

Εργαστήριο Σχεδιασμού Ολοκληρωμένων Κυκλωμάτων II Ακαδημαϊκό Έτος 2018-2019



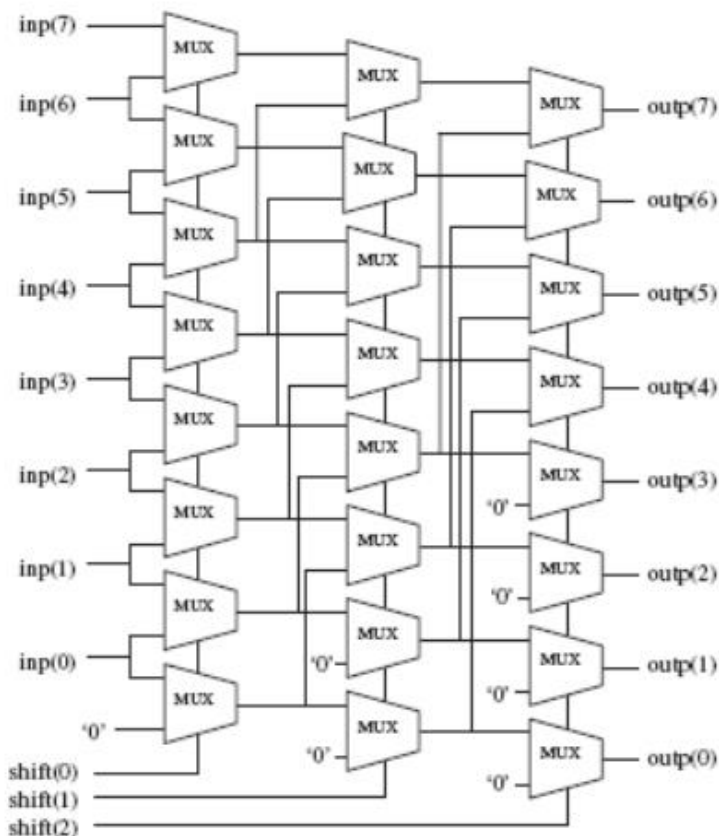
6^η Εργαστηριακή Άσκηση σε VHDL

Ομάδα 16

ΟΝΟΜΑΤΑ ΦΟΙΤΗΤΩΝ	A.M.
ΜΑΤΑΡΑΓΚΑΣ ΜΙΛΤΙΑΔΗΣ	1046865
ΑΛΜΠΙΑΝΗΣ ΙΩΑΝΝΗΣ	1018982

1. Σχεδίαση και υλοποίηση ενός 8 bit barrel shifter

Το δομικό διάγραμμα του barrel shifter φαίνεται παρακάτω:



Η είσοδος shift είναι αυτή που ορίζει κατά πόσα bits θα ολισθήσει η λέξη εισόδου. Η καθυστέρηση διάδοσης του σήματος εισόδου μέχρι την έξοδο είναι ίση τρεις τρεις φορές την καθυστέρηση ενός mux. Ο κώδικας που περιγράφει τον mux είναι ο παρακάτω:

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

entity mux is
    port ( in0,in1,s : in std_logic;
           o       : out std_logic);
end mux;

