# ΠΡΟΧΩΡΗΜΕΝΗ ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ

## ΑΝΑΦΟΡΆ 4ου ΕΡΓΑΣΤΗΡΙΟΥ

**LAB** 20335359

**Γιώργος Βιριράκης** 2016030035
**Μιχάλης Γαλάνης** 2016030036

# ΣΚΟΠΟΣ

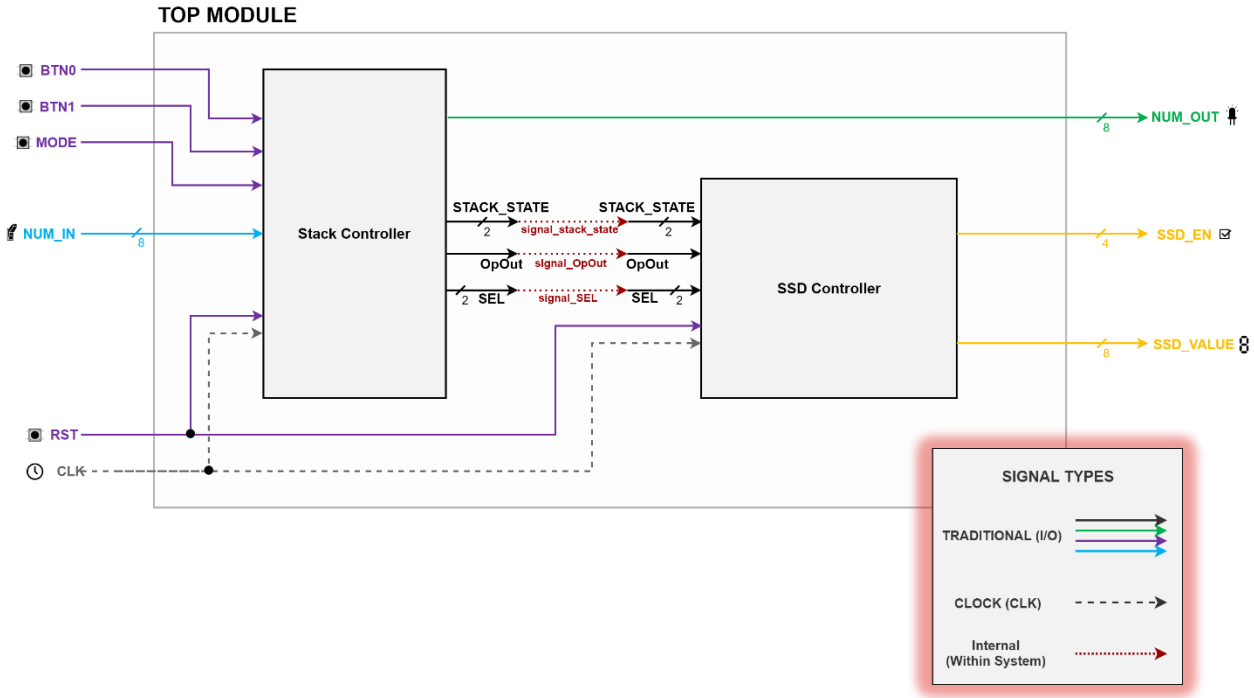
Σκοπός του 4ου εργαστηρίου είναι το πρώτο βήμα για τη δημιουργία μιας αριθμομηχανής που λειτουργεί με τη μέθοδο **Reverse Polish Notation** (5ο εργαστήριο). Αυτή εργαστηριακή άσκηση είναι επέκταση του εργαστηρίου 3 και ασχολούμαστε απλώς με την **αναγνώριση πράξεων** (δεν υλοποιούνται) ανάλογα με το **mode λειτουργίας** που είναι ενεργοποιημένο.

