ΠΡΟΧΩΡΗΜΕΝΗ ΛΟΓΙΚΗ ΣΧΕΔΙΑΣΗ

ΑΝΑΦΟΡΑ 1ου ΕΡΓΑΣΤΗΡΙΟΥ

LAB$_{20335359}$
Γιώργος Βιριράκης 2016030035
Μιχάλης Γαλάνης 2016030036

ΣΚΟΠΟΣ

Σκοπός της πρώτης εργαστηριακής άσκησης ήταν η εξοικείωση με τη γλώσσα περιγραφής υλικού **VHDL** και τη λειτουργία του εργαλείου σχεδίασης και προσομοίωσης κυκλωμάτων **Xilinx ISE.** Με την ολοκλήρωση του εργαστηρίου αυτού, περιμέναμε να γνωρίσουμε τη διαδικασία σχεδίασης αναδιατασσόμενων συσκευών FPGAs και τον προγραμματισμό τους (**VHDL description** -> **Synthesis** -> **Implementation** -> **Configuration**).

ΠΡΟΕΡΓΑΣΙΑ

Για τη 1η εργαστηριακή άσκηση μας ζητήθηκε να υλοποιήσουμε **2 κυκλώματα** με εισόδους και εξόδους σε γλώσσα **VHDL**.

1ο ΚΥΚΛΩΜΑ

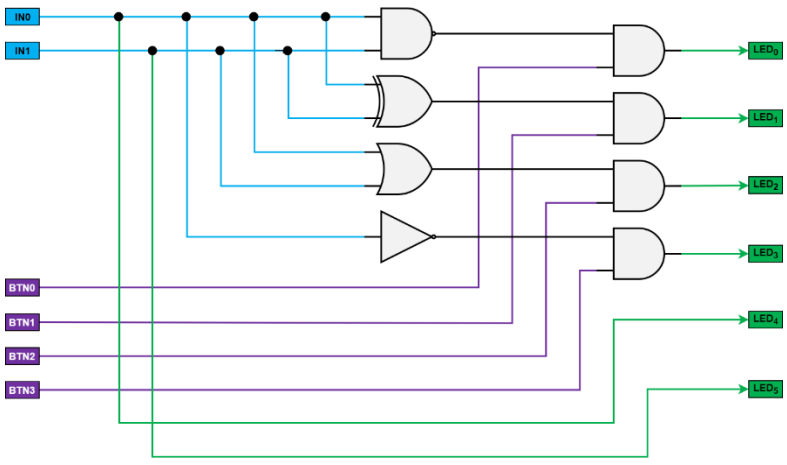
Το πρώτο κύκλωμα διαθέτει 6 εισόδους (2 switches & 4 buttons) και 6 εξόδους (1 bus πλάτους 6 bits).

Μας δόθηκαν αρχικά στην εκφώνηση κάποια στοιχεία για τη λειτουργία του κυκλώματος ώστε εμεις να υλοποιήσουμε τις συναρτήσεις. Παρατηρήσαμε απο τα στοιχεία αυτά ότι η έκφραση **«αν και μόνο αν»** ισοδυναμεί με τη λογική πύλη **AND**.

Παρακάτω παρουσιάζονται οι συναρτήσεις σε Άλγεβρα Boole και VHDL αντίστοιχα:

Algrebra Boole	VHDL Description
$LED_{[0]} = (IN_0 * IN_1)' * BTN_0$	LED(0) <= (IN0 NAND IN1) AND BTN0;
$LED_{[1]} = (IN_0 \oplus IN_1) * BTN_1$	LED(1) <= (IN0 XOR IN1) AND BTN1;
$LED_{[2]} = (IN_0 + IN_1) * BTN_2$	LED(2) <= (IN0 OR IN1) AND BTN2;
$LED_{[3]} = (IN_0)' * BTN_3$	LED(3) <= (NOT IN0) AND BTN3;
$LED_{[4]} = IN_0$	LED(4) <= IN0;
$LED_{[5]} = IN_1$	LED(5) <= IN1;

Αν σχεδιάσουμε στο **Draw.io** το κύκλωμα σύμφωνα με τα παραπάνω, δημιουργείται το παρακάτω κύκλωμα:

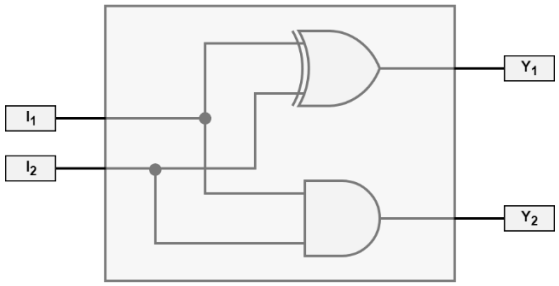


2ο ΚΥΚΛΩΜΑ

Στο δεύτερο ζητούμενο, κληθήκαμε να κατασκευάσουμε εναν **ημιαθροιστή** (Half Adder) με τη χρήση λογικών πυλών. Ύστερα θεωρούμε τον Half Adder **ως σύστημα** (component) για να κατασκευάσουμε με χρήση αυτού εναν **πλήρη αθροιστή** (Full Adder). Το τελικό ζητούμενο κύκλωμα είναι ένας πλήρης αθροιστής.

