ΕΘΝΙΚΌ ΜΕΤΣΌΒΙΟ ΠΟΛΥΤΕΧΝΕΊΟ

ΣΧΟΛΉ ΗΛΕΚΤΡΟΛΌΓΩΝ ΜΗΧΑΝΙΚΏΝ ΚΑΙ ΜΗΧΑΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΥΣΤΉΜΑΤΑ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ

(2019-2020)

1η ΟΜΆΔΑ ΑΣΚΗΣΕΩΝ

Ονοματεπώνυμο:

- Χρήστος Τσούφης – 03117176
- Ιάσων Χατζηθεοδώρου – 03117089

1η Άσκηση

Με βάση τους πίνακες από το αρχείο mLab, προκύπτει ο παρακάτω κώδικας:

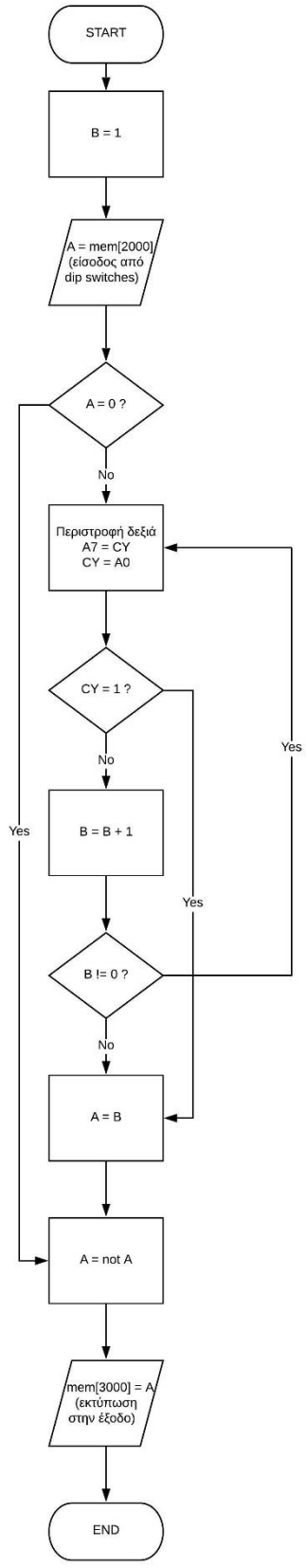
```
START:
MVI B,01H           ;B = 1
LDA 2000H           ;A = memory[2000]
CPI 00H             ;Σύγκριση του A με το 0
JZ L_2              ;Αν το A είναι μηδέν κάνε άλμα στο L_2
WHILE_LOOP:
RAR                 ;Δεξιά περιστροφή του A, CY
JC L_1              ;Αν το CY είναι 1 κάνε άλμα στο L_1
INR B               ;B = B + 1
JNZ WHILE_LOOP       ;Επανάλαβε αν το B είναι μη μηδενικό
L_1:
MOV A,B             ;A = B
L_2:
CMA                 ;A = not A
STA 3000H           ;memory[3000] = A
RST 1
END
```

Στη συνέχεια, χρησιμοποιώντας το μLab επαληθεύεται ότι η μετάφραση έγινε σωστά.

```
START:                          START:
0800   06     MVI B,01H        0800   06     MVI B,01H
0801   01                      0801   01
0802   3A     LDA 2000H        0802   3A     LDA 2000H
0803   00                      0803   00
0804   20                      0804   20
0805   FE     CPI 00H          0805   FE     CPI 00H
0806   00                      0806   00
0807   CA     JZ L_2           0807   CA     JZ L_2
0808   13                      0808   13
0809   08                      0809   08

WHILE_LOOP:                     WHILE_LOOP:
080A   1F     RAR              080A   1F     RAR
080B   DA     JC L_1           080B   DA     JC L_1
080C   12                      080C   12
080D   08                      080D   08
080E   04     INR B            080E   04     INR B
080F   C2     JNZ WHILE_LOOP   080F   C2     JNZ WHILE_LOOP
0810   0A                      0810   0A
0811   08                      0811   08

L_1:                            L_1:
0812   78     MOV A,B          0812   78     MOV A,B

L_2:                            L_2:
0813   2F     CMA              0813   2F     CMA
0814   32     STA 3000H        0814   32     STA 3000H
0815   00                      0815   00
0816   30                      0816   30
0817   CF     RST 1            0817   CF     RST 1
```

Για να επαναλαμβάνεται χωρίς τέλος αρκεί να αντικαταστήσουμε την εντολή RST 1 με την JMP START.

