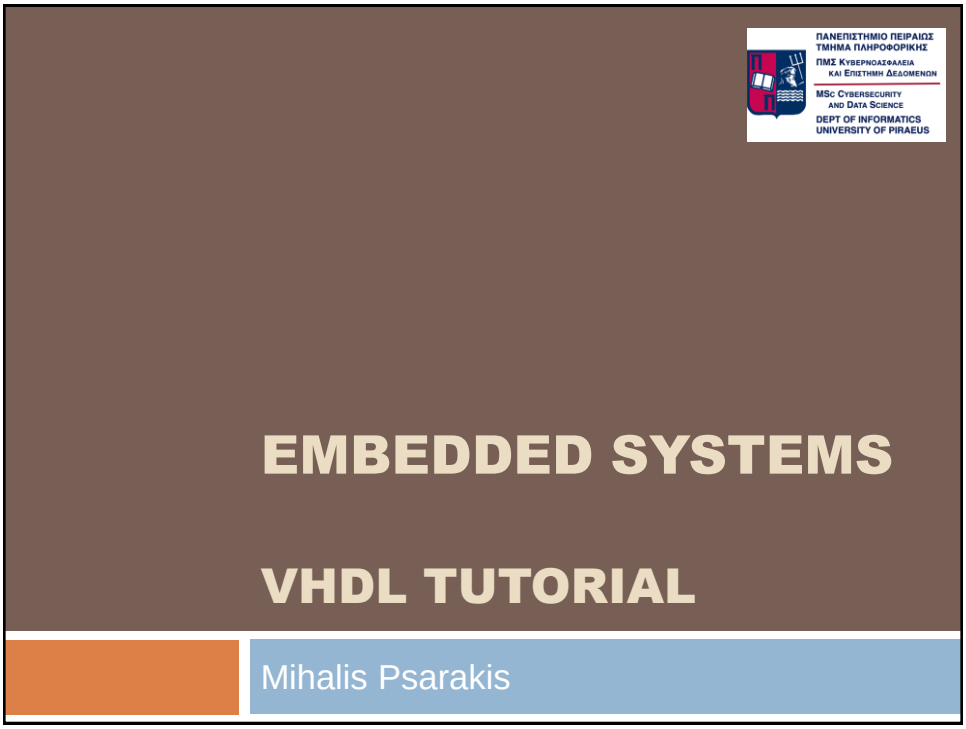


Ενότητα 1

- Γλώσσες περιγραφής υλικού (Hardware Description Languages, HDLs)
- VHDL για σύνθεση
  - Βασικές έννοιες της γλώσσας
- Τύποι δεδομένων της VHDL
  - Προκαθορισμένοι τύποι, ποιοι από αυτούς είναι συνθέσιμοι και ποιο είναι το αποτέλεσμα της σύνθεσής τους

Εργαλεία μοντελοποίησης

3

- Σχηματικό διάγραμμα (schematic diagram)
- Γλώσσες προδιαγραφών (specification languages)
  - SpecC, SystemC
- Γλώσσες περιγραφής υλικού (hardware description languages, HDLs)
  - VHDL
  - Verilog

Γλώσσες περιγραφής υλικού: VHDL

4

- **VHDL**: **V**HSIC **H**ardware **D**escription **L**anguage
  - VHSIC: **V**ery **H**igh-**S**peed **I**ntegrated **C**ircuits
- Ιστορική αναδρομή:
  - Ξεκίνησε το 1981 από το Υπουργείο Άμυνας των ΗΠΑ ως γλώσσα περιγραφής ολοκληρωμένων κυκλωμάτων
  - Οι εταιρείες IBM, Texas Instruments, Intermetrics ανάπτυξαν και κυκλοφόρησαν την 1η έκδοση το 1985
  - Έγινε πρότυπο από τον οργανισμό IEEE
    - IEEE Standard 1076-1987
    - IEEE Standard 1076-1993
- Πιο διαδεδομένη στην Ευρώπη

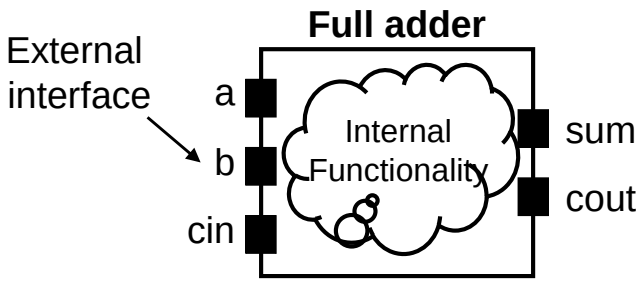
Γλώσσες περιγραφής υλικού: Verilog

5

- Ιστορική αναδρομή:
  - Αναπτύχθηκε ως γλώσσα μοντελοποίησης υλικού από την εταιρεία Gateway Design Automation το 1984 για ιδιωτική χρήση
  - Η εταιρεία Cadence Design Systems αγόρασε την Gateway το 1990
  - Η εταιρεία Cadence είναι υπεύθυνη για την προώθηση της Verilog ως γλώσσα μοντελοποίησης & προσομοίωσης
  - Η εταιρεία Synopsys είναι υπεύθυνη για την προώθηση της Verilog ως γλώσσα σύνθεσης
  - Ανοικτή γλώσσα από το 1991
  - Πρότυπο από τον οργανισμό IEEE το 1995
- Πιο διαδεδομένη στην Αμερική

Σχεδιαστική μονάδα στη VHDL

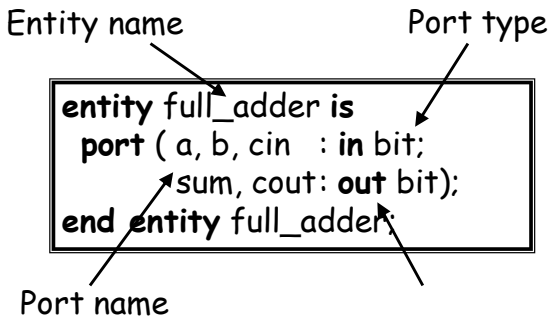
6



Οντότητα (entity)

7

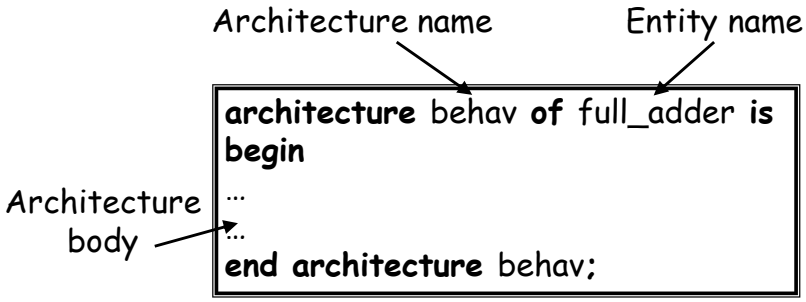
- Περιγράφει την εξωτερική διασύνδεση (external interface) της σχεδιαστικής μονάδας



Αρχιτεκτονική (architecture)

8

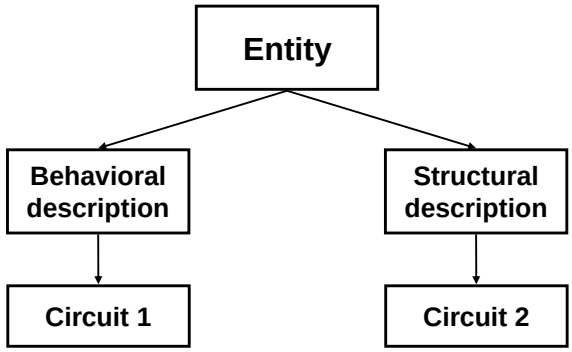
- Περιγράφει την εσωτερική συνάρτηση (internal functionality) της σχεδιαστικής μονάδας



Οντότητα και αρχιτεκτονικές

9

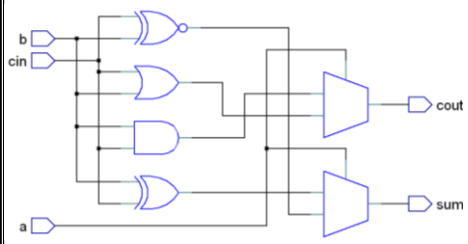
- Υπάρχουν διαφορετικές αρχιτεκτονικές για να περιγράψουν την συνάρτηση μίας οντότητας



Περιγραφή συμπεριφοράς (behavioral description)

10

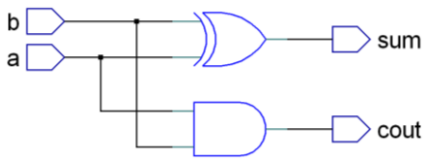
```
architecture behav of full_adder is
begin

p: process (a,b,cin) is
begin
if a = '1' then
    cout <= b or cin;
    sum <= b xnor cin;
 else
   cout <= b and cin;
   sum <= b xor cin;
end if;
end process;

end architecture behav;
```



Περιγραφή δομής (structural description)

11

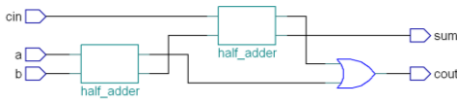
```
entity half_adder is
  port (a,b : in bit;
        sum,cout : out bit);
end entity half_adder;

architecture behav of half_adder is

begin

sum <= a xor b;
cout <= a and b;

end architecture behav;
```



Περιγραφή δομής (structural description)

12

```
architecture struct of full_adder is

signal sum1,cout1,cout2: bit;

begin

ha1: entity work.half_adder(behav)
port map(a,b,sum1,cout1);

ha2: entity work.half_adder(behav)
port map(cin,sum1,sum,cout2);

cout <= cout1 or cout2;

end architecture struct;
```



