4η Εργαστηριακή Άσκηση

ΑΝΑΓΝΩΡΙΣΗ ΠΡΑΞΕΩΝ ΓΙΑ ΣΧΕΔΙΑΣΗ ΜΙΑΣ ΑΡΙΘΜΟΜΗΧΑΝΗΣ

Ομάδα LAB20332061

ΛΑΜΠΡΙΑΝΗ ΤΣΑΠΑΝΟΥ 2014030015
ΑΝΑΣΤΑΣΙΟΣ ΜΠΟΚΑΛΙΔΗΣ 2014030069

Σκοπός εργαστηριακής άσκησης

Σκοπός του 4ου εργαστηρίου ήταν να επεκτείνουμε την λειτουργικότητα του 3ου εργαστηρίου έτσι ώσε να υποστηρίζονται 7 λογικές και αριθμητικές πράξεις (Push, Pop, Sub, Unary Sub, Εναλλαγη και Reset) μόνο με 4 πλήκτρα. Αυτό θα επιτευχθεί μέσω του BTN2 το οποιό λειτουργεί σαν mode, δηλαδή σαν μετάβαση σε διαφορετικό menu. Πλέον στα LED του 7-segment display θα απεικονίζεται η πρηροφορία Empty, Full, Overflow, Add, Sub, Unary Sub, Εναλλαγή καθώς και σε ποιο mode βρισκόμαστε εκείνη την χρονική στιγμή (E, F, OVF, A, S, U, [], 1 και 2). Σε αυτό το εργαστήριο θα επιτευχθεί μόνο η λειτουργικότητα του menu και όχι των αντίστοιχων επιλογών αυτού.

Προεργασία – Περιγραφή

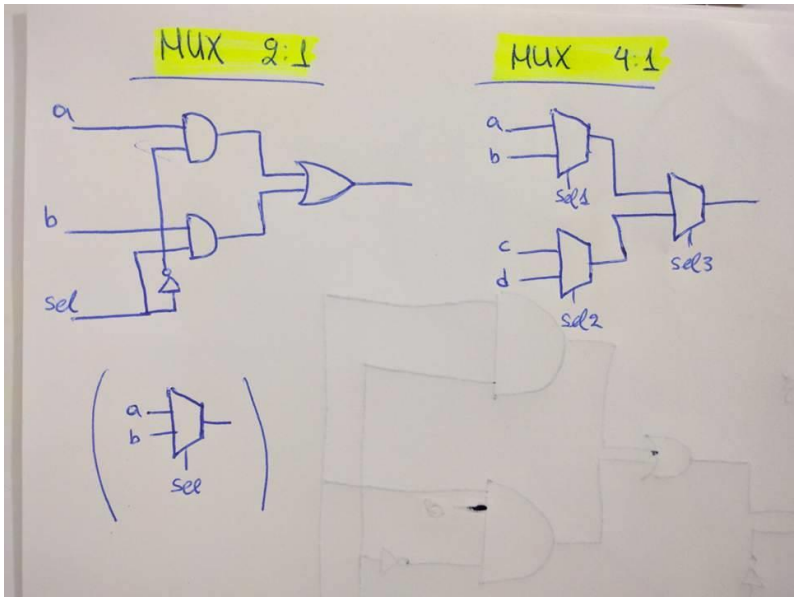
Το κύκλωμά μας αποτελείται από τα παρακάτω υποκυκλώματα:

- Την στοίβα (**Stack**) με μνήμη πλάτους 8-bit και μήκους 32-bit, η οποία μπορεί να αποθηκεύσει μέχρι και 31 στοιχεία. Υλοποιήθηκε αυτόματα μέσω του Xilinx Core Generator
- **5-bit Counter** (μετρητή) όπου χρειάζεται για το TOS και το TOS-1 (έναν counter για το καθένα). Τον υλοποιήσαμε σε structural μορφή. Για την υλοποίηση του counter χρησιμοποιήθηκαν **MUX 4:1** οι οποίες υλοποιήθηκαν μέσω της χρήσης **MUX 2:1**.
- Τον συγκριτή (**Comparator**). Οι 2 comparator του κυκλώματός μας έλεγχαν αν η στοίβα μας βρικόταν σε κατάσταση Empty ή Full.
- Την **FSM2** οπού διαβάζει το input του BTN2 (mode) και αλλάζει το mode που βρισκόμαστε και έχει ως έξοδο 00 για mode_0, 01 για mode_1 και 10 για mode_2.
- Την **FSM1** η οποία δέχεται ως είσοδο την έξοδο της FSM2, το output του BTN1 (λειτουργία ανάλογα το mode), BTN3 (reset) καθώς και την κατάσταση που βρίσκεται εκείνη την στιγμή η στοίβα μας (Empty, Full) από τις εξόδους των comparator. Έτσι ελέγχει σε ποια λειτουργία θα μεταβούμε ελέγχοντας τις εισόδους των counter.
- Το entity του **SSD** όπου αποτελείται από:
  Το control_path: ssd_counter, ssd_comparator, ssd_fsm
  Το data_path: ssd_encoder, ssd_decoder, ssd_mux

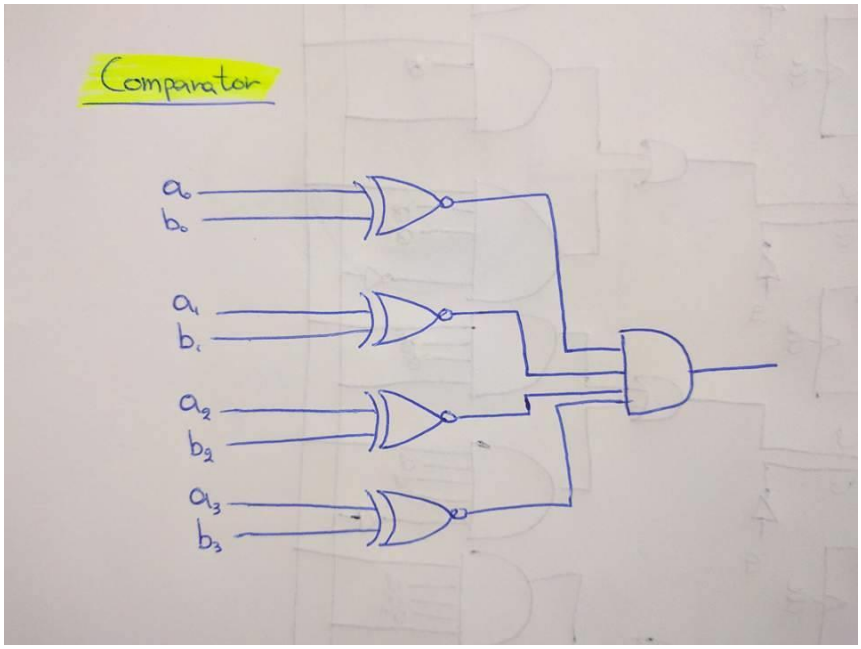
Η συνολική υλοποίηση και τη σύνδεδη των προηγουμένων υποκυκλωμάτων έγινε στον κώδικα του Top Level.

Παρακάτω παρατίθενται τα διαγράμματα των κυκλωμάτων που υλοποιήθηκαν:

