5η Εργαστηριακή Άσκηση

ΓΙΑ ΣΧΕΔΙΑΣΗ ΜΙΑΣ ΑΠΛΗΣ ΑΡΙΘΜΟΜΗΧΑΝΗΣ

Ομάδα LAB20332061

ΛΑΜΠΡΙΑΝΗ ΤΣΑΠΑΝΟΥ 2014030015
ΑΝΑΣΤΑΣΙΟΣ ΜΠΟΚΑΛΙΔΗΣ 2014030069

Σκοπός εργαστηριακής άσκησης

Σκοπός του 5ου εργαστηρίου είναι να εξελίξουμε τα 2 προηγούμενα εργαστήρια. Για την ακρίβεια σε αυτό το εργαστήριο πρέπει να υλοποιήσουμε τις λογικές πράξεις :

- Πρόσθεση αριθμών 2's complement (TOS + "TOS-1")
- Αφαίρεση αριθμών 2's complement (TOS – "TOS-1")
- Μοναδιαία αφαίρεση 2's complement (unary subtraction, το TOS γίνεται -TOS)
- Εναλλαγή TOS με TOS-1 (γνωστή και σαν X<>Y)

Φυσικά για να λειτουργήσουν οι παραπάνω πράξεις, πρέπει να λειτουργούν σωστά οι προηγούμενες λειτουργίες που υλοποιήσαμε στα εργαστήρια 3,4.

Προεργασία – Περιγραφή

Το κύκλωμα μας σε αυτό το εργαστήριο 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
- Τους 2 **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** όπου σε αυτήν περιέχουμε όλες τις καταστάσεις για όλες τις λειτουργίες που χρειάζεται σε αυτό το εργαστήριο. Στην ουσία είναι η FSM1 που χρησιμοποιήσαμε στα προηγούμενα εργαστήρια, απλώς προσθέσαμε κάποιες έξτρα καταστάσεις. Για παράδειγμα προσθέσαμε καταστάσεις όπου ελέγχουν πότε πρέπει να ενεργοποιούνται οι καταχωρητές που αποθηκεύουν τους αριθμούς που

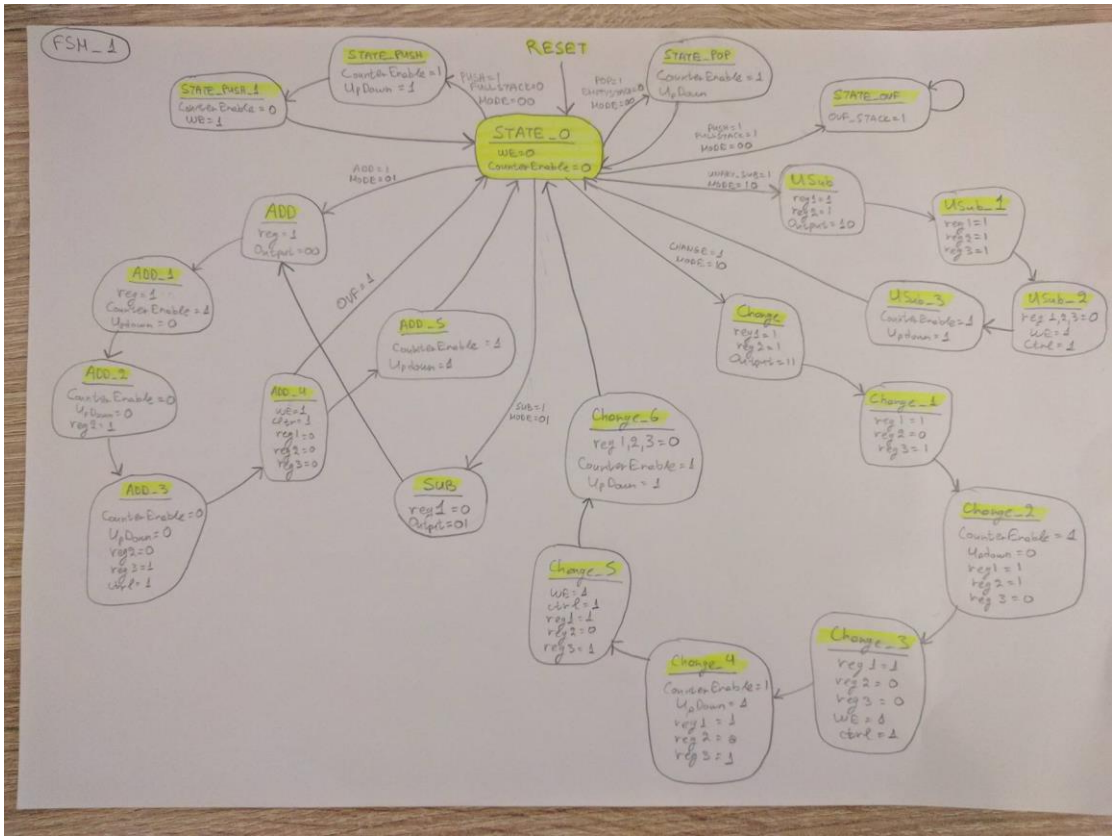
χρειάζονται για τις λογικές πράξεις, καθώς επίσης και το πότε θα πρέπει να κάνει pop και push η στοίβα μας πριν ή μετά την επιτυχή ή μη επιτυχή πράξη μας.