Παρακάτω φαίνεται το διάγραμμα ροής.



2η Άσκηση

Πηγαίος κώδικας:

```
START:
IN 10H
MVI A,00H                ;Temporary register that takes input and shows which LED will
be
                         ;turned ON
MVI B,00H                ;Register B saves the position of the last LED
MVI C,00H                ;Register C: 00->Increasing, 01H->Decreasing

MAIN_LOOP:
LDA 2000H                ;A = mem[2000H]
RAR
RAR                      ;CY has the second LSB
JC SECOND_LSB_ON

SECOND_LSB_OFF:
RAL
JC LSB_ON

LSB_OFF:
MOV A,B
CPI 00H                  ;Is it the first time?
JZ FIRST_TIME
CPI 80H                  ;Is LED 8 ON? (2^7 = 128 = 80H)
JNZ INCREASE        ;if it isn't then increase B
MVI B,01H
MVI A,01H                ;if it is then the next to open is LED 1
JMP PRINT_CURRENT

LSB_ON:
MOV A,B
CPI 00H                  ;Is it the first time?
JZ FIRST_TIME
CPI 01H                  ;Is LED 1 ON?
JZ INCREASE              ;If it is then start increasing
CPI 80H                  ;Is LED 8 ON? (2^7 = 128 = 80H)
JZ DECREASE              ;If it is start decreasing
MOV A,C
CPI 00H                  ;LED 1 and 8 are OFF. Are we increasing or decreasing?
JZ INCREASE
JMP DECREASE

FIRST_TIME:
MVI B,01H
MOV A,B
JMP PRINT_CURRENT

INCREASE:
MVI C,00H
MOV A,B
RLC
MOV B,A
JMP PRINT_CURRENT

DECREASE:
MVI C,01H
MOV A,B
RRC
MOV B,A
JMP PRINT_CURRENT

SECOND_LSB_ON:
MVI A,00H
JMP PRINT_CURRENT
```

```
PRINT_CURRENT:
CMA
STA 3000H
CALL DELB
JMP MAIN_LOOP
END
```

3η Άσκηση

Πηγαίος κώδικας:

```
START:
     LDA 2000H      ;read input
     CPI 63H        ;compare with 99=63H
     JNC MODULO     ;if A > 99 go to modulo
     MVI B,FFH      ;else, initialize B and continue
DECA:
     INR B          ;after H=FFH, here H=0
     SUI 0AH        ;continuous subtraction with 10=0AH
     JNC DECA       ;continue until A<0
     ADI 0AH        ;here A<0, add 10
     MOV M,A        ;M is used as a temporary variable to keep units
     MOV A,B        ;transfer units of B to A
     RLC            ;tens must be the 4 MSB's
     RLC
     RLC
     RLC
     ADD M          ;αdd M to 4 LSB's
     CMA            ;fill out A because outputs have negative logic
     LXI B,01F4H    ;set B-A for 500sec delay with DELB
     STA 3000H      ;save & print to Leds
     CALL DELB
     JMP START
MODULO:
     CPI C8H         ;compare A with 200
     JNC LESS_THAN_200
     SUI C8H        ;if A >= 200, A = A-200
     JMP START      ;go to START
LESS_THAN_200:
     SUI 64H        ;if A < 200, A = A-100
     JMP START      ;go to START
END
```

4η Άσκηση

Το κόστος κάθε τεχνολογίας εξαρτάται από το πλήθος των τεμαχίων που παράγονται. Η συνάρτηση κόστους K = K(x) για κάθε μία τεχνολογία, από τα δεδομένα λοιπόν της άσκησης, είναι:

Διακριτά Στοιχεία (ΔΣ) : $K_{\Delta\Sigma}(x) = 20.000 + (15 + 15) * x = 20.000 + 40x$, $x \geq 0$