Σήματα (signals) & θύρες (ports)

13

- Τα σήματα μεταφέρουν δεδομένα εντός της αρχιτεκτονικής
- Οι θύρες αποτελούν τα σήματα επικοινωνίας της οντότητας με τον «έξω κόσμο»

```
entity circuit is
  port (port specification);
end entity circuit;

architecture struct of full_adder is
signal declaration
begin
...
end architecture circuit;
```

Είδος θύρας (port mode)

14

- Καθορίζει την κατεύθυνση των δεδομένων της θύρας
  - in : input port
  - out : output port
  - inout : bidirectional port

Τύποι (types)

15

- VHDL = strongly-typed language
- Υπάρχουν προκαθορισμένοι τύποι στη γλώσσα
- Ο χρήστης μπορεί να ορίσει υπο-τύπους (subtypes)
- Ο χρήστης μπορεί να ορίσει πρόσθετους τύπους

Κλάσεις τύπων

16

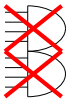
- Υπάρχουν 8 κλάσεις τύπων εκ των οποίων οι 4 είναι συνθέσιμες:
  - Τύποι απαρίθμησης (enumeration)
  - Τύποι ακεραίων (integer)
  - Τύποι πινάκων (array)
  - Τύποι εγγραφής (record)
  - Τύποι κινητής υποδιαστολής (floating point)
  - Φυσικοί τύποι (physical)
  - Τύποι προσπέλασης (access)
  - Τύποι αρχείων (file)




Προκαθορισμένοι τύποι (standard types)

17

Τύπος	Κλάση	
boolean	enumeration	Δηλώνονται στο package standard
bit	enumeration	
character	enumeration	
severity_level	enumeration	
integer	integer	
natural	integer subtype	
positive	integer subtype	
real	floating-point	
time	physical	
string	array of character	
bit_vector	array of bit	



std_logic types

18

- Προστέθηκαν στη γλώσσα για να μοντελοποιήσουν λογικές πύλες
- Δηλώνονται στο package std_logic_1164 της βιβλιοθήκης IEEE
- Είναι συνθέσιμοι

Τύπος	Κλάση
std_ulogic	enumeration
std_logic	subtype of std_ulogic
std_ulogic_vector	array of std_ulogic
std_logic_vector	array of std_logic

Τελεστές (operators)

19

- boolean — not, and, or, nand, nor, xor, **xnor**
- σύγκρισης — =, /=, <, >, <=, >=
- ολίσθησης — **sll, srl, sla, sra, rol, ror**
- αριθμητικοί — sign +, sign -, abs, +, -, *, /, mod, rem, **
- σύντμησης — &

Κάποιοι από τους τελεστές (bold) προστέθηκαν στην έκδοση VHDL'93 και δεν είναι συνθέσιμοι από εργαλεία που υποστηρίζουν μόνο την έκδοση VHDL'87

Τύπος bit

20

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

- boolean — not, and, or, nand, nor, xor, xnor
- σύγκρισης — =, /=, <, >, <=, >=

Σύνθεση του τύπου bit

21

- Μεταφράζεται στη σύνθεση σε ένα μονό σήμα
- Η τιμή '0' αναπαριστάται 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 αρκετές περιπτώσεις (π.χ. high-Z, unknown)

Χρησιμοποιήστε τον τύπο std_ulogic αντί του τύπου bit

Τύπος integer

22

```
type integer is range -2147483648 to +2147483647
```

- σύγκρισης =, /=, <, >, <=, >=
- αριθμητικοί +, -, *, /, mod, **

- Τα όρια του τύπου integer εξαρτώνται από την υλοποίηση των εργαλείων VHDL (τουλάχιστον από -2^31+1 έως 2^31–1)

Συμβουλευτείτε το εγχειρίδιο του εργαλείου VHDL για τα όρια των ακεραίων αριθμών πριν τους χρησιμοποιήσετε

Τύποι integer ορισμένοι από τον χρήστη

23

```
type short is range -128 to +127
```

- Ορίζει τους 8-bit ακέραιους σε αναπαράσταση συμπληρώματος ως προς 2
- Ισχύουν όλοι οι τελεστές που εφαρμόζονται στον τύπο integer
- Δεν επιτρέπονται πράξεις μεταξύ σημάτων τύπου integer και τύπου short (strongly-typed language)
- Όλες οι πράξεις μεταξύ σημάτων τύπου short πρέπει να δίνουν ενδιάμεσα/τελικά αποτελέσματα τύπου short (διαφορετικά θα συμβεί λάθος στην προσομοίωση)

Χρήση των τύπων integer

24

- Η χρήση τύπων integer συμβάλλει στην ανίχνευση σχεδιαστικών λαθών
- Η χρήση πολλών διαφορετικών τύπων integer μπορεί να δημιουργήσει προβλήματα
- Τύποι integer ορισμένοι από τον χρήστη εναντίον υποτύπων integer

```
signal a,b,c : integer range 0 to 15;
signal res : integer range 0 to 15;
...
res <= a – b + c;
```

Λάθος προσομοίωσης
(a=3, b=4, c=5)

```
subtype nat4 is natural range 0 to 15;
...
signal a,b,c : nat4;
signal res : nat4;
...
res <= a – b + c;
```

Σύνθεση του τύπου integer

25

- Μεταφράζεται σε ένα σύνολο σημάτων ικανά να αναπαραστήσουν όλες τις τιμές του τύπου
- Πόσα σήματα απαιτούνται για να αναπαραστήσουμε τους παρακάτω τύπους integer;

```
subtype nat4 is natural range 0 to 15;
type offset is range 30 to 31;
type negative is range -2147483648 to -1;
```

Προσοχή: Το διάστημα των σημάτων τύπου integer καθορίζει το μέγεθος του αριθμητικού κυκλώματος που παράγει το εργαλείο σύνθεσης

Τύποι signed και unsigned

26

```
signal x: signed(7 downto 0);
signal y: unsigned (0 to 3);
```

- Ο τύπος signed(7 downto 0) αναπαριστάνει όλους τους προσημασμένους αριθμούς (σε συμπλήρωμα ως προς 2) με 8 bit
- Ο τύπος unsigned(0 to 3) αναπαριστάνει όλους τους απρόσημους (θετικούς) αριθμούς με 4 bit
- Δηλώνονται σαν διανύσματα (όπως ο τύπος std_logic_vector) και όχι σαν ακέραιοι (όπως ο τύπος integer)
- Επιτρέπουν την εκτέλεση αριθμητικών λειτουργιών (σε αντίθεση με τα διανύσματα τύπου std_logic_vector)

Τύποι απαρίθμησης (enumeration)

27

```
type alu_func is (disable, pass, add, sub, mult, div);
```

- σύγκρισης =, /=, <, >, <=, >=
- Προκαθορισμένοι τύποι απαρίθμησης:
  - τύπος boolean: **type** boolean **is** (false,true);
  - τύπος bit: **type** bit **is** (‘0’, ‘1’);
  - τύπος character: **type** character **is** (‘a’, ‘b’, ‘c’, ….);

Σύνθεση του τύπου απαρίθμησης

28

- 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
  - Auto (area minimization)

Σύνθεση του τύπου απαρίθμησης

29

```
type alu_func is (disable, pass, add, sub, mult, div);
```

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

Τύπος πίνακα (array)

30

```
type word1 is array (0 to 31) of bit;
type word2 is array (31 downto 0) of bit;
type state is (initial, idle, active, error);
type state_a1 is array (state) of natural;
type state_a2 is array (state range initial to active) of natural;
```

- σύγκρισης =, /=, <, >, <=, >=
- σύντμησης &
- Για πίνακες τύπου bit, boolean, std_logic ορίζονται επιπλέον οι τελεστές:
  - boolean not, and, or, nand, nor, xor, xnor
  - ολίσθησης sll, srl, sla, sra, rol, ror

Τμήματα (slices) πινάκων

31

```
subtype halfword is bit_vector(0 to 15);

entity byte_swap is
   port (input: in halfword; output: out halfword);
end entity byte_swap;

architecture beh of byte_swap is
begin
  output(8 to 15) <= input(0 to 7);
  output(0 to 7) <= input(8 to 15);
end architecture beh;
```

Σύνθεση του τύπου πίνακα

32

- Εξαρτάται από τον τύπο των στοιχείων του πίνακα
- Πολλά σχεδιαστικά λάθη οφείλονται στην χρήση των πινάκων:
  - Αύξουσα και φθίνουσα σειρά των δεικτών
  - Καταχωρήσεις μεταξύ τμημάτων του πίνακα

Απλή καταχώρηση σήματος (signal assignment)

33

```
architecture df of full_adder is

signal sum1,c1,c2: bit;

begin

sum1 <= a xor b;
c1 <= a and b;
c2 <= sum1 and cin;
cout <= c1 or c2;
sum <= sum1 xor cin;

end architecture df;
```

Παράλληλη εκτέλεση των εντολών καταχώρησης

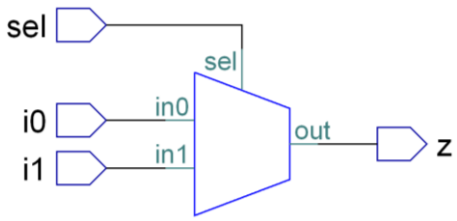
```
...
cout <= c1 or c2;
sum <= sum1 xor cin;
sum1 <= a xor b;
c1 <= a and b;
c2 <= sum1 and cin;
...
```

Τι θα συμβεί εάν αλλάξουμε την σειρά εκτέλεσης των εντολών;

Καταχώρηση σήματος υπό συνθήκη (conditional assignment)

