5/5/2017

4ή Εργαστηριακή Άσκηση

ΑΝΑΓΝΏΡΙΣΗ ΠΡΆΞΕΩΝ ΓΙΑ ΣΧΕΔΊΑΣΗ ΑΡΙΘΜΟΜΗΧΑΝΉΣ

(Behavioral and Structural VHDL)

Ομάδα LAB20332005

ΧΡΗΣΤΟΣ ΖΗΣΚΑΣ 2014030191
ΠΑΝΑΓΙΩΤΗΣ ΣΑΒΒΑΙΔΗΣ 2013030180

Σκοπός εργαστηριακής άσκησης

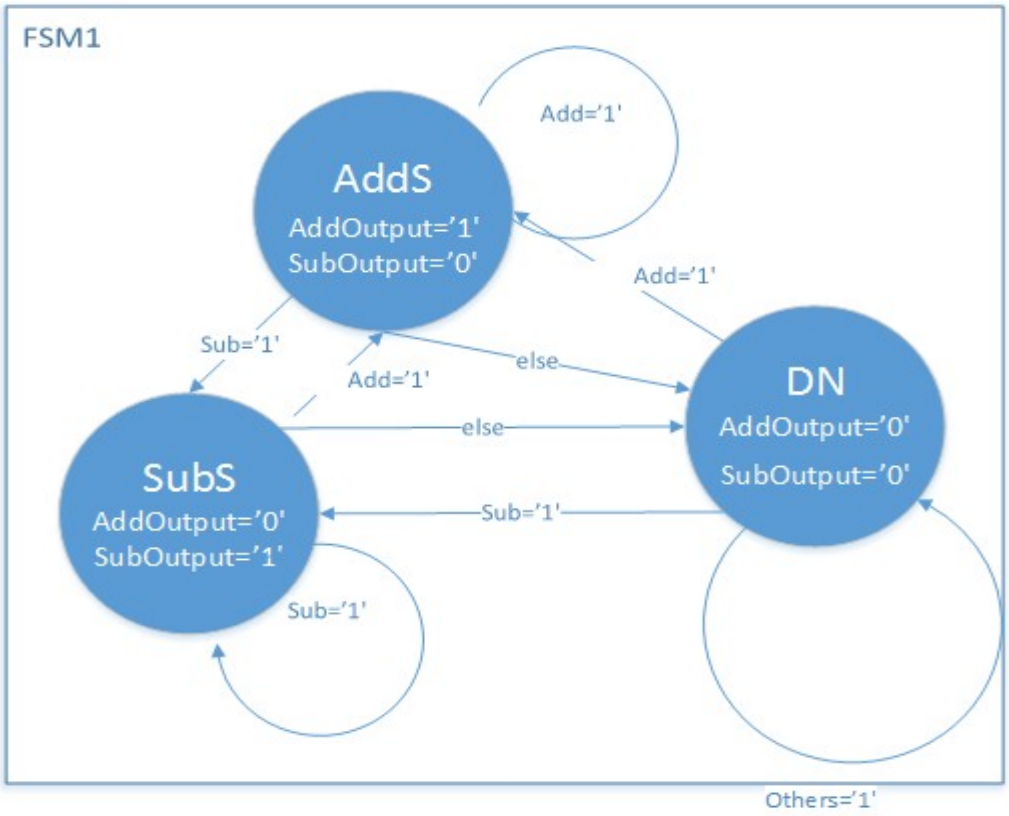
Σκοπός της εργαστηριακής άσκησης είναι η επέκταση της υλοποίησης του εργαστηρίου 3- επέκταση των λειτουργιών που δέχεται η στοίβα με περισσότερα εργαλεία ώστε να μεταμορφωθεί σε εφαρμογή αριθμομηχανής. Ο προσανατολισμός της σχεδίασης σε αυτό το εργαστήριο, κατευθύνεται στην διαμόρφωση του πλαισίου στο οποίο γίνεται η επιλογή των εκάστοτε πράξεων που επιθυμεί ο χρήστης με τρόπο, ώστε να συνυπάρξουν ομαλά όλες οι δυνατές πράξεις- 4 κουμπιά για 7 πράξεις. Απαιτείται η συνετή δόμηση της σχεδίασής για την εξυπηρέτηση της εύκολης αποσφαλμάτωσης.

