10/4/2017

3ή Εργαστηριακή Άσκηση

ΔΗΜΙΟΥΡΓΊΑ ΜΙΆΣ POST-INCREMENT, PRE-DECREMENT ΣΤΟΊΒΑΣ ΣΕ VHDL

(Behavioral and Structural VHDL)

24/03/2017

Ομάδα LAB20332005

ΧΡΗΣΤΟΣ ΖΗΣΚΑΣ 2014030191
ΠΑΝΑΓΙΩΤΗΣ ΣΑΒΒΑΙΔΗΣ 2013030180

Σκοπός εργαστηριακής άσκησης

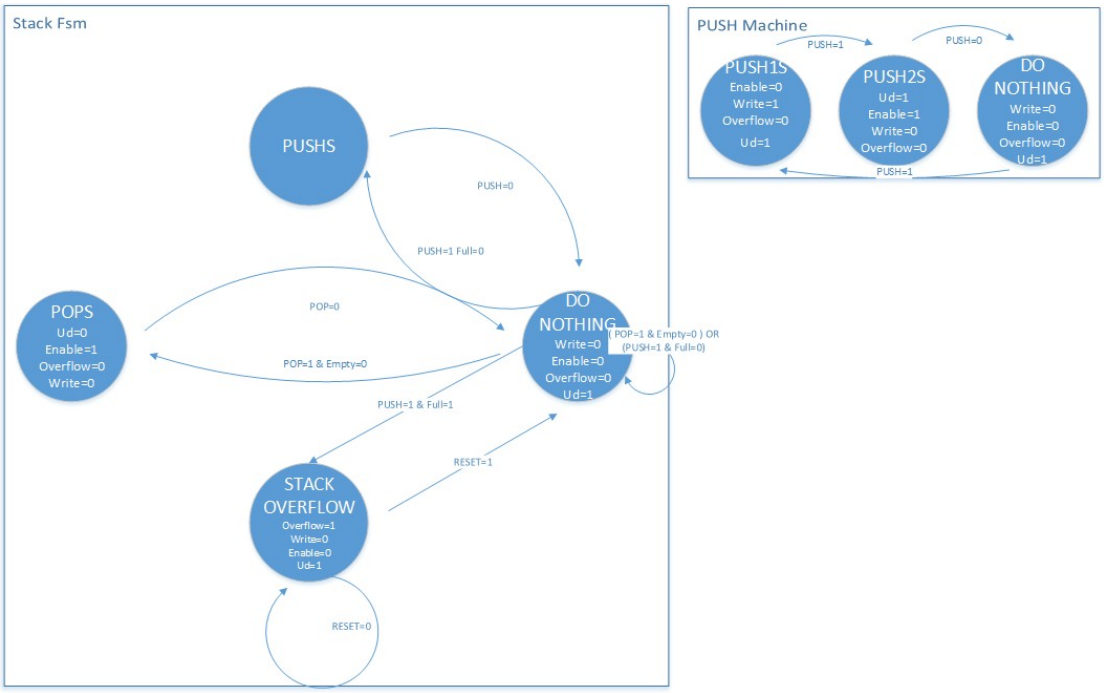
Είναι η περαιτέρω εξοικείωση με τη γλώσσα περιγραφής υλικού VHDL(VHSIC Hardware Description Language) όχι μόνο σε structural μορφή ,αλλά και σε behavioral σχεδίαση σε προχωρημένη σχεδίαση. Ακόμη πραγματοποιείται το ξεκίνημα κατασκευής δομής αποθήκευσης με δείκτη λεγόμενη ως στοίβα (οι δείκτες ονομάζονται Stack Pointer η Top of Stack (TOS)),με το χαρακτηριστικό που την διέπει να είναι πως η τελευταία εισαγωγή είναι η πρώτη έξοδος. Το ολοκληρωμένο σχεδιαστικό κύκλωμα αποτυπώνει την υλοποίηση της βασικής λειτουργικότητας μια post-increment, pre-decrement στοίβας. Το κύκλωμα θα εκτελεί μόνο τις απλές λειτουργίες Push/Pop(εκχώρηση, εξαγωγή) στοιχείων. Τέλος στα LED του αναπτυξιακού Basys 2 θα απεικονίζεται σε δυαδική μορφή το στοιχείο του TOS(top of stack) ενώ στα 7segment display θα απεικονίζονται χρήσιμες πληροφορίες (π.χ. Empty, Full, OVF)

Προεργασία

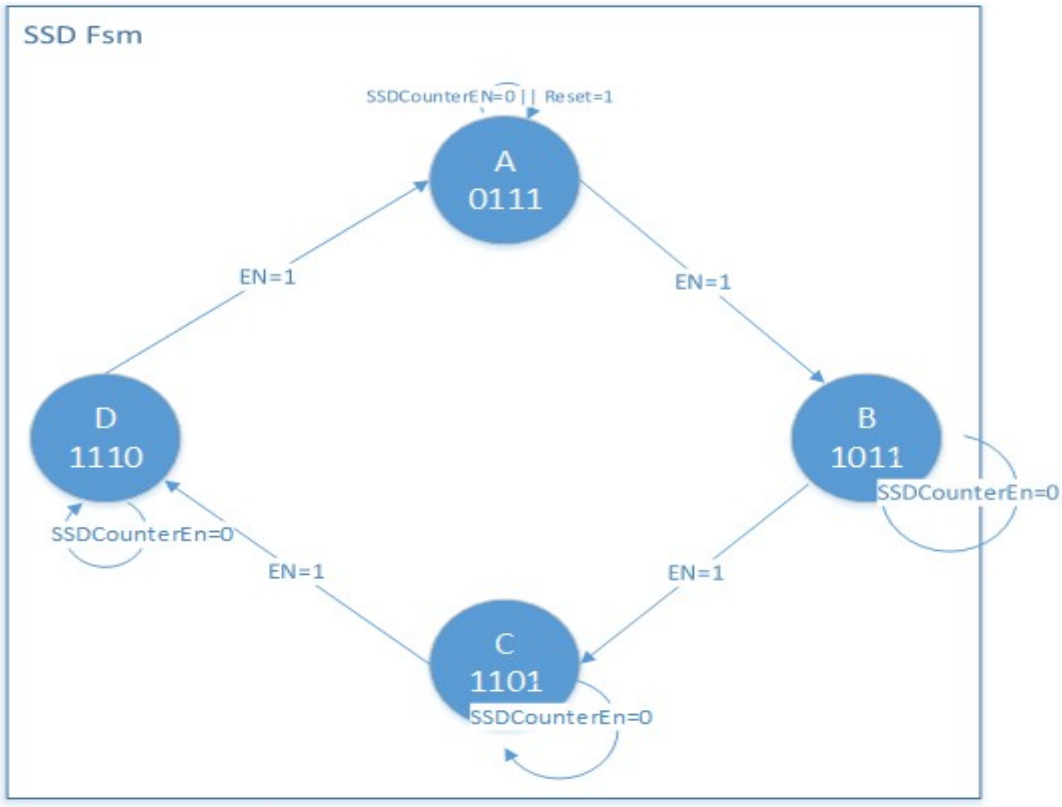
Η προετοιμασία του κυκλώματος απαιτεί :