34

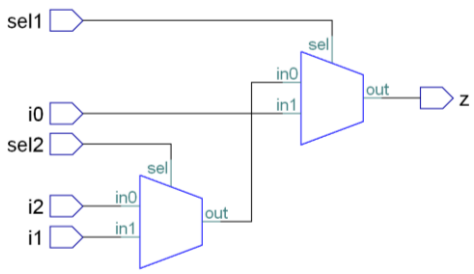
```
entity mux2x1 is
  port (i0,i1 : in bit;
        sel   : in bit;
        z     : out bit);
end entity mux2x1;

architecture beh of mux2x1 is
begin

z <= i0 when sel = '0' else
     i1;

end architecture beh;
```



Πολυ-επίπεδη καταχώρηση σήματος υπό συνθήκη

35

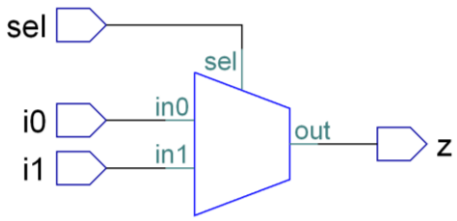
```
entity circ is
  port (i0,i1,i2 : in bit;
        sel1, sel2 : in bit;
        z : out bit);
end entity circ;

architecture beh of circ is
begin

z <= i0 when sel1 = '1' else
     i1 when sel2 = '1' else
     i2;

end architecture beh;
```



Καταχώρηση σήματος με επιλογή (selected assignment)

36

```
entity mux2x1 is
  port (i0,i1 : in bit;
        sel   : in bit;
        z     : out bit);
end entity mux2x1;

architecture beh of mux2x1 is
begin

with sel select
  z <=  i0 when '0',
        i1 when '1';

end architecture beh;
```



Αρχικές τιμές σημάτων

37

```
signal a : bit := '1';
```

- Έχουν νόημα μόνο για την προσομοίωση
- Δεν λαμβάνονται υπόψιν από το εργαλείο σύνθεσης
- Η σχεδίαση ενός μηχανισμού που θα αρχικοποιεί τα σήματα είναι πολύ σημαντική (κυρίως για ακολουθιακά κυκλώματα)

Τύπος std_ulogic

38

```
type std_ulogic is ('U',   -- uninitialized
                    'X',   -- forcing unknown
                    '0',   -- forcing 0
                    '1',   -- forcing 1
                    'Z',   -- high impedance
                    'W',   -- weak unknown
                    'L',   -- weak 0
                    'H',   -- weak 1
                    '-',   -- don't care
                   );
```

- Λογικός τύπος πολλαπλών τιμών
- Οι τιμές δεν ανταποκρίνονται στον πραγματικό κόσμο αλλά είναι χρήσιμες στην προσομοίωση

std_ulogic: τελεστής AND

39

```
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

40

- Στην προσομοίωση αντιμετωπίζεται σαν ένας τύπος απαρίθμησης
- Στην σύνθεση μεταφράζεται σε ένα μονό σήμα
- Η τιμή ‘0’ αναπαριστάται 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

41

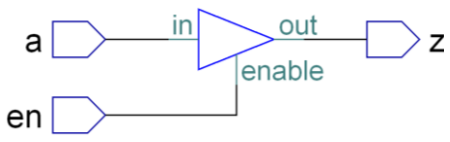
- Οι τιμές του std_ulogic που είναι συνθέσιμες είναι: ‘0’, ‘1’, ‘Z’
- Για τις υπόλοιπες τιμές τα εργαλεία σύνθεσης είτε θα τα μεταφράσουν σαν πραγματική τιμή (0 ή 1) είτε θα παράγουν λάθος 
- Χρήσιμες σε κάποιες περιπτώσεις είναι η τιμή don’t care (-)
  - επιτρέπει στο εργαλείο σύνθεσης την καλύτερη απλοποίηση του κυκλώματος

Σύνθεση του τύπου std_ulogic

42

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

entity trist is
  port ( a, en : in std_ulogic;
         z : out std_ulogic);
end entity trist;

architecture beh of trist is
begin

z <= a when en = '1' else
     'Z';

end architecture beh;
```

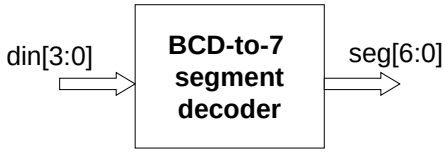
RTL Schematic



Παράδειγμα συνδυαστικού κυκλώματος

43

- Υλοποίηση ενός BCD-to-7 segment decoder



BCD-to-7 segment decoder

44

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

entity seven_seg is
  port(
   din : in std_ulogic_vector(3 downto 0);
   seg : out std_ulogic_vector(6 downto 0)
    );
end entity seven_seg;

architecture beh of  seven_seg is
begin
with din select
  seg <=
     "1111110" when "0000", -- 0
     "0110000" when "0001", -- 1
                                  continue…
```

```
…continue
     "1101101" when "0010", -- 2
     "1111001" when "0011", -- 3
     "0110011" when "0100", -- 4
     "1011011" when "0101", -- 5
     "1011111" when "0110", -- 6
     "1110000" when "0111", -- 7
     "1111111" when "1000", -- 8
     "1111011" when "1001", -- 9
     "0000000" when "1010", -- 10
     "0000000" when "1011", -- 11
     "0000000" when "1100", -- 12
     "0000000" when "1101", -- 13
     "0000000" when "1110", -- 14
     "0000000" when "1111"; -- 15

end architecture beh;
```

BCD-to-7 segment decoder

45

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

entity seven_seg is
  port(
   din : in std_ulogic_vector(3 downto 0);
   seg : out std_ulogic_vector(6 downto 0)
    );
end entity seven_seg;

architecture beh of  seven_seg is
begin
with din select
  seg <=
     "1111110" when "0000", -- 0
     "0110000" when "0001", -- 1
                                  continue…
```

```
…continue
     "1101101" when "0010", -- 2
     "1111001" when "0011", -- 3
     "0110011" when "0100", -- 4
     "1011011" when "0101", -- 5
     "1011111" when "0110", -- 6
     "1110000" when "0111", -- 7
     "1111111" when "1000", -- 8
     "1111011" when "1001", -- 9
     "0000000" when others; -- others

end architecture beh;
```

BCD-to-7 segment decoder

46

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

entity seven_seg is
  port(
   din : in std_ulogic_vector(3 downto 0);
   seg : out std_ulogic_vector(6 downto 0)
    );
end entity seven_seg;

architecture beh of  seven_seg is
begin
with din select
  seg <=
     "1111110" when "0000", -- 0
     "0110000" when "0001", -- 1
                                  continue…
```

```
…continue
     "1101101" when "0010", -- 2
     "1111001" when "0011", -- 3
     "0110011" when "0100", -- 4
     "1011011" when "0101", -- 5
     "1011111" when "0110", -- 6
     "1110000" when "0111", -- 7
     "1111111" when "1000", -- 8
     "1111011" when "1001", -- 9
     "_______" when others; -- others

end architecture beh;
```

- Καλύτερη απλοποίηση
- Προσοχή στην χρήση των τιμών '-'

Ενότητα 2

47

- Τελεστές της VHDL
  - Αποτέλεσμα της σύνθεσης των τελεστών

Λογικοί τελεστές (boolean)

48

not, and, or, nand, nor, xor, xnor

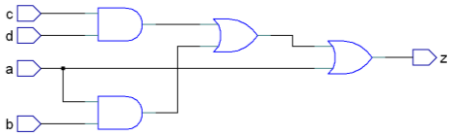
- Μεταφράζονται σε λογικές πύλες
- Τα εργαλεία σύνθεσης:
  - απλοποιούν τις λογικές συναρτήσεις
  - αντιστοιχίζουν τις λογικές συναρτήσεις σε cells της βιβλιοθήκης

Λογικοί τελεστές (boolean)

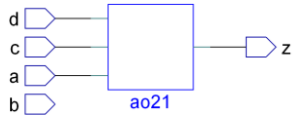
49

```
entity bool_op is
  port (
        a,b,c,d : in bit;
        z : out bit
        );
end entity bool_op;

architecture beh of bool_op is
begin

z <= (a and b) or (c and d) or a;

end architecture beh;
```

RTL Schematic



Technology Schematic



Τελεστές σύγκρισης

50

=, /=, <, >, <=, >=

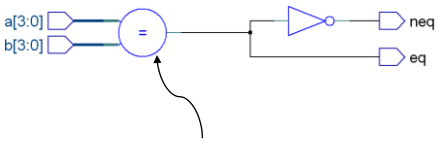
- Μεταφράζονται με την χρήση αριθμητικών κυκλωμάτων
- =, /=
  - σύγκριση ισότητας bit-by-bit με χρήση πυλών XOR
- <, >, <=, >=
  - χρήση αφαιρέτη και απλοποίηση του κυκλώματος
  - a > b ⇔ (a – b) > 0

Τελεστές =, /=

51

```
entity comp_eq is
  port (a,b : in integer range 0 to 15;
        eq, neq : out bit);
end entity comp_eq;

architecture beh of comp_eq is
begin

eq  <= '1' when a = b else '0';
neq <= '1' when a /= b else '0';

end architecture beh;
```

RTL Schematic



Υλοποιείται με πύλες XOR

Τελεστές ολίσθησης

52

sll, srl, sla, sra, rol, ror

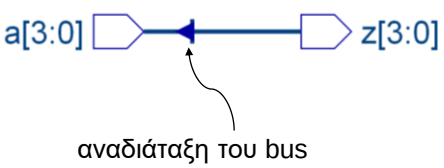
- Όταν το δεξί τελούμενο είναι σταθερή τιμή
  - κανένα λογικό κύκλωμα, αναδιάταξη του bus
- Όταν το δεξί τελούμενο είναι σήμα ή μεταβλητή
  - χρήση κυκλώματος ολισθητή

Τελεστές ολίσθησης

53

```
entity shift is
 port (
      a : in bit_vector (3 downto 0);
      z : out bit_vector (3 downto 0)
      );
end entity shift;

architecture beh of shift is
begin

z <= a sll 2;

end architecture beh;
```

RTL Schematic

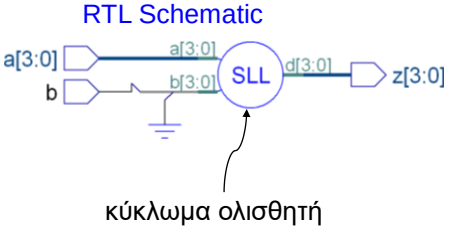


Τελεστές ολίσθησης

54

```
entity shift is
 port (
      a : in bit_vector (3 downto 0);
      b : in integer range 0 to 1;
      z : out bit_vector (3 downto 0)
      );
end entity shift;

architecture beh of shift is
begin

z <= a sll b;

end architecture beh;
```

RTL Schematic



Αριθμητικοί τελεστές

55

+, -, *, /, mod, **

- Το αποτέλεσμα της σύνθεσης εξαρτάται από το εργαλείο
- Τα εργαλεία σύνθεσης υποστηρίζουν διαφορετικές υλοποιήσεις ενός αριθμητικού κυκλώματος
- Το εργαλείο σύνθεσης εισάγει την μικρότερη υλοποίηση που ικανοποιεί τους χρονικούς περιορισμούς του κυκλώματος
- Ο σχεδιαστής μπορεί να επιλέξει μία συγκεκριμένη υλοποίηση ενός αριθμητικού κυκλώματος

Πρόσθεση, αφαίρεση

56

- Διαφορετικές υλοποιήσεις:
  - minimum area, minimum delay
- Αρχιτεκτονικές αθροιστών:
  - Ripple carry
  - Carry-Look-Ahead
  - Brent-Kung

Πολλαπλασιασμός

57

- Διαφορετικές υλοποιήσεις:
  - signed, unsigned multiplication
  - minimum area, minimum delay
- Αρχιτεκτονικές πολλαπλασιαστών:
  - Carry-save, carry-propagate array
  - Booth-recoded
  - Wallace tree

Διαίρεση

58

- Η πράξη της διαίρεσης είναι συνθέσιμη μόνο όταν ο διαιρέτης είναι δύναμη του 2
  - υλοποιείται παρόμοια με την πράξη της ολίσθησης
- Οι διαιρέτες είναι πολύ μεγάλα και πολύπλοκα κυκλώματα:
  - κάποια βιβλιοθήκες παρέχουν κυκλώματα διαίρεσης (π.χ. Synopsys' DesignWare)

Modulo

59

- Η πράξη modulo είναι συνθέσιμη μόνο όταν το δεύτερο τελούμενο είναι δύναμη του 2
- a mod b, a mod 5
  - μη συνθέσιμες πράξεις
- a mod 4
  - κανένα λογικό κύκλωμα: τα MSBs τίθενται στο μηδέν και τα 2 LSBs διατηρούν την τιμή τους

Exponent

60

- Γενικά, η πράξη exp είναι συνθέσιμη μόνο όταν το δεύτερο τελούμενο είναι 2 (x**2 = x*x)
  - χρήση κυκλώματος πολλαπλασιασμού

Σύντμηση (concatenation)

61

- Δεν παράγεται κύκλωμα από την σύνθεση
  - συγχώνευση αγωγών (buses)

```
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;
```

RTL Schematic

Ενότητα 3

62

- Παράλληλη (concurrent) VHDL vs. Ακολουθιακή (sequential) VHDL
- Διεργασία (process)
  - Συνδυαστικές διεργασίες

Παράλληλο πεδίο (concurrent domain) της VHDL

63

- Οι εντολές που βρίσκονται στο παράλληλο πεδίο εκτελούνται ταυτόχρονα
- Παραδείγματα παράλληλων εντολών:
  - καταχώρηση σήματος (signal assignment)
  - καταχώρηση σήματος υπό συνθήκη (when-else)
  - καταχώρηση σήματος με επιλογή (with-select)
  - διεργασία (process)

Ακολουθιακό πεδίο (sequential domain) της VHDL

64

- Οι εντολές που βρίσκονται στο ακολουθιακό πεδίο εκτελούνται με την σειρά
  - όπως στις γλώσσες προγραμματισμού
- Το πιο ισχυρό κομμάτι της γλώσσας
- Παραδείγματα ακολουθιακών εντολών:
  - καταχώρηση σήματος (signal assignment)
  - εντολές εκτέλεσης υπό συνθήκη (if-then-else)
  - εντολές εκτέλεσης με επιλογή (case)
  - εντολές loop

Παράλληλη και ακολουθιακή VHDL

65

```
architecture rtl of ex is
    concurrent declaration part
begin
    concurrent VHDL
    process(…)
        sequential declaration part
    begin
        sequential VHDL
    end process;
    concurrent VHDL
end architecture rtl;
```

Process = concurrent statement

Παράλληλη και ακολουθιακή VHDL: δηλώσεις

66

- Παράλληλη VHDL
  - δήλωση σημάτων
- Ακολουθιακή VHDL
  - δήλωση μεταβλητών
- Παράλληλη/ακολουθιακή
  - δήλωση τύπων, σταθερών

Παράλληλη και ακολουθιακή VHDL: εντολές

67

- Παράλληλη VHDL
  - process
  - component
  - when-else, with-select
- Παράλληλη/ακολουθιακή
  - καταχώρηση σημάτων
  - κλήση συναρτήσεων
  - assertion, report
- Ακολουθιακή VHDL
  - καταχώρηση μεταβλητών
  - if-then-else
  - case
  - wait
  - loop
  - exit
  - next
  - null

Παράλληλη και ακολουθιακή VHDL: σύνθεση

68

- Οι όροι παράλληλη και ακολουθιακή εκτέλεση εντολών αφορούν την προσομοίωση
  - το υλικό εκτελεί παράλληλα
- Παράλληλη VHDL:
  - εύκολη η μετάφραση των παράλληλων εντολών
- Ακολουθιακή VHDL:
  - δύσκολη η μετάφραση των ακολουθιακών εντολών
  - δεν είναι όλες οι ακολουθιακές εντολές συνθέσιμες
    - δεν υπάρχει ισοδύναμο υλικό
  - για να είναι συνθέσιμη η ακολουθιακή VHDL πρέπει να τηρηθούν κάποιοι κανόνες στον κώδικα (coding rules)

Διεργασία (process)

69

```
process sensitivity list
    declaration part
begin
    statement part
end process;
```

- Λίστα ευαισθησίας (sensitivity list)
  - λίστα σημάτων στα οποία η διεργασία είναι ευαίσθητη
- Τμήμα δηλώσεων (declaration part)
  - τοπικές μεταβλητές, δεν είναι ορατές έξω από την διεργασία
- Τμήμα εντολών (statement part)
  - περιέχει τις ακολουθιακές εντολές

Modulo

70

- Η πράξη modulo είναι συνθέσιμη μόνο όταν το δεύτερο τελούμενο είναι δύναμη του 2
- a mod b, a mod 5
  - μη συνθέσιμες πράξεις
- a mod 4
  - κανένα λογικό κύκλωμα: τα MSBs τίθενται στο μηδέν και τα 2 LSBs διατηρούν την τιμή τους

Exponent

71

- Γενικά, η πράξη exp είναι συνθέσιμη μόνο όταν το δεύτερο τελούμενο είναι 2 (x**2 = x*x)
  - χρήση κυκλώματος πολλαπλασιασμού

Σύντμηση (concatenation)

72

- Δεν παράγεται κύκλωμα από την σύνθεση
  - συγχώνευση αγωγών (buses)

```
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;
```

RTL Schematic

Λίστα ευαισθησίας (sensitivity list)

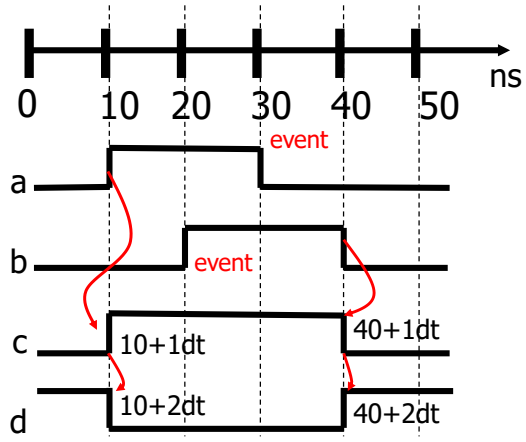
73

- Οι διεργασίες εκτελούνται σαν ένας ατέρμονος βρόγχος
  - όταν τελειώσει η εκτέλεση των εντολών της διεργασίας, η προσομοίωση αρχίζει από την αρχή
- Η εκτέλεση των εντολών της διεργασίας ξεκινάει όταν συμβεί ένα γεγονός (event) σε ένα από τα σήματα της λίστας ευαισθησίας
  - γεγονός = αλλαγή τιμής
  - όταν τελειώσει η εκτέλεση των εντολών της διεργασίας, η διεργασία «σταματά» μέχρι να συμβεί ένα γεγονός στην λίστα ευαισθησίας

Μοντέλο προσομοίωσης βασιζόμενο σε γεγονότα

74

- Event-driven VHDL simulation




```
c <= a or b;
d <= not c;
```

Μοντέλο προσομοίωσης βασιζόμενο σε γεγονότα

75

```
P1: process (R,Qn)
begin
  Q <= R nor Qn;
end process;

