ΠΡΟΧΩΡΗΜΕΝΗ ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ

ΑΝΑΦΟΡΆ 2ου ΕΡΓΑΣΤΗΡΙΟΥ

LAB 20335359

Γιώργος Βιριράκης 2016030035
Μιχάλης Γαλάνης 2016030036

1ο ΜΕΡΟΣ (Carry Look-Ahead Adder)

ΣΚΟΠΟΣ

Σκοπός του πρώτου μέρους της 2ης εργαστηριακής άσκησης είναι η περαιτέρω εξοικείωση με τη γλώσσα VHDL και την ιεραρχική σχεδίαση με πολλαπλά αρχεία. Συγκεκριμένα, θα σχεδιαστεί και υλοποιηθεί ένας full adder τεσσάρων bit με πρόβλεψη κρατουμένου (Carry look ahead).

ΠΡΟΕΡΓΑΣΙΑ

Ο πίνακας εισόδων/εξόδων του κυκλώματος Carry Look Ahead Adder δίνεται παρακάτω:

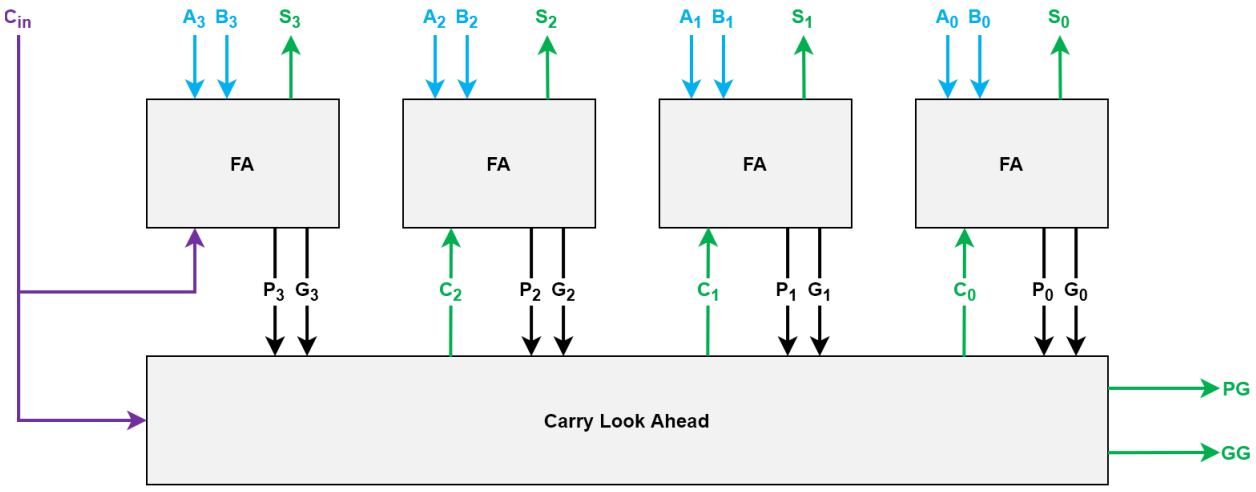
Name	IN / OUT	No. of Bits	Breadboard
A	in	4 (bus)	$SW_{[3:0]}$
B	in	4 (bus)	$SW_{[7:4]}$
C_{in}	in	1	BTN_0
S	out	4 (bus)	$LD_{[3:0]}$
C_3	out	1	LD_5

Οι εξισώσεις που υλοποιούμε είναι οι ακόλουθες και περιγράφουν τη λειτουργία του τελικού κυκλώματος:

$P_i = A_i \oplus B_i$
$G_i = A_i * B_i$

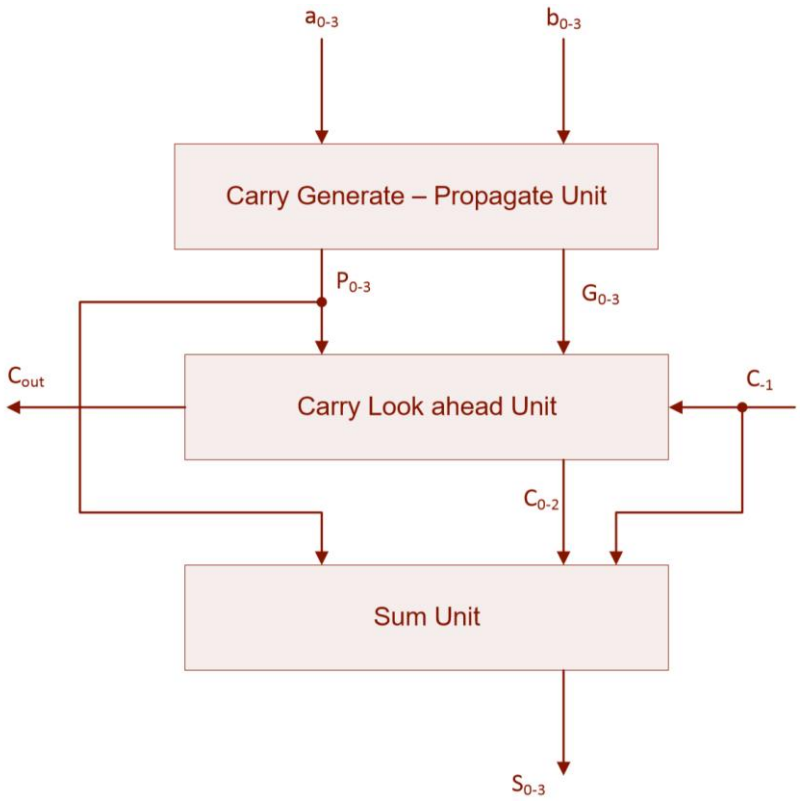
$C_0 = G_0 + (P_0 * C_{in})$
$C_1 = G_1 + (P_1 * G_0) + (P_1 * P_0 * C_{in})$
$C_2 = G_2 + (P_2 * G_1) + (P_2 * P_1 * G_0) + (P_2 * P_1 * P_0 * C_{in})$
$C_3 = G_3 + (P_3 * G_2) + (P_3 * P_2 * G_1) + (P_3 * P_2 * P_1 * G_0) + (P_3 * P_2 * P_1 * P_0 * C_{in})$

$S_0 = A_0 \oplus B_0 \oplus C_{in}$
$S_i = A_i \oplus B_i \oplus C_{i-1}$,for i>0



ΠΕΡΙΓΡΑΦΗ

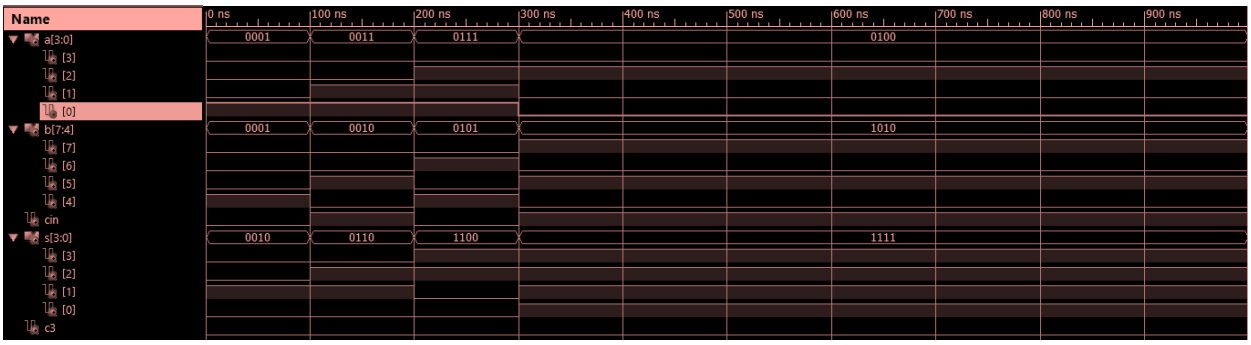
Η άσκηση απαιτούσε ιεραρχική σχεδίαση, οπότε εαν παρατηρήσουμε τον Carry Look Ahead Adder από μια άλλη οπτική γωνία τότε αμέσως βλέπουμε ότι το κύκλωμα **χωρίζεται σε μικρότερα τμήματα (components).** Άρα η κατασκευή του κυκλώματος στο Xilinx αποτελείται από 2 επίπεδα, στο χαμηλό επίπεδο **(behavioral)** περιγράφεται το κάθε **unit** ξεχωριστά (Carry Generate – Propagate Unit, Carry Look Ahead Unit, Sum Unit) και στο υψηλό επίπεδο **(structural)** έχουμε τον Carry Look Ahead Adder (συνολικό κύκλωμα).



