Σχεδίαση Ψηφιακών Συστημάτων - Εργασία Θεωρίας (Μέρος 1)

Χρήστος Μαργιώλης

Ιούλιος 2020

Περιεχόμενα

1 Κώδικας και τεκμηρίωση

1.1 mux2to1.vhd

Το παρακάτω κύκλωμα υλοποιεί έναν πολυπλέκτη 2-σε-1. Η υλοποίηση της αρχιτεκτονικής του έχει ως εξής: αν η είσοδος s έχει τιμή 1, τότε θέτουμε στην έξοδο d την τιμή της εισόδου a, ειδάλλως της b.

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2to1 is port (
        a: in std_logic;
        b: in std_logic;
        s: in std_logic;
        d: out std_logic
);
end mux2to1;

architecture dataflow of mux2to1 is
begin
        d <= a when s = '1' else b;
end dataflow;
```

1.2 dec2to4.vhd

Το παρακάτω κύκλωμα υλοποιεί έναν αποκωδικοποιητή 2-σε-4 – δηλαδή έχουμε είσοδο 2 bit και δημιουργούμε μία έξοδο 4 bit. Για την υλοποίηση του χρησιμοποιούμε τις λογικές εξισώσεις του αποκωδικοποιητή:

$$D_0 = \overline{A_0 A_1}$$

$$D_1 = \overline{A_0} A_1$$

$$D_2 = A_0 \overline{A_1}$$

$$D_3 = A_0 A_1$$

```
library ieee;
use ieee.std_logic_1164.all;

entity dec2to4 is port (
        a: in std_logic_vector(1 downto 0);
        d: out std_logic_vector(3 downto 0)
);
end dec2to4;

architecture dataflow of dec2to4 is
begin
        d(0) <= not a(0) and not a(1);
        d(1) <= not a(0) and a(1);
        d(2) <= a(0) and not a(1);
        d(3) <= a(0) and a(1);
end dataflow;
```

1.3 mux2to1gen.vhd

Στο παρακάτω κύκλωμα ξαναϋλοποιούμε τον πολυπλέκτη 2-σε-1, απλώς αυτή την φορά παραμετροποιούμε τις εισόδους και εξόδους του ώστε να τον μετατρέψουμε σε πολλαπλό πολυπλέκτη. Στην προκειμένη περίπτωση η παράμετρος sz έχει την τιμή 4, οπότε έχουμε έναν τετραπλό πολυπλέκτη 2-σε-1.

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2to1gen is
generic (
        sz:     natural := 4
);
port (
        a:      in std_logic_vector(sz-1 downto 0);
        b:      in std_logic_vector(sz-1 downto 0);
        s:      in std_logic;
        c:      out std_logic_vector(sz-1 downto 0)
);
end mux2to1gen;

architecture dataflow of mux2to1gen is
begin
        c <= a when s = '1' else b;
end dataflow;
```

1.4 mux2to1gen_tb.vhd

Στο παρακάτω κύκλωμα δοκιμάζουμε την λειτουργία του προηγούμενου κυκλώματος με τη χρήση testbench. Η δημιουργία ενός testbench έχει ως εξής:

- Ορίζουμε ένα άδειο entity για το testbench.
- Στην αρχικτεκτονική ορίζουμε την περιγραφή του κυκλώματος που θέλουμε να δοκιμάσουμε ως component, στην προκειμένη περίπτωση τον πολλαπλό πολυπλέκτη 2-σε-1.
- Δημιουργούμε σήματα ίδιου τύπου για κάθε ένα από τα στοιχεία του component.
- Αντιστοιχούμε τα σήματα με τα πεδία του κυκλώματος (port map).
- Δημιουργούμε process και δίνουμε τιμές στα σήματα.

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2to1gen_tb is
end mux2to1gen_tb;

architecture behav of mux2to1gen_tb is

signal s_sz:    natural := 4;
signal s_a:     std_logic_vector(s_sz-1 downto 0);
signal s_b:     std_logic_vector(s_sz-1 downto 0);
signal s_s:     std_logic;
signal s_c:     std_logic_vector(s_sz-1 downto 0);

component mux2to1gen is
generic (
        sz:     natural := 4
);
port (
        a:      in std_logic_vector(sz-1 downto 0);
        b:      in std_logic_vector(sz-1 downto 0);
        s:      in std_logic;
        c:      out std_logic_vector(sz-1 downto 0)
);
end component;

begin
        uut: mux2to1gen port map (
                a => s_a,
                b => s_b,
                s => s_s,
                c => s_c
```

```
        );

        process begin
                s_a <= "0000";
                s_b <= "1101";
                s_s <= '0';
                wait for 250 ns;

                s_a <= "0000";
                s_b <= "1101";
                s_s <= '1';
                wait for 250 ns;
        end process;
end behav;
```

2 Εκτέλεση

Οι παρακάτω κυματομορφές αφορούνε το testbench για τον πολλαπλό πολυπλέκτη 2-σε-1: