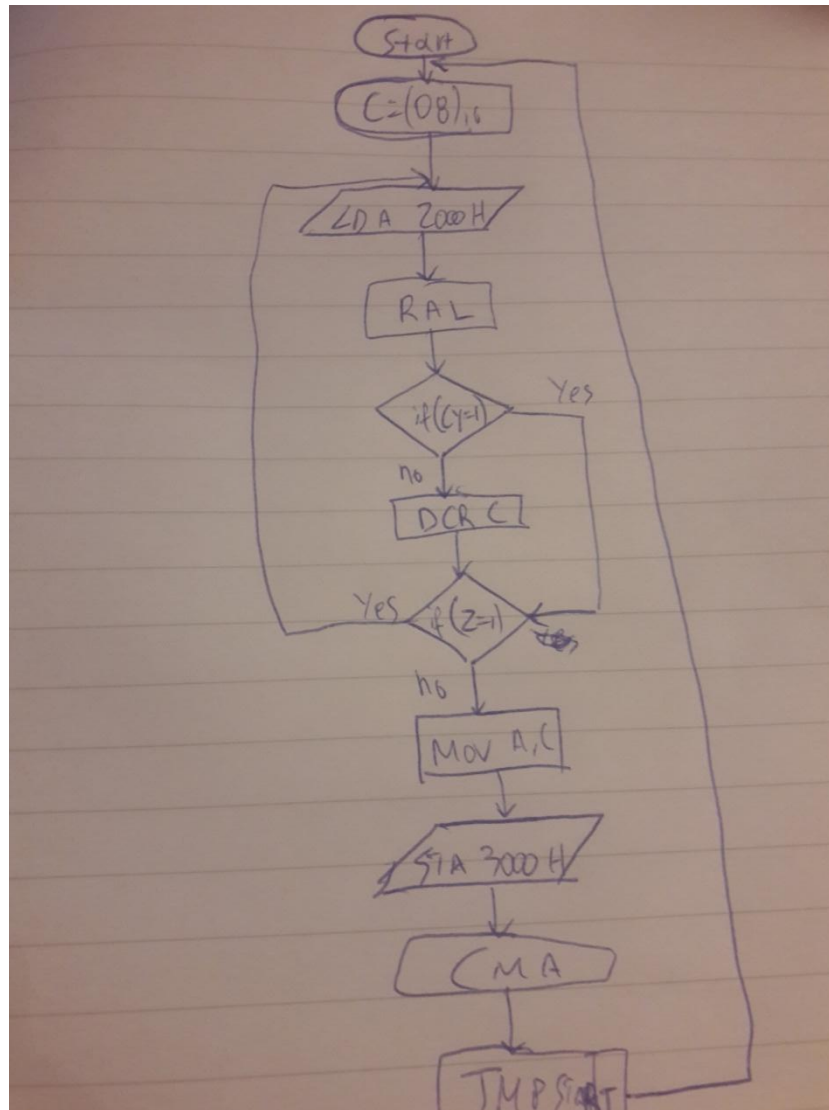


```
START:
0800 0E MVI C,08H
0801 08
0802 3A LDA 2000H
0803 00
0804 20

GOTO1:
0805 17 RAL
0806 DA JC GOTO2
0807 0D
0808 08
0809 0D DCR C
080A C2 JNZ GOTO1
080B 05
080C 08

GOTO2:
080D 79 MOV A,C
080E 2F CMA
080F 32 STA 3000H
0810 00
0811 30
0812 C3 JMP START
0813 00
0814 08
```

Mνήμη και εντολές --> Πρόγραμμα



2^η ΑΣΚΗΣΗ:

```

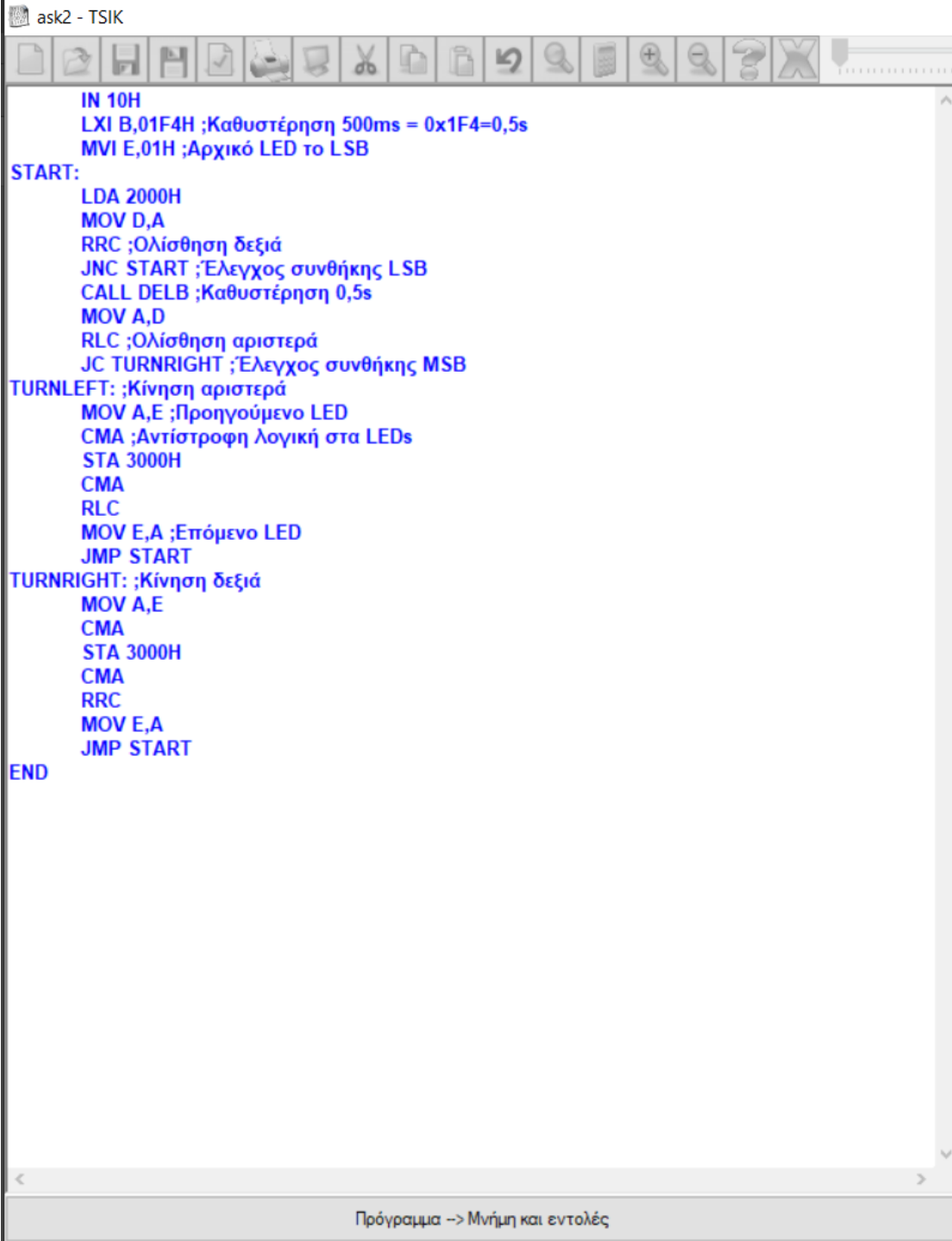
IN 10H
LXI B,01F4H ;Καθυστέρηση 500ms = 0x1F4=0,5s
MVI E,01H ;Αρχικό LED το LSB

START:
LDA 2000H
MOV D,A
RRC ;Ολίσθηση δεξιά
JNC START ;Έλεγχος συνθήκης LSB
CALL DELB ;Καθυστέρηση 0,5s
MOV A,D
RLC ;Ολίσθηση αριστερά
JC TURNRIGHT ;Έλεγχος συνθήκης MSB

TURNLEFT: ;Κίνηση αριστερά
MOV A,E ;Προηγούμενο LED
CMA ;Αντίστροφη λογική στα LEDs
STA 3000H
CMA
RLC
MOV E,A ;Επόμενο LED
JMP START

TURNRIGHT: ;Κίνηση δεξιά
MOV A,E
CMA
STA 3000H
  
