Σχεδίαση Ψηφιακών Κυκλωμάτων

Η γλώσσα περιγραφής υλικού VHDL - Μέρος I

Νικόλαος Καββαδίας
nkavv@uop.gr

01 Δεκεμβρίου 2010

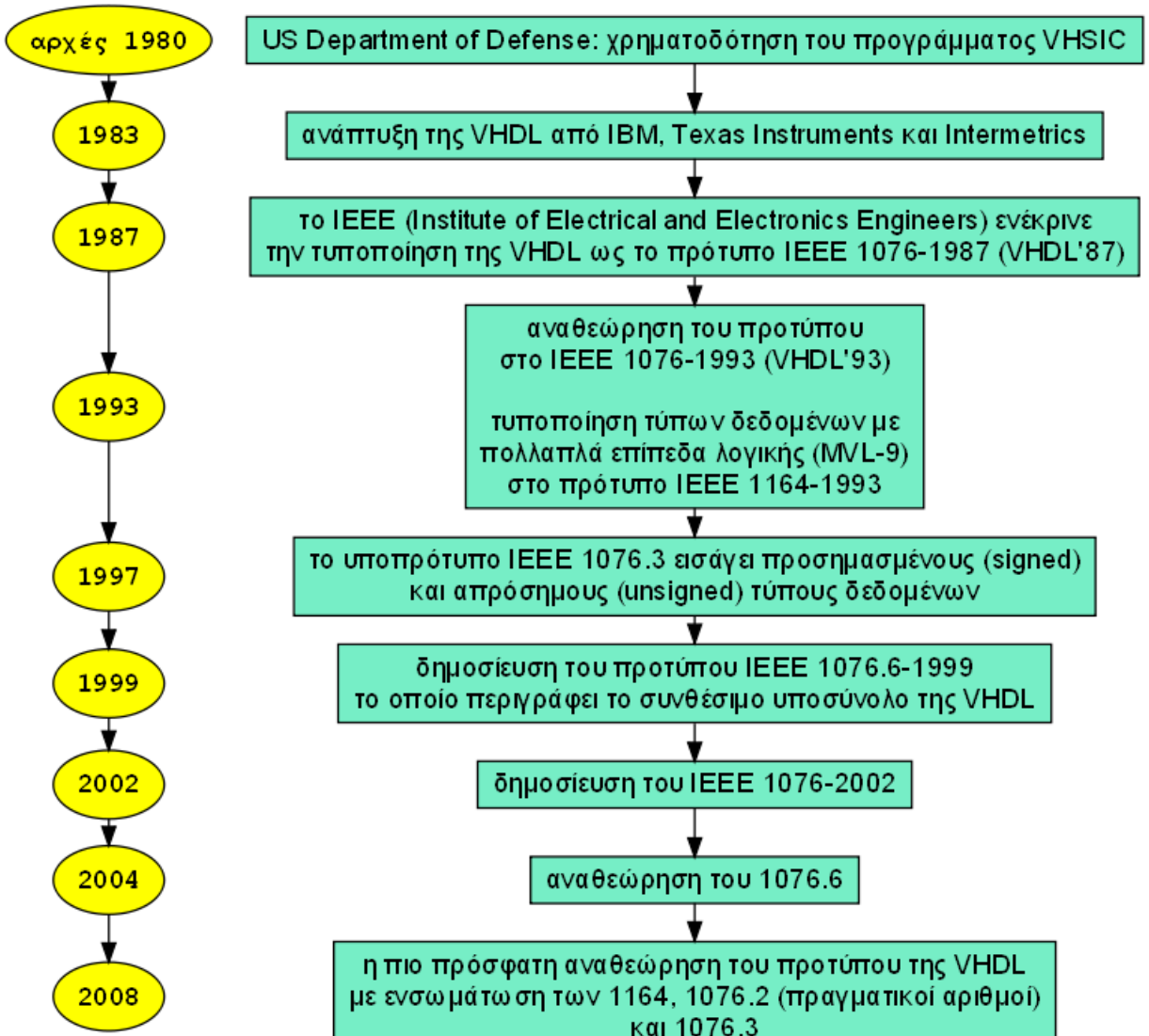
Σκιαγράφηση της διάλεξης

- Εισαγωγή στη VHDL
- Δομές ακολουθιακού και συντρέχοντος κώδικα
- Προχωρημένα στοιχεία της VHDL
- Σύνταξη παραμετρικών περιγραφών

