1ο ΣΕΤ ΑΣΚΗΣΕΩΝ

[ΑΞΕΛΟΣ ΧΡΗΣΤΟΣ, ΑΕΜ 1814] [21/12/16] Instructor name: [Χ.ΣΩΤΗΡΙΟΥ]

Άσκηση 1: Περιγράψτε σε Verilog το κύκλωμα Πρόγνωσης Κρατουμένου (CLA) ενός 4- bit Πλήρους Αθροιστή, εξηγώντας περιληπτικά πως προκύπτει

Λύση:

Έστω οι 4bit αριθμοι input wire [3:0] A, B. Για κάθε ένα bit εφαρμόζουμε έναν full adder, με input(A[j],B[j], C[j-1]) και output C[j], όπου όλοι τους τρέχουν παράλληλα.

Αρχικά υπολογίζουμε τα Gi = Ai*Bi και Pi = Ai^Bi, έτσι ώστε να χρησιμοποιηθούν για ένα κύκλωμα full adder. Σε κάθε ένα από τα τέσσερα κυκλώματα full adder, υπολογίζουμε την έκφραση

C[j] = G[j] + P[j]*C[j-1] = A[j]*B[j] + (A[j]+B[j])*C[j-1], για το κρατούμενο(output) και S[j] = P[j] ^ C[j] για το αποτέλεσμα εξόδου sum

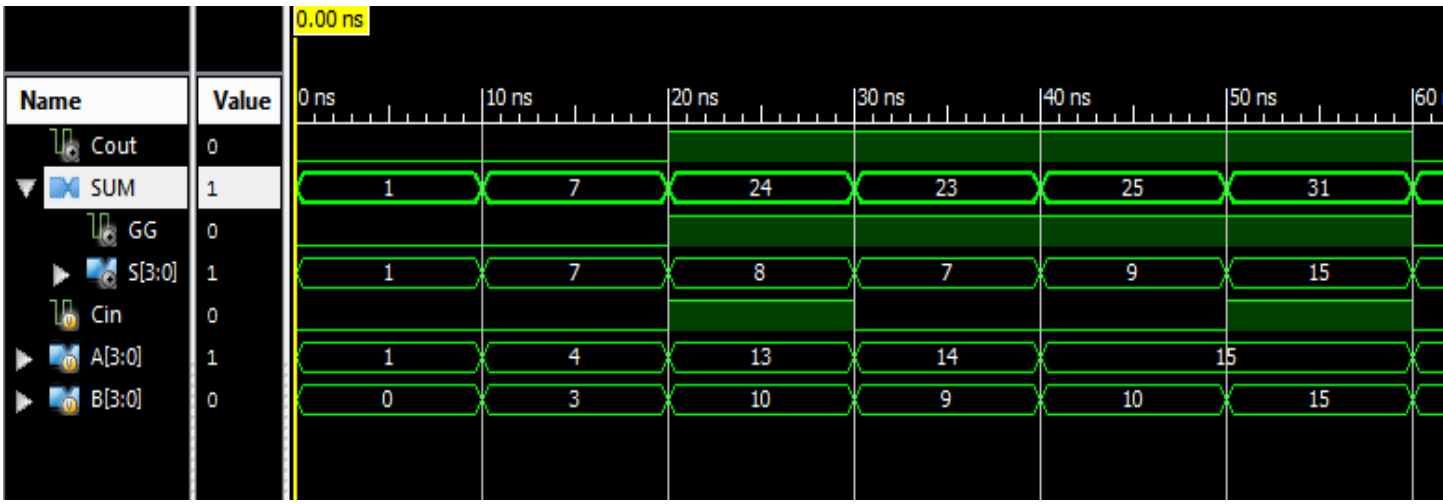
Στην πρώτη σχέση, το C[j-1] υπολογίζεται αναδρομικά εώς ότου το C[j] να υπολογίζεται συναρτήσει μόνο των G, P, C[0]=Cin

Το σήμα GroupGenerate(GG) γίνεται 1 σε περίπτωση που το άθροισμα 2 N-μπιτ αριθμών είναι N+1

```
1  `timescale 1ns / 1ps
2  module lookaheadadder(
3    input Cin,
4    input [3:0] A,
5    input [3:0] B,
6    output Cout, GG,
7    output [3:0] S
8      );
9  wire [3:0] C, P, G;
10
11 assign C[0] = Cin;//to Cin einai ena an iparxei alisida me adders
12 assign C[1] = G[0] | (P[0]&Cin);
13 assign C[2] = G[1] | ( G[0]&P[1]) | ( C[0]&P[0]&P[1]);
14 assign C[3] = G[2] | ( G[1]&P[2] ) | ( G[0]&P[1]&P[2] ) |  (C[0]&P[0]&P[1]&P[2]);
15 assign Cout = G[3] | (G[2]&P[3]) | (G[1]&P[2]&P[3]) | (G[0]&P[1]&P[2]&P[3]) | (C[0] &P[0]&P[1]&P[2]&P[3]);
16
17 assign G = A&B;
18 assign P = A^B;
19 assign S = P ^ C;
20 assign Cout = G[3] | (P[3] & G[2]) |
21              (P[3] & P[2] & G[1]) |
22              (P[3] & P[2] & P[1] & G[0]) |
23              (P[3] & P[2] & P[1] & P[0] & C[0]);
24
25 assign GG = G[3] | (G[2]&P[3]) | (G[1]&P[3]&P[2]) | (G[0]&P[3]&P[2]&P[1]);
26
27 endmodule
28
```

