26/5/2017

# 5ή Εργαστηριακή Άσκηση

## ΥΛΟΠΟΊΗΣΗ ΠΡΆΞΕΩΝ ΓΙΑ ΣΧΕΔΊΑΣΗ ΑΡΙΘΜΟΜΗΧΑΝΉΣ

## (Behavioral and Structural VHDL)

Ομάδα LAB20332005

| |
|---|
| *ΧΡΗΣΤΟΣ ΖΗΣΚΑΣ 2014030191* |
| *ΠΑΝΑΓΙΩΤΗΣ ΣΑΒΒΑΙΔΗΣ 2013030180* |


## Σκοπός εργαστηριακής άσκησης

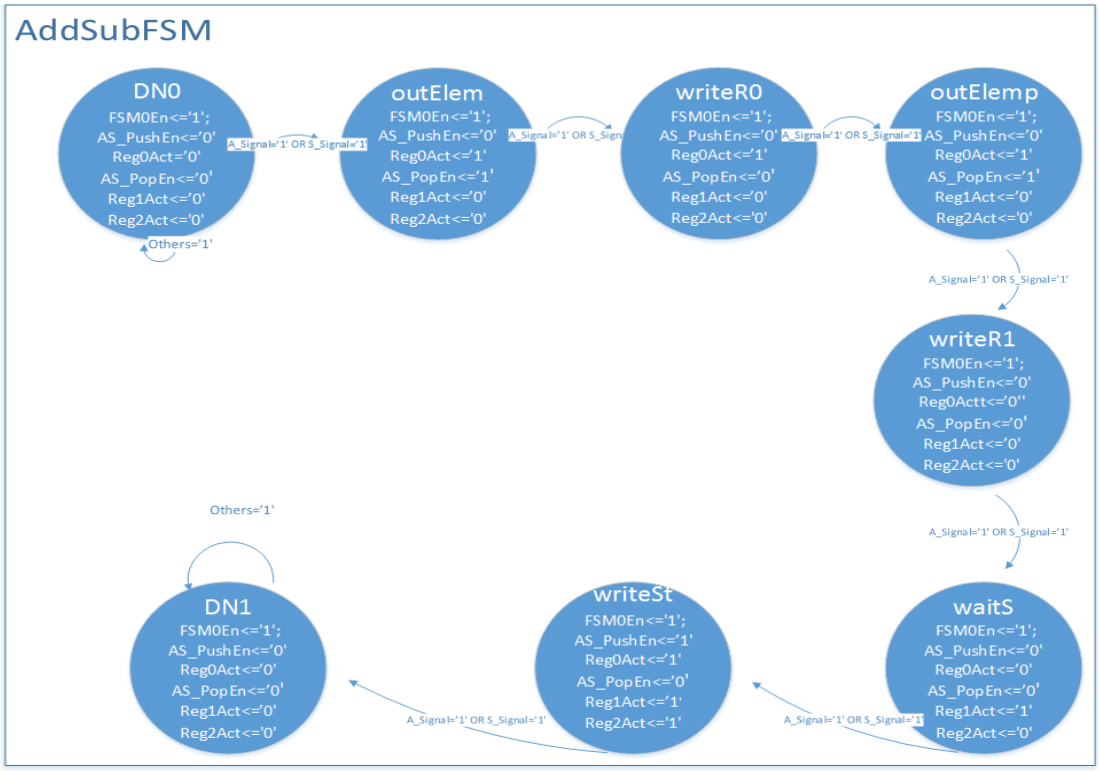
Σκοπός της τελευταίας εργαστηριακής άσκησης είναι η εκ-νέου επέκταση της υλοποίησης του εργαστηρίου 3- έχει διαμορφωθεί το σχέδιο για την αναγνώριση των πράξεων- με τον σχεδιασμό και την εκτέλεση των πράξεων ώστε να τελειοποιηθεί το μοντέλο της αριθμομηχανής. Η σχεδίαση επικεντρώνεται στην αντιπροσώπευση του κάθε κουμπιού στην εκάστοτε πράξη όταν το σύστημα βρίσκεται στα διάφορα mode. Η κάθε πράξη αποκωδικοποιείται από μια μηχανή πεπερασμένων καταστάσεων ενώ για την ολοκλήρωση της κάθε πράξης απαιτείται κάποιο υποσύστημα που διαμορφώνει το data path( πρόσθετης- αφαιρέτης, καταχωρητές , αναγνώριση της υπερχείλισης στον προσθέτη-αφαιρέτη ) και ολοκληρώνει τη μοντελοποίηση της στοίβας με τη σύνδεση συνδυαστικών κυκλωμάτων .

## Προεργασία

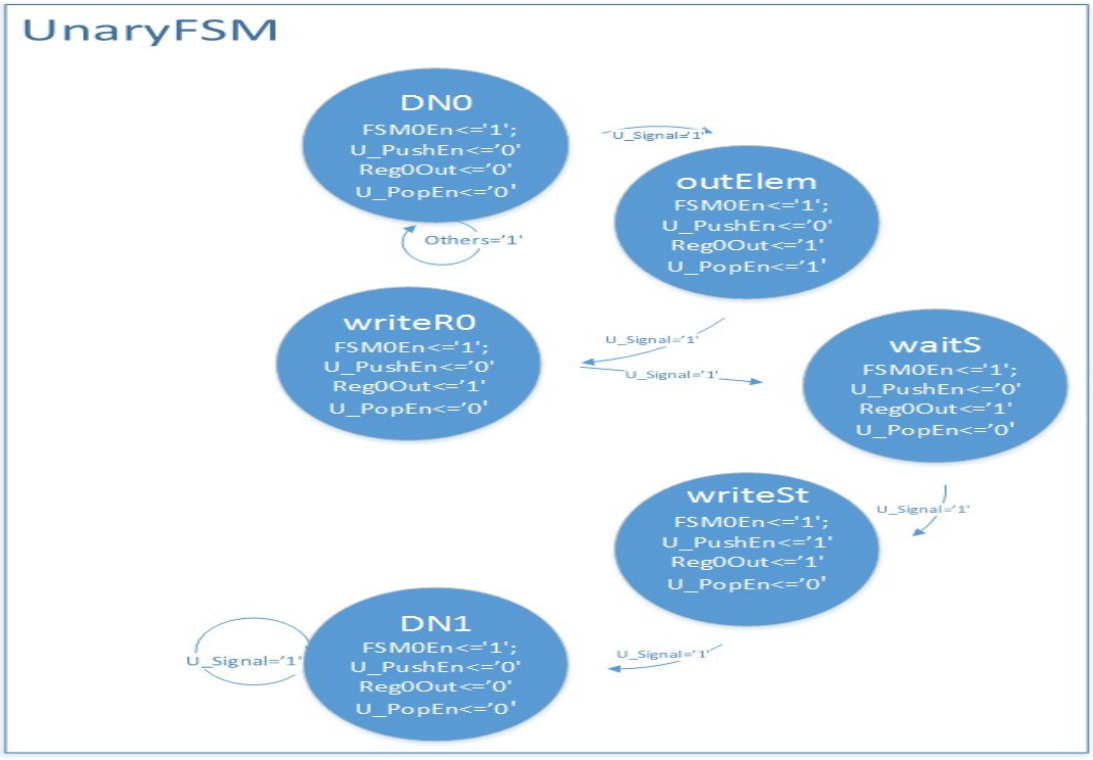
Η προετοιμασία του κυκλώματος απαιτεί :

- Το block diagram της συνολικής σχεδίασης – συμπεριλαμβάνει τις μηχανές πεπερασμένων καταστάσεων που κατευθύνουν την εκτέλεση των κουμπιών και των δομών που πραγματοποιούν τις πράξεις και το συγκροτημένο σχήμα της μνήμης.
- Επέκταση της διεπαφής του κυκλώματος με τα 7-segment display για την απεικόνιση των διαφόρων αποτελεσμάτων σε κάθε διασταύρωση του κυκλώματος
- Διαγράμματα των νέων μηχανών πεπερασμένων καταστάσεων (3 νέες fsm – αφορούν τις εκάστοτε πράξεις . Το add και το sub συγχρονίζονται σε μια μηχανή
- Κατασκευή κατάλληλων test για την αναγνώριση του σωστού σχεδιασμού του συστήματος στην προσομοίωση .

