3η Εργαστηριακή Άσκηση

ΔΗΜΙΟΥΡΓΙΑ ΜΙΑΣ POST-INCREMENT ,PRE-DECREMENT ΣΤΟΙΒΑΣ ΣΕ VHDL

Ομάδα LAB20332061

ΛΑΜΠΡΙΑΝΗ ΤΣΑΠΑΝΟΥ 2014030015
ΑΝΑΣΤΑΣΙΟΣ ΜΠΟΚΑΛΙΔΗΣ 2014030069

Σκοπός εργαστηριακής άσκησης

Σκοπός του 3ου εργαστηρίου ήταν περαιτέρω εξοικείωση με την γλώσσα VHDL καθώς και η εξοικείωση σχετικά με τις διαφορές μεταξύ της behavioral και structural μορφής. Σε αυτό το εργαστήριο σχεδιάσαμε ένα ολοκληρωμένο κύκλωμα το οποίο υλοποιεί την βασική λειτουργικότητα μιας **post-increment, pre-decrement** στοίβας. Το κύκλωμα μας εκτελεί τις απλές λειτουργίες **push** και **pop** της στοίβας.Στα LED του board θα απεικονίζεται σε δυαδική μορφή το στοιχείο του TOS(Top Of Stack)ενώ στα 7-segment display θα απεικονίζεται η πληροφορία Empty, Full ή Overflow (E, F ή OVF) ανάλογα με την κατάσταση της στοίβας μας εκείνη την χρονική στιγμή.

Προεργασία – Περιγραφή

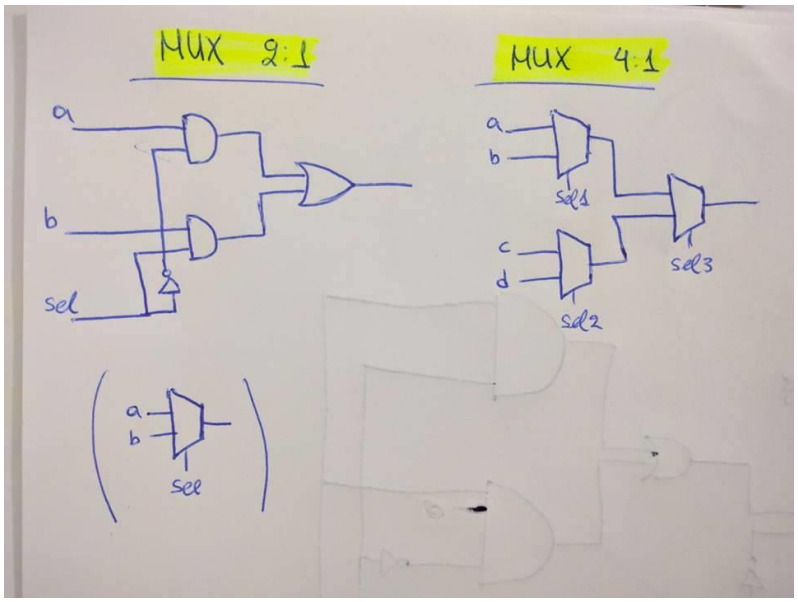
Το κύκλωμά μας αποτελείται από τα παρακάτω υποκυκλώματα:

- 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 οποίοι 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 **FSM** οπού διαβάζει το output του reset, push και pop καθώς και την κατάσταση που βρίσκεται εκείνη την στιγμή η στοίβα μας (Empty, Full) από τις εξόδους των comparator και ελέγχει 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