-Για την προσωμοίωση, χρησιμοποιώ το testbench της wikipediahttps://en.wikipedia.org/wiki/Carry-lookahead_adder που καλύπτει διάφορα cases

```
1  `timescale 1ns / 1ps
2  module tester;
3     // Inputs
4     reg Cin;
5     reg [3:0] A;
6     reg [3:0] B;
7     // Outputs
8     wire Cout;
9     wire GG;
10    wire [3:0] S;
11    // Instantiate the Unit Under Test (UUT)
12    lookaheadadder uut (
13       .Cin(Cin),
14       .A(A),
15       .B(B),
16       .Cout(Cout),
17       .GG(GG),
18       .S(S)
19    );
20    initial begin
21     A=4'b0001;B=4'b0000;Cin=1'b0;
22     #10 A=4'b100;B=4'b0011;Cin=1'b0;
23     #10 A=4'b1101;B=4'b1010;Cin=1'b1;
24     #10 A=4'b1110;B=4'b1001;Cin=1'b0;
25     #10 A=4'b1111;B=4'b1010;Cin=1'b0;
26     #10 A=4'b1111;B=4'b1111;Cin=1'b1;
27     #10 A=4'b1010;B=4'b0101;Cin=1'b0;
28     end
```



Άσκηση 2

Για την συνάρτηση:

$$f = w'y'z' + w'xy'z + wxy'z + x'yz + w'x'yz' + wx'y$$

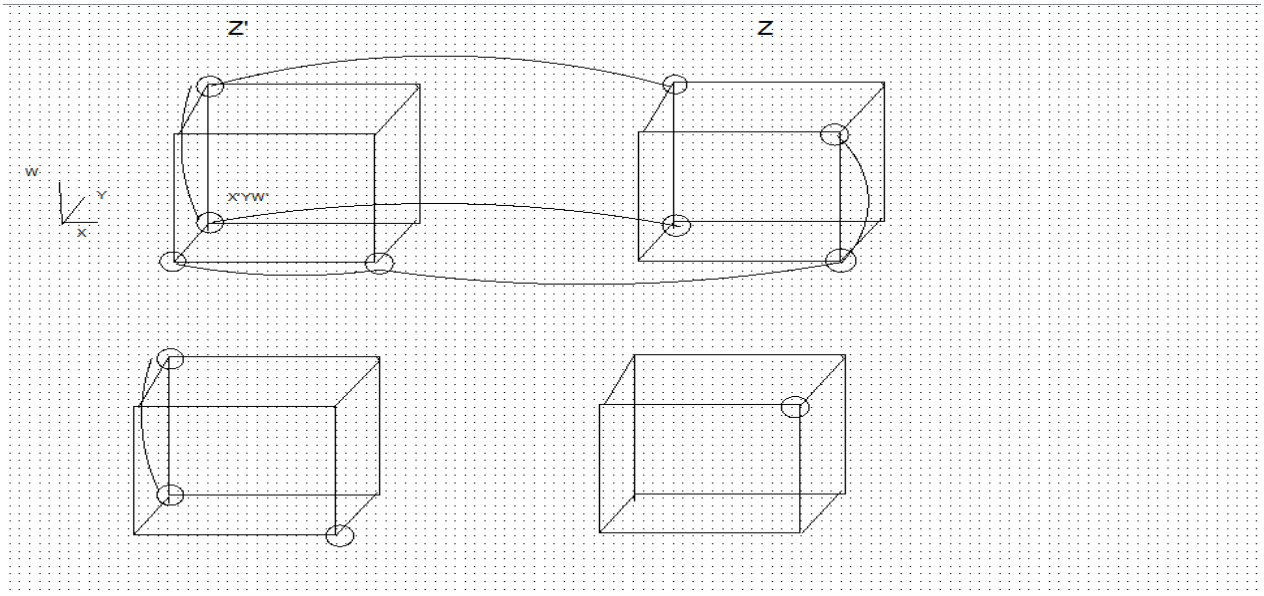
(a) Απεικονίστε την στον Δυαδικό Κύβο επιδεικνύοντας: (1) τους κύβους

της, όπως φαίνονται στην παραπάνω εξίσωση, (2) του κύβους της βέλτιστης, δι-επίπεδης υλοποίησης της.

(b) Με βάση το (2), γράψτε την *f* στην ελάχιστη της μορφή, και σχολιάστε το κέρδος σε εμβαδό. Επιπλέον, επιδείξτε πώς μπορούν να προκύψουν οι κύβοι του (2), χρησιμοποιώντας μόνο αλγεβρικές πράξεις.