P2: process (S,Q)
begin
  Qn <= S nor Q;
end process;
```

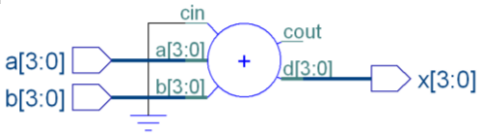
	R	S	Q	Qn
t	1	0	1	0
t+1dt	1	0	0	0
t+2dt	1	0	0	1
t+3dt	1	0	0	1

Συνδυαστική διεργασία (combinational process)

76

```
entity adder is
  port (
     a,b : in integer range 0 to 15;
     x   : out integer range 0 to 15
     );
end entity adder;

architecture beh of adder is
begin

  p: process (a,b)
  begin
    x <= a + b;
  end process;

end architecture beh;
```

RTL Schematic



Όλες οι εισόδοι του συνδυαστικού κυκλώματος πρέπει να είναι στην λίστα ευαισθησίας της διεργασίας

Συνδυαστική διεργασία (combinational process)

77

```
entity adder is
  port (
     a,b : in integer range 0 to 15;
     x   : out integer range 0 to 15
     );
end entity adder;

architecture beh of adder is
begin

  p: process (a)
  begin
    x <= a + b;
  end process;

end architecture beh;
```

Προσοχή:

- Το κύκλωμα είναι συνθέσιμο, αλλά ...
- Διαφορά μεταξύ του μοντέλου προσομοίωσης και του μοντέλου σύνθεσης

Wait

78

```
p: process (a, b)
  begin
    x <= a + b;
  end process;
```

```
p: process
  begin
    x <= a + b;
    wait on a, b;
  end process;
```

Ισοδύναμες διεργασίες

Καταχώρηση σημάτων

79

```
p1: process (din)
begin
        sum1 <= din + 1;
        sum2 <= sum1 + 1;
end process;
```

```
p2: process (din)
begin
        sum2 <= sum1 + 1;
        sum1 <= din + 1;
end process;
```

- Τα σήματα (signals) χρησιμοποιούνται για την επικοινωνία μεταξύ διεργασιών
- Οι διεργασίες p1 & p2 είναι ισοδύναμες. Γιατί;
- Καταχωρήσεις σημάτων υπό συνθήκη (when-else) και με επιλογή (with-select) δεν μπορούν να χρησιμοποιηθούν στις διεργασίες

Σήματα έναντι μεταβλητών

80

```
signal sum1, sum2: integer;
…
p1: process (din)
begin
        sum1 <= din + 1;
        sum2 <= sum1 + 1;
end process;
```

Time	din	Sum1	Sum2
…	0	0	0
t1	1	0	0
t1+1dt	1	2	1
t2	2	2	1
t2+1dt	2	3	3

```
p2: process (din)
variable sum1,sum2: integer;

begin
        sum1 := din + 1;
        sum2 := sum1 + 1;
end process;
```

Time	din	Sum1	Sum2
…	0	0	0
t1	1	2	3
t1+1dt	1	2	3
t2	2	3	4
t2+1dt	2	3	4

Εντολή if

81

```
entity compare is
  port (a, b : in bit_vector(3 downto 0);
        equal : out bit);
end entity compare;

architecture beh of compare is
begin
 p: process (a,b)
 begin
   if a = b then
     equal <= '1';
   else
     equal <= '0';
   end if;
 end process;
end architecture beh;
```

- Ποιά είναι η ισοδύναμη παράλληλη εντολή;
- Ποιό είναι το αποτέλεσμα της σύνθεσης;

Εντολή if με πολλαπλές διακλαδώσεις

82

```
entity compare is
  port (a,b : in bit_vector(3 downto 0);
        result : out bit_vector(1 downto 0));
end entity compare;

architecture beh of compare is
begin
 p: process (a,b)
 begin
   if a = b then
     result <= "00";
   elsif a < b then
     result <= "01";
   else
     result <= "10";
   end if;
 end process;
end architecture beh;
```

- Ποιά είναι η ισοδύναμη παράλληλη εντολή;
- Ποιό είναι το αποτέλεσμα της σύνθεσης;

Εντολή case

83

```
entity seven_seg is
  port(
      din  : in bit_vector(3 downto 0);
      seg : out bit_vector(6 downto 0)
      );
end entity seven_seg;

begin

p: process (din)
begin
  case din is
   when "0000" => seg <= "1111110";
   when "0001" => seg <= "0110000";
   when "0010" => seg <= "1101101";
   when "0011" => seg <= "1111001";
                                continue…
```

```
…continue
   when "0100" => seg <= "0110011";
   when "0101" => seg <= "1011011";
   when "0110" => seg <= "1011111";
   when "0111" => seg <= "1110000";
   when "1000" => seg <= "1111111";
   when "1001" => seg <= "1111011";
   when others => seg <= "0000000";
 end case;
end process;

end architecture beh_case;
```

- Ποιά είναι η ισοδύναμη παράλληλη εντολή;
- Ποιό είναι το αποτέλεσμα της σύνθεσης;

Ελλιπής εντολή if

84

```
entity inc_if is
  port (a, b, en : in bit;
        z : out bit);
end entity inc_if;

architecture beh of inc_if is
begin

p: process (a,b,en)
begin
  if en = '1' then
     z <= b;
  end if;
end process;

end architecture beh;
```

- Πολύ συχνό σχεδιαστικό λάθος
- Δεν περιγράφει συνδυαστική λογική
- Υπάρχει ανάδραση στο κύκλωμα
- Το εργαλείο σύνθεσης δεν μπορεί να γνωρίζει το λάθος και εισάγει ανάδραση

Ελλιπής εντολή if

85

```
entity inc_if is
  port (a, b, en : in bit;
        z, y : out bit);
end entity inc_if;

architecture beh of inc_if is
begin
p: process (a,b,en)
begin
  if en = '1' then
     z <= a;
  else
     y <= b;
  end if;
end process;
end architecture beh;
```

- Πολύ συχνό σχεδιαστικό λάθος
- Δεν καταχωρείται τιμή για κάποιο σήμα σε κάποια διακλάδωση
- Δεν περιγράφει συνδυαστική λογική
- Υπάρχει ανάδραση στο κύκλωμα

Ελλιπής εντολή if

86

```
p: process (a,b,en)
begin
  if en = '1' then
     z <= b;
  else
     z <= a;
  end if;
end process;
```

```
p: process (a,b,en)
begin

  z <= a;
  if en = '1' then
     z <= b;
  end if;
end process;
```

- Δύο σχεδιαστικές τεχνικές για την αποφυγή των ελλιπών if εντολών
  - καταχώρηση όλων των σημάτων σε όλες τις διακλαδώσεις και χρήση της διακλάδωσης else
  - default values

Ενότητα 4

87

- Ακολουθιακές διεργασίες
  - Πρότυπα μανταλωτών (latches), flip-flops
- Μοντελοποίηση βασικών ακολουθιακών κυκλωμάτων
  - Μετρητές (counters)
  - Καταχωρητές ολίσθησης (shift registers)

Περιγραφή ενός καταχωρητή στην VHDL

88

- Η περιγραφή ενός καταχωρητή στην VHDL γίνεται μόνο με την χρήση διεργασίας (process)
- Δεν ορίζεται δομή της VHDL που να αντιστοιχίζεται στο υλικό σαν καταχωρητής
- Υπάρχουν πολλοί τρόποι για να περιγράψεις την συμπεριφορά ενός καταχωρητή στη VHDL (simulation model)
  - Δεν είναι όλες οι περιγραφές συνθέσιμες

Πρότυπα σύνθεσης καταχωρητή στην VHDL

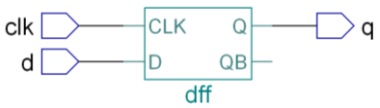
89

- Τα εργαλεία σύνθεσης αναγνωρίζουν συγκεκριμένες VHDL περιγραφές ενός καταχωρητή
  - πρότυπα σύνθεσης (synthesis templates)
- Τα πρότυπα σύνθεσης μπορεί να διαφέρουν μεταξύ των εργαλείων σύνθεσης

Πρότυπο καταχωρητή

90

```
entity dff is
  port (clk, d : in bit;
        q : out bit);
end entity;

architecture bef of dff is
begin

 process
 begin
   wait until clk'event and clk = '1';
   q <= d;
 end process;

end architecture;
```



Πρότυπα καταχωρητή με if και λίστα ευαισθησίας

91

```
process (clk)
begin
 if clk'event and clk = '1' then
   q <= d;
 end if;
end process;
```

Ανοδική & καθοδική ακμή του ρολογιού

92

```
process (clk)
begin
 if clk'event and clk = '1' then
   q <= d;
 end if;
end process;
```

Rising edge

```
process (clk)
begin
 if clk'event and clk = '0' then
   q <= d;
 end if;
end process;
```

Falling edge

Σήμα επίτρεψης των δεδομένων (data enable)

93

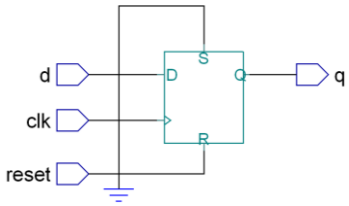
```
process (clk)
begin
 if clk'event and clk = '1' then
   if en = '1' then
     q <= d;
   end if;
 end if;
end process;
```

```
process (clk)
begin
 if clk'event and clk = '1'  and en = '1' then
   q <= d;
 end if;
end process;
```

Πρότυπο καταχωρητή με ασύγχρονη είσοδο μηδενισμού

94

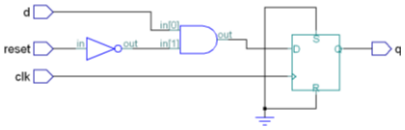
```
process (clk, reset) is
begin
  if reset = '1' then
    q <= '0';
  elsif clk'event and clk = '1'  then
    q <= d;
  end if;
end process;
```



Πρότυπο καταχωρητή με σύγχρονη είσοδο μηδενισμού

95