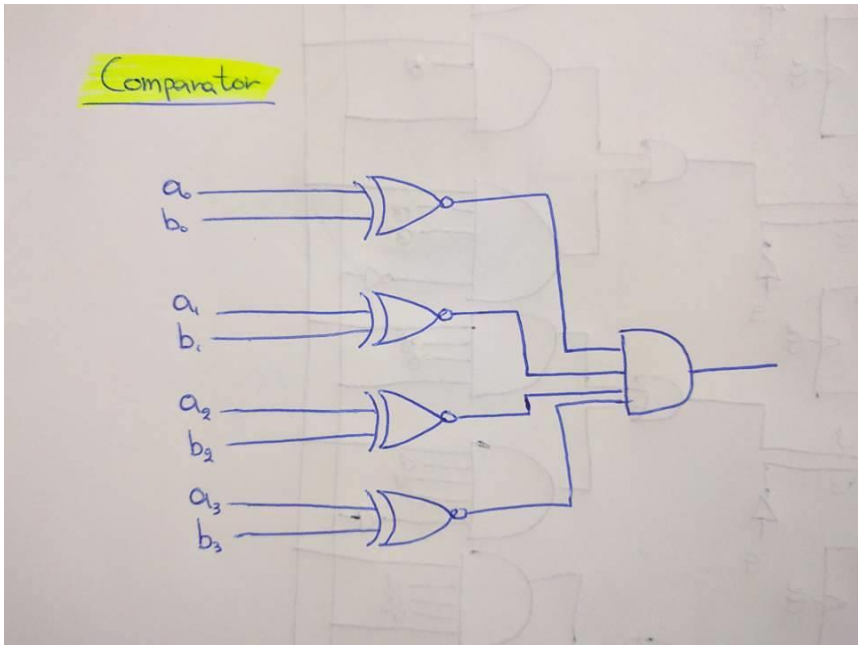
Η συνολική υλοποίηση και τη σύνδεδη των προηγουμένων υποκυκλωμάτων έγινε στον κώδικα του Top Level.

Παρακάτω παρατίθενται τα διαγράμματα των κυκλωμάτων που υλοποιήθηκαν:

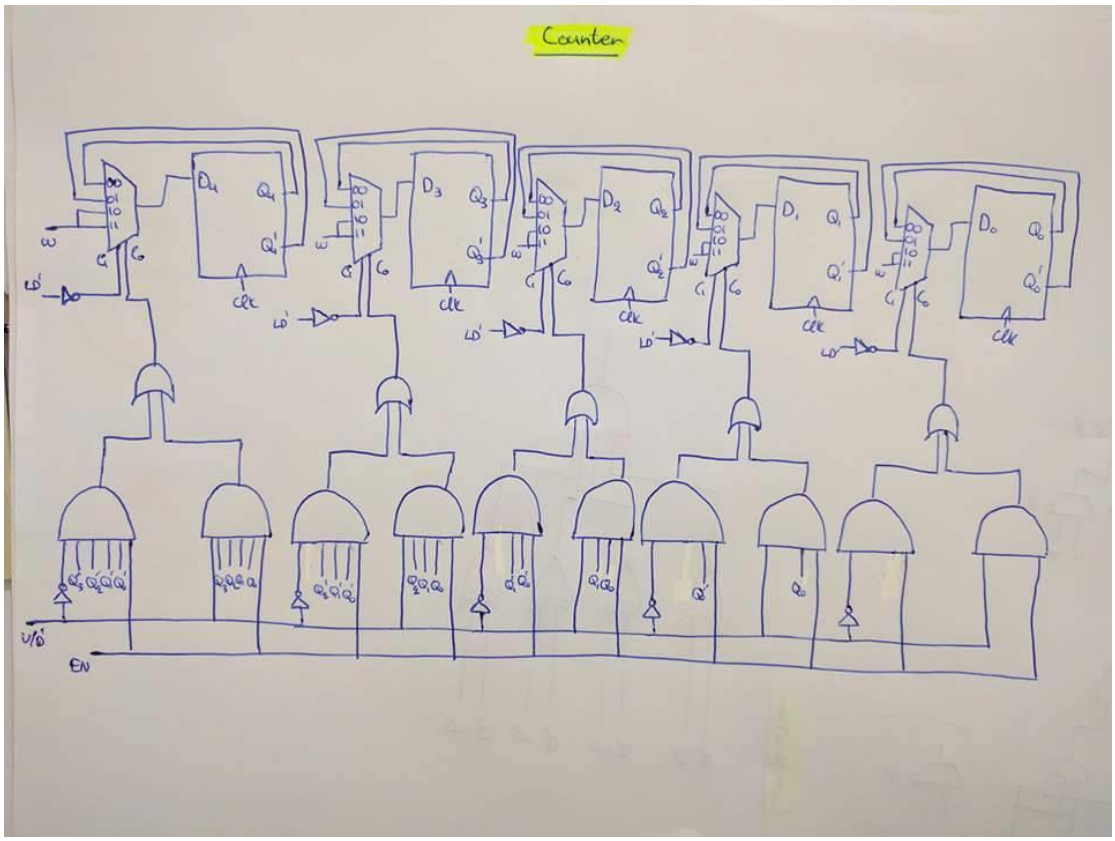
MUX 2:1 και **MUX 4:1**



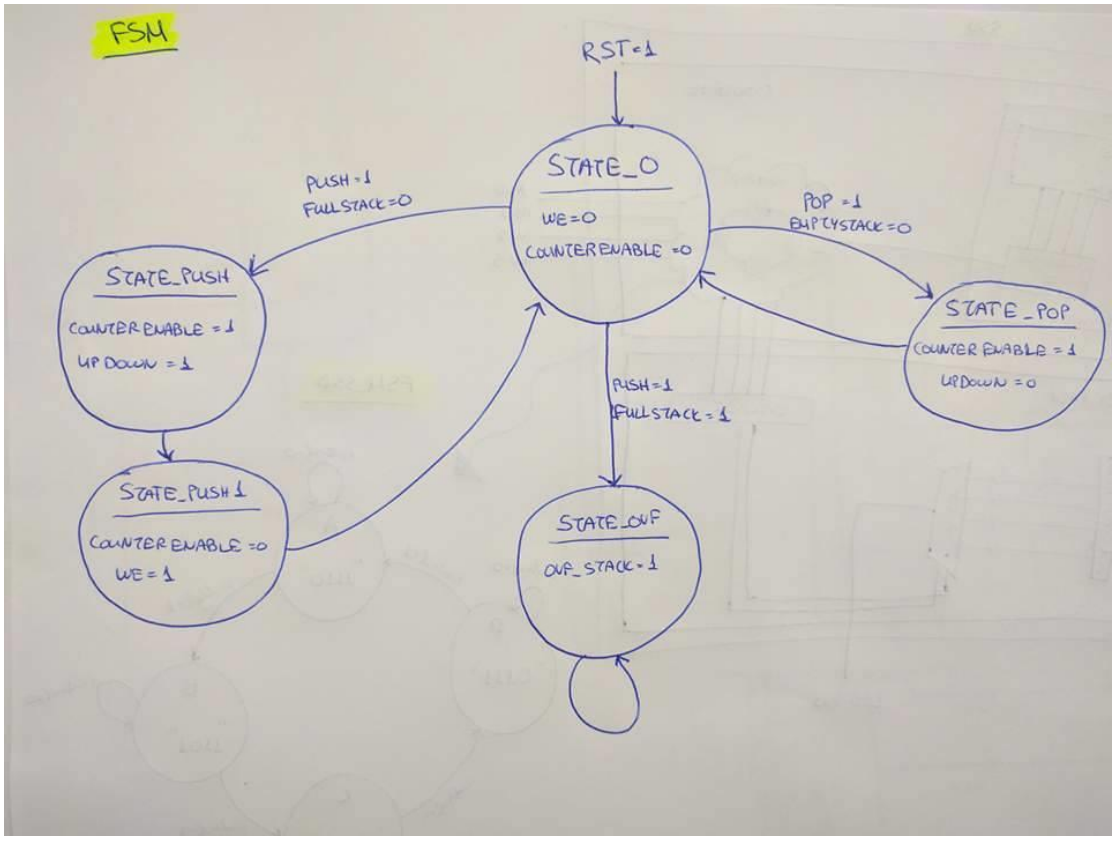