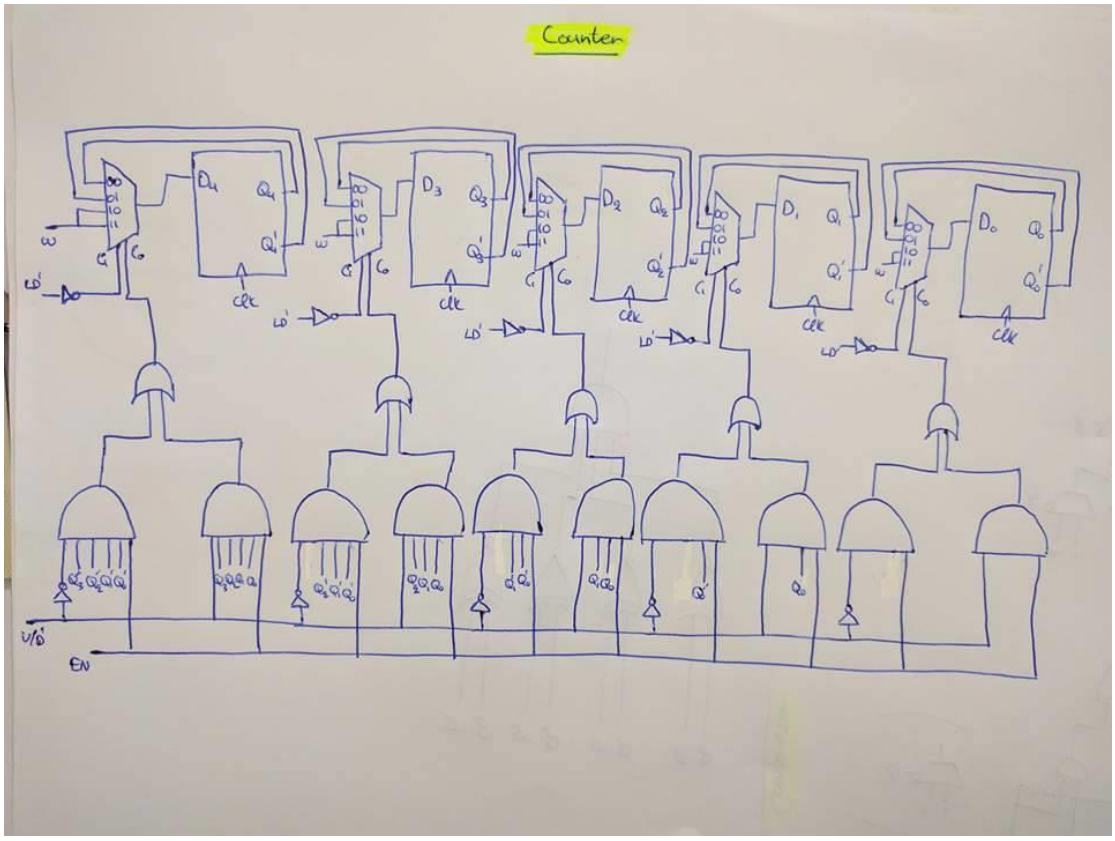
MUX 2:1 και **MUX 4:1**



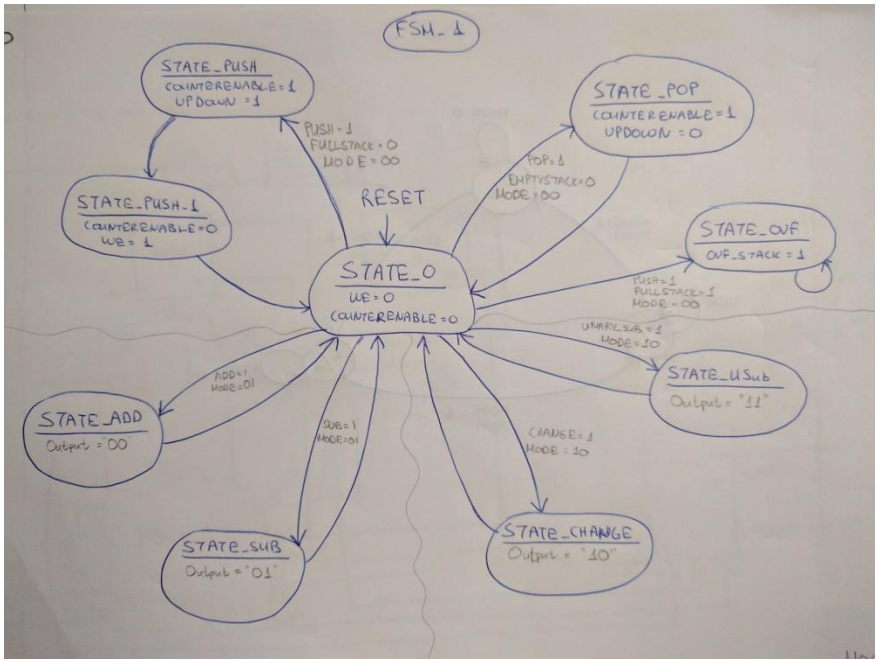
Comparator



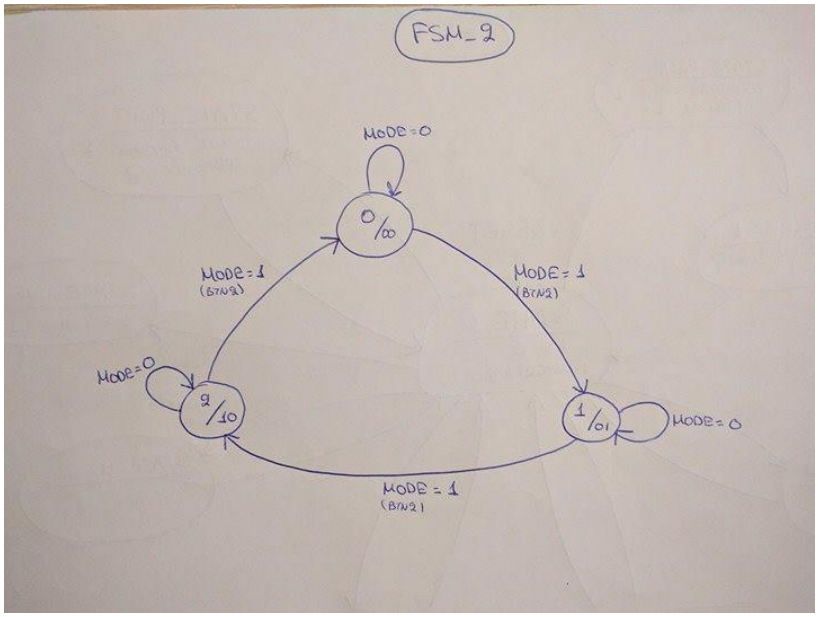
Counter (structural)