- την σχεδίαση block diagram της συνολικής σχεδίαση της μνήμης αλλά και των υποσυστημάτων για την αποτύπωση των συμπερασμάτων στα led
- σχεδίαση με τις αλληλεπιδράσεις μεταξύ των υποκυκλωμάτων.
- αναλυτικά διαγράμματα των FSM για ανάλυση των εισόδων/εξόδων και των μεταβάσεων/
- γνώση των συσχετίσεων της FSM με την μνήμη.

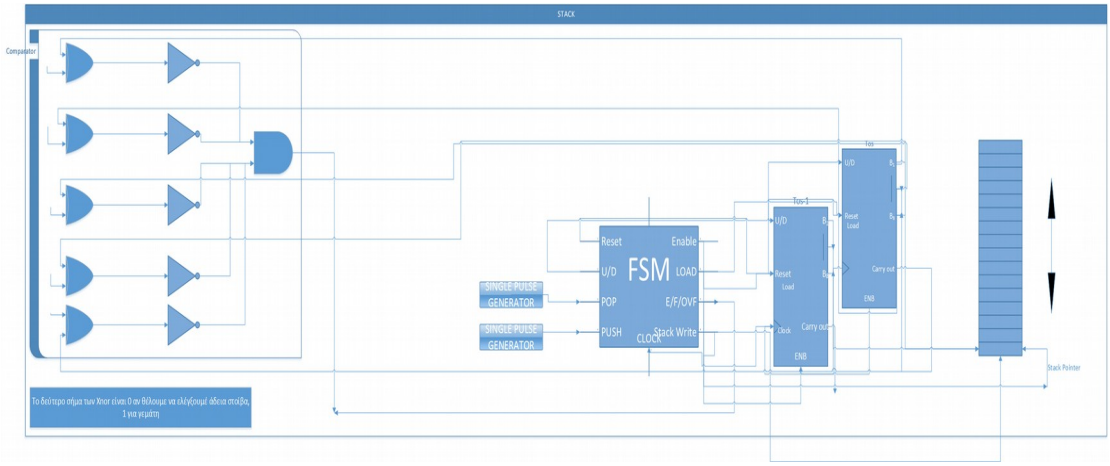
Μηχανή Πεπερασμένων Καταστάσεων για τη μνήμη (Stack FSM)



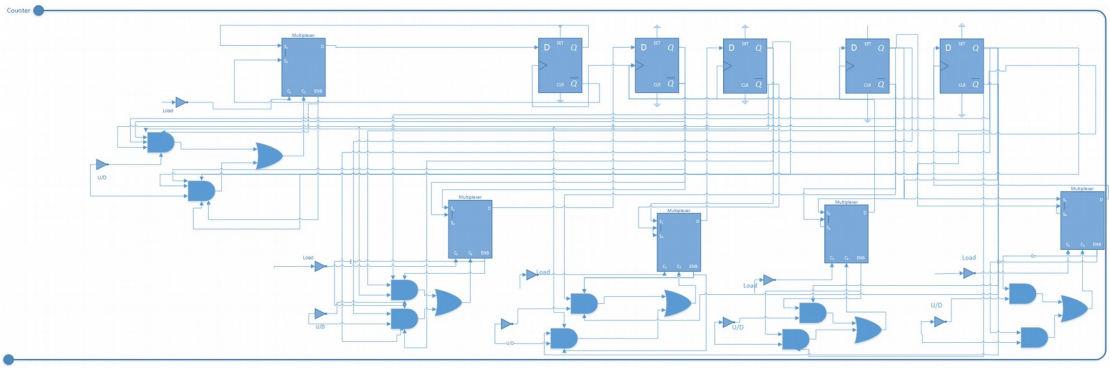
Μηχανή Πεπερασμένων Καταστάσεων για το 7Segment (7segment FSM)



Υλοποίηση Ολοκληρωμένου κυκλώματος Στοίβας (Stack)



Κύκλωμα μετρητή 5 bit (Counter 5-bit)



Περιγραφή

Το κύκλωμα διεκπεραιώνει βασικές λειτουργίες εισαγωγής και εξαγωγής στοιχείων στη δομή της στοίβας. Η στοίβα διαθέτει 32 θέσεις στην μνήμη και αποθηκεύονται αριθμοί των 8 bit. Η στοίβα αρχικοποιείται με την λειτουργία Reset. Με την μεσολάβηση των διακοπτών επιλέγεται ο αριθμός που επιθυμείται για εγγραφή. Όταν υπάρχει εισαγωγή, τότε τοποθετούμε το στοιχείο στη στοίβα και μετά αυξάνουμε τον Stack Pointer -η αλλιώς TOS. Όταν υπάρχει εξαγωγή, τότε μειώνουμε τον Stack Pointer και μετά εξάγουμε το στοιχείο από τη στοίβα(χάνεται το τελευταίο στοιχείο)

Ουσιαστικά η δομή διαθέτει :

Push=>Εισαγωγή ενός αριθμού των 8 bit στη στοίβα.

***Pop*=>**Αφαίρεση ενός αριθμού των 8 bit από την στοίβα – όχι αφαίρεση σαν πράξη.

Reset=> Όταν αρχικοποιείται η στοίβα ο δείκτης της στοίβας αρχικοποιείται στη θέση 0 και τα περιεχόμενα τις στοίβας γίνονται μηδενικά. Επειδή η στοίβα είναι Post-increment η θέση θα πανωγραφεί όταν εκτελεστεί μια εκχώρηση.

Όταν η στοίβα υπερβεί το μέγιστο αριθμό στοιχείων μέσω διαδοχικών εκχωρήσεων, το σύστημα σταματά να ανταποκρίνεται σε κάθε είσοδο εξαιρουμένης της αρχικοποίησης Reset.

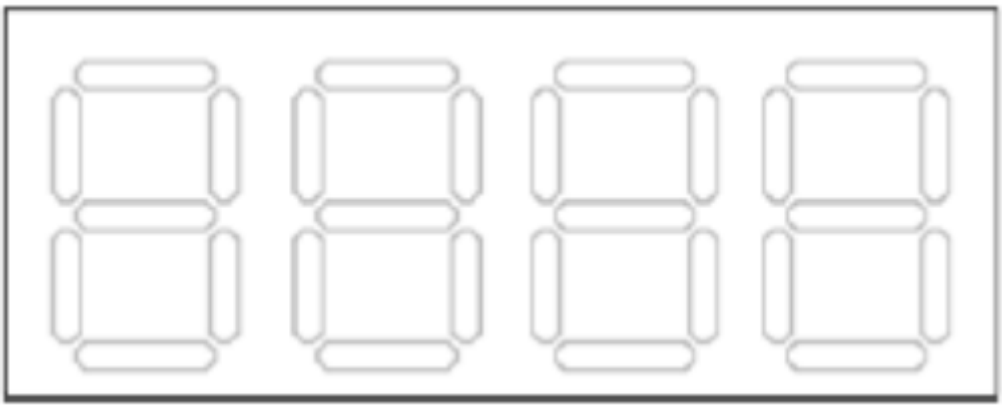
Επιπρόσθετα, εκτός από την δεικτοδότηση της στοίβας για την τρέχουσα θέση της στοίβας, χρησιμοποιείται και επιπλέον δείκτης για την θέση της στοίβας που γράφτηκε τελευταία.(2 δείκτες: 1 στο TOS , 1 στο TOS-1).Οι δείκτες διαμορφώνονται ως μετρητές 5 bit που αυξάνονται σε κάθε εισαγωγή(η σχεδίασή τους πραγματοποιήθηκε σε structural model).Στην 1η εισαγωγή αυξάνεται μόνο ο TOS και σε κάθε επόμενη εισαγωγή αυξάνονται παράλληλα. Ο TOS μεγαλώνει μέχρι τη τελευταία θέση της στοίβας όπου για επόμενο Push το κύκλωμα αδυνατεί να ανταποκρίνεται (κατάσταση υπερχείλισης). Όταν δοθεί σήμα POP τότε ο μετρητής πρώτα μειώνεται κατά μία θέση και μετά αφαιρείται το στοιχείο. Σε γεμάτη στοίβα τότε χάνουμε ένα παραπάνω στοιχείο(το τελευταίο στοιχείο). Σε άδεια στοίβα δεν υφίστανται αλλαγές στη στοίβα .Στη θέση Reset και οι δυο βρίσκονται στη θέση 0.

Οι λειτουργίες αυτές αποτυπώνονται στα κουμπιά της αναδιατασσόμενης συσκευής (FPGA).

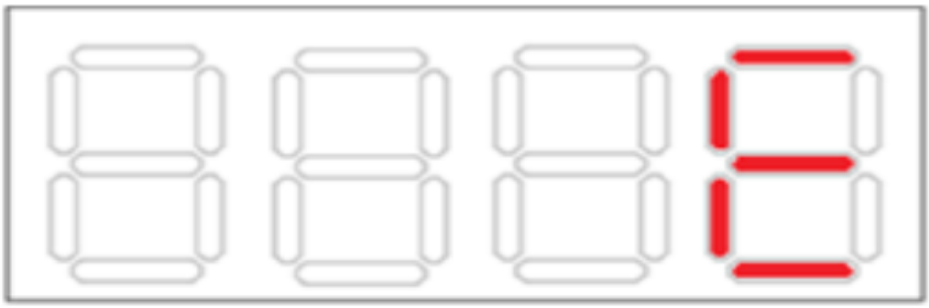
Η λειτουργικότητα του κυκλώματος επαληθεύεται μέσω της απεικόνισης στα Seven Segment Display. Τα Segment λειτουργούν ως εξής:

Υπάρχουν 4 εικόνες για την σωστή εφαρμογή του κυκλώματος:

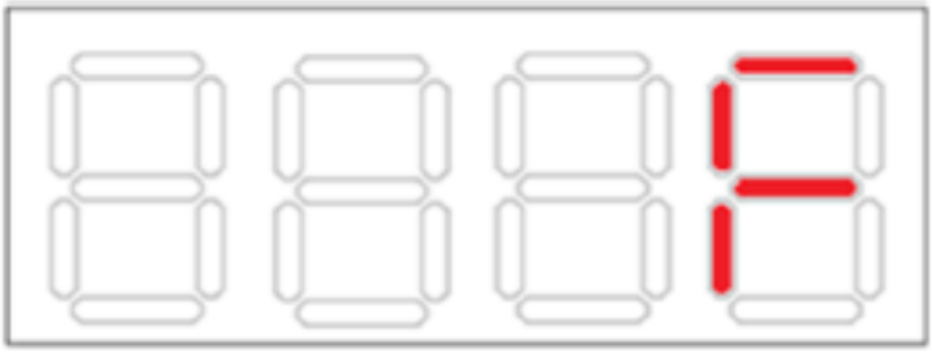
Σβηστό - όταν η στοίβα λειτουργεί σε κανονικό σενάριο(όταν αποφεύγονται οι οριακές καταστάσεις (άδεια,γεμάτη,υπερχείλιση))



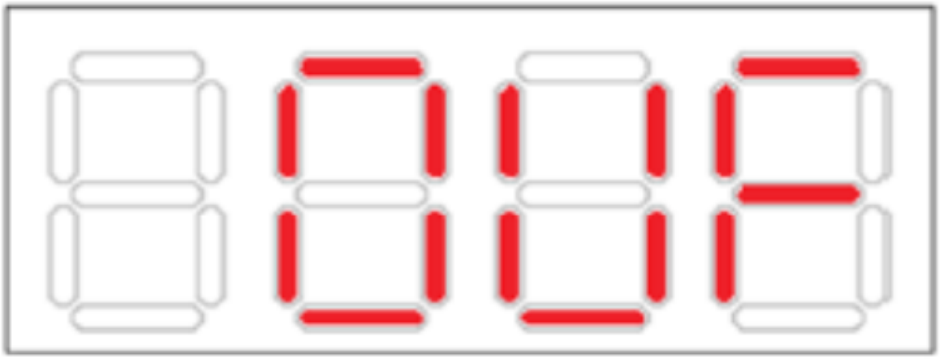
Ανοιχτό – Ανάβει το τελευταίο segment δηλώνοντας Empty(Ισχύει όταν η στοίβα είναι άδεια).



Ανάβει το τελευταίο segment δηλώνοντας Full(Ισχύει όταν η στοίβα είναι γεμάτη).

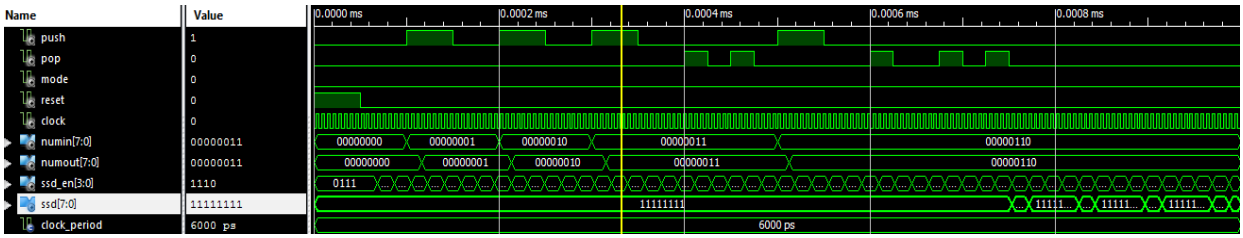


Ανάβουν τα τρία segment δηλώνοντας Overflow(Ισχύει όταν η στοίβα βρίσκεται σε κατάσταση αδράνειας/μη ανταπόκρισης).

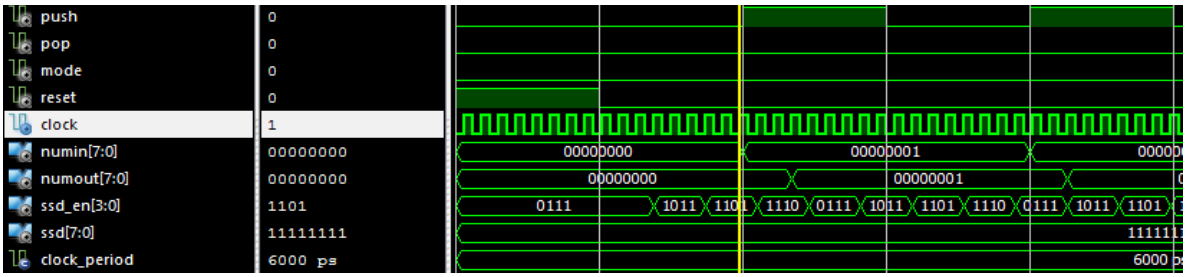


Κυματομορφές-Προσομοίωση

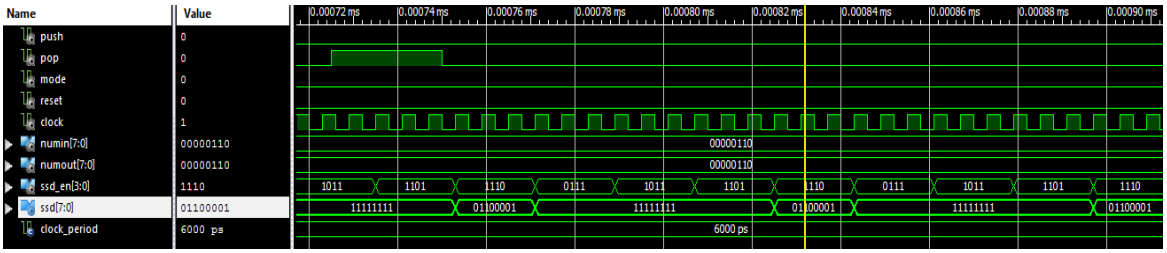
Παρουσιάζουμε τις κυμματομορφές του συστήματος



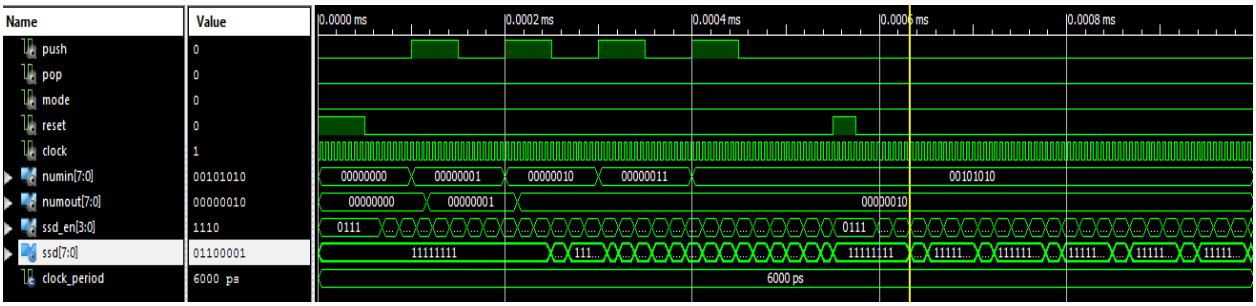
Σύμφωνα με την κυμματομορφή , έχουμε 4 συνολικά Push και 5 συνολικά Pop. Αρχικά θεωρούμαστε σε κατάσταση Reset για τα πρώτα 50 ns. (Ο counter στο κύκλωμα των segment μετράει για 3 κύκλους).



