Σχεδίαση Υπολογιστικών Συστημάτων

Σχεδίαση συνδυαστικών κυκλωμάτων στην VHDL

Μιχάλης Ψαράκης

1-1

Σχεδίαση συνδυαστικών κυκλωμάτων

- Λογικές συναρτήσεις και πύλες (boolean functions)
- Πολυπλέκτες (multiplexers)
- Αποκωδικοποιητές (decoders)

Συνδυαστικά κυκλώματα

- Κυκλώματα των οποίων οι τιμές των εξόδων εξαρτώνται αποκλειστικά από τις τρέχουσες τιμές των εισόδων
  - καμία αποθήκευση των προηγουμένων τιμών των εισόδων
  - καμία *κατάσταση*
- Μπορούν να αναλυθούν με χρήση κανόνων λογικής
  - Λογική άλγεβρα ή άλγεβρα Boole

Λογικές συναρτήσεις

- Συναρτήσεις που λειτουργούν σε εισόδους των δύο τιμών και παράγουν εξόδους των δύο τιμών
  - 0, υλοποιείται ως χαμηλό επίπεδο τάσης
  - 1, υλοποιείται ως υψηλό επίπεδο τάσης
- Η συνάρτηση καθορίζει την τιμή της εξόδου για όλους τους πιθανούς συνδυασμούς της τιμής της εισόδου

Πίνακες αληθείας και πύλες

- Ορισμός μιας λογικής συνάρτησης με πίνακα

Λογικό OR

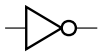
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Λογικό AND

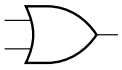
x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

Λογικό NOT

x	x'
0	1
1	0



αντιστροφέας



πύλη OR

πύλη AND

Λογικές εξισώσεις

- Λογικές εξισώσεις και πίνακες αληθείας είναι και τα δύο έγκυροι τρόποι για να τον ορισμό μιας συνάρτησης

$$f = (x + y) \cdot z'$$

- Υπολογίζουμε την f για κάθε συνδυασμό των τιμών εισόδου και συμπληρώνουμε τον πίνακα

x	y	z	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

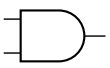
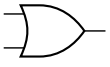
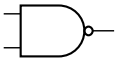
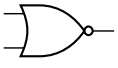
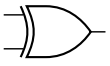
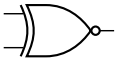
Λογικές εξισώσεις στην VHDL

- Χρησιμοποιούμε λογικούς τελεστές στις εντολές ανάθεσης σημάτων

$$f = \left(x + (y \cdot z')\right) + (y \cdot z)'$$

```
library ieee; use ieee.std_logic_1164.all;
entity circuit is
  port ( x, y, z  : in  std_logic;
         f        : out std_logic );
end entity circuit;
architecture boolean_eqn of circuit is
begin
  f <= (x or (y and not z)) and not (y and z);
end architecture boolean_eqn;
```


Λογικοί τελεστές στην VHDL

a **and** b	$a \cdot b$	
a **or** b	$a + b$	
a **nand** b	$\overline{a \cdot b}$	
a **nor** b	$\overline{a + b}$	
a **xor** b	$a \oplus b$	
a **xnor** b	$\overline{a \oplus b}$	
not a	$\overline{a}$	

- Προτεραιότητα
  - το **not** έχει την υψηλότερη
  - οι υπόλοιποι τελεστές έχουν ίση προτεραιότητα
  - χρησιμοποιούμε παρενθέσεις για να ξεκαθαρίσουμε τη σειρά υπολογισμού
- Τιμές bit στην VHDL
  - '0' και '1'

Τύποι δεδομένων

- Η VHDL είναι μια γλώσσα με ισχυρούς τύπους (strongly typed language)
  - Κάθε αντικείμενο (π.χ. σήμα ή μεταβλητή) μπορεί μόνο να λάβει τιμές του δηλωμένου τύπου του
  - Συμβάλλει στην ανίχνευση σχεδιαστικών λαθών
- Η VHDL περιέχει προκαθορισμένους τύπους
- Ο χρήστης μπορεί να ορίσει δικούς του τύπους