Να σημειωθεί ότι το implementation στο **topModule** περιλαμβάνει **(3)** ενδιάμεσα εσωτερικά σήματα **P_signal, C_signal, G_signal.**

ΚΥΜΑΤΟΜΟΡΦΕΣ

Προχωρώντας στην προσομοίωση, μερικές δοκιμαστικές πράξεις αρκούν στο **testbench** για την επιβεβαίωση της ορθότητας του κυκλώματος και τελικά παράχθηκαν οι παρακακάτω κυματομορφές:



2ο ΜΕΡΟΣ (FSM)

ΣΚΟΠΟΣ

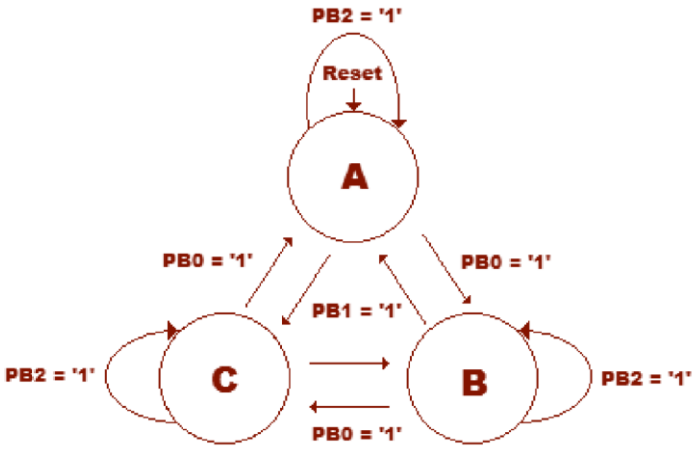
Σκοπός στο δεύτερο μέρος ήταν η πρώτη μας γνωριμία με τις FSM στη γλώσσα VHDL. Θα μάθουμε επίσης να εισάγουμε και σήμα ρολογιού (**CLK**) καθώς τώρα εισερχόμαστε σε σύγχρονα κυκλώματα.

ΠΡΟΕΡΓΑΣΙΑ

Αυτή τη φορά το κύκλωμά διαθέτει **4 εισόδους** (4 buttons) και **8 εξόδους** (8 LEDs – 1 bus πλάτους 8 bits).

Name	IN / OUT	No. of Bits	Breadboard
RST	in	1	BTN_3
IN_0	in	1	BTN_0
IN_1	in	1	BTN_1
IN_2	in	1	BTN_2
LED	out	8 (bus)	$LD_{[7:0]}$

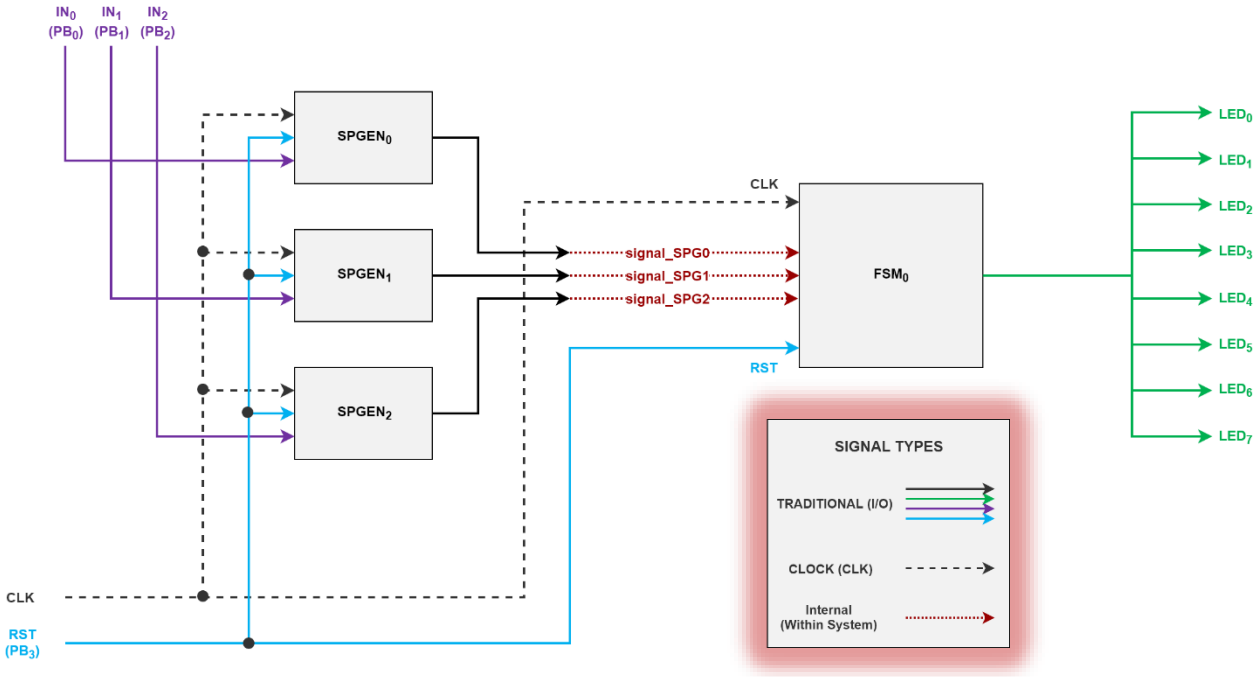
Με αυτό το I/O θέλουμε να κατασκευάσουμε μια FSM που περιγράφεται από το παρακάτω σχήμα. Να σημειωθεί ότι οι μεταβλητές $\mathbf{PB_x}$ σημαίνουν **push buttons** και το καθένα ισοδυναμεί με το αντίστοιχο $\mathbf{IN_x}$.



Έπειτα κατασκευάσαμε τον πίνακα καταστάσεων σύμφωνα με το διάγραμμα αλλά και με τις προϋποθέσεις της εκφώνησης:

Current State	Next State			Output
	$PB_0 = 1$	$PB_1 = 1$	$PB_2 = 1$	
A	B	C	A	1111_1111
B	C	A	B	1111_0000
C	A	B	C	0000_1111

Παρακάτω φαίνεται το ολοκληρωμένο κύκλωμα σχεδιασμένο στο draw.io σύμφωνα με τον κώδικα μας:



ΠΕΡΙΓΡΑΦΗ

Η παραπάνω μηχανή πεπερασμένων καταστάσεων είναι ξεκάθαρα τύπου **Moore**, καθώς οι εξόδους της σε κάθε κατάσταση εξαρτώνται μόνο από την τρέχουσα κατάσταση και όχι από τις εισόδους.

Το πρόγραμμα Xilinx έχει native support για κατασκευή τέτοιων μηχανών, οπότε ο σχεδιαστής / προγραμματιστής **ασχολείται μόνο με τη διαχείρηση καταστάσεων**. Η διαχείρηση καταστάσεων επιτυγχάνεται με τις εντολές **CASE / WHEN X** και με εντολές **IF / ELSE** καταγράφουμε τις κατάλληλες συνθήκες για να μεταβαίνουν ορθά οι καταστάσεις στη μηχανή.

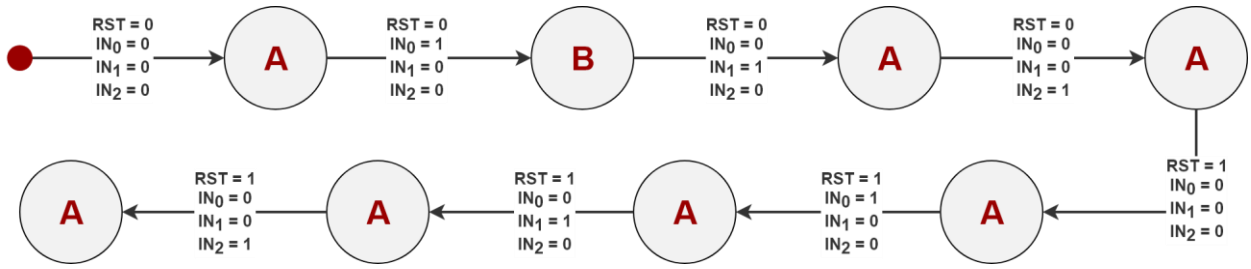
Όμως παρατηρούμε ότι για αυτή τη μηχανή πεπερασμένων καταστάσεων, έχουμε μεμονωμένους παλμούς ρολογιού, με άλλα λόγια, οι ίδιες οι εισόδους **(IN_0, IN_1, IN_2)** με τη βοήθεια του βοηθητικού module **singlepulsegen** παράγουν ένα παλμό ρολογιού από ένα ρολόι αυθαίρετων κύκλων.