FPGA : $K_{FPGA}(x) = 10.000 + (60 + 10) * x = 10.000 + 70x$, $x \geq 0$

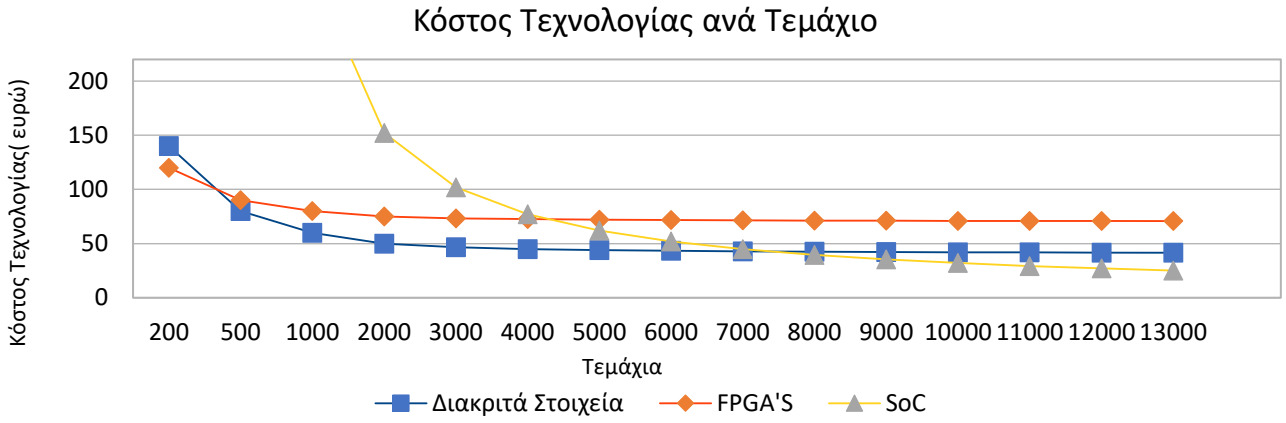
SoC : $K_{SoC}(x) = 300.000 + (1 + 1) * x = 300.000 + 2x$, $x \geq 0$

Οι αντίστοιχες συναρτήσεις κόστους ανά τεμάχιο είναι:

Διακριτά Στοιχεία (ΔΣ) : $K_{\Delta\Sigma}(x) = \frac{20.000}{x} + 40$, $x > 0$

FPGA : $K_{FPGA}(x) = \frac{10.000}{x} + 70$, $x > 0$

SoC : $K_{SoC}(x) = \frac{300.000}{x} + 2x$, $x > 0$



Παρατήρηση: Καθώς $x \to \infty$, το κόστος για κάθε τεχνολογία φθίνει σε μια σταθερά που ισούται με το κόστος παραγωγής ενός τεμαχίου. Επιπλέον, για 1 – 400 τεμάχια, υπερισχύουν τα FPGA's , για 400 – 7.000 τεμάχια, υπερισχύουν τα Διακριτά Στοιχεία και τέλος, για 7.000 κι έπειτα, η τεχνολογία SoC.

Στη γενική περίπτωση, αν το κόστος του IC της τεχνολογίας FPGA είναι μια μεταβλητή ποσότητα k τότε η αντίστοιχη συνάρτηση κόστους μπορεί να γραφεί στην μορφή:

$K_{FPGA}(x) = 10.000 + (k + 10) * x$

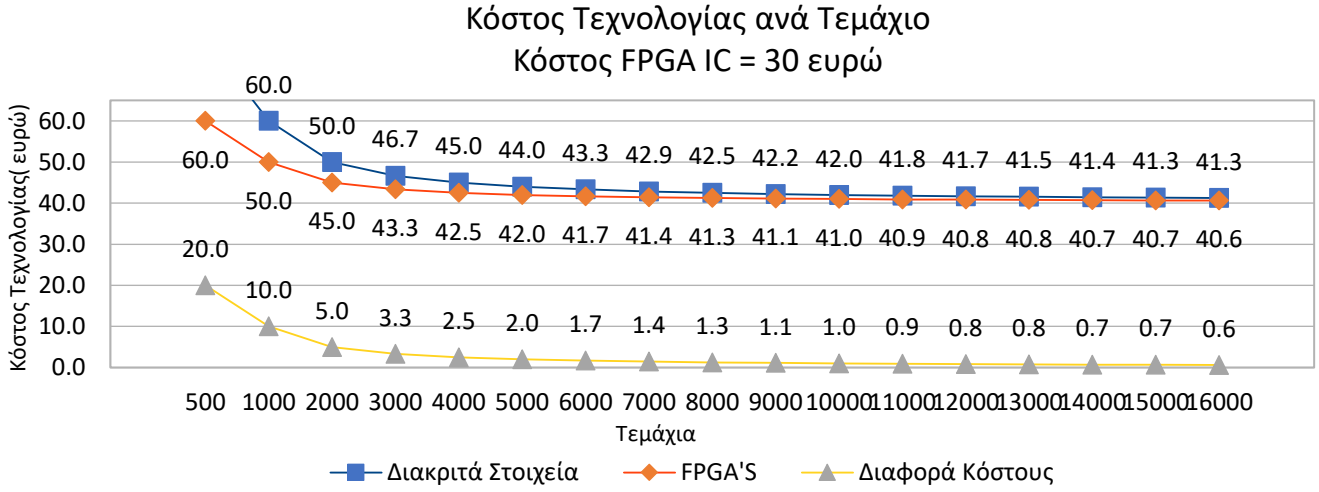
Η τεχνολογία FPGA συμφέρει για κάθε ποσότητα τεμαχίων x ή ανά τεμάχιο αν και μόνο αν για κάθε x η αντίστοιχη συνάρτηση κόστους της τεχνολογίας FPGA υπολείπεται της συνάρτησης κόστους της τεχνολογίας των ΔΣ. Μαθηματικά θα πρέπει, η διαφορά,

$K_{\Delta}(x) = K_{\Delta\Sigma}(x) - K_{FPGA}(x) > 0 \to K_{\Delta}(x) = 10.000 + (30 - k) * x > 0$

Οπότε, ανά τεμάχιο ισχύει: $K_{\Delta x}(x) = \frac{K_{\Delta}(x)}{x} = \frac{10.000}{x} + 30 - k > 0$

Καθώς $x \to \infty$, ο πρώτος όρος τείνει ασυμπτωτικά στο 0 και ο δεύτερος όρος είναι σταθερός και εξαρτάται από το k. Οπότε μέχρι κάποιο x ο πρώτος όρος κυριαρχεί και τότε ισχύει:

$k < \frac{10.000}{x} + 30$ Προκύπτει και γραφικά ότι:



Το ασυμπτωτικό όριο της τιμής του I.C. της τεχνολογίας FPGA, k είναι τα 30 €. Το k μπορεί να μεταβάλλεται επομένως για ένα εύρος τεμαχίων από 1.000 – 16.000 θα πρέπει το κάθε ολοκληρωμένο να στοιχίζει 30 €. Ενδεικτικά,