Λύση:

(α) Στο παρακάτω σχήμα, τα 2 πάνω σχήματα είναι τα σχήματα όπως προκύπτουν από την f, ενώ τα 2 κάτω σχήματα προκύπτουν από από την απλοποίηση



(β) Η f με βάση τους απλοποιημένους κύβους μπορεί να γραφτεί ως

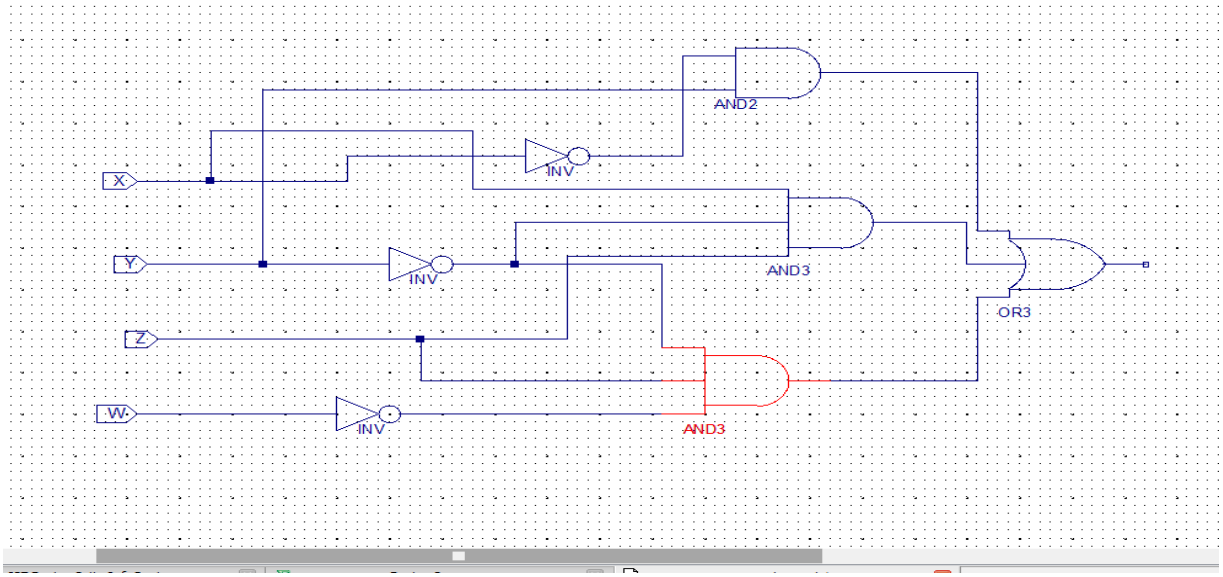
F= x'y + w'y'z' + xy'z, η οποία έχει εμβαδό = 8 με κέρδος =ΑρχΕμβ-ΤελΕμβ = 21-8=13 λόγω της χρήσης λιγότερων inputs.Το βάθος του κυκλώματος(μεγαλύτερο μονοπάτι) παραμένει 2

Το αποτέλεσμα αυτό επαληθεύεται με πράξεις,

$$f = w'y'z' + w'xy'z + wxy'z + x'yz + w'x'yz' + wx'y =$$

$$xy'z+w'xy'z'+x'yz+ w'x'yz' + wx'y = xy'z + w'y'z' + x'y(z + w'z' + w) = x'y+xy'z + w'y'z$$

, αφού από πίνακα αλήθειας z + w'z'+w = 1



Άσκηση 3

Για τις παρακάτω συναρτήσεις:

xyz	f1	f2
000	0	0
001	1	0
010	0	1
011	1	1
100	1	1
101	1	1
110	1	1
111	0	0

(α) Υπολογίστε αλγεβρικά τις αρνήσεις των *f1* και *f2*, ως άθροισμα γινομένων (όχι απαραίτητα το ελάχιστο). Κατόπιν, απεικονίστε τις δυο

