Εργαστήριο 1

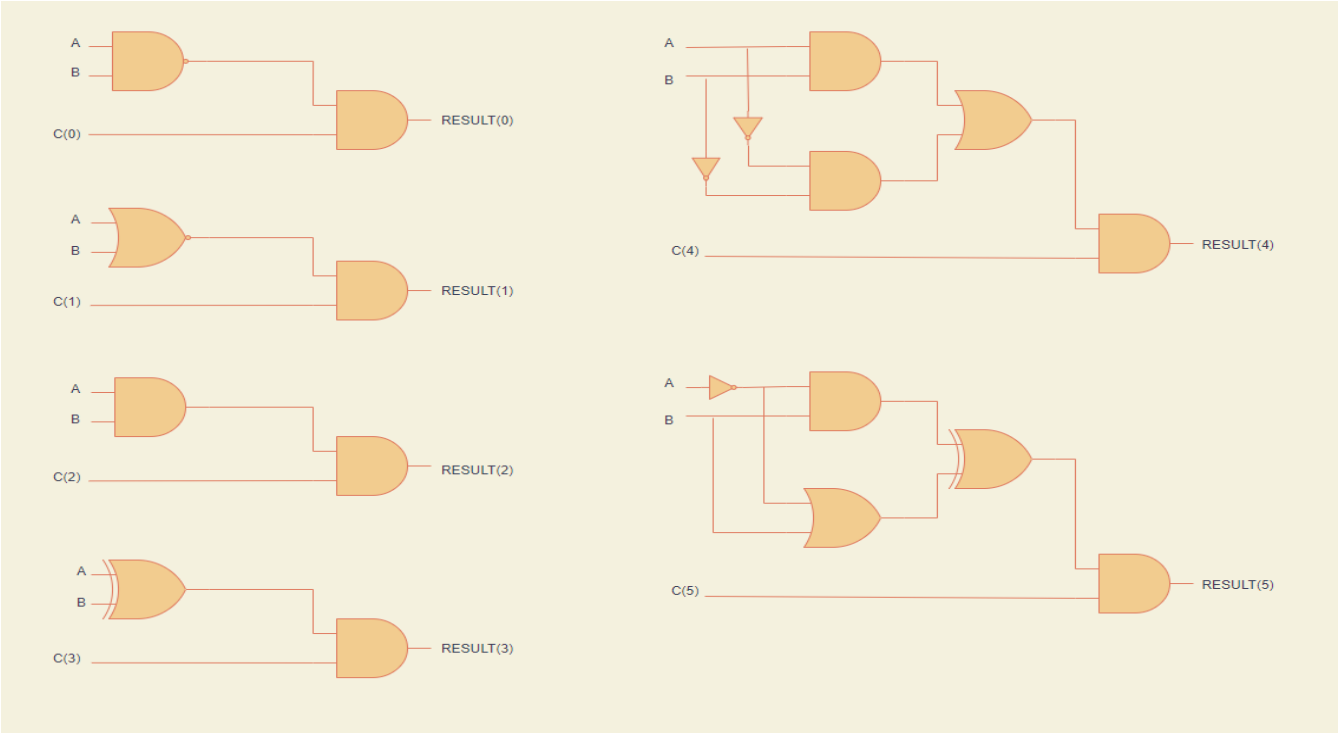
ΕΞΟΙΚΕΙΩΣΗ ΜΕ ΤΗ ΓΛΩΣΣΑ ΠΕΡΙΓΡΑΦΗΣ ΥΛΙΚΟΥ VHDL ΚΑΙ ΤΗΝ ΙΕΡΑΡΧΙΚΗ ΣΧΕΔΙΑΣΗ

Μέρος Α - Κύκλωμα 1

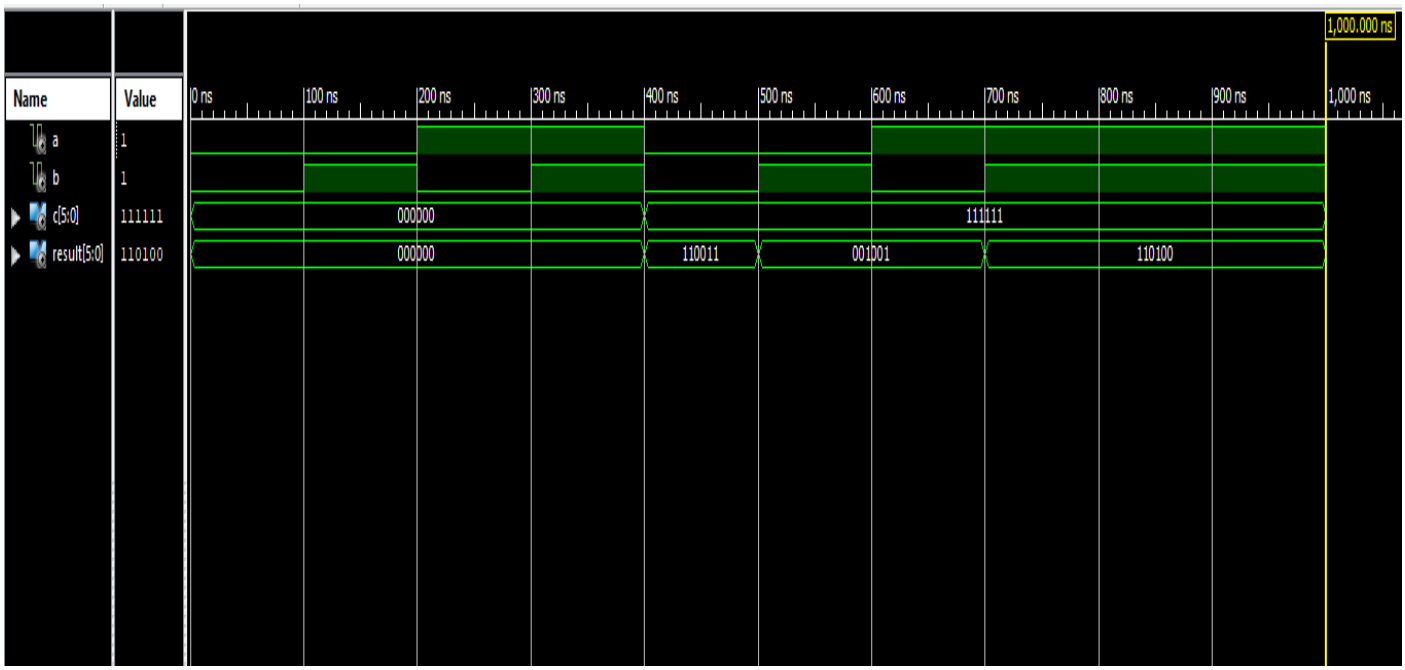
Πίνακας Αληθείας

Outputs								Outputs					
A	B	C(0)	C(1)	C(2)	C(3)	C(4)	C(5)	RESULT(0)	RESULT(1)	RESULT(2)	RESULT(3)	RESULT(4)	RESULT(5)
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0	1	1
0	1	1	1	1	1	1	1	1	0	0	1	0	0
1	0	1	1	1	1	1	1	1	0	0	1	0	0
1	1	1	1	1	1	1	1	0	0	1	0	1	1

Κυκλώματα



Simulation



Η προσομοίωση είναι εξαντλητική, καθώς εξετάζει όλες τις περιπτώσεις εισόδων που αφορούν την κάθε έξοδο. Δηλαδή, εξετάζουμε την περίπτωση για το vector C = “000000”, για όλες τις πιθανές τιμές εισόδων των A και B (2^2=4) και αντίστοιχα για το vector C = “111111” (συνολικά 8 περιπτώσεις). Δεν χρειάζεται να δοκιμάσουμε κανέναν άλλο συνδυασμό για τα διάφορα bit του C, καθώς όπως συμπεραίνουμε από τις συναρτήσεις και από τη σχεδίαση του κυκλώματος το κάθε bit του C αφορά ένα συγκεκριμένο bit εξόδου (το C(0) αφορά το RESULT(0), το C(1) αφορά το RESULT(1) κλπ).