- Τους 3 **Registers** μας όπου σε αυτούς αποθηκεύουμε τα στοιχεία μας από το TOS και το TOS-1 της στοίβας μας και το αποτέλεσμα του Αθροιστή/Αφαιρετή. Επίσης τον ένα register τον χρησιμοποιούμε διαφορετικά στις λειτουργίες UnarySub και TOS<>TOS-1 καθώς πρέπει να τον αρχικοποιούμε στο μηδέν.
- Τον **Αθροιστή/Αφαιρετή** μας ο οποίος κάνει τις πράξεις που χρειάζονται σε αυτό το εργαστήριο. Σε αυτόν προσθέσαμε και την λειτουργία που χρειάζεται στις αφαιρέσεις, δηλαδή την λειτουργία όπου μετατρέπει τον δεύτερο αριθμό στον αντίστοιχο αρνητικό του. Στην ουσία αυτή η λειτουργία είναι αυτή η οποία καθορίζει για τον αν είναι το κύκλωμα αυτό Αθροιστής ή Αφαιρετής.
- Την λειτουργία όπου ελέγχει αν μετά τις πράξεις υπάρχει overflow στο αποτέλεσμα σύμφωνα με την περιγραφή που μας δώσατε στο αντίστοιχο φροντιστήριο.
- 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

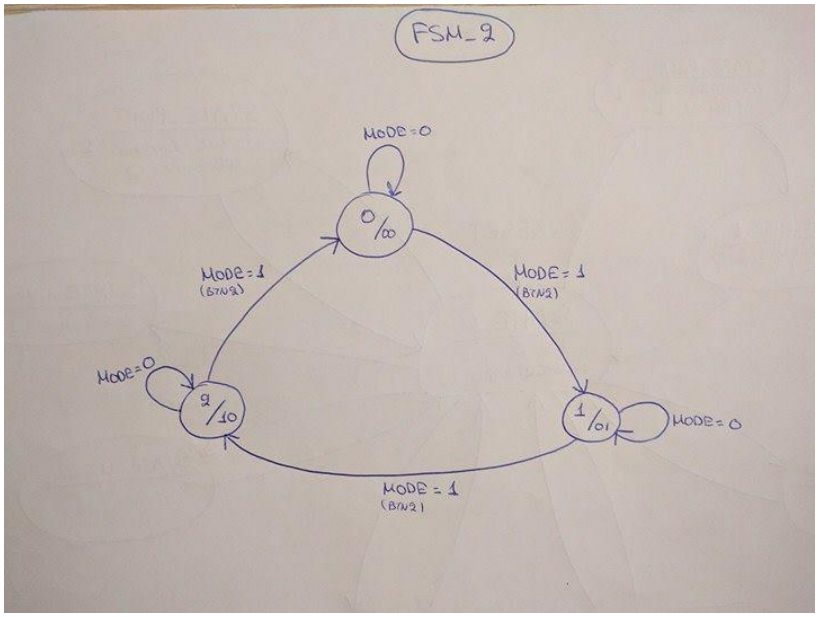
Η συνολική σύνδεση των προηγουμένων υποκυκλωμάτων έγινε στον κώδικα του Top Level.

Παρακάτω παρατίθενται τα διαγράμματα των κυκλωμάτων που υλοποιήθηκαν:

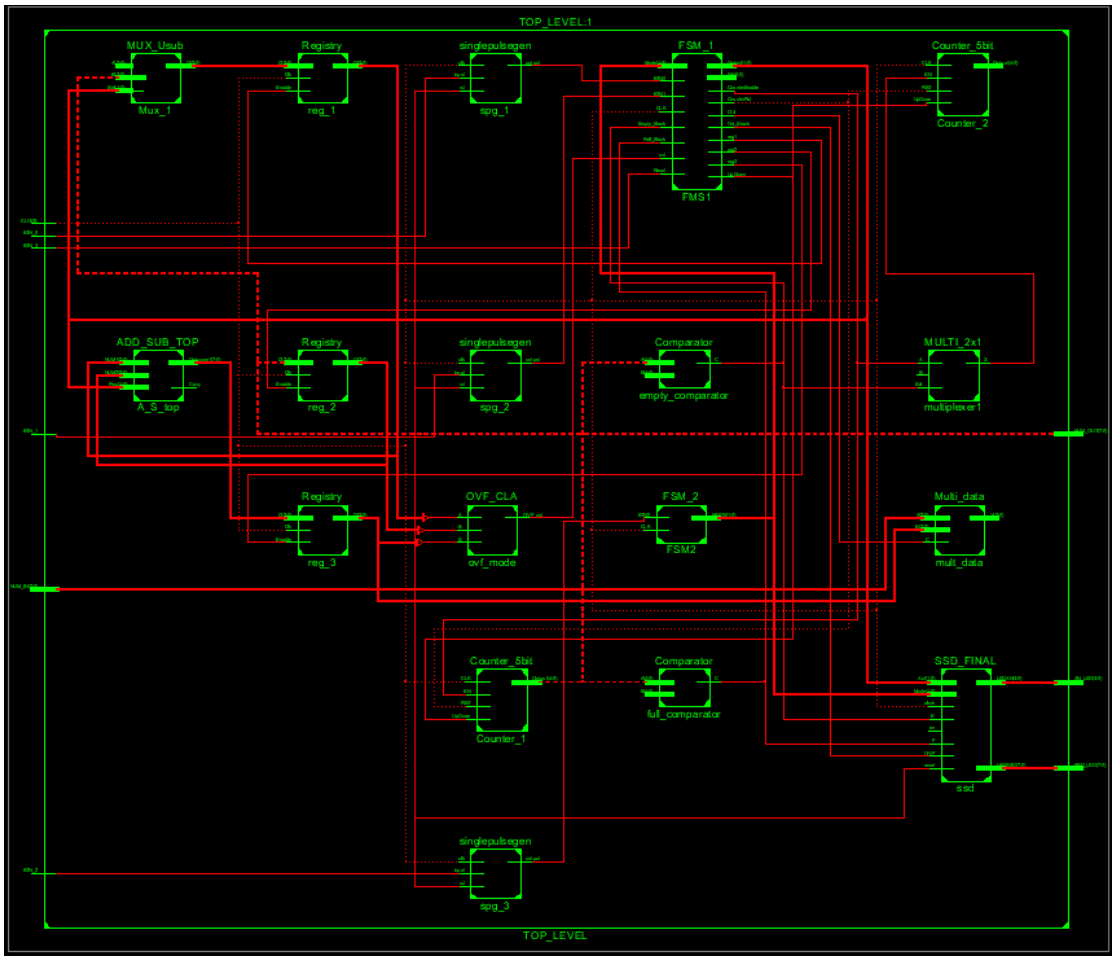
Fsm1



Fsm2

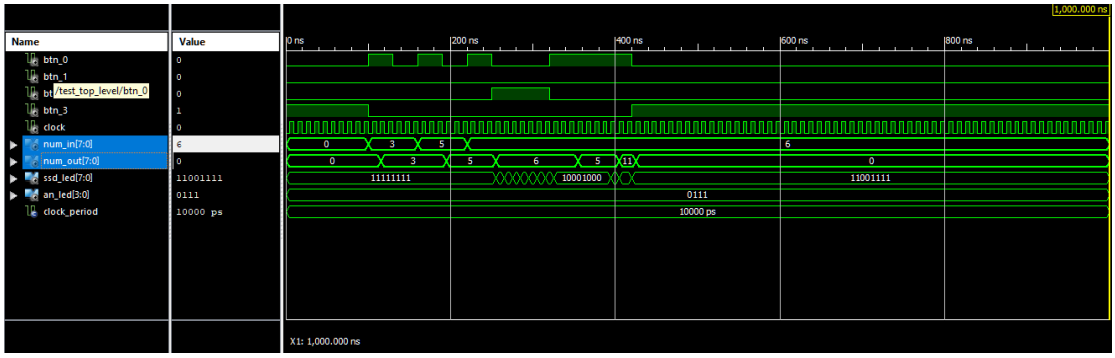


Top Level

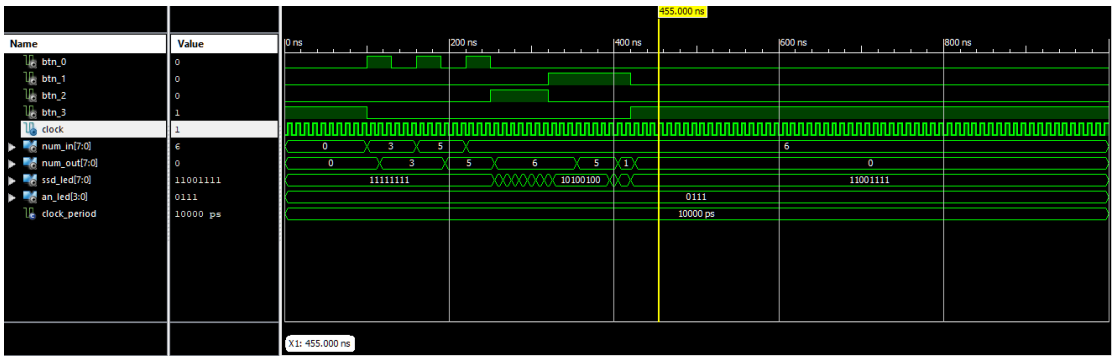


Κυματομορφές-Προσομοίωση

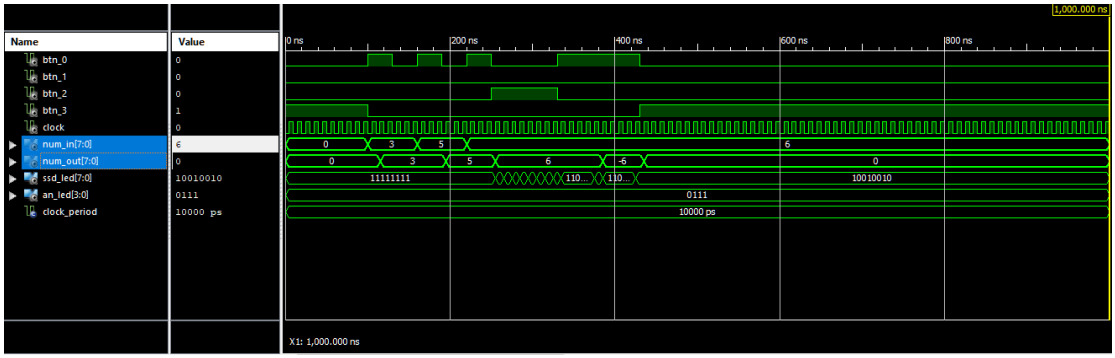
Add



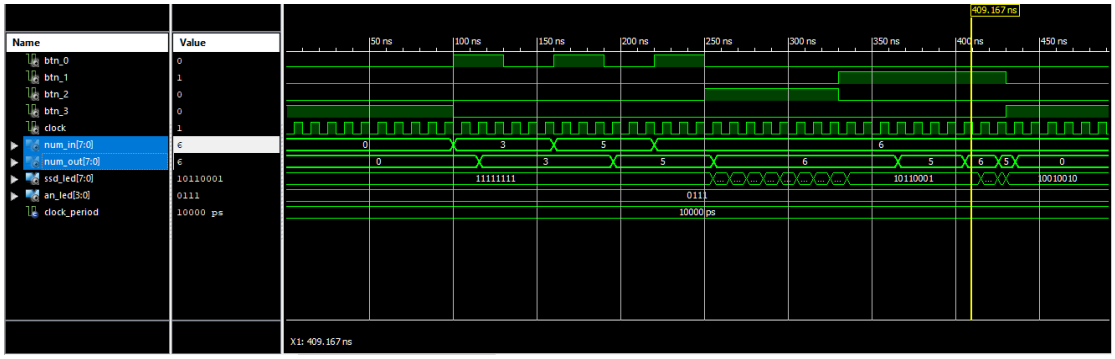
Sub



U_Sub



TOS <> TOS-1



Κώδικας

CLA

```
32  entity CLA is
33      Port ( P : in  STD_LOGIC_VECTOR (3 downto 0);
34             G : in  STD_LOGIC_VECTOR (3 downto 0);
35             Cin : in  STD_LOGIC;
36             C : out  STD_LOGIC_VECTOR (2 downto 0);
37             Cout : out  STD_LOGIC);
38  end CLA;
39
40  architecture Structural of CLA is
41
42  begin
43  C(0) <= G(0) OR (P(0) AND Cin);
44  C(1) <= G(1) OR (P(1) AND G(0)) OR (P(1) AND P(0) AND Cin);
45  C(2) <= G(2) OR (P(2) AND G(1)) OR (P(2) AND P(1) AND G(0)) OR (P(2) AND P(1) AND P(0) AND Cin);
46  Cout <= G(3) OR (P(3) AND G(2)) OR (P(3) AND P(2) AND G(1)) OR (P(3) AND P(2) AND P(1) AND P(0) AND Cin);
47
48  end Structural;
```

CPG

```