Comparator



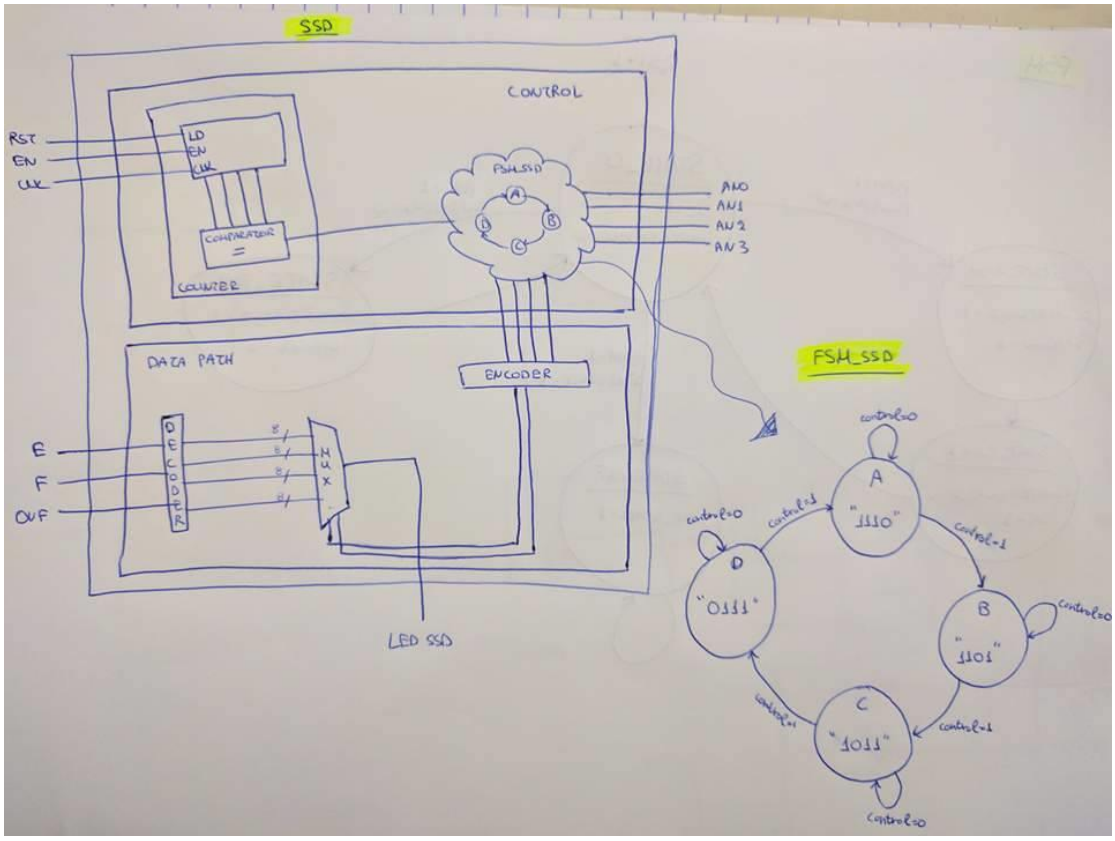
Counter (structural)