Κώδικας VHDL

```
1  ------------------------------------lst circuit-----------------------------------------------
2
3  library IEEE;                                                --library declaration
4  use IEEE.STD_LOGIC_1164.ALL;
5
6
7  entity circuitl is
8      Port ( A : in  STD_LOGIC;                                --signal declarations:
9             B : in  STD_LOGIC;                                --1 bit input
10            C : in  STD_LOGIC_VECTOR (5 downto 0);            --6 bits input
11            RESULT : out  STD_LOGIC_VECTOR (5 downto 0));     --6 bits output
12 end circuitl;
13
14 architecture dataflow of circuitl is                         --architecture implementation
15
16 begin
17
18    RESULT(0) <= (A NAND B) AND C(0);                         --(A'+B')*C(0)
19    RESULT(1) <= (A NOR B) AND C(1);                          --(A'*B')*C(1)
20    RESULT(2) <= (A AND B) AND C(2);                          --(A*B)*C(2)
21    RESULT(3) <= (A XOR B) AND C(3);                          --(A*B'+A'*B)*C(3)
22    RESULT(4) <= ((A AND B) OR ((NOT A) AND (NOT B))) AND C(4);  --(A*B+A'*B')*C(4)
23    RESULT(5) <= (((NOT A) AND B) XOR ((NOT A) OR B)) AND C(5);  --(A*B+A'*B')*C(5)
24
25 end dataflow;
26
27 ------------------------------------------------------------------------------------------------
```

Test bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY testbench_circuit_1 IS
END testbench_circuit_1;

ARCHITECTURE behavior OF testbench_circuit_1 IS

    COMPONENT circuit1
    PORT(
         A : IN  std_logic;
         B : IN  std_logic;
         C : IN  std_logic_vector(5 downto 0);
         RESULT : OUT  std_logic_vector(5 downto 0)
        );
    END COMPONENT;

   signal A : std_logic := '0';
   signal B : std_logic := '0';
   signal C : std_logic_vector(5 downto 0) := (others => '0');

   signal RESULT : std_logic_vector(5 downto 0);

BEGIN

   uut: circuit1 PORT MAP (
          A => A,
          B => B,
          C => C,
          RESULT => RESULT
        );

   stim_proc: process
  begin
```

```
          C <= "000000";              --All bits of vector C = 0

          A <= '0';                   --Every combination for A and B (2^2=4)
          B <= '0';
      wait for 100 ns;
          A <= '0';
          B <= '1';
      wait for 100 ns;
          A <= '1';
          B <= '0';
      wait for 100 ns;
          A <= '1';
          B <= '1';
      wait for 100 ns;

          C <= "111111";              --All bits of vector C = 1

          A <= '0';                   --Every combination for A and B(2^2=4)
          B <= '0';
      wait for 100 ns;
          A <= '0';
          B <= '1';
      wait for 100 ns;
          A <= '1';
          B <= '0';
      wait for 100 ns;
          A <= '1';
          B <= '1';
      wait for 100 ns;

      wait;
   end process;