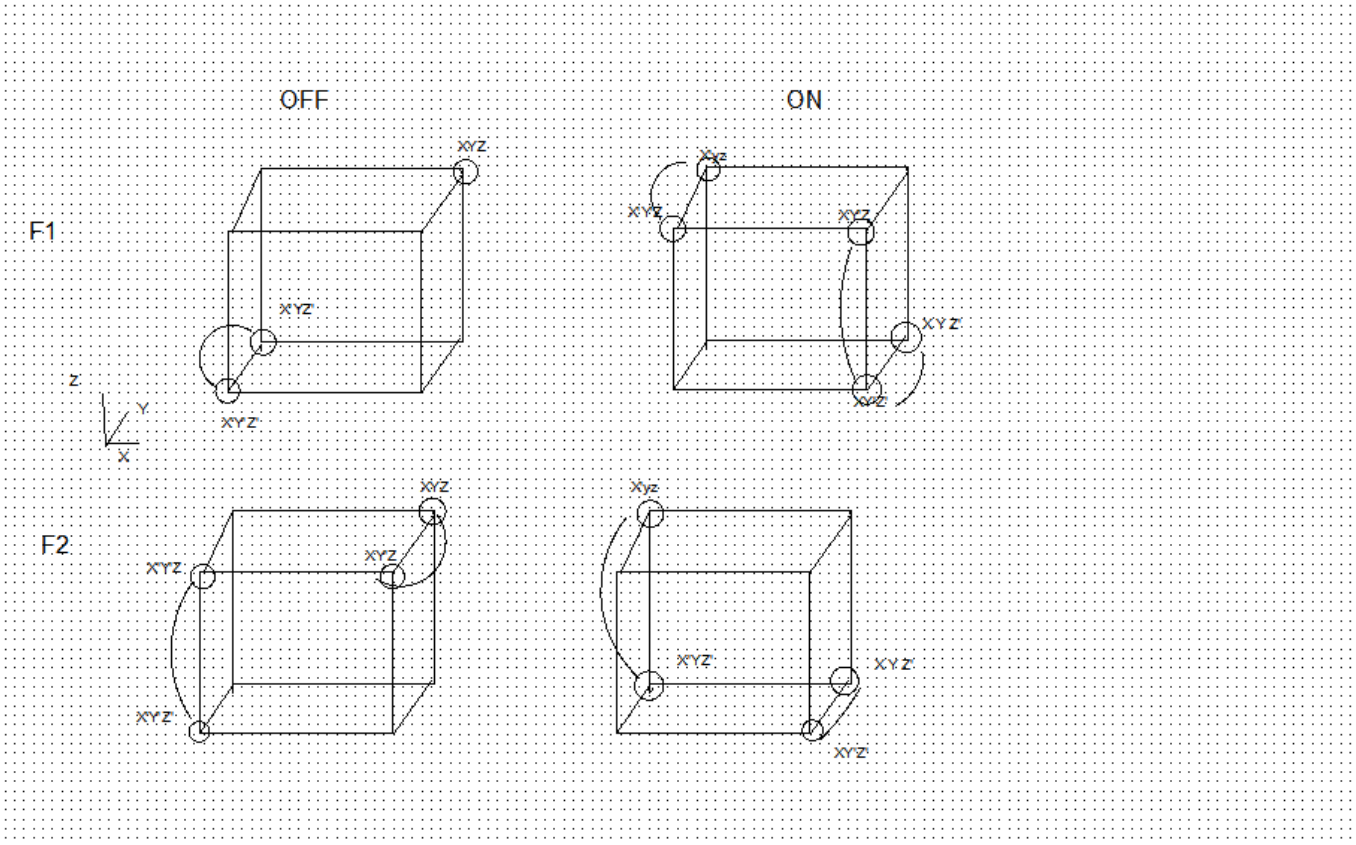
συναρτήσεις στον Δυαδικό Κύβο, επιδεικνύοντας τους κύβους των συνόλων ON και OFF. Ελαχιστοποιείστε τα δυο σύνολα στον δυαδικό κύβο, επαληθεύοντας τις αλγεβρικές αρνήσεις που υπολογίσατε

β) Βάση του (α), και παρουσιάζοντας εναλλακτικές κυκλωματικές υλοποιήσεις για τις *f1*, *f2*, ως λογικές εξισώσεις και ως σχηματικά, παρουσιάστε την υλοποίηση/υλοποιήσεις με μικρότερο αριθμό πυλών και την υλοποίηση/υλοποιήσεις με μικρότερη καθυστέρηση. (Μπορείτε να χρησιμοποιήσετε κοινούς κύβους/πύλες γις την υλοποίηση των *f1*, *f2*.)