```

```
CMA
RRC
MOV E, A
JMP START
END
```



```
ask2 - TSIK
IN 10H
LXI B,01F4H ;Καθυστέρηση 500ms = 0x1F4=0,5s
MVI E,01H ;Αρχικό LED το LSB
START:
LDA 2000H
MOV D,A
RRC ;Ολίσθηση δεξιά
JNC START ;Έλεγχος συνθήκης LSB
CALL DELB ;Καθυστέρηση 0,5s
MOV A,D
RLC ;Ολίσθηση αριστερά
JC TURNRIGHT ;Έλεγχος συνθήκης MSB
TURNLEFT: ;Κίνηση αριστερά
MOV A,E ;Προηγούμενο LED
CMA ;Αντίστροφη λογική στα LEDs
STA 3000H
CMA
RLC
MOV E,A ;Επόμενο LED
JMP START
TURNRIGHT: ;Κίνηση δεξιά
MOV A,E
CMA
STA 3000H
CMA
RRC
MOV E,A
JMP START
END
```

Πρόγραμμα -> Μνήμη και εντολές

3^η ΑΣΚΗΣΗ:

START:

```
MVI B,00H
LDA 2000H
```

EKATO:

```
CPI 64H      ; Compare with 100
JC DECA      ; if smaller than 100 Jump
SUI 64H      ; else subtract 100 and check again
JMP EKATO
```

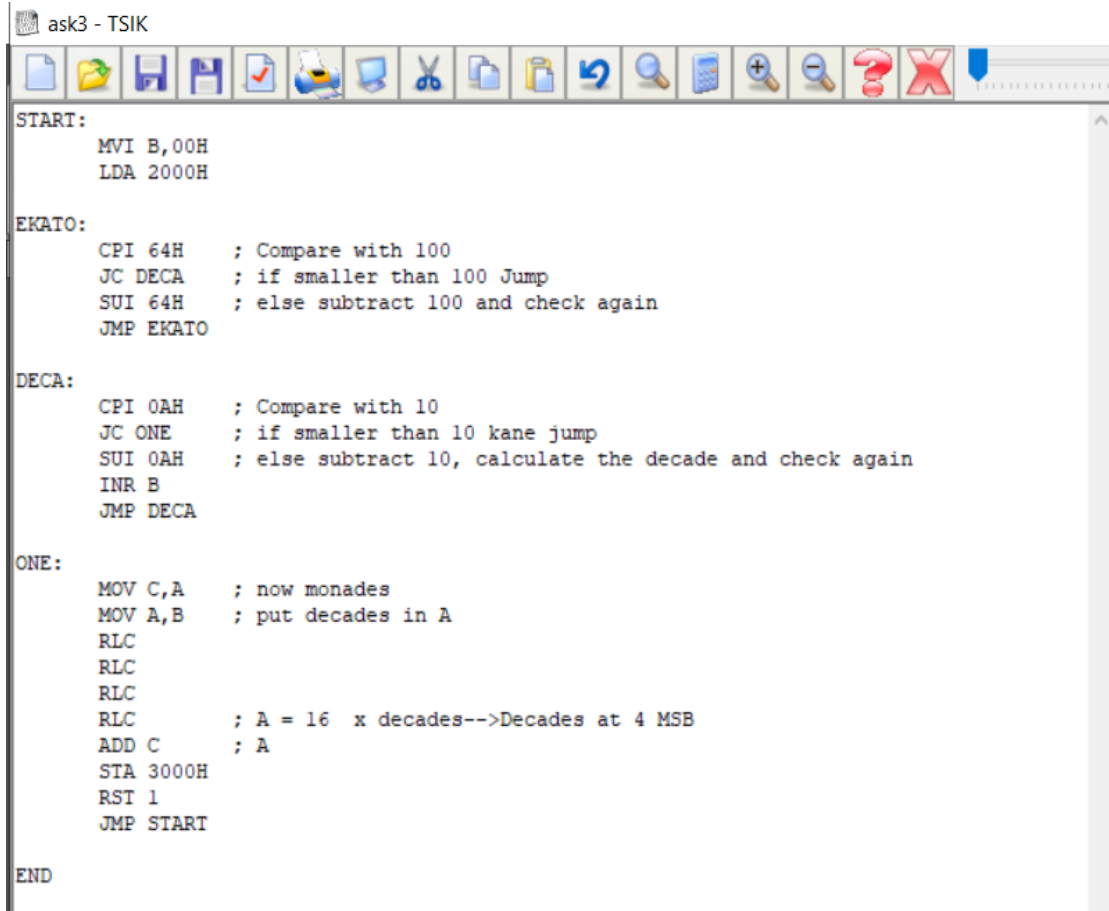
DECA:

```
CPI 0AH      ; Compare with 10
JC ONE       ; if smaller than 10 kane jump
SUI 0AH      ; else subtract 10, calculate the decade and check again
INR B
JMP DECA
```

ONE:

```
MOV C,A      ; now monades
MOV A,B      ; put decades in A
RLC
RLC
RLC
RLC          ; A = 16 x decades-->Decades at 4 MSB
ADD C        ; A
STA 3000H
RST 1
JMP START
```

END



```
ask3 - TSIK
START:
    MVI B,00H
    LDA 2000H

EKATO:
    CPI 64H      ; Compare with 100
    JC DECA      ; if smaller than 100 Jump
    SUI 64H      ; else subtract 100 and check again
    JMP EKATO

DECA:
    CPI 0AH      ; Compare with 10
    JC ONE       ; if smaller than 10 kane jump
    SUI 0AH      ; else subtract 10, calculate the decade and check again
    INR B
    JMP DECA

ONE:
    MOV C,A      ; now monades
    MOV A,B      ; put decades in A
    RLC
    RLC
    RLC
    RLC          ; A = 16 x decades-->Decades at 4 MSB
    ADD C        ; A
    STA 3000H
    RST 1
    JMP START

END
```

ask3* - TSIK

```
START:
0800 06 MVI B,00H
0801 00
0802 3A LDA 2000H
0803 00
0804 20

EKATO:
0805 FE CPI 64H
0806 64
0807 DA JC DECA
0808 0F
0809 08
080A D6 SUI 64H
080B 64
080C C3 JMP EKATO
080D 05
080E 08

DECA:
080F FE CPI 0AH
0810 0A
0811 DA JC ONE
0812 1A
0813 08
0814 D6 SUI 0AH
0815 0A
0816 04 INR B
0817 C3 JMP DECA
0818 0F
0819 08

ONE:
081A 4F MOV C,A
081B 78 MOV A,B
081C 07 RLC
081D 07 RLC
081E 07 RLC
081F 07 RLC
0820 81 ADD C
0821 32 STA 3000H
0822 00
0823 30
0824 CF RST 1
0825 C3 JMP START
0826 00
0827 08
```

Μνήμη και εντολές -> Πρόγραμμα

4^η ΑΣΚΗΣΗ:

Στην 4^η σειρά, κάνουμε την επίλυση χειρόγραφα, τις απαντήσεις τις παραθέτουμε και είναι οι εξής:

$$300000 + 2x = 200000 + 30x$$

ΘΕΩΡΗΤΙΚΕΣ ΑΣΚΗΣΕΙΣ.