END;
```

Μέρος Β - Κύκλωμα 2

Πίνακας Αληθείας Half Adder:

Inputs		Outputs	
A	B	RESULT	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Πίνακας Αληθείας Full Adder 1 bit:

Inputs			Outputs	
A	B	Cin	RESULT	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Πίνακας Αληθείας Full Adder 2 bits:

Inputs					Outputs		
A(0)	B(0)	A(1)	B(1)	Cin	RESULT(0)	RESULT(1)	Cout
0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	1	0	1	0	0	0	1
0	1	1	0	1	0	0	1
1	0	1	0	1	0	0	1
1	1	1	0	0	0	0	1
0	0	1	1	0	0	0	1
0	0	0	0	1	0	1	0
0	1	0	0	0	0	1	0
1	0	0	0	0	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	1	1
0	0	1	1	1	0	1	1
0	1	1	1	0	0	1	1
1	0	1	1	0	0	1	1
0	1	0	0	1	1	0	0
1	0	0	0	1	1	0	0
1	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0
0	1	1	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	1	1	0	1	0	1
1	1	0	0	1	1	1	0
0	0	0	1	1	1	1	0
0	1	0	1	0	1	1	0
1	0	0	1	0	1	1	0
0	0	1	0	1	1	1	0
0	1	1	0	0	1	1	0
1	0	1	0	0	1	1	0
1	1	1	1	1	1	1	1

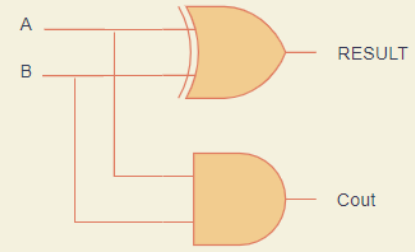
Για να αποφασίσουμε ποιες περιπτώσεις εισόδων θα εξετάσουμε στο top level test bench κατασκευάσαμε τον πίνακα αληθείας του full adder 2 bit για όλες τις περιπτώσεις εισόδων (2^5 = 32). Έπειτα, ομαδοποιήσαμε τις περιπτώσεις αυτές με βάση ποιες δίνουν την ίδια έξοδο(2^3 = 8 διαφορετικές έξοδοι). Για να ελέγξουμε την σωστή λειτουργία του κυκλώματος, επιλέξαμε μία από τις εισόδους για κάθε διαφορετική έξοδο. Παρακάτω φαίνεται ο τελικός πίνακας αληθείας:

Τελικός Πίνακας Αληθείας Full Adder 2 bits:

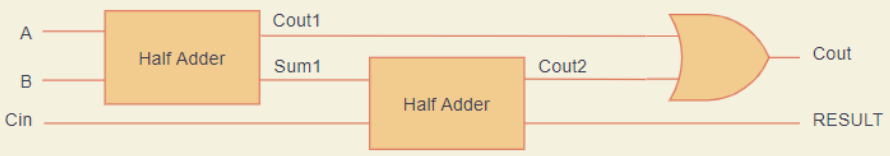
Inputs					Outputs		
A(0)	B(0)	A(1)	B(1)	Cin	RESULT(0)	RESULT(1)	Cout
0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1
0	1	0	0	0	0	1	0
1	1	0	1	1	0	1	1
0	1	0	0	1	1	0	0
0	1	1	1	1	1	0	1
1	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1

Κύκλωμα και Block Diagrams

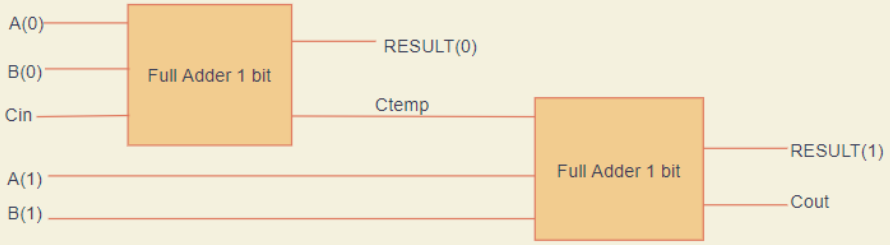
- **Half Adder:**



