24/3/2017

2η Εργαστηριακή Άσκηση
ΕΞΟΙΚΕΙΩΣΗ ΜΕ ΤΗ ΓΛΩΣΣΑ ΠΕΡΙΓΡΑΦΗΣ ΥΛΙΚΟΥ VHDL ΚΑΙ ΤΗΝ ΙΕΡΑΡΧΙΚΗ ΣΧΕΔΙΑΣΗ (STRUCTURAL VHDL)
3/3/2017

Ομάδα LAB20332061

ΛΑΜΠΡΙΑΝΗ ΤΣΑΠΑΝΟΥ 2014030015
ΑΝΑΣΤΑΣΙΟΣ ΜΠΟΚΑΛΙΔΗΣ 2014030069

Σκοπός εργαστηριακής άσκησης

Σκοπός της 2ης εργαστηριακής άσκησης ήταν η περαιτέρω εξοικείωση με την γλώσσα VHDL, μέσω της σχεδίασης ενός 4-bit Carry Look Ahead Adder και μίας μηχανής πεπερασμένων καταστάσεων (FSM).

1ο Κύκλωμα

Προεργασία – Περιγραφή

Στο κύκλωμα αυτό μας έχει ζητηθεί να κατασκευάσουμε έναν 4-bit Carry Look Ahead Adder με ιεραρχική σχεδίαση.

Ο adder των τεσσάρων bit αποτελείται από τρεις διαφορετικές μονάδες που θα συνδέονται κατάλληλα μεταξύ τους:

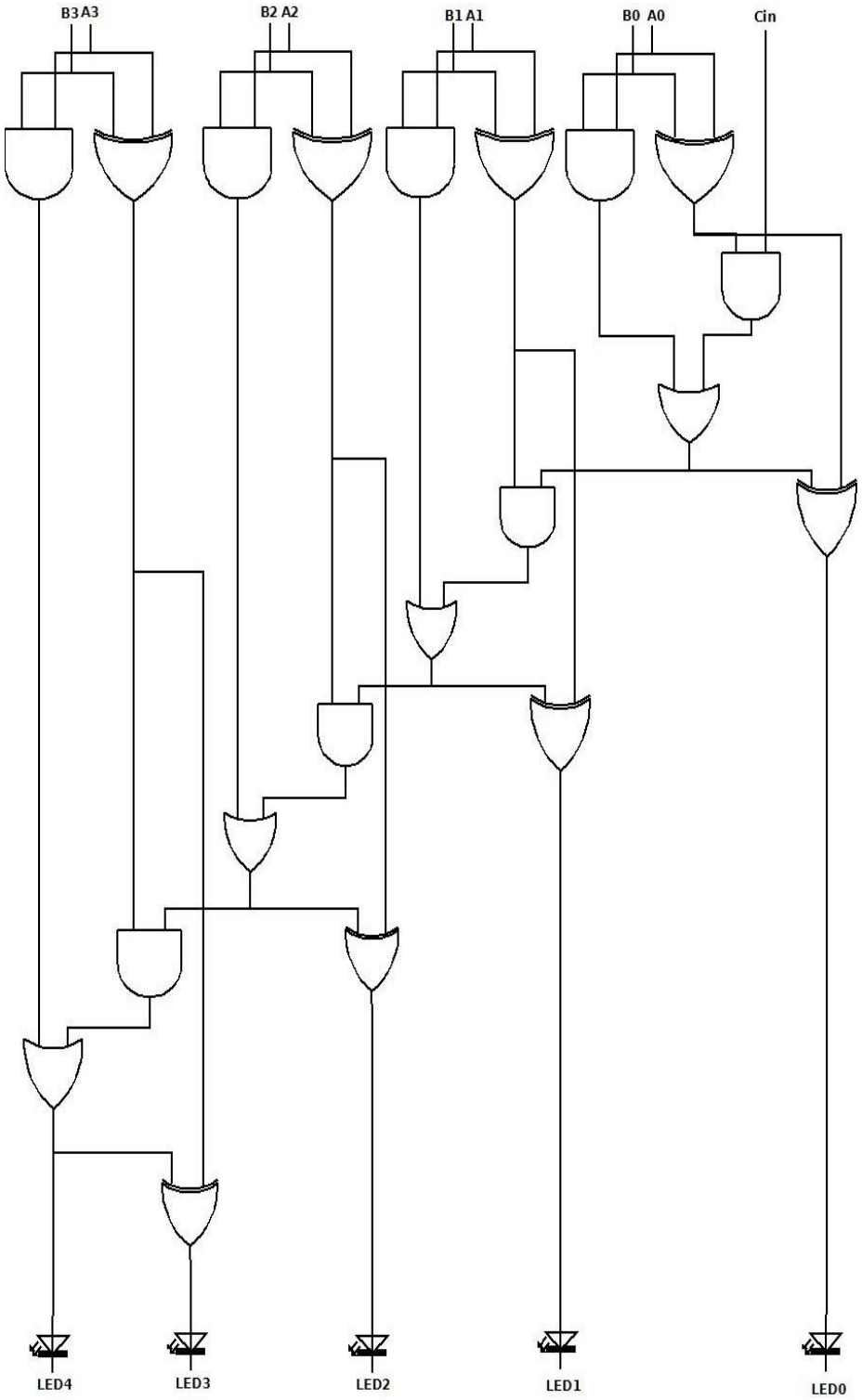
- **Carry Generate/Propagate Unit**
  Εξισώσειςπουθαυλοποιηθούν:
  Pi= Ai XOR Bi (Propagate)
  Gi= Ai AND Bi (Generate)