Προκαθορισμένοι τύποι

- Δηλώνονται σε πακέτα (packages):
- Πακέτο *standard* της βιβλιοθήκης *std*:
  - Τύποι: BIT, BOOLEAN, INTEGER, και REAL
- Πακέτο *std_logic_1164* της βιβλιοθήκης *ieee*:
  - Τύποι: STD_LOGIC και STD_ULOGIC
- Πακέτο *std_logic_arith* της βιβλιοθήκης *ieee*:
  - Τύποι: SIGNED και UNSIGNED

Πακέτα

- Για να χρησιμοποιήσουμε έναν τύπο που περιέχεται σε ένα πακέτο:
  - Πριν τη δήλωση της οντότητας:
  - Δηλώνουμε τη βιβλιοθήκη στην οποία βρίσκεται το πακέτο
  - Ορίζουμε το πακέτο με μια φράση χρήσης (use clause)

```
library library_name;
use library_name.package_name.package_parts;
```


Δήλωση πακέτων

- Συνήθως σε μια σχεδίαση χρησιμοποιούμε τύπους από 3 πακέτα:
  - ieee.std_logic_1164 (ieee library)
  - standard (std library)
  - work (work library)

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

library std;
use std.standard.all;

library work;
use work.all;
```

Είναι εξ'ορισμού ορατές. Δεν χρειάζεται να δηλωθούν

Τύπος bit

```
type bit is ('0', '1');
```

- Μεταφράζεται στη σύνθεση σε ένα μονό σήμα
- Η τιμή '0' αναπαριστάνεται από το λογικό επίπεδο 0 και η τιμή '1' από το λογικό επίπεδο 1
- Τα δύο λογικά επίπεδα δεν επαρκούν για να αναπαραστήσουν τη συμπεριφορά των σημάτων σε κάποιες περιπτώσεις (π.χ. high-Z)

Τύπος boolean

```
type boolean is (false, true);
```

- Μεταφράζεται στη σύνθεση σε ένα μονό σήμα
- Η τιμή false αναπαριστάνεται από το λογικό επίπεδο 0 και η τιμή true από το λογικό επίπεδο 1
- Ο τύπος boolean σπάνια χρησιμοποιείται για σήματα λογικών εκφράσεων
- Αντίθετα, χρησιμοποιείται έμμεσα από την γλώσσα για να αναπαραστήσει το αποτέλεσμα μίας πράξης σύγκρισης

Τύπος std_ulogic

```
type std_ulogic is ('U',   -- Uninitialized
                    'X',   -- Forcing Unknown
                    '0',   -- Forcing zero
                    '1',   -- Forcing one
                    'Z',   -- High Impedance
                    'W',   -- Weak Unknown
                    'L',   -- Weak zero
                    'H',   -- Weak one
                    '-' ); -- Don't care
```

- Μοντελοποιεί ηλεκτρικές ιδιότητες των σημάτων
- Χρήσιμος για την προσομοίωση των σημάτων

std_ulogic: τελεστής AND

```
FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;