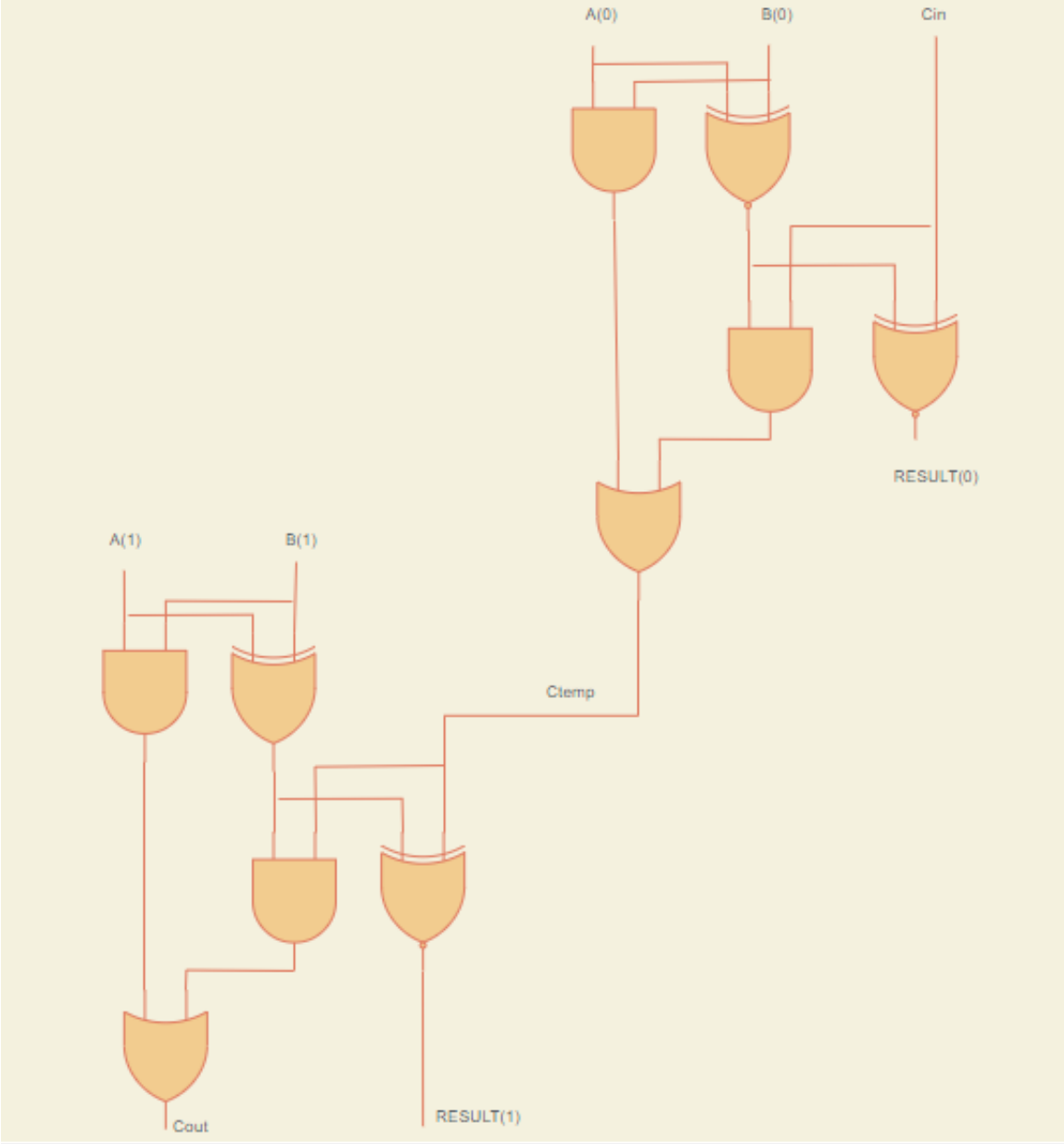
- **Full Adder 1bit:**



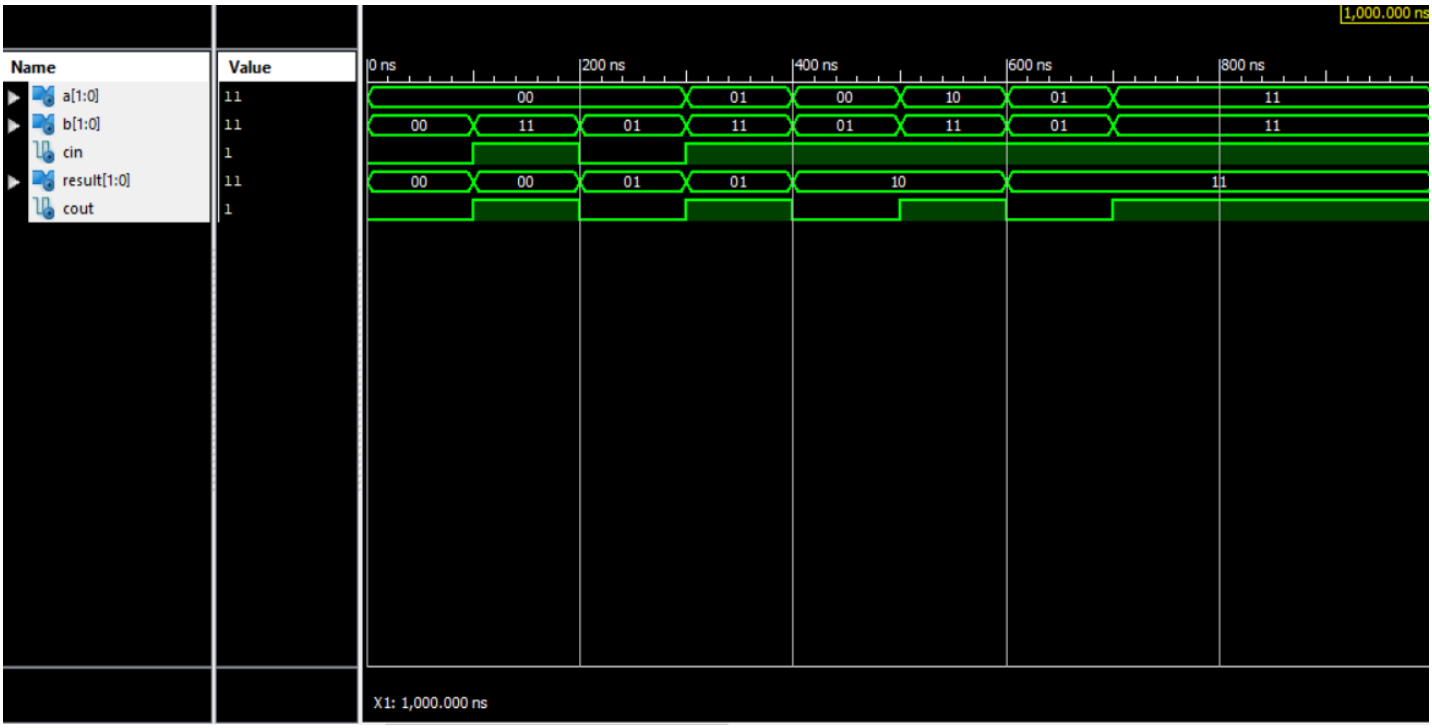
- **Full Adder 2bit:**



Το ίδιο συμπέρασμα για την επιλογή των εισόδων προκύπτει αν σχεδιάσουμε ολόκληρο το κύκλωμα και δούμε για ποιες εισόδους δεν αλλάζει η έξοδος. Για παράδειγμα, τα A(0), B(0) και αντίστοιχα A(1), B(1) είτε πάρουν την τιμή “01” είτε “10” δεν επηρεάζουν την έξοδο του κυκλώματος, αφού και οι 2 είσοδοι A και B συνδέονται με τις 2 πύλες AND και XOR. Επίσης, για κάποιες τιμές των A(0), B(0) και Cin εξουδετερώνεται το ενδιάμεσο κρατούμενο (Ctemp=0) και άρα ο $2^{ος}$ αθροιστής 1 bit μετατρέπεται σε ημιαθροιστή (δηλαδή η έξοδός του εξαρτάται μόνο από τα A(1) και B(1)). Βρίσκουμε για ποιες από αυτές τις τιμές το RESULT(0) παραμένει ίδιο, έτσι ώστε όλα τα bit εξόδου να είναι ίδια. Τέλος, διαγράφουμε τις επιπλέον εισόδους που δίνουν την ίδια έξοδο και κατασκευάζουμε το test bench.



Simulation



Κώδικες VHDL