Το κύκλωμα αυτό αποτελείται απο 3 εισόδους (3 switches) και 2 εξόδους (1 bus πλάτους 2 bits).

Ξεκινάμε δημιουργώντας τον Half Adder. Το κύκλωμα του Half Adder αποτελείται και αυτό απο **2 εισόδους (I_1, I_2)** και **2 εξόδους (Y1, Y2)** που αντιπροσωπεύουν το **Sum** και το **Carry** αντίστοιχα. Το λογικό διάγραμμα του Half Adder εμφανίζεται παρακάτω:



HA

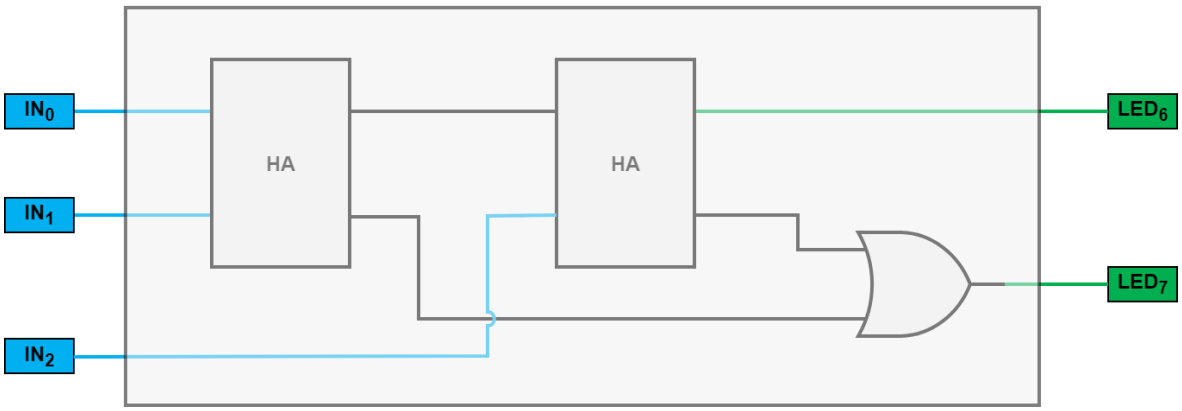
Ο **Full Adder**, γνωρίζουμε από τη θεωρία ότι αποτελείται από **2 Half Adders** και μια **πύλη OR.**

Οι συναρτήσεις που παράγονται σε Άλγεβρα Boole είναι:

$LED_{[6]}$ = (IN0 ⊕ IN1) ⊕IN2

$LED_{[7]}$ = ((IN0 ⊕ IN1) * IN2) + (IN0 * IN1)

Στις παραπάνω σχέσεις παρατηρούμε ότι το **(IN0 ⊕ IN1)** είναι η πρώτη έξοδος του πρώτου Half Adder και το **$LED_{[6]}$** είναι η πρώτη έξοδος του δεύτερου Half Adder. Επίσης το **(IN0 * IN1)** είναι η δεύτερη έξοδος του πρώτου Half Adder και το **((IN0 ⊕ IN1) * IN2)** είναι η δεύτερη έξοδος του δεύτερου Half Adder. Σχηματικά, το παραπάνω αναπαρίσταται ώς εξής:



FA

Να σημειωθεί ότι είχαμε **ξεχωριστό VHD Module** για το Half Adder και **ξεχωριστό για το Full Adder**. Η λογική του Half Adder εχει περιγραφεί στο VHD Module του Half Adder, ενω στο VHD Module του Full Adder, ο Half Adder αντιμετωπίζεται **ως ολοκληρωμένο component** και μας ενδιαφέρουν μόνο τα I/O του.