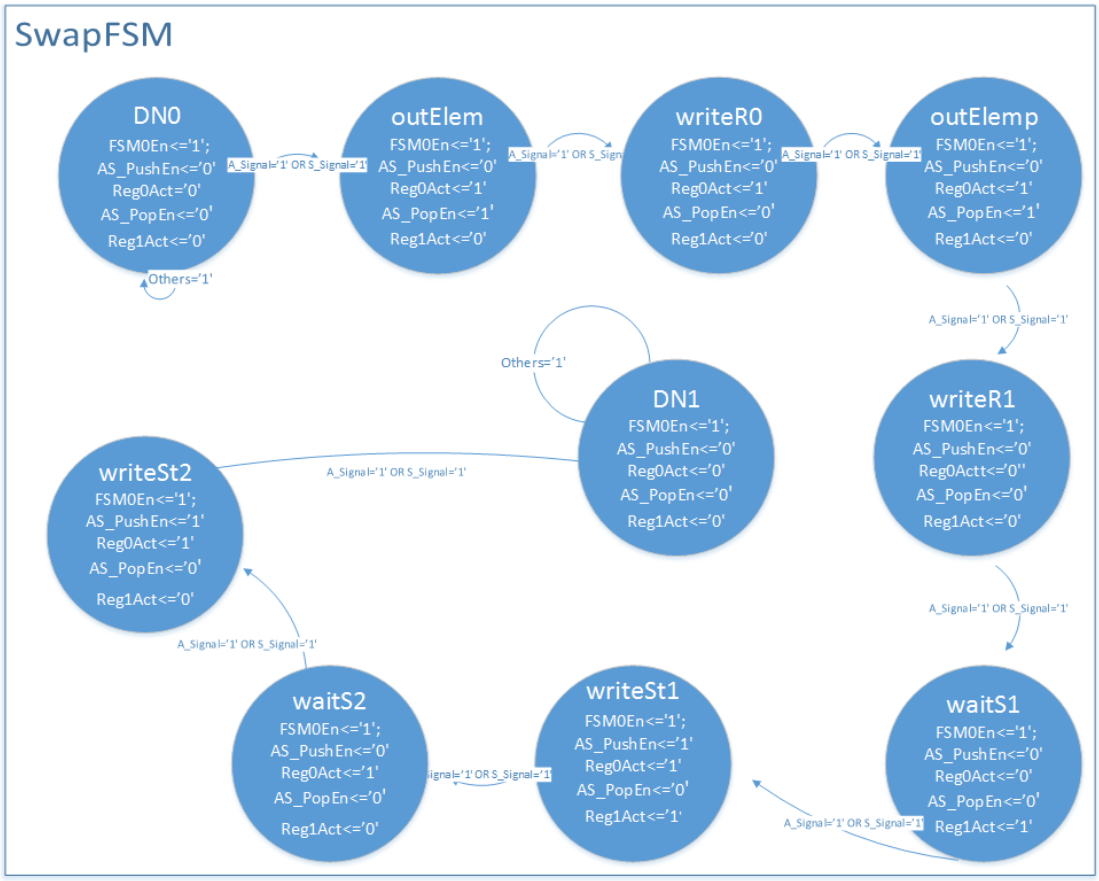
Μηχανή Πεπερασμένων Καταστάσεων για την εκάστοτε πράξη Add-Sub(ASFsm)



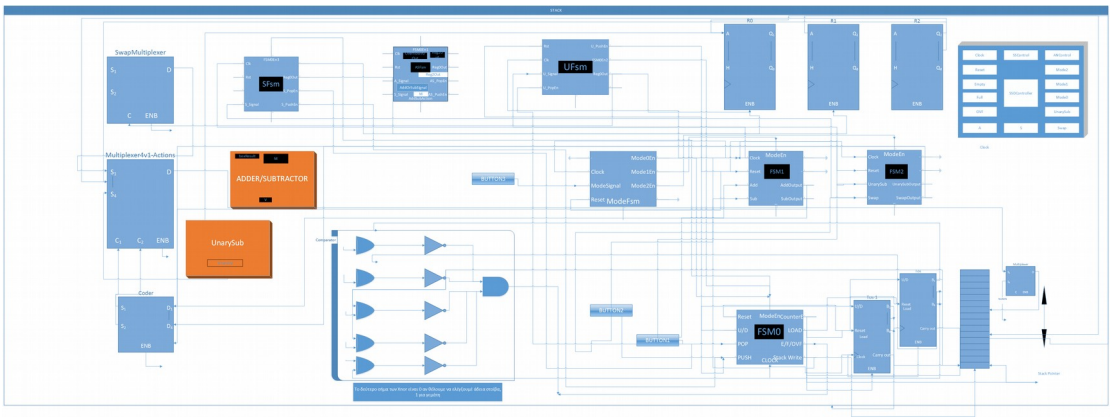

Μηχανή Πεπερασμένων Καταστάσεων για την πράξη UnarySub(UFsm)



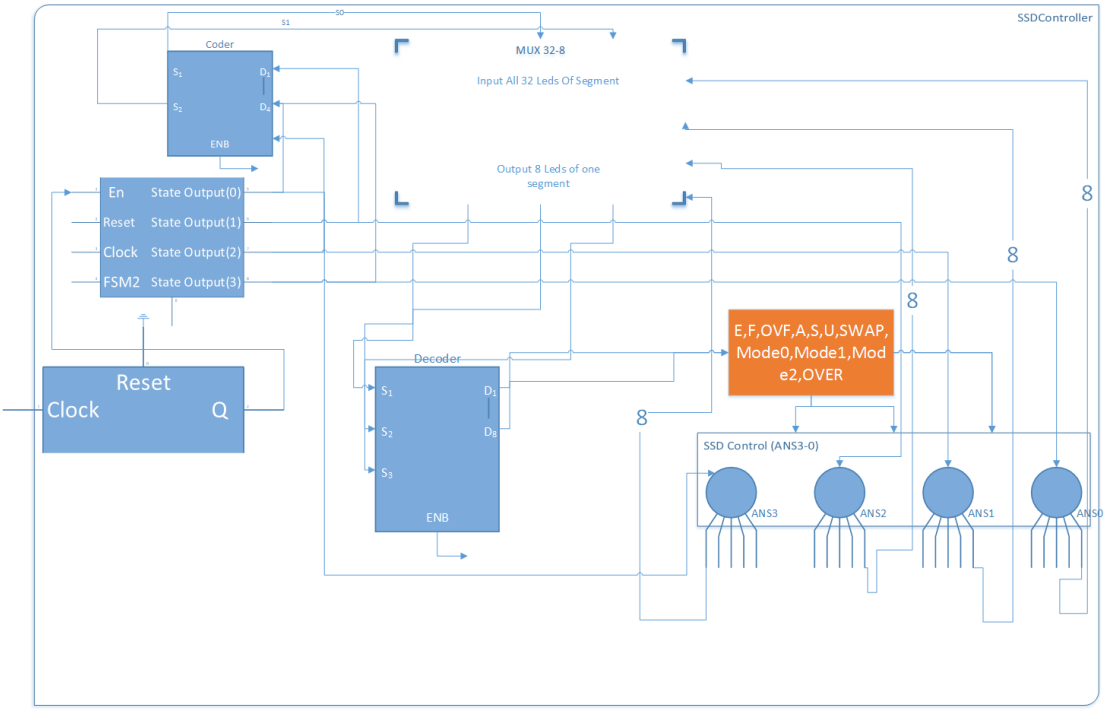
Μηχανή Πεπερασμένων Καταστάσεων για την πράξη Swap (SwapFSM)




Υλοποίηση Ολοκληρωμένου κυκλώματος Στοίβας (Stack)




Υλοποίηση 7-segment Display




## Υλοποίηση δομής προσθαφαιρέτη

Οι υπόλοιποι μηχανισμοί δεν διαφοροποιούνται

# Περιγραφή

Το κύκλωμα εξελίσσεται με την αναπαράσταση του τελευταίου σταδίου του καθώς η αναγνώριση κάθε αριθμητικής πράξης συμβαδίζει και με την περάτωση τής. Κάθε πράξη αφορά στην αναγνώριση της από το σύστημα, την ανταπόκριση του συστήματος στην εκάστοτε διαδικασία βημάτων που απευθύνεται στην μνήμη, την εξαγωγή αριθμών και την προσωρινή αποθήκευση τους σε εργαλεία συνδυαστικής λογικής καθώς και την επεξεργασία τους από την δομή του data path επίσης. Το αποτέλεσμα καθοδηγείται από πολυπλέκτες ώστε να καταλήξει η επιθυμητή τιμή στην μνήμη. Η εντολή για εκχώρηση στη μνήμη δίνεται από την fsm που ασχολείται με τη διαδικασία που υλοποιεί την πράξη ενώ η κάθε τέτοια fsm επικοινωνεί με την fsm που ασχολείται με την εισαγωγή/εξαγωγή τιμών. Κάθε τέτοια fsm έχει ως έξοδο ένα σήμα που εκπροσωπεί την

Η λειτουργικότητα της σχεδίασης εκφράζεται ως εξής:

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

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

***Add***=> Αποτελείται από 2 καταχωρήσεις pop από την στοίβα (εκχώρηση στους καταχωρητές) – 1 push με την επεξεργασία από τον προσθαφαιρέτη (αποθηκεύεται πρώτα σε έναν 3ο καταχωρητή.)

***Sub=>*** Αποτελείται από 2 καταχωρήσεις pop από την στοίβα (εκχώρηση στους καταχωρητές) – 1 push στην μνήμη μετά την επεξεργασία από τον προσθαφαιρέτη (αποθηκεύεται πρώτα σε έναν 3ο καταχωρητή)

**Unary Sub**=> Αποτελείται από 1 καταχώρηση pop (εκχώρηση στον καταχωρητή) – 1 push στην μνήμη μετά την επεξεργασία από τον προσθαφαιρέτη

**Swap** => Αποτελείται από 2 καταχωρήσεις pop (εκχώρηση στον καταχωρητή) – 2 διαδοχικά push στην μνήμη (πρώτα του 2ου καταχωρητή και ύστερα του 1ου)

Ο προσθαφαιρέτης για την πρόσθεση/αφαίρεση χρησιμοποιεί ως ορίσματα τους δυο αριθμούς που κάνει pop ενώ για το εργαλείο επεξεργασίας Unary Sub χρησιμοποιείται το ίδιο module με την διαφορά ότι το 1ο όρισμα είναι το “00000000” και το 2ο είναι αυτό που γίνεται pop. Σημειώνεται επίσης ότι υπάρχει και μηχανισμός εντοπισμού υπερχείλισης(λογικές πύλες) στον προσθαφαιρέτη που αναλογεί σε αναγνώριση της συγκεκριμένης κατάστασης από τα 7-segment display

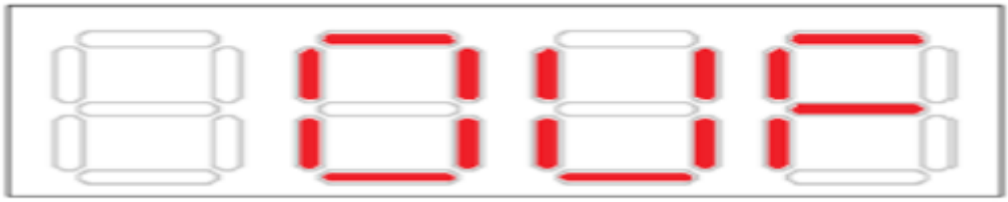
Παραμένει το κουμπί εναλλαγής mode χωρίς καμιά τροποποίηση – Καθοδηγείται από την fsm ως εξής

| Present State<br><br>Next State<br><br>(Εναλλαγή Mode) | Mode 0<br><br>Mode 1 | Mode 1<br><br>Mode 2 | Mode 2<br><br>Mode 0 |
|---|---|---|---|

Στα διάφορα mode παραμένουν οι αντίστοιχες πράξεις ( όπως στα προηγούμενα εργαστήρια)

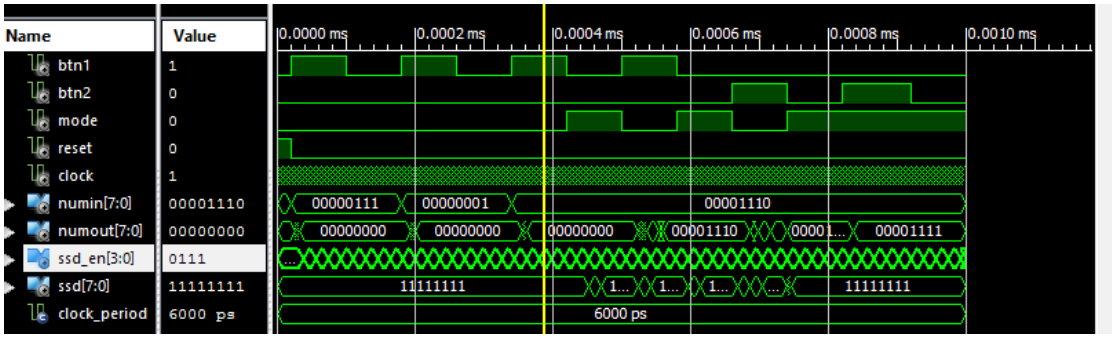
Το Reset παραμένει στο κουμπί και η λειτουργία του δεν διαφοροποιείται αφού προκαλεί την αρχικοποίηση της μνήμης ,αντιγράφονται δηλαδή οι εκχωρημένες τιμές σε κάθε νέα εγγραφή. Στην αδράνεια είναι η μόνη λειτουργία που μπορεί να πραγματοποιηθεί. Φυλάσσονται τα χαρακτηριστικά των μετρητών της στοίβας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

Μόνη τροποποίηση στην οποία έχει επιδεχθεί το σύστημα του display αφορά την αριθμητική υπερχείλιση . Το κύκλωμα αδρανοποιείται και στο αριθμητικό overflow αποτυπώνοντας ανάλογο σήμα overflow (Στο αριθμητικό overflow δείχνουμε την απεικόνιση OF χρησιμοποιώντας τα δυο αριστερότερα segment

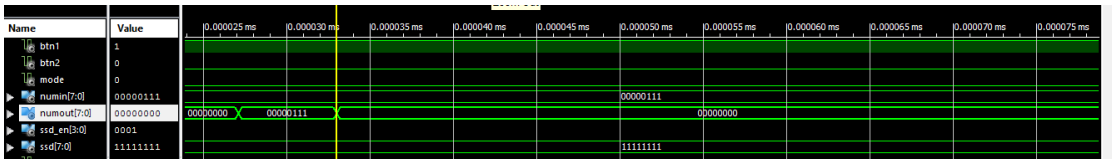


## Κυματομορφές-Προσομοίωση

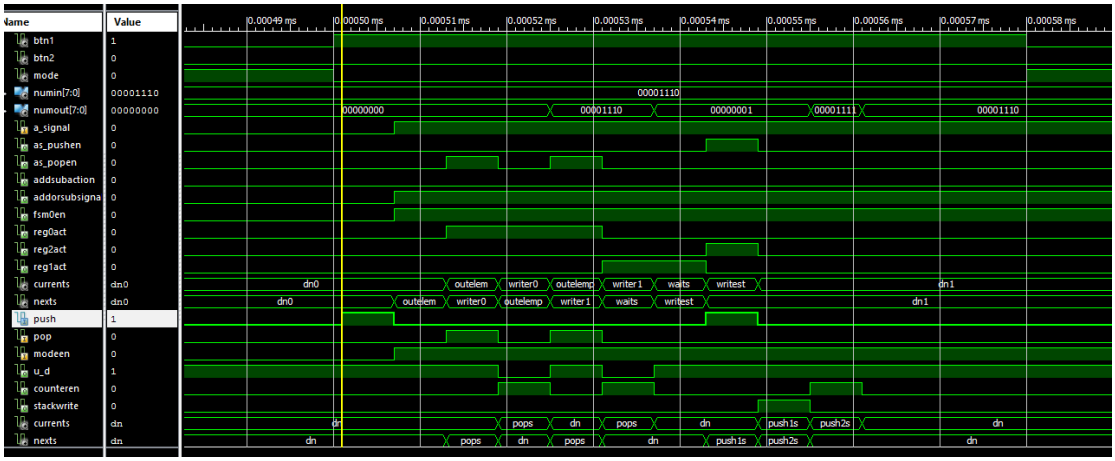
Παρουσιάζουμε τις κυμματομορφές του συστήματος.



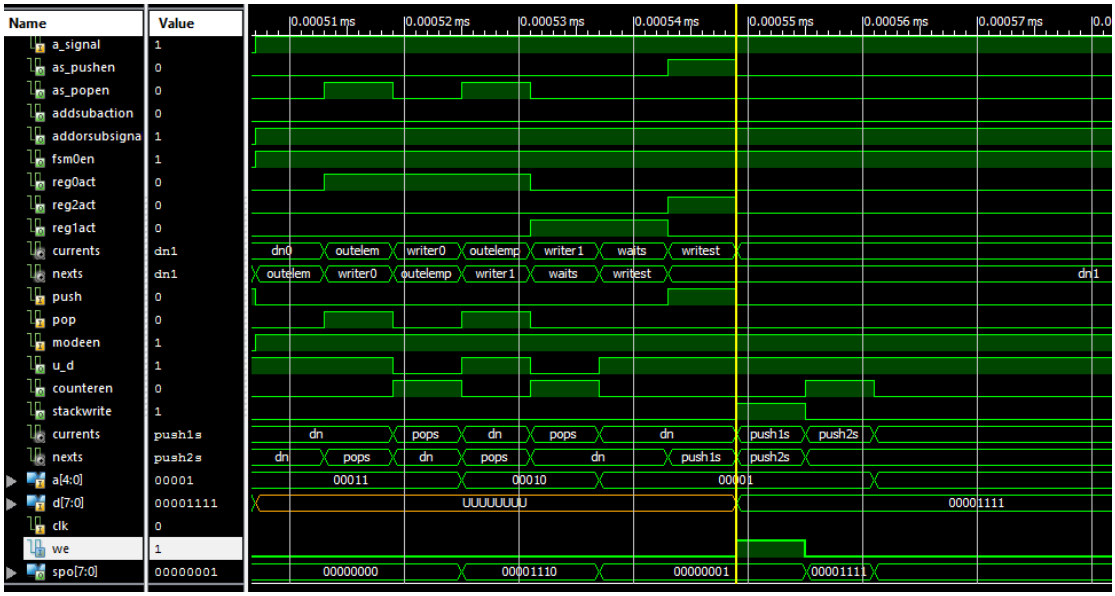

Στην παραπάνω κυμματομορφή ελέγχουμε τις πράξεις πρόσθεσης και μοναδιαίας αφαίρεσης. Αρχικά εισάγουμε τιμές στην μνήμη για να είναι αποτελεσματικές οι πράξεις (βλέπουμε ότι στο numout υπάρχει ένας θόρυβος της κυμματομορφής γιατί εκχωρείται η τιμή




Η μηχανή που εκπροσωπεί το mode 1 στέλνει σήμα ενεργοποίησης της μηχανής που αφορά την πρόσθεση ώστε να ξεκινήσει η διαδικασία, δηλαδή να πραγματοποιηθούν αποβολές τιμών από την μνήμη – πρώτα της κορυφής και ύστερα της προηγούμενης τιμής . Την ευθύνη της εκτέλεσης αποβολής τιμών από την μνήμη την διαχειρίζεται η μηχανή που αφορά το Mode 0 – είναι η μόνη που εκτελεί push/pop η οποία ενεργοποιείται από την fsm κάθε πράξης .



Στην παρακάτω κυμματομορφή εξετάζουμε τις διεργασίες της κάθε μηχανής . Παρατηρούμε ότι το push σήμα δίνεται μόλις πατηθεί το κουμπί για το add . Από εκεί και έπειτα ενεργοποιείται το σήμα ότι ζητείται η διαδικασία της πρόσθεσης και η μηχανή push/pop ενεργοποιείται (στον επόμενο κύκλο έχουμε ModeEn σηκωμένο). Οι δύο μηχανές εκτελούνται παράλληλα στέλνοντας εντολές ενεργοποίησης σημάτων αμφότερες. Με το που δίνεται το enable δίνεται και η εντολή για την εκτέλεση του Pop στις μηχανές (το u/d του μετρητή κατεβαίνει και το enable του μετρητή είναι 1 οπότε έχει μειωθεί μια θέση στην μνήμη και ο καταχώρητης #0 ενεργοποιείται. Για λόγους συγχρονισμού των τιμών έχουμε αφήσει αρκετή διάρκεια ενεργοποιημένους τους καταχωρήτες . Θα μπορούσε να παραληφθεί). Αναμένεται και το δεύτερο σήμα αποβολής τιμής ακολουθώντας την ίδια μέθοδο μόνο που ενεργοποιείται τώρα ο καταχωρητής #1 (χρειαζόμαστε 1 κύκλο για να λάβει το σήμα) . Οι δυο καταχωρητές βρίσκονται ως είσοδοι στο προσθαφαιρέτη που μαζί με το σήμα που αναλογεί στην πρόσθεση εκτελείται η πράξη και αποθηκεύεται στον καταχωρητή (ο προσθαφαιρέτης δεν διαθέτει ρολόι, ακολουθιακό κύκλωμα, ενώ ο δεύτερος τελεστέος περνάει μέσα από πύλες XOR) ενώ δίνεται και σήμα για την εκχώρηση της τιμής του αποτελέσματος στην μνήμη.