```
1  ----------------------------------half adder-----------------------------------------------
2  library ieee;                                                --library declaration
3  use ieee.std_logic_1164.all;
4
5  entity half_adder is
6      Port (
7             A, B : in std_logic;                             --input declarations
8             RESULT,Cout: out std_logic);                     --output declarations
9
10 end half_adder;
11
12 architecture dataflow_1 of half_adder is                    --architecture implementation
13
14 begin
15
16     RESULT <= A xor B;                                      --sum = A'*B + A*B'
17     Cout <= A and B;                                        --carry = A*B
18
19 end dataflow_1;
20 ----------------------------------------------------------------------------------------------
```

```
--------------------------------1 bit full adder-----------------------------------------------------------
library ieee;                                                   --library declaration
use ieee.std_logic_1164.all;

entity full_adder_1bit is
port (
          A, B, Cin : in  std_logic;                            --input declarations
          RESULT, Cout : out  std_logic);                       --output declarations

end full_adder_1bit;

architecture dataflow_2 of full_adder_1bit is                   --architecture implementation

component half_adder                                            --component hald_adder declaration

port (  A, B : in std_logic;
        RESULT, Cout : out std_logic);

end component;

signal sum1, Cout1, Cout2: std_logic;                           --temporary in-between signals

begin

HA1: half_adder                                                 --named association conectivity of 1st half_adder
       port map ( A => A,                                       --A = A
                  B => B,                                       --B = B
                  RESULT => sum1,                               --RESULT = result of 1st half_adder
                  Cout => Cout1);                               --Cout = carry output of 1st half_adder
HA2: half_adder                                                 --named association conectivity of 2nd half_adder
       port map ( A => sum1,                                    --A = result of 1st half_adder
                  B => Cin ,                                    --B = carry input
                  RESULT => RESULT,                             --RESULT = result of 2nd half_adder
                  Cout => Cout2);                               --Cout = carry output of 2nd half_adder

           Cout <= Cout2 or Cout1;                              --Cout = Cout1 + Cout2

end dataflow_2;
-------------------------------------------------------------------------------------------------------------
```

