Ομάδα: Χρίστος Χατζηχριστοφή(03117711)

Δημήτρης Λάμπρος(03117070)

Μάθημα: Συστήματα Μικροϋπολογιστών

Σχολή – Εξάμηνο: ΗΜΜΥ 6ο

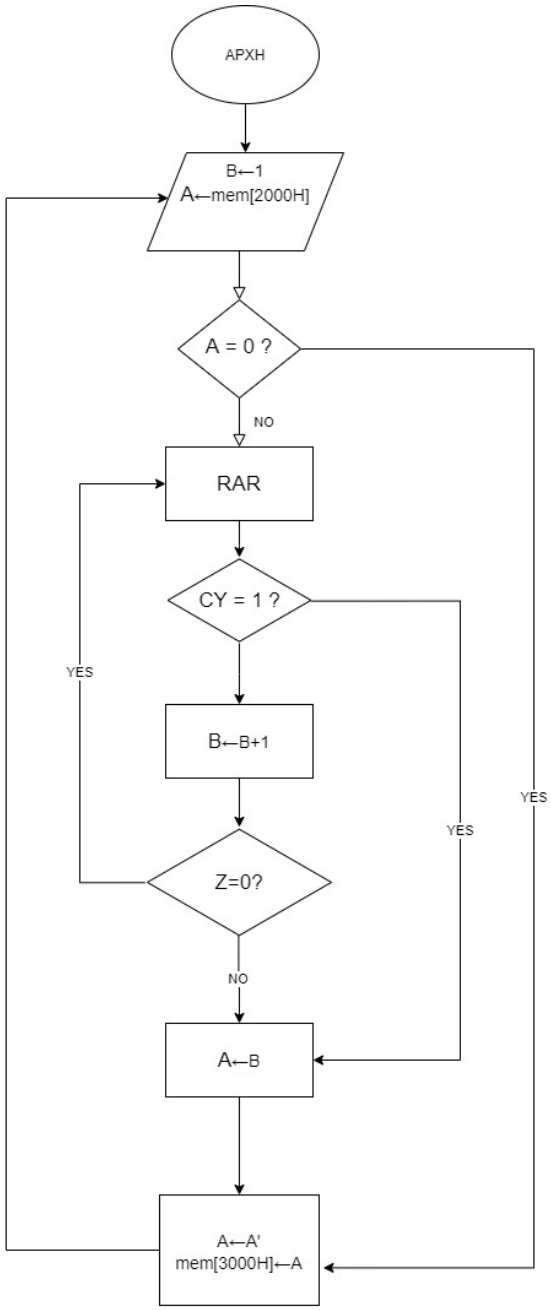
Πρώτο σετ ασκήσεων

Άσκηση 1:

Αφού κάναμε το πρόγραμμα που μας δώθηκε disassemble παρατηρήσαμε ότι η λειτουργία του είναι η εξής: Παίρνει ένα δυαδικό αριθμό απο το 0 εώς το 255, τον οποίο δίνουμε εμείς μέσω των led. Στη συνέχεια αρχίζει να μετράει πόσες φορές έγινε διαίρεση χωρις υπόλοιπο, μέχρι που εν τέλει να βρεί μια διαίρεση που να αφήνει υπόλοιπο. Τελικά επιστρέφει πόσες φορές έγινε διαίρεση χωρίς υπόλοιπο, μετρώντας και την τελευταία διαίρεση που βρέθηκε το υπόλοιπο. Μια άλλη εκδοχή του τι μπορεί να κάνει το πρόγραμμα είναι το να εμφανίζει ποιός ήταν ο πρώτος άσσος του A ξεκινώντας απο το LSB προς το MSB. Έπειτα κάνει complementary το περιεχόμενο του A και εμφανίζει στα led το αποτέλεσμα.

Σημείωση: Ο λόγος που κάνει το complementary, είναι για να μπορεί να αναπαραστήσει το αποτέλεσμα, καθώς τα led είναι αρνητικής λογικής.