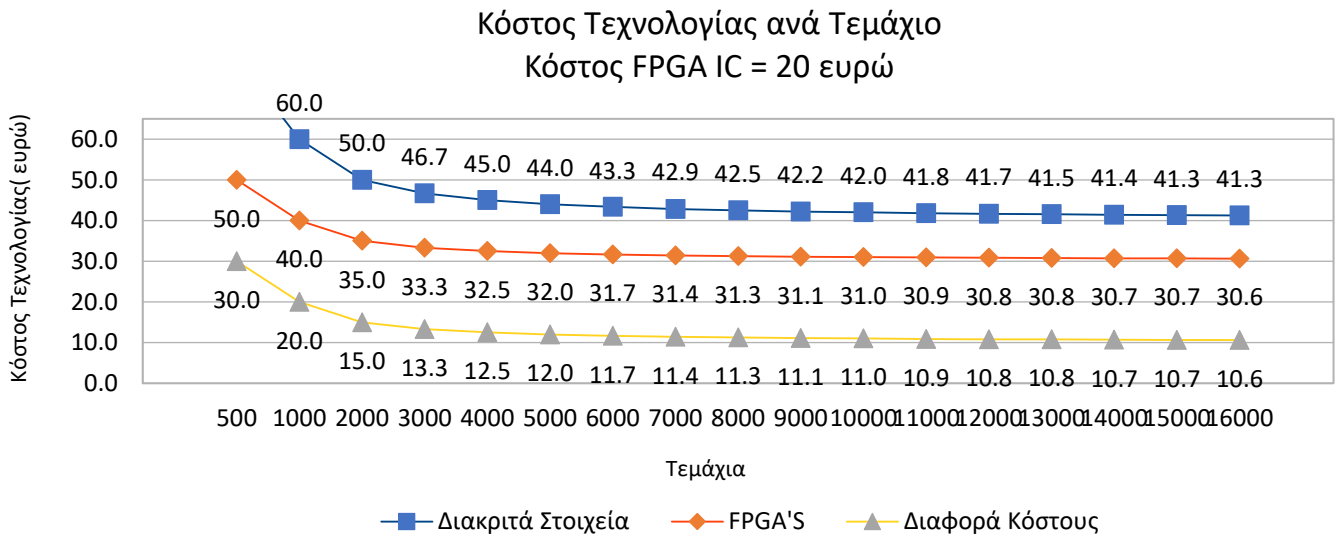
$x = 1.000 \rightarrow k \leq 40.00$ €

$x = 2.000 \rightarrow k \leq 35.00$ €

$x = 16.000 \rightarrow k \leq 30.06$ €

Πράγματι, αυτές οι τιμές επαληθεύονται από το διάγραμμα. Επίσης, δείχνει ότι η διαφορά στο κόστος των δύο τεχνολογιών μειώνεται σχετικά γρήγορα και ασυμπτωτικά συνεχώς. Από ένα σημείο και ύστερα η διαφορά αυτή είναι αμελητέα. Ήδη, από τα 5.000 τεμάχια, η διαφορά κόστους μειώνεται κατά 80% σε σχέση με την αρχική για 1.000 τεμάχια, διότι ο σταθερός όρος που αναφέρθηκε παραπάνω κυριαρχεί τελικά στη σχέση και επιδρά καθοριστικά.

Πιο συγκεκριμένα, ο σταθερός όρος 30 – k καθορίζει την ασυμπτωτική διαφορά. Για k = 30 € η διαφορά ισούται με 0 €, για k = 20 € είναι 10 € κλπ. Άρα, για μεγαλύτερη και αξιόλογη μείωση του κόστους επιβάλλεται μια μείωση του k όσο το δυνατόν κάτω από τα 30 €. Έτσι, για k = 20 € προκύπτει το ακόλουθο διάγραμμα, όπου γίνεται σαφές ότι για τιμές του k γύρω στα 20 € η διαφορά στο κόστος καθίσταται αξιόλογη (10 €). Τέλος, επισημαίνεται ότι όσο αυξάνεται ο πλήθος των παραγόμενων τεμαχίων τόσο το κόστος προσεγγίζει την τιμή του σταθερού όρου.



5η Άσκηση

i.

```
//F1 = A(CD+B)
module circuit_1 (A, B, C, D, F1);
   input A, B, C, D;
   output F1;

   wire   Cnot, Dnot, right;
   not
     G1 (Cnot, C),
     G2 (Dnot, D);
   and G6 (right, B, Cnot, Dnot);

   wire   CD, CD_or_B, left;
   and G3 (CD, C, D);
   or G4 (CD_or_B, CD, B);
   and G5 (left, CD_or_B, A);

   or G7 (F1, left, right);
endmodule

//F2 = S(0, 2, 3, 5, 7, 8, 10, 11, 14, 15)
module circuit_2 (A, B, C, D, F2);
   input A, B, C, D;
   output F2;

   wire   Anot, Bnot, Cnot, Dnot;
   not
     G1 (Anot, A),
     G2 (Bnot, B),
     G3 (Cnot, C),
     G3 (Dnot, D);


   wire [9:0] x;
   and
     G4 (x[0], Anot, Bnot, Cnot, Dnot),
     G5 (x[1], Anot, Bnot, C, Dnot),
     G6 (x[2], Anot, Bnot, C, D),
     G7 (x[3], Anot, B, Cnot, D),
     G8 (x[4], Anot, B, C, D),
     G9 (x[5], A, Bnot, Cnot, Dnot),
     G10 (x[6], A, Bnot, C, Dnot),
     G11 (x[7], A, Bnot, C, D),
     G12 (x[8], A, B, C, Dnot),
     G13 (x[9], A, B, C, D);

   or G14 (F2, x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8], x[9]);
endmodule
```