▷ 4^η ΑΣΚΗΣΗ

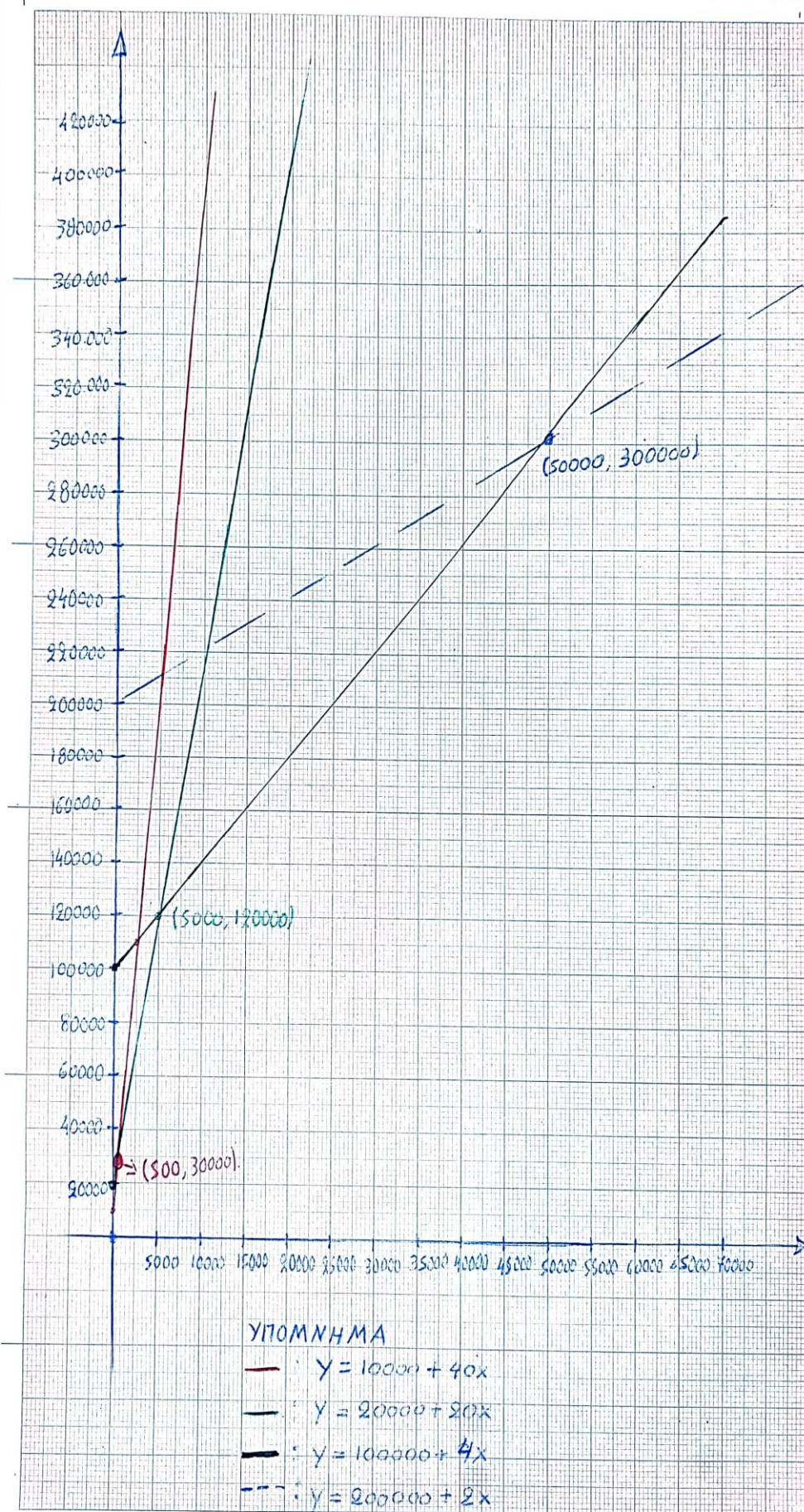
1) Χρήση διακριτών στοιχείων και ολοκληρωμένων μονάδων (I.C) ή ως μικροελεγκτών, περιφερειακών / νητών κ.λ.π τα οποία συναρμολογούνται σε μια σχετικά μεγάλη πλακέτα. Αρχικό κόστος 20.000 € και το κόστος των I.C ανά τεμάχιο θεωρούμε ότι είναι 10€, η κατασκευή της πλακέτας /ε τη συναρμολόγηση της είναι επίσης 10€ /τεμάχιο. Άρα η έκφραση του κόστους γραμμική και έχει /ε $y = 20000 + 10x + 10x = 20000 + 20x$ /ε $x \in \mathbb{N}$, τεμάχιο για x .

2) Χρήση FPGA και μικρού αριθμού περιφερειακών τοποθετημένων σε μια πλακέτα. Αρχικό κόστος οι 10.000 € και επίσης: κόστος /τεμάχιο των I.C: 30€ και κόστος πλακέτας ανά τεμάχιο και συναρμολόγησης: 10€
Άρα, η έκφραση του κόστους είναι $y = 10000 + 30x + 10x = 10000 + 40x$.

3) Σχεδίαση ειδικού SoC-1 /ε μια μικρή πλακέτα. Αρχικό κόστος σχεδίασης: 100.000 € κόστος ανά τεμάχιο των I.C: 2€ κόστος πλακέτας και συναρμολόγησης ανά τεμάχιο 2€, άρα η γραμμική έκφραση του κόστους, αποτελεί η: $y = 100.000 + 2x + 2x \Rightarrow y = 100.000 + 4x$.

4) Σχεδίαση ειδικού SoC-2 /ε μια πολύ μικρή πλακέτα. Αρχικό κόστος: 200.000 €, κόστος ανά τεμάχιο των I.C: 1€ και κόστος πλακέτας και συναρμολόγησης ανά τεμάχιο: 1€. Άρα, η έκφραση του κόστους, είναι $y = 200.000 + x + x = 200000 + 2x$.

Παρακάτω φαίνονται οι 4 ζητούμενες καρτύτες, σχεδιασμένες στο χέρι και /ε μιλιμετρικό χαρτί:



Υπολογίζοντας τα γυφία τoής των καρτών,
βρίσκω ποια τεχνολογία συμφέρει για κάθε
διάστημα.

Τεράχια:

- 0 έως 500 : η τεχνολογία 2
- 500 έως 5000 : η τεχνολογία 1
- 5000 έως 50000 : η τεχνολογία 3
- 50000 ή περισσότερα τεράχια: η τεχνολογία 4

Εάν τώρα θέλουμε να εμφανιστεί η επιλογή
της 1^{ης} τεχνολογίας, θα πρέπει κάποια
τεχνολογία να αλλάξει είτε το αρχικό κόστος
είτε ~~το αρχικό~~ η τιμή ανά τεράχιο κά-
ποιου υλικού. Εδώ, αλλάζουμε την τιμή κόστους
ανά τεράχιο των I.C των τεχνολογία
των FPGAs (αντί των 30€).