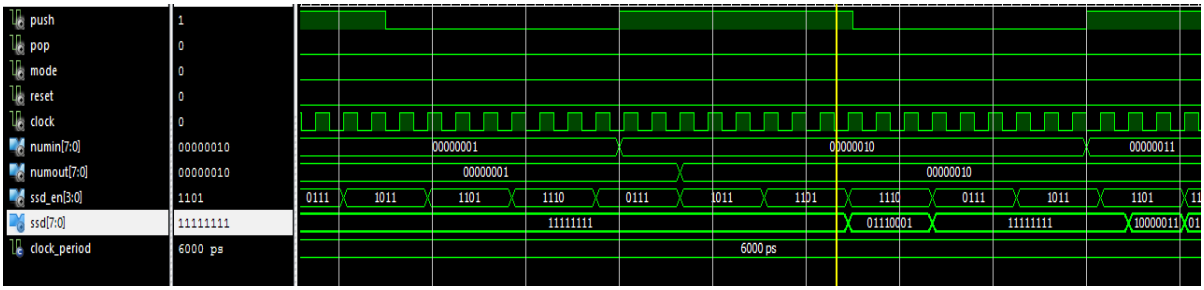
Παρατηρούμε ότι μετά την κατάσταση Reset θα έπρεπε να θεωρούμαστε empty(Το segment να δηλώσει την τρέχουσα κατάσταση). Όμως έχει προλάβει να πατηθεί το push πριν φτάσει το segment στο τελευταίο του control και να εμφανίσει το 01100001(E) οπότε και η “άδεια” κατάσταση έχει προσπελασθεί οπότε και βρισκόμαστε σε κάποια μη ακραία κατάσταση 11111111. Συνεχίζονται οι εκχωρήσεις αριθμών και φθάνουμε στα 2 pop. Παραμένουμε σε κατάσταση 11111111 (Ο Tos-1 βρίσκεται στο 1ο στοιχείο και ο Tos στην 2η θέση-1 κενή μετά το Tos-1) . Εισάγουμε ξανά στοιχείο στη στοίβα. Τελικά εξάγουμε όλα τα στοιχεία από τη στοίβα κατεβάζοντας κάθε φόρα τους δείκτες (μειώνουμε τους μετρητές). Στις πρώτες δυο εξαγωγές οι δείκτες δείχνουν στην ίδια θέση(θέση 0) και στην επόμενη εξαγωγή παρουσιάζεται στα segment η κατάσταση 01100001(E).



Σε άλλη προσομοίωση, θέλουμε να ελέγξουμε τις ακραίες καταστάσεις στις οποίες μεταβαίνει το κύκλωμα και αναλύονται από το Seven segment display (ssd).

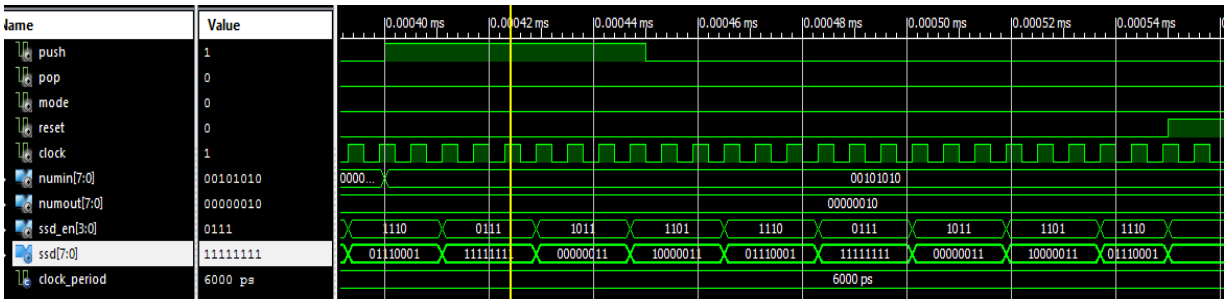


(Θεωρούμε ότι ο συγκριτής που εξετάζει την πληρότητα της στοίβας ελέγχει ότι η στοίβα γεμίζει όταν βρεθεί σε διεύθυνση 00011- Δηλαδή έχουν φορτωθεί 2 στοιχεία ,αφού με 2 στοιχεία ο Tos δείχνει στη θέση 00011)

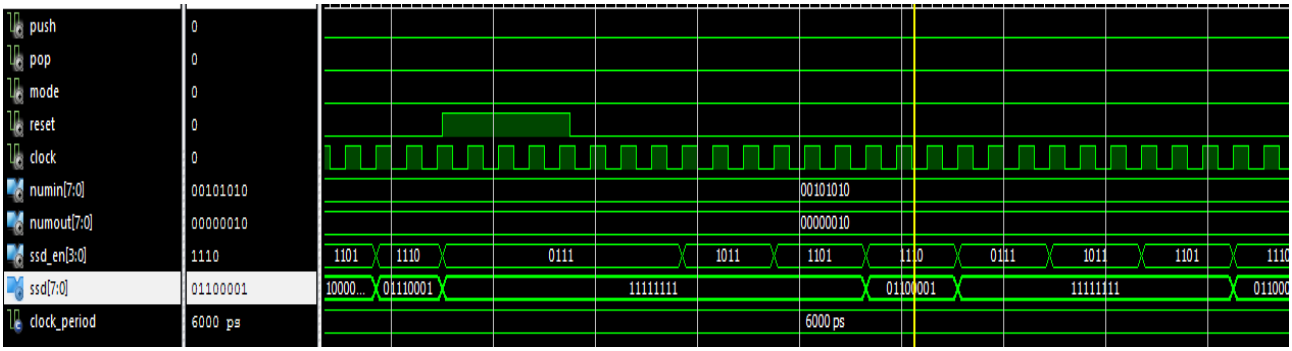


Βλέπουμε ότι η έκφραση της γεμάτης στοίβας αποδίδεται στα display μετά από τα ανάλογα push και συνεχίζοντας για επόμενη εκχώρηση το κύκλωμα κλειδώνει στην κατάσταση “υπερχείλιση”

(Στην υπερχείλιση ενεργοποιούνται το ένα μετά το άλλο τα 3 τελευταία Led του segment δηλαδή 1011,1101,1110 απεικονίζοντας τις ανάλογες εικόνες – 1011(O), 1101(V),1110(F))



Επαναφέρουμε το κύκλωμα σε μηδενισμό μετά την ενεργοποίηση του Reset.