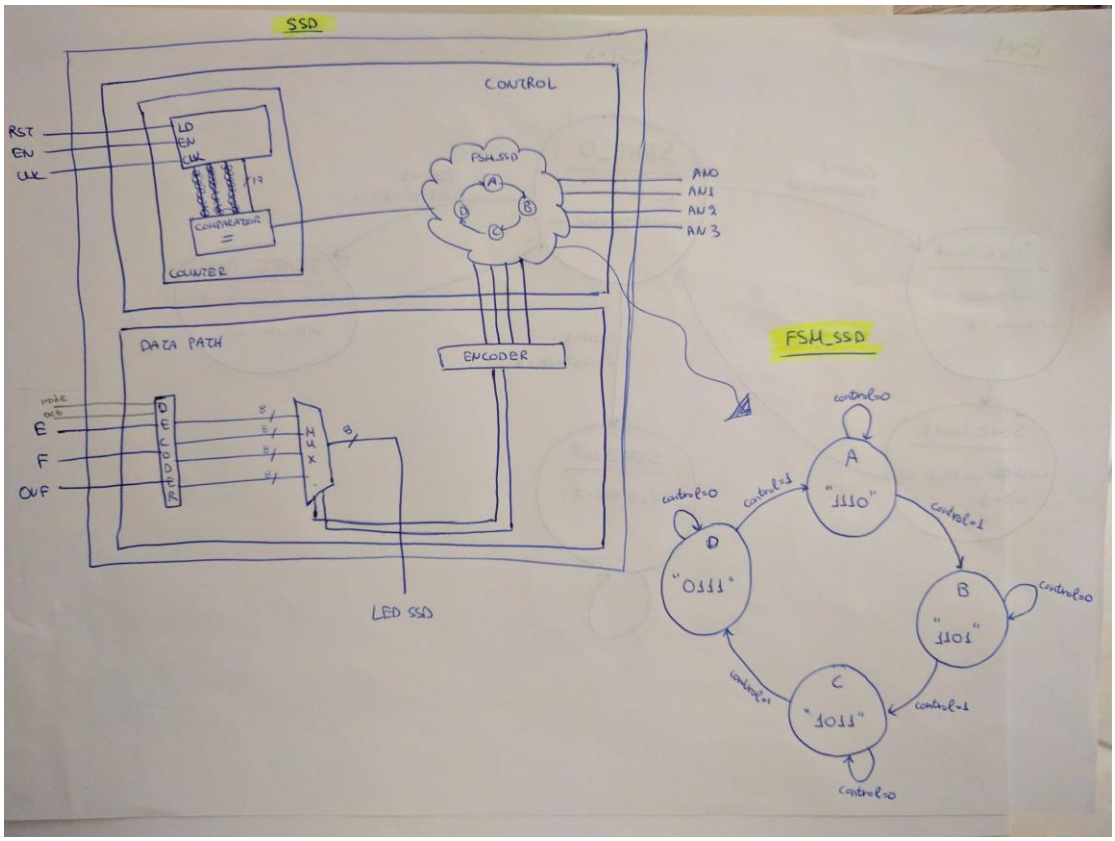
Fsm1



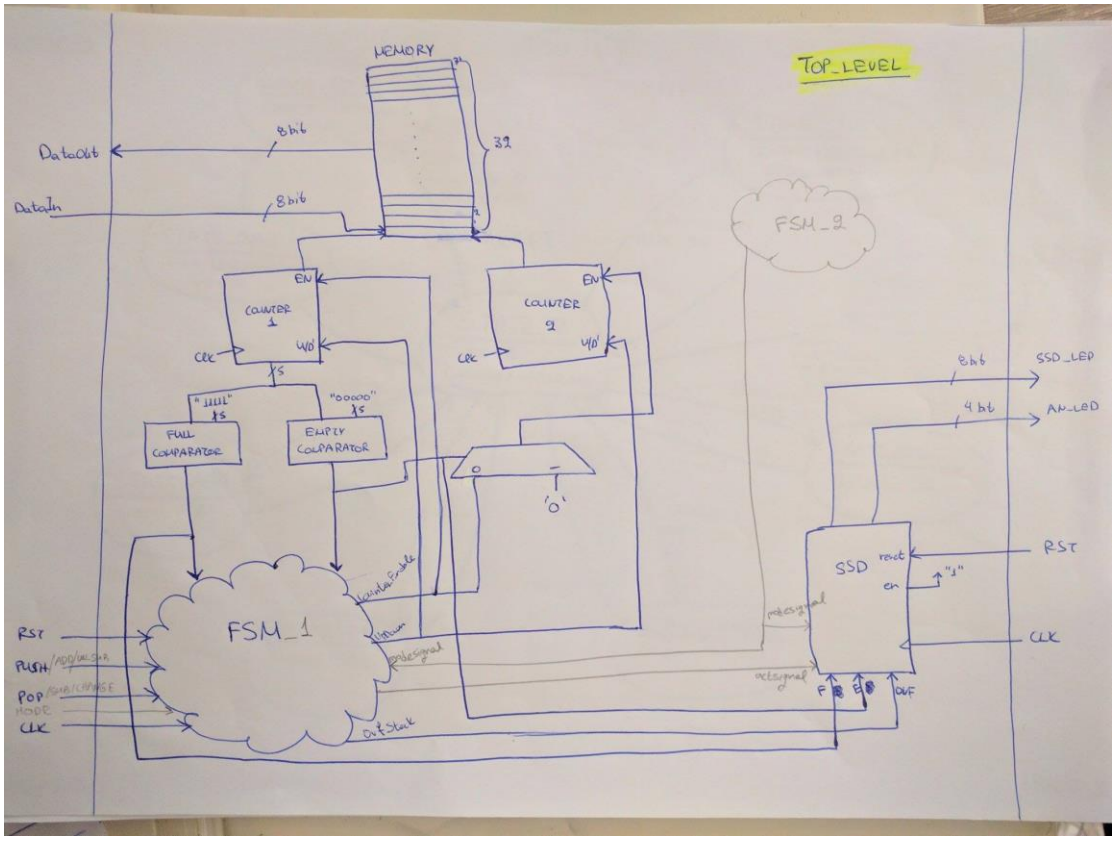
Fsm2



SSD και **FSM_SSD**



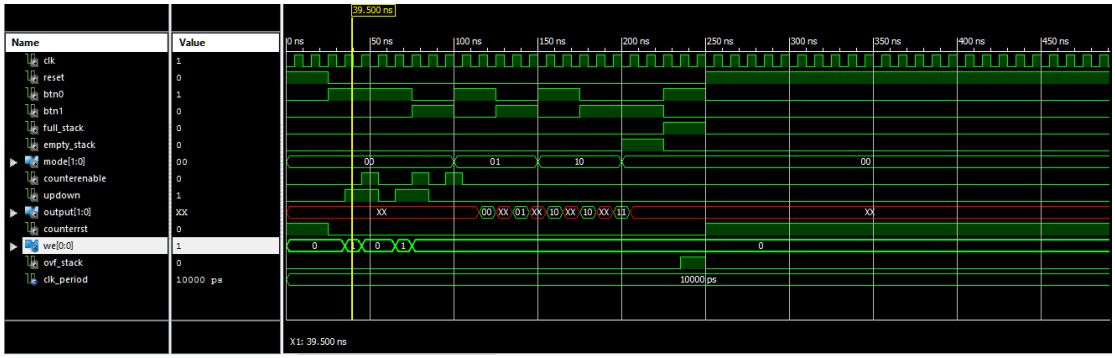
Top Level



Ο κώδικας για MUX 2:1 και MUX 4:1, comparator και counter χρησιμοποιήθηκαν πανομοιοτύπως από το προηγούμενο εργαστήριο.