32  entity CPG is
33      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
34             B : in  STD_LOGIC_VECTOR (3 downto 0);
35             P : out  STD_LOGIC_VECTOR (3 downto 0);
36             G : out  STD_LOGIC_VECTOR (3 downto 0));
37  end CPG;
38
39  architecture Structural of CPG is
40
41  begin
42
43  P(0) <= A(0) XOR B(0);
44  P(1) <= A(1) XOR B(1);
45  P(2) <= A(2) XOR B(2);
46  P(3) <= A(3) XOR B(3);
47
48  G(0) <= A(0) AND B(0);
49  G(1) <= A(1) AND B(1);
50  G(2) <= A(2) AND B(2);
51  G(3) <= A(3) AND B(3);
52
53  end Structural;
54
```

SUM

```
32  entity SUM is
33      Port ( P : in  STD_LOGIC_VECTOR (3 downto 0);
34             C : in  STD_LOGIC_VECTOR (2 downto 0);
35             Cin : in  STD_LOGIC;
36             S : out  STD_LOGIC_VECTOR (3 downto 0));
37  end SUM;
38
39  architecture Structural of SUM is
40
41  begin
42
43  S(0) <= P(0) XOR Cin;
44  S(1) <= P(1) XOR C(0);
45  S(2) <= P(2) XOR C(1);
46  S(3) <= P(3) XOR C(2);
47
48  end Structural;
```

CLA_4_bit_Adder

```
31
32  entity CLA_4bit_ADDER is
33      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
34             B : in  STD_LOGIC_VECTOR (3 downto 0);
35             Cin : in  STD_LOGIC;
36             S : out  STD_LOGIC_VECTOR (3 downto 0);
37             C3 : out  STD_LOGIC);
38  end CLA_4bit_ADDER;
39
40  architecture Structural of CLA_4bit_ADDER is
41
42  component CPG
43  Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
44             B : in  STD_LOGIC_VECTOR (3 downto 0);
45             P : out  STD_LOGIC_VECTOR (3 downto 0);
46             G : out  STD_LOGIC_VECTOR (3 downto 0));
47  end component;
48
49  component CLA
50  Port ( P : in  STD_LOGIC_VECTOR (3 downto 0);
51             G : in  STD_LOGIC_VECTOR (3 downto 0);
52             Cin : in  STD_LOGIC;
53             C : out  STD_LOGIC_VECTOR (2 downto 0);
54             Cout : out  STD_LOGIC);
55  end component;
56
57  component SUM
58  Port ( P : in  STD_LOGIC_VECTOR (3 downto 0);
59             C : in  STD_LOGIC_VECTOR (2 downto 0);
60             Cin : in  STD_LOGIC;
61             S : out  STD_LOGIC_VECTOR (3 downto 0));
62  end component;
63
64  signal G_signal:std_logic_vector(3 downto 0);
65  signal P_signal:std_logic_vector(3 downto 0);
66  signal C_signal:std_logic_vector(2 downto 0);
67
68  begin
69  CPG_unit: CPG
70  port map( A=>A,
71            B=>B,
72            P=>P_signal,
73            G=>G_signal);
74
```

```
75  CLA_unit: CLA
76  port map( P=>P_signal,
77            G=>G_signal,
78            Cin=>Cin,
79            C=>C_signal,
80            Cout=>C3);
81
82  SUM_unit: SUM
83  port map( P=>P_signal,
84            C=>C_signal,
85            Cin=>Cin,
86            S=>S);
87
88  end Structural;
89
```

Οι παραπάνω εικόνες κώδικα είναι τα υποκυκλώματα του CLA που χρειάζονται για την υλοποίηση του σε structural δομή όπως μας ζητήθηκε τώρα και στο 2ο εργαστήριο. Για αυτό το εργαστήριο ενώσαμε 2 CLA ώστε να έχουμε τα 8 bit που χρειαζόμασταν.

ADD_SUB Function

```
32  entity ADD_SUB is
33      Port ( Praxis : in  STD_LOGIC_VECTOR (1 downto 0);
34             Num : in  STD_LOGIC_VECTOR (7 downto 0);
35             F_num : out  STD_LOGIC_VECTOR (7 downto 0);
36             Mode : out STD_LOGIC);
37  end ADD_SUB;
38
39  architecture Behavioral of ADD_SUB is
40
41  begin
42
43  process(Praxis,Num)
44  begin
45
46  If(Praxis="00") then F_num<=(Num XOR "00000000");
47                       Mode<='0';
48  elsif(Praxis="01") then F_num<=(Num XOR "11111111");
49                          Mode<='1';
50  elsif(Praxis="10") then F_num<=(Num XOR "11111111");
51                          Mode<='1';
52  else F_num<=(Num XOR "00000000");
53       Mode<='0';
54
55  end if;
56  end process;
57  end Behavioral;
```

ADDER/SUBER

```
32  entity ADD_SUB_TOP is
33      Port ( Prax : in  STD_LOGIC_VECTOR (1 downto 0);
34             NUM1 : in  STD_LOGIC_VECTOR (7 downto 0);
35             NUM2 : in  STD_LOGIC_VECTOR (7 downto 0);
36             Carry : out  STD_LOGIC;
37             Outcome1 : out  STD_LOGIC_VECTOR (7 downto 0));
38  end ADD_SUB_TOP;
39
40  architecture Structural of ADD_SUB_TOP is
41
42  component Final_CLA is
43      Port ( A_in : in  STD_LOGIC_VECTOR (7 downto 0);
44             B_in : in  STD_LOGIC_VECTOR (7 downto 0);
45             C_in : in  STD_LOGIC;
46             Outcome : out  STD_LOGIC_VECTOR (7 downto 0);
47             C_out : out  STD_LOGIC);
48  end component;
49
50  component ADD_SUB is
51      Port ( Praxis : in  STD_LOGIC_VECTOR (1 downto 0);
52             Num : in  STD_LOGIC_VECTOR (7 downto 0);
53             F_num : out  STD_LOGIC_VECTOR (7 downto 0);
54             Mode : out STD_LOGIC);
55  end component;
56
57  signal data : std_logic_vector(7 downto 0);
58  signal c : std_logic;
59  begin
60
61  CLA : Final_CLA
62        Port map(A_in=>NUM1,
63                 B_in=>data,
64                 C_in=>c,
65                 C_out=>Carry,
66                 Outcome=>Outcome1
67                 );
68  A_S : ADD_SUB
69        Port map(Praxis=>Prax,
70                 Num=>NUM2,
71                 F_Num=>data,
72                 Mode=>c
73                 );
74  end Structural;
```