Σημειώνουμε σε αυτό το σημείο ότι και σε αυτό το μέρος υπάρχει ιεραρχία μεταξύ των modules. Στο χαμηλότερο επίπεδο (behavioral) έχουμε την κατασκευή των singlepulsegen και της FSM, και στο υψηλό επίπεδο (structural – topLevel) τα συνδέουμε για να κατασκευάσουμε το τελικό μας κύκλωμα **"TOP_FSM".** Χρησιμοποιούμε επίσης 3 εσωτερικά σήματα για να καταλήξουμε τις εξόδους των singlepulsegen στις εισόδους της FSM.

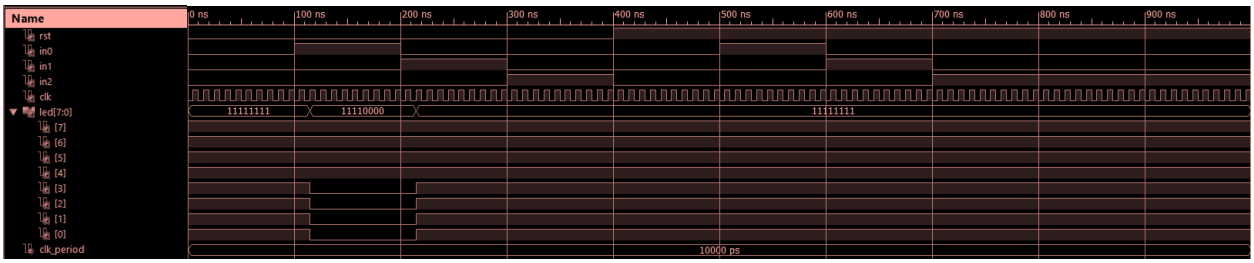
ΚΥΜΑΤΟΜΟΡΦΕΣ

Δημιουργήσαμε έπειτα το **testbench** για το **"TOP_FSM".** Συμπεριλάβαμε και το ρολόι με **50% duty cycle** ($1^η$ ημιπερίοδος -> 0, $2^η$ ημιπερίοδος -> 1) και περίοδο **10 ns** (συχνότητα 100MHz) για να είναι συμβατό με μια από τις 3 διαθέσιμες συχνότητες του **LOC B8 (mCLK).**

Για να επαληθεύσουμε την ορθότητα του κυκλώματος επιλέξαμε την παρακάτω διαδρομή καταστάσεων:



Με την εκτέλεση της προσομοίωσης του μοντέλου, παράγονται οι παρακάτω κυματομορφές οι οποίες συμφωνούν με τα αποτελέσματα που περιμέναμε:



Παρατηρώντας τις κυματομορφές, βλέπουμε μια καθυστέρηση του σήματος εξόδου κατά ένα κύκλο ρολογιού. Αυτό είναι φυσιολογικό, καθώς η μηχανή πεπερασμένων καταστάσεων χρησιμοποιεί καταχωρητές (flip flops συγκεκριμένα) για να θυμάται την κάθε κατάσταση. Οι καταχωρητές αυτοί καθυστερούν την έξοδο και έτσι, η αλλαγή της κατάστασης πραγματοποιείται ένα κύκλο αργότερα.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Με την ολοκλήρωση αυτής της εργαστηριακής άσκησης, εξοικειωθήκαμε περαιτέρω με την ιεραρχική σχεδίαση (1ο μέρος) και κάναμε μια πρώτη επαφή με τις μηχανές πεπερασμένων καταστάσεων (FSM).

ΠΑΡΑΡΤΗΜΑ – ΚΩΔΙΚΑΣ

1ο ΜΕΡΟΣ

CLAU

```
architecture Behavioral of Carry_Look_Ahead_Unit is

begin

C(0) <= G(0) OR (P(0) AND Cin);
C(1) <= G(1) OR (P(1) AND G(0)) OR (P(1) AND P(0) AND Cin);
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);
C(3) <= 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 G(0)) OR (P(3) AND P(2) AND P(1) AND P(0) AND Cin);


end Behavioral;
```

CGPU

```
architecture Behavioral of Carry_Generate_Propagate_Unit is

begin

P(0) <= A(0) XOR B(0);
P(1) <= A(1) XOR B(1);
P(2) <= A(2) XOR B(2);
P(3) <= A(3) XOR B(3);

G(0) <= A(0) AND B(0);
G(1) <= A(1) AND B(1);
G(2) <= A(2) AND B(2);
G(3) <= A(3) AND B(3);

end Behavioral;
```