Προεργασία

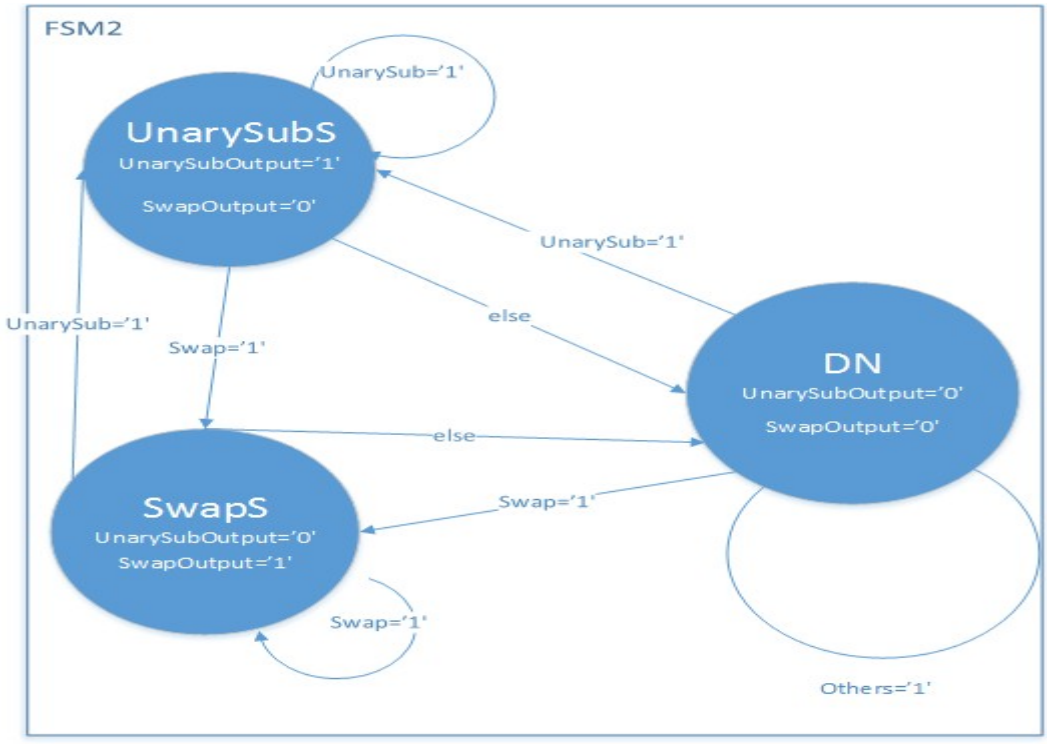
Η προετοιμασία του κυκλώματος απαιτεί :

- Σχεδίαση block diagram της συνολικής σχεδίασης ,της διεπαφής αλλά και των υποκυκλωμάτων για την απεικόνιση των αποτελεσμάτων στα led (διαφέρει από το προηγούμενο block diagram καθώς έχουν εισαχθεί νέες συνδέσεις και νέα συνδυαστικά κυκλώματα).
- Διεύρυνση του μηχανισμού του αποκωδικοποιητή για αποτύπωση στα led της επιθυμητής πράξης
- Διαγράμματα των διαφόρων FSM για την αλληλουχία των πράξεων αλλά και των Modes (2 FSM για κάθε 2 πράξεις , και 1 FSM που αφορά το mode που αντιστοιχεί στο ζευγάρι πράξεων)

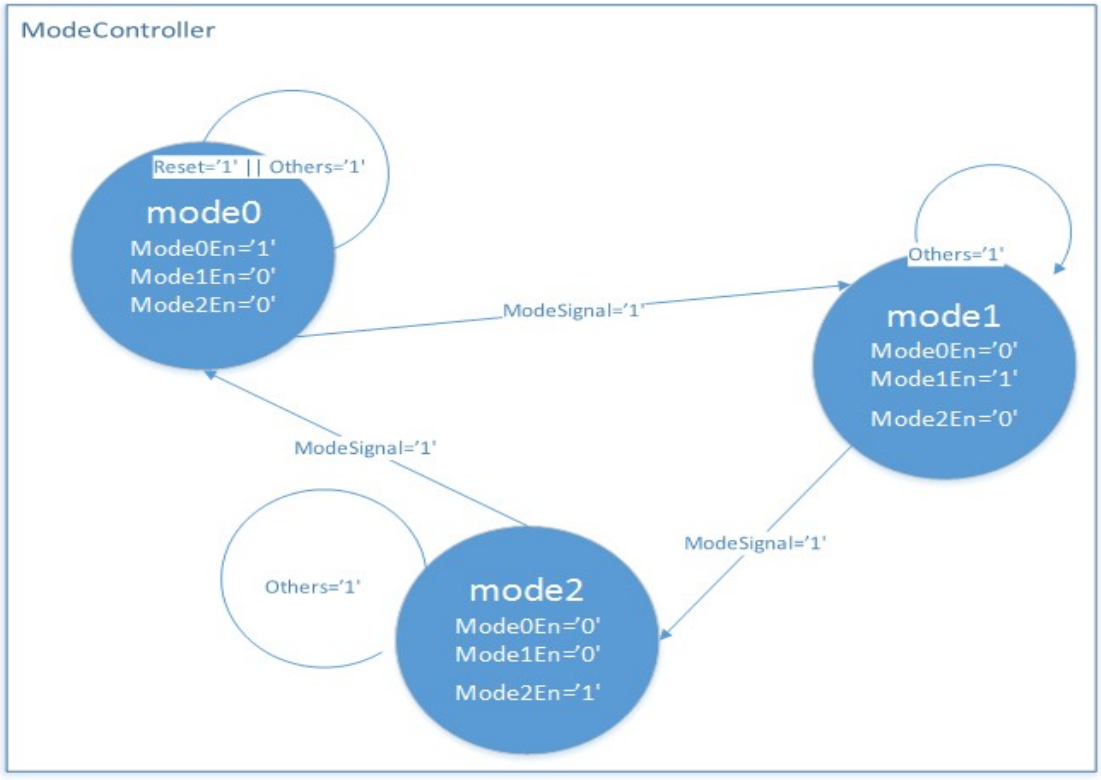
Μηχανή Πεπερασμένων Καταστάσεων για τα ζεύγη πράξεων Add-Sub(FSM1)