```
process (clk) is
begin
  if clk'event and clk = '1' then
    if reset = '1' then
       q <= '0';
    else
       q <= d;
    end if;
  end if;
end process;
```



Σχεδίαση ακολουθιακής λογικής

96

```
process (clk)
begin
 if clk'event and clk = '1' then
   q1 <= a and b;
   q2 <= c or d;
 end if;
end process;
```

Ποιο είναι το αποτέλεσμα της σύνθεσης;

Σχεδίαση ακολουθιακής λογικής

97

```
process (clk)
begin
 if clk'event and clk = '1' then
   q1 <= a;
   q2 <= b;
   q3 <= q1 and q2;
 end if;
end process;
```

Ποιο είναι το αποτέλεσμα της σύνθεσης;

Πώς υπολογίζεται η συχνότητα λειτουργίας του κυκλώματος;

Σχεδίαση ακολουθιακής λογικής

98

```
entity dreg8 is
 port (clk : in bit;
       d : in bit_vector(7 downto 0);
       q : out bit_vector(7 downto 0));
end entity;

architecture bef of dreg8 is
begin

 process (clk)
 begin
  if clk'event and clk = '1' then
    q <= d;
  end if;
 end process;

end architecture;
```

Ποιο είναι το αποτέλεσμα της σύνθεσης;

Σχεδίαση ακολουθιακής λογικής

99

```
process (clk, reset)
 begin
  if reset = '1' then
    q <= (others => '0');
  elsif clk'event and clk = '1' then
    q <= d;
  end if;
 end process;
```

```
process (clk, reset)
 begin
  if reset = '1' then
    q <= "00001111";
  elsif clk'event and clk = '1' then
    q <= d;
  end if;
 end process;
```

Ποιο είναι το αποτέλεσμα της σύνθεσης;

Σχεδίαση μετρητών (counters)

100

```
entity counter4 is
 port (clk, reset : in bit;
       count : out integer range 0 to 15);
end entity;

architecture bef of counter4 is
signal counter : integer range 0 to 15;
begin

count <= counter;
 process (clk, reset)
 begin
  if reset = '1' then
     counter <= 0;
  elsif clk'event and clk = '1' then
     counter <= (counter + 1) mod 16;
  end if;
 end process;
end architecture;
```

4-bit σύγχρονος δυαδικός μετρητής με ασύγχρονη είσοδο μηδένισης

Σχεδίαση μετρητών (counters)

101

- Υλοποιήστε του ακόλουθους τύπους μετρητών:
  - BCD μετρητές
  - με σύγχρονη είσοδο μηδένισης
  - με είσοδο επίτρεψης
  - με παράλληλη φόρτωση
- Υλοποιήστε διαιρέτες συχνότητας με την χρήση μετρητών

Ενότητα 5

102

- Ιεραρχική σχεδίαση στη VHDL
  - Πως σχεδιάζουμε ιεραρχικά
  - Πως υλοποιούμε παραμετροποιημένες μονάδες
  - Πως χρησιμοποιούνται οι εντολές generate

Σημασία της ιεραρχικής σχεδίασης

- Σχεδίαση με βάση την μέθοδο «διαίρει & βασίλευε» (divide & conquer)
  - η σχεδίαση τεμαχίζεται σε υπομονάδες & η πολυπλοκότητα απλοποιείται
  - κάθε υπομονάδα σχεδιάζεται και πιστοποιείται ξεχωριστά
- Ενσωμάτωση διαθέσιμων μονάδων στη σχεδίαση
  - μειώνεται ο χρόνος σχεδίασης
  - αυξάνεται η πιστότητα της σχεδίασης

Δομική σχεδίαση (structural description)

- Η ιεραρχική σχεδίαση βασίζεται στη δομική σχεδίαση
- Δομική σχεδίαση = περιγράφει τη σύνδεση των μονάδων (components) του κυκλώματος
  - συνδυασμός μικρότερων μονάδων και περιγραφή της διασύνδεσής τους για τη δημιουργία μίας μεγαλύτερης μονάδας
  - παρόμοια με το σχηματικό διάγραμμα (schematic diagram)
  - το διάγραμμα δικτύωσης ενός κυκλώματος (circuit netlist) αποτελεί δομική περιγραφή

Δομική σχεδίαση στη VHDL

105

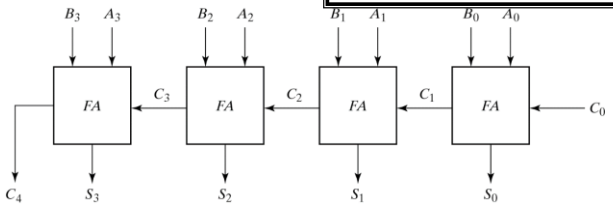
- Βασίζεται στη χρήση μονάδων (components)
- Για τη χρήση μίας μονάδας απαιτείται:
  - δήλωση μονάδας (component declaration)
  - καθορισμός διαμόρφωσης (configuration specification)
  - στιγμιότυπα μονάδας (component instances)

Δομική σχεδίαση στη VHDL: παράδειγμα ...

106

```
entity Full_adder is
  port (A, B, Cin  : in bit;
        Sum, Cout : out bit);
end entity Full_adder;

architecture beh of Full_adder is
 ...
end architecture beh;
```

```
entity ripple_adder is
  port (a, b  : in bit_vector(3 downto 0);
        cin   : in bit;
        sum   : out bit_vector(3 downto 0);
        cout : out bit);
end entity ripple_adder;

architecture struct of ripple_adder is
...
end architecture struct;
```



Δομική σχεδίαση στη VHDL: ... παράδειγμα

107

```
architecture struct of ripple_adder is

component full_adder
 port (a, b, cin  : in bit;
       sum, cout : out bit);
end component;

for all : full_adder use entity work.Full_adder (beh)
          port map (A=>a, B=>b, Cin=>cin, Sum=>sum, Cout=>cout);

signal c1,c2,c3 : bit;
begin
  bit0: full_adder port map (a=>a(0), b=>b(0), cin=>cin, sum=>sum(0), cout=>c1);
  bit1: full_adder port map (a=>a(1),  b=>b(1), cin=>c1,   sum=>sum(1),  cout=>c2);
  bit2: full_adder port map (a=>a(2), b=>b(2), cin=>c2,  sum=>sum(2), cout=>c3);
  bit3: full_adder port map (a=>a(3), b=>b(3), cin=>c3,  sum=>sum(3), cout=>cout);

end architecture struct;
```

Component declaration

Configuration specification

Component instances

Στιγμιότυπα μονάδας (component instances)

108

- Παράλληλες εντολές (concurrent statements)
- Port map:
  - Συσχέτιση ονόματος (named association)

bit3: full_adder **port map** (a=>a(3), b=>b(3),..., cout=>cout);

  - Συσχέτιση θέσης (positional association)

bit3: full_adder **port map** (a(3), b(3), c3, sum(3), cout);

  - Ασύνδετες θύρες (unconnected ports)

bit3: full_adder **port map** (a=>a(3), b=>b(3),..., cout=>open);

Ενότητα 6

- Testbenches στη VHDL

Testbenches

- Χρησιμοποιούνται για την εξομοίωση της σχεδίασης
- Ένα testbench είναι μία οντότητα:
  - χωρίς εισόδους και εξόδους
  - περιλαμβάνει στιγμιότυπο της μονάδας υπό έλεγχο (design under test, DUT)
  - εφαρμόζει ακολουθίες εισόδων στη μονάδα
  - παρακολουθεί τις εξόδους της μονάδας

Testbench: παράδειγμα ...

111

```
entity tb_fa is
end entity tb_ra;

architecture bench of tb_fa is

component full_adder
 port (a, b, cin  : in bit;
       sum, cout : out bit);
end component;

 signal a, b, cin, sum, cout  : bit;

begin
  bit0: full_adder port map (a=>a, b=>b, cin=>cin, sum=>sum, cout=>cout);
  ...
```

Testbench: ... παράδειγμα

112

```
...
stimulus : process is
   begin
      a <= '0';  b <= '0';  cin <= '0'; wait for 20 ns;
      a <= '0';  b <= '0';  cin <= '1'; wait for 20 ns;
      a <= '0';  b <= '1';  cin <= '0'; wait for 20 ns;
      a <= '0';  b <= '1';  cin <= '1'; wait for 20 ns;
      ...
      wait;
   end process stimulus;
...
```

Testbench: … παράδειγμα