architecture Structural of mux is
begin
    o <= in0 when s = '0' else
        in1;
end Structural;
```

Με χρήση της εντολής generate κάνουμε τις κατάλληλες διασυνδέσεις μεταξύ των πολυπλεκτών και προκύπτει το επιθυμητό κύκλωμα. Οι διασυνδέσεις περιγράφονται παρακάτω:

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

entity Barrel_shifter is
    Port ( input : in STD_LOGIC_VECTOR (7 downto 0);
          shift : in STD_LOGIC_VECTOR (2 downto 0);
          output : out STD_LOGIC_VECTOR (7 downto 0));
end Barrel_shifter;

architecture Structural of Barrel_shifter is
    component mux
        port ( in0,in1,s : in std_logic;
              o : out std_logic);
    end component mux;

    signal t1 : STD_LOGIC_VECTOR (7 downto 0);
    signal t2 : STD_LOGIC_VECTOR (7 downto 0);

begin
    E1: mux port map (in0 => input(7) , in1 => '0' , s => shift(0) , o => t1(7));
    G1: for i in 7 downto 1 generate
        E2: mux port map (in0 => input(i-1) , in1 => input(i) , s => shift(0) , o => t1(i-1));
    end generate G1;




    E3: mux port map (in0 => t1(7) , in1 => '0' , s => shift(1) , o => t2(7));
    E4: mux port map (in0 => t1(6) , in1 => '0' , s => shift(1) , o => t2(6));
    G2: for i in 7 downto 2 generate
        E5: mux port map (in0 => t1(i-2) , in1 => t1(i) , s => shift(1) , o => t2(i-2));
    end generate G2;

    G3: for i in 7 downto 4 generate
        E6: mux port map (in0 => t2(i) , in1 => '0' , s => shift(2) , o => output(i));
    end generate G3;

    G4: for i in 7 downto 4 generate
        E8: mux port map (in0 => t2(i-4) , in1 => t2(i) , s => shift(2) , o => output(i-4));
    end generate G4;

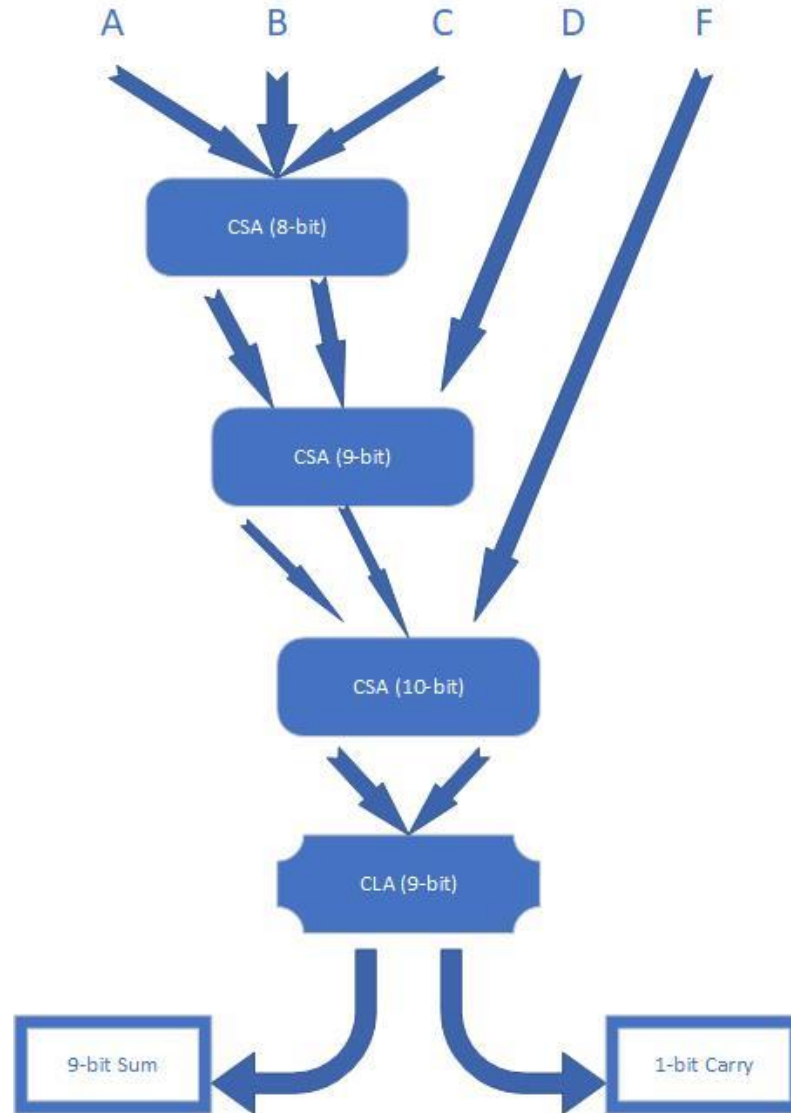
end Structural;
```

Στο επόμενο διαγράμματο διαπιστώνεται η ορθή λειτουργία του κυκλώματος:

 /barrel_shifter/input	-No Data-	8'b00110101			8'b01000011		
 /barrel_shifter/shift	-No Data-	3'b001	3'b011			3'b001	
 /barrel_shifter/output	-No Data-	8'b00011010	8'b00000110		8'b00001000	8'b00100001	

2. Σχεδίαση και υλοποίηση ενός αθροιστή που αποτελείται από CSAs και CLAs (Carry Look-Ahead Adders).

Η λογική που ακολουθήθηκε είναι αυτή του παρακάτω λογικού διαγράμματος:



Όπου κάθε block αποτελείται από τον αντίστοιχο αριθμό CSA ή CLA Blocks του ενός bit.

Ο κώδικας που περιγράφει το κάθε block δίνεται πιο κάτω.

```
-- 31/05/2019
-- Carry Select Adder (Full adder but the input Carry is now the 3rd word's bit)
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity CSA_Block is
    port (x,y,z: in std_logic;
          s,c: out std_logic);
end CSA_Block;
architecture CSA of CSA_Block is
begin
    s <= (x xor y) xor z;
    c <= ((x xor y) and z) or (x and y);
end CSA;
```

```
-- 01/06/2019
-- Carry Look-Ahead Adder
library ieee;
use ieee.std_logic_1164.all;

entity CLA_Block is
    port (a,b,Cin: in std_logic;
          p,g,sum,cout: out std_logic);
end CLA_Block;
architecture CLA of CLA_Block is
    signal prop, gen:std_logic;
begin
-- Carry_Propagate
    prop <= a xor b;
    p <= prop;
-- Carry_Generate
    gen <= a and b;
    g <= gen;
-- Sum Output
    sum <= prop xor Cin;
-- Carry Output
    cout <= (prop and Cin) or gen;
end CLA;
```

Ας σημειωθεί πως ο Carry Look Ahead Adder μεταδίδει με υψηλή ταχύτητα τον υπολογισμό του κρατουμένου καθώς δεν απαιτείται ο υπολογισμός των προηγούμενων carry για την εύρεση του τελικού. Με την προσομοίωση επαληθεύουμε την ορθή λειτουργία του κυκλώματός μας

+ /fiveword8bitcsa/a	-No Data-	8hFF	8h01			8h1D	
+ /fiveword8bitcsa/b	-No Data-	8hFF	8h01	8h03			
+ /fiveword8bitcsa/c	-No Data-	8hFF	8h01				8hAC
+ /fiveword8bitcsa/d	-No Data-	8hFF	8h01		8h07		
+ /fiveword8bitcsa/e	-No Data-	8hFF	8h01			8h09	
+ /fiveword8bitcsa/sum	-No Data-	9h0FB	9h005	9h007	9h00D	9h015	9h031
+ /fiveword8bitcsa/Cout	-No Data-						9h0DC