ΠΕΡΙΓΡΑΦΗ

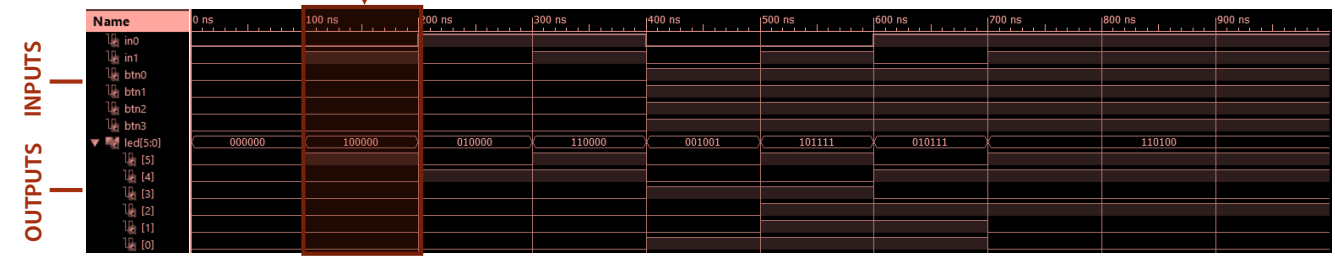
Στο προηγούμενο τμήμα της αναφοράς περιγράψαμε πως κατασκευάσαμε βήμα - βήμα τα διαγράμματα.

ΚΥΜΑΤΟΜΟΡΦΕΣ – ΠΡΟΣΟΜΟΙΩΣΗ

Για κάθε κύκλωμα, δημιουργήσαμε **ξεχωριστό testbench** με τα αντίστοιχα LED που χρησιμοποιούνται. Οι τιμές των outputs διαμορφώνονται με βάση τις συναρτήσεις που περιγράψαμε παραπάνω. Για παράδειγμα, βλέπουμε στη κυματομορφή του 1ου κυκλώματος, ότι για: **$IN_0 = 0$, $IN_1 = 1$** έχουμε **$LED_{0-4} = 0$**, **$LED_5 = 1$**.

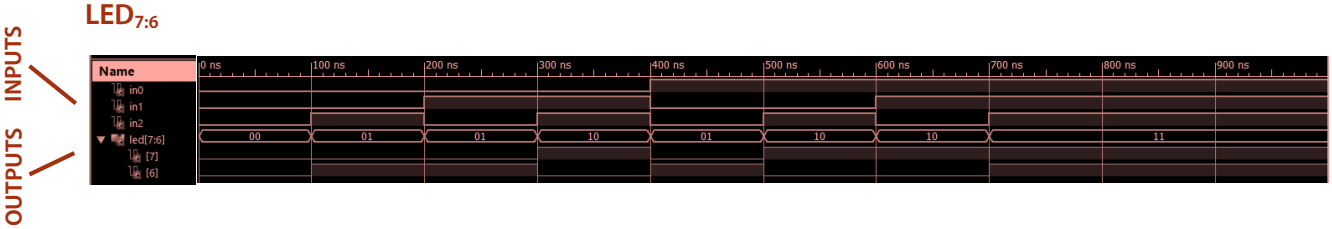
1ο ΚΥΚΛΩΜΑ

$LED_{5:0}$



2ο ΚΥΚΛΩΜΑ

$LED_{7:6}$



ΣΥΜΠΕΡΑΣΜΑΤΑ

Με το πέρας του πρώτου εργαστηρίου, βρισκόμαστε πλέον σε θέση να περιγράψουμε ένα κύκλωμα σε VHDL και να το υλοποιήσουμε σε ένα FPGA. Επίσης εξοικειωθήκαμε με το περιβάλλον του Xilinx ISE (μάθαμε να προσωμοιώνουμε κυκλώματα, να μελετάμε τις κυματομορφές τους και να τα εντάσσουμε στο FPGA - configuration).

ΠΑΡΑΡΤΗΜΑ – ΚΩΔΙΚΑΣ

1ο ΚΥΚΛΩΜΑ

VHD MODULE

```
--Libraries used by our program are here
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--I/O ports of "LAB01" is described here
entity LAB01 is
    Port ( IN0, IN1 : in  STD_LOGIC;
           BTN0, BTN1, BTN2, BTN3 : in  STD_LOGIC;
           LED : out  STD_LOGIC_VECTOR (5 downto 0)
           );
end LAB01;

--The logic of "LAB01" entity is described here
architecture Behavioral of LAB01 is

begin

    -- Each LED's output is described by the corresponding function we analyzed.
    LED(0) <= (IN0 NAND IN1) AND BTN0;
    LED(1) <= (IN0 XOR IN1) AND BTN1;
    LED(2) <= (IN0 OR IN1) AND BTN2;
    LED(3) <= NOT IN0 AND BTN3;
    LED(4) <= IN0;
    LED(5) <= IN1;

end Behavioral;
```