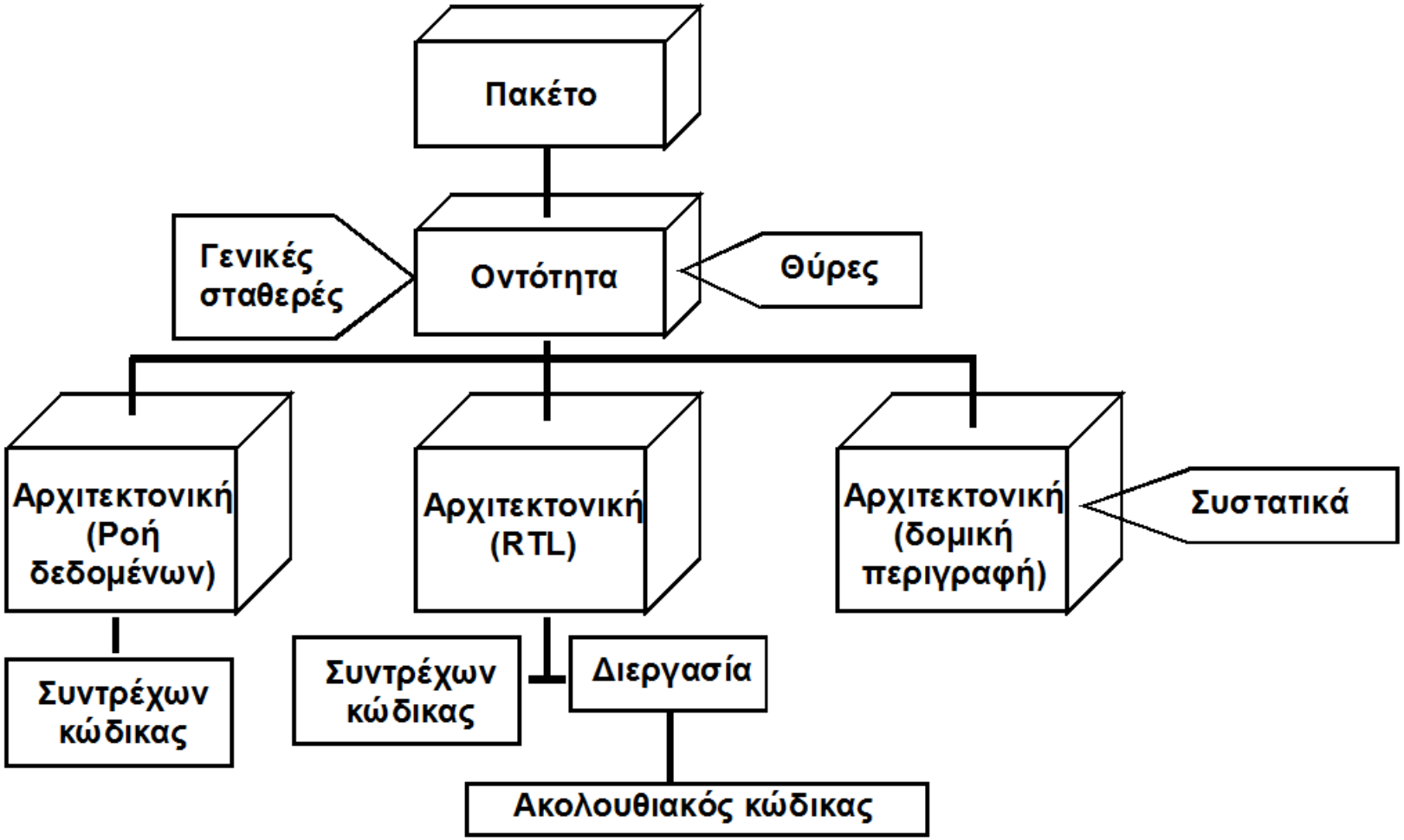
Εισαγωγικά

- Η VHDL αποτελεί μια γλώσσα για τη μοντελοποίηση της δομής και της συμπεριφοράς υλικού
- Επιτρέπει τη χρήση διαφορετικών μεθοδολογιών σχεδίασης
- Προσφέρει ανεξαρτησία από την εκάστοτε τεχνολογία υλοποίησης (standard cell VLSI, FPGA)
- Διευκολύνει την επικοινωνία σχεδίων μεταξύ συνεργαζόμενων ομάδων σχεδίασης
- Βοηθά στην καλύτερη διαχείριση του έργου της σχεδίασης
- Στη VHDL μπορεί να περιγραφεί ένα μεγάλο εύρος ψηφιακών κυκλωμάτων
- Βασικά κριτήρια στην επιλογή HDL (Verilog ή VHDL) είναι: η διαθεσιμότητα εργαλείων ανάπτυξης, η δυνατότητα επαναχρησιμοποίησης κώδικα, και η οικειότητα με τις συντακτικές δομές της γλώσσας

Ιστορική αναδρομή της ανάπτυξης της VHDL



Ιεραρχική σχεδίαση με τη VHDL



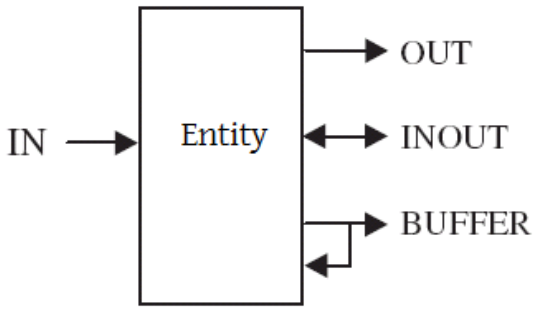
Θεμελιώδεις γνώσεις για τη συγγραφή κώδικα VHDL

- Οι βασικές δομές της VHDL είναι η **ΟΝΤΟΤΗΤΑ** (ENTITY) και η **ΑΡΧΙΤΕΚΤΟΝΙΚΗ** (ARCHITECTURE) της περιγραφής ενός κυκλώματος
  - ENTITY: Η διεπαφή του κυκλώματος (θύρες εισόδου και εξόδου) με το περιβάλλον
  - ARCHITECTURE: Καταγράφει τους απαιτούμενους μηχανισμούς λειτουργίας για την υλοποίηση του κυκλώματος
- Δεν υφίσταται ευαισθησία πεζών-κεφαλαίων (case insensitivity). Κεφαλαίοι και πεζοί χαρακτήρες χρησιμοποιούνται ελεύθερα για τη σύνταξη λέξεων-κλειδιών, τελεστών (τερματικά) και αναγνωριστικών (identifiers)
- Η γραμμή σχολίου δηλώνεται με δύο διαδοχικές παύλες: --
- Ο τύπος δεδομένων BIT είναι προκαθορισμένος και μπορεί να πάρει τις τιμές 0 ή 1
- Ο τύπος BIT_VECTOR αποτελεί διάνυσμα από BIT

ENTITY (ΟΝΤΟΤΗΤΑ) (1)

- Περιγράφει τον τρόπο διασύνδεσης του κυκλώματος
- Δήλωση των θυρών εισόδου/εξόδου προς και από το κύκλωμα
- Δήλωση *γενικών σταθερών* με εμβέλεια την οντότητα και τις αρχιτεκτονικές που υπάγονται σε αυτή

- Για μια θύρα δηλώνονται: όνομα, κατευθυντικότητα, τύπος δεδομένων
- Τύποι θυρών: *IN*, *OUT*, *INOUT*, *BUFFER*
  - IN: Είσοδος
  - OUT: Έξοδος (δεν διαβάζεται εσωτερικά)
  - INOUT: Είσοδος και έξοδος
  - BUFFER: Έξοδος με δυνατότητα εσωτερικής ανάγνωσης



ENTITY (ΟΝΤΟΤΗΤΑ) (2)

- Καλό είναι να αποφεύγεται η χρήση του τύπου θύρας **buffer**. Όταν χρησιμοποιούνται ιεραρχικά, οι θύρες buffer μπορούν να συνδεθούν μόνο με άλλες θύρες buffer κάτι που υποχρεώνει το ιεραρχικό κύκλωμα να παρουσιάζει διεπαφή τύπου buffer. Αυτό περιορίζει τη διασυνδεσιμότητα του κυκλώματος με άλλες μονάδες

Σύνταξη μιας ENTITY:

```
entity name-of-entity is
  generic (
    generic_list with initializations
  );
  port (
    port_list
  );
end [entity] name-of-entity;
```

ARCHITECTURE (ΑΡΧΙΤΕΚΤΟΝΙΚΗ)

- Στο σώμα της αρχιτεκτονικής (architecture body) περιγράφονται οι μηχανισμοί λειτουργίας του κυκλώματος
- Καταγράφει δηλώσεις υποπρογραμμάτων, σταθερών, συστατικών και σημάτων στην περιοχή δηλώσεων
- Μπορεί να περιλαμβάνει συντρέχοντα ή/και ακολουθιακό κώδικα
- Κάθε κύκλωμα περιγράφεται από μία μόνο οντότητα, αλλά επιτρέπονται περισσότερες της μιας αρχιτεκτονικές υλοποιήσεις

Σύνταξη μιας ARCHITECTURE

```
architecture architecture-name is
  [architecture declarations]
begin
  [concurrent statements]
  [sequential statements]
  [structural code]
end [architecture] architecture-name;
```

Ο πλήρης αθροιστής δυαδικού ψηφίου σε VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity full_adder is
  port (
    a    : in  STD_LOGIC;
    b    : in  STD_LOGIC;
    cin  : in  STD_LOGIC;
    s    : out STD_LOGIC;
    cout : out STD_LOGIC
  );
end full_adder;

architecture structural of full_adder is
begin
  s    <= a xor b xor cin;
  cout <= (a and b) or (a and cin) or (b and cin);
end structural;
```

Συχνά χρησιμοποιούμενοι τύποι της VHDL

ΤΥΠΟΣ	ΤΙΜΗ	ΠΡΟΕΛΕΥΣΗ
std_ulogic	'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'	std_logic_1164
std_ulogic_vector	array of std_ulogic	std_logic_1164
std_logic	resolved std_ulogic	std_logic_1164
std_logic_vector	array of std_logic	std_logic_1164
unsigned	array of std_logic	numeric_std, std_logic_arith
signed	array of std_logic	numeric_std, std_logic_arith
boolean	true, false	standard
character	191 / 256 characters	standard
string	array of character	standard
integer	$-(2^{31}-1)$ to $2^{31}-1$	standard
real	$-1.0E38$ to $1.0E38$	standard
time	1 fs to 1 hr	standard

Τύποι δεδομένων στο package STANDARD

- BIT: `'0', '1'`
- BIT_VECTOR: `"001100", X"00FF"` στο δεκαεξαδικό
- BOOLEAN: `true, TRUE, TruE` για το αληθές και `False, false, FALsE` για το ψευδές
- CHARACTER: `'A', 'a', '@', '''`. Ένας πίνακας από `CHARACTER` αποτελεί συμβολοσειρά (string): `"hold time out of range", "i'll be back", "0$#1324"`
- REAL: `-1.0, +2.35, 36.6, -1.0E+38`
- INTEGER με εύρος τιμών {-2,147,483,647, +2,147,483,647}: `+1, 862, -257, +15`
- TIME: `10 ns, 100 us, 6.3 ns`

Αναπαράσταση ακεραίων με διανύσματα

Δυαδικό και δεκαεξαδικό (απρόσημοι)

Ακέραιος	Δυαδικό	hex
0	"00000"	X"0"
1	"00001"	X"1"
2	"00010"	X"2"
3	"00011"	X"3"
4	"00100"	X"4"
5	"00101"	X"5"
6	"00110"	X"6"
7	"00111"	X"7"
8	"01000"	X"8"
9	"01001"	X"9"
10	"01010"	X"A"
11	"01011"	X"B"
12	"01100"	X"C"
13	"01101"	X"D"
14	"01110"	X"E"
15	"01111"	X"F"
16	"10000"	X"10"
17	"10001"	X"11"

Συμπλήρωμα ως προς 2

Ακέραιος	Δυαδικό	hex
-8	"1000"	X"8"
-7	"1001"	X"9"
-6	"1010"	X"A"
-5	"1011"	X"B"
-4	"1100"	X"C"
-3	"1101"	X"D"
-2	"1110"	X"E"
-1	"1111"	X"F"
0	"0000"	X"0"
1	"0001"	X"1"
2	"0010"	X"2"
3	"0011"	X"3"
4	"0100"	X"4"
5	"0101"	X"5"
6	"0110"	X"6"
7	"0111"	X"7"

CONSTANT, VARIABLE και SIGNAL

- CONSTANT: αντικείμενο στο οποίο ανατίθεται μία αμετάβλητη τιμή

```
constant identifier : type-indication [:=expression];
constant PI : REAL := 3.147592;
constant FIVE : BIT_VECTOR := "0101";
```

- VARIABLE: αντικείμενο στο οποίο κάποια στιγμή ανατίθεται μία τιμή η οποία μπορεί να μεταβληθεί εντός μιας PROCESS

```
variable identifier : type-indication [constraint] [:=expression];
variable index: INTEGER range 1 to 50 := 50;
variable x, y : INTEGER;
variable memory : STD_LOGIC_VECTOR(0 to 7);
```

- SIGNAL: υλοποίηση διασυνδέσεων εντός ενός κυκλώματος αλλά και για την εξωτερική διασύνδεση διαφορετικών μονάδων σχεδιασμού

```
signal identifier : type-indication [constraint] [:=expression];
signal control: BIT := '0';
signal count: INTEGER range 0 to 100;
signal y: STD_LOGIC_VECTOR(7 downto 1);
```

Τελεστές της VHDL (VHDL operators)

Συνοπτικός πίνακας των τελεστών

λογικοί	and	or	nand	nor	xor	xnor	not
αριθμητικοί	+	-	*	/	mod	rem	
σύγκρισης	=	/=	<	<=	>	>=	
ολίσθησης	sll	srl	sla	sra	rol	ror	
μοναδιαίοι	+	-					
άλλοι	**	abs	&				

- Οι τελεστές αναγωγής σε δύναμη "**", απόλυτης τιμής "abs", και υπολογισμού ακέραιου υπολοίπου ("mod", "rem") δεν είναι συνθέσιμοι
- Οι τελεστές πολλαπλασιασμού και διαίρεσης υποστηρίζονται από ορισμένα εργαλεία λογικής σύνθεσης υπό προϋποθέσεις
- Η διαφορά των mod και rem είναι ότι: το `A rem B` παίρνει το πρόσημο του A ενώ το `A mod B` το πρόσημο του B

Αναθέσεις σε VARIABLE και SIGNAL

- Η ανάθεση σε `VARIABLE` αντικαθιστά την τρέχουσα τιμή της με μια νέα τιμή
- Τα `SIGNAL` προσφέρουν επικοινωνία μεταξύ διαφορετικών PROCESS και `COMPONENT` instances