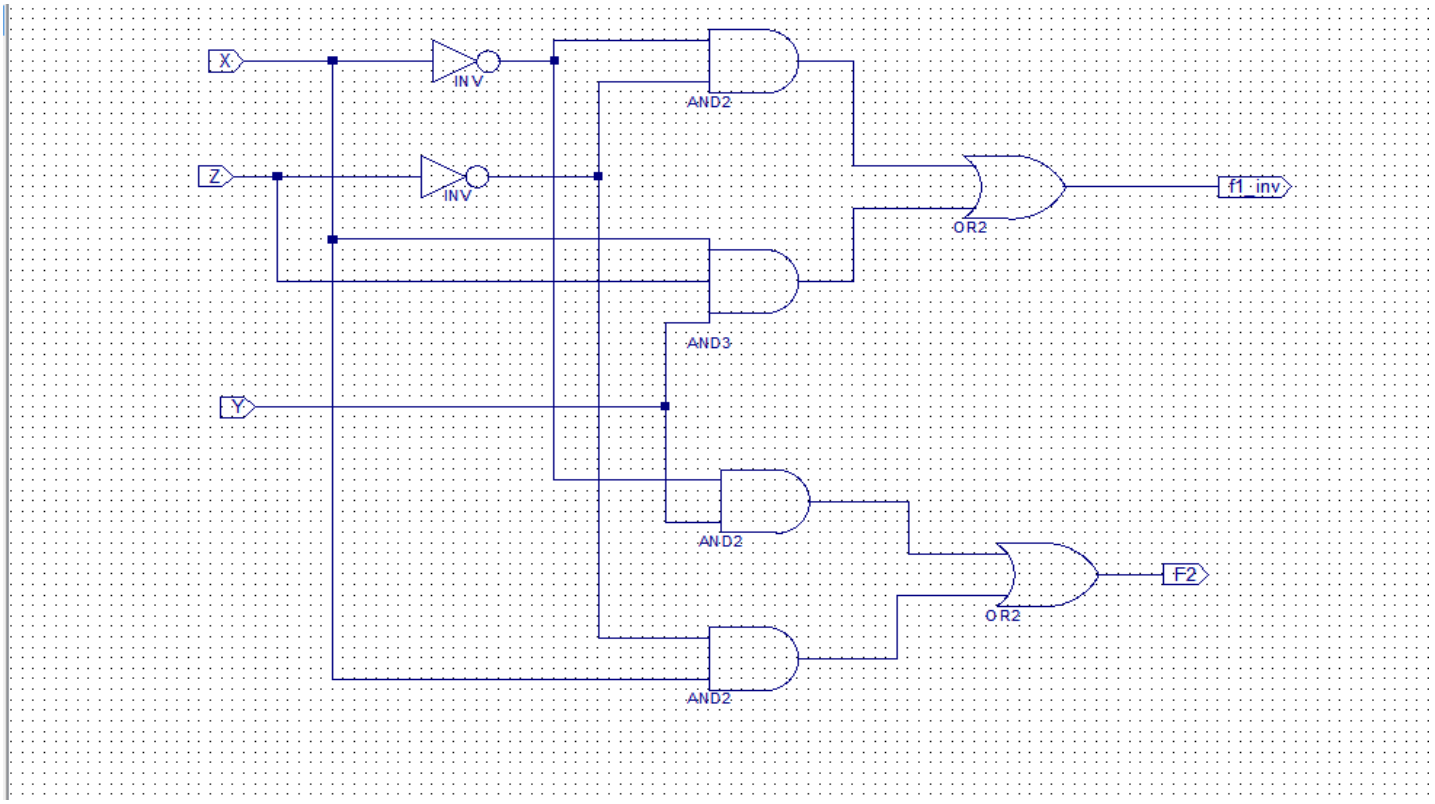
Λύση:

(α) Απο τον πίνακα υπολογίζω τις αρνήσεις των συναρτήσεων όπου

f1' = x'y'z' + x'yz' + x'yz' + xyz και f2' = x'y'z' + x'y'z + xyz. Οι συναρτήσεις βρίσκονται σε μορφή άθροισμα γινομένων



(β) Ελαχιστοποιώντας τους κύβους παίρνουμε τις f1' = x' z' + xyz και f2 = x'y + xz'. Δεν βρήκα τρόπο ώστε η f2 να γραφτεί συναρτήσει της f1, οπότε το τελικό κύκλωμα που προκύπτει



Άσκηση 4: Περιγράψτε σε Verilog το κύκλωμα ενός σειριακού πολλαπλασιαστή 8-bit Ολίσθησης, Πρόσθεσης. Σχεδιάζοντας και το κατάλληλο Πλαίσιο Δοκιμής για το κύκλωμα, το οποίο θα εφαρμόζει την σειριακή είσοδο, και θα μετράει κατάλληλα τον αριθμό των κύκλων για το αποτέλεσμα, παρουσιάστε τις κυμματομορφές, και τα αποτελέσματα της προσομείωσης δυο διαδοχικών πολλαπλασιασμών

Λύση:

Η υλοποίηση βασίζεται στο σχήμα των διαφανειών για τον shift-add-multiplier. Κάθε πολλαπλασμός N-bit αριθμών διαρκεί N κύκλους. Όσο

διαρκεί ένας πολλαπλασμιασμός το σήμα MULT_EN είναι 1. Για 0 κάνει reset στο κύκλωμα του πολλαπλασιαστή, αρχικοποιώντας τα registers και ανανεώνοντας τα inputs