CONSTANT and_table : stdlogic_table := (
--  ----------------------------------------------------
--  |  U    X    0    1    Z    W    L    H    -  |
--  ----------------------------------------------------
    ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ), -- | U |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | X |
    ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | 0 |
    ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | 1 |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | Z |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | W |
    ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | L |
    ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | H |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )  -- | - |
);
```

Σύνθεση του τύπου std_ulogic

- Στην προσομοίωση αντιμετωπίζεται σαν ένας τύπος απαρίθμησης
- Στην σύνθεση μεταφράζεται σε ένα μονό σήμα
- Οι τιμές του std_ulogic που είναι συνθέσιμες είναι: '0', '1', 'Z'
  - Η τιμή '0' αναπαριστάνεται από το λογικό επίπεδο 0, η τιμή '1' από το λογικό επίπεδο 1 και η τιμή 'Z' από την υψηλή εμπέδηση (high-impedance)

Σύνθεση του τύπου std_ulogic (συν.)

- Για τις υπόλοιπες τιμές τα εργαλεία σύνθεσης είτε θα τα μεταφράσουν σαν πραγματική τιμή (0 ή 1) είτε θα παράγουν λάθος
- Χρήσιμες σε κάποιες περιπτώσεις είναι η τιμή don't care (-)
  - επιτρέπει στο εργαλείο σύνθεσης την καλύτερη απλοποίηση του κυκλώματος

Τύπος std_logic

- std_ulogic: unresolved type
- std_logic:
  - υποτύπος του std_ulogic
  - resolved type
  - χρησιμοποιείται όταν θέλουμε ένα σήμα να έχει πολλαπλούς οδηγούς
    - μια συνάρτηση επίλυσης (resolution function) υπολογίζει την τιμή του σήματος
  - θα τον μελετήσουμε αργότερα όταν ασχοληθούμε με τρισταθής διαύλους (tristate busses)

Υλοποίηση συνδυαστικής λογικής

- Λογική ελέγχου ενός κλιματιστικού
  - τρεις λογικές εξισώσεις για την ενεργοποίηση του συστήματος θέρμανσης (heater), ψύξης (cooler) και του ανεμιστήρα (fan)

heater_on = temp_low · auto_temp + manual_heat
cooler_on = temp_high · auto_temp + manual_cool
fan_on = heater_on + cooler_on + manual_fan

```
library ieee; use ieee.std_logic_1164.all;
entity aircon is
  port (
    temp_low, temp_high, auto_temp : in std_logic;
    manual_heat, manual_cool, manual_fan : in std_logic;
    heater_on, cooler_on, fan_on : out std_logic );
end entity aircon;
```

Υλοποίηση συνδυαστικής λογικής (2)

heater_on = temp_low · auto_temp + manual_heat
cooler_on = temp_high · auto_temp + manual_cool
fan_on = heater_on + cooler_on + manual_fan

```
architecture eqns of aircon is
  signal heater_on_tmp, cooler_on_tmp : std_logic;
begin
  heater_on_tmp <= (temp_low and auto_temp) or manual_heat;
  cooler_on_tmp <= (temp_high and auto_temp) or manual_cool;
  fan_on <= heater_on_tmp or cooler_on_tmp or manual_fan;
  heater_on <= heater_on_tmp;
  cooler_on <= cooler_on_tmp;
end architecture eqns;
```

- Προσοχή: Χρησιμοποιούνται εσωτερικά σήματα
  - Δεν επιτρέπεται να διαβαστούν οι θύρες εξόδου

Δυαδική κωδικοποίηση

- Πώς αναπαριστάνουμε πληροφορία με περισσότερες από δύο πιθανές τιμές;
  - Πολλαπλά δυαδικά σήματα (πολλαπλά bit)
- (a_1, a_0): (0, 0), (0, 1), (1, 0), (1, 1)
  - Αυτός είναι ένας *δυαδικός κώδικας*
  - Κάθε ζεύγος τιμών είναι μια *κωδική λέξη*
  - Χρησιμοποιεί δύο αγωγούς σημάτων για τα a_1, a_0

Μέγεθος κωδικής λέξης

- Ένας κώδικας των n bit έχει 2^n κωδικές λέξεις
- Για να αναπαραστήσουμε N πιθανές τιμές
  - χρειαζόμαστε τουλάχιστον $\lceil \log_2 N \rceil$ bit για τις λέξεις
  - περισσότερα bit μπορούν να είναι χρήσιμα σε κάποιες περιπτώσεις
- Παράδειγμα: κώδικας εκτυπωτή ψεκασμού
  - μαύρο, κυανό, ιώδες, κίτρινο, κόκκινο, μπλε
  - έξι τιμές, $\lceil \log_2 6 \rceil = 3$
  - μαύρο: (0, 0, 1), κυανό: (0, 1, 0), ιώδες: (0, 1, 1), κίτρινο: (1, 0, 0), κόκκινο: (1, 0, 1), μπλε: (1, 1, 0)

Κώδικες one-hot

- Κάθε κωδική λέξη έχει ακριβώς ένα bit 1
- Φωτεινός σηματοδότης:
  - κόκκινο: (1,0,0), πορτοκαλί: (0,1,0), πράσινο: (0,0,1)
  - τρεις αγωγοί σημάτων: κόκκινο, πορτοκαλί, πράσινο
- Κάθε bit ενός κώδικα one-hot αντιστοιχεί σε μια κωδικοποιημένη τιμή
  - δεν απαιτείται υλικό για να αποκωδικοποιήσουμε τιμές

Δυαδικοί κώδικες στην VHDL

- Πολλαπλά bit αναπαριστάνονται από ένα διάνυσμα

