Εργαστήριο Μικροεπεξεργαστών και Υλικού
Τμήμα ΗΜΜΥ
Πολυτεχνείο Κρήτης

# ΗΡΥ 211

# Προχωρημένη Λογική Σχεδίαση

# Εισαγωγή στη VHDL

Έκδοση 1.0

Εαρινό Εξάμηνο 2013

Καθ. Απόστολος Δόλλας

**Επιμέλεια :** Π. Μαλακωνάκης

Ε.Σωτηριάδης

**Χανιά 2013**

Η VHDL είναι η γλωσσά περιγραφής υλικού που χρησιμοποιείται για τη σχεδίαση και υλοποίηση λογικών κυκλωμάτων.

**VHDL : V**HSIC (Very High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage

Η VHDL δεν είναι case sensitive γλωσσά, δηλαδή δεν έχει διάφορα αν οι χαρακτήρες είναι πεζά η κεφαλαία.

Ένα παράδειγμα κώδικα **VHDL** μιας πύλης AND**:**

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

ENTITY and_gate IS
        PORT(
                a   : IN STD_LOGIC;
                b   : IN STD_LOGIC;
                z   : OUT STD_LOGIC
    );
END and_gate;

ARCHITECTURE model OF and_gate IS
BEGIN
        z <= a AND b;
END model;
```

Δήλωση βιβλιοθηκών:

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

Το *entity* είναι η περιγραφή της διεπαφής που θα έχει το κύκλωμά. Δηλαδή, είναι η δήλωση των εισόδων και εξόδων του κυκλώματος.

```
ENTITY and_gate IS
        PORT(
                a   : IN STD_LOGIC;
                b   : IN STD_LOGIC;
                z   : OUT STD_LOGIC
    );
END and_gate;
```

Στη VHDL χρησιμοποιούνται κυρίως 2 τύποι σημάτων.

**STD_LOGIC** : Δηλώνει ένα σήμα 1 bit

**STD_LOGIC_VECTOR**(n **DOWNTO** 0) : δηλώνει ένα σήμα n+1 bit.

Το *architecture* είναι η περιγραφή του κυκλώματος που αντιστοιχεί στο entity που έχουμε δηλώσει παραπάνω.

Ανάμεσα στο *begin* και το *end* γίνεται η περιγραφή του κυκλώματος.

```
ARCHITECTURE model OF and_gate IS
BEGIN
        z <= a AND b;
END model;
```

# Structural VHDL

Η *structural* περιγραφή της VHDL επιτρέπει τη σχεδίαση επιμέρους τμημάτων του κυκλώματος και ένωση τους σε ένα top level κύκλωμα.

Ένα παράδειγμα κώδικα **VHDL** μιας πύλης AND 3 εισόδων αποτελούμενη από 2 πύλες AND 2 εισόδων**:**

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

ENTITY and3 IS
	PORT(
		a   : IN STD_LOGIC;
		b   : IN STD_LOGIC;
		c   : IN STD_LOGIC;
		z   : OUT STD_LOGIC
    );
END and3;

architecture structural of and3 is

SIGNAL a1_out: STD_LOGIC;

COMPONENT and_gate
	PORT(
		a   : IN STD_LOGIC;
		b   : IN STD_LOGIC;
		z   : OUT STD_LOGIC
    );
END COMPONENT;

BEGIN
	a1: and_gate PORT MAP
				(a => a,
				 b => b,
				 z  => a1_out);

	a2: and_gate PORT MAP
				(a => a1_out,
				 b => c,
				 z  => z);

END structural;
```

Κάτω από τη δήλωση του architecture και πριν το begin γίνεται η δήλωση των signal και component.

Η δήλωση *signal* χρησιμοποιείται για να δηλώσουμε βοηθητικά σήματα τα όποια αφορούν συνδέσεις στο εσωτερικό του architecture της σχεδίασης.
Για παράδειγμα το σήμα **a1_out** στον παραπάνω κώδικα χρησιμοποιείται για τη σύνδεση της εξόδου της πρώτης πύλης AND με τη μια είσοδο της δεύτερης.

```
SIGNAL a1_out: STD_LOGIC;
```

Η δήλωση *component* χρησιμοποιείται για να δηλώσουμε τα επιμέρους κυκλώματα που θα χρησιμοποιήσουμε στη σχεδίαση. Το παρακάτω component αντιστοιχεί στο κύκλωμα που περιγράφεται στο πρώτο παράδειγμα.

```
COMPONENT and_gate
        PORT(
                a   : IN STD_LOGIC;
                b   : IN STD_LOGIC;
                z   : OUT STD_LOGIC
    );
END COMPONENT;
```

Για να χρησιμοποιήσουμε τα υποκυκλώματα που έχουν δηλωθεί ως component χρησιμοποιούμε το **port map.**

```
a1: and_gate PORT MAP
                (a => a,
                 b => b,
                 z  => a1_out);
```

Για κάθε είσοδο – έξοδο στο αντίστοιχο component στο port map πρέπει να αντιστοιχηθεί κάποιο σήμα, είτε από τις εισόδους – εξόδους του κυκλώματος (entity) είτε κάποιο signal.

# Behavioral VHDL

Behavioral VHDL χρησιμοποιούμε όταν θέλουμε να περιγράψουμε ένα κύκλωμα χωρίς τη χρήση για παράδειγμα λογικών πυλών ή λογικών εκφράσεων.

Ένα παράδειγμα κώδικα Behavioral VHDL είναι η παρακάτω υλοποίηση ενός πολυπλέκτη 4 σε 1:

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

entity mux4to1 is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : in  STD_LOGIC;
          d : in  STD_LOGIC;
          sel : in  STD_LOGIC_VECTOR (1 downto 0);
          muxout : out  STD_LOGIC);
end mux4to1;

architecture Behavioral of mux4to1 is

begin
process(sel,a,b,c,d)
begin

            if sel = "00" then
                        muxout <= a;
            elsif sel = "01" then
                        muxout <= b;
            elsif sel = "10" then
                        muxout <= c;
            elsif sel = "11" then
                        muxout <= d;
            end if;

end process;

end Behavioral;
```

Behavioral κώδικα γραφούμε πάντα μέσα σε process. Μέσα στην παρένθεση δηλώνουμε όλα τα σήματα των οποίων η αλλαγή επηρεάζει την έξοδο του κυκλώματος. Η παρένθεση ονομάζεται sensitivity list του process.

```
process(sel,a,b,c,d)
begin
……..
end process;
```

Μέσα στο process συνήθως χρησιμοποιούνται if-statement ή case-statement.

*If-statement*:

```
if sel = "00" then
        muxout <= a;
elsif sel = "01" then
        muxout <= b;
elsif sel = "10" then
        muxout <= c;
elsif sel = "11" then
        muxout <= d;
end if;
```

Το αντίστοιχο *case-statement*:

```
case sel is
        when "00" =>
                muxout <= a;
        when "01" =>
                muxout <= b;
        when "10" =>
                muxout <= c;
        when "11" =>
                muxout <= d;
        when others =>
                muxout <= a;
end case;
```

Το case-statement χρειάζεται πάντα δήλωση τιμής για την περίπτωση *when others* ακόμα και αν έχουμε συμπεριλάβει όλες τις περιπτώσεις όπως παραπάνω.

# Σύγχρονα Κυκλώματα

Τα σύγχρονα κυκλώματα ελέγχονται από το ρολόι. Οι αλλαγές στις τιμές των σημάτων συμβαίνουν στη θετική ακμή του ρολογιού. Αυτό δηλώνεται με την έκφραση:

```
Clock'EVENT AND Clock = '1'
```

Παρακάτω δίνεται ένα παράδειγμα ενός register 8 bit. Στη θετική ακμή του ρολογιού τα δεδομένα από την είσοδο προωθούνται στην έξοδο.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg8 IS
        PORT (  D       : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                Reset   : IN STD_LOGIC;
                Clock   : IN STD_LOGIC;
                Q       : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END reg8;

ARCHITECTURE Behavior OF reg8 IS
BEGIN
        PROCESS
        BEGIN
                Wait until Clock'EVENT AND Clock = '1' ;
                IF Reset = '0' THEN
                        Q <= "00000000";
                ELSE
                        Q <= D;
                END IF;
        END PROCESS;
END Behavior;
```