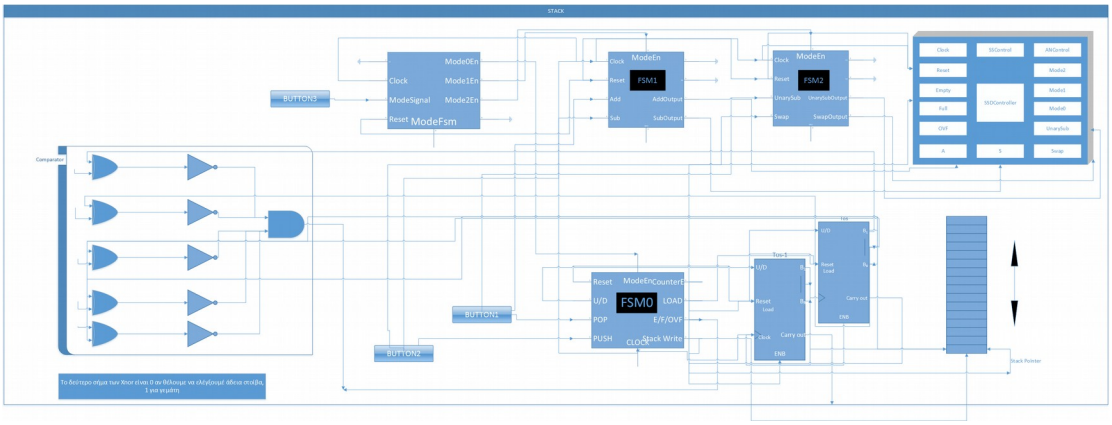
Μηχανή Πεπερασμένων Καταστάσεων για τα ζεύγη πράξεων UnarySub-Swap(FSM2)



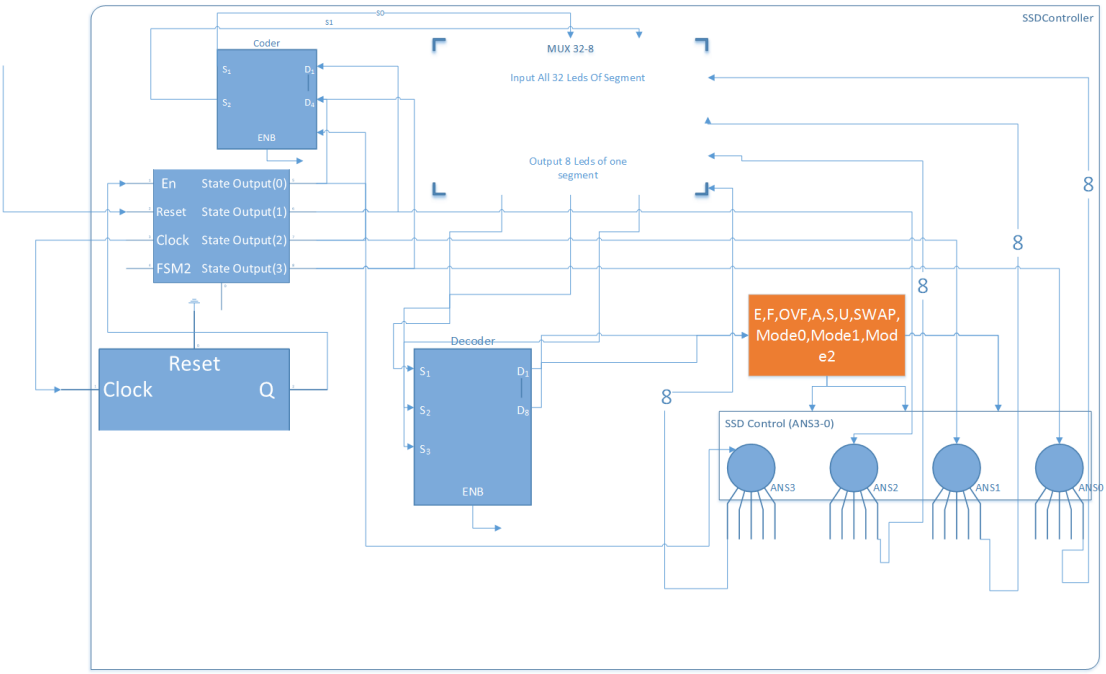
Μηχανή Πεπερασμένων Καταστάσεων για το Mode Controller (ModeFSM)



Υλοποίηση Ολοκληρωμένου κυκλώματος Στοίβας (Stack)



Υλοποίηση 7-segment Display



*Το κύκλωμα του μετρητή 5 bit δεν διαφοροποιείται

Περιγραφή

Το κύκλωμα διατηρεί αναλλοίωτες τις αρχικές του ιδιότητες εκχώρησης και αποβολή τιμής από την στοίβα. Δέχεται όμως μετατροπές και προσθήκες καθώς το κύκλωμα συνεχίζει τη λειτουργία του σύμφωνα με συγκεκριμένες μεταβάσεις modes που οδηγούν το κύκλωμα σε συγκεκριμένα ζευγάρια πράξεων. Οι νέες αριθμητικές και λογικές πράξεις που υποστηρίζονται από την διευρυμένη δομή είναι συνολικά εφτά (μαζί με το push/pop) και υλοποιείται η αναγνώριση τους και όχι η λειτουργία τους. Οι 7 πράξεις δεν μπορούν να αναγνωρισθούν από τέσσερα κουμπιά που υπάρχουν πάνω στην αναδιατασσόμενη μηχανή . Το πρόβλημα αυτό λύνεται με τα modes στα οποία μεταβαίνει το κύκλωμα

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.

