25/3/2017

2ή Εργαστηριακή Άσκηση

Εξοικείωση με την γλώσσα περιγραφής υλικού και την ιεραρχική σχεδίαση

(Structural Vhdl)

25/03/2017

Ομάδα LAB20332005

ΧΡΗΣΤΟΣ ΖΗΣΚΑΣ 2014030191
ΠΑΝΑΓΙΩΤΗΣ ΣΑΒΒΑΙΔΗΣ 2013030180

Σκοπός εργαστηριακής άσκησης

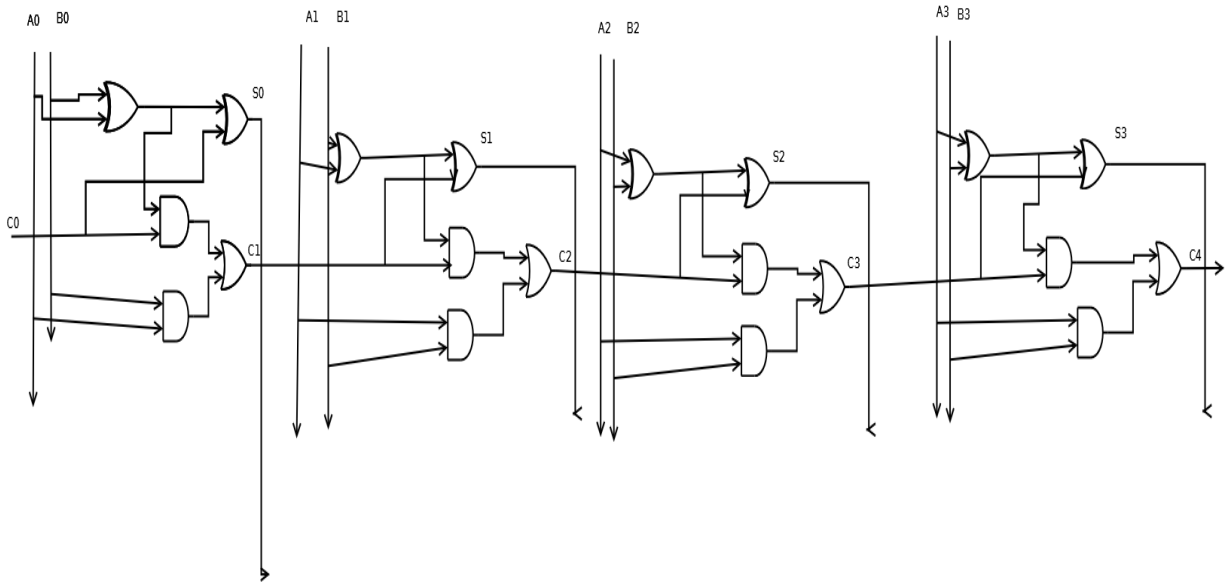
Είναι η εξοικείωση με τη γλώσσα περιγραφής υλικού VHDL(VHSIC Hardware Description Language) για περαιτέρω εξοικείωση σε προχωρημένη σχεδίαση, καθώς επίσης και με την υλοποίηση ιεραρχικής δομής πολλαπλών αρχείων. Επιπλέον, γίνεται μια επιπρόσθετη προσέγγιση πάνω στο σχεδιαστικό πρόγραμμα Xilinx ISE δημιουργώντας έναν adder 4-bit ιεραρχικής σχεδίασης (αθροιστής πρόβλεψης κρατουμένου CLA) καθώς επίσης και την κατασκευή συνδυαστικού κυκλώματος 3 καταστάσεων - Μηχανή πεπερασμένων καταστάσεων (FSM).

Προεργασία

Παρουσιάζουμε τις σχέσεις για τον αθροιστή 4-bit που αφορά τις συσχετίσεις του κρατουμένου Carry για τα διάφορα bit πρόσθεσης- σε ποια θέση bit δημιουργείται το κρατούμενο η αν προέρχεται από κάποιο προηγούμενο πρόσθεση bit- καθώς και την σχεδίαση του κυκλώματος σε διάγραμμα πυλών. Στο 2ο κύκλωμα απεικονίζονται η διάφορες εναλλαγές καταστάσεων για τις εκάστοτε εισόδους καθώς και η έξοδος τις μηχανής για την κάθε κατάσταση. (δεν διαφοροποιείται η έξοδος του κυκλώματος από την διαφορετική είσοδο, μόνο κατάσταση στην οποία γίνεται η μετάβαση).