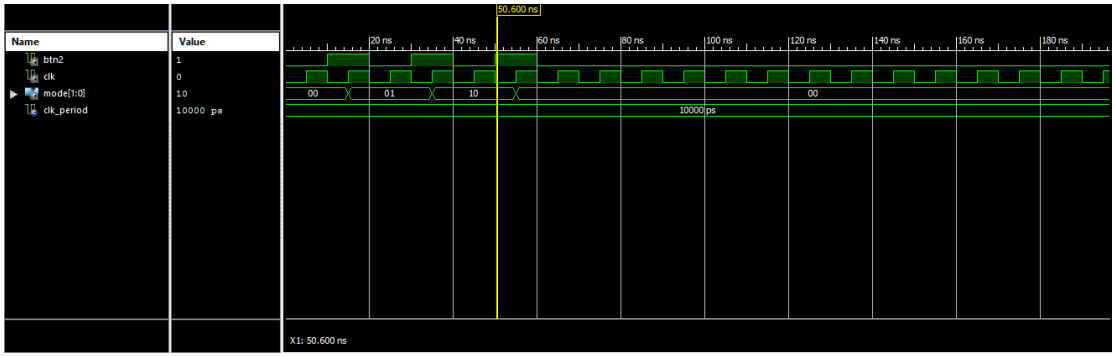
Κυματομορφές-Προσομοίωση

Fms1

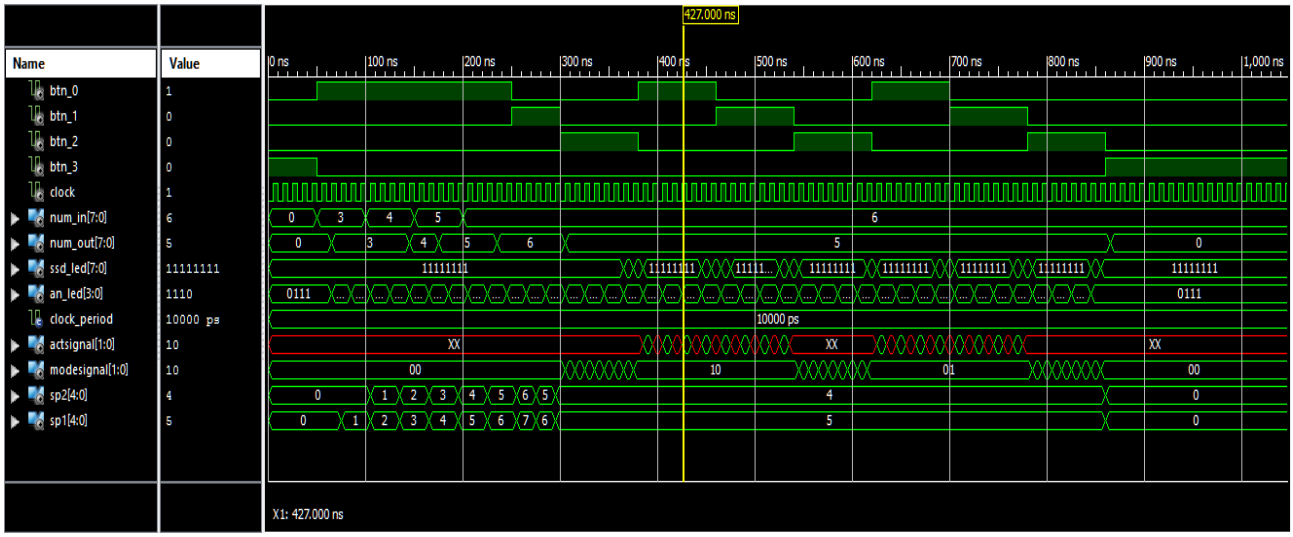


Στην κυματομορφή υπάρχουν κάποια κόκκινα XX. Δεν είναι κάτι λάθος αλλά εμείς μέσω του κώδικα μας επιλέξαμε να δείχνει έτσι. Είναι το σήμα όπου δείχνει τι πράξη κάνουμε ανάλογα με το mode που βρισκόμαστε.