Και για να εμφανιστεί η επιλογή της 1^{ης}
τεχνολογίας, θέλουμε στα 5000 τεράχια (γυφίο
τοής της πράσινης και μαύρης καρτών, η
2^η τεχνολογία να είναι πιο φθηνή από την
1^η. Έτσι, για να απορρίψουμε ασυφικτι των
1^η τεχνολογία, όποτε η καρτν θα είναι
τώρα $y = 10000 + kx + 10x$, να έχουμε:

$k = 5000$:

$$10000 + k \cdot 5000 + 10 \cdot 50000 = 100000 + 4 \cdot 5000 \Rightarrow$$

$$10000 + 5000k + 50000 = 120000 \Rightarrow$$

$$5000k = 60000 \Rightarrow \boxed{k = 12}$$

Άρα, πρέπει η τιμή των I.C των τεχνολογία
των FPGAs να είναι το πολύ 12 ευρώ. Πράγματι
για 5000 τεράχια, το κόστος της 2^{ης} τεχνολογίας
θα είναι 120.000 ευρώ, που είναι ίσο με το
κόστος 1^{ης} και 3^{ης} τεχνολογίας για 5000 τεράχια.

5^η ΑΣΚΗΣΗ:

(i) Δομική περιγραφή των ζητούμενων συναρτήσεων σε επίπεδο πυλών:

```
module exc5a(A, B, C, D, E, F1, F2, F3, F4);  
  
    output F1, F2, F3, F4;  
  
    input A, B, C, D, E;  
  
    wire Anot, Bnot, Cnot, Dnot, w1, w2, w3, w4;  
  
    not  
        (Anot, A),  
        (Bnot, B),  
        (Cnot, C),  
        (Dnot, D);  
  
    and(w1, B, C);  
  
    or(w2, w1, D);  
  
    and(w3, w2, A);  
  
    and(w4, Bnot, Cnot, D);  
  
    or(F1, w3, w4);  
  
  
    wire m0, m2, m3, m5, m7, m8, m10, m11, m14, m15;  
  
    and  
        (m0, Anot, Bnot, Cnot, Dnot),  
        (m2, Anot, Bnot, C, Dnot),  
        (m3, Anot, Bnot, C, D),  
        (m5, Anot, B, Cnot, D),  
        (m7, Anot, B, C, D),  
        (m9, A, Bnot, Cnot, D),  
        (m10, A, Bnot, C, Dnot),
```

```

(m11, A, Bnot, C, D),
(m13, A, B, Cnot, D),
(m14, A, B, C, Dnot);
or(F2, m0, m2, m3, m5, m7, m9, m10, m11, m13, m14);

```

```

wire w5, w6, w7, w8, w9;

and(w5, A, B, C);

and(w6, A, D);

and(w7, B,C,D);

and(w8, B, D, E);

and(w9, C, D, E);

or(F3, w5 ,w6 , w7, w8, w9);

```

```

wire ww9, w10, w11, w12;

and(ww9, C, D);

or(w10, ww9, B, E);

and(w11, w10, A);

and(w12,B, C, D, E);

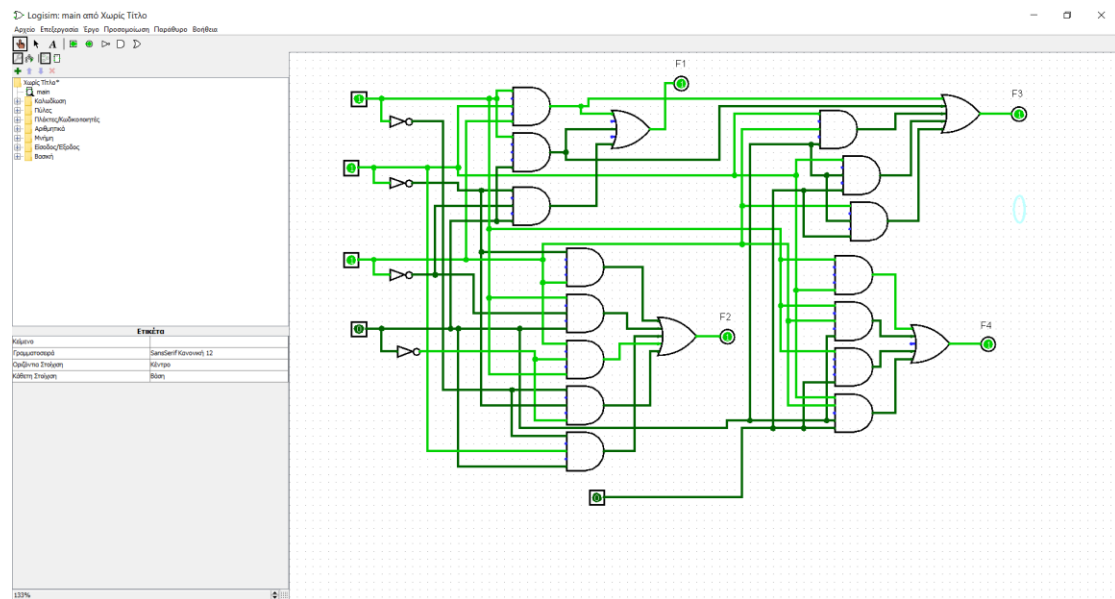
or(F4, w11, w12);

```

endmodule

Το αποτέλεσμα της εκτέλεσης αυτής, είναι να προκύψει ένα «σύνθετο» κύκλωμα που έχει 5 εισόδους (A,B,C,D,E) και 4 εξόδους, τις F1,F2,F3 και F4 και το αποτέλεσμα της εκτέλεσης αυτής (το ίδιο ακριβώς βγαίνει στο στο ii) ερώτημα, στο επόμενο δηλαδή) το αποτυπώνω με τη βοήθεια του logisim σε αυτό το ερώτημα, με τον δε

προσομοιωτή στο link <http://digitaljs.tilk.eu/#> και λαμβάνουμε ότι:



Και δοκιμάζω το κύκλωμα με αλλαγές στις εισόδους A,B,C,D,E και διαπιστώνω ορθή λειτουργία του κυκλώματος

(ii) Οι ίδιες συναρτήσεις, με μοντελοποίηση ροής δεδομένων:

```
module exc5b(A, B, C, D, E, F1, F2, F3, F4);
```

```
    output F1, F2, F3, F4;
```

```
    input A, B, C, D, E;
```

```
    assign
```

```
        F1 = (A&((B&C)|D))|(~B&~C&D),
```