Κύκλωμα 1

Συνδέσεις κρατουμένου για αθροιστή πρόβλεψης κρατουμένου 4-bit (CLA διάγραμμα πυλων)



Για τον συγκεκριμένο αθροιστή δημιουργούνται 3 μονάδες

Carry Generate/Propagate Unit: Υπολογίζει τα τέσσερα σήματα propagate και carry generate.

Pi=AiBi'+Ai'Bi

Gi=AiBi

Carry Look Ahead Unit: Υπολογίζει τα 3 σήματα Carry Look Ahead και το Carry Out του αθροιστή.

C0=G0+P0Cin

C1=G1+P1G0+P1P0Cin

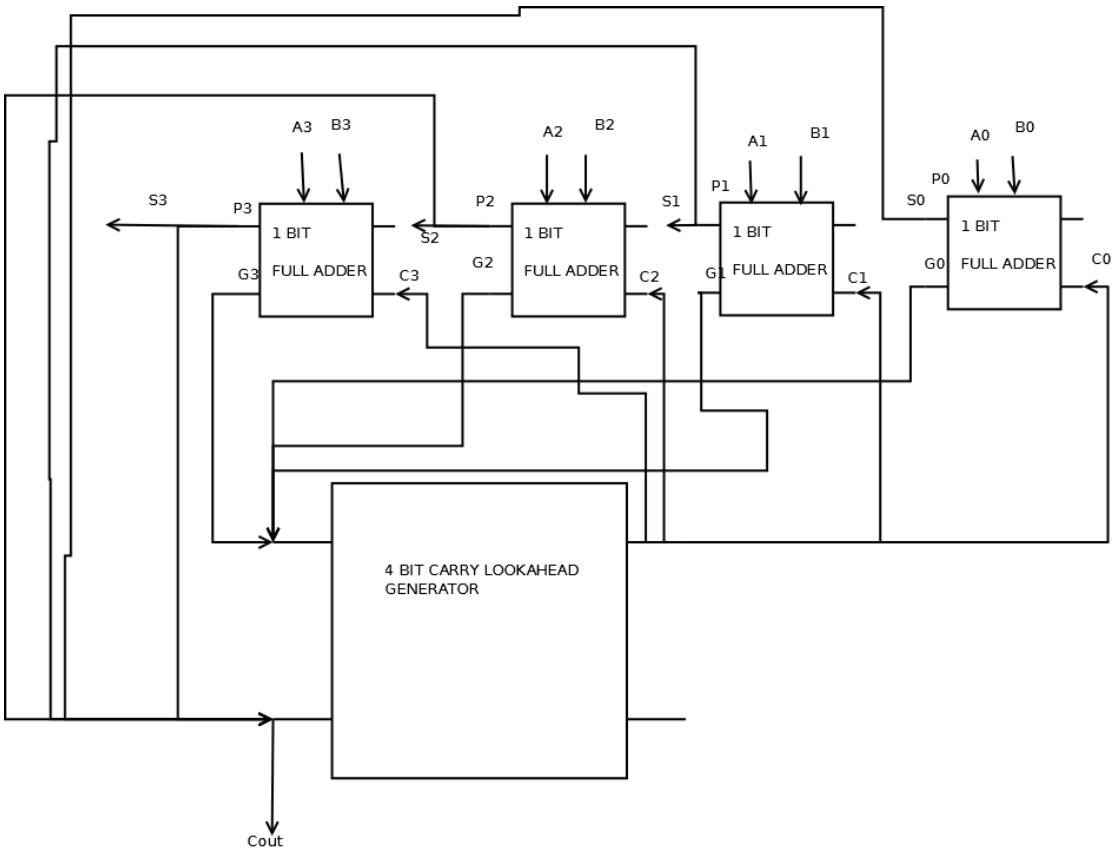
C2=G2+P2G1+P2P1G0+P2P1P0Cin

C3=G3+P3G2+P3P2G1+P3P2P1PG0+P3P2P1P0Cin

Sum Unit: Υπολογίζει τα τέσσερα σήματα του αθροίσματος.