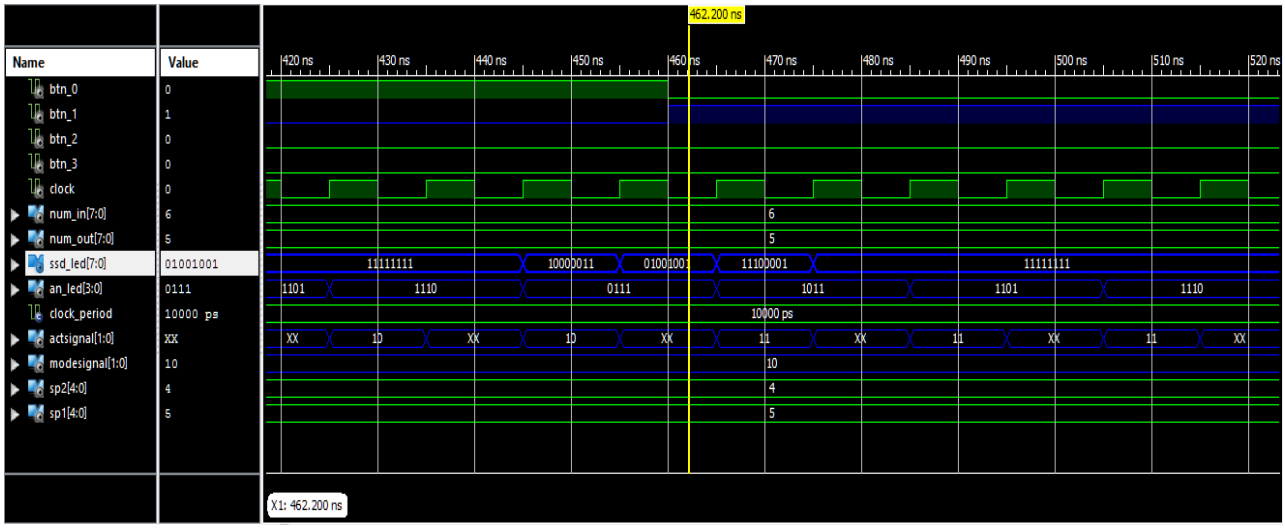
Fsm2



Top level



Τα XX κόκκινα σε αυτήν την κυματομορφή είναι τα ίδια με αυτά που προανέφερα στην FSM 1. Αυτή είναι μια εικόνα που δείχνει στο σύνολο το TOP LEVEL μας.



Σε αυτήν την εικόνα αλλάξαμε το χρώμα σε μπλε τα σήματα που θέλουμε να τονίσουμε. Είναι τα σήματα των LED και των AN του SSD. Δηλαδή όταν βρισκόμαστε σε ένα mode και πατήσουμε ένα BTN τι θα εμφανίζεται στα SSD-LED και ποιο SSD-AN θα είναι ενεργοποιημένο.

Κώδικας

Παραθέτουμε τα σημαντικότερα σημεία του κώδικά μας.