# ΠΡΟΕΡΓΑΣΙΑ – 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 | $BTN_1$ |
| Mode | 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 ανανεωμένο **TOP MODULE** (με νέο MODE input και εσωτερικά σήματα) φαίνεται παρακάτω:



Το τελικό μας κύκλωμα σύμφωνα με την εκφώνηση πρέπει να υποστηρίζει **6 αριθμητικές/λογικές πράξεις:**

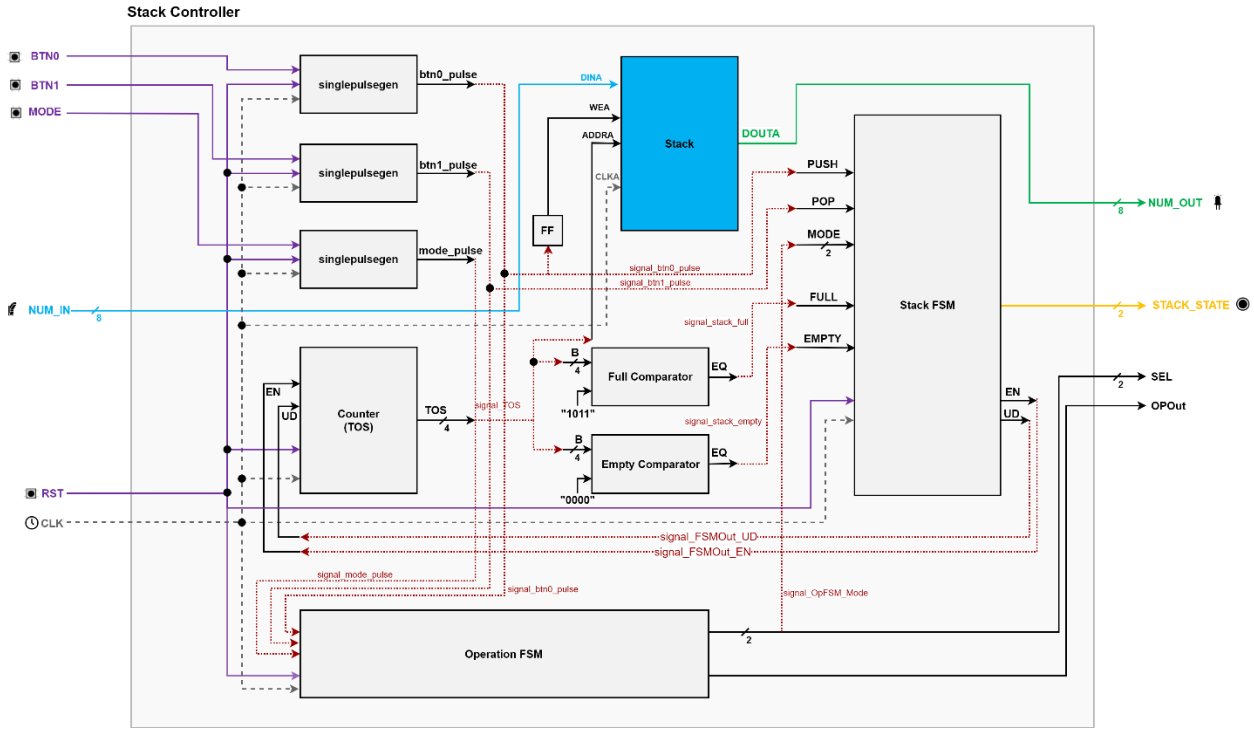
- **Push**
- **Pop**
- **Πρόσθεση αριθμών (A)**
- **Αφαίρεση αριθμών (S)**
- **Μοναδιαία αφαίρεση (αντίθετος αριθμός) (U)**
- **Εναλλαγή TOS με TOS-1 (<>)**

Στο FPGA board μας όμως έχουμε διαθέσιμα **μόνο 4 push buttons.** Η λύση σε αυτό το πρόβλημα είναι η δημιουργία ενός συστήματος **mode-based** όπου το **$BTN_2$** είναι υπεύθυνο για την εναλλαγή του mode, τα **$BTN_0$, $BTN_1$** είναι υπεύθυνα για τις πράξεις και το **$BTN_3$** πραγματοποιεί το reset. Άρα χωρίζοντας τις 6 πράξεις σε 3 modes έχουμε:

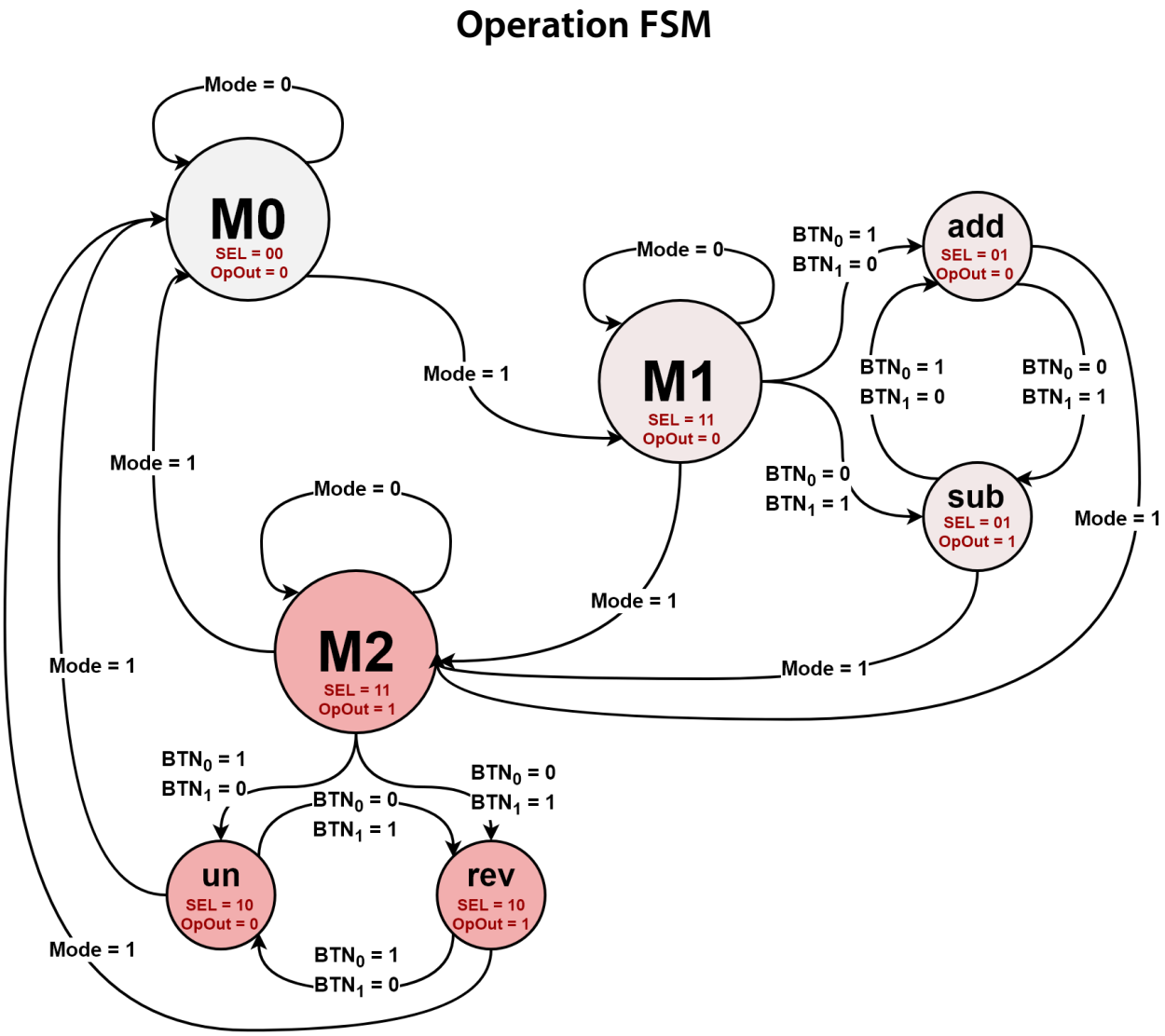
| Button | Action | | |
|---|---|---|---|
| | Mode 0 | Mode 1 | Mode 2 |
| $BTN_0$ | Push | Add (A) | Unary (U) |
| $BTN_1$ | Pop | Sub (S) | Rev (<>) |

**Σημείωση:** Η κάθε πράξη είναι ανεξάρτητη και δε διακόπτεται από την αλλαγή του mode.

Με τα παραπάνω δεδομένα, ο ανανεωμένος **Stack Controller** παρουσιάζεται σε επόμενο σχήμα:



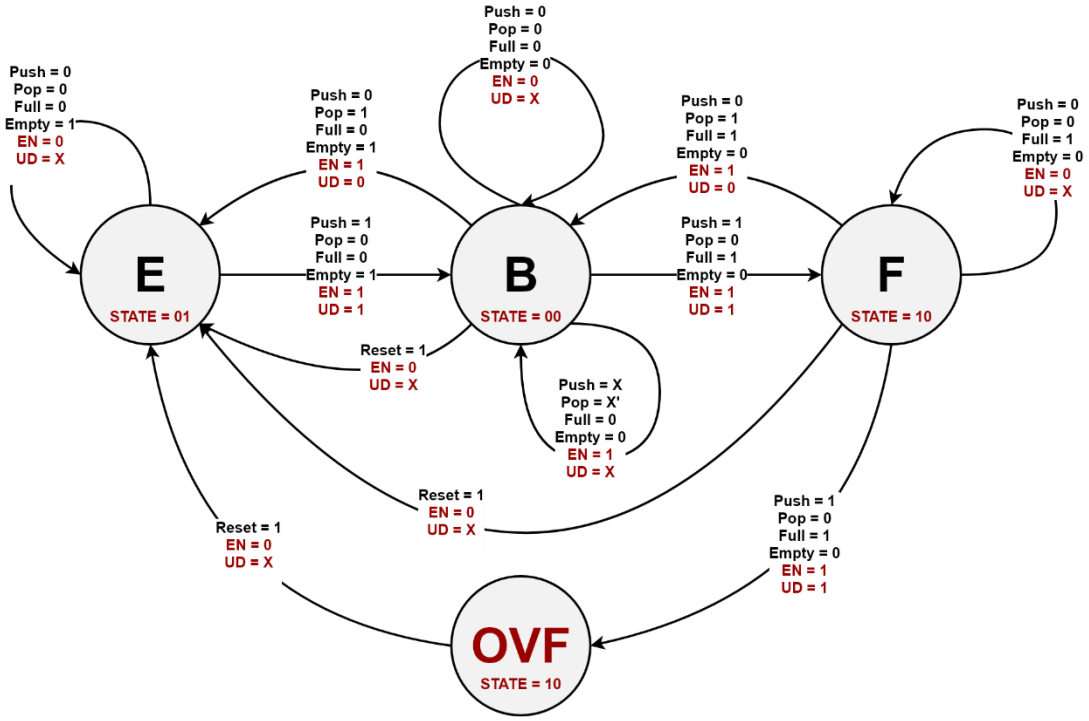
Έχουμε λοιπόν μια καινούργια FSM (Operation FSM) η οποία είναι η υπεύθυνη για την επιλογή των modes στο κύκλωμά μας. Είναι τύπου **Moore** και το διάγραμμα καταστάσεων της φαίνεται παρακάτω:




**Σημείωση:** Το σήμα **$BTN_3$ (Reset)** επαναφέρει τη μηχανή στην κατάσταση **M0** (δε φαίνεται στο σχήμα για λόγους διατήρησης απλότητας).