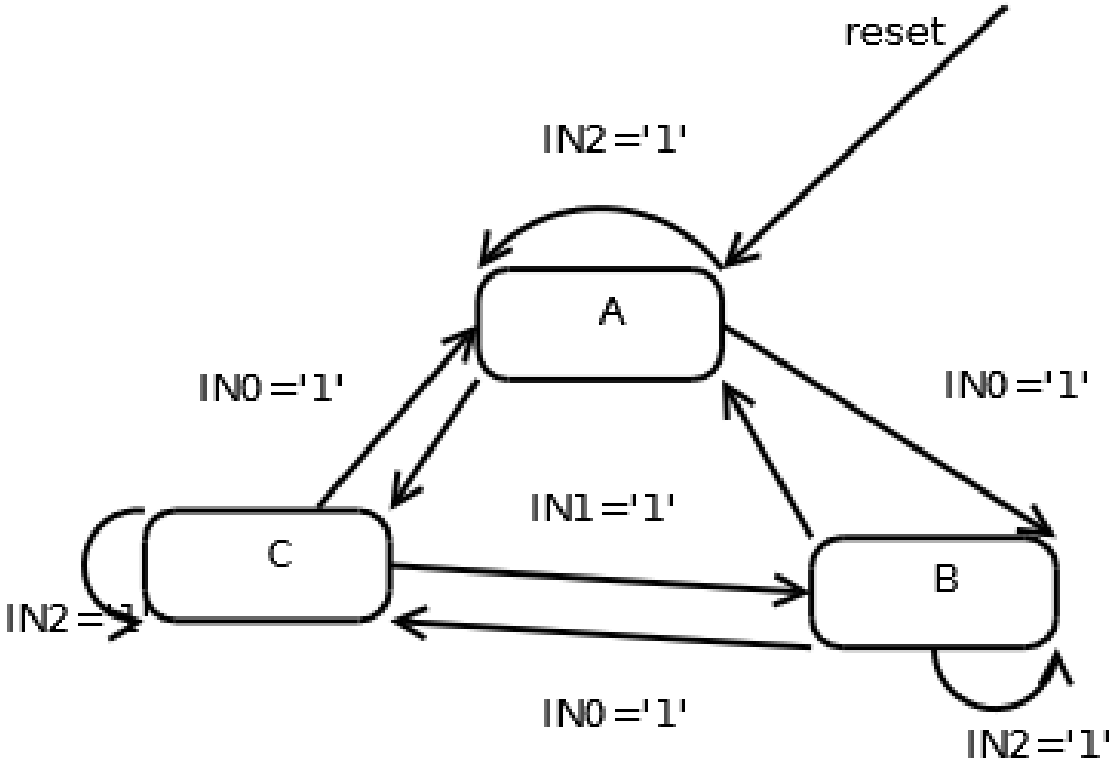
Si=Ai'BiCi+AiBi'Ci+AiBiCi'+AiBiCi'

Συνδέσεις κρατουμένου για αθροιστή πρόβλεψης κρατουμένου 4-bit (CLA block diagram)