```
-----------------------------------2 bits full adder-----------------------------------------------------
library ieee;                                           --library declaration
use ieee.std_logic_1164.all;

entity full_adder_2bit is
port (
          A,B: in std_logic_vector(1 downto 0);
          Cin : in  std_logic;                          --input declarations
          RESULT : out  std_logic_vector(1 downto 0);
          Cout : out  std_logic);                       --output declarations

end full_adder_2bit;

architecture dataflow_3 of full_adder_2bit is           --architecture implementation

component full_adder_1bit                               --component full_adder_1bit declaration

port (  A, B, Cin : in std_logic;
       RESULT, Cout : out std_logic);

end component;

signal Ctemp: std_logic;                                --temporary in-between signal

begin

FA1: full_adder_1bit                                    --named association conectivity of 1st full_adder_1bit
       port map ( A => A(0),                            --A = A(0)
                  B => B(0),                            --B = B(0)
                  Cin => Cin,                           --Cin = carry input
                  RESULT => RESULT(0),                  --RESULT = result of 1st full_adder_1bit
                  Cout => Ctemp);
FA2: full_adder_1bit                                    --named association conectivity of 2nd full_adder_1bit
       port map ( A => A(1),                            --A = A(1)
                  B => B(1),                            --B = B(1)
                  Cin => Ctemp,                         --Cin = carry output of 1st full_adder_1bit
                  RESULT => RESULT(1),                  --RESULT = result of 2nd full_adder_1bit
                  Cout => Cout);                        --Cout = carry output of 2nd full_adder_1bit

end dataflow_3;
-------------------------------------------------------------------------------------------------------------
```

Test bench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY testbench_full_adder_2bits IS
END testbench_full_adder_2bits;

ARCHITECTURE behavior OF testbench_full_adder_2bits IS

    COMPONENT full_adder_2bit
    PORT(
         A : IN  std_logic_vector(1 downto 0);
         B : IN  std_logic_vector(1 downto 0);
         Cin : IN  std_logic;
         RESULT : OUT  std_logic_vector(1 downto 0);
         Cout : OUT  std_logic
        );
    END COMPONENT;

   signal A : std_logic_vector(1 downto 0) := (others => '0');
   signal B : std_logic_vector(1 downto 0) := (others => '0');
   signal Cin : std_logic := '0';

   signal RESULT : std_logic_vector(1 downto 0);
   signal Cout : std_logic;

BEGIN

   uut: full_adder_2bit PORT MAP (
          A => A,
          B => B,
          Cin => Cin,
          RESULT => RESULT,
          Cout => Cout
        );
```

```
    stim_proc: process
    begin

       A <= "00";           --Input: all 0
       B <= "00";
       Cin <= '0';
    wait for 100 ns;

       A(0) <= '0';         --Input: "01011"
       B(0) <= '1';
       A(1) <= '0';
       B(1) <= '1';
       Cin <= '1';
    wait for 100 ns;

       A(0) <= '0';         --Input: "01000"
       B(0) <= '1';
       A(1) <= '0';
       B(1) <= '0';
       Cin <= '0';
    wait for 100 ns;

       A(0) <= '1';         --Input: "11011"
       B(0) <= '1';
       A(1) <= '0';
       B(1) <= '1';
       Cin <= '1';
    wait for 100 ns;

       A(0) <= '0';         --Input: "01001"
       B(0) <= '1';
       A(1) <= '0';
       B(1) <= '0';
       Cin <= '1';
```

```
62        Cin <= '1';
63     wait for 100 ns;
64
65        A(0) <= '0';         --Input: "01001"
66        B(0) <= '1';
67        A(1) <= '0';
68        B(1) <= '0';
69        Cin <= '1';
70     wait for 100 ns;
71
72        A(0) <= '0';         --Input: "01111"
73        B(0) <= '1';
74        A(1) <= '1';
75        B(1) <= '1';
76        Cin <= '1';
77     wait for 100 ns;
78
79        A(0) <= '1';         --Input: "11001"
80        B(0) <= '1';
81        A(1) <= '0';
82        B(1) <= '0';
83        Cin <= '1';
84     wait for 100 ns;
85
86        A <= "11";            --Input: all 1
87        B <= "11";
88        Cin <= '1';
89     wait for 100 ns;
90
91        wait;
92     end process;
93
94  END;
95
```