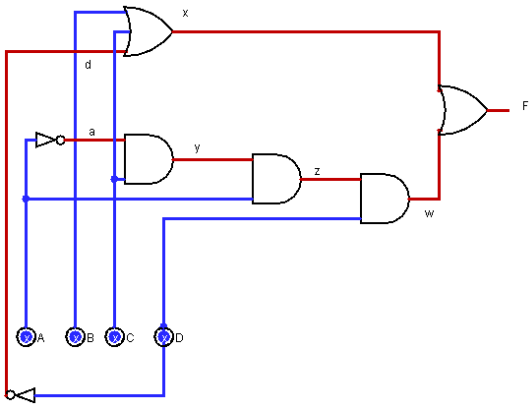
```
//F3 = ABC + (A + B)CD
module circuit_3 (A, B, C, D, E, F3);
   input A, B, C, D, E;
   output F3;

   wire   ABC;
   and G1 (ABC, A, B, C);

   wire   CD, A_or_B, middle;
   and G2 (CD, C, D);
   or G3 (A_or_B, A, B);
   and G4 (middle, A_or_B, CD);

   wire   B_or_CD, right;
   or G5 (B_or_CD, B, CD);
   and G6 (right, B_or_CD, E);

   or (F3, ABC, middle, right);
endmodule

//F4 = A(BC + D + E) + CDE
module circuit_4 (A, B, C, D, E, F4);
   input A, B, C, D, E;
   output F4;

   wire   CDE;
   and G1 (CDE, C, D, E);

   wire   BC, BC_or_D_or_E, left;
   and G2 (BC, B, C);
   or G3 (BC_or_D_or_E, BC, D, E);
   and G4 (left, A, BC_or_D_or_E);

   or G5 (F4, left, CDE);
endmodule
```

ii.

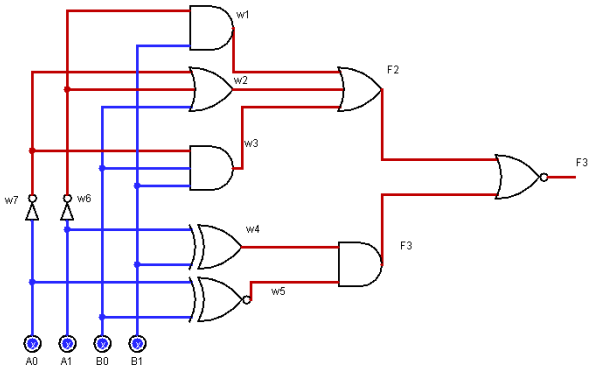
```
module circuit_1 (A, B, C, D, F1);
   input A, B, C, D;
   output F1;

   assign F1 = (A&((C&D)|B))|(B&(~C)&(~D));
endmodule // circuit_1

module circuit_2 (A, B, C, D, F2);
   input A, B, C, D;
   output F2;

   assign 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);
endmodule

module circuit_3 (A, B, C, D, E, F3);
   input A, B, C, D, E;
   output F3;

   assign F2 = (A&B&C)|((A|B)&C&D)|((B|(C&D))&E);
endmodule // circuit_3

module circuit_4 (A, B, C, D, E, F4);
   input A, B, C, D, E;
   output F4;

   assign F4 = (A&((B&C)|D|E))|(C&D&E);
endmodule // circuit_4
```

6η Άσκηση

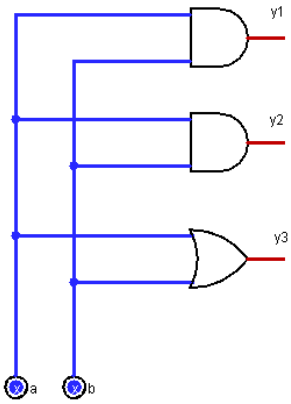
i. a)



b)



c)



ii.

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

//full adder
module full_adder (output S, C, input x, y, z);
   wire S1, C1, C2;
   half_adder ha_1 (S1, C1, x, y), ha_2 (S, C2, S1, z);
   or G1 (C, C2, C1);
endmodule // full_adder

//2-1 multiplexer
//S = 0 -> A
//S = 1 -> B
module mux2_1 (output w, input A, B, S);
   wire Snot, Aselect, Bselect;

   not (Snot, S);
   and
     (Aselect, A, Snot),
     (Bselect, B, S);
   or (w, Aselect, Bselect);
endmodule

//four-bit add-subtract
//sub = 1 -> subtract (B - A)
//sub = 0 -> add (B + A)
module adder_subtractor_4 (output [3:0] S, output c, input [3:0] A,B, input
sub);
   wire [3:1] C;
   wire [3:0] W;
   wire [3:0] Anot;

   not
     N0 (Anot[0], A[0]),
     N1 (Anot[1], A[1]),
     N2 (Anot[2], A[2]),
     N3 (Anot[3], A[3]);

   //Addition -> Choose A
   //Subtraction -> Choose not A
   mux2_1
     M0 (W[0], A[0], Anot[0], sub),
     M1 (W[1], A[1], Anot[1], sub),
     M2 (W[2], A[2], Anot[2], sub),
     M3 (W[3], A[3], Anot[3], sub);

   full_adder
     FA0 (S[0], C[1], W[0], B[0], sub),
     FA1 (S[1], C[2], W[1], B[1], C[1]),
     FA2 (S[2], C[3], W[2], B[2], C[2]),
     FA3 (S[3], c, W[3], B[3], C[3]);
endmodule
```

iii.

```
//four-bit add-subtract
//sub = 1 -> subtract (A - B)
//sub = 0 -> add (A + B)
module adder_subtractor_4 (output [3:0] S, output c, input [3:0] A,B, input
sub);
   assign {c, S} = (sub) ? A-B : A+B;
endmodule // adder_subtractor_4
```

7η Άσκηση

i.

```
module mealy_fsm (x, state, clock, y);
   input x, clock;
   output y;
   reg    y;
   reg [1:0] state;

   //00 -> a, 01 -> b, 10 -> c, 11 -> d
   parameter sa = 2'b00, sb = 2'b01, sc = 2'b10, sd = 2'b11;

   always @(posedge clock)
     case (state)
       sa: if (x)
         begin
            state <= sc;
            y = 0;
         end
       else
         begin
            state <= sb;
            y = 1;
         end

       sb: if (x)
         begin
            state <= sd;
            y = 1;
         end
       else
         begin
            state <= sc;
            y = 0;
         end

       sc: if (x)
         begin
            state <= sd;
            y = 1;
         end
       else
         begin
            state <= sb;
            y = 0;
         end

       sd: if (x)
         begin
            state <= sa;
            y = 0;
         end
       else
         begin
            state <= sc;
            y = 1;
         end
     endcase // case (state)
endmodule
```

ii.

```
module moore_fsm (x, state, clock, y);
   input x, clock;
   output y;
   reg    y;
   reg [1:0] state;

   //00 -> a, 01 -> b, 10 -> c, 11 -> d
   parameter sa = 2'b00, sb = 2'b01, sc = 2'b10, sd = 2'b11;

   always @(posedge clock)
     case (state)
       sa:
         begin
            y = 0;
            if (x) state <= sc;
            else state <= sb;
         end

       sb:
         begin
            y = 1;
            if (x) state <= sd;
            else state <= sc;
         end

       sc:
         begin
            y = 1;
            if (x) state <= sd;
            else state <= sb;
         end

       sd:
         begin
            y = 0;
            if (x) state <= sa;
            else stae <= sc;
         end
     endcase // case (state)
endmodule
```

iii.

```
// D flip-flop with asynchronous reset
module DFF ( output reg Q, input D, Clk, rst);
   always @ ( posedge Clk, negedge rst)
     if (~rst) Q <= 1'b0; // Same as: if (rst == 0)
     else Q <= D;
endmodule

module counter_4(clear, count_up, clock, count);
   input clear, count_up, clock;
   reg [3:0] D;
   output [3:0] count;

   parameter zero = 4'b0000, one = 4'b0001;

   DFF
     D1 (count[0], D[0], clock, clear),
     D2 (count[1], D[1], clock, clear),
     D3 (count[2], D[2], clock, clear),
     D4 (count[3], D[3], clock, clear);

   always @(posedge clock)
     begin
        else if (count_up) D <= D + one;

        //if we arent counting up, then we are counting down
        else D <= D - one;
     end
endmodule
```