```
signal s: std_logic_vector(4 downto 0);
```

  - Αυτό είναι ένα σήμα των πέντε στοιχείων
  - s(4), s(3), s(2), s(1), s(0)

```
signal a: std_logic_vector(1 to 3);
```

  - Αυτό είναι ένα σήμα των τριών στοιχείων
  - a(1), a(2), a(3)

Παράδειγμα δυαδικής κωδικοποίησης

- Ελεγκτής φωτεινού σηματοδότη με κώδικα 1-hot
  - enable = 1: lights_out = lights_in
  - enable = 0: lights_out = (0, 0, 0)

```
library ieee; use ieee.std_logic_1164.all;
entity light_controller is
  port ( lights_in  : in  std_logic_vector(1 to 3);
         enable     : in  std_logic;
         lights_out : out std_logic_vector(1 to 3) );
end entity light_controller;
```

Παράδειγμα δυαδικής κωδικοποίησης

```
architecture and_enable of light_controller is
begin
  lights_out(1) <= lights_in(1) and enable;
  lights_out(2) <= lights_in(2) and enable;
  lights_out(3) <= lights_in(3) and enable;
end architecture and_enable;
```

```
architecture conditional_enable of light_controller is
begin
  lights_out <= lights_in when enable = '1' else
                "000";
end architecture conditional_enable;
```


Τύποι απαρίθμησης

- Σύνολο από ονόματα για τις κωδικοποιημένες τιμές κάποιων σημάτων
  - Αντί για άμεση δυαδική κωδικοποίηση

```
type alu_function is (pass, add, subtract,
                      multiply, divide);
type octal_digit is ('0', '1', '2', '3', '4',
                     '5', '6', '7');
…
variable alu_op : alu_function;
variable last_digit : octal_digit := '0';
…
alu_op := subtract;
last_digit := '7';
```

Προκαθορισμένοι τύποι απαρίθμησης

- Character
  - `type character is (nul, … , '0', '1', … , 'A', 'B', … , 'a', 'b', …)`
  - Περιλαμβάνει όλους τους χαρακτήρες του συνόλου χαρακτήρων 8-bit Latin-1
- Boolean
  - `type boolean is (false, true);`
- Bit
  - `type bit is ('0', '1');`

Σύνθεση του τύπου απαρίθμησης

- Μεταφράζεται σε ένα σύνολο σημάτων ικανά να κωδικοποιήσουν όλες τις τιμές του τύπου
- Τα εργαλεία σύνθεσης υποστηρίζουν διαφορετικές μορφές κωδικοποίησης των τύπων απαρίθμησης
  - Binary
  - Onehot
  - Gray

Σύνθεση του τύπου απαρίθμησης

```
type alu_function is (pass, add, subtract,
                      multiply, divide);
```

- Πώς θα κωδικοποιηθεί ο παραπάνω τύπος απαρίθμησης αν επιλέξουμε μία από τις ακόλουθες μορφές κωδικοποίησης;
  - Binary
  - Onehot
  - Gray

Τύποι πινάκων (arrays)

- Παραδείγματα μονοδιάστατων πινάκων

Δηλώσεις τύπων πινάκων

```
type word is array (0 to 31) of bit;   -- Δείκτης: αύξουσα σειρά
type word is array (31 downto 0) of bit; -- Δείκτης: φθίνουσα σειρά
…
-- Δείκτης: τύπος απαρίθμησης
type controller_state is (initial, idle, active, error);
```

Αντικείμενα πινάκων

```
variable buffer_register, data_register : word;
variable state : controller_state;
```

Προτάσεις ανάθεσης

```
state := idle;
buffer_register(10) := '1';
data_register := buffer_register;
```

Προκαθορισμένοι τύποι πινάκων

- Αλφαριθμητικά (strings)

```
type string is array (positive range <>) of character;
```

- Διανύσματα bit (bit vectors)

```
type bit_vector is array (natural range <>) of bit;
…
variable channel_busy_register : bit_vector(1 to 4);
```

- Πίνακες πρότυπης λογικής (standard-logic arrays)

```
type std_ulogic_vector is array ( natural range <> ) of std_ulogic;
…
signal csr_offset : std_ulogic_vector(2 downto 1);
```


Παράδειγμα: bit vs. bit_vector

- Οι λογικοί τελεστές **not, and, or, nand, nor, xor, xnor** εφαρμόζονται στους τύπους:
  - bit, std_ulogic, std_logic, bit_vector, std_ulogic_vector, std_logic_vector

```
entity and2 is
port (a,b: in bit;
      x  : out bit);
end and2;

architecture and2 of and2 is
begin
   x <= a and b;
end and2;
```

```
entity and2 is
port(a,b: in bit_vector (0 to 3);
     x  : out bit_vector (0 to 3));
end and2;

architecture and2 of and2 is
begin
   x <= a and b;
end and2;
```

- Ποια είναι η διαφορά των δυο παραπάνω κυκλωμάτων;

Παράδειγμα: τμήματα πινάκων

```
subtype halfword is bit_vector(0 to 15);
…
entity byte_swap is
  port (input : in halfword; output : out halfword);
end entity byte_swap;

architecture behavior of byte_swap is
begin

  output(8 to 15) <= input(0 to 7);
  output(0 to 7) <= input(8 to 15);

end architecture behavior;
```

- Ποια είναι η λογική του κυκλώματος;

Τελεστής & (concatenation)

```
entity conc is
  port (a,b : in bit_vector(3 downto 0);
        z   : out bit_vector(7 downto 0) );
end entity conc;

architecture beh of conc is
begin

  z <= a & b;

end architecture beh;
```

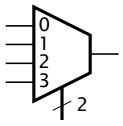
- Τι κύκλωμα προκύπτει;

Συνδυαστικά στοιχεία

- Μπορούμε να κατασκευάσουμε σύνθετα συνδυαστικά στοιχεία από πύλες
  - Πολυπλέκτες
  - Αποκωδικοποιητές, κωδικοποιητές
  - ...
- Τα χρησιμοποιούμε ως υπομονάδες για να κατασκευάσουμε μεγαλύτερα συστήματα
  - Αφαίρεση και επαναχρησιμοποίηση

Πολυπλέκτες (multiplexers)

4-to-1 mux



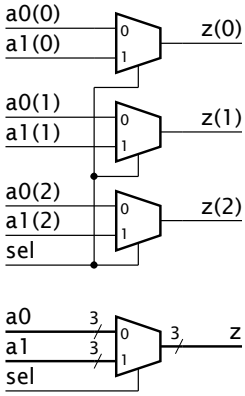
sel	z
00	a_0
01	a_1
10	a_2
11	a_3