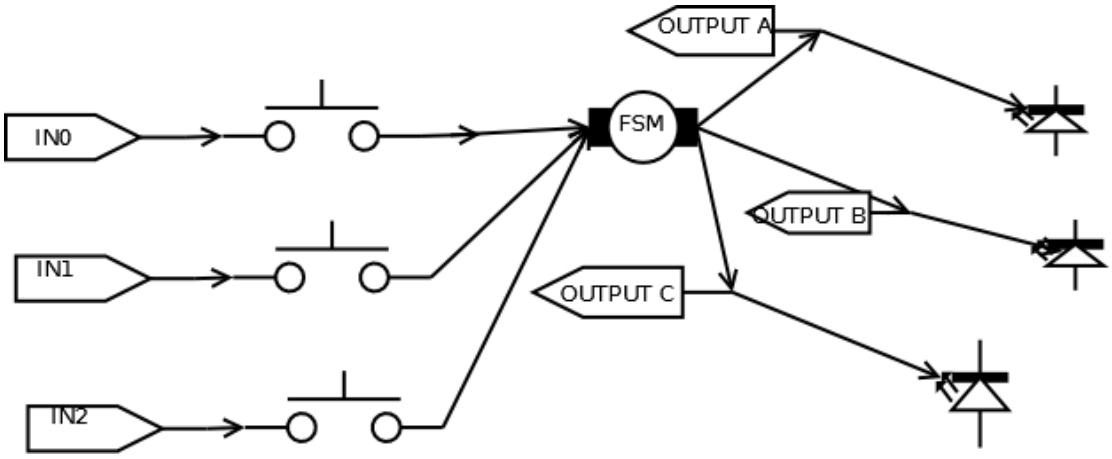
Κύκλωμα 2

Υλοποίηση Μηχανής Πεπερασμένων Καταστάσεων (FSM)



Περιγραφή

Για το κύκλωμα 1, ζητείται να υλοποιηθεί συμπεριλαμβανομένου και της σχεδίασης ενός κυκλώματος αθροιστή 4-bit με αριθμητικές εισόδους A,B και κρατουμένου εισόδου Cin. Δημιουργούνται τα σήματα propagate και generate με πύλες AND και OR. Το συνολικό SUM δημιουργείται από τον συνδυασμό των XOR των propagate μαζί με το carry από την πρόσθεση των εκάστοτε bit. Τα διάφορα κρατούμενα δημιουργούνται είτε από κάποιο generate στα τρέχων bit η έχει προέρθει από κάποιο propagate προηγούμενων bit .Τα κρατούμενα συνδέονται ως είσοδο σε επόμενο κρατούμενο(Τα Ci βρίσκονται ως είσοδος σε πύλη AND με τα propagate των προηγούμενων bit). Το Cout θεωρούμε το τελευταίο κρατούμενο που δημιουργείται .Για την υλοποίηση του κυκλώματος 2 , θεωρούμε τρεις καταστάσεις A,B,C με κατάσταση RESET την A. Για είσοδο IN0 γίνεται μετάβαση στην επόμενη κατάστασή ενώ για είσοδο IN1 ,προορισμός γίνεται η προηγούμενη κατάσταση. Για είσοδο IN2 η μηχανή παραμένει στην τρέχουσα θέση της. Κάθε κατάσταση αποδίδει διαφορετική έξοδο που δεν περιορίζεται από τις εκάστοτε εισόδους.. Δυο κουμπιά ταυτόχρονα καθίσταται αδύνατη περίπτωση. Ουσιαστικά η υλοποίηση ακολουθεί την διάταξη:



Για τις εξόδους:

OUTPUT A=> ΌΛΑ ΤΑ LED ΑΝΑΜΜΈΝΑ

OUTPUT B=> ΤΑ ΔΥΟ ΔΕΞΙΆ ΚΑΙ ΤΑ ΔΥΟ ΑΡΙΣΤΕΡΆ LED ΑΝΑΜΜΈΝΑ ΚΑΙ ΤΑ ΥΠΌΛΟΙΠΑ ΣΒΗΣΤΆ

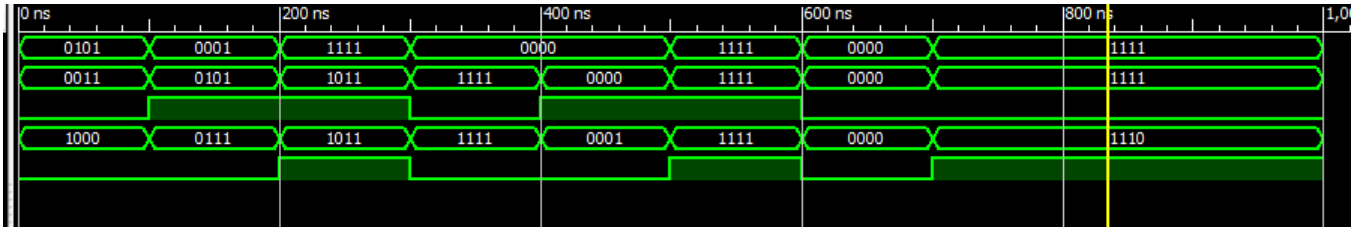
OUTPUT C=> ΤΑ ΤΈΣΣΕΡΑ ΜΕΣΑΊΑ LED ΑΝΑΜΜΈΝΑ ΚΑΙ ΤΑ ΥΠΌΛΟΙΠΑ ΣΒΗΣΤΆ

Κυματομορφές-Προσομοίωση

Παρουσιάζουμε τις κυμματομορφές των 2 κυκλωμάτων

Κύκλωμα 1

Αθροιστή με πρόβλεψη κρατουμένου 4-bit(Carry Look Ahead Adder -CLA-)

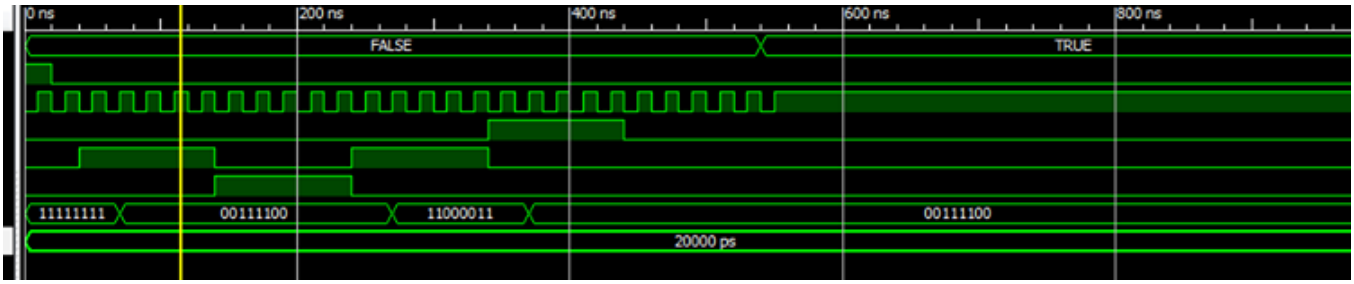


Για την απεικόνιση των πράξεων στην FPGA απαιτούνται 5 led καθώς οι αριθμητές είναι 4 bit και μπορούν να παράξουν αριθμό έως 5 bit με το MSB να αναπαριστά το κρατούμενο εξόδου. Το κρατούμενο μπορεί να έχει προέρθει από το generate στην θέση i ή από κάποιο propagate σε θέση πριν από το I. Τα κρατούμενα μπορεί να προχωρούν στις επόμενες θέσεις bit ,ενώ για την παραγωγή του C(0) συνεισφέρει το Cin. Τα 4 bit που απεικονίζονται ως αποτέλεσμα τις προσθέσεις των 2 αριθμών προέρχονται από μια πύλη XOR των propagate και carry

Παρατηρούμε ότι δίνοντας ως εισόδους τα σήματα 0101 και 0011 (αναπαριστούν τους αριθμούς 5,3) και χωρίς να πυροδοτήσουμε το Cin μας δίνουν τον αριθμό 1000 που αναμέναμε(το LSB είναι SUM- 0 έχει ως P-propagate 0(1 XOR 1=>0) , ως G-Generate 1(1 and 1=>1) -Cin 0-- C(0) 1 (1 OR (0 AND 0))- - P XOR Cin =0 => ΣΩΣΤΗ ΑΠΕΙΚΌΝΙΣΗ, Για το 2ο bit έχουμε P-propagate 1 (0 XOR 1=>1) , ως G-generate 0(1 AND 0=>0) - - P XOR C => (1 XOR 1=>0 = SUM, κλπ για τα υπόλοιπα bit) . Στις ακραίες περιπτώσεις για αριθμούς 1111 και 1111 με CarryIn παρατηρούμε ότι η απόκριση του κυκλώματος δίνει CarryOut 1 και ως SUM 1111.(15 +15+1=31 , 1111+1111+Cin). Για αριθμούς 0000 και 0000 με carryIN =1 βλέπουμε ότι το αρχικό carry συνεισφέρει μόνο στο propagate(0000+0000+1=>0001 ΕΠΑΛΗΘΕΥΣΗ 0+0+1=1).