Το reset πρέπει να είναι σύγχρονο και όχι ασύγχρονο. Το παρακάτω παράδειγμα είναι η υλοποίηση του ίδιου register με σύγχρονο reset.

```
PROCESS
BEGIN
WAIT UNTIL Clock'EVENT AND Clock = '1';
        IF Reset = '0' THEN
                Q <= "00000000";
        ELSE
                Q <= D;
        END IF;
        END PROCESS;
```

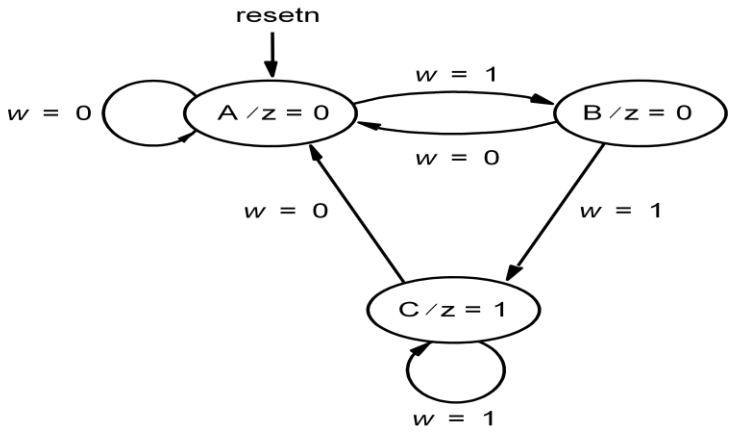
Όταν χρησιμοποιείται η έκφραση *wait until* στο process δε δίνεται sensitivity list.

# FSMs

Στη VHDL η υλοποίηση μιας FSM γίνεται όπως φαίνεται στο παρακάτω παράδειγμα.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY simple IS
        PORT (  clock     : IN STD_LOGIC;
                resetn    : IN STD_LOGIC;
                w         : IN STD_LOGIC ;
                z         : OUT STD_LOGIC ) ;
END simple;

ARCHITECTURE Behavior OF simple IS
TYPE State_type IS (A, B, C);
SIGNAL y : State_type;

BEGIN
PROCESS

        BEGIN
        Wait until (Clock'EVENT AND Clock = '1');
        IF resetn = '0' THEN
                y <= A ;
        ELSE
                CASE y IS
                        WHEN A =>
                                IF w = '0' THEN
                                        y <= A ;
                                ELSE
                                        y <= B ;
                                END IF ;
                        WHEN B =>
                                IF w = '0' THEN
                                        y <= A ;
                                ELSE
                                        y <= C ;
                                END IF ;
                        WHEN C =>
                                IF w = '0' THEN
                                        y <= A ;
                                ELSE
                                        y <= C ;
                                END IF ;
                END CASE ;
        END IF ;
END PROCESS ;
    z <= '1' WHEN y = C ELSE '0';
END Behavior ;
```

Ο παραπάνω κώδικας είναι η υλοποίηση της FSM που φαίνεται στο παρακάτω διάγραμμα καταστάσεων.




Για να δηλώσουμε τις καταστάσεις δημιουργούμε ένα νέο τύπο (State_type). Οι (A, B, C) αποτελούν τις καταστάσεις της FSM. Επίσης δηλώνουμε ένα σήμα τύπου State_type, το οποίο την κάθε χρονική στιγμή δείχνει το state που βρίσκεται το κύκλωμα.

```
TYPE State_type IS (A, B, C);
SIGNAL y : State_type;
```

Για την υλοποίηση των μεταβάσεων της FSM χρησιμοποιείται case-statement.

```
CASE y IS
        WHEN A =>
                IF w = '0' THEN
                        y <= A ;
                ELSE
                        y <= B ;
                END IF ;
        WHEN B =>
                IF w = '0' THEN
                        y <= A ;
                ELSE
                        y <= C ;
                END IF ;
        WHEN C =>
                IF w = '0' THEN
                        y <= A ;
                ELSE
                        y <= C ;
                END IF ;
```

```
END CASE ;
```