ΠΡΟΧΩΡΗΜΕΝΗ ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ

ΑΝΑΦΟΡΆ 3ου ΕΡΓΑΣΤΗΡΙΟΥ

LAB 20335359

Γιώργος Βιριράκης 2016030035
Μιχάλης Γαλάνης 2016030036

ΣΚΟΠΟΣ

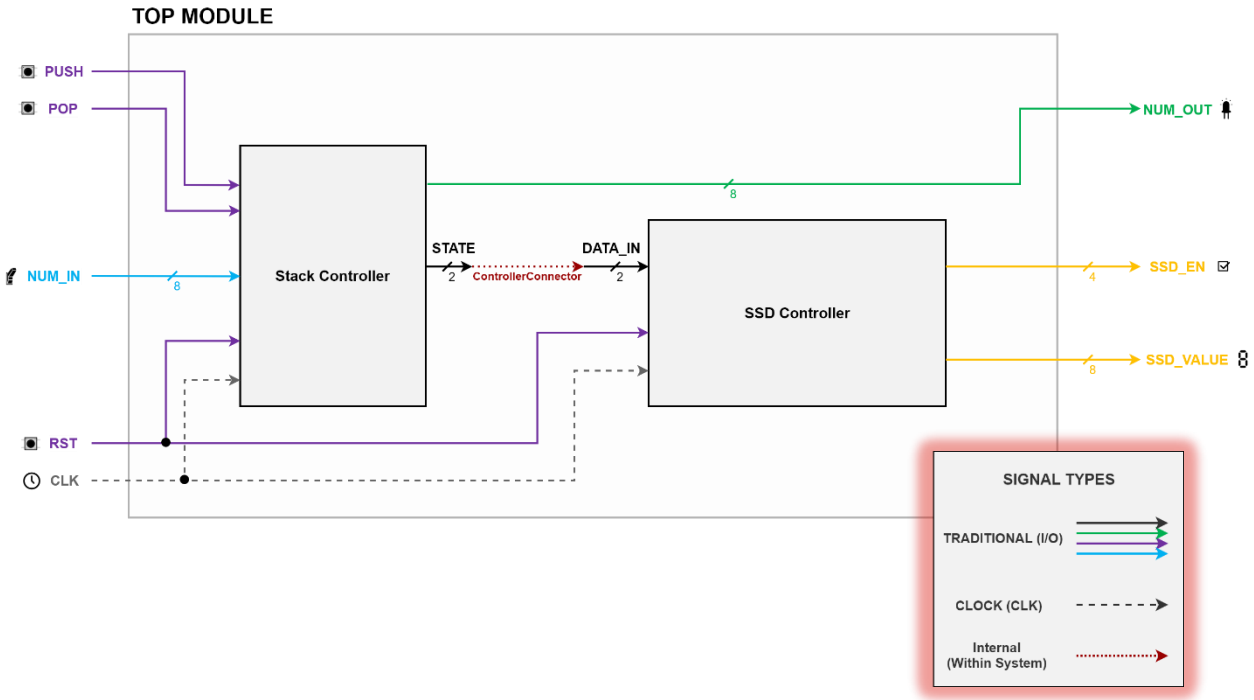
Σκοπός του 3ου εργαστηρίου ήταν η περαιτέρω εμβάθυνση και εξοικείωση με την λειτουργία της VHDL. Συγκεκριμένα δόθηκε έμφαση στη δημιουργία και χρήση της στοίβας (Stack) , συγκεκριμένα μιας **pre-increment, post-decrement** στοίβας, τη διαχείρηση της με λειτουργίες **push/pop** καθώς και την επικοινωνία της με τα περιφερειακά modules. Τέλος, γνωρίσαμε τα **Seven Segment Display (SSD)** που μας απεικονίζουν πληροφορία σχετικά με τη κατάσταση της στοίβας **(Empty, Full, Overflow)**.

ΠΡΟΕΤΟΙΜΑΣΙΑ - ΠΕΡΙΓΡΑΦΗ

Οι είσοδοι και έξοδοι του κυκλώματος παρουσιάζονται στον παρακάτω πίνακα:

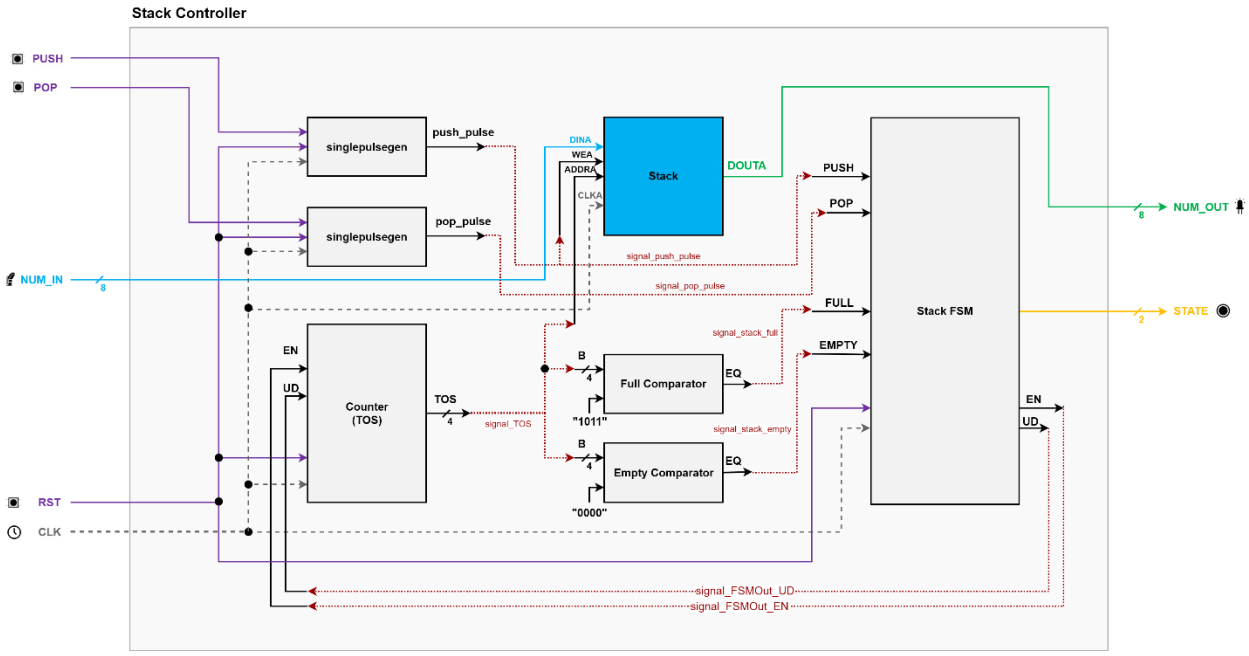
Name	IN / OUT	No. of Bits	FPGA Pins
Clock	in	1	MCLK
Push	in	1	BTN_0
Pop	in	1	BTN_1
Unknown	in	1	BTN_2
Reset	in	1	BTN_3
Num_In	in	8 (Bus)	$SW_{[7:0]}$
Num_Out	out	8 (Bus)	$LD_{[7:0]}$
SSD_En	out	4 (Bus)	$AN_{[3:0]}$
SSD_Value	out	8 (bus)	$SEG_{[7:0]}$

Το κύκλωμα της εργαστηριακής άσκησης χωριζόταν σε δυο μεγάλα υποσυστήματα **(controllers)**, τον **Stack Controller** και τον **SSD Controller.** Σε επόμενο σχήμα διαφαίνεται η structural δομή στα υψηλότερα δύο υψηλότερα επίπεδα του κυκλώματος.