Λογικό Διάγραμμα:



Κώδικας:

```
0800   06     MVI B,01H
0801   01
0802   3A     LDA 2000H
0803   00
0804   20
0805   FE     CPI 00H
0806   00
0807   CA     JZ LABEL1
0808   13
0809   08

LABEL3:
080A   1F     RAR
080B   DA     JC LABEL2
080C   12
080D   08
080E   04     INR B
080F   C2     JNZ LABEL3
0810   0A
0811   08

LABEL2:
0812   78     MOV A,B

LABEL1:
0813   2F     CMA
0814   32     STA 3000H
0815   00
0816   30
0817   CF     RST 1
```

```
ORG 0800H
            MVI B,01H    ; B <-- 01H
            LDA 2000H    ; Read input from dip switches
            CPI 00H      ; Compare input of dip switches to zero
            JZ LABEL1    ; If z=1 then jump to LABEL1

LABEL3:     RAR          ; If last switch is ON then
            JC LABEL2    ; CY=1 then jump to LABEL2
            INR B        ; else b++
            JNZ LABEL3   ; If Z=0 jump LABEL3 else continue

LABEL2:     MOV A,B      ; A <-- B

LABEL1:     CMA          ; A --> A complementary
            STA 3000H    ; [3000H] = A
            RST 1
END
```

Μνήμη και εντολές -> Πρόγραμμα

Οι αλλαγές που πρέπει να γίνουν ώστε το πρόγραμμα να τρέχει αενάως είναι:
JMP BEGIN αντί για RST 1, και προστέθηκε το LABEL BEGIN στην αρχή του προγράμματος.

```
BEGIN:
0800   06     MVI B,01H
0801   01
0802   3A     LDA 2000H
0803   00
0804   20
0805   FE     CPI 00H
0806   00
0807   CA     JZ LABEL1
0808   13
0809   08

LABEL3:
080A   1F     RAR
080B   DA     JC LABEL2
080C   12
080D   08
080E   04     INR B
080F   C2     JNZ LABEL3
0810   0A
0811   08

LABEL2:
0812   78     MOV A,B

LABEL1:
0813   2F     CMA
0814   32     STA 3000H
0815   00
0816   30
0817   C3     JMP BEGIN
0818   00
0819   08
```

```
ORG 0800H
BEGIN:      MVI B,01H    ; B <-- 01H
            LDA 2000H    ; Read input from dip switches
            CPI 00H      ; Compare input of dip switches to zero
            JZ LABEL1    ; If z=1 then jump to LABEL1

LABEL3:     RAR          ; If last switch is ON then
            JC LABEL2    ; CY=1 then jump to LABEL2
            INR B        ; else b++
            JNZ LABEL3   ; If Z=0 jump LABEL3 else continue

LABEL2:     MOV A,B      ; A <-- B

LABEL1:     CMA          ; A --> A complementary
            STA 3000H    ; [3000H] = A
            JMP BEGIN    ; loop without ending condition
END
```