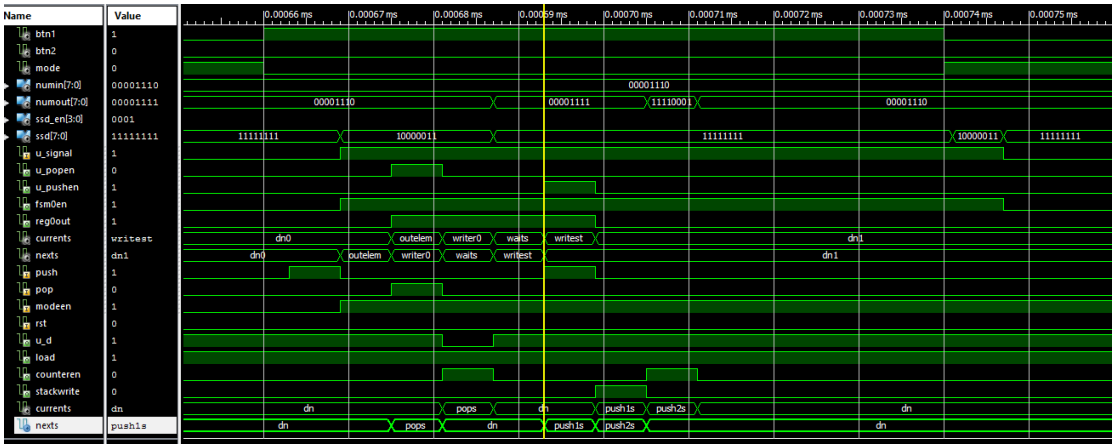
Θυμόμαστε ότι το Push περνάει από δυο φάσεις

- Τοποθέτηση τιμής
- Αύξηση μετρητή

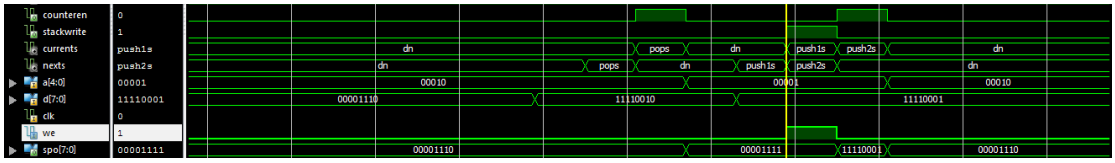
Οπότε δίνεται το σήμα και η τιμή εκχωρείται

Τα αποτελέσματα των πράξεων περνάνε από έναν πολυπλέκτη 4-1 όπου το αποτέλεσμα της πρόσθεσης και της αφαίρεσης είναι βραχυκυκλωμένο ενώ το σήμα έλεγχου είναι ένας κωδικοποιητής που έχει ως σήματα εισόδου το σήμα που δηλώνει την πράξη που διαλέγεται

Η προσομοίωση συνεχίζεται επιτελώντας την πράξη  Unary . Ομοίως υπάρχει η μηχανή του mode 2 που στέλνει σήμα στην μηχανή που αφορά την μοναδιαία αφαίρεση και εκείνη ενεργοποιεί την μηχανή push/pop . Υπενθυμίζεται ότι απαιτείται ένας κύκλος ρολογιού κάθε φορά που μια μηχανή πρέπει να στείλει μήνυμα ενεργοποίησης στην άλλη – χρειάζεται επίσης ένας κύκλος αφού ενεργοποιηθεί η μηχανή να ξεκινήσει την διαδικασία).

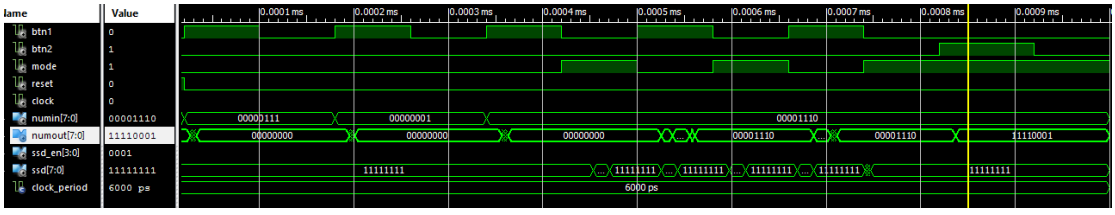



Για ModeEn=1 θέλουμε να περάσουμε την τιμή που βρίσκεται στην κορυφή της στοίβας ( το αποτέλεσμα της πρόσθεσης) σε καταχωρητή. Δίνεται η εντολή pop . Η μηχανή της πράξης επικοινωνεί με την Push/pop μηχανή  και η τιμή σώζεται στον καταχωρητή . Η μοναδιαία αφαίρεση υλοποιείται από έναν προσθαφαιρέτη που διαθέτει στον πρώτο τελεστέο τον αριθμό “00000000” και ο δεύτερος τελεστέος εισέρχεται μέσα από πύλες XOR.

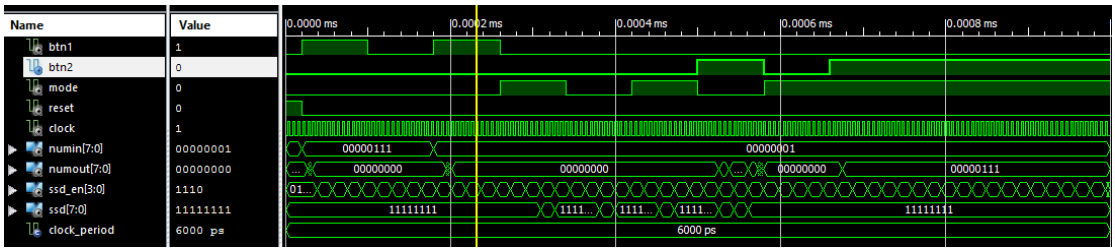



| Τελεστές | Πρόσθεση | Μοναδιαία αφαίρεση |
|---|---|---|
| 00001110 | 00001111 | 11110001 |
| 00000001 | | |

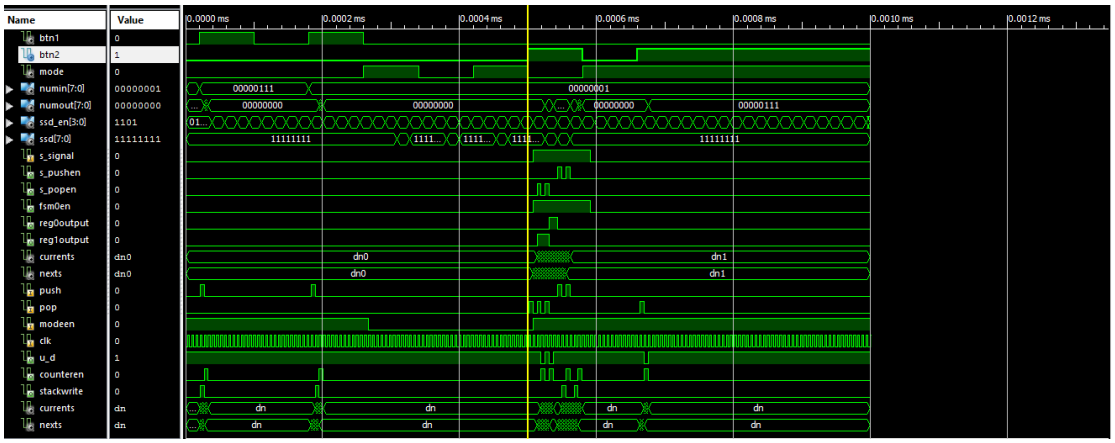
Θέλοντας να επαληθεύσουμε την αλληλουχία των πράξεων εκτελούμε μια αποβολή τιμής



Συνεχίζεται η ανάλυση του συστήματος διερευνώντας στην προσομοίωση την τελευταία πράξη της εναλλαγής τιμών



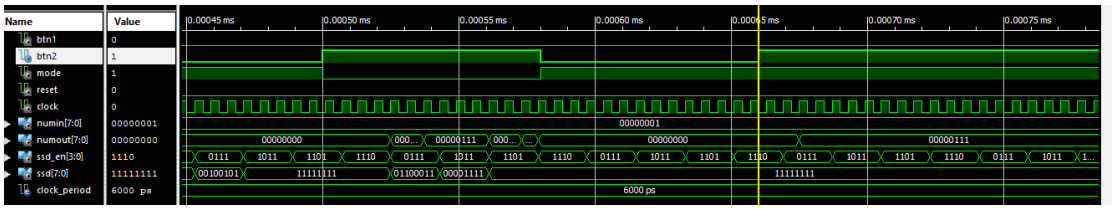

Με την ίδια μεθοδολογία ανακαλύπτουμε τις πτυχές που αναπτύσσει και η πράξη swap. Η fsm του mode 2 όταν βρίσκεται σε σύνδεση λειτουργίας και επιλεχθεί η εκπλήρωση του swap, τότε δίνεται σήμα ενεργοποίησης στην μηχανή που ανάγεται στη διεργασία της πράξης και αυτή με της σειρά της επικοινωνεί με την μηχανή push/pop.