```
library ieee; use ieee.std_logic_1164.all;
entity multiplexer_4_to_1 is
  port ( a   : in std_logic_vector (3 downto 0);
         sel : in std_logic_vector (1 downto 0);
         z   : out std_logic );
end entity multiplexer_4_to_1;

architecture eqn of multiplexer_4_to_1 is
begin
  with sel select
    z <= a(0) when "00", a(1) when "01",
         a(2) when "10", a(3) when others;
end architecture eqn;
```

Πολυπλέκτες πολλαπλών bit

- Επιλογή μεταξύ N λέξεων των m-bit
  - Σύνδεση m πολυπλεκτών N-to-1

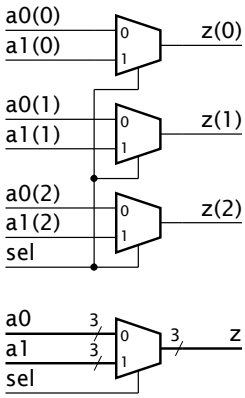



```
library ieee; use ieee.std_logic_1164.all;
entity multiplexer_3bit_2_to_1 is
  port (
    a0,a1 : in std_logic_vector (2 downto 0);
    sel   : in std_logic;
    z     : out std_logic_vector (2 downto 0) );
end entity multiplexer_3bit_2_to_1;

architecture eqn of multiplexer_3bit_2_to_1 is
begin
  z <= a0 when sel = '0' else
       a1;
end architecture eqn;
```


Πολυπλέκτες πολλαπλών bit (συν.)

- Δομική περιγραφή



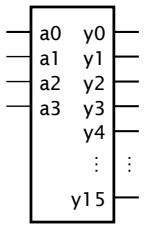

```
library ieee; use ieee.std_logic_1164.all;
entity multiplexer_3bit_2_to_1 is
  port (
    a0,a1 : in std_logic_vector (2 downto 0);
    sel   : in std_logic;
    z     : out std_logic_vector (2 downto 0) );
end entity multiplexer_3bit_2_to_1;

architecture str of multiplexer_3bit_2_to_1 is
begin
  mux0: entity work.multiplexer_2_to1(eqn)
    port map (a0(0), a1(0), sel, z(0));
  mux0: entity work.multiplexer_2_to1(eqn)
    port map (a0(1), a1(1), sel, z(1));
  mux0: entity work.multiplexer_2_to1(eqn)
    port map (a0(2), a1(2), sel, z(2));
end architecture eqn;
```

Αποκωδικοποιητές (decoders)

- Αποκωδικοποιητής 4-σε-16
  - Έξοδος για (a3, a2, a1, a1) = (1, 0, 1, 1)

$$y_{11} = a_3 \cdot \overline{a_2} \cdot a_1 \cdot a_0$$



- **Παράδειγμα:** εκτυπωτής ink-jet με έξι έγχρωμα μελάνια:
  - black, cyan, magenta, yellow, light cyan και light magenta

Color	Codeword (c_2, c_1, c_0)
black	0, 0, 1
cyan	0, 1, 0
magenta	0, 1, 1
yellow	1, 0, 0
red	1, 0, 1
blue	1, 1, 0

- Ο αποκωδικοποιητής για την λογική ελέγχου του εκτυπωτή
  - 3 bit εισόδου για την επιλογή χρώματος και
  - 6 bit εξόδου, ένα για την επιλογή κάθε μελανιού

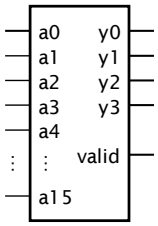
Αποκωδικοποιητές (συν.)

```
library ieee; use ieee.std_logic_1164.all;
entity ink_jet_decoder is
  port ( color2, color1, color0 : in std_logic;
         black, cyan, magenta,
         yellow, red, blue      : out std_logic );
end entity ink_jet_decoder;