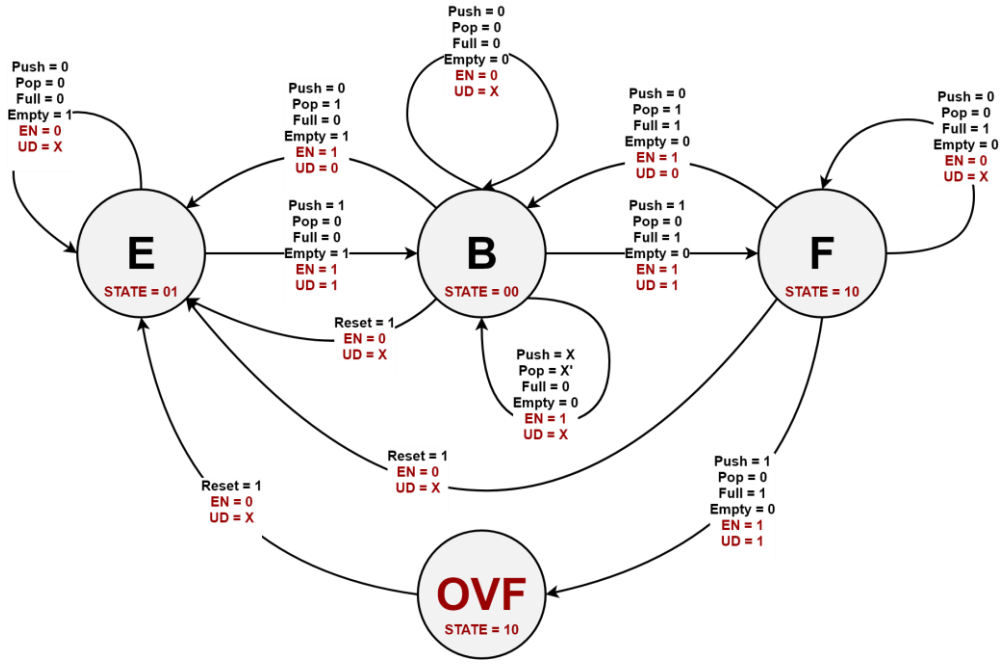


Ο πρώτος controller **(Stack Controller)** ήταν υπεύθυνος για τη διαχείρηση της στοίβας: **εισαγωγή (εγγραφή)** – **διαγραφή (ανάγνωση)** στοιχείων με λειτουργία μεμωνομένων παλμών **push / pop** και η αναγνώριση της κατάστασης της στοίβας **(Empty, Full, Overflow)**. Η μέτρηση στοιχείων της στοίβας επιτυγχάνοταν με **μετρητή** και ο έλεγχος για το αν ήταν άδεια/γεμάτη γινόταν με **συγκριτή.** Η αναγνώριση της κατάστασης της στοίβας γενικότερα ήταν δυνατή λόγω μιας μηχανής πεπερασμένων καταστάσεων. Τέλος, βοηθητικά modules όπως **singlepulsegen** ήταν χρήσιμα για την εξαγωγή μεμωνομένων παλμών ρολογιού.

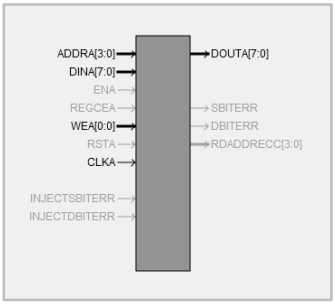
Παρακάτω παρουσιάζεται αναλυτικά η δομή του Stack Controller και τα components που αποτελείται:



Η Stack FSM είναι τύπου Mealy και το διάγραμμα καταστάσεων της είναι το ακόλουθο:



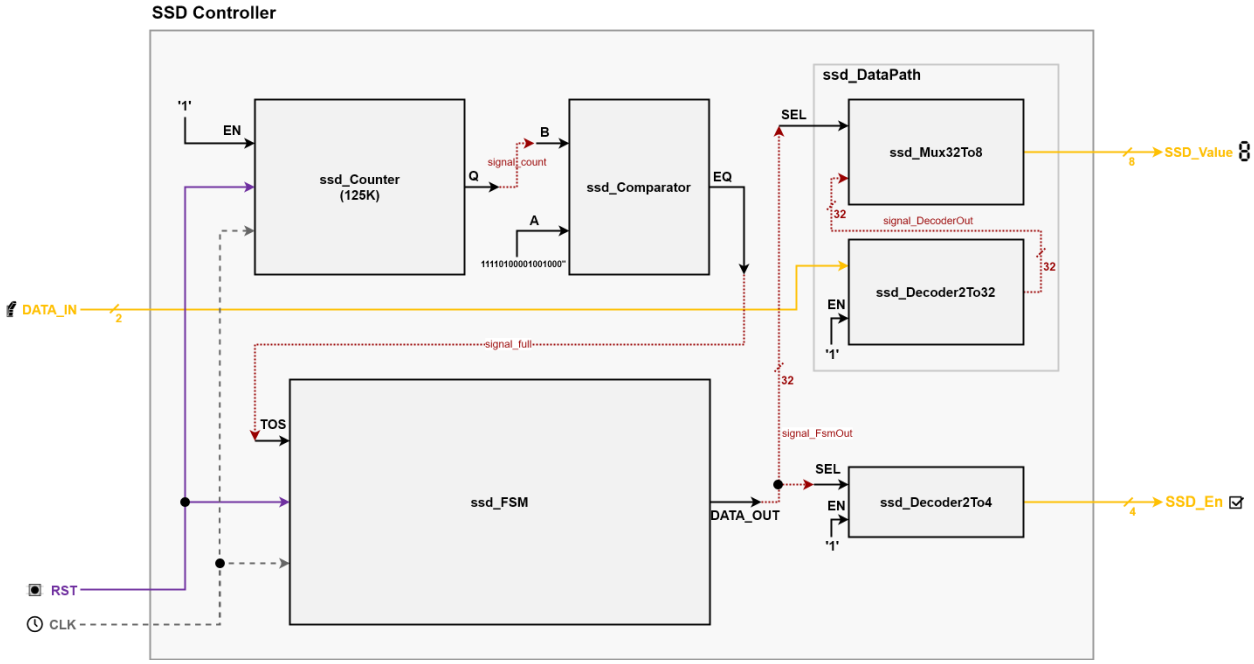
Η μνήμη Stack δημιουργήθηκε από το εργαλείο **Memory Block Generator** του Xilinx και έχει πλάτος 8 bit και βάθος 12 στοιχείων.



Ο δεύτερος controller **(SSD Controller)** ήταν υπεύθυνος για τη διαχείριση των **seven segment displays.** Το ζητούμενο είναι ανάλογα με τη κατάσταση της στοίβας να εκτυπώνονται και τα κατάλληλα γράμματα στα SSD όπως **"- - - E"**, **"- - - -"**, **"- - - F"**, **"- O V F".** Το κάθε SSD ενεργοποιείται με βάση ένα control bit **($AN_{[3:0]}$)** και τα ssd έχουν κοινό data bus (8-bit) για τα segment τους.