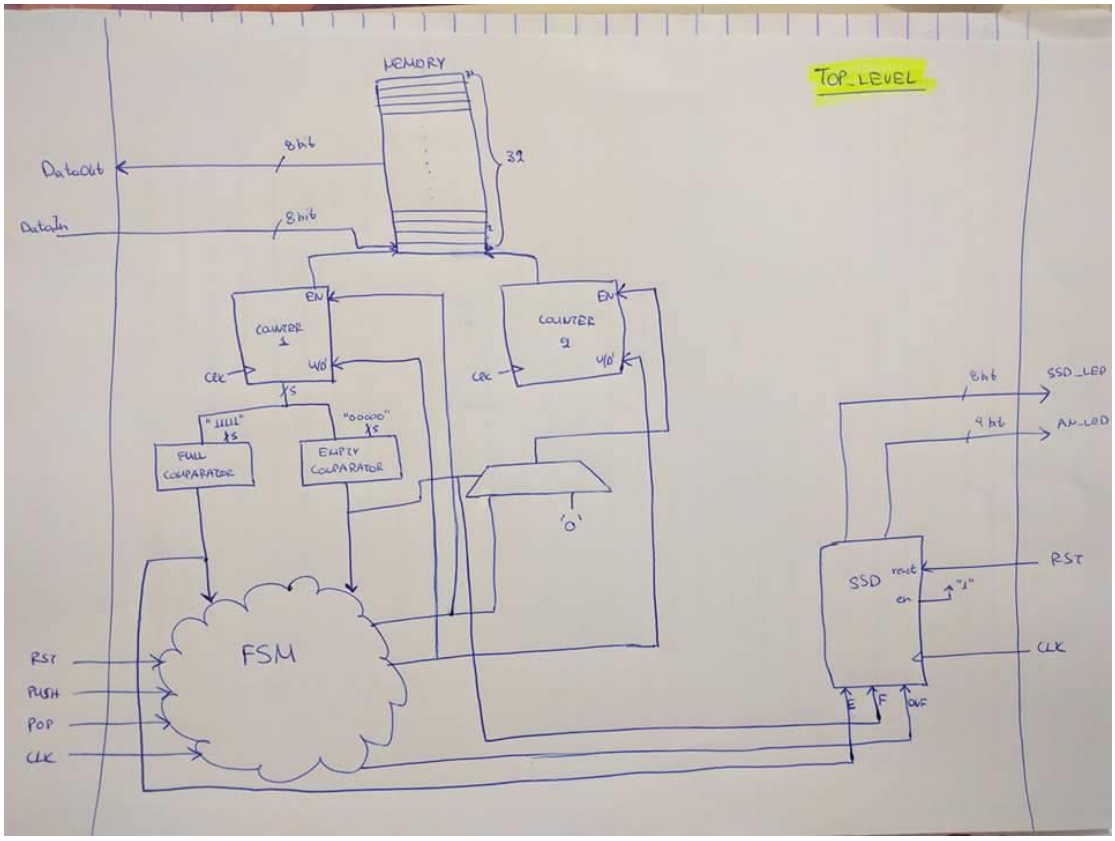
Fsm



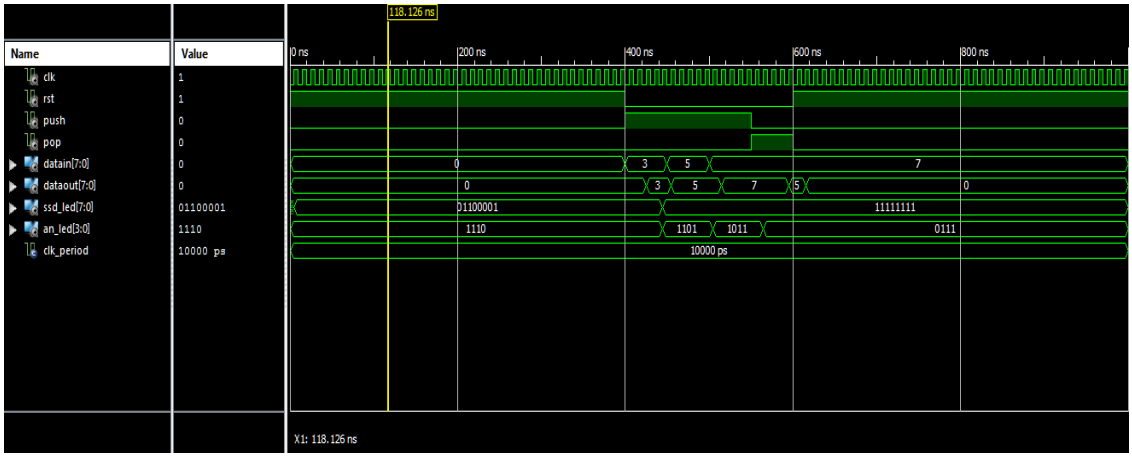
SSD και **FSM_SSD**



Top Level



Κυματομορφές-Προσομοίωση



Στην παραπάνω κυματομορφή του **Top Level** παρατηρούμε την συμπεριφορά του κυκλώματός μας στις λειτουργείες reset, push και pop. Το datain είναι τα στοιχεία που κάνουμε push μέσω των switches. Το dataout είναι το τελευταίο στοιχείο που βάζουμε στην στοίβα μας. Το ssd_led είναι το ποια led είναι αναμένα στα ssd ενώ το an_led δείχνει ποιο AN(0->3) των ssd είναι αναμένο.

Κώδικας

Παραθέτουμε τα σημαντικότερα σημεία του κώδικά μας.