- **Carry LookAheadUnit**
  Φέρνουμε τις εξισώσεις που μας δώθηκαν σε μια απλούστερη μορφή:
  C0=G0+P0*Cin
  C1=G1+P1*G0+P1*P0*Cin =>C1= G1+P1*C0
  C2=G2+P2*G1+P2*P1*G0+P2*P1*P0*Cin =>C2=G2+P2*C1
  C3=G3+P3*G2+P3*P2*G1+P3*P2*P1*G0+P3*P2*P1*P0*Cin =>C3=G3+P3*C2

- **Sum Unit**
  Εξισώσεις που θα υλοποιηθούν:
  Si = A XOR B XOR C (Sum)

Διάγραμμα

Καταλήγουμε στο παρακάτω διάγραμμα:



Κυματομορφές-Προσομοίωση

Για το 1^{ο} κύκλωμα δημιουργήσαμε 4 παραδείγματα εισόδων:

- Πρόσθεση **5+3 με Cin=0** και αποτέλεσμα το 8 και Cout=0 (ή 0101+0011=1000)
- Πρόσθεση **9+4 με Cin=1** και αποτέλεσμα το 14 και Cout=0 (ή 1001+0100=1110)
- Πρόσθεση **2+13 με Cin=0** και αποτέλεσμα το 15 και Cout=0 (ή 0010+1101=1111)
- Πρόσθεση **7+13 με Cin=0** και αποτέλεσμα το 20 και Cout=1 (ή 0111+1101=10100)

Κώδικας

Carry Generate/Propagate Unit (CPG):

```
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
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 Behavioral 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 Behavioral;
```

Carry Look Ahead Unit (CLA):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity CLA is
    Port ( P : in  STD_LOGIC_VECTOR (3 downto 0);
           G : in  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           C : out  STD_LOGIC_VECTOR (2 downto 0);
           Cout : out  STD_LOGIC);
end CLA;

architecture Behavioral of CLA 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);
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);

end Behavioral;
```

Sum Unit (SUM):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SUM is
    Port ( P : in  STD_LOGIC_VECTOR (3 downto 0);
           C : in  STD_LOGIC_VECTOR (2 downto 0);
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC_VECTOR (3 downto 0));
end SUM;

architecture Behavioral of SUM 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;


```

4-bit Carry Look Ahead Adder:

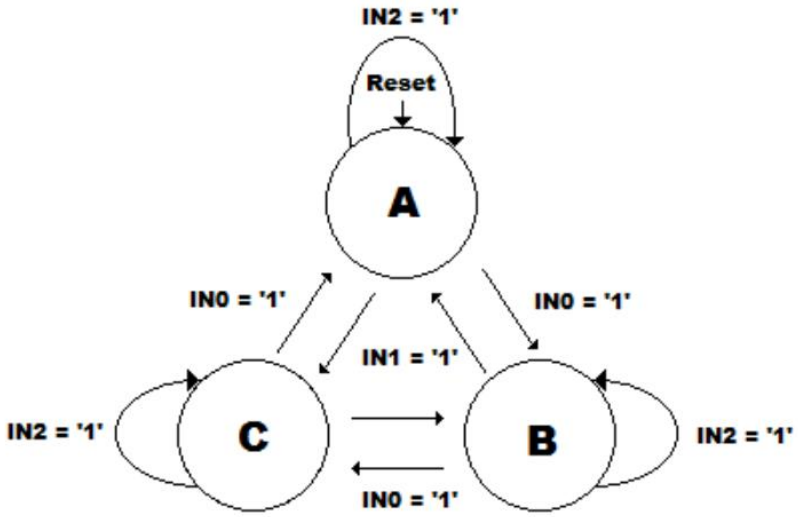
```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity CLA_4bit_ADDER is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           B : in  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC_VECTOR (3 downto 0);
           C3 : out  STD_LOGIC);
end CLA_4bit_ADDER;

architecture Behavioral of CLA_4bit_ADDER is

component CPG
Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           B : in  STD_LOGIC_VECTOR (3 downto 0);
           P : out  STD_LOGIC_VECTOR (3 downto 0);
           G : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

component CLA
Port ( P : in  STD_LOGIC_VECTOR (3 downto 0);
           G : in  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           C : out  STD_LOGIC_VECTOR (2 downto 0);
           Cout : out  STD_LOGIC);
end component;

component SUM
Port ( P : in  STD_LOGIC_VECTOR (3 downto 0);
           C : in  STD_LOGIC_VECTOR (2 downto 0);
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

signal G_signal:std_logic_vector(3 downto 0);
signal P_signal:std_logic_vector(3 downto 0);
signal C_signal:std_logic_vector(2 downto 0);

begin
CPG_unit: CPG
port map( A=>A,
          B=>B,
          P=>P_signal,
          G=>G_signal);

CLA_unit: CLA
port map( P=>P_signal,
          G=>G_signal,
          Cin=>Cin,
          C=>C_signal,
          Cout=>C3);

SUM_unit: SUM
port map( P=>P_signal,
          C=>C_signal,
          Cin=>Cin,
          S=>S);

end Behavioral;
```

2ο Κύκλωμα

Προεργασία – Περιγραφή

Στο δεύτερο κύκλωμα μα ζητήθηκε μια FSM, 3 καταστάσεων (A, B, C). Σε κάθε μια κατάσταση θα ανάβουν διαφορετικά LEDs με αποτέλεσμα να αντιλαμβανόμαστε σε ποια κατάσταση βρισκόμαστε. Για την υλοποίηση του προγράμματος χρειαστήκαμε διάφορες εντολές όπως IF-THEN ELSE,PROCESS.

Οι καταστάσεις της FSM που μας ζητήθηκε φαίνονται παρακάτω:



Πίνακας 4: Σχηματική παρουσίαση της FSM

ΚΑΤΑΣΤΑΣΗ/ΕΙΣΟΔΟΙ	IN0=1	IN1=1	IN2=1	RESET=1
A	*B*	*C*	*A*	*A*
B	*C*	*A*	*B*	*A*
C	*A*	*B*	*C*	*A*

Έξοδοι ανά κατάσταση (LEDs):

A = ”11111111”

B = ”11000011”

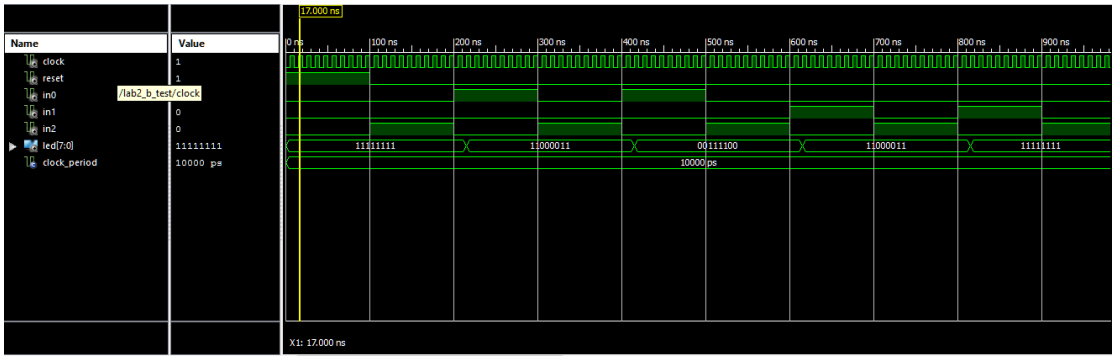
C = ”00111100”

Στην αρχή του προγράμματος κάνουμε RESET ώστε να είμαστε σίγουροι ότι ξεκινάμε από την κατάσταση A.