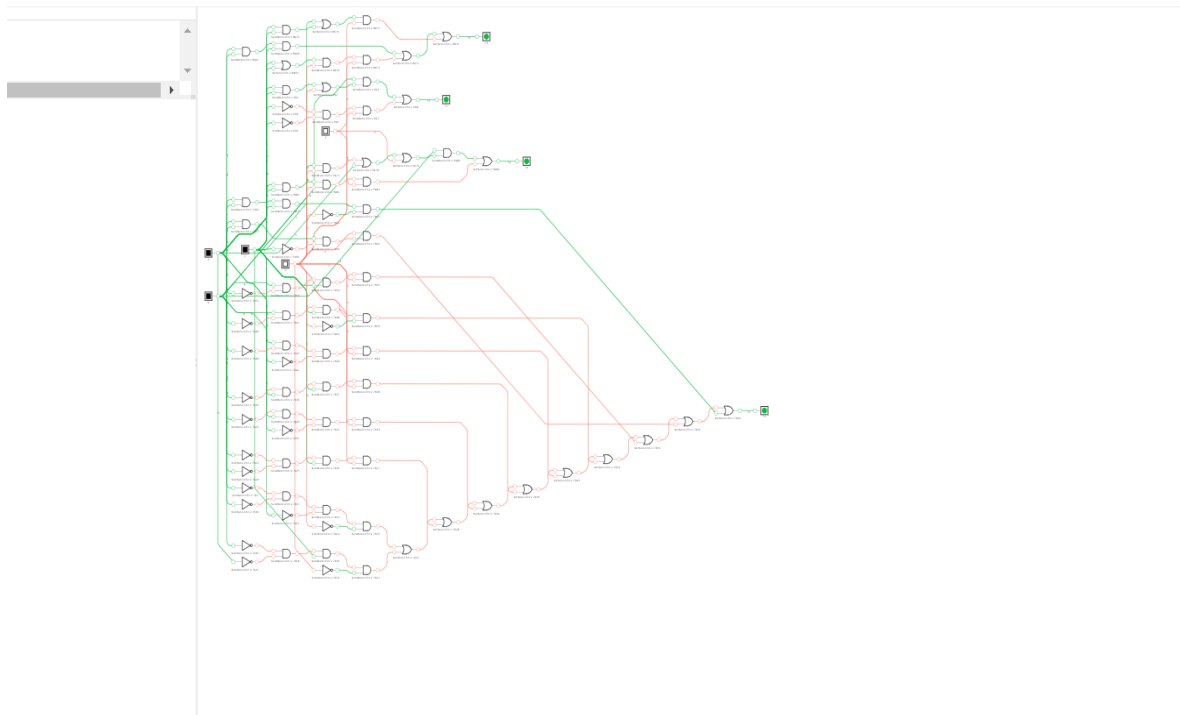
```
        F2 = (~A&~B&~C&~D)|(~A&~B&C&~D)|(~A&~B&C&D)|(~A&B&~C&D)|
              |(~A&B&C&D)|(A&~B&~C&D)|(A&~B&C&~D)|(A&~B&C&D)|(A&B&~
              C&D)|(A&B&C&~D),
```

```
        F3 = (A&B&C)|((B|C)&D&E)|((A|(B&C))&D),
```

```
        F4 = (A&((C&D)|B|E))|(B&C&D&E);
```

```
endmodule
```

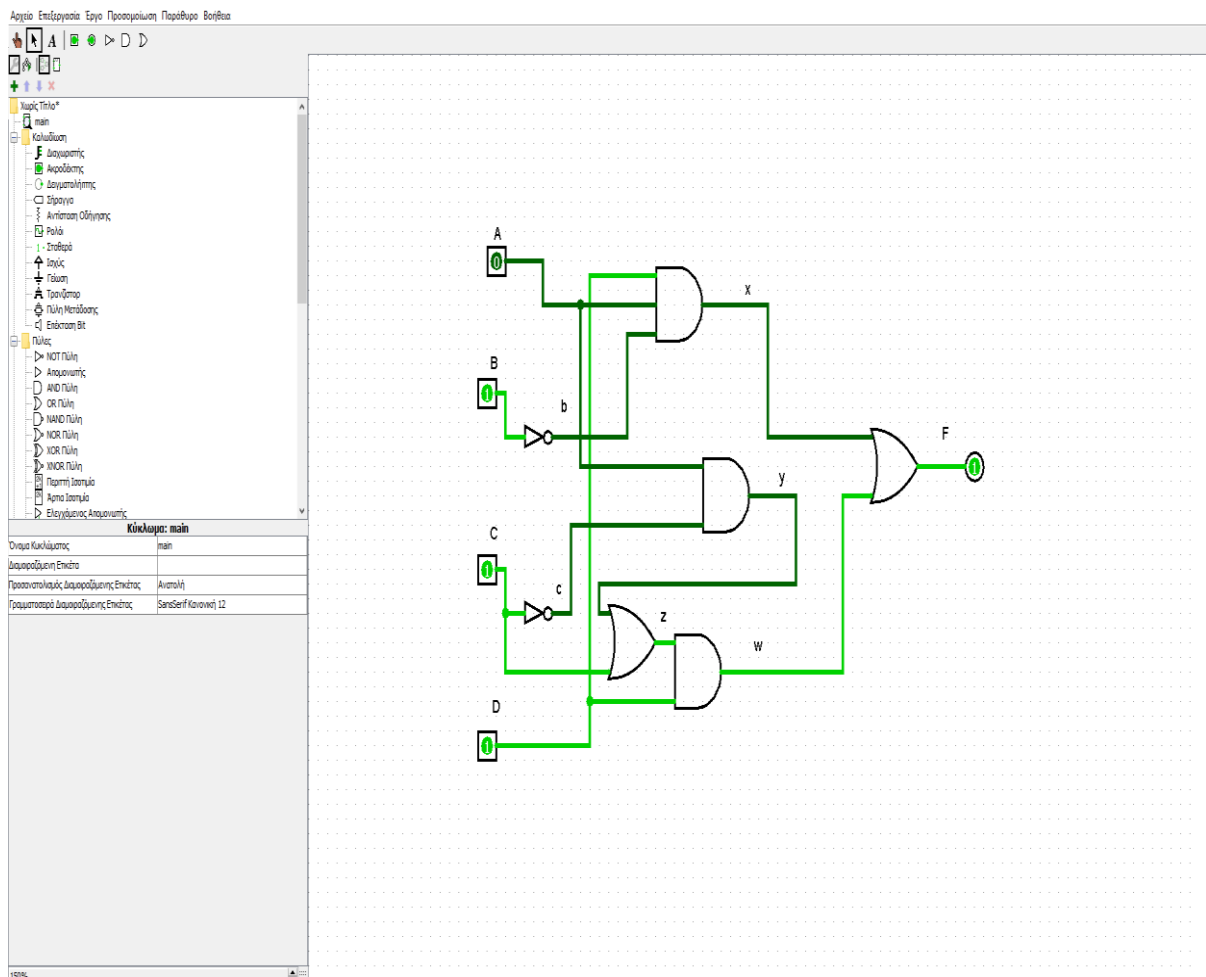
Το αποτέλεσμα της εκτέλεσης αυτής, είναι να προκύψει ένα «σύνθετο» κύκλωμα που έχει 5 εισόδους (A,B,C,D,E) και 4 εξόδους, τις F1,F2,F3 και F4 και το αποτέλεσμα της εκτέλεσης αυτής αποτελεί το εξής:



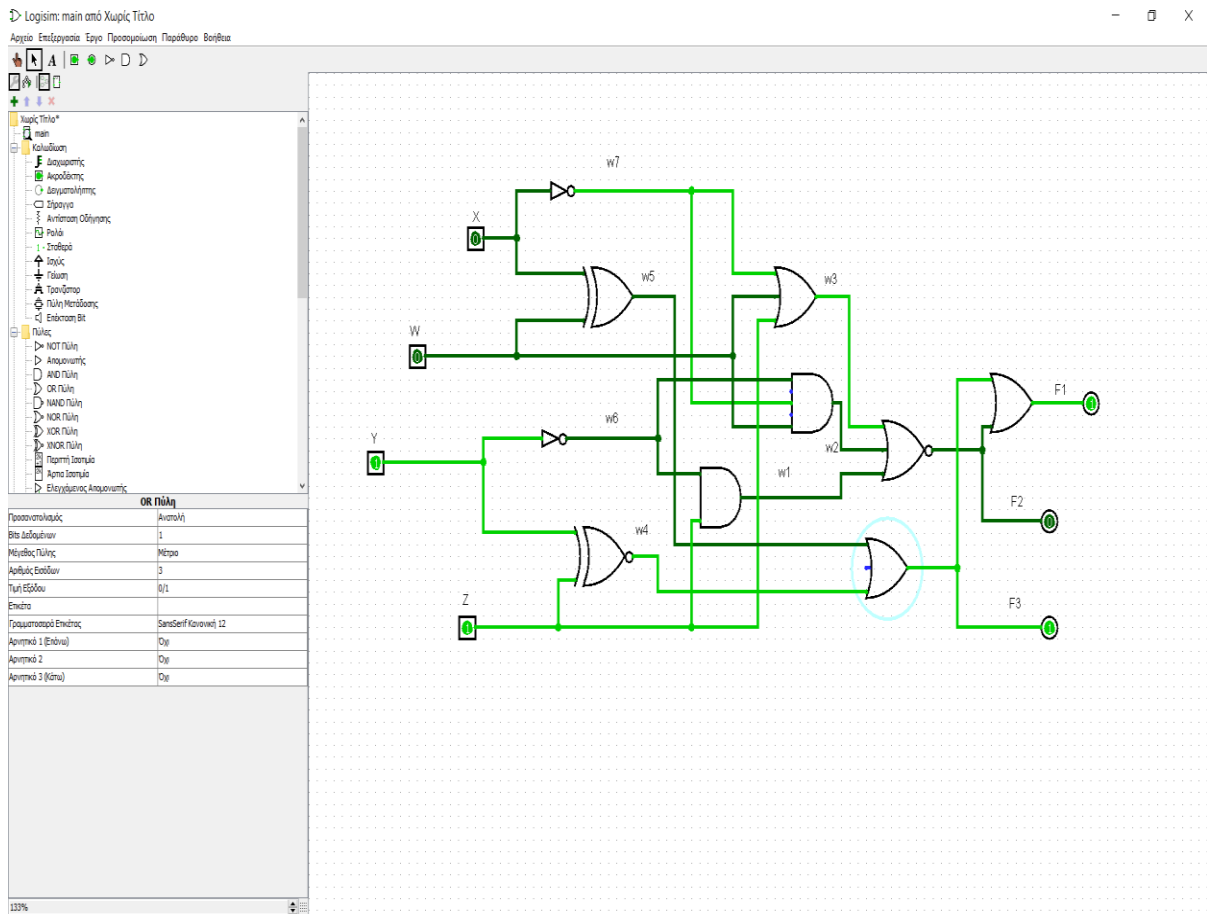
το οποίο υλοποιεί τα F1,F2,F3,F4 και αποτελεί το παραπάνω (φαίνεται πιο καθαρά με zoom, με μεγέθυνση). Επίσης ελέγχω μέσω των εισόδων και διαπιστώνω ορθή λειτουργία του κυκλώματος, την επιδιωκόμενη.

6^η ΑΣΚΗΣΗ

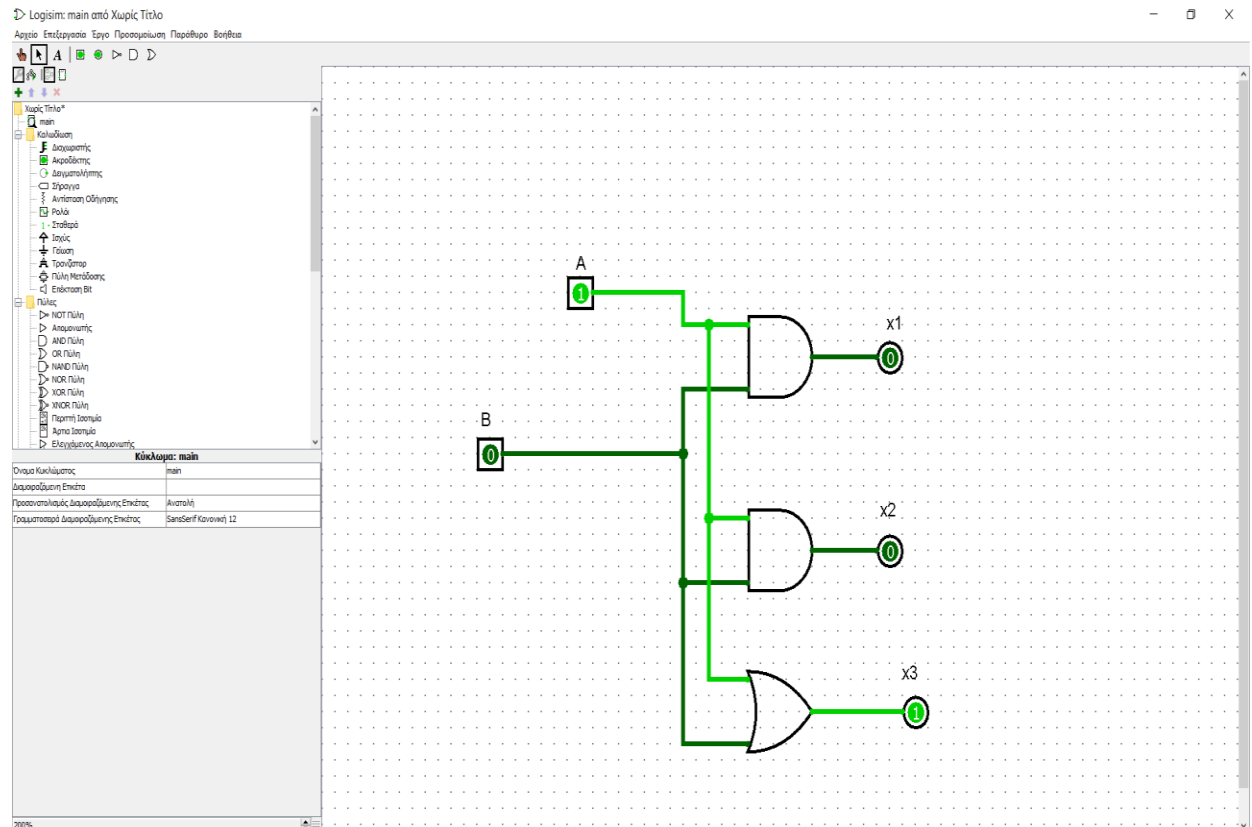
(i) (a) Παρακάτω φαίνεται το ζητούμενο λογικό διάγραμμα:



(b) Παρακάτω φαίνεται το ζητούμενο λογικό διάγραμμα:



(c) Παρακάτω φαίνεται το ζητούμενο λογικό διάγραμμα:



Τα ίδια αποτελέσματα βγάζω και επιβεβαιώνω στο link <http://digitaljs.tilk.eu/#>

(ii) Περιγραφή του ζητούμενου αθροιστή-αφαιρέτη σε επίπεδο πυλών:

```
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 bit4_adder_subtractor(output [3: 0] Sum, output C4, input [3: 0] A, B, input  
C0);
```

```
    wire C1, C2, C3;
```

```
    wire [3: 0] M;
```

```
    xor
```

```
        (M[0], B[0], C0),
```

```
        (M[1], B[1], C0),
```

```
        (M[2], B[2], C0),
```

```
        (M[3], B[3], C0);
```

```
    full_adder
```

```
        FA0(Sum[0], C1, A[0], M[0], C0),
```