TESTBENCH

```
begin

 -------------------------------------------------------
    --Initial values are (IN0,IN1,BTN0,BTN1,BTN2,BTN3) = (0,0,0,0,0,0,0)
    --This section creates truth table for inputs IN0,IN1,BTN0.
    --Only needed values are changed.

    --(IN0,IN1,BTN0) = (0,0,    0)
    wait for 100 ns;

    --(IN0,IN1,BTN0) = (0,1,    0)
    IN1 <= '1';
    wait for 100 ns;

    --(IN0,IN1,BTN0) = (1,0,    0)
    IN0 <= '1';
    IN1 <= '0';
    wait for 100 ns;

    --(IN0,IN1,BTN0) = (1,1,    0)
    IN1 <= '1';
    wait for 100 ns;

    --(IN0,IN1,BTN0) = (0,0,    1)
    BTN0 <= '1';
    BTN1 <= '1';
    BTN2 <= '1';
    BTN3 <= '1';

    IN0 <= '0';
    IN1 <= '0';
    wait for 100 ns;

    --(IN0,IN1,BTN0) = (0,1,    1)
    IN1 <= '1';
    wait for 100 ns;

    --(IN0,IN1,BTN0) = (1,0,    1)
    IN0 <= '1';
    IN1 <= '0';
    wait for 100 ns;

    --(IN0,IN1,BTN0) = (1,1,    1)
    IN1 <= '1';
    wait for 100 ns;
    -------------------------------------------------------

  wait;
end process;
```

2ο ΚΥΚΛΩΜΑ

VHD MODULE (Half Adder)

```
--Libraries used by our program are here
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--I/O ports of "Half Adder" is described here
entity Half_Adder is
    Port ( I1 : in  STD_LOGIC;
           I2 : in  STD_LOGIC;
           Y1 : out  STD_LOGIC;
           Y2 : out  STD_LOGIC);
end Half_Adder;

--The logic of "Half_Adder" entity is described here
architecture Behavioral of Half_Adder is

begin

Y1 <= I1 XOR I2;
Y2 <= I1 AND I2;

end Behavioral;
```

VHD MODULE (Full Adder)

```
--Libraries used by our program are here
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--I/O ports of "Full_Adder" is described here
entity Full_Adder is
    Port ( IN0,IN1,IN2 : in  STD_LOGIC;
           LED : out  STD_LOGIC_VECTOR (7 downto 6));
end Full_Adder;

--The logic of "Full_Adder" entity is described here
architecture Structural of Full_Adder is
--We use signals to connect outputs of a system to inputs of a next system.
signal HA1_OUT1, HA1_OUT2, HA2_OUT1, HA2_OUT2 : STD_LOGIC;

--Standard structure of Half_Adder
component Half_Adder is
    Port ( I1,I2 : in STD_LOGIC;
           Y1,Y2 : out STD_LOGIC
         );
end component;

begin
    --To create a Full Adder, 2 Half Adders are needed.
    --The Half Adder is used as a component, the logic of which is described in a separate VHD Module.

    HA1: Half_Adder port map (  I1 => IN0,
                                I2 => IN1,
                                Y1 => HA1_OUT1,
                                Y2 => HA1_OUT2
                                );

    HA2: Half_Adder port map (  I1 => HA1_OUT1,
                                I2 => IN2,
                                Y1 => HA2_OUT1,
                                Y2 => HA2_OUT2
                                );

    LED(6) <= HA2_OUT1;
    LED(7) <= HA2_OUT2 OR HA1_OUT2;

end Structural;
```

TESTBENCH

```
begin

  -------------------------------------------------------------
    --Initial values are (IN0,IN1,IN2) = (0,0,0)
    --This section creates truth table for inputs IN0,IN1,IN2.
    --Only needed values are changed.

    --(0,0,0)
  wait for 100 ns;

    --(0,0,1)
    IN2 <= '1';
    wait for 100 ns;

    --(0,1,0)
    IN1 <= '1';
    IN2 <= '0';
    wait for 100 ns;

    --(0,1,1)
    IN2 <= '1';
    wait for 100 ns;

    --(1,0,0)
    IN0 <= '1';
    IN1 <= '0';
    IN2 <= '0';
    wait for 100 ns;

    --(1,0,1)
    IN2 <= '1';
    wait for 100 ns;

    --(1,1,0)
    IN1 <= '1';
    IN2 <= '0';
    wait for 100 ns;

    --(1,1,1)
    IN2 <= '1';
    wait for 100 ns;
    -------------------------------------------------------------

  wait;
end process;
```