Add=>Πρόσθεση 2’ complement αριθμών που βρίσκονται στην οροφή της στοίβας (TOS+’’TOS-1’’)

Sub=>Αφαίρεση 2’ complement αριθμών που βρίσκονται στην οροφή της στοίβας (TOS+’’TOS-1’’)

Unary Sub=> Μοναδιαία αφαίρεση 2’ complement του αριθμού που βρίσκεται στην οροφή της στοίβας

Swap => Εναλλαγή των τιμών που βρίσκονται στην οροφή και πριν από αυτήν (TOS<>”TOS-1”)

Mode => Με διαδοχικά πατήματα του κουμπιού , αναγνωρίζουμε σε ποίο ζευγάρι πράξεων βρισκόμαστε στην μνήμη.

Το Reset παραμένει στο κουμπί και η λειτουργία του δεν διαφοροποιείται αφού προκαλεί την αρχικοποίηση της μνήμης ,αντιγράφονται δηλαδή οι εκχωρημένες τιμές σε κάθε νέα εγγραφή. Στην αδράνεια είναι η μόνη λειτουργία που μπορεί να πραγματοποιηθεί. Φυλάσσονται τα χαρακτηριστικά των μετρητών της στοίβας ως έχουν όπως και τα υπόλοιπα γνωρίσματα της μνήμης (comparators, muxes, stack)

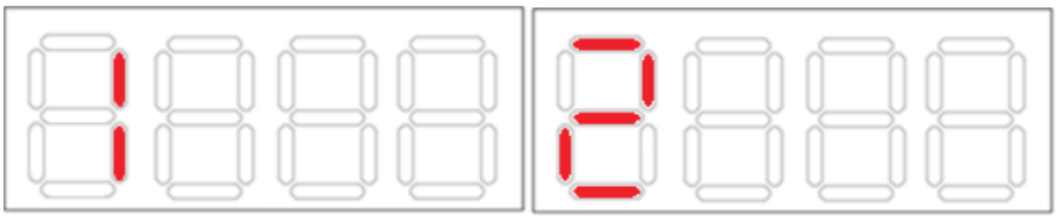
Οι συνολικές ενέργειες του κυκλώματος αποτυπώνονται στον παρακάτω πίνακα

Mode / Action/Button	0	1	2
0	Push	Add	Unary Sub
1	Pop	Sub	Swap
2	Change Mode	Change Mode	Change Mode
3	Reset	Reset	Reset

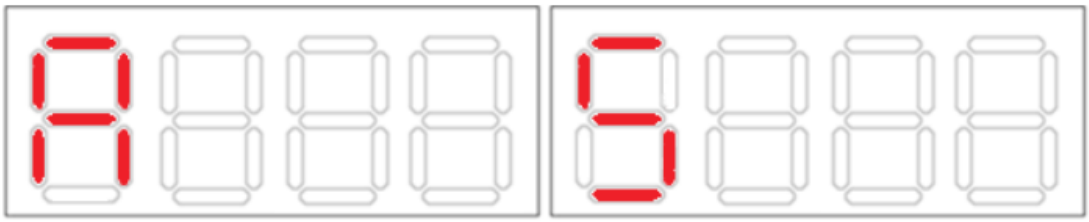
Παρόμοια διάρθρωση πραγματοποιήθηκε και στην εφαρμογή των 7-Segment displays ώστε να εκπληρωθεί το κύκλωμα και να παρουσιαστεί η απεικόνιση του κυκλώματος.

Υπάρχουν 6 νέες εικόνες για την σωστή εφαρμογή του κυκλώματος:

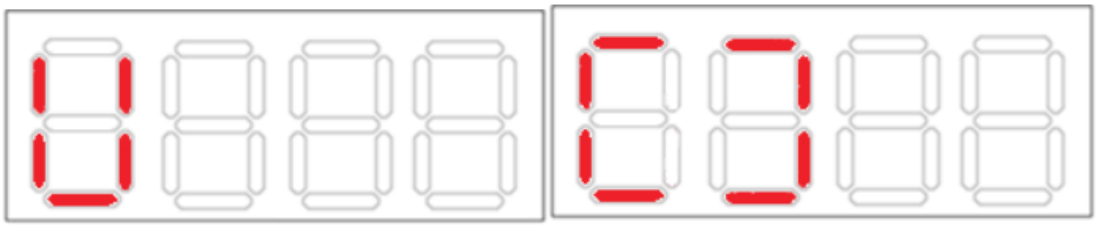
Ανοιχτό: Ανάβει το αριστερότερο segment και εκφράζει τις διάφορες εναλλαγές που υφίσταται το σύστημα όσον αφορά την μετάβαση από το κάθε mode στο επόμενο του.