REGISTER

```
entity Registry is
Port ( D : in  STD_LOGIC_VECTOR (7 downto 0);
       Enable : in  STD_LOGIC;
       Clk : in  STD_LOGIC;
       Q : out  STD_LOGIC_VECTOR (7 downto 0));
end Registry;

architecture Behavioral of Registry is

Begin

Process

Begin

Wait until( Clk'EVENT and Clk = '1' );
If (Enable = '1') then
Q <= D ;
End if;

End process ;

End Behavioral;
```

OVF ADDER/SUBER (ανίχνευση)

```
entity OVF_CLA is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           S : in  STD_LOGIC;
           OVF_out : out  STD_LOGIC);
end OVF_CLA;

architecture Behavioral of OVF_CLA is

begin
process(A,B,S)
begin

if(A='0' and B='0' and S='1') then OVF_out<='1';
elsif(A='1' and B='1' and S='0') then OVF_out<='1';
else OVF_out<='0';

end if;
end process;
end Behavioral;

```

FSM_1

```
entity FSM_1 is
    Port ( CLK : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           BTN0 : in  STD_LOGIC;
           BTN1 : in  STD_LOGIC;
           Full_Stack : in  STD_LOGIC;
           Empty_Stack : in  STD_LOGIC;
           ovf : in  STD_LOGIC;
           Mode : in STD_LOGIC_VECTOR(1 downto 0);
           CounterEnable : out  STD_LOGIC;
           UpDown : out  STD_LOGIC;
           Output : out  STD_LOGIC_VECTOR(1 downto 0); --praksi
           CounterRst : out  STD_LOGIC;
           We : out  STD_LOGIC_VECTOR(0 downto 0);
           Ovf_Stack : out  STD_LOGIC;
           Ctrl : out  STD_LOGIC;
           reg1 : out std_logic;
           reg2 : out std_logic;
           reg3 : out std_logic);
end FSM_1;

architecture Behavioral of FSM_1 is

type state_type is (state_0,state_Push,state_Push1,state_Pop,state_Ovf,state_Add,state_Sub,state_USub,state_Change,
                    state_Add_1,state_Add_2,state_Add_3,state_Add_4,state_Add_5,state_Add_6,state_Usub1,state_Usub2,state_Usub3,
                    state_Change1,state_Change2,state_Change3,state_Change4,state_Change5,state_Change6,state_Change7,state_Sub_1,
                    state_Sub_2,state_Sub_3,state_Sub_4,state_Sub_5,state_Sub_6);
signal state_next, state: state_type;
```

```
63  begin
64
65  fsm_comb: process(state, state_next, BTN0, BTN1, Empty_Stack, Full_Stack, Mode,ovf)
66  begin
67     case state is
68
69        when state_0=> We<="0";
70                       CounterEnable<='0';
71                       UpDown<='0';
72                       Ctrl<='0';
73                       reg1<='0';
74                       reg2<='0';
75                       reg3<='0';
76                       Ovf_Stack<='0';
77                       Output<="XX";
78                       If(BTN0='1' AND Full_Stack='0' AND Mode="00") then state_next<=state_Push;
79                       elsif(BTN1 ='1' AND Empty_Stack='0' AND Mode="00") then state_next<=state_Pop;
80                       elsif(BTN0='1' AND Full_Stack='1' AND Mode="00") then state_next<=state_Ovf;
81                       elsif(BTN0='1' AND Mode="01") then state_next<=state_Add;
82                       elsif(BTN1='1' AND Mode="01") then state_next<=state_Sub;
83                       elsif(BTN0='1' AND Mode="10") then state_next<=state_USub;
84                       elsif(BTN1='1' AND Mode="10") then state_next<=state_Change;
85                       else state_next<=state_0;
86                       end if;
87
88
89        when state_Push1=> state_next<=state_0;
90                           CounterEnable<='1';
91                           UpDown<='1';
92                           Ctrl<='0';
93                           reg1<='0';
94                           reg2<='0';
95                           reg3<='0';
96                           Ovf_Stack<='0';
97                           We<="0";
98                           Ctrl<='0';
99                           Output<="XX";
100
101       when state_Push=> state_next<=state_Push1;
102                          CounterEnable<='0';
103                          UpDown<='1';
104                          Ctrl<='0';
105                          reg1<='0';
106                          reg2<='0';
107                          reg3<='0';
108                          We<="1";
109                          Ctrl<='0';
110                          Ovf_Stack<='0';
111                          Output<="XX";
112
113       when state_Pop=> state_next<=state_0;
114                        CounterEnable<='1';
115                        UpDown<='0';
```

```
                    Ctrl<='0';
                    reg1<='0';
                    reg2<='0';
                    reg3<='0';
                    We<="0";
                    Ovf_Stack<='0';
                    Output<="XX";

    when state_Ovf=> Ovf_Stack<='1';
                    We<="0";
                    CounterEnable<='0';
                    UpDown<='0';
                    Ctrl<='0';
                    reg1<='0';
                    reg2<='0';
                    reg3<='0';
                    state_next<=state_Ovf;
                    Output<="XX";

    when state_Add=> We<="0";
                    Ctrl<='0';
                    CounterEnable<='0';
                    UpDown<='0';
                    reg2<='0';
                    reg3<='0';
                    reg1<='1';
                    Ovf_Stack<='0';
                    Output <= "00";
                    state_next<=state_Add_1;

    when state_Add_1=> We<="0";
                      Ctrl<='0';
                      CounterEnable<='1';
                      UpDown<='0';
                      reg1<='0';
                      Ovf_Stack<='0';
                      reg2<='0';
                      reg3<='0';
                      Output <= "00";
                      state_next<=state_Add_2;

    when state_Add_2=> We<="0";
                      Ctrl<='0';