SU

```
architecture Behavioral of Sum_Unit is

begin

S(0) <= P(0) XOR Cin;
S(1) <= P(1) XOR C(0);
S(2) <= P(2) XOR C(1);
S(3) <= P(3) XOR C(2);

end Behavioral;
```

CLAA (TOP Module)

```
signal P_SIGNAL : STD_LOGIC_VECTOR (3 downto 0);
signal G_SIGNAL : STD_LOGIC_VECTOR (3 downto 0);
signal C_SIGNAL : STD_LOGIC_VECTOR (2 downto 0);
```

```
begin
  CGPU: Carry_Generate_Propagate_Unit port map(
                                              --Inputs
                                              A(0) => A(0),
                                              A(1) => A(1),
                                              A(2) => A(2),
                                              A(3) => A(3),

                                              B(0) => B(4),
                                              B(1) => B(5),
                                              B(2) => B(6),
                                              B(3) => B(7),

                                              --Outputs
                                              P(0) => P_SIGNAL(0),
                                              P(1) => P_SIGNAL(1),
                                              P(2) => P_SIGNAL(2),
                                              P(3) => P_SIGNAL(3),

                                              G(0) => G_SIGNAL(0),
                                              G(1) => G_SIGNAL(1),
                                              G(2) => G_SIGNAL(2),
                                              G(3) => G_SIGNAL(3)
                                                );
```

```
CLAU: Carry_Look_Ahead_Unit port map (
                                                    --Inputs
                                                    P(0) => P_SIGNAL(0),
                                                    P(1) => P_SIGNAL(1),
                                                    P(2) => P_SIGNAL(2),
                                                    P(3) => P_SIGNAL(3),

                                                    G(0) => G_SIGNAL(0),
                                                    G(1) => G_SIGNAL(1),
                                                    G(2) => G_SIGNAL(2),
                                                    G(3) => G_SIGNAL(3),

                                                    --Outputs
                                                    Cin => Cin,

                                                    C(0) => C_SIGNAL(0),
                                                    C(1) => C_SIGNAL(1),
                                                    C(2) => C_SIGNAL(2),

                                                    C(3) => C3
                                                     );
```

```
SU: Sum_Unit port map (
                                                --Inputs
                                                P(0) => P_SIGNAL(0),
                                                P(1) => P_SIGNAL(1),
                                                P(2) => P_SIGNAL(2),
                                                P(3) => P_SIGNAL(3),

                                                Cin => Cin,

                                                C(0) => C_SIGNAL(0),
                                                C(1) => C_SIGNAL(1),
                                                C(2) => C_SIGNAL(2),

                                                --Ouputs
                                                S(0) => S(0),
                                                S(1) => S(1),
                                                S(2) => S(2),
                                                S(3) => S(3)
                                                );
```

```
end Structural;
```

2ο ΜΕΡΟΣ

```
architecture Structural of TOP_FSM is

signal signal_SPG0, signal_SPG1, signal_SPG2 : STD_LOGIC;
Component singlepulsegen is
Port ( clk     : in std_logic;
       rst     : in std_logic;
       input   : in std_logic;
       output  : out std_logic
      );
end Component;


Component FSM is
Port ( RST : in  STD_LOGIC;
       IN0 : in  STD_LOGIC;
       IN1 : in  STD_LOGIC;
       IN2 : in  STD_LOGIC;
       CLK : in  STD_LOGIC;
       LED : out  STD_LOGIC_VECTOR(7 downto 0)
     );
end Component;


begin

SPGEN0: singlepulsegen port map( clk => CLK,
                                 rst => RST,
                                 input => IN0,
                                 output => signal_SPG0
                               );

SPGEN1: singlepulsegen port map( clk => CLK,
                                 rst => RST,
                                 input => IN1,
                                 output => signal_SPG1
                               );

SPGEN2: singlepulsegen port map( clk => CLK,
                                 rst => RST,
                                 input => IN2,
                                 output => signal_SPG2
                               );

FSM0: FSM port map ( CLK => CLK,
                     RST => RST,
                     IN0 => signal_SPG0,
                     IN1 => signal_SPG1,
                     IN2 => signal_SPG2,
                     LED => LED
                     );

end Structural;
```