Για την ανταλλαγή τιμών απαιτούνται 2 δράσεις pop (πρώτα της κορυφής και της προηγούμενη τιμής αυτής) και στη συνέχεια 2 δράσεις push (πρώτα της κορυφής και της τιμής κάτω της κορυφής). Μελετώντας την μηχανή για το Push/pop παρατηρούμε ότι δέχεται το σήμα pop από την μηχανή για το swap , η μηχανή έχει από εδώ και πέρα ανοιχτό enable και ξεκινάει το pop όπου το enable που βγαίνει στον counter ενεργοποιείται ενώ το up/down’ γίνεται 0 και ο μετρητής μετράει προς τα κάτω. Επαναλαμβάνεται και για το 2ο pop. Οι τιμές αποθηκεύονται στους 2ο καταχωρητές. Η διαδικασία εξακολουθεί να διεξάγεται εκτελώντας τα push . Δίνονται τα push από την μια μηχανή στην άλλη και γράφονται οι τιμές (το stack write γίνεται 1 και στο επόμενο state αυξάνεται ο counter)

Οι καταχωρητές οδηγούνται από έναν πολυπλέκτη 2 σε 1 του οποίου η έξοδος βρίσκεται ως είσοδο στον πολυπλέκτη 4-1 ώστε μια φορά να περνάει η τιμή του ενός και μία του άλλου καταχωρητή ώστε να διοχετεύονται με επάρκεια στην μνήμη.

Ελέγχουμε την αλλαγή των τιμών κάνοντας Pop μια τιμή



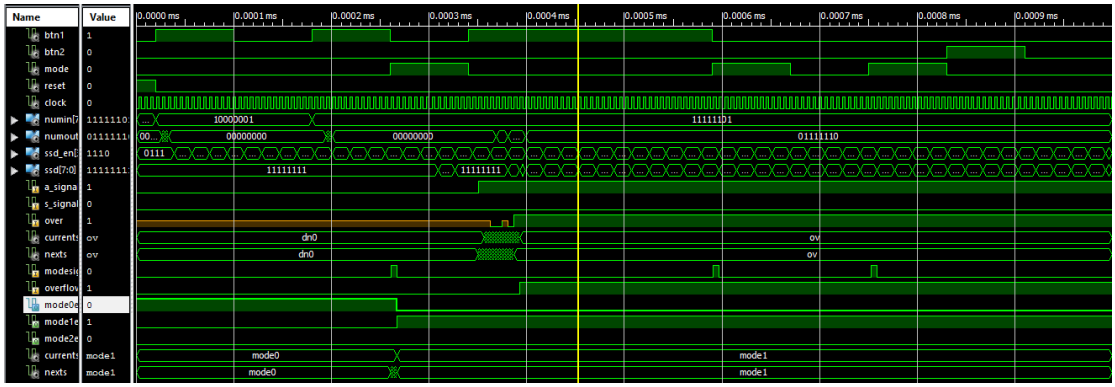

| Μνήμη πριν το Swap (σειρά από κάτω προς τα πάνω) | Μνήμη μετά το Swap (σειρά από κάτω προς τα πάνω) | Μνήμη μετά το Pop (σειρά από κάτω προς τα πάνω) |
|---|---|---|
| 00000001 | 00000111 | |
| 00000111 | 00000001 | 00000001 |

Η έξοδος του πολυπλέκτη 4-1 πηγαίνει σε ένα πολυπλέκτη 2-1 που έχει ως δεύτερη είσοδο τον αριθμό προς εισαγωγή ενώ ως σήμα ελέγχου δέχεται μια πύλη OR 3 εισόδων που έχει τα enable εξόδου που ενεργοποιούν την μηχανή Push/pop. Δηλαδή εφόσον πατήθηκε κάποια διεργασία mode εκτός του 0 τότε το αποτέλεσμα προέρχεται από τον

πολυπλέκτη 4-1 Αλλιώς γίνεται εισαγωγή κάποιου αριθμού.

Σε μια τελευταία προσομοίωση για να ελέγξουμε την ακινητοποίηση του συστήματος στο αριθμητικό overflow πραγματοποιούμε εισαγωγή 2 αριθμών ικανών να παρασκευάσουν υπερχείλιση και τους προσθέτουμε

Ιδού τα αποτελέσματα

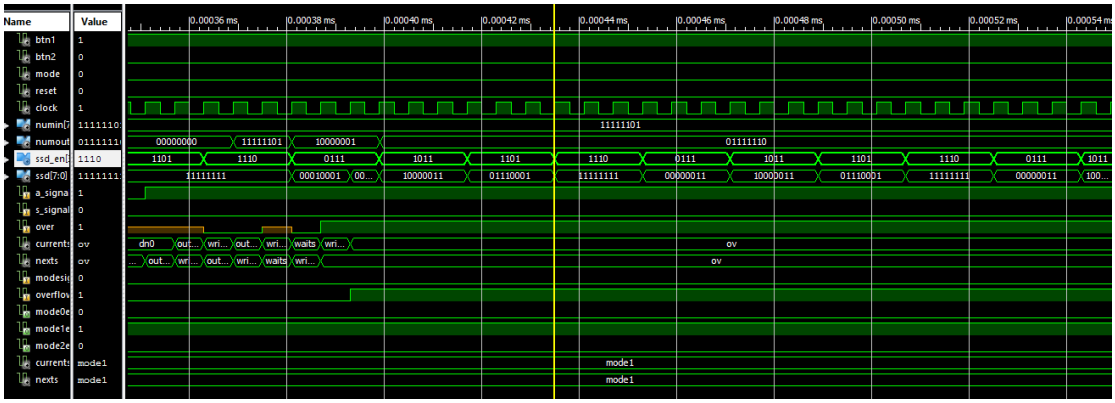



(Στην προσομοίωση έχουμε προσθέσει και την μηχανή που αλλάζει Modes αλλά και την μηχανή της πράξης add)

Διακρίνουμε τα εξής :

- Προσθέτουμε 2 αριθμούς 10000001 + 11110001 = 01111110
- Η αρμόδια fsm για τα modes παραμένει στο mode ενώ περιμένει reset για να αρχικοποιηθεί

- Παραμένουμε σε overflow ανεξάρτητα των κινήσεων
- Το seven segment display εμφανίζουν ανάλογο μήνυμα ( εμφανίζεται το 00000011100000011011000111111111 => OVF όμως ξεκινάει από το MSB σε αντίθεση με το overflow της γεμάτης στοίβας που εμφανίζει 11111111000000111000001101110001 => OVF αλλά ξεκινάει από το δεύτερο αριστερό bit




## Συμπεράσματα

Ύστερα από το πέρας της εργαστηριακής άσκησης δημιουργούνται αξιόλογα συμπεράσματα πάνω στην μελέτη και σχεδίαση υλικού . Αδιαμφισβήτητα η σχεδίαση και η επεξεργασία δεδομένων ενός προβλήματος αντιμετωπίζεται πλέον με μεγαλύτερη ευχέρεια όσον αφορά τον προγραμματισμό καθώς η χρησιμοποίηση των κατάλληλων εργαλείων σχεδίασης (μηχανές fsm, πολυπλέκτες , κωδικοποιητές και λοιπά συνδυαστικά και ακολουθιακά κυκλώματα) προσδίδει εμπειρία και ευρύτερη γνώση της δουλείας για κάθε μηχανισμό. Η συνολική σχεδίαση διαθέτει ενδιαφέροντες λειτουργικότητες (σχεδιασμός μηχανής που εκτελεί συγκεκριμένες αριθμητικές και μη λειτουργίες , 7s displays , αρχικοποίηση , εναλλαγές καταστάσεων ) που εκφράζονται σε πραγματικές δράσεις πάνω στην αναδιατασσόμενη συσκευή και παρουσιάζουν μεγάλο φάσμα των δυνατοτήτων του υλικού . Η υλοποίηση της δομής της αριθμομηχανής ολοκληρώθηκε.