architecture eqn of ink_jet_decoder is
begin
  black   <= not color2 and not color1 and     color0;
  cyan    <= not color2 and     color1 and not color0;
  magenta <= not color2 and     color1 and     color0;
  yellow  <=     color2 and not color1 and not color0;
  red     <=     color2 and not color1 and     color0;
  blue    <=     color2 and     color1 and not color0;
end architecture eqn;
```

Κωδικοποιητές (coders)

- Κωδικοποιητής 16-σε-4
  - Με έξοδο valid για να υποδεικνύει τις μη-έγκυρες κωδικές λέξεις



- **Παράδειγμα:** Κωδικοποιητής για ένα σύστημα συναγερμού με 8 ζώνες
  - 8 bit εισόδου: ένας αισθητήρας σε κάθε ζώνη
  - 3 bit εξόδου για την κωδικοποίηση των ζωνών

Zone	Codeword
Zone 1	0, 0, 0
Zone 2	0, 0, 1
Zone 3	0, 1, 0
Zone 4	0, 1, 1
Zone 5	1, 0, 0
Zone 6	1, 0, 1
Zone 7	1, 1, 0
Zone 8	1, 1, 1

Κωδικοποιητές (συν.)

```
library ieee;
use ieee.std_logic_1164.all;
entity alarm is
  port ( zone          : in  std_logic_vector (1 to 8);
         intruder_zone : out std_logic_vector(2 downto 0);
         valid         : out std_logic );
end entity alarm;
architecture eqn of alarm is
begin
  intruder_zone(2) <= zone(5) or zone(6) or zone(7) or
                      zone(8);
  intruder_zone(1) <= zone(3) or zone(4) or zone(7) or
                      zone(8);
  intruder_zone(0) <= zone(2) or zone(4) or zone(6) or
                      zone(8);
  valid <= zone(1) or zone(2) or zone(3) or zone(4) or
           zone(5) or zone(6) or zone(7) or zone(8);
end architecture eqn;
```

Κωδικοποιητής προτεραιότητας

- Εάν είναι 1 περισσότερες από μία είσοδοι
  - Κωδικοποίηση της εισόδου με τη μεγαλύτερη προτεραιότητα

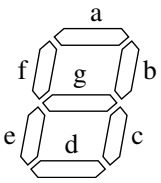
zone								intruder_zone			valid
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(2)	(1)	(0)	
1	–	–	–	–	–	–	–	0	0	0	1
0	1	–	–	–	–	–	–	0	0	1	1
0	0	1	–	–	–	–	–	0	1	0	1
0	0	0	1	–	–	–	–	0	1	1	1
0	0	0	0	1	–	–	–	1	0	0	1
0	0	0	0	0	1	–	–	1	0	1	1
0	0	0	0	0	0	1	–	1	1	0	1
0	0	0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0	–	–	–	0

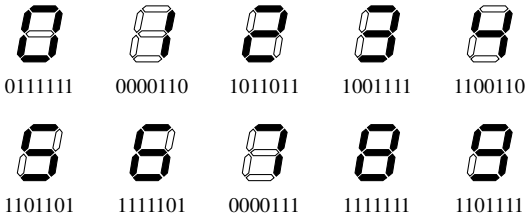
Κωδικοποιητής προτεραιότητας(συν.)

```
architecture priority_1 of alarm is
begin
  intruder_zone <= "000" when zone(1) = '1' else
                   "001" when zone(2) = '1' else
                   "010" when zone(3) = '1' else
                   "011" when zone(4) = '1' else
                   "100" when zone(5) = '1' else
                   "101" when zone(6) = '1' else
                   "110" when zone(7) = '1' else
                   "111" when zone(8) = '1' else
                   "000";
  valid <= zone(1) or zone(2) or zone(3) or zone(4)
           or zone(5) or zone(6) or zone(7) or zone(8);
end architecture priority_1;
```

Παράδειγμα: BCD-to-7 segment decoder

- Αποκωδικοποιεί έναν BCD αριθμό για να οδηγήσει ένα 7-segment LED
  - Segments: (g, f, e, d, c, b, a)





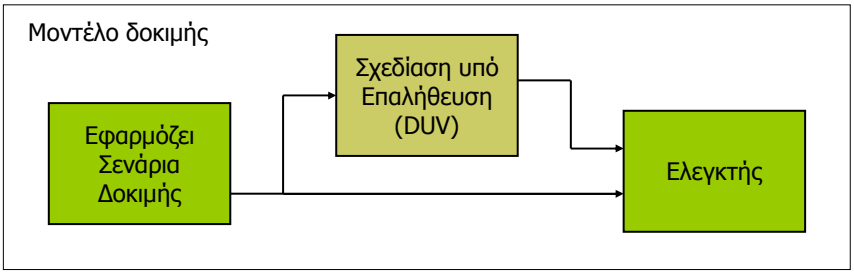