Μνήμη και εντολές -> Πρόγραμμα

Άσκηση 2:

```
ORG 0800H
                    IN 10H
                    MVI D,01H
                    LDA 2000H
                    CMA
                    STA 3000H           ;D=1, LOADS 1st "bit" of switches
                    LXI B,0244H         ;0244H = 580
                    CALL DELB           ;delay 0.58s
INCREASE:           LDA 2000H
                    CPI 01H
                    JC CYCLE            ;if A<1 then CY=1, GO TO CYCLE
                    CPI 02H
                    JZ LIGHT_LED0       ; if A=2 turn on led 0 only
                    CMC                 ;get rid of carry
                    MOV A,D             ;else A=D;
                    RAL                 ;A=2*A;
                    CPI 00H             ;if A==0 (leftmost bit of A has gone a full cycle) go to DECREASE
                    JZ DECREASE
                    MOV D,A             ; ELSE  (D=A=2*Aold)
                    CMA
                    STA 3000H           ;complement A to turn on corresponding led.
                    LXI B,0244H         ;0244H = 580
                    CALL DELB           ;delay 0.58s
                    JMP INCREASE        ;repeat till full cycle or switches change

DECREASE:           LDA 2000H
                    CPI 01H
                    JC CYCLE            ; If A<1 then CY=1 , go to cycle
                    CPI 02H
                    JZ LIGHT_LED0       ;if A=2 turn on led 0 only
                    CMC
                    MOV A,D
                    RAR                 ; A=A/2,right shift
                    CPI 00H
                    JZ INCREASE         ;Do this till all bits have been shifted right,
                    MOV D,A             ;so A=0 ,then go to increase
                    CMA
                    STA 3000H           ;complement A to turn on corresponding led.
                    LXI B,0244H         ;0244H = 580
                    CALL DELB           ;delay 0.58s
                    JMP DECREASE

CYCLE_INIT: MVI D,01H
                    MOV A,D
                    CMA
                    STA 3000H
                    LXI B,0244H         ;0244H = 580
                    CALL DELB           ;delay 0.58s

CYCLE:      LDA 2000H
                    CPI 01H
                    JZ INCREASE         ;IF A==1 go to increase-decrease mode
                    CPI 02H
                    JZ LIGHT_LED0       ; If A=2 turn on led 0 only
                    CMC
                    MOV A,D
                    RAL                 ;else continue from LED that was last on and do cycle mode
                    CPI 00H
                    JZ CYCLE_INIT       ;Difference from increase-decrease is we reinit starting
                    MOV D,A             ;led when led-on reaches last led.
                    CMA
                    STA 3000H           ;print
                    LXI B,0244H         ;0244H = 580
                    CALL DELB           ;delay 0.58s
                    JMP CYCLE           ;delay

LIGHT_LED0: LDA 2000H
                    CPI 01H
                    JZ INCREASE         ; if A=1 go to increase-decrease mode
                    JC CYCLE            ; if  A=1 go to cycle
                    RAR
                    CMA                 ;do a right shift so that led 1 turns on
                    STA 3000H
                    JMP LIGHT_LED0      ;repeat till A changes


END
```

Άσκηση 3:

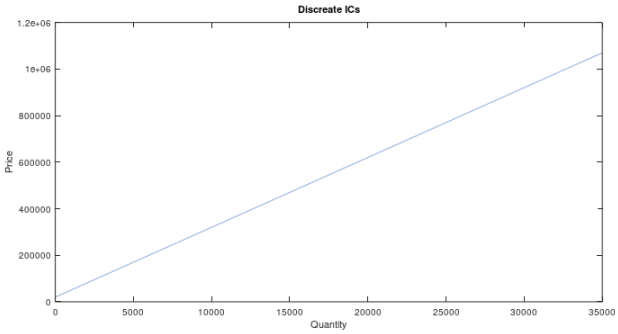
```
; a program that converts any 8bit binary number to its BCD where
; the first 4 MSB digits represent the tens and the 4 LSB  represent the units

ORG 0800H
LOOP1:              LDA 2000H ; reads the input from the dip switches
                    MVI B,00H ; counter for tens
STILL_GREATER:      CPI 64H ; input == 100(decimal) ?
                    JNC GREATER_OR_EQUAL_100
                    CMC ; we need to reset CY flag due to previous comparison
TENS:               CPI 0AH ; compare input with 10
                    JC UNITS ; if A is less than 10 then we have no tens
                    SUI 0AH ; Subtract 10 from accumulator(input)
                    INR B ; tens++
                    JMP TENS ; loop until no more TENS exist

UNITS:              CMC ; we need to reset CY flag due to previous comparison
                    MOV C,A ; temporary hold of the units
                    MOV A,B ; pass the tens to the accumulator
                    RAL ; with the next 4 RAL , we move the last four
                    RAL ; digits of accumulator to the 4 MSB digits
                    RAL ; because A now holds the tens
                    RAL ; then we sum A and C to have the
                    ADD C ;complete form TTTTUUUU(T = TENS , U = UNITS)
                    CMA ; now do the printing
                    STA 3000H
                    JMP LOOP1

GREATER_OR_EQUAL_100:      SUI 64H ; subtract 100 from the accumulator
                                   JMP STILL_GREATER
END
```