Comparator

```
32  entity Comparator is
33      Port ( A : in  STD_LOGIC_VECTOR (4 downto 0);
34             B : in  STD_LOGIC_VECTOR (4 downto 0);
35             C : out  STD_LOGIC);
36  end Comparator;
37
38  architecture Structural of Comparator is
39
40  Component AND_gate is
41  Port ( A : in  STD_LOGIC;
42         B : in  STD_LOGIC;
43         C : in  STD_LOGIC;
44         D : in  STD_LOGIC;
45         E : in  STD_LOGIC;
46         F : out STD_LOGIC
47        );
48  end Component;
49
50  Component XNOR_gate is
51  Port( A : in std_logic;
52        B : in std_logic;
53        C : out std_logic
54       );
55  end Component;
56
57  signal s: std_logic_vector(4 downto 0);
58
59  begin
60
61  XNOR_1: XNOR_gate
62         port map( A=>A(0),
63                   B=>B(0),
64                   C=>s(0)
65         );
66  XNOR_2: XNOR_gate
67         port map( A=>A(1),
68                   B=>B(1),
69                   C=>s(1)
70         );
71
```

```
72  XNOR_3: XNOR_gate
73         port map( A=>A(2),
74                   B=>B(2),
75                   C=>s(2)
76         );
77
78  XNOR_4: XNOR_gate
79         port map( A=>A(3),
80                   B=>B(3),
81                   C=>s(3)
82         );
83
84  XNOR_5: XNOR_gate
85         port map( A=>A(4),
86                   B=>B(4),
87                   C=>s(4)
88         );
89
90  AND_EQUAL: AND_gate
91             port map( A=>s(0),
92                       B=>s(1),
93                       C=>s(2),
94                       D=>s(3),
95                       E=>s(4),
96                       F=>C
97              );
98
99  end Structural;
100
101
```

FSM_1

```
34  entity FSM_1 is
35      Port ( CLK : in  STD_LOGIC;
36             Reset : in  STD_LOGIC;
37             Push : in  STD_LOGIC;
38             Pop : in  STD_LOGIC;
39             Full_Stack : in  STD_LOGIC;
40             Empty_Stack : in  STD_LOGIC;
41             CounterEnable : out  STD_LOGIC;
42             UpDown : out  STD_LOGIC;
43             CounterRst : out  STD_LOGIC;
44             We : out  STD_LOGIC_VECTOR(0 downto 0);
45             Ovf_Stack : out  STD_LOGIC);
46  end FSM_1;
47
48  architecture Behavioral of FSM_1 is
49
50  signal x : std_logic_vector(4 downto 0);
51
52  type state_type is (state_0, state_Push, state_Push1, state_Pop, state_Ovf);
53  signal state_next, state: state_type;
54
55  begin
56
57  fsm_comb: process(state, state_next, Push, Pop, Empty_Stack, Full_Stack)
58  begin
59     case state is
60
61        when state_0=> We<="0";
62                       CounterEnable<='0';
63                       If(Push='1' AND Full_Stack='0') then state_next<=state_Push;
64                       elsif(Pop ='1') then if(Empty_Stack='0') then state_next<=state_Pop;
65                                            else state_next<=state_0;
66                                            end if;
67                       else state_next<=state_Ovf;
68                       end if;
69
70        when state_Push=> state_next<=state_Push1;
71                          CounterEnable<='1';
72                          UpDown<='1';
73
74        when state_Push1=> state_next<=state_0;
75                           CounterEnable<='0';
76                           We<="1";
77
```

```
77
78         when state_Pop=> state_next<=state_0;
79                          CounterEnable<='1';
80                          UpDown<='0';
81
82         when state_Ovf=> Ovf_Stack<='1';
83                          state_next<=state_Ovf;
84
85         when others=> state_next<=state_0;
86
87  end case;
88  end process fsm_comb;
89
90  fsm_synch: process(Reset, CLK)
91  begin
92     if (Reset='1') then state <= state_0;
93                         CounterRst<='1';
94     elsif (rising_edge(CLK)) then state <= state_next;
95                                   CounterRst<='0';
96     end if;
97  end process fsm_synch;
98
99
100 end Behavioral;
101
102
```