Ανοιχτό – Ανάβει το πρώτο segment δηλώνοντας Add η Sub(το κύκλωμα βρίσκεται στο Mode 1).



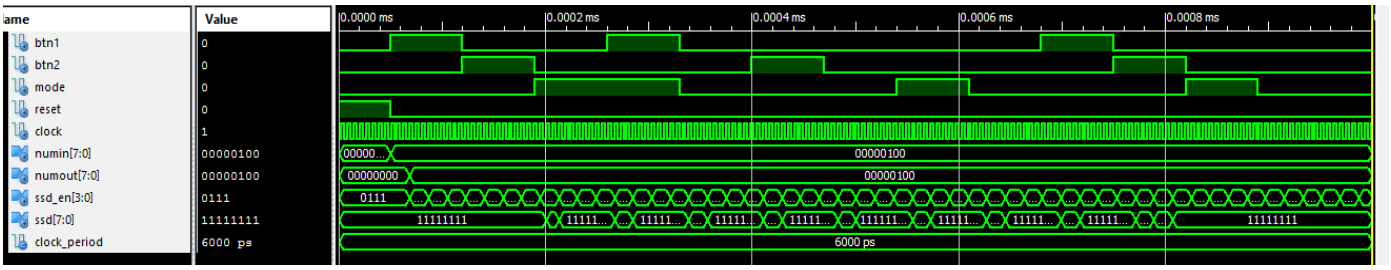
Ανοιχτό – Ανάβουν τα πρώτα segment δηλώνοντας Unary Subtraction η Swap (το κύκλωμα βρίσκεται στο Mode 2)



*Όταν το κύκλωμα παραμένει στο mode 0 τότε δηλώνεται στα segment η κατάσταση της στοίβας (E,F,OVF)

Κυματομορφές-Προσομοίωση

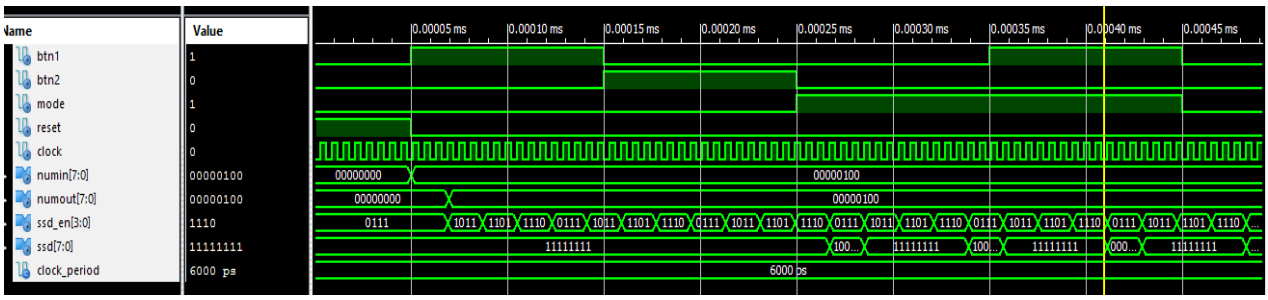
Παρουσιάζουμε τις κυμματομορφές του συστήματος



Διαμορφώνουμε τη προσομοίωση ως εξής:

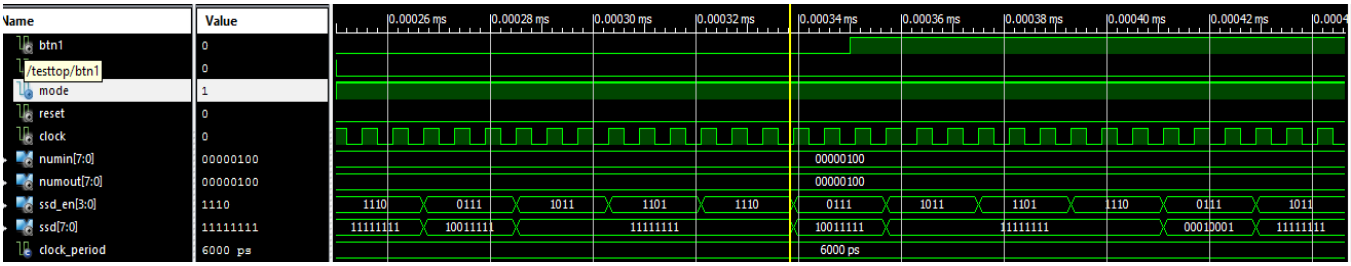
- Αρχικά το κύκλωμα βρίσκεται σε Reset
- Αφήνουμε το Reset (το κύκλωμα βρίσκεται σε mode 0) και ενεργοποιούμε το btn1(Push)
- Ελέγχουμε την λειτουργία του Pop
- Μεταβάλουμε το mode από 0 -> 1
- Πατάμε ξανά το κουμπί για αναγνώριση του Add
- Ελέγχουμε την λειτουργία του Sub
- Αλλάζουμε σε mode 2
- Εκτελούμε την πράξη Unary Subtraction
- Παρατηρούμε την εναλλαγή τιμών Swap
- Επαναφορά στην αρχική κατάσταση
- Επισημαίνουμε τις μεταβολές στα Led

Σημειώνεται ότι επιθυμούμε μόνο την αναγνώριση των πράξεων και <u>όχι</u> την εκτέλεση τους!!