## Κώδικας

- Στον κώδικα δεν περιλαμβάνονται οι FSM (ssdFsm) για τα Seven segment display και την στοίβα (FSM0,FSM1,FSM2, ModeController , ASFsm, Ufsm, SwapFSM) 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 StackOut:STD_LOGIC_VECTOR(7 DOWNTO 0):="00000000";
```

```
signal Mode0Sig,Mode1Sig,Mode2Sig,ModePipe:std_logic:='0';

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

signal UnarySubOut,SwapOut:std_logic:='0';

signal Reg0Val,Reg1Val,Reg2Val:std_logic_vector(7 downto 0);

signal RegEn0,RegEn1,RegEn2:std_logic:='0';

signal actionSignal:std_logic;

signal UnaryVal:std_logic_vector(7 downto 0);

signal boxCarry,boxV:std_logic:='0';

SIGNAL boxResult:STD_LOGIC_VECTOR(7 DOWNTO 0);

signal PopSignal,PushSignal:std_logic;

signal PopSign,PushSign:std_logic;

signal PopS,PushS:std_logic;

signal R0En,R1En:std_logic;

signal enableReg0,enableReg1:std_logic;

signal temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp9,temp10:std_logic:='0';

SIGNAL multi2v1_1Val,multi2v1_2Val,multi4v1Val:std_logic_vector(7 downto
0):="00000000";

signal FSM0En1,FSM0En2,FSM0En3:std_Logic;

signal addsubsig:std_Logic;

signal overflowSignal:std_logic;

signal signCoder:std_logic_vector(1 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 Reg is
                                    Port ( Clock : in  STD_LOGIC;
                                   D : in  STD_LOGIC_VECTOR(7 DOWNTO 0);
                                        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;
                                                      arOv: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;
overflowAr: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;
                       overflow:in std_logic;
            Mode0En : out  STD_LOGIC;
            Mode1En : out  STD_LOGIC;
            Mode2En : out  STD_LOGIC);
             end component;

          component UnaryOp is
    Port ( NIn : in  STD_LOGIC_vector(7 downto 0);
        NOut : out  STD_LOGIC_vector(7 downto 0));
             end component;

          COMPONENT ASFsm is
```

```
Port ( Clk : in  STD_LOGIC;
       Rst : in  STD_LOGIC;
  A_Signal : in  STD_LOGIC;
                 OVER:in std_logic;
  S_Signal : in  STD_LOGIC;
 AS_PushEn : out  STD_LOGIC;
 AS_PopEn : out  STD_LOGIC;
          AddSubAction:out std_logic;
         AddOrSubSignal:out std_logic;
            FSM0En:out std_logic;
            Reg2Act:out std_logic;
           overflow:out std_logic;
  Reg0Act : out  STD_LOGIC;
          Reg1Act : out  STD_LOGIC);
 end COMPONENT;


component CLA8BIT is
Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
       B : in  STD_LOGIC_VECTOR (7 downto 0);
         M : in  STD_LOGIC;
  Result : out  STD_LOGIC_VECTOR (7 downto 0);
         COUT : out  STD_LOGIC;
          V : out  STD_LOGIC);
 end component;
```

```
component UFsm is
Port ( Clock : in  STD_LOGIC;
       Reset : in  STD_LOGIC;
     U_Signal : in  STD_LOGIC;
     U_PopEn : out  STD_LOGIC;
     U_PushEn : out  STD_LOGIC;
                FSM0En:out std_logic;
     Reg0Out: out  STD_LOGIC);
     end component;

component SwapFSM is
Port ( Clk : in  STD_LOGIC;
       Rst : in  STD_LOGIC;
     S_Signal : in  STD_LOGIC;
     S_PushEn : out  STD_LOGIC;
                FSM0En:out std_logic;
     S_PopEn : out  STD_LOGIC;
     Reg0Output : out  STD_LOGIC;
     Reg1Output : out  STD_LOGIC);
     end component;

COMPONENT Multiplexer2v1 is
Port ( Q0 : in  STD_LOGIC_VECTOR (7 downto 0);
       Q1 : in  STD_LOGIC_VECTOR (7 downto 0);
       Sign : in  STD_LOGIC;
     OUTM : out  STD_LOGIC_VECTOR (7 downto 0));
```

```
end COMPONENT;

component Multiplexer4v1 is
Port ( Q0 : in  STD_LOGIC_VECTOR (7 downto 0);
       Q1 : in  STD_LOGIC_VECTOR (7 downto 0);
       Q2 : in  STD_LOGIC_VECTOR (7 downto 0);
       Q3 : in  STD_LOGIC_VECTOR (7 downto 0);
      Sign : in  STD_LOGIC_VECTOR (1 downto 0);
     OUTM : out  STD_LOGIC_VECTOR (7 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

coderSign:Coder
port map(Input(0)=>NOT AddOut,
                Input(1)=>NOT SubOut,
               Input(2)=>NOT UnarySubOut,
                Input(3)=>NOT SwapOut,
                 Output=>signCoder);
```

```
SFsm:SwapFSM
Port map(Clk=>Clock,
Rst=>Reset,
S_Signal=>SwapOut,
S_PopEn=>PopS,
FSM0En=>FSM0En3,
S_PushEn=>PushS,
Reg0Output=>enableReg0,
Reg1Output=>enableReg1);

Multi4v1:Multiplexer4v1
port map(Q0=>Reg2Val,
Q1=>Reg2Val,
Q2=>UnaryVal,
Q3=>multi2v1_1Val,
Sign=>signCoder,
OUTM=>multi4v1Val);

UnFsm:UFsm
Port map(Clock=>Clock,
Reset=>Reset,
U_Signal=>UnarySubOut,
U_PopEn=>PopSign,
U_PushEn=>PushSign,
```

```
                                FSM0En=>FSM0En2,
                            Reg0Out=>R0En);

    temp9<=FSM0En1 or FSM0En3 or FSM0En2;
            StackMultiplexer:Multiplexer2v1
                port map(Q0=>NUMIN,
                                Q1=>multi4v1Val,
                                  Sign=>temp9,
                               OUTM=>multi2v1_2Val);

            swapMultiplexer:Multiplexer2v1
                port map(Q0=>Reg0Val,
                                Q1=>Reg1Val,
                               Sign=>enableReg1,
                              OUTM=>multi2v1_1Val);


                  UnaryACT: UnaryOp
                   PORT MAP(NIn=>Reg0Val,
                                    NOut=>UnaryVal);


      temp4<=RegEn0 or R0En or enableReg0;
                          Reg0: Reg
                        PORT MAP(Clock => Clock,
                                    D => StackOut,
```

```
                                        Enable =>temp4,
                                        Q => Reg0Val);

temp5<= RegEn1 or R1En or enableReg1;
                Reg1: Reg
              PORT MAP(Clock => Clock,
                                        D => StackOut,
                                        Enable =>temp5,
                                        Q => Reg1Val);

                Reg2: Reg
              Port map(ClOCK=>Clock,
                                        D=>boxResult ,
                                        Enable=>RegEn2,
                                        Q=>Reg2Val);