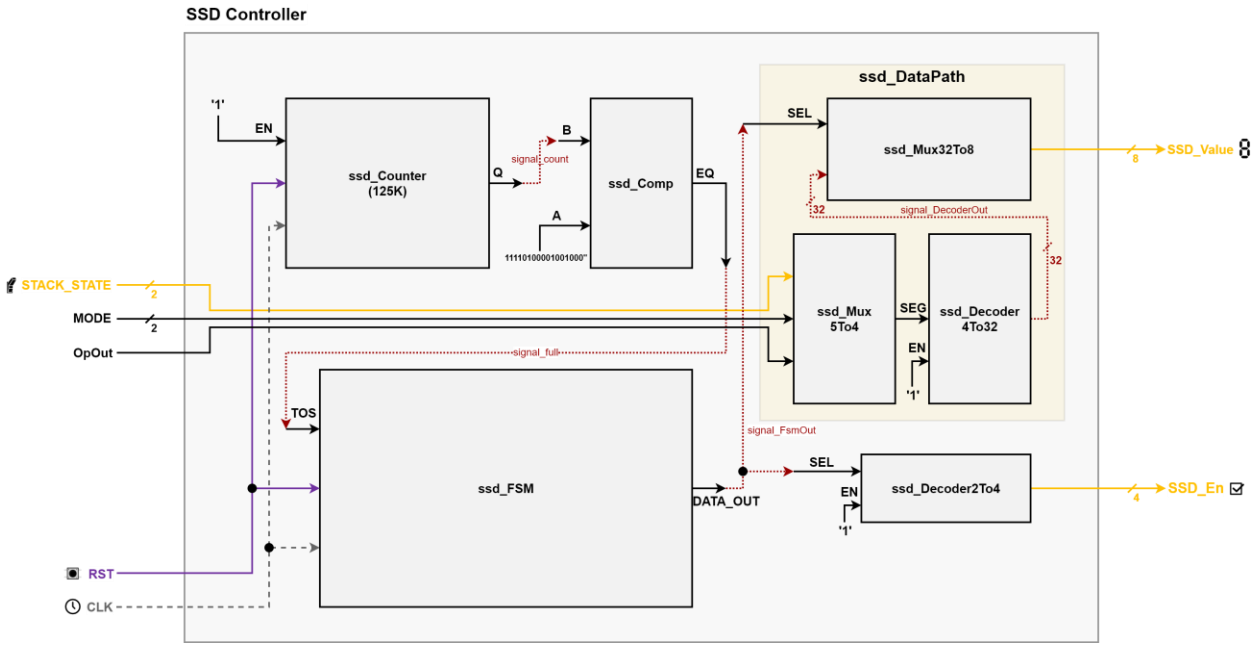
Η Stack FSM μεταβάλλεται ως εξής: Δέχεται μια παραπάνω είσοδο **MODE** (2-bit έξοδος της OperationFSM). Εάν το **MODE** έχει τιμή **"00"** ισχύουν οι μεταβάσεις του παρακάτω διαγράμματος. Διαφορετικά η FSM παράγει εξόδους: **STATE = "00", EN = '0', UD = '0'** (δε φαίνονται στο διάγραμμα για αποφυγή πολυπλοκότητας).

**IF MODE = “00”:**




**ELSE STATE = “00”, EN = ‘0’, UD = ‘0’** (οι καταστάσεις δεν επηρεάζονται από το Mode)**.**

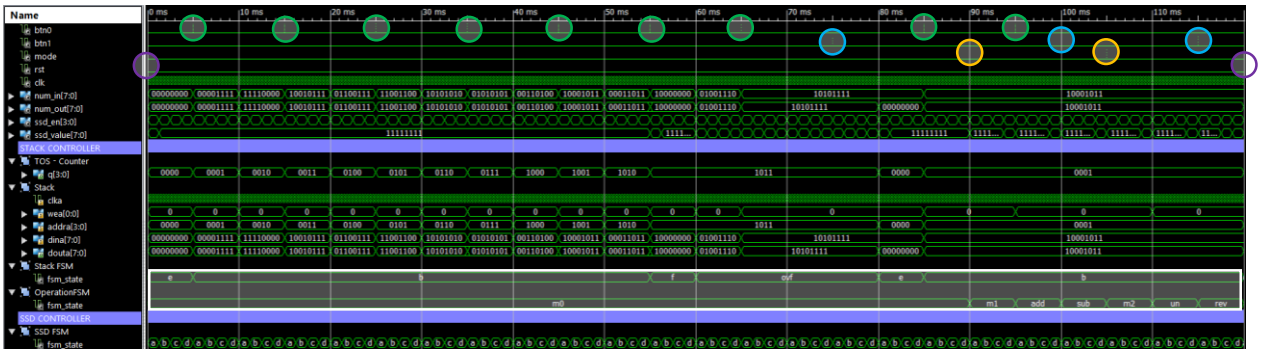
Το κομμάτι του SSD Controller δεν αλλάζει ιδιαίτερα, παρα μόνο το module **ssd_DataPath**. Πιο συγκεκριμένα, εκτός από είσοδο **STACK_STATE**, έχουμε πλέον και εισόδους **OpOut** και **SEL.** Θα χρειαστούμε λοιπόν έναν νέο πολυπλέκτη πριν τον **ssd_Decoder** που θα διαχειρίζεται και τα 2 καινούργια σήματα.