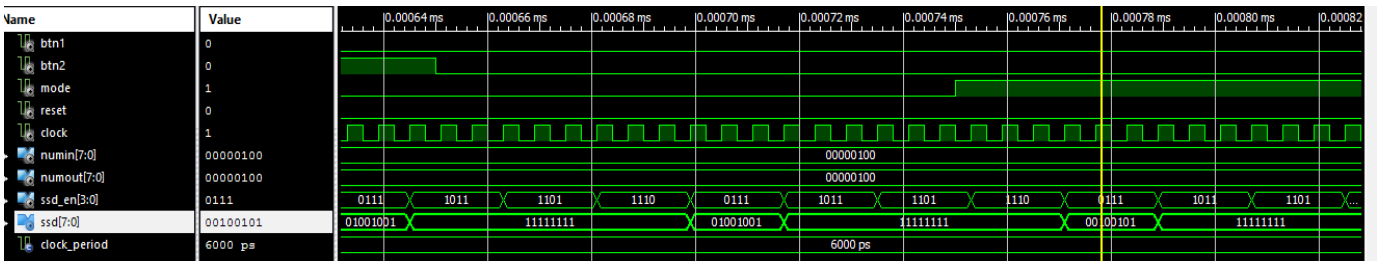


Στο Reset το σύστημα είναι ακινητοποιημένο. Με το πρώτο πάτημα για εκτέλεση πράξεων – πράξη Push η στοίβα διαθέτει τουλάχιστον ένα στοιχείο και αποφεύγουμε η στοίβα να αδειάσει (Empty). Στο πάτημα του κουμπιού btn2 το στοιχείο αφαιρείται και η στοίβα δεν έχει ξεμείνει από στοιχεία (Στο επόμενο Pop εμφανίζει 01100001 – E-).

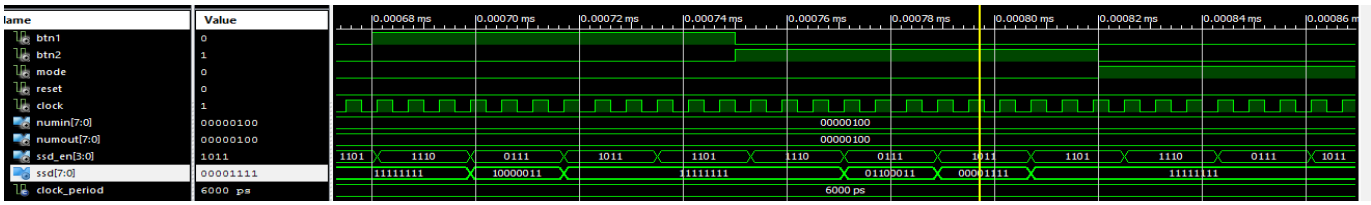
Επιθυμούμε αλλαγή κατάστασης mode αλλά και δραστηριοποίηση της πράξης που αναγνωρίζει το btn1 στο mode1 (πράξη Add).



Με παραμονή στην ίδια κατάσταση επιθυμούμε την εκτέλεση που αναγνωρίζει το btn2 (πράξη Sub) και μετά από εξέταση της πράξης μεταβαίνουμε στην επόμενη κατάσταση για την οποία εστιάζουμε την ανίχνευση της στα led.



Ακολούθως γίνεται η πράξη της μοναδιαίας αφαίρεσης και της εναλλαγής τιμών διαδοχικά ενώ αφού έχει ολοκληρωθεί ο έλεγχος της εγκυρότητας των πράξεων το σύστημα επαναφέρεται στην αρχική του θέση.



Τα αποτελέσματα των παραπάνω διεργασιών αποδίδονται στα Leds* ως εξής:

11111111 11111111 11111111 11111111----------→Mode 0 , Btn1 (1ο Push)

11111111 11111111 11111111 11111111----------→Mode 0, Btn2(1ο Pop)

10011111 11111111 11111111 11111111----------→Mode 1

00010001 11111111 11111111 11111111----------→Mode 1 Btn1(1ο Add)

01001001 11111111 11111111 11111111----------→Mode 1 Btn2(1ο Sub)

00100101 11111111 11111111 11111111----------→Mode 2

10000011 11111111 11111111 11111111----------→Mode 2 Btn1(1ο Unary)

01100011 00001111 11111111 11111111----------→Mode 2 Btn2(1ο Swap)

11111111 11111111 11111111 11111111----------→Mode 0

Διαπιστώνεται ότι για τις παραπάνω ενέργειες η παρουσίαση στα segments είναι η αναμενόμενη

*Υπενθυμίζεται ότι τα Led’s είναι αρνητικά ακμοπυροδότητα και για την αναπαράσταση των modes χρησιμοποιούνται τα αριστερότερα segments.