                      CounterEnable<='0';
                      UpDown<='0';
                      reg2<='0';
                      Ovf_Stack<='0';
                      Output <= "00";
                      reg1<='0';
                      reg3<='0';
                      state_next<=state_Add_3;

    when state_Add_3=> We<="0";
                      Ctrl<='0';
                      CounterEnable<='0';
                      UpDown<='0';
                      reg2<='1';
                      reg3<='0';
                      Ovf_Stack<='0';
                      Output <= "00";
                      reg1<='0';
                      state_next<=state_Add_4;

    when state_Add_4=> We<="0";
                      Ctrl<='0';
                      CounterEnable<='0';
                      UpDown<='0';
                      reg2<='0';
                      reg3<='1';
                      Ovf_Stack<='0';
                      Output <= "00";
                      reg1<='0';
                      state_next<=state_Add_5;

    when state_Add_5=> if(ovf='1') then
                      We<="0";
                      CounterEnable<='0';
                      UpDown<='0';
```

```
193                         UpDown<='0';
194                         Ctrl<='0';
195                         reg1<='0';
196                         reg2<='0';
197                         reg3<='0';
198                         Ovf_Stack<='0';
199                         Output <= "XX";
200                         state_next<=state_0;
201                         else
202                         We<="1";
203                         Ctrl<='1';
204                         CounterEnable<='0';
205                         UpDown<='0';
206                         reg1<='0';
207                         reg2<='0';
208                         reg3<='1';
209                         Ovf_Stack<='0';
210                         Output <= "00";
211                         state_next<=state_Add_6;
212                         end if;
213
214       when state_Add_6=> We<="0";
215                         Ctrl<='0';
216                         CounterEnable<='1';
217                         UpDown<='1';
218                         reg1<='0';
219                         reg2<='0';
220                         reg3<='0';
221                         Ovf_Stack<='0';
222                         Output <= "00";
223                         state_next<=state_0;
224
225       when state_Sub=> We<="0";
226                        CounterEnable<='0';
227                        UpDown<='0';
228                        Ctrl<='0';
229                        reg1<='1';
230                        Ovf_Stack<='0';
231                        Output <= "01";
232                        reg2<='0';
233                        reg3<='0';
234                        state_next<=state_Sub_1;
235
236       when state_Sub_1=> We<="0";
237                         Ctrl<='0';
238                         CounterEnable<='1';
239                         UpDown<='0';
240                         reg1<='0';
241                         Ovf_Stack<='0';
242                         reg2<='0';
243                         reg3<='0';
244                         Output <= "01";
245                         state_next<=state_Sub_2;
246
247       when state_Sub_2=> We<="0";
248                         Ctrl<='0';
249                         CounterEnable<='0';
250                         UpDown<='0';
251                         reg2<='0';
252                         Ovf_Stack<='0';
253                         Output <= "01";
254                         reg1<='0';
255                         reg3<='0';
256                         state_next<=state_Sub_3;
257
258       when state_Sub_3=> We<="0";
259                         Ctrl<='0';
260                         CounterEnable<='0';
261                         UpDown<='0';
262                         reg2<='1';
263                         reg3<='0';
264                         Ovf_Stack<='0';
265                         Output <= "01";
266                         reg1<='0';
267                         state_next<=state_Sub_4;
268
269       when state_Sub_4=> We<="0";
270                         Ctrl<='0';
```

```
                        CounterEnable<='0';
                        UpDown<='0';
                        reg2<='0';
                        reg3<='1';
                        Ovf_Stack<='0';
                        Output <= "01";
                        reg1<='0';
                        state_next<=state_Sub_5;

      when state_Sub_5=> if(ovf='1') then
                        We<="0";
                        CounterEnable<='0';
                        UpDown<='0';
                        Ctrl<='0';
                        reg1<='0';
                        reg2<='0';
                        reg3<='0';
                        Ovf_Stack<='0';
                        Output <= "XX";
                        state_next<=state_0;
                        else
                        We<="1";
                        Ctrl<='1';
                        CounterEnable<='0';
                        UpDown<='0';
                        reg1<='0';
                        reg2<='0';
                        reg3<='1';
                        Ovf_Stack<='0';
                        Output <= "01";
                        state_next<=state_Sub_6;
                        end if;

      when state_Sub_6=> We<="0";
                        Ctrl<='0';
                        CounterEnable<='1';
                        UpDown<='1';
                        reg1<='0';
                        reg2<='0';
                        reg3<='0';
                        Ovf_Stack<='0';
                        Output <= "01";