# ΚΥΜΑΤΟΜΟΡΦΕΣ

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

- **BTN$_0$** pulse (push, add, un)
- **BTN$_1$** pulse (pop, sub, rev)
- **BTN$_2$** pulse (mode)
- **reset**



# ΣΥΜΠΕΡΑΣΜΑΤΑ

Στο εργαστήριο αυτό θέσαμε τις βάσεις για τη δημιουργία μιας αριθμομηχανής που θα υλοποιεί κάποιες βασικές πράξεις (αναγνώριση μόνο). Εξοικειωθήκαμε περαιτέρω με τη γλώσσα VHDL αλλά και τις λειτουργίες ενός FPGA και αναμένουμε το 5ο και τελευταίο εργαστήριο.

# ΠΑΡΑΡΤΗΜΑ – ΚΩΔΙΚΑΣ

Στο εργαστήριο 4, το μόνο αξιοσημείωτο τμήμα του κώδικα θεωρήσαμε την **Operation FSM** (Mode Selector) η οποία φαίνεται στην επόμενη σελίδα.

# Operation FSM

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

entity OperationFSM is
    Port ( BTN0 : in  STD_LOGIC;
           BTN1 : in  STD_LOGIC;
           MODE : in  STD_LOGIC;
           CLK : in STD_LOGIC;
           RST : in  STD_LOGIC;
           OPOut : out  STD_LOGIC;
           SEL : out  STD_LOGIC_VECTOR (1 downto 0));
end OperationFSM;

architecture Behavioral of OperationFSM is

   type states is (M0,M1,M2,add,sub,un,rev);
   signal fsm_state : states;

begin


FSM: Process (CLK)
begin



   if rising_edge(CLK) then
      if (RST = '1') then fsm_state <= M0;
      else
         case fsm_state is
            when M0 => if (MODE = '1') then fsm_state <= M1;
                       else fsm_state <= M0;
                       end if;
            when M1 => if (MODE = '1') then fsm_state <= M2;
                       elsif (BTN0 = '1' and BTN1 = '0') then fsm_state <= add;
                       elsif (BTN1 = '0' and BTN1 = '1') then fsm_state <= sub;
                       else fsm_state <= M1;
                       end if;
            when M2 => if (MODE = '1') then fsm_state <= M0;
                       elsif (BTN0 = '1' and BTN1 = '0') then fsm_state <= un;
                       elsif (BTN1 = '0' and BTN1 = '1') then fsm_state <= rev;
                       else fsm_state <= M2;
                       end if;
            when add => if (MODE = '1') then fsm_state <= M2;
                        elsif (BTN0 = '0' and BTN1 = '1') then fsm_state <= sub;
                        else fsm_state <= add;
                        end if;
            when sub => if (MODE = '1') then fsm_state <= M2;
                        elsif (BTN0 = '1' and BTN1 = '0') then fsm_state <= add;
                        else fsm_state <= sub;
                        end if;
            when un => if (MODE = '1') then fsm_state <= M0;
                        elsif (BTN0 = '0' and BTN1 = '1') then fsm_state <= rev;
                        else fsm_state <= un;
                        end if;
            when rev => if (MODE = '1') then fsm_state <= M0;
                        elsif (BTN0 = '1' and BTN1 = '0') then fsm_state <= un;
                        else fsm_state <= rev;
                        end if;
            when others => fsm_state <= M0;
         end case;
      end if;
   end if;




end Process;


Outputs: Process (fsm_state)
begin
   case fsm_state is
      when M0 => SEL <= "00";
                 OpOut <= '0';

      when M1 => SEL <= "11";
                 OpOut <= '0';

      when M2 => SEL <= "11";
                 OpOut <= '1';
      when add => SEL <= "01";
                  OpOut <= '0';
      when sub => SEL <= "01";
                  OpOut <= '1';
      when un => SEL <= "10";
                  OpOut <= '0';
      when rev => SEL <= "10";
                  OpOut <= '1';
      when others => SEL <= "00";
                     OPOut <= '0';


   end case;
end Process;


end Behavioral;
```