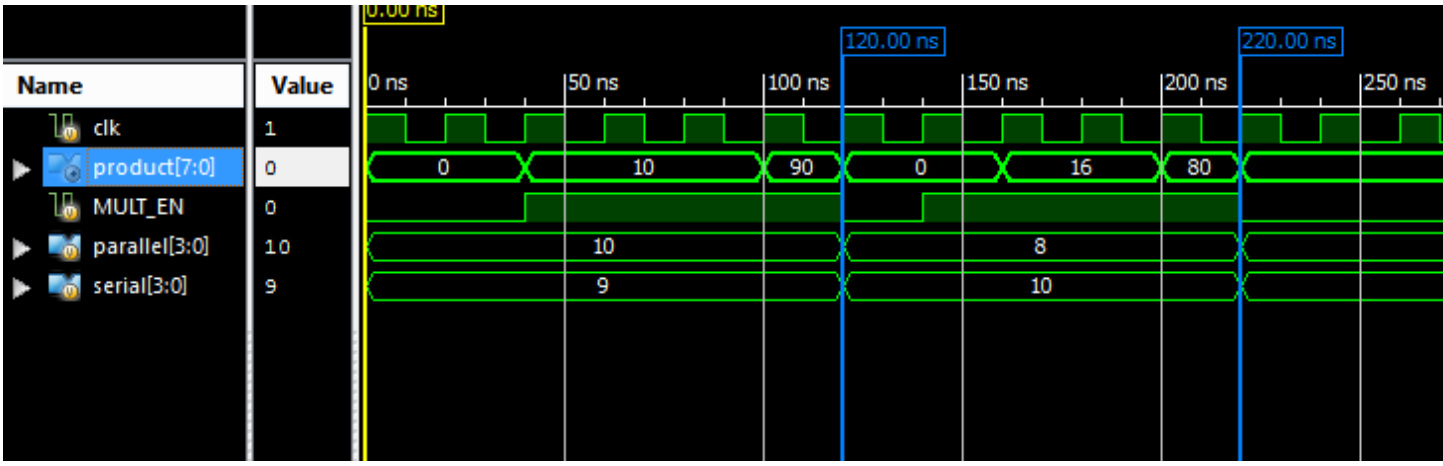
Ανάμεσα σε 2 διαδοχικούς πολλαπλασιασμούς, ρίχνω το MULT_EN για έναν κύκλο. Αυτό θα μπορούσα να το κάνω διαφορετικά χρησιμοποιώντας ένα input START εκτός από το MULT_EN, ώστε να γίνεται reset το κύκλωμα

Ακολοθεί ο κώδικας, το testbench και οι κυματομορφές

```
1  `timescale 1ns / 1ps
2
3  module shift_add_mult(clk, parallel_IN, serial_IN, MULT_EN, product);
4
5  parameter N = 4;
6  input wire clk;
7  input wire [N-1:0] parallel_IN;
8  input wire [N-1:0] serial_IN;
9  input wire MULT_EN;
10 output reg [2*N-1:0] product;
11
12 reg [N-1:0] serial_IN_SAVED;
13 reg [2*N-1:0] parallel_IN_SAVED;
14
15 wire [2*N-1:0] add_to_product;
16
17 /*
18 memory part
19 */
20
21 always @(posedge clk) begin
22     if (MULT_EN == 0) begin
23       parallel_IN_SAVED = {4'b0, parallel_IN};
24       serial_IN_SAVED = serial_IN;
25     end
26     else begin
27       parallel_IN_SAVED =   {parallel_IN_SAVED[2*N-2:0], 1'b0};
28       serial_IN_SAVED = {1'b0, serial_IN_SAVED[ N-1:1]};
29     end
30 end
```

```
/*
pass data to add if serial[i] is 1
*/
genvar i;
generate
  for (i = 0; i < 2*N; i = i + 1) begin:m
    assign  add_to_product[i] = parallel_IN_SAVED[i] & serial_IN_SAVED[0];
  end
endgenerate
/*
save each add until adding
only zeros
*/
always @(posedge clk) begin
  if (MULT_EN == 0)
    product = 0;
  else
    product = product + add_to_product;

end

endmodule

```

```
`timescale 1ns / 1ps
`define  PERIOD 20//ns
/*
thelw na ipologisw to 10x9
*/
module testB;

parameter N = 4;
reg [N-1:0] parallel;
reg [N-1:0] serial;
wire [2*N-1:0] product;
reg clk, MULT_EN;

shift_add_mult shift_add_mult0(clk, parallel, serial, MULT_EN, product );
initial begin
  clk = 1;  MULT_EN = 0; parallel = 4'b1010; serial = 4'b1001;
  #40 MULT_EN = 1;
  #(`PERIOD*N) MULT_EN = 0; parallel = 4'b1000; serial = 4'b1010;
  #(`PERIOD) MULT_EN = 1;
  #(`PERIOD*N) MULT_EN = 0; parallel = 4'b1111; serial = 4'b1111;
end

always
  #(`PERIOD/2) clk = ~clk;

endmodule