Συμπεράσματα

Η διεκπεραίωση της εργαστηριακής άσκησης καταλήγει στην εξέλιξη συγκεκριμένων σχεδίων πάνω στη οργάνωση και εξάσκηση σε εφαρμογές για σχεδίαση υλικού(εντολές Xilinx) . Η σχεδίαση κυκλώματος με ενεργοποίηση γεννήτριας πυρήνα αποθήκευσης – αλλιώς στοίβα δομής post increment/pre decrement – θεωρείται χρήσιμη για την κατανόηση των χαρακτηριστικών ανάγνωσης και εγγραφής σε αυτή καθώς και την αλληλουχία μεταξύ των μετρητών ως δείκτες στη στοίβα. Εμπειρία αποφέρεται και λόγω της δημιουργίας διαφόρων εργαλείων σχεδίασης(ακολουθιακών και μη) για την ολοκλήρωση του συγκεκριμένου έργου (πολυπλέκτες , decoders,flip-flop) και η καταγραφή του από τα seven segment display οδηγεί στην ολοκλήρωση της σχεδίασης.

Κώδικας

- Στον κώδικα δεν περιλαμβάνονται οι FSM για τα Seven segment display και την στοίβα καθώς επίσης και τα singe pulse generator

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity COUNTER is

   Port ( En : in  STD_LOGIC;

          input : in  STD_LOGIC_Vector(4 downto 0);

          output : out  STD_LOGIC_Vector(4 downto 0);

          Clock : in  STD_LOGIC;

          Load : in  STD_LOGIC;

          U_D : in  STD_LOGIC);
```

```
end COUNTER;

architecture Behavioral of COUNTER is

component MUX4v1 is
   Port ( Q0 : in  STD_LOGIC;
          Q1 : in  STD_LOGIC;
          Q2 : in  STD_LOGIC;
          Q3 : in  STD_LOGIC;
          Sign0 : in  STD_LOGIC;
          Sign1 : in  STD_LOGIC;
          OUTM : out  STD_LOGIC);
end component;

component FlipFlop is
   Port ( Clock : in  STD_LOGIC;
          D : in  STD_LOGIC;
          Q : out  STD_LOGIC);
end component;
signal mout1,mout2,mout3,mout4,mout5:std_logic;
signal fout1,fout2,fout3,fout4,fout5:std_logic;
signal ms1,ms2,ms3,ms4,ms5:std_logic;
signal nfout1,nfout2,nfout3,nfout4,nfout5:std_logic;
signal nload1,nload2,nload3,nload4,nload5:std_logic;
signal nUD:std_logic;
begin
```

```
nUD<=NOT U_D;

ms1<=(En AND U_D) or (En AND nUD);

nfout1<=not fout1;

nload1<=not Load;

M1: MUX4v1 PORT
MAP(Q0=>fout1,Q1=>nfout1,Q2=>input(0),Q3=>input(0),Sign1=>nload1,Sign0=>ms1,OUTM
=>mout1);

F1: FlipFlop PORT MAP(D=>mout1,Clock=>Clock,Q=>fout1);


nload2<=not Load;

nfout2<=not fout2;

ms2<=(En AND fout1 AND U_D) or (En AND nfout1 AND nUD );

M2: MUX4v1 PORT
MAP(Q0=>fout2,Q1=>nfout2,Q2=>input(1),Q3=>input(1),Sign1=>nload2,Sign0=>ms2,OUTM
=>mout2);

F2: FlipFlop PORT MAP(D=>mout2,Clock=>Clock,Q=>fout2);


nload3<=not Load;

nfout3<=not fout3;

ms3<=(En AND fout1 AND fout2 AND U_D) or (En AND nfout1 AND nfout2 AND nUD);

M3: MUX4v1 PORT
MAP(Q0=>fout3,Q1=>nfout3,Q2=>input(2),Q3=>input(2),Sign1=>nload3,Sign0=>ms3,OUTM
=>mout3);

F3: FlipFlop PORT MAP(D=>mout3,Clock=>Clock,Q=>fout3);


nfout4<=not fout4;

nload4<=not Load;
```

```
ms4<=(En AND fout1 AND fout2 AND fout3 AND U_D) or (En AND nfout1 AND nfout2 AND
nfout3 AND nUD);

M4: MUX4v1 PORT
MAP(Q0=>fout4,Q1=>nfout4,Q2=>input(3),Q3=>input(3),Sign1=>nload4,Sign0=>ms4,OUTM
=>mout4);

F4: FlipFlop PORT MAP(D=>mout4,Clock=>Clock,Q=>fout4);


nload5<=not Load;

nfout5<=not fout5;

ms5<=(En AND fout1 AND fout2 AND fout3 AND fout4 AND U_D) or (En AND nfout1 AND
nfout2 AND nfout3 AND nfout4 AND nUD);

M5: MUX4v1 PORT
MAP(Q0=>fout5,Q1=>nfout5,Q2=>input(4),Q3=>input(4),Sign1=>nload5,Sign0=>ms5,OUTM
=>mout5);

F5: FlipFlop PORT MAP(D=>mout5,Clock=>Clock,Q=>fout5);


output(0)<=fout1;

output(1)<=fout2;

output(2)<=fout3;

output(3)<=fout4;

output(4)<=fout5;


end Behavioral;
```

MUX2V1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity MUX2v1 is
   Port ( Q0 : in  STD_LOGIC;
          Q1 : in  STD_LOGIC;
          Sign : in  STD_LOGIC;
          OUTM : out  STD_LOGIC);
end MUX2v1;


architecture Behavioral of MUX2v1 is
signal nsign:std_logic;
begin
nsign<=not Sign;
OUTM<=(Q0 AND nsign) OR (Q1 AND Sign);
end Behavioral;
```

MUX4V1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;




entity MUX4v1 is
   Port ( Q0 : in  STD_LOGIC;
          Q1 : in  STD_LOGIC;
          Q2 : in  STD_LOGIC;
          Q3 : in  STD_LOGIC;
          Sign0 : in  STD_LOGIC;
          Sign1 : in  STD_LOGIC;
          OUTM : out  STD_LOGIC);
end MUX4v1;


architecture Behavioral of MUX4v1 is
Signal out1,out2:std_logic;
component MUX2v1 is
   Port ( Q0 : in  STD_LOGIC;
          Q1 : in  STD_LOGIC;
          Sign : in  STD_LOGIC;
          OUTM : out  STD_LOGIC);
end component;
```