Συμπεράσματα

Η περάτωση της εργαστηριακής άσκησης διεγείρει ορισμένα χρήσιμα συμπεράσματα για την λειτουργία των μηχανών πεπερασμένων καταστάσεων για την ορθή αναγνώριση των διαδικασιών στα Leds , την σημασία των διαφόρων εργαλείων στην επίτευξη αυτής της συνοχής , την παρουσίαση περισσότερων αποτελεσμάτων σε συνδυασμό με την ανακατασκευή των μηχανών με την μνήμη. Ολοκληρώνοντας αποτελεί σημαντικό βήμα για την ολοκλήρωση της βασικής δομής της αριθμομηχανής

Κώδικας

- Στον κώδικα δεν περιλαμβάνονται οι FSM (ssdFsm) για τα Seven segment display και την στοίβα (FSM0,FSM1,FSM2, ModeController) καθώς επίσης και τα singe pulse generator.

TOP

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

entity Top is
Port ( BTN1 : in  STD_LOGIC;
BTN2 : 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 Top;

architecture Behavioral of Top is
signal StackPointer:std_logic_vector(4 downto 0):="00000";
SIGNAL BTN1P, BTN2P, UDP, LD_P, WriteEnable,FlagEnable, SecCounEn, OVFP, CounterEnable,
FullSignal, EmptySignal : STD_LOGIC;
SIGNAL out_tos,out_tosm1: STD_LOGIC_VECTOR(4 DOWNTO 0);
signal Mode0Sig,Mode1Sig,Mode2Sig,ModePipe:std_logic:='0';
```

```
signal AddOut,SubOut:std_logic:='0';

signal UnarySubOut,SwapOut:std_logic:='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 Reg is

Port ( Clock : in  STD_LOGIC;

D : in  STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";

Enable : in  STD_LOGIC;

Q : out  STD_LOGIC_vector(7 downto 0));

end component;


COMPONENT FSM0 IS

Port ( Push : in  STD_LOGIC;

Pop : in  STD_LOGIC;

Clk : in  STD_LOGIC;

ModeEn: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 FSM1 is
           Port( Clk:in std_logic;
                 Rst:in std_logic;
                ModeEn:in std_logic;
                 Add:in std_logic;
                 Sub:in std_logic;
               AddOutput:out std_logic;
               SubOutput:out std_logic);
     end component;

    component FSM2 is
           Port( Clk:in std_logic;
                 Rst:in std_logic;
                ModeEn:in std_logic;
               UnarySub:in std_logic;
                 Swap:in std_logic;
             UnarySubOutput:out std_logic;
              SwapOutput: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;
                  A:in std_logic;
                  S:in std_logic;
                  U:in std_logic;
                SWAP:in std_logic;
               Mode0:in std_logic;
               Mode1:in std_logic;
               Mode2:in std_logic;
ANcontrol : out STD_LOGIC_VECTOR(3 downto 0);
SSControl : out STD_LOGIC_VECTOR(7 downto 0));
    end COMPONENT;

 Component ModeFsm is
Port ( Clock : in  STD_LOGIC;
       Reset : in  STD_LOGIC;
  ModeSignal : in  STD_LOGIC;
   Mode0En : out  STD_LOGIC;
   Mode1En : out  STD_LOGIC;
  Mode2En : out  STD_LOGIC);
     end component;

          begin
```

```
SPG0 : singlepulsegen
    PORT MAP( clk => Clock,
                rst => Reset,
              input => BTN1,
             output => BTN1P);

SPG1 : singlepulsegen
    PORT MAP( clk => Clock,
                rst => Reset,
              input => BTN2,
             output => BTN2P);

SPG2 : singlepulsegen
      Port Map(clk=>Clock,
                rst=>Reset,
              input=>Mode,
            output=>ModePipe);

   FSM_M0 : FSM0
  PORT MAP( Push => BTN1P,
                Pop => BTN2P,
                Clk => Clock,
            ModeEn=>Mode0Sig,
                Rst => Reset,
                U_D => UDP,
```

```
                Empty => EmptySignal,
                Full => FullSignal,
                LOAD => LD_P,
                CounterEn => CounterEnable,
                StackWrite => WriteEnable,
                STACK_OVF =>OVFP);

    FSM_M1:FSM1
     Port MAP(Clk=>Clock,
                Rst=>Reset,
                ModeEn=>Mode1Sig,
                Add=>BTN1P,
                Sub=>BTN2P,
                AddOutput=>AddOut,
                SubOutput=>SubOut);

    FSM_M2:FSM2
     Port MAP(Clk=>Clock,
                Rst=>Reset,
                ModeEn=>Mode2Sig,
                UnarySub=>BTN1P,
                Swap=>BTN2P,
                UnarySubOutput=>UnarySubOut,
                SwapOutput=>SwapOut);

  ModeController:ModeFsm
```

```
Port map(Clock=>Clock,
                Reset=>Reset,
            ModeSignal=>ModePipe,
             Mode0En=>Mode0Sig,
             Mode1En=>Mode1Sig,
             Mode2En=>Mode2Sig);


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 => "11111",
                          B2 => out_tos,
                          Y1 => FullSignal);


ssdControl: SSDController
                PORT MAP(Clk => Clock,
                         Rst => Reset,
                         Empty => EmptySignal,
                         Full => FullSignal,
                         A=>AddOut,
                         S=>SubOut,
                         U=>UnarySubOut,
```