Κύκλωμα 2

Μηχανή Πεπερασμένων Καταστάσεων (Finite State Machine -Moore Fsm-)



Η κυμματομορφή της σχεδίασης της μηχανής πεπερασμένων καταστάσεων (μηχανή τύπου Moore- η έξοδος διαφέρει στην κατάσταση ,όχι στην είσοδο) αποδίδει ολοκληρωτικά την απαιτούμενη κίνηση των καταστάσεων δεδομένων των εισόδων του. Κρατώντας το Reset η αρχική κατάσταση οδηγείται στο A(έξοδος 11111111). Με είσοδο IN1 και περιμένοντας 1 κύκλο ρολογιού(σύγχρονο κύκλωμα) πραγματοποιείται μετάβαση στην κατάσταση C(έξοδος 00111100 => προηγούμενη κατάσταση). Αναμένοντας την είσοδο IN2 παραμένουμε στην ίδια κατάσταση ενώ ξανά για είσοδο IN1 ξαναγίνεται μετάβαση σε προηγούμενη κατάσταση B(έξοδος 11000011) .Για είσοδο IN0 καταλήγουμε στην επόμενη κατάσταση (κατάσταση C με έξοδο 00111100). Όταν η συνθήκη διακοπής του ρολογιού γίνει αληθής παραμένουμε στην τελευταία κατάσταση.

Καταστάσεις. A=>B=>C=>A . IN2 ΠΑΡΑΜΟΝΉ ΣΕ ΚΑΤΆΣΤΑΣΗ

IN1 ΜΕΤΆΒΑΣΗ ΣΕ ΕΠΌΜΕΝΗ ΚΑΤΆΣΤΑΣΗ

IN0 ΜΕΤΆΒΑΣΗ ΣΕ ΠΡΟΗΓΟΎΜΕΝΗ ΚΑΤΆΣΤΑΣΗ

Συμπεράσματα

Η ολοκλήρωση της εργαστηριακής άσκησης οδηγεί στη διαμόρφωση διαφόρων παρατηρήσεων πάνω στη μελέτη και εξάσκηση στις εφαρμογές για προγραμματισμό υλικού(εντολές Xilinx) , εξοικείωση με υλοποίηση των συστημάτων αυτών , παρατηρήσεις

πάνω στην αλληλουχία λογισμικού/υλικού και στην προετοιμασία για εκτέλεση κυκλωμάτων,υπεύθυνων για πολύπλοκες διεργασίες.

Κώδικας

Single Pulse Generator

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

entity singlepulsegen is
Port ( clk      : in std_logic;      -- connect it to the Clock of the board
       rst      : in std_logic;      -- connect it to the Reset Button of the
                                        board
       input    : in std_logic;      -- connect it to the Push Button of the board
       output   : out std_logic      -- connect it to the input of your circuit
);
end singlepulsegen;

architecture behavioral of singlepulsegen is
type stateType is (S0, S1, S2);
signal currentS, nextS: stateType;

begin-- begin architecture

fsm_comb: process (currentS, input)
begin
```

```
                    case currentS IS
  when S0       =>    if input = '1' then     nextS <= S1;
                              else            nextS <= S0;
                          end if;
  when S1       =>
          nextS <= S2;
when S2                 =>    if input = '1' then     nextS <= S2;
                          else
                    nextS <= S0;
                              end if;
  when others   =>
  nextS <= S0;
                        end case;
                end process fsm_comb;