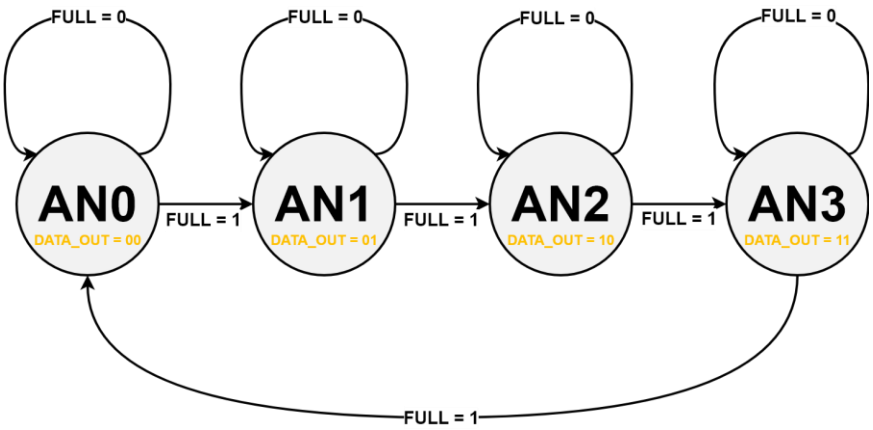
Το πρόβλημα είναι πως το **Spartan 3E Basys 2 FPGA** μπορεί να ενεργοποιήσει **μόνο ένα SSD κάθε φορά**. Η λύση σε αυτό το πρόβλημα είναι η ενεργοποίησή τους διαδοχικά σε ρυθμό που το ανθρώπινο μάτι δε θα καταλάβαινε τη διαφορά με το να ήταν ενεργοποιημένα ταυτόχρονα. Το παραπάνω επιτυγχάνεται με έναν μετρητή ο οποίος μετραέι συνεχόμενα και επαναληπτικά μέχρι έναν αριθμό (για την άσκηση αυτή, τον θέτουμε **125.000**) και μια μηχανή πεπερασμένων καταστάσεων η οποία παριστάνει τα 4 control bits και μεταβαίνει στο επόμενο για κάθε επανάληψη του μετρητή. Χρειαζόμαστε φυσικά και ένα συγκριτή ο οποίος θα ελέγχει πότε ο μετρητής φτάσει τον αριθμό για να αλλάξει η κατάσταση στην FSM.

Παρακάτω επισυνάπτεται η αναλυτική δομή του SSD Controller:



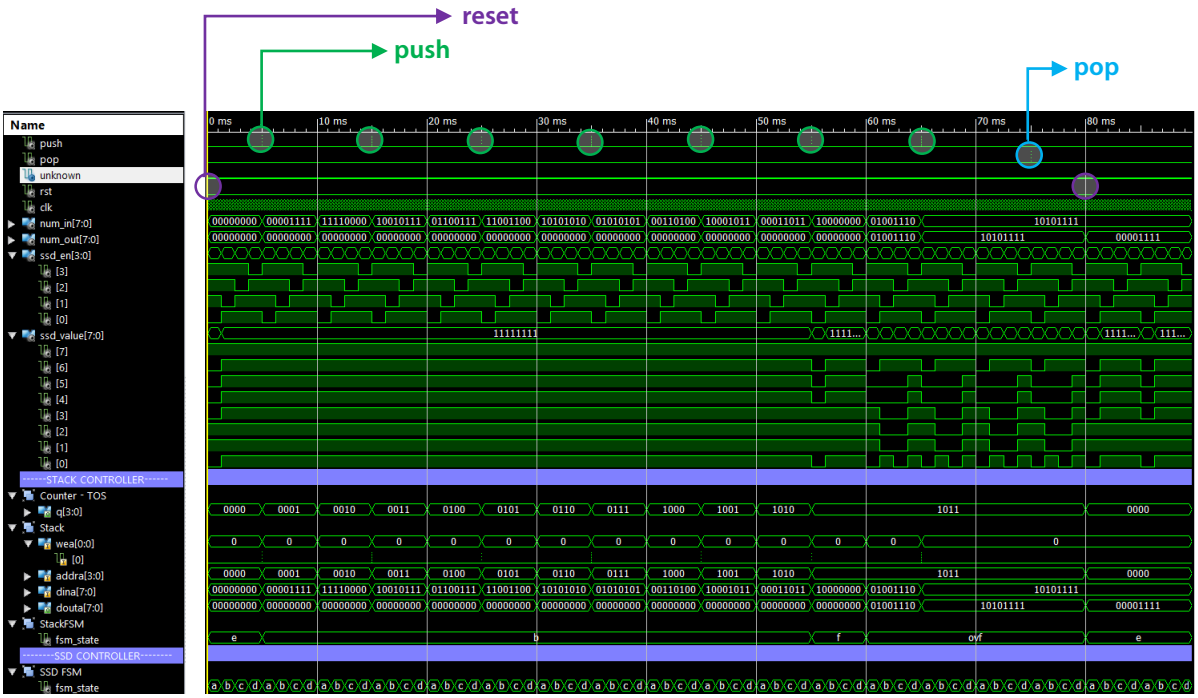
Παρατήρηση: Επειδή τα **ssd_Mux32To8** και **ssd_Decoder2To32** διαμορφώνουν τα segments, τα ομαδοποιήσαμε σε ένα structural **ssd_dataPath.**