```
ix := 'a';
y := "0000";
```

```
ix <= 'a';
y <= "0000";
```

Απόδοση τιμής	SIGNAL <=	VARIABLE :=
Χρησιμότητα	Αναπαριστά κυκλωματικές διασυνδέσεις	Αναπαριστά τοπική πληροφορία
Εμβέλεια	Μπορεί να είναι καθολική	Τοπική (διεργασία, συνάρτηση ή διαδικασία)
Συμπεριφορά	Η ενημέρωση δεν είναι άμεση σε ακολουθιακό κώδικα	Άμεση ενημέρωση
Χρήση	PACKAGE, ENTITY, ARCHITECTURE	PROCESS, FUNCTION, PROCEDURE

Συντρέχων και ακολουθιακός κώδικας

- Στη VHDL ο κώδικας είναι από τη φύση του παράλληλα εκτελούμενος (συντρέχων)
- Η VHDL διαθέτει προγραμματιστικές δομές για την περιγραφή ακολουθιακού κώδικα, προκειμένου την εξασφάλιση της διαδοχικής εκτέλεσης εντολών όταν αυτό είναι επιθυμητό
- Ακολουθιακός κώδικας στη VHDL: μέσα σε PROCESS, FUNCTION, PROCEDURE
- Συντρέχων κώδικας: ανάθεση σε SIGNAL, υπολογισμός έκφρασης με χρήση τελεστών και ανάθεση σε SIGNAL, εντολή WHEN, εντολή WITH/SELECT, εντολή GENERATE
- Μία διεργασία αποτελεί τμήμα συντρέχοντος κώδικα (μία διεργασία εκτελείται παράλληλα σε σχέση με τυχόν άλλες διεργασίες)

Ακολουθιακός κώδικας

- Ο ακολουθιακός κώδικας στη VHDL συντάσσεται εντός μιας PROCESS, η συμπεριφορά της οποίας προσομοιώνεται βήμα προς βήμα (εντολή προς εντολή)
  - Δομές ελέγχου: εντολές IF και CASE
  - Δομές επανάληψης: εντολές FOR και WHILE
  - Εντολές NEXT και EXIT για τον εσωτερικό έλεγχο σε μια δομή επανάληψης
- Στον ίδιο χρόνο προσομοίωσης, επιτρέπεται να είναι ενεργές (active) περισσότερες από μία PROCESS. Έτσι η PROCESS αποτελεί δομικό λίθο για την ανάπτυξη συντρέχοντος κώδικα
- Μέσα σε μία PROCESS μπορούν να χρησιμοποιηθούν υποπρογράμματα

PROCESS

- Η PROCESS προσφέρει τη δυνατότητα σχεδιασμού ακολουθιακού κώδικα

```
[process_label:] process [(sensitivity list)]
  [declarations (subprogram, type, subtype, constant, variable, file,
  alias, attribute), attribute specification, use clause]
begin
  sequential_statements
end process [process_label];
```

- Η λίστα ευαισθησίας αποτελεί κατάλογο εισόδων και SIGNAL για μεταβολές των οποίων μία PROCESS υποχρεούται να αναμένει
- Παράδειγμα:

```
process (a, b)
begin
  if (a /= b) then
    cond <= '1';
  else
    cond <= '0';
  end if;
end process;
```

Παραδείγματα σύνταξης μιας PROCESS

- Παράδειγμα 1: Μετατροπή βαθμών Κελσίου σε Φαρενάιτ (μη συνθέσιμος κώδικας)

```
CtoF: process
variable c, f, g: real;
begin
  c := 0.0;
  while (c < 40.0) loop
    f := 1.8 * c  + 32.0;
    c := c + 2.0;
  end loop;
  wait for 1 ns;
end process CtoF;
```

- Παράδειγμα 2: 2-to-1 multiplexer (συνθέσιμος κώδικας)

```
process(sel, a, b)
begin
  if (sel = '1') then
    outp <= b;
  else
    outp <= a;
  end if;
end process;
```

Ευαισθησία επιπέδου (level-sensitivity) και ακμοπυροδότηση (edge triggering)

- Οι μεταβολές των σημάτων που δηλώνονται σε μία λίστα ευαισθησίας και οι οποίες ενεργοποιούν τον υπολογισμό μεταβλητών και σημάτων σε μία PROCESS είναι δύο τύπων:
  - Μεταβολή επιπέδου (για σήματα επίτρεψης/ενεργοποίησης και δεδομένα)
  - Ανερχόμενη ή κατερχόμενη ακμή (για σήματα ρολογιού)

- Μανδαλωτής (μεταβολή επιπέδου)

```
process (en, a)
begin
  if (en = '1') then
    temp <= a;
  end if;
end process;
```

- Συγχρονισμός ως προς ανερχόμενη ακμή

```
process (clk, a)
begin
  if (clk = '1' and clk'EVENT) then
    temp <= a;
  end if;
end process;
```

☞ Η έκφραση `clk'EVENT` είναι TRUE όταν έχει συμβεί μεταβολή ($0 \rightarrow 1$ ή $1 \rightarrow 0$) στο σήμα `clk`

ℹ Για ορισμένα εργαλεία σύνθεσης, τα σήματα εκτός του `clk` μπορούν να παραλειφθούν από μια λίστα ευαισθησίας

Δομές ελέγχου σε ακολουθιακό κώδικα

- Η εντολή IF αποτελεί τη θεμελιώδη δομή για την εκτέλεση κώδικα υπό συνθήκη

```
if ... then
  s1;
[elsif ... then
  s2;]
[else
  s3;]
end if;
```

- Η εντολή CASE χρησιμοποιείται για την περιγραφή δομών αποκωδικοποίησης