```
begin
MUX2: MUX2v1 PORT MAP(Q0=>Q2,Q1=>Q3,Sign=>Sign0,OUTM=>out2);
MUX1: MUX2v1 PORT MAP(Q0=>Q0,Q1=>Q1,Sign=>Sign0,OUTM=>out1);
MUX3: MUX2v1 PORT MAP(Q0=>out1,Q1=>out2,Sign=>Sign1,OUTM=>OUTM);

end Behavioral;
```

Flip-Flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FlipFlop is
   Port ( Clock : in  STD_LOGIC;
          D : in  STD_LOGIC;
          Q : out  STD_LOGIC);
end FlipFlop;

architecture Behavioral of FlipFlop is

Begin
Process
            Begin
                        WAIT UNTIL Clock'EVENT AND Clock='1';
                        Q<=D;
```

```
                    END PROCESS;
end Behavioral;
```

Comparator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Comparator_Equality is
   Port ( B1 : in  STD_LOGIC_VECTOR (4 downto 0);
          B2 : in  STD_LOGIC_VECTOR (4 downto 0);
          Y1 : out  STD_LOGIC);
end Comparator_Equality;

architecture Behavioral of Comparator_Equality is
signal xnout0,xnout1,xnout2,xnout3,xnout4:std_logic;
signal andout0,andout1,andout2,equal_out:std_logic;
component xnor_gate is
   Port ( B1 : in  STD_LOGIC;
          B2 : in  STD_LOGIC;
          Y1 : out  STD_LOGIC);
end component;
component and_gate is
   Port ( B1 : in  STD_LOGIC;
          B2 : in  STD_LOGIC;
```

```
        Y1 : out  STD_LOGIC);
end component;



begin
gate_xnor0: xnor_gate PORT MAP(B1=>B1(0),B2=>B2(0),Y1=>xnout0);
gate_xnor1: xnor_gate PORT MAP(B1=>B1(1),B2=>B2(1),Y1=>xnout1);
gate_xnor2: xnor_gate PORT MAP(B1=>B1(2),B2=>B2(2),Y1=>xnout2);
gate_xnor3: xnor_gate PORT MAP(B1=>B1(3),B2=>B2(3),Y1=>xnout3);
gate_xnor4: xnor_gate PORT MAP(B1=>B1(4),B2=>B2(4),Y1=>xnout4);

gate_and0 : and_gate PORT MAP(B1=>xnout0,B2=>xnout1,Y1=>andout0);
gate_and2 : and_gate PORT MAP(B1=>andout0,B2=>xnout2,Y1=>andout1);
gate_and3 : and_gate PORT MAP(B1=>andout1,B2=>xnout3,Y1=>andout2);
gate_and4 : and_gate PORT MAP(B1=>andout2,B2=>xnout4,Y1=>equal_out);

Y1<=equal_out;

end Behavioral;
```

Behavioral Counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BehCounter is
```

```
   Port ( Clk : in  STD_LOGIC;
          Rst : in  STD_LOGIC;
                                oo : out integer;
                                Q : out  STD_LOGIC);
end BehCounter;

architecture Behavioral of BehCounter is
signal outs: integer := 0;
begin
counter: process(Rst, Clk, outs)
begin
 If (Rst = '1') then outs<=0; Q<='0';
 elsif (rising_edge(Clk)) then
                                outs <=outs + 1;
                                Q<='0';
 end if;
 if outs=3 then outs<=0;Q<='1';
 end if;
end process;


oo <= outs;

end Behavioral;
```

MUX32V8

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX32x8 is
   Port ( INPUT : in  STD_LOGIC_VECTOR (31 downto 0);
          control : in  STD_LOGIC_VECTOR(1 downto 0);
          OUTPUT : out  STD_LOGIC_VECTOR (7 downto 0));
end MUX32x8;

architecture Behavioral of MUX32x8 is

begin

 mux: process (INPUT, control)
 begin
  IF(control = "00") then OUTPUT(0) <= INPUT(0);
                                                    OUTPUT(1) <= INPUT(1);
                                                    OUTPUT(2) <= INPUT(2);
                                                    OUTPUT(3) <= INPUT(3);
                                                    OUTPUT(4) <= INPUT(4);
                                                    OUTPUT(5) <= INPUT(5);
                                                    OUTPUT(6) <= INPUT(6);
                                                    OUTPUT(7) <= INPUT(7);
  ELSIF(control = "01") then OUTPUT(0) <= INPUT(8);
```

```
                                                    OUTPUT(1) <= INPUT(9);
                                                    OUTPUT(2) <= INPUT(10);
                                                    OUTPUT(3) <= INPUT(11);
                                                    OUTPUT(4) <= INPUT(12);
                                                    OUTPUT(5) <= INPUT(13);
                                                    OUTPUT(6) <= INPUT(14);
                                                    OUTPUT(7) <= INPUT(15);
ELSIF(control = "10") then OUTPUT(0) <= INPUT(16);
                                                    OUTPUT(1) <= INPUT(17);
                                                    OUTPUT(2) <= INPUT(18);
                                                    OUTPUT(3) <= INPUT(19);
                                                    OUTPUT(4) <= INPUT(20);
                                                    OUTPUT(5) <= INPUT(21);
                                                    OUTPUT(6) <= INPUT(22);
                                                    OUTPUT(7) <= INPUT(23);
ELSIF(control = "11") then OUTPUT(0) <= INPUT(24);
                                                    OUTPUT(1) <= INPUT(25);
                                                    OUTPUT(2) <= INPUT(26);
                                                    OUTPUT(3) <= INPUT(27);
                                                    OUTPUT(4) <= INPUT(28);
                                                    OUTPUT(5) <= INPUT(29);
                                                    OUTPUT(6) <= INPUT(30);
                                                    OUTPUT(7) <= INPUT(31);
ELSE OUTPUT <= "11111111";
END IF;
END PROCESS;
```

```
end Behavioral;
```

Coder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Coder is
   Port ( Input : in  STD_LOGIC_VECTOR (3 downto 0);
          Output : out  STD_LOGIC_VECTOR (1 downto 0));
end Coder;

architecture Behavioral of Coder is
begin

coder : process(Input)

begin
          IF(Input = "1110") THEN Output <= "00";
          ELSIF(Input = "1101") THEN Output <= "01";
          ELSIF(Input = "1011") THEN Output <= "10";
          ELSIF(Input = "0111") THEN Output <= "11";
          ELSE Output <= "00";
          END IF;
end process;
```

```
end Behavioral;
```

Decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity Decoder is
   Port ( E : in  STD_LOGIC;
          F : in  STD_LOGIC;
                                OV : in  STD_LOGIC;
          segmentLED : out  STD_LOGIC_VECTOR (31 downto 0));
end Decoder;

architecture Behavioral of Decoder is
begin

DEcoder: process(E, F, OV)
begin
            IF(E='1') THEN segmentLED <= "11111111111111111111111101100001";
            ELSIF(OV='1') THEN segmentLED <= "11111111000000111000001101110001";
            ELSIF(F='1') THEN segmentLED <= "11111111111111111111111101110001";
            ELSE segmentLED <= "11111111111111111111111111111111";
            END IF;
end process;
end Behavioral;
```

TopModule

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

entity FSMmode0 is
   Port ( Push : in  STD_LOGIC;
          Pop : in  STD_LOGIC;
          Mode: in  STD_LOGIC;
          Reset : in  STD_LOGIC;
          Clock : in  STD_LOGIC;
          NUMIN : in  STD_LOGIC_VECTOR (7 downto 0);
          NUMOUT : out  STD_LOGIC_VECTOR (7 downto 0);
          SSD_EN : out  STD_LOGIC_VECTOR (3 downto 0);
          SSD : out  STD_LOGIC_VECTOR (7 downto 0));
end FSMmode0;

architecture Behavioral of FSMmode0 is
SIGNAL num:std_logic_vector(7 downto 0);
signal StackPointer:std_logic_vector(4 downto 0):="00000";
SIGNAL PUSHP, POPP, UDP, LD_P, WriteEnable,FlagEnable, SecCounEn, OVFP, CounterEnable,
FullSignal, EmptySignal : STD_LOGIC;
SIGNAL out_tos,out_tosm1: STD_LOGIC_VECTOR(4 DOWNTO 0);

COMPONENT singlepulsegen IS
   Port ( clk                  : in std_logic;              -- connect it to the Clock of the board
```

```
        rst                 : in std_logic;             -- connect it to the Reset Button of the
board
        input       : in std_logic;             -- connect it to the Push Button of the board
        output      : out std_logic             -- connect it to the input of your circuit
       );
end COMPONENT;

COMPONENT FSM IS
   Port ( Push : in  STD_LOGIC;
        Pop : in  STD_LOGIC;
        Clk : in  STD_LOGIC;
        Rst : in  STD_LOGIC;
                        Empty : in STD_LOGIC;
                        Full: in STD_Logic;
        U_D : out  STD_LOGIC;
        LOAD : out  STD_LOGIC;
        CounterEn : out  STD_LOGIC;
                        StackWrite : out STD_LOGIC;
                        STACK_OVF : out  STD_LOGIC);
END COMPONENT;

COMPONENT Comparator_Equality is
   Port ( B1 : in  STD_LOGIC_VECTOR (4 downto 0);
        B2 : in  STD_LOGIC_VECTOR (4 downto 0);
        Y1 : out  STD_LOGIC);
end COMPONENT;
```

```
COMPONENT STACK IS
        PORT ( a: in STD_LOGIC_VECTOR(4 downto 0);
            d: in STD_LOGIC_VECTOR(7 downto 0);
                    clk: in STD_LOGIC;
                    we: in STD_LOGIC;
                    spo: out STD_LOGIC_VECTOR(7 downto 0));
END COMPONENT;

COMPONENT COUNTER IS
   Port ( En : in  STD_LOGIC;
      input : in  STD_LOGIC_Vector(4 downto 0);
      output : out  STD_LOGIC_Vector(4 downto 0);
      Clock : in  STD_LOGIC;
      Load : in  STD_LOGIC;
      U_D : in  STD_LOGIC);
end COMPONENT;

COMPONENT SSDController is
        PORT (Clk : in  STD_LOGIC;
      Rst : in  STD_LOGIC;
      Empty : in  STD_LOGIC;
      Full : in  STD_LOGIC;
      Stack_OVF : in  STD_LOGIC;
                    ANcontrol : out STD_LOGIC_VECTOR(3 downto 0);
                    SSControl : out STD_LOGIC_VECTOR(7 downto 0));
end COMPONENT;
```

```
begin


SPG0 : singlepulsegen
        PORT MAP( clk => Clock,
                                rst => Reset,
                                input => Push,
                                output => PUSHP);


SPG1 : singlepulsegen
        PORT MAP( clk => Clock,
                                rst => Reset,
                                input => Pop,
                                output => POPP);
MY_FSM : FSM
        PORT MAP( Push => PUSHP,
                                Pop => POPP,
                                Clk => Clock,
                                Rst => Reset,
                                U_D => UDP,
                                Empty => EmptySignal,
                                Full => FullSignal,
                                LOAD => LD_P,
                                CounterEn => CounterEnable,
                                StackWrite => WriteEnable,
                                STACK_OVF =>OVFP);
```

```
myStack : STACK
        PORT MAP( a => StackPointer, --<<
                                        d => NUMIN,
                                        clk => Clock,
                                        we => WriteEnable,
                                        spo => NUMOUT);


Counter2 : COUNTER
        PORT MAP( input => StackPointer,
                                        U_D => UDP,
                                        En => CounterEnable,
                                        Clock => Clock,
                                        Load=>LD_P,
                                        output => out_tos);


Counter1 : COUNTER
        PORT MAP( input =>  StackPointer,
                                        U_D => UDP,
                                        En => SecCounEn, --Blepe telos kodika
                                        Clock => Clock,
                                        Load=>LD_P,
                                        output => out_tosm1);
```

```
EmptyComparator : Comparator_Equality
        PORT MAP (B1 => "00000",
                                B2 => out_tos,
                                Y1 => EmptySignal);

FullComparator : Comparator_Equality
        PORT MAP (B1 => "00011",
                                B2 => out_tos,
                                Y1 => FullSignal);



ssdControl: SSDController
        PORT MAP(Clk => Clock,
                                Rst => Reset,
                                Empty => EmptySignal,
                                Full => FullSignal,
                                Stack_OVF=> OVFP,
                                ANControl => SSD_EN,
                                SSControl => SSD);

secondCounterEnable :PROCESS(EmptySignal)
BEGIN
        IF (EmptySignal = '1') THEN --<<<<<
                FlagEnable <= '0';
        ELSE
                FlagEnable <= '1';
```

```
        END IF;
END PROCESS;




SecCounEn <= CounterEnable AND FlagEnable;


end behavioral;
```