                        state_next<=state_0;



      when state_Usub=> We<="0";
                       CounterEnable<='0';
                       UpDown<='0';
                       Ctrl<='0';
                       reg1<='1';
                       reg2<='1';
                       Ovf_Stack<='0';
                       Output <= "10";
                       reg3<='0';
                       state_next<=state_Usub1;

      when state_Usub1=> We<="0";
                        CounterEnable<='0';
                        UpDown<='0';
                        Ctrl<='0';
                        reg1<='1';
                        reg2<='1';
                        reg3<='1';
                        Ovf_Stack<='0';
                        Output <= "10";
                        state_next<=state_Usub2;

      when state_Usub2=> We<="1";
                        Ctrl<='1';
                        CounterEnable<='0';
                        UpDown<='0';
                        reg1<='0';
                        reg2<='0';
                        reg3<='0';
                        Ovf_Stack<='0';
```

```
348                         Output <= "10";
349                         state_next<=state_Usub3;
350
351         when state_Usub3=> We<="0";
352                         Ctrl<='0';
353                         CounterEnable<='1';
354                         UpDown<='1';
355                         reg1<='0';
356                         reg2<='0';
357                         reg3<='0';
358                         Ovf_Stack<='0';
359                         Output <= "10";
360                         state_next<=state_0;
361
362         when state_Change=> We<="0";
363                          CounterEnable<='0';
364                          UpDown<='0';
365                          reg1<='1';
366                          reg2<='1';
367                          Ovf_Stack<='0';
368                          Output <= "11";
369                          reg3<='1';
370                          Ctrl<='0';
371                          state_next<=state_Change1;
372
373         when state_Change1=>We<="0";
374                          CounterEnable<='1';
375                          UpDown<='0';
376                          reg1<='0';
377                          reg2<='0';
378                          reg3<='0';
379                          Ovf_Stack<='0';
380                          Output <= "11";
381                          Ctrl<='0';
382                          state_next<=state_Change2;
383
384         when state_Change2=>We<="0";
385                          CounterEnable<='0';
386                          UpDown<='0';
387                          reg1<='0';
388                          reg2<='0';
389                          reg3<='1';
390                          Ovf_Stack<='0';
391                          Output <= "11";
392                          Ctrl<='0';
393                          state_next<=state_Change3;
394
395         when state_Change3=>We<="0";
396                          Ctrl<='0';
397                          CounterEnable<='0';
398                          UpDown<='0';
399                          reg1<='0';
400                          reg2<='1';
401                          reg3<='0';
402                          Ovf_Stack<='0';
403                          Output <= "11";
404                          state_next<=state_Change4;
405
406         when state_Change4=>We<="1";
407                          Ctrl<='1';
408                          CounterEnable<='0';
409                          UpDown<='0';
410                          reg1<='0';
411                          reg2<='0';
412                          reg3<='1';
413                          Ovf_Stack<='0';
414                          Output <= "11";
415                          state_next<=state_Change5;
416
417         when state_Change5=>We<="0";
418                          Ctrl<='0';
419                          CounterEnable<='1';
420                          UpDown<='1';
421                          reg1<='0';
422                          reg2<='0';
423                          reg3<='0';
424                          Ovf_Stack<='0';
425                          Output <= "11";
```

```
                        state_next<=state_Change6;

      when state_Change6=>We<="1";
                        Ctrl<='1';
                        CounterEnable<='0';
                        UpDown<='0';
                        reg1<='0';
                        reg2<='0';
                        reg3<='1';
                        Ovf_Stack<='0';
                        Output <= "11";
                        state_next<=state_Change7;

      when state_Change7=>We<="0";
                        Ctrl<='0';
                        CounterEnable<='1';
                        UpDown<='1';
                        reg1<='0';
                        reg2<='0';
                        reg3<='0';
                        Ovf_Stack<='0';
                        Output <= "11";
                        state_next<=state_0;

      when others=> state_next<=state_0;
                    Ovf_Stack<='0';
                    We<="0";
                    CounterEnable<='0';
                    UpDown<='0';
                    Ctrl<='0';
                    reg1<='0';
                    reg2<='0';
                    reg3<='0';
                    Output<="XX";

end case;
end process fsm_comb;

fsm_synch: process(Reset, CLK)
begin
   if (Reset='1') then state <= state_0;
                  CounterRst<='1';
   elsif (rising_edge(CLK)) then state <= state_next;
                                 CounterRst<='0';
   end if;
end process fsm_synch;


end Behavioral;
```