Κυματομορφές-Προσομοίωση

Για το Test bench του 2ου κυκλώματος εφαρμόσαμε τις εξής εναλλαγές:
Αρχικά RESET οπότε αρχικοποιούμε στην κατάσταση Α. Έπειτα:

- IN2=1 (μένει κατάσταση Α)
- IN0=1 (πάει κατάσταση Β)
- IN2=1 (μένει κατάσταση Β)
- IN0=1 (πάει κατάσταση C)
- IN2=1 (μένει κατάσταση C)
- IN1=1 (πάει κατάσταση Β)
- IN2=1 (μένει κατάσταση Β)
- IN1=1 (πάει κατάσταση Α)
- IN2=1 (μένει κατάσταση Α)



Παρατηρούμε ότι οι έξοδοι συμπίπτουν με την κατάσταση στην οποία πρέπει να βρίσκεται.

Κώδικας

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.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;
           LED : out  STD_LOGIC_VECTOR (7 downto 0));
end fsm;

architecture Behavioral of fsm is

TYPE State IS (A,B,C);
Signal fsm_state : State;

begin

state_change : process
begin

wait until CLK' event and CLK='1';

if RST='1' then fsm_state <= A;
else
case fsm_state is
    when A => if (IN0='1') then fsm_state <= B;
              elsif (IN1='1') then fsm_state <= C;
              elsif (IN2='1') then fsm_state <= A;
              end if;
    when B => if (IN0='1') then fsm_state <= C;
              elsif (IN1='1') then fsm_state <= A;
              elsif (IN2='1') then fsm_state <= B;
              end if;
    when C => if (IN0='1') then fsm_state <= A;
              elsif (IN1='1') then fsm_state <= B;
              elsif (IN2='1') then fsm_state <= C;
              end if;
end case;
end if;
end process state_change;

state_output : process (fsm_state)
begin
     case fsm_state is
     when A => LED <= "11111111";
     when B => LED <= "11000011";
     when C => LED <= "00111100";
end case;
end process state_output;

end Behavioral;


```

Total FSM:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FINAL_FSM is
    Port ( CLOCK : in  STD_LOGIC;
           RESET : in  STD_LOGIC;
           IN0 : in  STD_LOGIC;
           IN1 : in  STD_LOGIC;
           IN2 : in  STD_LOGIC;
           LED : out  STD_LOGIC_VECTOR (7 downto 0));
end FINAL_FSM;

architecture Behavioral of FINAL_FSM is

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;
           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 component;

signal signal_1 : std_logic;
signal signal_2 : std_logic;
signal signal_3 : std_logic;

begin

singlepulsegen_1_instance: singlepulsegen
port map(clk => CLOCK,
         rst => RESET,
         input => IN0,
         output => signal_1);

singlepulsegen_2_instance: singlepulsegen
port map(clk => CLOCK,
         rst => RESET,
         input => IN1,
         output => signal_2);

singlepulsegen_3_instance: singlepulsegen
port map(clk => CLOCK,
         rst => RESET,
         input => IN2,
         output => signal_3);

fsm_instance : fsm
port map(RST => RESET,
         CLK => CLOCK,
         IN0 => signal_1,
         IN1 => signal_2,
         IN2 => signal_3,
         LED => LED);
```

Συμπεράσματα

Καταλήγοντας, σε αυτό το εργαστήριο κατανοήσαμε την λειτουργία της FSM και τον τρόπο με τον οποίο αλλάζουν οι καταστάσεις χρησιμοποιώντας την case. Επίσης χρησιμοποιήσαμε τον έτοιμο κώδικα που μας δώθηκε για Single Pulse Generator και τον αξιοποιήσαμε μέσω των instances που δημιουργήσαμε.