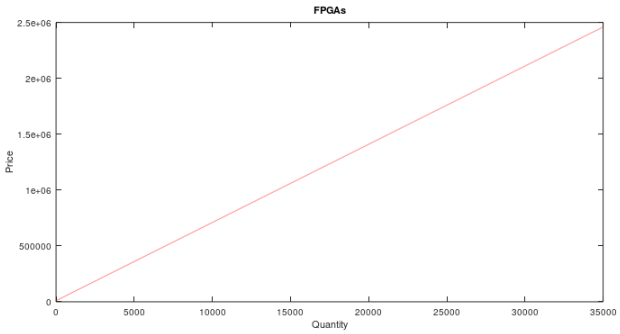
Άσκηση 4:

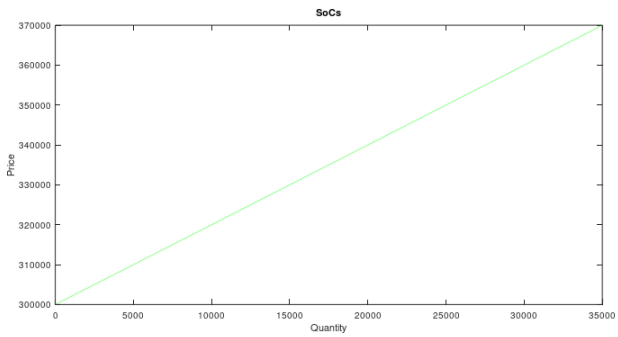
Αρχικά πρέπει να υλοποιήσουμε τις γραφικές παραστάσεις. Επομένως πρέπει να καταστρώσουμε τις εξισώσεις που ορίζουν τις καταστάσεις αυτές. Επομένως, για τις πιο κάτω περιπτώσεις έχουμε τις εξής εξισώσεις:

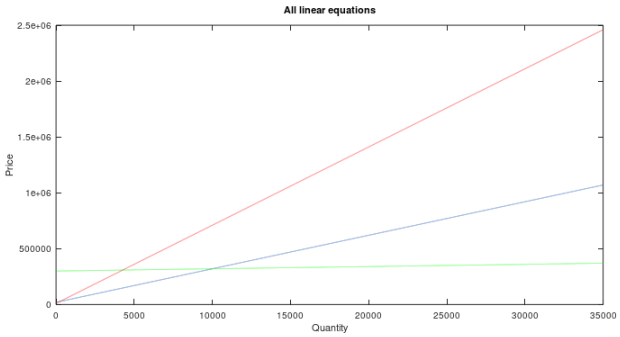
1. Στην πρώτη περίπτωση υπάρχει σταθερό κόστος 20000€ και 30€ κόστος ανα πλακέτα. Συνεπώς $y = 30x + 20000$.
2. Στην δεύτερη περίπτωση υπάρχει σταθερό κόστος 10000€ και 70€ κόστος ανα πλακέτα. Συνεπώς $y = 70x + 10000$.
3. Στην τρίτη περίπτωση υπάρχει σταθερό κόστος 300000€ και 2€ κόστος ανα πλακέτα. Συνεπώς $y = 2x + 300000$.

Με την χρήση του Octave μπορούμε να αναπαραστήσουμε τις πιο κάτω γραφικές, οι οποίες εκφράζουν τις πάραπανω εξισώσεις.









Παρατηρώντας τις πιο πάνω γραφικές μπορούμε να υποδείξουμε ποια τεχνολογία είναι συμφερότερη σε κάποιο εύρος τιμών. Πιο συγκεκριμένα:

- Απο 0 εώς 250 τεμάχια συμφέρει η τεχνολογία των FPGAs.
- Μεγαλύτερο των 250 εώς 10000 τεμαχίων είναι η τεχνολογία των Discrete ICs.
- Μεγαλύτερο των 10000 τεμαχίων είναι η τεχνολογιά των SoCs.

Το εύρος τιμών για το οποίο η τεχνολογία των FPGAs μπορεί να εξαφανίσει την τεχνολογία των Discrete ICs μπορεί να βρεθεί λύνοντας την εξής ανίσωση:

$$20000 + 30x \geq 10000 + x(10 + k) \Rightarrow \frac{10000}{x} + 20 \geq k$$

Εφαρμόζωντας το πάνω όριο που συμφέρει η τεχνολογία των Discrete ICs πάνω στην ανίσωση καταφθάνουμε στο αποτέλεσμα το οποίο είναι : $0 \leq k \leq 21$. Προφανώς η μέγιστη τιμή στην οποία μπορούμε να συμβιβαστούμε είναι η 21.