```
case expression is
  when value => s1; s2; ... sn;
  when value1 | value2 | ... | valuen
    => s1; s2; ... sn;
  when value1 to value2 => ...
  when others => ...
end case;
```

Δομές επανάληψης

- Η εντολή LOOP προσφέρει έναν βολικό τρόπο για την περιγραφή επαναληπτικών κυκλωματικών δομών

```
[loop_label:] [iteration_scheme] loop
  sequence_of_statements
end loop [loop_label];
```

- Το σχήμα επανάληψης (iteration scheme) μπορεί να είναι τύπου WHILE ή τύπου FOR:

```
while condition
for identifier in discrete_range
```

Παράδειγμα υπολογισμού τετραγώνων ακεραίων με FOR

```
FOR i IN 1 to 10 LOOP
  i_squared := i * i;
END LOOP;
```

Παράδειγμα υπολογισμού τετραγώνων ακεραίων με WHILE

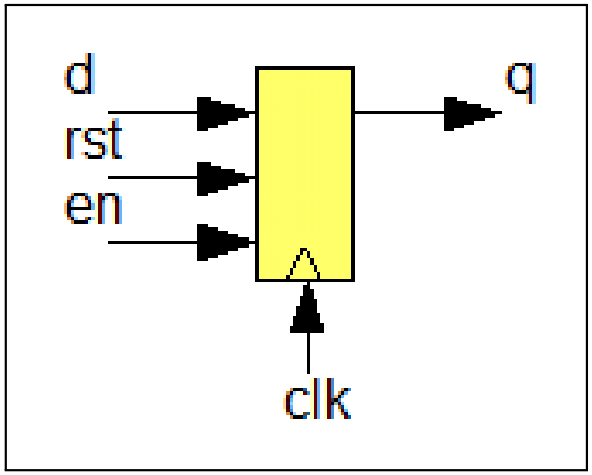
```
i := 1;
WHILE (i<11) LOOP
  i_squared := i * i;
  i := i + 1;
END LOOP;
```

Καταχωρητής με επίτρεψη φόρτωσης (load enable)

Κώδικας VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity dreg is
  port (
    clk, d, rst : in std_logic;
    en          : in std_logic;
    q           : out std_logic
  );
end dreg;

architecture rtl of dreg is
signal temp: std_logic;
begin
process (clk, rst, d, en)
  if (clk'event and clk = '1') then
    if (rst = '1') then
      temp <= '0';
    else
      if (en = '1') then
        temp <= d;
      end if;
    end if;
  end if;
end process;
q <= temp;
end rtl;
```

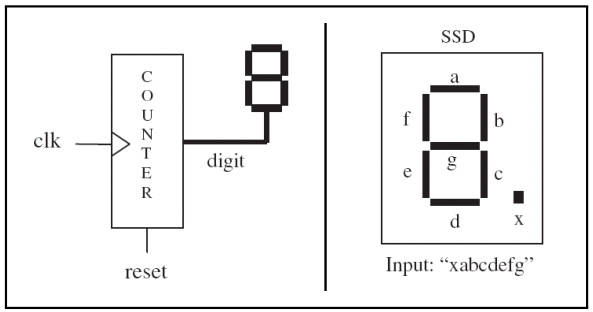
Σχηματικό διάγραμμα



Παράδειγμα περιγραφής κυκλώματος: Απαριθμητής ψηφίου με έξοδο σε display επτά τμημάτων (1)

- Δεκαδικός απαριθμητής ενός ψηφίου (μετρά $0 \rightarrow 9$) που επιπλέον μετατρέπει τους δυαδικά κωδικοποιημένους δεκαδικούς (BCD: Binary Coded Decimal) σε μορφή κατάλληλη για απεικόνιση σε ενδείκτη επτά τμημάτων (SSD: Seven Segment Display)
- Το κύκλωμα διασυνδέεται με τον ενδείκτη ως εξής: `abcdefg`, με το πιο σημαντικό bit (MSB) να τροφοδοτεί το τμήμα a και το LSB το τμήμα g. Η υποδιαστολή x δεν χρησιμοποιείται

```
entity bcdcounter is
  port (
    clk, reset: in std_logic;
    digit: out std_logic_vector(6 downto 0)
  );
end bcdcounter;
```

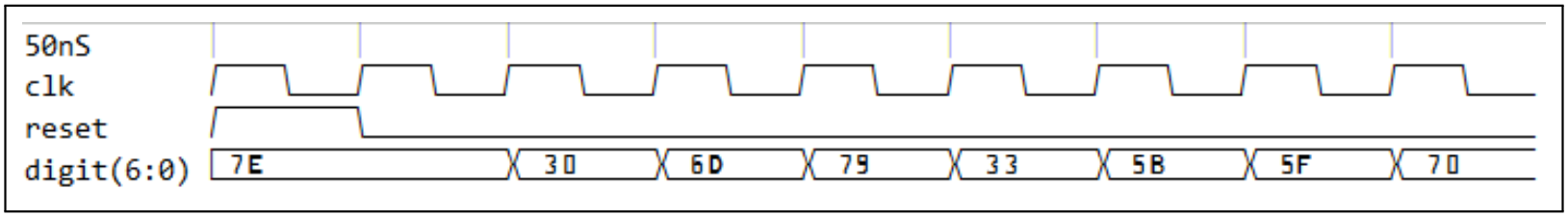


Παράδειγμα περιγραφής κυκλώματος: Απαριθμητής ψηφίου με έξοδο σε display επτά τμημάτων (2)

```
architecture rtl of bcdcounter is
begin
  process (clk, reset)
  variable temp : integer range 0 to 10;
  begin
    -- counter
    if (reset = '1') then
      temp := 0;
    elsif (clk'EVENT and clk='1') then
      temp := temp + 1;
      if (temp = 10) then
        temp := 0;
      end if;
    end if;
    -- BCD->SSD conversion
    case temp is
      when 0 => digit <= "1111110"; -- 7E
      when 1 => digit <= "0110000"; -- 30
      when 2 => digit <= "1101101"; -- 6D
      when 3 => digit <= "1111001"; -- 79
      when 4 => digit <= "0110011"; -- 33
      when 5 => digit <= "1011011"; -- 5B
      when 6 => digit <= "1011111"; -- 5F
      when 7 => digit <= "1110000"; -- 70
      when 8 => digit <= "1111111"; -- 7F
      when 9 => digit <= "1111011"; -- 7B
      when others => digit <= "1001111"; -- 9F ('E' for error)
    end case;
  end process;
end rtl;
```