```
                SWAP=>SwapOut,
                Mode0=>Mode0Sig,
                Mode1=>Mode1Sig,
                Mode2=>Mode2Sig,
                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;
```

ssdControl

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

entity SSDController is
    PORT (Clk : in  STD_LOGIC;
          Rst : in  STD_LOGIC;
          Empty : in  STD_LOGIC;
          Full : in  STD_LOGIC;
          Stack_OVF : in  STD_LOGIC;
                      A:in std_logic;
                      S:in std_logic;
                      U:in std_logic;
                      SWAP:in std_logic;
                      Mode0:in std_logic;
                      Mode1:in std_logic;
                      Mode2:in std_logic;
          ANcontrol : out STD_LOGIC_VECTOR(3 downto 0);
          SSControl : out STD_LOGIC_VECTOR(7 downto 0));
    end SSDController;

architecture Behavioral of SSDController is
```

```
Signal CounterPipe : STD_LOGIC;
Signal FSMOutPipe : STD_LOGIC_VECTOR(3 downto 0);
Signal muxInPipe: STD_LOGIC_VECTOR(31 downto 0);
Signal muxControl : STD_LOGIC_VECTOR(1 downto 0);

COMPONENT BehCounter IS
Port ( Clk : in  STD_LOGIC;
       Rst : in  STD_LOGIC;
         Q : out  STD_LOGIC);
END COMPONENT;

COMPONENT SSDFSM is
Port ( Clk : in  STD_LOGIC;
       Rst : in  STD_LOGIC;
        En  : in Std_logic;
State_output : out  STD_LOGIC_VECTOR (3 downto 0));
end COMPONENT;

COMPONENT 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 COMPONENT;

COMPONENT Decoder is
Port ( E : in  STD_LOGIC;
```

```
F : in  STD_LOGIC;
OV : in  STD_LOGIC;
A:in std_logic;
S:in std_logic;
U:in std_logic;
SWAP:in std_logic;
Mode0:in std_logic;
Mode1:in std_logic;
Mode2:in std_logic;
segmentLED : out  STD_LOGIC_VECTOR (31 downto 0));
end COMPONENT;

COMPONENT Coder is
Port ( Input : in  STD_LOGIC_VECTOR (3 downto 0);
Output : out  STD_LOGIC_VECTOR (1 downto 0));
end COMPONENT;

begin

count: BehCounter
PORT MAP (Clk => Clk,
Rst => Rst,
Q => CounterPipe);

TheFSM: SSDFSM
PORT MAP (Clk => Clk,
```

```
                    Rst => Rst,
                    En => CounterPipe,
                    State_output => FSMOutPipe);

    Mux: MUX32x8
PORT MAP(INPUT => muxInPipe,
                    control=>muxControl,
                    OUTPUT => SSControl);

  myDecoder: Decoder
    PORT MAP(E => Empty,
                    F => Full,
                    OV => Stack_OVF,
                    A=>A,
                    S=>S,
                    U=>U,
                    SWAP=>SWAP,
                    Mode0=>Mode0,
                    Mode1=>Mode1,
                    Mode2=>Mode2,
                    segmentLED => muxInPipe);

    myCoder: Coder
PORT MAP(Input => FSMOutPipe,
                    Output => muxControl);
```

```
ANControl <= FSMOutPipe;

end Behavioral;
```

myCoder

```
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;
```

myDecoder

```
----------------------------------------------------------------------------------

----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity Decoder is
Port ( E : in  STD_LOGIC;
F : in  STD_LOGIC;
OV : in  STD_LOGIC;
A:in std_logic;
S:in std_logic;
U:in std_logic;
SWAP:in std_logic;
Mode0:in std_logic;
Mode1:in std_logic;
Mode2:in std_logic;
```

```
segmentLED : out  STD_LOGIC_VECTOR (31 downto 0));
end Decoder;

architecture Behavioral of Decoder is
begin

coder: process(E, F, OV,Mode0,Mode1,Mode2,A,S,U,SWAP)
begin
IF(Mode0='1') then
IF(E='1') THEN segmentLED <= "11111111111111111111111101100001";
ELSIF(OV='1') THEN segmentLED <= "11111111000000111000001101110001";
ELSIF(F='1') THEN segmentLED <= "11111111111111111111111101110001";
ELSE segmentLED <= "11111111111111111111111111111111";
END IF;
ELSIF(Mode1='1')then
If(A='1') then segmentLED<="00010001111111111111111111111111";
ELSIF(S='1') then segmentLED<="01001001111111111111111111111111";
ELSE segmentLED<="10011111111111111111111111111111";
END IF;
ELSIF(Mode2='1')then
IF(U='1')THEN segmentLED<="10000011111111111111111111111111";
ELSIF(SWAP='1')then segmentLED<="01100011000011111111111111111111";
ELSE segmentLED<="00100101111111111111111111111111";
END IF;
ELSE
segmentLED<="11111111111111111111111111111111";
```

```
end if;
end process;
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;
```

ssdCounter

```
----------------------------------------------------------------------------------

----------------------------------------------------------------------------------
                            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=125000 then outs<=0;Q<='1';
end if;
end process;


oo <= outs;


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;
```

Counter

```
----------------------------------------------------------------------------------

----------------------------------------------------------------------------------
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;
```