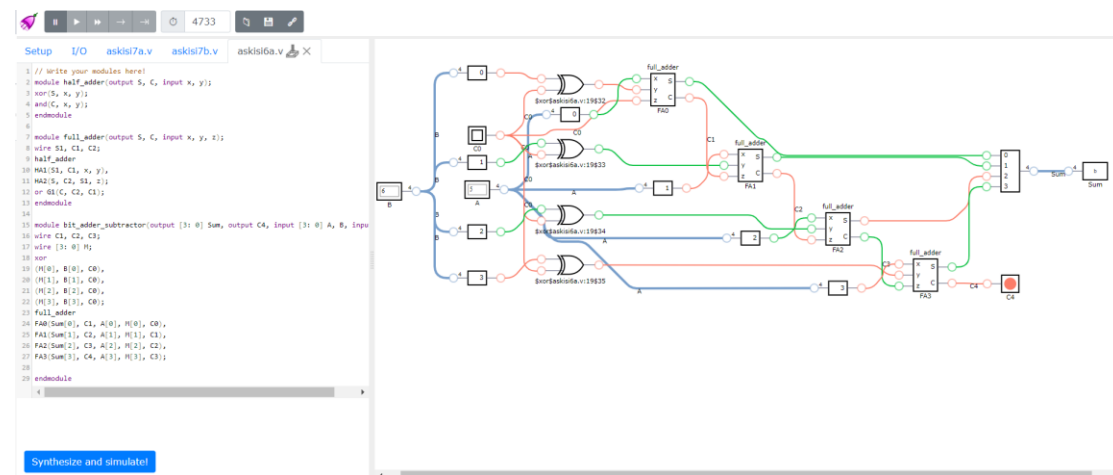
```
FA1(Sum[1], C2, A[1], M[1], C1),
```

```
FA2(Sum[2], C3, A[2], M[2], C2),
```

```
FA3(Sum[3], C4, A[3], M[3], C3);
```

```
endmodule
```

Το αποτέλεσμα της εκτέλεσης αυτής, είναι να προκύψει ένα «σύνθετο» κύκλωμα που έχει 3 εισόδους (4bit-αριθμός A, 4bit-αριθμός B, κρατούμενο C0) και 2 εξόδους, το κρατούμενο C4 και το Sum, που είναι 4 bit. Και είναι το εξής, που προκύπτει εικονικά:



ΟΠΟΙΟΔΗΠΟΤΕ ΚΑΙ ΝΑ ΕΙΝΑΙ ΤΑ Α,Β ΠΡΑΓΜΑΤΙ ΔΙΝΟΥΝ ΤΟ ΑΘΡΟΙΣΜΑ Α+Β . ΕΝΔΕΙΚΤΙΚΑ ΕΒΑΛΑ 5 ΚΑΙ 6, ΟΠΟΤΕ ΚΑΙ ΤΟ ΑΠΟΤΕΛΕΣΜΑ ΠΟΥ ΛΑΜΒΑΝΟΥΜΕ ΕΙΝΑΙ b, ΔΗΛΑΔΗ 11

Και δοκιμάζω το κύκλωμα με αλλαγές στις εισόδους A,B και στο C0. Συνεχώς διαπιστώνω ότι όταν βάζω αριθμό A και B, των οποίων το άθροισμα αποτυπώνεται με χρήση 4 bit, τότε λαμβάνω C0 = 0 και το sum πράγματι το άθροισμα των A και B. Εάν από την άλλη βάλω άθροισμα παραπάνω από 16 (τότε χρειάζονται τουλάχιστον 5 bits για την αναπαράσταση του αριθμού) ,τότε λαμβάνω είτε C4 = 1, είτε 0, δήλωση υπερχειρίσης (καλής ή κακής εξαρτάται από τα C3,C4). Σε κάθε συνδυασμό και περίπτωση νούμερων A και B, διαπιστώνω ότι πράγματι επιτελώ σωστά τη λειτουργία και λαμβάνω ορθά αποτελέσματα.

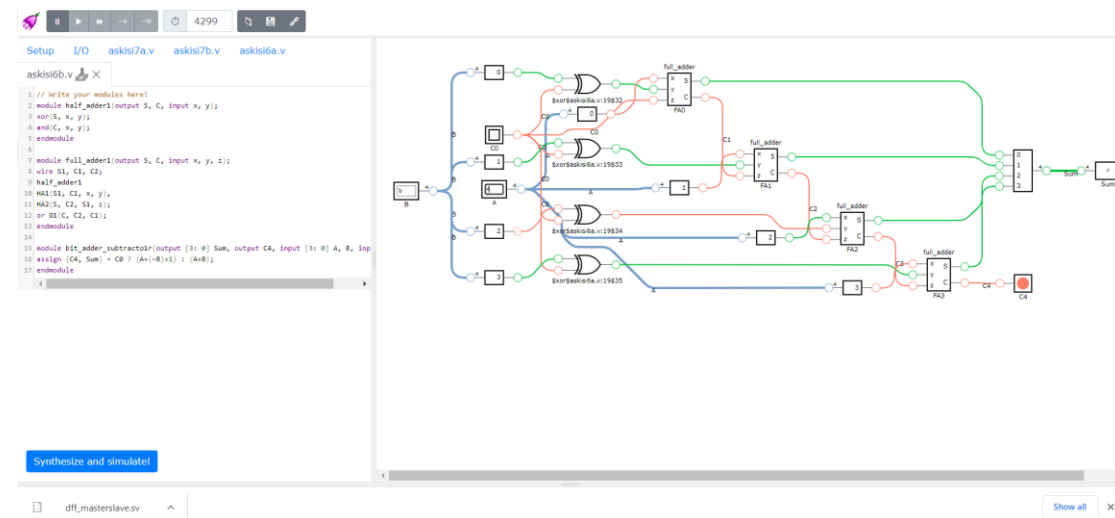
(iii) Ο ίδιος αθροιστής-αφαιρέτης με περιγραφή ροής δεδομένων:

```
module 4_bit_adder_subtractor(output [3:0] Sum, output C4, input [3:0] A, B, input C0);
```

```
assign {C4, Sum} = C0 ? (A+(~B)+1) : (A+B);
```

```
endmodule
```

Όμοια με πριν, λαμβάνω το ίδιο αποτέλεσμα, αλλά εδώ τώρα γίνεται με χρήση περιγραφής δεδομένων. Χρησιμοποιώ πάλι το link <http://digitaljs.tilk.eu/#>, τρέχω τον κώδικα όπως πριν και παράγω κύκλωμα. Το αποτέλεσμα είναι το ίδιο και είναι το εξής: (φαίνεται με zoom λίγο αναλυτικότερα)



ΟΜΟΙΩΣ ΚΑΙ ΕΔΩ, ΕΠΙΤΕΛΕΙΤΑΙ Η ΙΔΙΑ ΛΕΙΤΟΥΡΓΙΑ ΟΡΘΑ

7^η ΑΣΚΗΣΗ

ι) Η υλοποίηση του πρώτου μοντέλου είναι η παρακάτω και φαίνεται η απλοποίηση της εξόδου y: (00: a, 01: b, 10: c, 11: d)