          fsm_synch: process (clk, rst)
                        begin
if (rst='1')                            then   currentS <= S0;
  elsif (rising_edge(clk))  then   currentS <= nextS;
                        end if;
                end process fsm_synch;

  output <= '1' when currentS = S1 else '0';

        end Behavioral;-- end architecture
```

FSM

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

entity Fsm is
Port ( RST : in  STD_LOGIC;
CLK : in  STD_LOGIC;
IN0 : in  STD_LOGIC;
IN1 : in  STD_LOGIC;
IN2 : in  STD_LOGIC;
OUTPUT : out  STD_LOGIC_VECTOR (7 downto 0));
end Fsm;

architecture Behavioral of Fsm is

TYPE STATE IS (A, B, C);
SIGNAL CURRENTS, NEXTS: STATE;
begin

FSM_COMB:PROCESS(CURRENTS, IN0, IN1, IN2)
BEGIN
CASE CURRENTS IS
WHEN A =>                OUTPUT<="11111111";
                                IF IN0 = '1' THEN NEXTS <= B;
```

```
                                        ELSIF IN1 = '1' THEN NEXTS <= C;
                                        ELSIF IN2='1' THEN NEXTS <= A;
                                              ELSE NEXTS<= A;
                                              END IF;
        WHEN B =>           OUTPUT<="11000011";
                                         IF IN0 = '1' THEN NEXTS <= C;
                                        ELSIF IN1 = '1' THEN NEXTS <= A;
                                        ELSIF IN2='1' THEN NEXTS <= B;
                                              ELSE NEXTS<= B;
                                              END IF;
        WHEN C =>           OUTPUT<="00111100";
                                         IF IN0 = '1' THEN NEXTS <= A;
                                        ELSIF IN1 = '1' THEN NEXTS <= B;
                                        ELSIF IN2='1' THEN NEXTS <= C;
                                              ELSE NEXTS<= C;
                                              END IF;

      WHEN OTHERS => OUTPUT <= "11111111";
                                                   NEXTS <= A;
            END CASE;
            END PROCESS;

     fsm_synch:PROCESS(CLK,RST)
                  Begin
       IF(RST='1') THEN CURRENTS<=A;
    ELSIF(rising_edge(CLK)) THEN CURRENTS<=NEXTS;
```

```
END IF;

END PROCESS;




end Behavioral;
```

TOP FSM

```
----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;



entity TopFsm is

Port ( RST : in  STD_LOGIC;

CLK : in  STD_LOGIC;

IN0 : in  STD_LOGIC;

IN1 : in  STD_LOGIC;

IN2 : in  STD_LOGIC;

LED : out  STD_LOGIC_VECTOR (7 downto 0));

end TopFsm;


architecture Behavioral of TopFsm is

SIGNAL pipe0,pipe1,pipe2:STD_LOGIC;

Component Fsm is
```

```
Port ( RST : in  STD_LOGIC;
       CLK : in  STD_LOGIC;
       IN0 : in  STD_LOGIC;
       IN1 : in  STD_LOGIC;
       IN2 : in  STD_LOGIC;
    OUTPUT : out  STD_LOGIC_VECTOR (7 downto 0));
end Component;

component singlepulsegen is
    Port ( clk    : in std_logic;      -- connect it to the Clock of the board
           rst    : in std_logic;      -- connect it to the Reset Button of the
                                          board
           input  : in std_logic;      -- connect it to the Push Button of the board
           output : out std_logic      -- connect it to the input of your circuit
                                       );
end component;

begin

PUSH0: singlepulsegen PORT MAP(clk=>CLK,rst=>RST,input=>IN0,output=>pipe0);
PUSH1: singlepulsegen PORT MAP(clk=>CLK,rst=>RST,input=>IN1,output=>pipe1);
PUSH2: singlepulsegen PORT MAP(clk=>CLK,rst=>RST,input=>IN2,output=>pipe2);
```

```
TOP_FSM:Fsm PORT
MAP(RST=>RST,CLK=>CLK,IN0=>pipe0,IN1=>pipe1,IN2=>pipe2,OUTPUT=>LED);


                                    end Behavioral;
```