Παράδειγμα περιγραφής κυκλώματος: Απαριθμητής ψηφίου με έξοδο σε display επτά τμημάτων (3)

Διάγραμμα χρονισμού για το κύκλωμα του απαριθμητή ψηφίου



Δομές συντρέχοντος κώδικα

- Η εντολή WHEN/ELSE αποτελεί μία συντρέχουσα εντολή η οποία έχει ένα στόχο (target) επιλέγοντας από περισσότερες από μία εκφράσεις

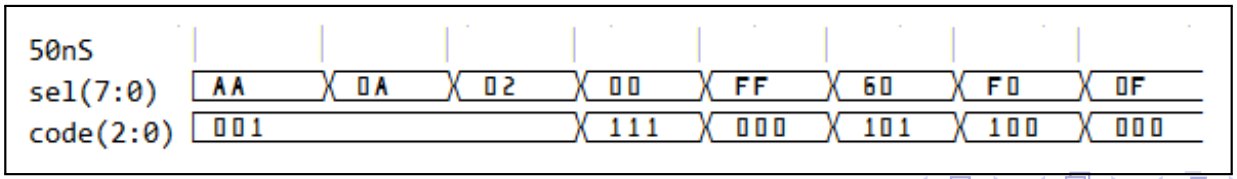
```
target <= {expression when condition else} expression;
outp <= "000" WHEN (inp='0' OR reset='1') ELSE
        "001" WHEN (ctl='1') ELSE
        "010";
```

- Η εντολή WITH/SELECT προσφέρει τη δυνατότητα επιλεκτικής ανάθεσης σε ένα στόχο (target) επιλέγοντας από περισσότερες από μία εκφράσεις

```
WITH expression SELECT
  target <= {expression WHEN choices,} expression;
WITH control SELECT
  output <= reset WHEN "000",
            set WHEN "111",
            UNAFFECTED WHEN others;
```

Κωδικοποιητής προτεραιότητας (priority encoder)

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

entity priority is
  port (
    sel  : in  std_logic_vector(7 downto 0);
    code : out unsigned(2 downto 0)
  );
end priority;

architecture imp of priority is
begin
  code <= "000" when sel(0) = '1' else
          "001" when sel(1) = '1' else
          "010" when sel(2) = '1' else
          "011" when sel(3) = '1' else
          "100" when sel(4) = '1' else
          "101" when sel(5) = '1' else
          "110" when sel(6) = '1' else "111";
end imp;
```

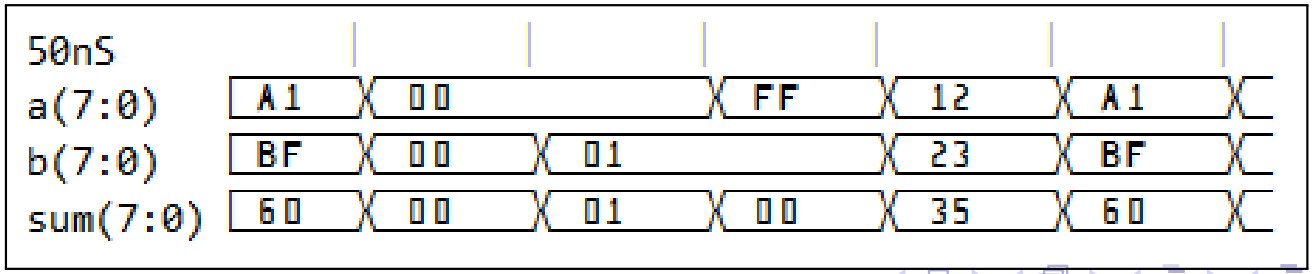


Αθροιστές απρόσημων και προσημασμένων (2's complement) ακεραίων (1)

- Περιγραφή για απρόσημους αριθμούς

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity adder is
  port (
    a,b : in std_logic_vector(7 downto 0);
    sum : out std_logic_vector(7 downto 0)
  );
end adder;

architecture rtl of adder is
  signal temp : std_logic_vector(8 downto 0);
begin
  temp <= ('0' & a)  + ('0' & b);
  sum <= temp(7 downto 0);
end rtl;
```

- Διάγραμμα χρονισμού

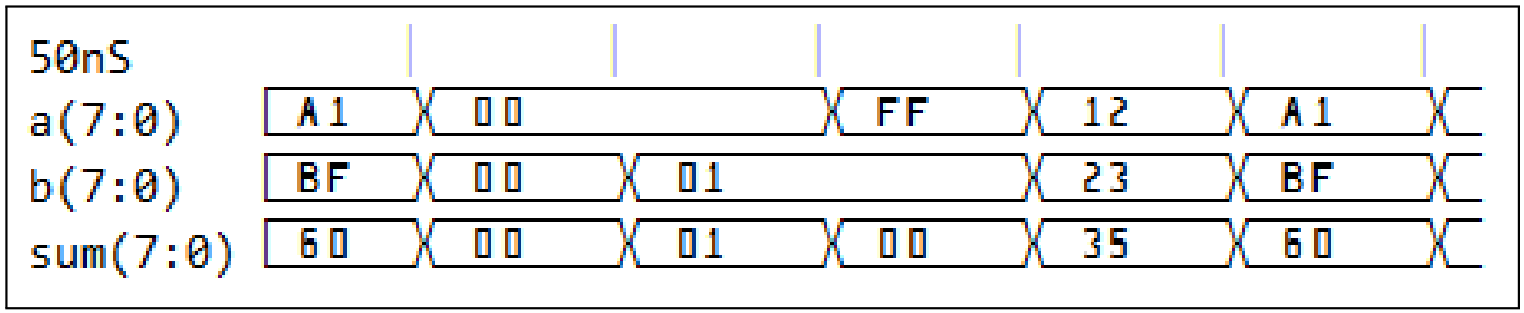


Αθροιστές απρόσημων και προσημασμένων (2's complement) ακεραίων (2)