AB\χ	0	1
00	1	0
01	1	0
11	0	1
10	1	0

$$y = A'x' + ABx + AB'x'$$

module Mealy(y, x, clock, reset);

output 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)

if (reset == 0) state <= a;

else case (state)

a: if ($\sim x$) state <= d; else state <= a;

b: if ($\sim x$) state <= c; else state <= a;

c: if ($\sim x$) state <= d; else state <= b;

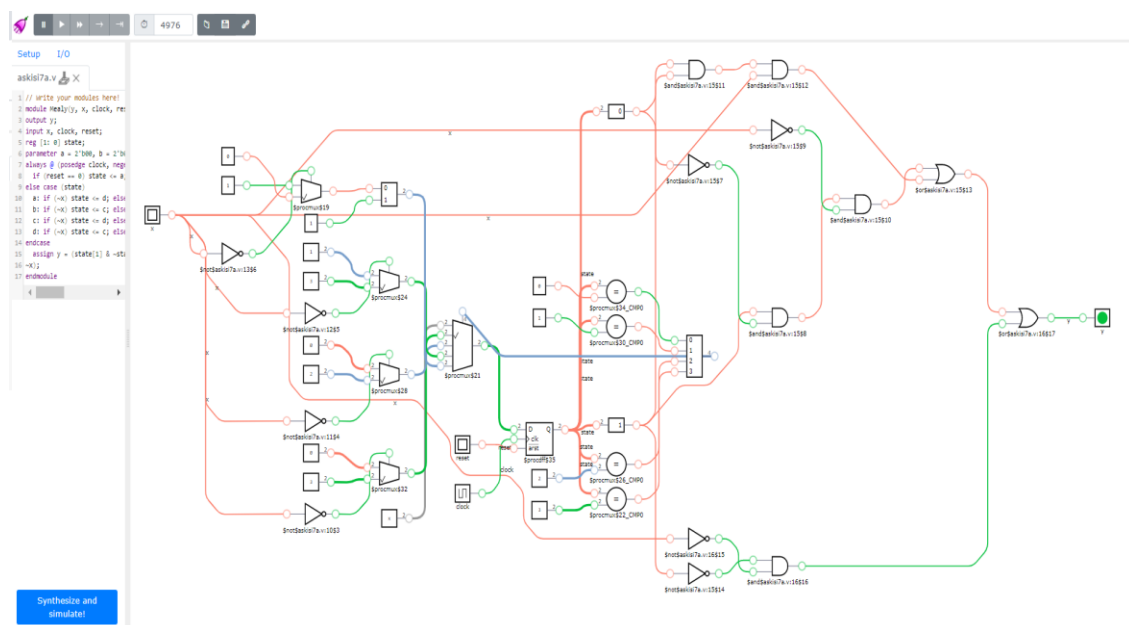
d: if ($\sim x$) state <= c; else state <= d;

endcase

assign y = (state[1] & \sim state[0] & $\sim x$) | (state[1] & state[0] & x) | (\sim state[1] & $\sim x$);

endmodule

Πραγματοποίησα την εκτέλεση με την χρήση του προσομοιωτή online, με link το <http://digitaljs.tilk.eu/#> και το αποτέλεσμα του κυκλώματος που έβγαλα είναι το κάτωθι:



Πραγματοποιώ προσομοίωση πάνω στο κύκλωμα και αλλάζω τις αρχικές καταστάσεις. Οπότε και πράγματι, διαπιστώνω την ορθή λειτουργία του κυκλώματος

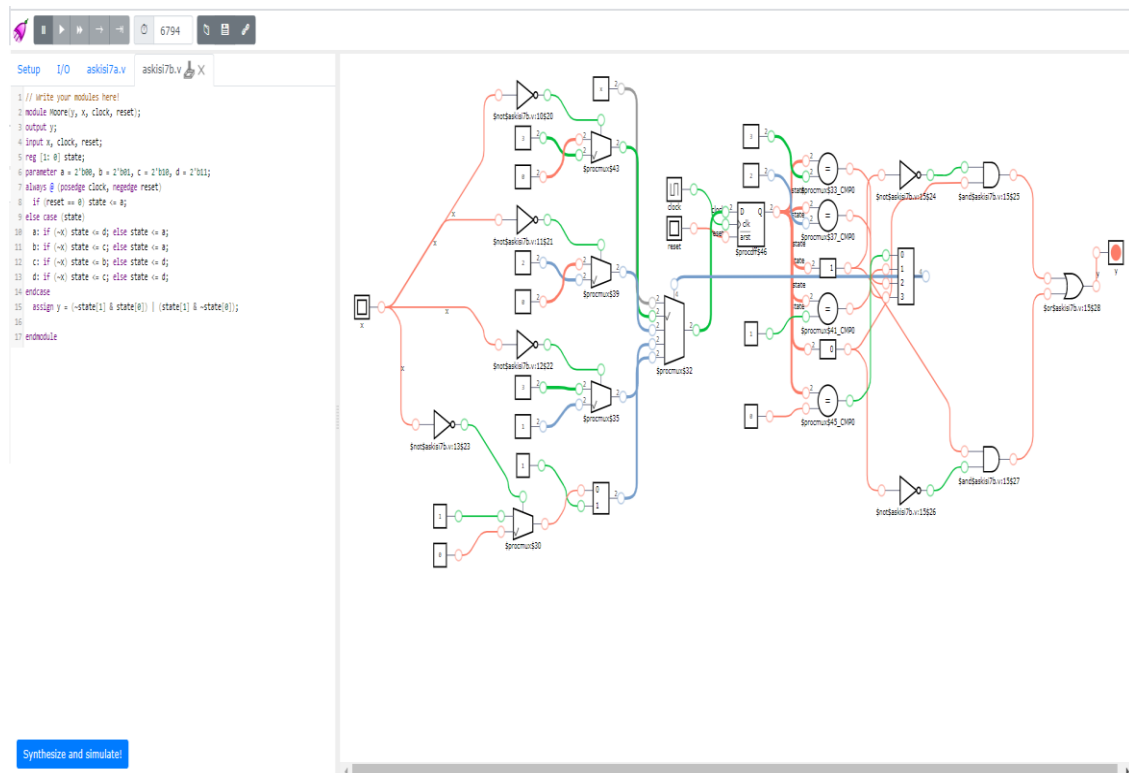
(ii) Παρακάτω φαίνεται η απλοποίηση της εξόδου

A\B	0	1
0	0	1
1	1	0

$$y = A'B + AB' = A \text{ XOR } B$$

```
module Moore(y, x, clock, reset);  
  
    output 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)  
        if (reset == 0) state <= a;  
        else case (state)  
            a: if (~x) state <= d; else state <= a;  
            b: if (~x) state <= c; else state <= a;  
            c: if (~x) state <= b; else state <= d;  
            d: if (~x) state <= c; else state <= d;  
        endcase  
  
    assign y = state[1] ^ state[0];  
  
endmodule
```

Πραγματοποίησα και εδώ την εκτέλεση με την χρήση του προσομοιωτή online, με link το <http://digitaljs.tilk.eu/#> και το αποτέλεσμα του κυκλώματος που έβγαλα είναι το κάτωθι:



Πραγματοποιώ και εδώ προσομοίωση πάνω στο κύκλωμα και αλλάζω τις αρχικές καταστάσεις. Οπότε και διαπιστώνω την ορθή λειτουργία του κυκλώματος.