            AddSubFSM: ASFsm
              PORT MAP(Clk=>Clock,
                                        Rst=>Reset,
                                        A_Signal=>AddOut,
                                        S_Signal=>SubOut,
                                        OVER=>boxV,
```

```
                AS_PushEn=>PushSignal,
                AS_PopEn=>PopSignal,
                FSM0En=>FSM0En1,
               AddSubAction=>actionSignal,
               AddOrSubSignal=>addsubsig,
                Reg0Act=>RegEn0,
               overflow=>overflowSignal,
                Reg2Act=>RegEn2,
                Reg1Act=>RegEn1);

AddSubBox: CLA8BIT
    PORT MAP(A=>Reg0Val,
          B=>Reg1Val,
         M=>actionSignal,
        Result=>boxResult,
         COUT=>boxCarry,
          V=>boxV);

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);

temp1<=     BTN1P or  (PushSignal or PushSign or PushS) ;
temp2<=     BTN2P or   (PopSignal or PopSign or PopS);
temp3<=     Mode0Sig OR FSM0En1 or FSM0En2 or FSM0En3;
        FSM_M0 : FSM0
        PORT MAP( Push => temp1,
                  Pop => temp2,
                  Clk => Clock,
                  ModeEn=>temp3,
                  Rst => Reset,
                  U_D => UDP,
                  Empty => EmptySignal,
                  arOv=>overflowSignal,
```

```
                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 AND (NOT overflowSignal),
                        Swap=>BTN2P AND (NOT overflowSignal),
                        UnarySubOutput=>UnarySubOut,
                        SwapOutput=>SwapOut);

    temp10<=overflowSignal or OVFP;
        ModeController:ModeFsm
```

```
Port map(Clock=>Clock,
                Reset=>Reset,
            ModeSignal=>ModePipe,
              overflow=>temp10,
              Mode0En=>Mode0Sig,
              Mode1En=>Mode1Sig,
              Mode2En=>Mode2Sig);


myStack : STACK
PORT MAP( a => out_tos, --<<
                d =>multi2v1_2Val,
                  clk => Clock,
                we => WriteEnable,
                 spo => StackOut);


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,
overflowAr=>overflowSignal,
ANControl => SSD_EN,
SSControl => SSD);

secondCounterEnable :PROCESS(EmptySignal,UDP,out_tos)
BEGIN
IF ((EmptySignal = '1')or (UDP='0' AND out_tos="00001")) THEN --<<<<<
FlagEnable <= '0';
ELSE
FlagEnable <= '1';
END IF;
END PROCESS;

NUMOUT<=StackOut;
SecCounEn <= CounterEnable AND FlagEnable;

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;
```

# MULTIPLEXER 4-1

---

---

-- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Multiplexer4v1 is

Port ( Q0 : in  STD_LOGIC_VECTOR (7 downto 0);

Q1 : in  STD_LOGIC_VECTOR (7 downto 0);

Q2 : in  STD_LOGIC_VECTOR (7 downto 0);

Q3 : in  STD_LOGIC_VECTOR (7 downto 0);

Sign : in  STD_LOGIC_VECTOR (1 downto 0);

OUTM : out  STD_LOGIC_VECTOR (7 downto 0));

end Multiplexer4v1;

architecture Behavioral of Multiplexer4v1 is

begin

process(Sign,Q0,Q1,Q2,Q3)

BEGIN

if(Sign="00") then

OUTM<=Q0;

ELSIF Sign<="01" then

```
                OUTM<=Q1;
ELSIF Sign<="10" then
                OUTM<=Q2;
ELSIF Sign<="11" then
                OUTM<=Q3;
ELSE OUTM<=Q0;
END IF;
END PROCESS;
end Behavioral;
```

## MULTIPLEXER 2-1

---

---

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

entity Multiplexer2v1 is
Port ( Q0 : in  STD_LOGIC_VECTOR (7 downto 0);
       Q1 : in  STD_LOGIC_VECTOR (7 downto 0);
       Sign : in  STD_LOGIC;
       OUTM : out  STD_LOGIC_VECTOR (7 downto 0));
end Multiplexer2v1;
```

```
architecture Behavioral of Multiplexer2v1 is

                    begin

                process(Q0,Q1,Sign)

                    begin

                if Sign='0' then

                        OUTM<=Q0;

              ELSIF Sign='1' then

                        OUTM<=Q1;

                        ELSE

                  OUTM<="00000000";

                    END IF;

                END PROCESS;



                end Behavioral;
```

## UNARY SUBTRACTOR

---

---

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
```

```
entity UnaryOp is
Port ( NIn : in  STD_LOGIC_vector(7 downto 0);
    NOut : out  STD_LOGIC_vector(7 downto 0));
end UnaryOp;

architecture Behavioral of UnaryOp is
COMPONENT CLA8BIT is
Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
    B : in  STD_LOGIC_VECTOR (7 downto 0);
    M : in  STD_LOGIC;
  Result : out  STD_LOGIC_VECTOR (7 downto 0);
    COUT : out  STD_LOGIC;
    V : out  STD_LOGIC);
end component;

begin
complement2s : CLA8BIT PORT MAP(A=>"00000000",
                                B=>NIn,
                                M=>'1',
                              Result=>NOut);

end Behavioral;
```

# REGISTER

---

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


entity 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 Reg;

architecture Behavioral of Reg is

begin
process(Clock,Enable)
begin
If rising_edge(Clock) then
if Enable='1' then
Q<=D;
end if;
end if;
end process;
```

```
end Behavioral;
```

## ADDER-SUBTRACTOR

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

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity CLA8BIT is

Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);

       B : in  STD_LOGIC_VECTOR (7 downto 0);

       M : in  STD_LOGIC;

  Result : out  STD_LOGIC_VECTOR (7 downto 0);

    COUT : out  STD_LOGIC;

       V : out  STD_LOGIC);

end CLA8BIT;


architecture Behavioral of CLA8BIT is

component CLA is

Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);

       B : in  STD_LOGIC_VECTOR (3 downto 0);

                                  M : in STD_LOGIC;

                             Cin : in  STD_LOGIC;

                        SUM : out  STD_LOGIC_VECTOR (3 downto 0);

                                    V:out std_logic;

                                  Cout: out STD_LOGIC

                                  );

end component;
```

```
signal Csignal:std_logic;
begin
CLA1: CLA PORT MAP (A(0)=>A(0),
                    A(1)=>A(1),
                    A(2)=>A(2),
                    A(3)=>A(3),
                    B(0)=>B(0),
                    B(1)=>B(1),
                    B(2)=>B(2),
                    B(3)=>B(3),
                      M=>M,
                    Cin=>M,
                  SUM(0)=>Result(0),
                  SUM(1)=>Result(1),
                  SUM(2)=>Result(2),
                  SUM(3)=>Result(3),
                 Cout=>Csignal
                        );

CLA2: CLA PORT MAP (A(0)=>A(4),
                    A(1)=>A(5),
                    A(2)=>A(6),
                    A(3)=>A(7),
                    B(0)=>B(4),
                    B(1)=>B(5),
                    B(2)=>B(6),
```

```
B(3)=>B(7),
M=>M,
Cin=>Csignal,
SUM(0)=>Result(4),
SUM(1)=>Result(5),
SUM(2)=>Result(6),
SUM(3)=>Result(7),
Cout=>COUT,
V=>V);

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;

## 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;
          overflowAr: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;
overflowAr: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,
                overflowAr=>overflowAr,
                    SWAP=>SWAP,
                    Mode0=>Mode0,
                    Mode1=>Mode1,
                    Mode2=>Mode2,
                segmentLED => muxInPipe);

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

```
ANControl <= FSMOutPipe;

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;
```