FSM_1

```
33
34  entity FSM_1 is
35      Port ( CLK : in  STD_LOGIC;
36             Reset : in  STD_LOGIC;
37             BTN0 : in  STD_LOGIC;
38             BTN1 : in  STD_LOGIC;
39             Full_Stack : in  STD_LOGIC;
40             Empty_Stack : in  STD_LOGIC;
41             Mode : in STD_LOGIC_VECTOR(1 downto 0);
42             CounterEnable : out  STD_LOGIC;
43             UpDown : out  STD_LOGIC;
44             Output : out  STD_LOGIC_VECTOR(1 downto 0); --praksi
45             CounterRst : out  STD_LOGIC;
46             We : out  STD_LOGIC_VECTOR(0 downto 0);
47             Ovf_Stack : out  STD_LOGIC);
48  end FSM_1;
49
50  architecture Behavioral of FSM_1 is
51
52  type state_type is (state_0,state_Push,state_Push1,state_Pop,state_Ovf,state_Add,state_Sub,state_USub,state_Change);
53  signal state_next, state: state_type;
54
55  begin
56
57  fsm_comb: process(state, state_next, BTN0, BTN1, Empty_Stack, Full_Stack, Mode)
58  begin
59     case state is
60
61        when state_0=> We<="0";
62                       CounterEnable<='0';
63                       UpDown<='0';
64                       Ovf_Stack<='0';
65                       Output<="XX";
66                       If(BTN0='1' AND Full_Stack='0' AND Mode="00") then state_next<=state_Push;
67                       elsif(BTN1 ='1' AND Empty_Stack='0' AND Mode="00") then state_next<=state_Pop;
68                       elsif(BTN0='1' AND Full_Stack='1' AND Mode="00") then state_next<=state_Ovf;
69                       elsif(BTN0='1' AND Mode="01") then state_next<=state_Add;
70                       elsif(BTN1='1' AND Mode="01") then state_next<=state_Sub;
71                       elsif(BTN0='1' AND Mode="10") then state_next<=state_USub;
72                       elsif(BTN1='1' AND Mode="10") then state_next<=state_Change;
73                       else state_next<=state_0;
74                       end if;
75
76
```

```
77         when state_Push1=> state_next<=state_0;
78                            CounterEnable<='1';
79                            UpDown<='1';
80                            Ovf_Stack<='0';
81                            We<="0";
82                            Output<="XX";
83
84         when state_Push=> state_next<=state_Push1;
85                           CounterEnable<='0';
86                           UpDown<='1';
87                           We<="1";
88                           Ovf_Stack<='0';
89                           Output<="XX";
90
91         when state_Pop=> state_next<=state_0;
92                          CounterEnable<='1';
93                          UpDown<='0';
94                          We<="0";
95                          Ovf_Stack<='0';
96                          Output<="XX";
97
98         when state_Ovf=> Ovf_Stack<='1';
99                          We<="0";
100                         CounterEnable<='0';
101                         UpDown<='0';
102                         state_next<=state_Ovf;
103                         Output<="XX";
104
105        when state_Add=> We<="0";
106                         CounterEnable<='0';
107                         UpDown<='0';
108                         Ovf_Stack<='0';
109                         Output <= "00";
110                         state_next<=state_0;
111
112        when state_Sub=> We<="0";
113                         CounterEnable<='0';
114                         UpDown<='0';
115                         Ovf_Stack<='0';
116                         Output <= "01";
117                         state_next<=state_0;
118
```

```
119         when state_Usub=> We<="0";
120                           CounterEnable<='0';
121                           UpDown<='0';
122                           Ovf_Stack<='0';
123                           Output <= "10";
124                           state_next<=state_0;
125
126         when state_Change=> We<="0";
127                             CounterEnable<='0';
128                             UpDown<='0';
129                             Ovf_Stack<='0';
130                             Output <= "11";
131                             state_next<=state_0;
132
133         when others=> state_next<=state_0;
134                       Ovf_Stack<='0';
135                       We<="0";
136                       CounterEnable<='0';
137                       UpDown<='0';
138                       Output<="XX";
139
140 end case;
141 end process fsm_comb;
142
143 fsm_synch: process(Reset, CLK)
144 begin
145    if (Reset='1') then state <= state_0;
146                        CounterRst<='1';
147    elsif (rising_edge(CLK)) then state <= state_next;
148                                  CounterRst<='0';
149    end if;
150 end process fsm_synch;
151
152
153 end Behavioral;
```

FSM_2

```
32  entity FSM_2 is
33      Port ( BTN2 : in  STD_LOGIC;
34             CLK : in  STD_LOGIC;
35             MODE : out  STD_LOGIC_VECTOR (1 downto 0));
36  end FSM_2;
37
38  architecture Behavioral of FSM_2 is
39
40  type state_type is (state_0,state_1,state_2); --mode0->2
41  signal state_next, state: state_type;
42
43  begin
44
45  fsm_change : process(state,state_next)
46  begin
47        case state is
48
49        when state_0 => MODE<="00";
50                        state_next<=state_1;
51
52        when state_1 => MODE<="01";
53                        state_next<=state_2;
54
55        when state_2 => MODE<="10";
56                       state_next<=state_0;
57
58        when others => MODE<="00";
59                       state_next<=state_0;
60  end case;
61  end process fsm_change;
62
63  fsm_true : process(BTN2,state_next,CLK)
64  begin
65     if(rising_edge(Clk)) then
66        if (BTN2='1') then state <= state_next;
67        else state <= state;
68        end if;
69     else state <= state;
70     end if;
71  end process fsm_true;
72
73
74
75  end Behavioral;
```