Counter (structural)

```
32  entity Counter_5bit is
33      Port ( CLK : in  STD_LOGIC;
34             RST : in  STD_LOGIC;
35             EN : in  STD_LOGIC;
36             UpDown : in  STD_LOGIC;
37             Output : out  STD_LOGIC_VECTOR (4 downto 0));
38  end Counter_5bit;
39
40  architecture Structural of Counter_5bit is
41
42  signal s: std_logic_vector(4 downto 0);
43  signal a: std_logic_vector(4 downto 0);
44  signal b: std_logic_vector(4 downto 0);
45  signal W: std_logic_vector(4 downto 0);
46  signal v: std_logic;
47
48  Component MUX_4x2 is
49  Port ( IN0 : in  STD_LOGIC;
50         IN1 : in  STD_LOGIC;
51         IN2 : in  STD_LOGIC;
52         IN3 : in  STD_LOGIC;
53         C : in  STD_LOGIC_VECTOR(1 downto 0);
54         Output : out  STD_LOGIC
55       );
56  end Component;
57
58  Component D_FF is
59  Port ( CLK : in  STD_LOGIC;
60         RST : in  STD_LOGIC;
61         D : in  STD_LOGIC;
62         Q : out  STD_LOGIC
63       );
64  end Component;
65
66  begin
67
68  process
69
70  begin
71
72  if RST='1' then
73     W<="00000";
74  end if;
75
76  wait until(CLK' EVENT AND CLK = '1');
```

```
76  wait until(CLK' EVENT AND CLK = '1');
77
78  end process;
79
80  v<='0';
81  Output<=b;
82
83  s(0)<= ((NOT UpDown) AND EN) OR (EN AND UpDown);
84  s(1)<= ((NOT UpDown) AND EN AND (NOT b(0)) ) OR (UpDown AND EN AND b(0) );
85  s(2)<= ((NOT UpDown) AND EN AND (NOT b(1)) AND (NOT b(0)) ) OR (UpDown AND EN AND b(0) AND b(1));
86  s(3)<= ((NOT UpDown) AND EN AND (NOT b(0)) AND (NOT b(1)) AND (NOT b(2)) ) OR (UpDown AND EN AND b(0) AND b(1) AND b(2));
87  s(4)<= ((NOT UpDown) AND EN AND (NOT b(0)) AND (NOT b(1)) AND (NOT b(2)) AND (NOT b(3)) ) OR (UpDown AND EN AND b(0) AND b(1) AND b(2) AND b(3));
88
89  D0:   D_FF
90        port map ( CLK=>CLK,
91                   RST=>RST,
92                   D=>a(0),
93                   Q=>b(0)
94                 );
95
96  D1:   D_FF
97
98        port map ( CLK=>CLK,
99                   RST=>RST,
100                  D=>a(1),
101                  Q=>b(1)
102                );
103
104 D2:   D_FF
105       port map ( CLK=>CLK,
106                  RST=>RST,
107                  D=>a(2),
108                  Q=>b(2)
109                );
110
111 D3:   D_FF
112       port map ( CLK=>CLK,
113                  RST=>RST,
114                  D=>a(3),
115                  Q=>b(3)
116                );
117
```

```
118  D4:   D_FF
119        port map ( CLK=>CLK,
120                   RST=>RST,
121                   D=>a(4),
122                   Q=>b(4)
123                 );
124
125  MUX0: MUX_4x2
126        port map ( IN0=>b(0),
127                   IN1=>(NOT b(0)),
128                   IN2=>W(0),
129                   IN3=>W(0),
130                   C(1)=>v,
131                   C(0)=>s(0),
132                   Output=>a(0)
133                  );
134
135  MUX1: MUX_4x2
136        port map ( IN0=>b(1),
137                   IN1=>(NOT b(1)),
138                   IN2=>W(1),
139                   IN3=>W(1),
140                   C(1)=>v,
141                   C(0)=>s(1),
142                   Output=>a(1)
143                  );
144
145  MUX2: MUX_4x2
146        port map ( IN0=>b(2),
147                   IN1=>(NOT b(2)),
148                   IN2=>W(2),
149                   IN3=>W(2),
150                   C(1)=>v,
151                   C(0)=>s(2),
152                   Output=>a(2)
153                  );
154
```

```
154
155   MUX3: MUX_4x2
156         port map ( IN0=>b(3),
157                    IN1=>(NOT b(3)),
158                    IN2=>W(3),
159                    IN3=>W(3),
160                    C(1)=>v,
161                    C(0)=>s(3),
162                    Output=>a(3)
163                   );
164
165   MUX4: MUX_4x2
166         port map ( IN0=>b(4),
167                    IN1=>(NOT b(4)),
168                    IN2=>W(4),
169                    IN3=>W(4),
170                    C(1)=>v,
171                    C(0)=>s(4),
172                    Output=>a(4)
173                   );
174
175  end Structural;
```