```
verify : process is
  begin
    wait for 20 ns;
    assert a = '0' and b = '0' and cout = '0'
      report "error response on carry out"
      severity error;
    wait on a, b, cout;
  end process verify;

end architecture bench;
```

Ενότητα 7

- Μηχανές πεπερασμένων καταστάσεων (finite state machines) στη VHDL
  - πρότυπα σχεδίασης
  - μηχανές Moore & Mealy
  - σχεδιαστικές συμβουλές

Μηχανή Πεπερασμένων Καταστάσεων

- Finite State Machine (FSM):
  - Μια μηχανή η οποία έχει πεπερασμένο (finite) αριθμό καταστάσεων
  - Χαρακτηρίζεται από:
    - Ένα σύνολο γεγονότων εισόδου
    - Ένα σύνολο γεγονότων εξόδου
    - Ένα σύνολο καταστάσεων
    - Μια συνάρτηση που αντιστοιχίζει τις καταστάσεις και την είσοδο στην έξοδο
    - Μια συνάρτηση που αντιστοιχίζει τις καταστάσεις και την είσοδο στην επόμενη κατάσταση
  - Χρησιμοποιείται σε σχεδίαση κυκλωμάτων ελέγχου

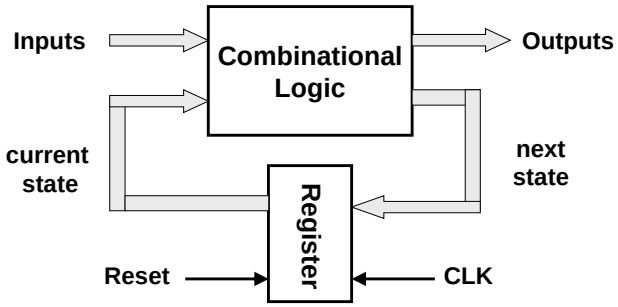
FSM: Τύποι μηχανών

- Μηχανή Moore
  - Οι έξοδοι εξαρτώνται μόνο από την παρούσα κατάσταση της μηχανής
- Μηχανή Mealy
  - Οι έξοδοι εξαρτώνται από τις εισόδους και την παρούσα κατάσταση της μηχανής

FSM: Μοντελοποίηση

117

- Μοντελοποιείται στη VHDL με:
  - ένα μπλοκ συνδυαστικής λογικής
    - υλοποιεί τις συναρτήσεις
  - και ένα μπλοκ καταχωρητών
    - αποθηκεύει τις καταστάσεις



FSM: Σύνθεση

118

- Τα εργαλεία σύνθεσης εκτελούν βελτιστοποίηση των καταστάσεων των FSMs
- Οι καταστάσεις αναπαριστώνται από έναν τύπο απαρίθμησης
  - τα εργαλεία σύνθεσης εκμεταλλεύονται τον τύπο απαρίθμησης για αποδοτικότερη βελτιστοποίηση
- Τα εργαλεία σύνθεσης υποστηρίζουν διαφορετικές μορφές κωδικοποίησης των καταστάσεων:
  - binary, onehot, gray, user-defined

FSM: VHDL περιγραφή

119

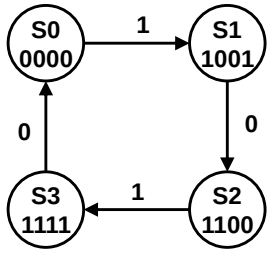
- Πόσες διεργασίες θα χρησιμοποιήσουμε για την περιγραφή της μηχανής;
- Πώς θα περιγράψουμε τις μεταβάσεις των καταστάσεων της μηχανής;
- Πώς θα καθορίσουμε τις εξόδους της μηχανής;
- Πώς θα αρχικοποιήσουμε την μηχανή;
- Πώς θα εξασφαλίσουμε ότι η μηχανή έχει μηχανισμό επανεκκίνησης (re-entrant);

Μηχανή Moore: παράδειγμα

120

```
entity moore_machine is
  port (reset : in std_ulogic;
        clk : in std_ulogic;
        in1 : in std_ulogic;
        out1 : out std_ulogic_vector(3 downto 0));
end entity moore_machine;

architecture beh of moore_machine is

type state_type is (s0,s1,s2,s3);
signal state : state_type;

begin
  ...
end architecture;
```

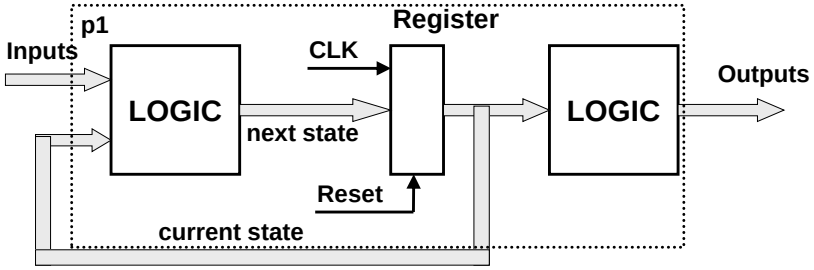
Κωδικοποίηση καταστάσεων



Υλοποίηση με μία διεργασία

121

- Μία διεργασία που περιγράφει:
  - τους καταχωρητές,
  - τις μεταβάσεις των καταστάσεων και
  - τις εξόδους



Υλοποίηση με μία διεργασία: VHDL κώδικας

122

```
p1: process (reset,clk)
begin

  if (reset = '1') then
    state <= s0;
    out1 <= "0000";

  elsif clk'event and clk = '1' then
    case state is
      when s0 =>
        if in1 = '1' then
          state <= s1;
        end if;
        out1 <= "0000";
      when s1 =>
        if in1 = '0' then
          state <= s2;
        end if;
        out1 <= "1001";
                                  continue ...
```

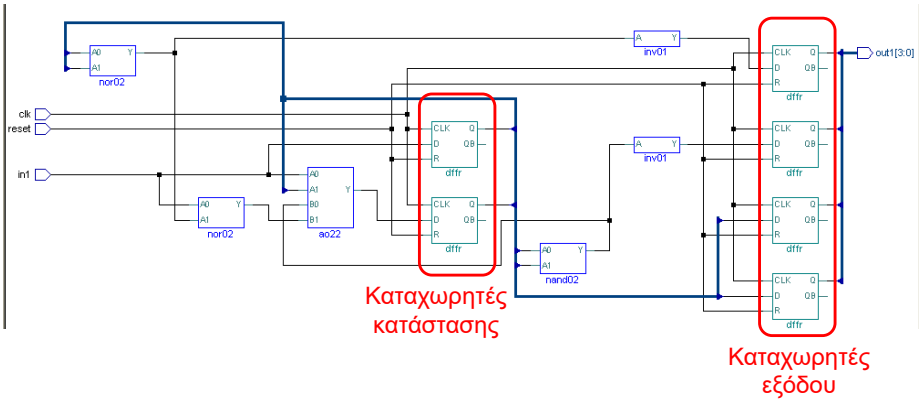
Αρχικοποίηση μηχανής

Ένα σήμα περιγράφει την παρούσα και την επόμενη κατάσταση

```
... continue
      when s2 =>
        if in1 = '1' then
          state <= s3;
        end if;
        out1 <= "1100";
      when s3 =>
        if in1 = '0' then
          state <= s0;
        end if;
        out1 <= "1111";
    end case;
  end if;

end process;
```

Υλοποίηση με μία διεργασία: συνθέσιμο κύκλωμα

123

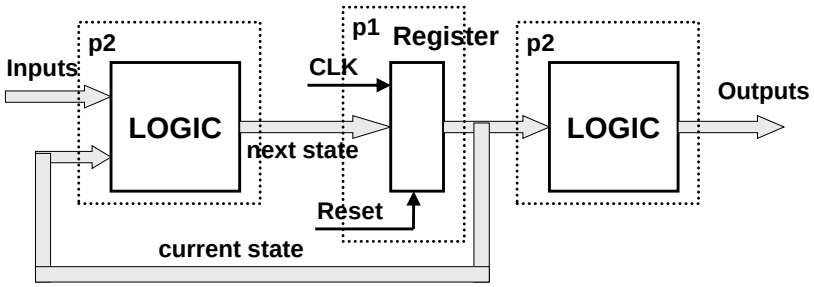


- Δυαδική κωδικοποίηση των καταστάσεων (binary encoding)
- Οι έξοδοι προέρχονται από καταχωρητές (registered outputs)

Υλοποίηση με δύο διεργασίες

124

- Μία ακολουθιακή διεργασία (p1) που περιγράφει τους καταχωρητές και
- Μία διεργασία (p2) που περιγράφει την συνδυαστική λογική: μεταβάσεις καταστάσεων και εξόδους



Υλοποίηση με δύο διεργασίες: VHDL κώδικας

125

```
...
type state_type is (s0,s1,s2,s3);
signal current_state : state_type;
signal next_state : state_type;
...

p1: process (reset,clk)
begin

  if (reset = '1') then
     current_state <= s0;
  elsif clk'event and clk = '1' then
     current_state <= next_state;
  end if;

end process;
```

Δύο σήματα τύπου state_type

Αλλαγή κατάστασης

```
p2: process (current_state,in1)
begin

  next_state <= current_state;

  case current_state is
    when s0 =>
      if in1 = '1' then
        next_state <= s1;
      end if;
      out1 <= "0000";
...
    when s3 =>
      if in1 = '0' then
        next_state <= s0;
      end if;
      out1 <= "1111";
  end case;

end process;
```

Για την αποφυγή latches

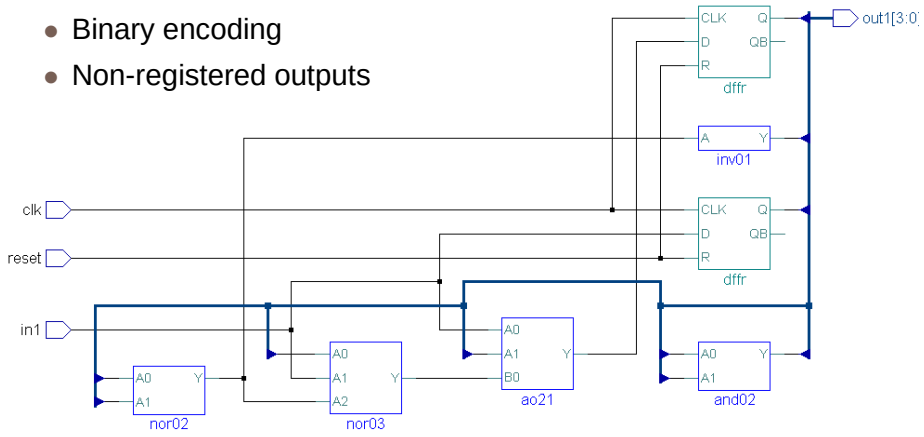
Επιλογή επόμενης κατάστασης

Η έξοδος εξαρτάται μόνο από την παρούσα κατάσταση

Υλοποίηση με δύο διεργασίες: συνθέσιμο κύκλωμα

126

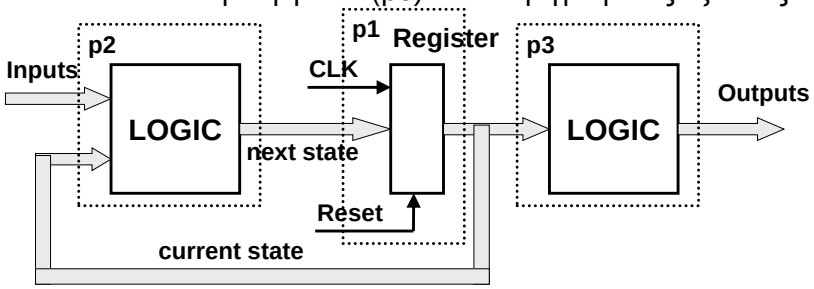
- Binary encoding
- Non-registered outputs



Υλοποίηση με τρεις διεργασίες

127

- Μία ακολουθιακή διεργασία (p1) που περιγράφει τους καταχωρητές,
- Μία συνδυαστική διεργασία (p2) που περιγράφει τις μεταβάσεις καταστάσεων και
- Μία συνδυαστική διεργασία (p3) που περιγράφει τις εξόδους



Υλοποίηση με τρεις διεργασίες: VHDL κώδικας

128