Αυτή τη φορά η **ssd_FSM** είναι τύπου Moore και οι εξόδους της είναι συγκεκριμένες σε κάθε κατάσταση:

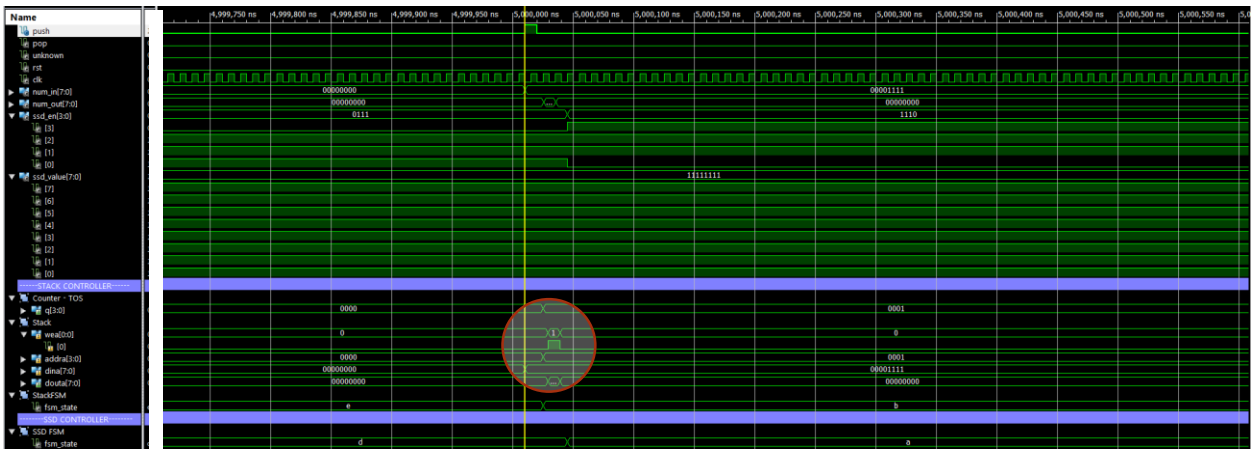


ΚΥΜΑΤΟΜΟΡΦΕΣ

Θεωρήσαμε ότι το simulation του TOP MODULE με εσωτερικά σήματα που αναδεικνύουμε είναι αρκετό για να περιγράψουμε πλήρως το κύκλωμά μας.



Για να σιγουρευτούμε ότι η στοίβα είναι pre increment, πρέπει να μεγενθύνουμε τη κυματομορφή τη στιγμή που επιτυγχάνεται το push. Στη παρακάτω εικόνα, φαίνεται ότι το **WEA** γίνεται '1' ένα κύκλο αργότερα από την έξοδο του μετρητή **($q_{[0:3]}$).**



ΣΥΜΠΕΡΑΣΜΑΤΑ

Στο εργαστήριο αυτό, υλοποιήσαμε μια στοίβα **pre-increment**, **post-decrement** με όλες τις λειτουργίες της (push/pop) και μάθαμε να διαχειριζόμαστε τα seven segment displays απεικονίζοντας σε αυτά πληροφορίες σχετικά με τη στοίβα.