- Για την άθροιση προσημασμένων (συμπλήρωμα-ως-προς-2) απαιτείται η επέκταση προσήμου (sign extension) του ενδιάμεσου αποτελέσματος
- Περιγραφή για προσημασμένους αριθμούς

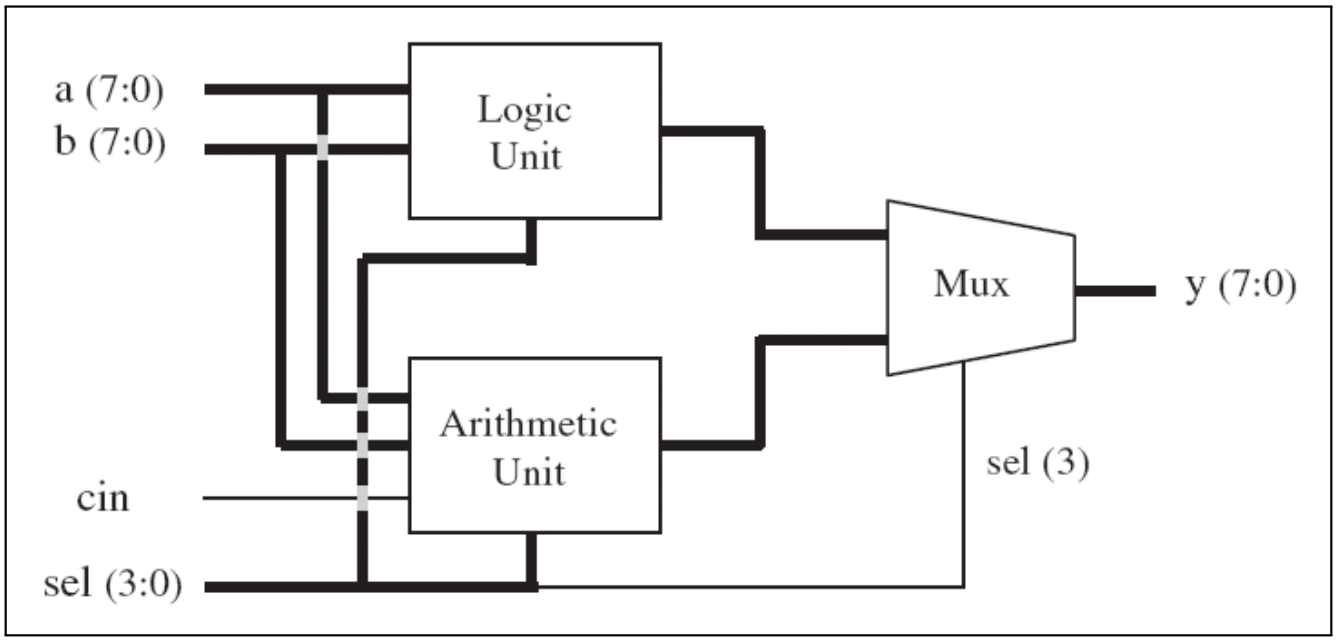
```
...
  signal temp : std_logic_vector(8 downto 0);
begin
  temp <= (a(7) & a) + (b(7) & b);
  sum <= temp(7 downto 0);
end rtl;
```

- Διάγραμμα χρονισμού



Αριθμητική-λογική μονάδα (ALU) (1)

Σχηματικό διάγραμμα μιας ALU για έναν υποθετικό 8-bit επεξεργαστή



Αριθμητική-λογική μονάδα (ALU) (2)

- Προδιαγραφές μιας ALU για έναν υποθετικό 8-bit επεξεργαστή
- Ρεπερτόριο εντολών

Opcode	Κωδικοπ.	Πράξη	Λειτουργία
Αριθμητική μονάδα			
`MOVA`	0000	`y <= a`	Μεταφορά του a
`INCA`	0001	`y <= a + 1`	Αύξηση κατά 1 του a
`DECA`	0010	`y <= a - 1`	Μείωση κατά 1 του a
`MOVB`	0011	`y <= b`	Μεταφορά του b
`INCB`	0100	`y <= b + 1`	Αύξηση κατά 1 του b
`DECB`	0101	`y <= b - 1`	Μείωση κατά 1 του b
`ADD`	0110	`y <= a + b`	Άθροιση των a,b
`ADC`	0111	`y <= a + b + cin`	Άθροιση των a,b με κρατούμενο
Λογική μονάδα			
`NOTA`	1000	`y <= not a`	Αντιστροφή του a
`NOTB`	1001	`y <= not b`	Αντιστροφή του b
`AND`	1010	`y <= a and b`	Λογική πράξη AND
`IOR`	1011	`y <= a or b`	Λογική πράξη OR
`NAND`	1100	`y <= a nand b`	Λογική πράξη NAND
`NOR`	1101	`y <= a nor b`	Λογική πράξη NOR
`XOR`	1110	`y <= a xor b`	Λογική πράξη XOR
`XNOR`	1111	`y <= a xnor b`	Λογική πράξη XNOR

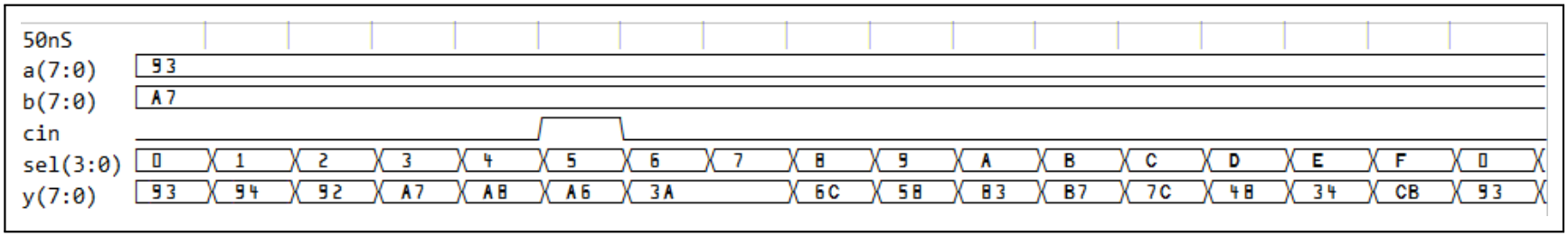
Αριθμητική-λογική μονάδα (ALU): Κώδικας (3)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity alu is
  port (
    a,b : in std_logic_vector(7 downto 0);
    cin : in std_logic;
    sel : in std_logic_vector(3 downto 0);
    y : out std_logic_vector(7 downto 0)
  );
end alu;

architecture dataflow of alu is
signal arith: std_logic_vector(7 downto 0);
signal logic: std_logic_vector(7 downto 0);
begin
  -- Arithmetic unit
  with sel(2 downto 0) select
    arith <= a when "000",
             a+1 when "001",
             a-1 when "010",
             b when "011",
             b+1 when "100",
             b-1 when "101",
             a+b when "110",
             a+b+cin when "111";
```