Άσκηση 5:

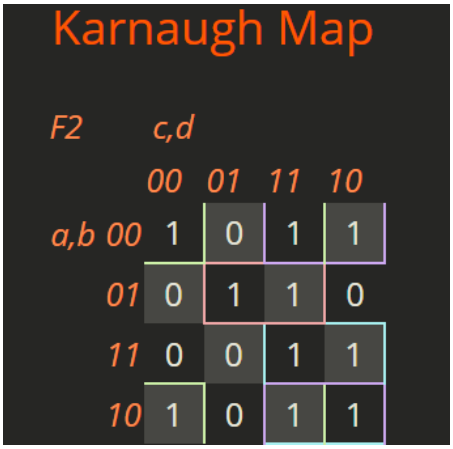
EXERCISE 5i

```
module ex5_i (A,B,C,D,E,F1,F2,F3,F4);
    output F1,F2,F3,F4;
    input A,B,C,D,E;
    wire a,b,c,d ;
    not (a,A);
    not (b,B);
    not (c,C);
    not (d,D);
/* ----------F1---------- */
//F1=A(CD + B) + BC'D'
    wire w1,w2,w3,w4;
    and (w1,C,D);  //CD
    or (w2,w1,B);  //B+CD
    and (w3,A,w2); // A(B+CD)
    and (w4,B,c,d); //BC'D'
    or (F1,w4,w3);
/*-------END OF F1-------*/

/*----------F2-----------*/
    //F2=B'D' + A'BD + AC + B'C
    wire w5,w6,w7,w8;
    and (w5,b,d);   //B'D'
    and (w6,a,B,D); //A'BD
    and (w7,A,C);   // AC
    and (w8,b,C);   // B'C
    or (F2,w5,w6,w7,w8);
/*-------END OF F2-------*/
/* ----------F3---------- */
//F3=ABC + (A + B)CD +(B+ CD)E
    wire w9,w10,w11,w12,w13,w14;
    and (w9,A,B,C);  //ABC
    and (w10,C,D);  // CD
    or (w11,A,B); // A+B
    and(w12,w11,w10); //(A+B)CD
    or (w13,w10,B);     //CD+B
    and (w14,E,w13);    //(CD+B)E
    or (F3,w14,w12,w9);
/*-------END OF F3-------*/

/* ----------F4---------- */
//F4=A(BC + D + E) + CDE
    wire w15,w16,w17,w18;
    and (w15,w10,E); //CDE where w10==CD(look F3)
    and (w16,B,C); //BC
    or (w17,w16,D,E); // BC+D+E
    and (w18,w17,A); //A(BC+D+E)
    or (F4,w18,w15);
/*-------END OF F4-------*/
endmodule
```