```
library ieee; use ieee.std_logic_1164.all;
entity seven_seg_decoder is
  port ( bcd   : in std_logic_vector (3 downto 0);
         blank : in std_logic;
         seg   : out std_logic_vector (7 downto 1) );
end entity seven_seg_decoder;
```


BCD-to-7 segment decoder

```
architecture behavior of seven_seg_decoder is
  signal seg_tmp : std_logic_vector (7 downto 1);
begin
  with bcd select
    seg_tmp <= "0111111" when "0000", -- 0
               "0000110" when "0001", -- 1
               "1011011" when "0010", -- 2
               "1001111" when "0011", -- 3
               ...
               "1101111" when "1001", -- 9
               "1000000" when others; -- "-"
  seg <= "0000000" when blank = '1' else
         seg_tmp;
end architecture behavior;
```

Επαλήθευση συνδυαστικών κυκλωμάτων

- Συνδυαστικά κυκλώματα: η έξοδος είναι συνάρτηση των εισόδων
  - Λειτουργική επαλήθευση: επιβεβαιώνει ότι είναι η σωστή συνάρτηση!



Παράδειγμα επαλήθευσης

- Επαληθεύστε τη λειτουργία του ελεγκτή του φωτεινού σηματοδότη
- Ιδιότητα που πρέπει να ελέγξετε
  - enable = '1' ⇒ lights_out = lights_in
  - enable = '0' ⇒ όλα τα φώτα είναι απενεργοποιημένα
- Αναπαραστήστε την ιδιότητα ως έναν ισχυρισμό (assertion) στον ελεγκτή

Μοντέλο δοκιμής: entity/architecture

```
entity light_testbench is
end entity light_testbench;
```

```
library ieee; use ieee.std_logic_1164.all;
architecture verify of light_testbench is
  signal lights_in : std_logic_vector (1 to 3);
  signal enable : std_logic;
  signal lights_out : std_logic_vector (1 to 3);
begin
  duv : entity work.light_controller(and_enable)
    port map (lights_in, enable, lights_out);
```


Εφαρμογή των σεναρίων δοκιμής

```
apply_test_cases : process is
begin
  enable <= '0'; lights_in <= "000"; wait for 1 sec;
  enable <= '0'; lights_in <= "001"; wait for 1 sec;
  enable <= '0'; lights_in <= "010"; wait for 1 sec;
  enable <= '0'; lights_in <= "100"; wait for 1 sec;
  enable <= '1'; lights_in <= "001"; wait for 1 sec;
  enable <= '1'; lights_in <= "010"; wait for 1 sec;
  enable <= '1'; lights_in <= "100"; wait for 1 sec;
  enable <= '1'; lights_in <= "000"; wait for 1 sec;
  enable <= '1'; lights_in <= "111"; wait for 1 sec;
  wait;
end process apply_test_cases;
```

Έλεγχος ισχυρισμών

```
  check_outputs : process is
  begin
    wait on enable, lights_in;
    wait for 10 ms;
    assert (enable = '1' and lights_out = lights_in)
           or (enable = '0' and lights_out = "000");
  end process check_outputs;
end architecture verify;
```


Λειτουργική κάλυψη

- Έχουμε δοκιμάσει όλα τα πιθανά σενάρια εισόδου;
- Για μεγάλες σχεδιάσεις, η εξαντλητική δοκιμή δεν είναι εφικτή
  - *N* είσοδοι: αριθμός σεναρίων = 2^N
- Λειτουργική κάλυψη
  - Ποσοστό των σεναρίων δοκιμής που καλύπτονται από το μοντέλο δοκιμής
  - Είναι δύσκολο να αποφασίσουμε πόση δοκιμή είναι αρκετή