ΠΑΡΑΡΤΗΜΑ – ΚΩΔΙΚΑΣ

TOP MODULE

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

entity TOP_MODULE is
    Port ( PUSH : in  STD_LOGIC;
           POP : in  STD_LOGIC;
           UNKNOWN : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           NUM_IN : in  STD_LOGIC_VECTOR (7 downto 0);
           NUM_OUT : out  STD_LOGIC_VECTOR (7 downto 0);
           SSD_EN : out  STD_LOGIC_VECTOR (3 downto 0);
           SSD_VALUE : out  STD_LOGIC_VECTOR (7 downto 0));
end TOP_MODULE;

architecture Structural of TOP_MODULE is
signal ControllerConnector : STD_LOGIC_VECTOR (1 downto 0);

Component STACK_CONTROLLER is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           PUSH : in  STD_LOGIC;
           POP : in  STD_LOGIC;
           NUM_IN : in  STD_LOGIC_VECTOR (7 downto 0);
           NUM_OUT : out  STD_LOGIC_VECTOR (7 downto 0);
           STATE : out STD_LOGIC_VECTOR (1 downto 0));
end Component;

Component SSD_CONTROLLER is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           DATA_IN : in  STD_LOGIC_VECTOR (1 downto 0);
           DATA_OUT : out STD_LOGIC_VECTOR (7 downto 0);
           AN : out STD_LOGIC_VECTOR (3 downto 0));
end Component;

begin

STACK : STACK_CONTROLLER port map ( CLK => CLK,
                                    RST => RST,
                                    PUSH => PUSH,
                                    POP => POP,
                                    NUM_IN => NUM_IN,
                                    NUM_OUT => NUM_OUT,
                                    STATE => ControllerConnector
                                       );

SSD : SSD_CONTROLLER port map (  CLK => CLK,
                                 RST => RST,
                                 DATA_IN => ControllerConnector,
                                 DATA_OUT => SSD_VALUE,
                                 AN => SSD_EN
                                 );



end Structural;

```

Stack Controller

```
14  architecture Structural of STACK_CONTROLLER is
15
16  signal signal_push_pulse, signal_pop_pulse : STD_LOGIC; --Needed signals for clock pulses
17  signal signal_TOS : STD_LOGIC_VECTOR (3 downto 0);
18  signal signal_stack_full, signal_stack_empty : STD_LOGIC;
19  signal signal_FSMOut_EN, signal_FSMOut_UD : STD_LOGIC;
20
21
22  Component singlepulsegen is
23  Port ( clk     : in std_logic;
24         rst     : in std_logic;
25         input   : in std_logic;
26         output  : out std_logic
27       );
28  end Component;
29
30  Component StackFSM is
31  Port ( CLK     : in std_logic;
32         RST     : in std_logic;
33         PUSH    : in std_logic;
34         POP   : in std_logic;
35         FULL    : in std_logic;
36         EMPTY   : in std_logic;
37         EN    : out std_logic;
38         UD    : out  std_logic;
39         STATE : out std_logic_vector (1 downto 0)
40       );
41  end Component;
42
43  Component Comparator_4Bit is
44  Port ( A       : in  STD_LOGIC_VECTOR (3 downto 0);
45         B       : in  STD_LOGIC_VECTOR (3 downto 0);
46         EQ      : out  STD_LOGIC
47       );
48  end Component;
49
50  Component Counter is
51  Port ( CLK : in  STD_LOGIC;
52         RST : in  STD_LOGIC;
53         EN  : in  STD_LOGIC;
54         U_D : in STD_LOGIC;
55         Q : out  STD_LOGIC_VECTOR(3 downto 0)
56           );
57  end Component;
58
59  Component Stack is
60  Port ( clka : in STD_LOGIC;
61         wea : in STD_LOGIC_VECTOR(0 downto 0);
62         addra: in STD_LOGIC_VECTOR(3 downto 0);
63         dina: in STD_LOGIC_VECTOR(7 downto 0);
64         douta: out STD_LOGIC_VECTOR(7 downto 0)
65         );
66  end Component;
67
68  begin
69
70  SPGEN_PUSH: singlepulsegen port map(   clk => CLK,
71                                         rst => RST,
72                                         input => PUSH,
73                                         output => signal_push_pulse
74                                     );
75
76  SPGEN_POP: singlepulsegen port map( clk => CLK,
77                                      rst => RST,
78                                      input => POP,
79                                      output => signal_pop_pulse
80                                    );
81
82
83  TOS: Counter port map( CLK => CLK,
84                         RST => RST,
85                         U_D => signal_FSMOut_UD,
86                         EN => signal_FSMOut_EN,
87                         Q => signal_TOS
88                         );
89
90
91  FullComparator: Comparator_4Bit port map( A => "1011",
92                                            B => signal_TOS,
93                                            EQ => signal_stack_full
94                                            );
95
96  EmptyComparator: Comparator_4Bit port map( A => "0000",
97                                             B => signal_TOS,
98                                             EQ => signal_stack_empty
99                                             );
100
101 Stack_1: Stack port map( clka => CLK,
102                          wea(0) => signal_push_pulse,
103                          addra => signal_TOS,
104                          dina => NUM_IN,
105                          douta => NUM_OUT
106                          );
107
108
109
110
111 FSM: StackFSM port map ( CLK => CLK,
112                           RST => RST,
113                           PUSH => signal_push_pulse,
114                           POP => signal_pop_pulse,
115                           FULL => signal_stack_full,
116                           EMPTY => signal_stack_empty,
117                           EN => signal_FSMOut_EN,
118                           UD => signal_FSMOut_UD,
119                           STATE => STATE
120                          );
121
122 end Structural;
```

SSD Controller

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_unsigned.all;

entity SSD_CONTROLLER is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           DATA_IN : in  STD_LOGIC_VECTOR (1 downto 0);
           DATA_OUT : out STD_LOGIC_VECTOR (7 downto 0);
           AN : out STD_LOGIC_VECTOR (3 downto 0));
end SSD_CONTROLLER;

architecture Structural of SSD_CONTROLLER is

signal signal_count : std_logic_vector (16 downto 0); -- Q of 125K-Counter
signal signal_full : std_logic; --output of comparator
signal signal_FsmOut : std_logic_vector (1 downto 0); --Output of FSM

Component ssd_Counter is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           EN : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (16 downto 0));
end Component;


Component ssd_Comparator is
    Port ( A,B : in STD_LOGIC_VECTOR(16 downto 0);
           EQ : out  STD_LOGIC);
end Component;


Component ssd_Fsm is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           FULL : in  STD_LOGIC;
           DATA_OUT : out  STD_LOGIC_VECTOR (1 downto 0));
end Component;


Component ssd_Decoder2To4 is
    Port ( SEL : in  STD_LOGIC_VECTOR (1 downto 0);
           EN : in  STD_LOGIC;
           AN : out  STD_LOGIC_VECTOR (3 downto 0));
end Component;

Component ssd_DataPath is
    Port ( STATE : in  STD_LOGIC_VECTOR (1 downto 0);
           SEL : in  STD_LOGIC_VECTOR (1 downto 0);
           SEG : out  STD_LOGIC_VECTOR (7 downto 0));
end Component;


begin


FullComparator : ssd_Comparator port map ( --A is 125000 in binary
                                           --11110100001001000 is 125K in binary
                                           A => "11110100001001000",
                                           B => signal_count,
                                           EQ =>   signal_full
                                               );

Counter125K : ssd_Counter port map( CLK => CLK,
                                   RST => RST,
                                   EN => '1',
                                   Q => signal_count
                                      );


Fsm : ssd_Fsm port map ( CLK => CLK,
                         RST => RST,
                         FULL => signal_full,
                         DATA_OUT => signal_FsmOut
                             );

DataPath : ssd_DataPath port map ( STATE => DATA_IN,
                                   SEL => signal_FsmOut,
                                   SEG => DATA_OUT
                                       );


AnSelector : ssd_Decoder2To4 port map ( SEL => signal_FSMOut,
                                        EN => '1',
                                        AN => AN
                                       );

end Structural;
```