```

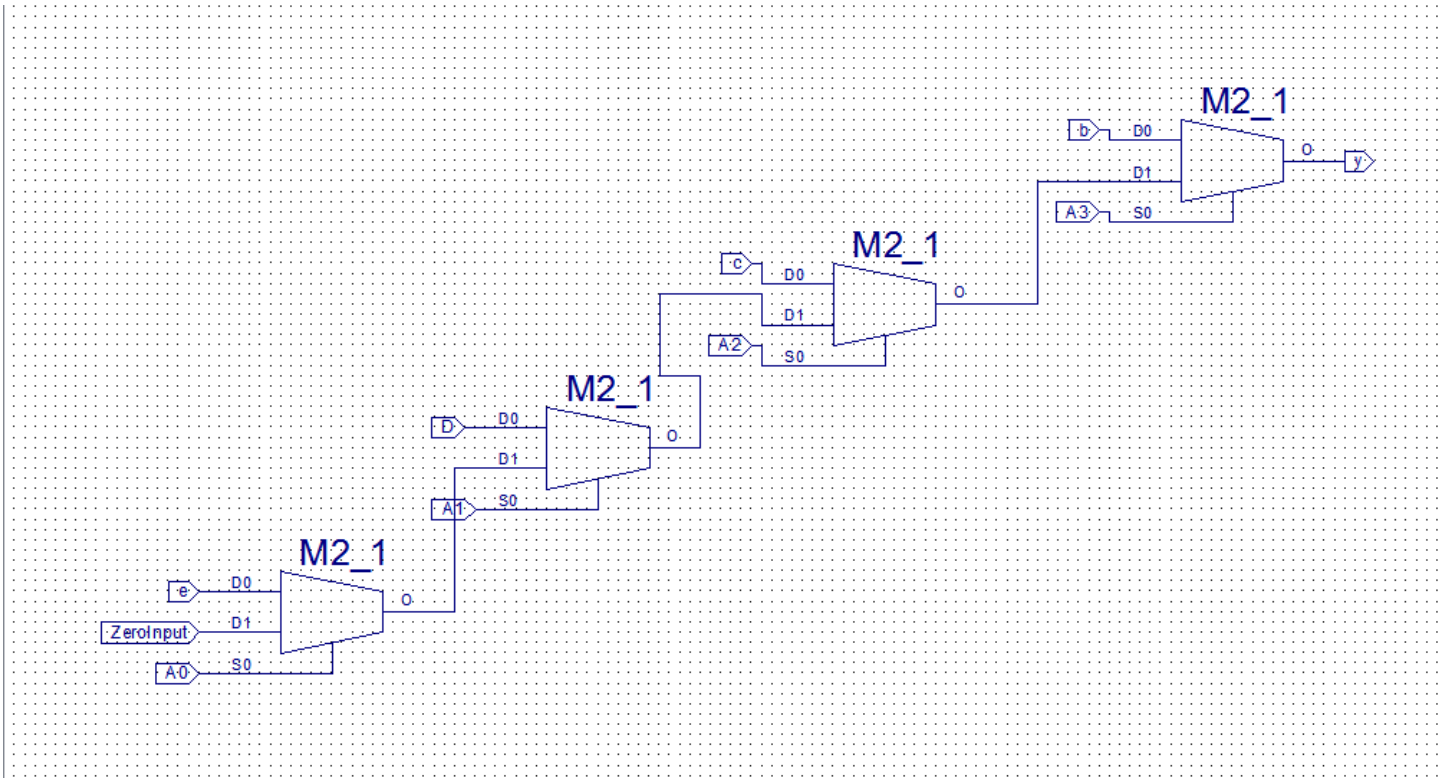


Άσκηση 5: Για τις παρακάτω περιγραφές κυκλωμάτων σε Verilog, παρουσιάστε την αντίστοιχη υλοποίηση τους σε κυκλωματική μορφή, χρησιμοποιώντας πολυπλέκτες 2 ή 4 εισόδων, βασικές λογικές πύλες, και FF, με ή χωρίς ασύγχρονη αρχικοποίηση

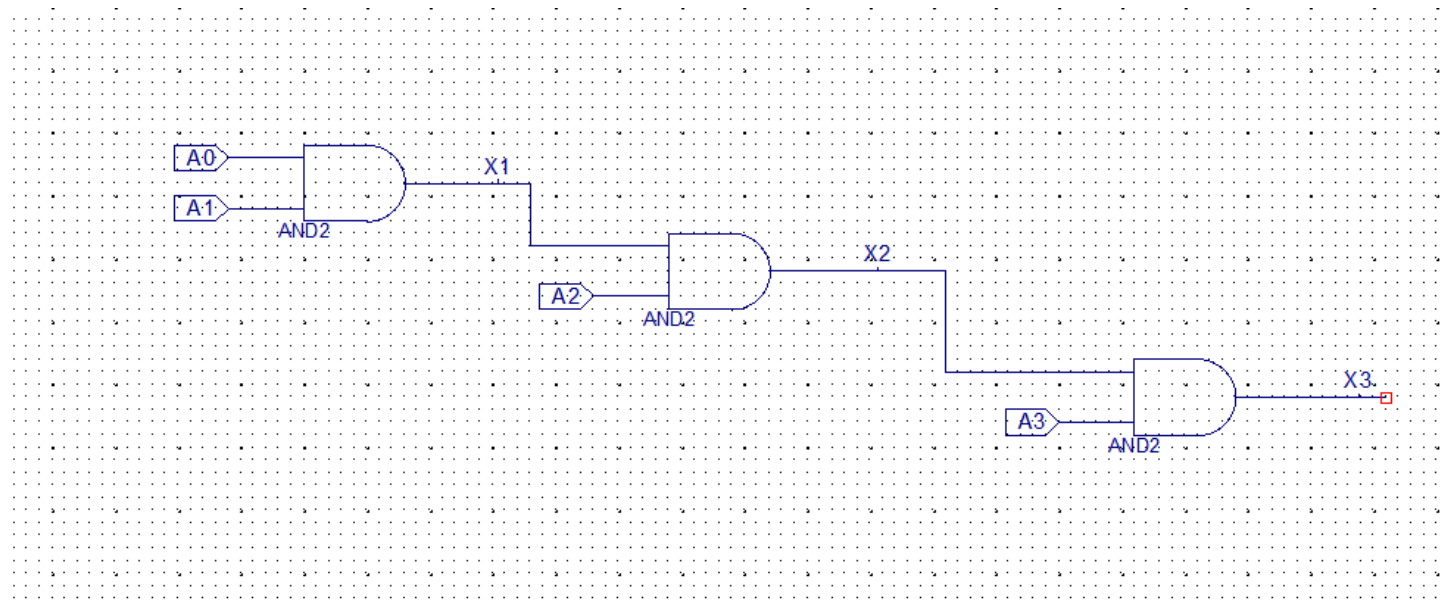
Λύση:

Η υλοποίηση γίνεται με το schematics του ise

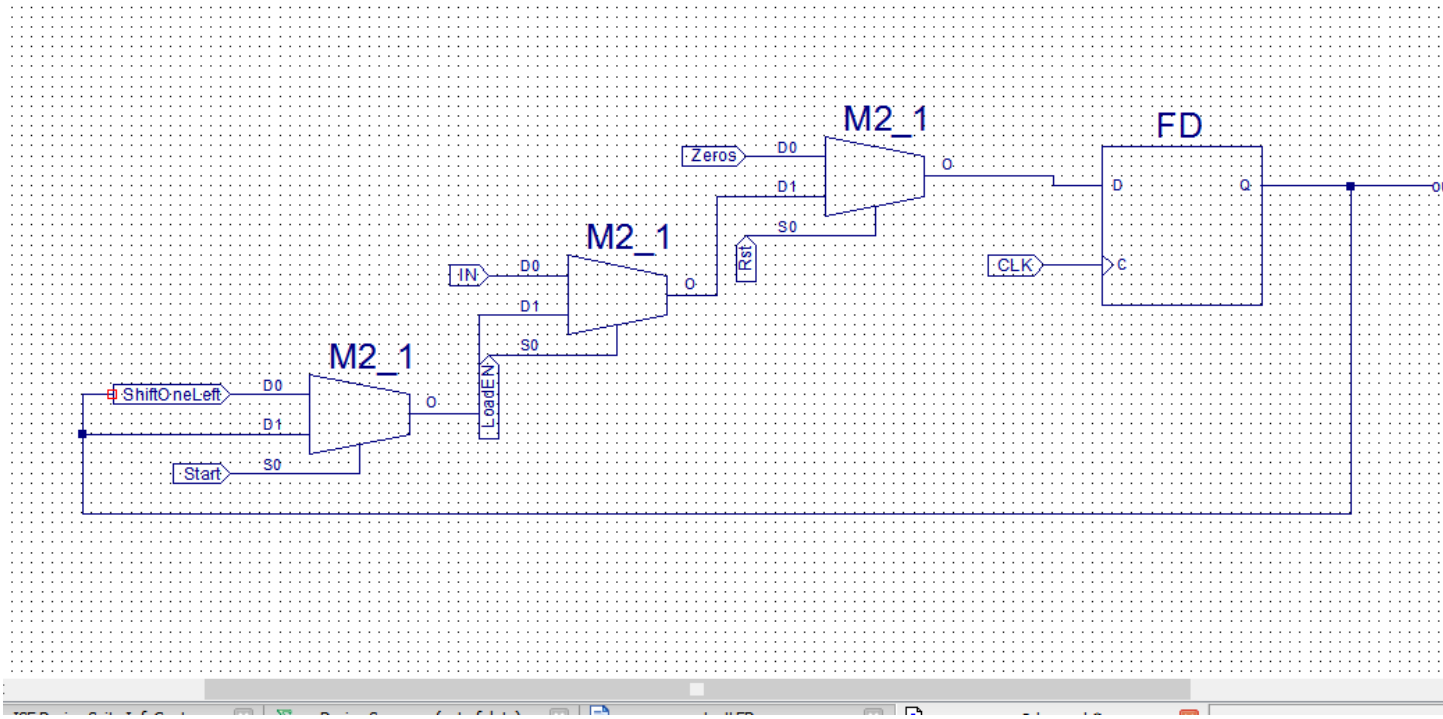
1ο


M2_1
M2_1
M2_1
M2_1
b
D0
D1
A3
S0
O
y
c
D0
D1
A2
S0
O
D
D0
D1
A1
S0
O
e
D0
ZeroInput
D1
A0
S0
O

2o


A0
A1
AND2
X1
A2
AND2
X2
A3
AND2
X3

3o


M2_1
FD
Zeros
D0
D1
S0
O
Rst
M2_1
IN
D0
D1
S0
O
M2_1
ShiftOneLeft
D0
D1
O
Start
S0
LoadEN
D
Q
CLK
C