EXERCISE 5ii

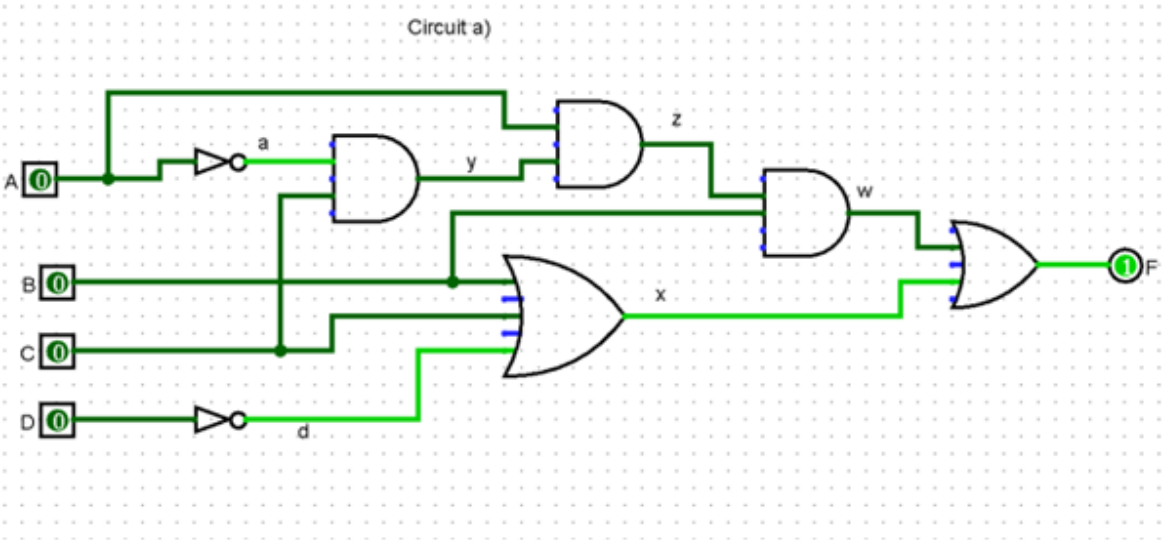
```
module ex5_ii(A,B,C,D,E,F1,F2,F3,F4);
    output F1,F2,F3,F4;
    input A,B,C,D,E;

/* ---------F1-------*/
//F1=A(CD + B) + BC'D'
    assign F1= (A&((C&D)|B))|(B&~C&~D);
/*-------END OF F1-------*/

/*----------F2-----------*/
//F2=B'D' + A'BD + AC + B'C
    assign F2= (~B&~D)|(~A&B&D)|(A&C)|(~B&C);
/*-------END OF F2-------*/

/* ---------F3-------*/
//F3=ABC + (A + B)CD +(B+ CD)E
    assign F3= (A&B&C)|((A|B)&C&D)|((B|(C&D))&E);
/*-------END OF F3-------*/

/* ---------F4-------*/
//F4=A(BC + D + E) + CDE
    assign F4= (A&((B&C)|D|E))|(C&D&E);
/*-------END OF F4-------*/
endmodule
```

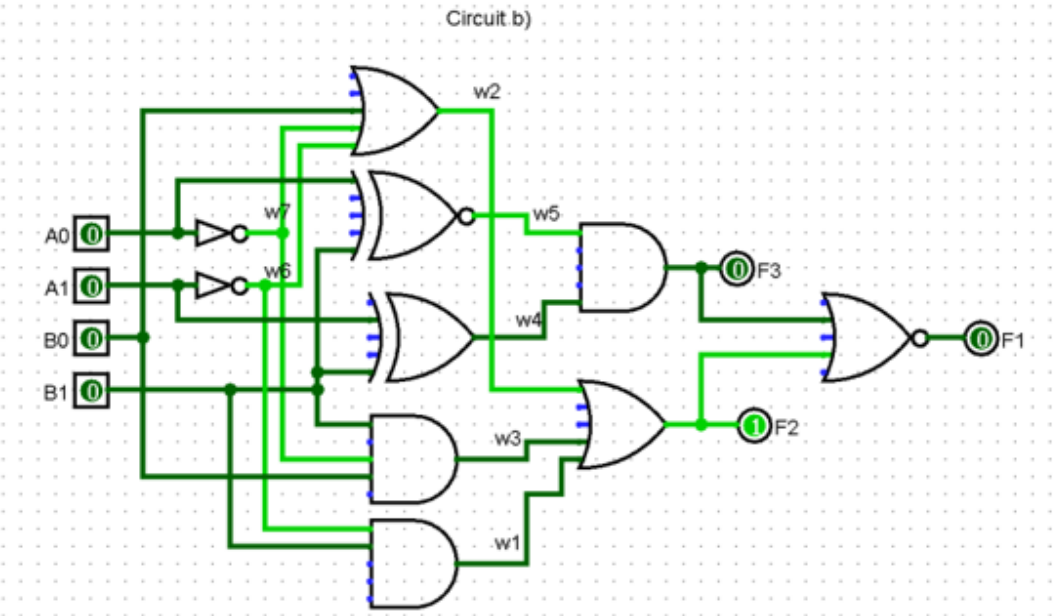
Σημείωση: Μετά απο απλοποίηση Karnaugh προέκυψε η απλοποιημένη F2. Παρατείθεται ο χάρτης Karnaugh:

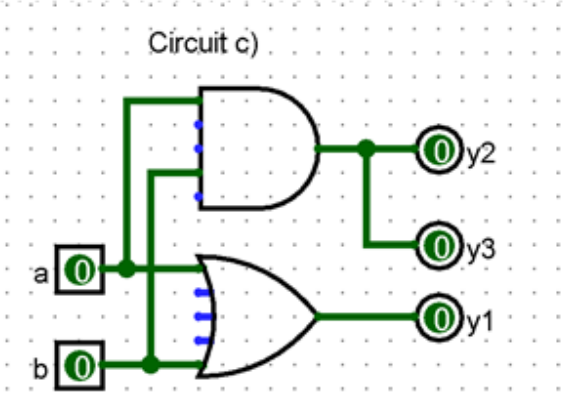


Άσκηση 6:

(i)


Circuit a)
A
B
C
D
a
y
z
w
x
d
F


Circuit b)
A0
A1
B0
B1
w7
w6
w2
w5
w4
w3
w1
F3
F1
F2


Circuit c)
a
b
y2
y3
y1

(ii)

```
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); // Instantiate HAs
    half_adder HA2 (S, C2, S1, z);
    or G1 (C, C2, C1);
endmodule
// Description of four-bit adder
module Four_bit_adder ( output [3: 0] Sum,output C4, input [3: 0] A, B, input C0);
    wire C1, C2, C3; // Intermediate carries
    // Instantiate chain of full adders
    full_adder  FA0 (Sum[0], C1, A[0], B[0], C0),
                FA1 (Sum[1], C2, A[1], B[1], C1),
                FA2 (Sum[2], C3, A[2], B[2], C2),
                FA3 (Sum[3], C4, A[3], B[3], C3);
endmodule
module Four_bit_adder_subtracter (input M,input [3: 0] A, B,output [3: 0] Sum,output C4);
    wire [3: 0] second_operand;
    xor (second_operand[0],A[0],M); //Depending on input M, addition(M=0) or substraction(M=1)
    xor (second_operand[1],A[1],M); //selected. XOR output is A if M=0 and
    xor (second_operand[2],A[2],M); // A' if  M=1 (1s' complement)
    xor (second_operand[3],A[3],M);
    Four_bit_adder FourBAS(Sum,C4,B,second_operand,M); // If M=1, Cin=1 so we get 2s'
                                                       // complement of A
endmodule
```

(iii)

```
// Dataflow description of four-bit adder
module binary_adder_subtracter (Sum, Cout, A,B,M);

    output [3: 0] Sum;
    output Cout;
    input [3: 0] A, B;
    input M;
    assign {Cout, Sum} = (M==1)?(A + B):(A-B);
endmodule
```

Άσκηση 7:

(i)

```
// Mealy model FSM
module Mealy_Model (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; //initial 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[0]^state[1]^x); // FROM y truth table
endmodule
```

Πίνακας αληθείας για την Έξοδο Y:

A	B	X	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Όπου με βάση τον πιο πάνω πίνακα παρατηρήσαμε ότι η έξοδος Y = XNOR(A,B,X).

(ii)

```
// Moore model FSM
module Moore_Model (y, x, clock, reset);
    output y;
    input x, clock, reset;
    reg [1: 0] state;
    parameter  a= 2'b00, b = 2'b01, c = 2'b11, d = 2'b10;
    always @ ( posedge clock, negedge reset)
    if (reset == 0) state <= a; //initial 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[0]; // Έξοδος των flip-flops
endmodule
// Η εντολή parameter επιτρέπει ορισμό σταθερών.
```

(iii)

```
// Four-bit binary counter with UP/DOWN and  Clear
module Binary_Counter_4 (output reg [3:0] A,input U_D,CLK,RST);
always @ ( posedge CLK, negedge RST)

if (~RST) A<=4'b0000 ;          // IF CLEAR==1 , PRESET==0 THEN RESET TO ZERO
else if (U_D) A <= A + 1'b1;    // IF UPDOWN input ==1 , count upwards
else A <= A - 1'b1;             //else updown input==0 , count downwards.
//We don't have to check for A==1111 , because 1111+0001=0000
//and we don't have to check for A==0000, because 0000-1=1111
endmodule
```