```
p1: process (reset,clk)
begin

  if (reset = '1') then
     current_state <= s0;
  elsif clk'event and clk = '1' then
     current_state <= next_state;
  end if;
end process;

p3: process (current_state)
begin

  case current_state is
     when s0 => out1 <= "0000";
     when s1 => out1 <= "1001";
     when s2 => out1 <= "1100";
     when s3 => out1 <= "1111";
  end case;
end process;
```

Η έξοδος εξαρτάται μόνο από την παρούσα κατάσταση

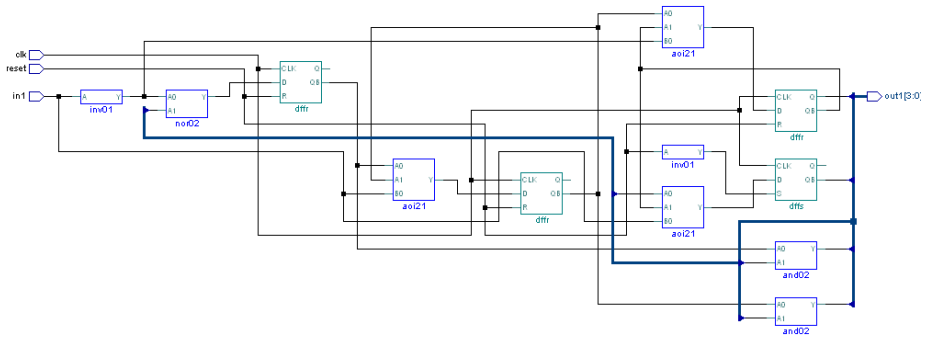
```
p2: process (current_state,in1)
begin

  next_state <= current_state;

  case current_state is
     when s0 =>
        if in1 = '1' then
           next_state <= s1;
        end if;
...
     when s3 =>
        if in1 = '0' then
           next_state <= s0;
        end if;
  end case;

end process;
```

Για την αποφυγή latches

Επιλογή επόμενης κατάστασης

Υλοποίηση με τρεις διεργασίες: συνθέσιμο κύκλωμα

129



- One-hot encoding
- Non-registered outputs

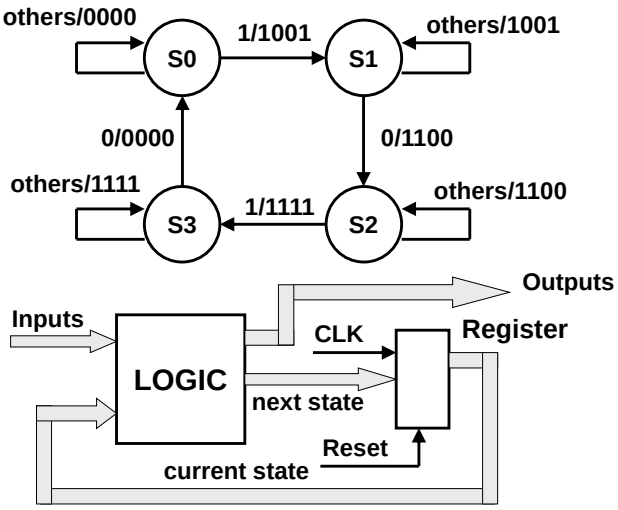
Σχόλια

130

- Προτιμάτε τις υλοποιήσεις με 2 ή 3 διεργασίες
  - ξεχωρίζουν την ακολουθιακή από τη συνδυαστική λογική της μηχανής
  - επιτρέπουν να χρησιμοποιηθεί η έξοδος της συνδυαστικής λογικής σε άλλες διεργασίες ελέγχου ή στον καθορισμό άλλων εξόδων
- Η υλοποίηση με 3 διεργασίες επιτρέπει την εισαγωγή ή όχι καταχωρητών στις εξόδους, ανάλογα με τον τύπο της τρίτης διεργασίας:
  - ακολουθιακή διεργασία: registered outputs
  - συνδυαστική διεργασία: non-registered outputs

Μηχανή Mealy: παράδειγμα

131



Υλοποίηση με δύο διεργασίες

132

```
...
type state_type is (s0,s1,s2,s3);
signal curent_state : state_type;
signal next_state : state_type;
...
p1: process (reset,clk)
begin

  if (reset = '1') then
     current_state <= s0;
  elsif clk'event and clk = '1' then
     current_state <= next_state;
  end if;

end process;
```

```
p2: process (current_state,in1)
begin

  next_state <= current_state;

  case current_state is
    when s0 =>
      if in1 = '1' then
        next_state <= s1;
        out1 <= "1001";
      else
        out1 <= "0000";
      end if;
...
    end case;

end process;
```

Η έξοδος εξαρτάται από την παρούσα κατάσταση και την είσοδο

Υλοποίηση με τρεις διεργασίες

133

```
p1: process (reset,clk)
begin
 if (reset = '1') then
   current_state <= s0;
 elsif clk'event and clk = '1' then
   current_state <= next_state;
 end if;
end process;

p3: process (current_state,in1)
begin
 case current_state is
   when s0 =>
     if in1 = '1' then
       out1 <= "1001";
     else
       out1 <= "0000";
     end if;
  ...
end process;
```

Η έξοδος εξαρτάται από την παρούσα κατάσταση και την είσοδο

```
p2: process (current_state,in1)
begin

 next_state <= current_state;

 case current_state is
   when s0 =>
     if in1 = '1' then
       next_state <= s1;
     end if;
...
   when s3 =>
    if in1 = '0' then
      next_state <= s0;
    end if;

 end case;

end process;
```

Κωδικοποίηση των καταστάσεων από τον χρήστη

134

- Ο χρήστης μπορεί να καθορίσει την κωδικοποίηση των καταστάσεων, π.χ:

```
constant s0: std_ulogic_vector(1 downto 0) :="00";
constant s1: std_ulogic_vector(1 downto 0) :="01";
constant s2: std_ulogic_vector(1 downto 0) :="10";
constant s3: std_ulogic_vector(1 downto 0) :="11";
signal cur_state, next_state : std_ulogic_vector(1 downto 0);
```

- Προτιμάτε τη χρήση των τύπων απαρίθμησης
  - παρέχει στο εργαλείο σύνθεσης μεγαλύτερη ευχέρεια στη βελτιστοποίηση της μηχανής
  - δηλώστε την αρχική κατάσταση της μηχανής στην αριστερή θέση του τύπου απαρίθμησης

Αχρησιμοποίητες καταστάσεις

- Μία μηχανή μπορεί να έχει αχρησιμοποίητες καταστάσεις
  - π.χ. σε μία μηχανή 5 καταστάσεων
  - S0="000", S1="001", S2="010", S3="011", S4="100"
  - αχρησιμοποίητες καταστάσεις: "101", "110", "111"
- Τι θα συμβεί εάν η μηχανή εισέλθει σε μία αχρησιμοποίητη κατάσταση;
  - η συμπεριφορά της μηχανής δεν είναι προβλέψιμη
  - μπορεί να «κλειδώσει» σε μη-προβλέψιμη κατάσταση
- Σε κρίσιμες σχεδιάσεις πρέπει να προβλέπεται μηχανισμός επανεκκίνησης (re-entrant FSM)

Σχεδίαση ασφαλών μηχανών κατάστασης

- Εάν η μηχανή βρεθεί σε μία αχρησιμοποίητη κατάσταση πρέπει να επανέλθει σε μία γνωστή κατάσταση (π.χ. κατάσταση αρχικοποίησης)
- Συμβουλές:
  - Φροντίστε ο αριθμός των καταστάσεων (τύπος απαρίθμησης) να είναι δύναμη του 2
  - Στην εντολή **case** που επιλέγει την επόμενη κατάσταση, χρησιμοποιήστε την εντολή **when others** ώστε η μηχανή να επιστρέψει σε μία γνωστή κατάσταση
- Σε κρίσιμες σχεδιάσεις προτιμάτε την δυαδική κωδικοποίηση και όχι την κωδικοποίηση onehot

Μηχανή πεπερασμένων καταστάσεων: παράδειγμα

- Σχεδιάστε ένα κύκλωμα που ανιχνεύει μία συγκεκριμένη ακολουθία ψηφίων (υπογραφή) σε μία σειριακή είσοδο δεδομένων
  - να υλοποιεί το διάγραμμα καταστάσεων