```
  -- Logic unit
  with sel(2 downto 0) select
    logic <= not a when "000",
             not b when "001",
             a and b when "010",
             a or b when "011",
             a nand b when "100",
             a nor b when "101",
             a xor b when "110",
             a xnor b when "111";
  -- Multiplexer
  with sel(3) select
    y <= arith when '0',
         logic when others;
end dataflow;
```

Αριθμητική-λογική μονάδα (ALU) (4)

Διάγραμμα χρονισμού για την ALU



Συστατικό στοιχείο (COMPONENT) - Στιγμιότυπο ενός COMPONENT

- Το COMPONENT αποτελεί δήλωση για τη χρησιμοποίηση ενός κυκλώματος ως υπομονάδα

```
COMPONENT <component name> IS
  [GENERIC (
    <generic name> : <type> [:= <initialization>];
    ...
  )]
  PORT (
    <port name> : [direction] <signal type>;
    ...
  );
END COMPONENT;
```

- Το στιγμιότυπο (instance) ενός COMPONENT αποτελεί ένα αντίτυπό του που χρησιμοποιείται στα πλαίσια της δομικής περιγραφής ενός κυκλώματος

```
<label>: <component name>
  [GENERIC MAP (
    [<generic name> =>] <expression>,
    ... );
  PORT MAP (
    [<port name> =>] <expression>,
    ... );
```

PACKAGE

- Το πακέτο (PACKAGE) αποτελεί μέσο για την οργάνωση τμημάτων κώδικα

```
PACKAGE <package name> IS
  statements
END <package name>;
[PACKAGE BODY <package name> IS
  implementation of functions and procedures
END <package name>;]
```

- Δήλωση χρήσης ενός πακέτου

```
USE <library name>.<package name>.<package parts>;
```

- Παράδειγμα

```
PACKAGE my_package IS
  TYPE state IS (st1, st2, st3, st4);
  FUNCTION positive_edge(SIGNAL s: BIT) RETURN BOOLEAN;
END my_package;
PACKAGE BODY my_package IS
  FUNCTION positive_edge(SIGNAL s: BIT) RETURN BOOLEAN IS
  BEGIN
    RETURN (s'EVENT and s='1');
  END positive_edge;
END my_package;
```

FUNCTION (ΣΥΝΑΡΤΗΣΗ)

- Οι συναρτήσεις αποτελούν είδος υποπρογράμματος το οποίο επιστρέφει μία μοναδική τιμή
- Δεν μπορούν να τροποποιήσουν τις παραμέτρους εισόδου τους

```
FUNCTION <function name> [<parameter list>] RETURN <type> IS
  [statements]
BEGIN
  sequential statements
END <function name>;
```

- Παράδειγμα: Συνάρτηση $\lceil log_2 \rceil$

```
function LOG2C(input: INTEGER) return INTEGER is
  variable temp,log: INTEGER;
  begin
    log := 0; temp := 1;
    for i in 0 to input loop
      if temp < input then
        log  := log + 1;
        temp := temp * 2;
      end if;
    end loop;
    return (log);
end function LOG2C;
```

PROCEDURE (ΔΙΑΔΙΚΑΣΙΑ)

- Μπορούν να τροποποιούν τις τιμές των παραμέτρων τους
- Δεν περιλαμβάνουν εντολή RETURN
- Παράμετροι (CONSTANT, SIGNAL ή VARIABLE) με κατευθυντικότητα (IN, OUT, INOUT)

```
PROCEDURE <procedure name> [<parameter list>] IS
  [statements]
BEGIN
  sequential statements
END <procedure name>;
```

- Παράδειγμα: Διαδικασία sort

```
PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 to limit;
  SIGNAL min, max: OUT INTEGER RANGE 0 to limit) IS
BEGIN
  IF (in1 >= in2) THEN
    max <= in1;  min <= in2;
  ELSE
    max <= in2; min <= in1;
  END IF;
END sort;
...
sort (inp1, inp2, outp1, outp2); -- usage
```

GENERATE

- Η εντολή GENERATE παρέχει τη δυνατότητα περιγραφής επαναλαμβανόμενων κυκλωματικών δομών με συμπαγή τρόπο σε συντρέχοντα κώδικα
- Διευκολύνει την περιγραφή δομών που παρουσιάζουν κανονικότητα
- Ειδικότερα, μια εντολή GENERATE μπορεί να περικλείει στιγμιότυπα συστατικών ή άλλες GENERATE
- Σχήματα FOR και IF

```
<label> : (FOR|IF) parameter_specification GENERATE
  [declaration_statements]
BEGIN
  {concurrent_statements}
END GENERATE <label>;
```

Παραμετρικός αθροιστής ριπής κρατουμένου (parameterized ripple-carry adder)

- Η γενική σταθερά N καθορίζει το εύρος bit του αθροιστή
- Για κάθε m στην εντολή GENERATE δημιουργείται ένας πλήρης αθροιστής με διασύνδεση του κρατουμένου εξόδου του στο κρατούμενο εισόδου της επόμενης βαθμίδας

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

entity rca is
  generic (N: integer := 8);
  port (
    a, b : in unsigned(N-1 downto 0);
    cin : in std_logic;
    sum : out unsigned(N-1 downto 0);
    cout : out std_logic
  );
end rca;
```

```
architecture gatelevel of rca is
  signal c : unsigned(N downto 0);
begin
  c(0) <= cin;
  G1: for m in 0 to N-1 generate
    sum(m) <= a(m) xor b(m) xor c(m);
    c(m+1) <= (a(m) and b(m)) or
              (b(m) and c(m)) or
              (a(m) and c(m));
  end generate G1;
  cout <= c(N);
end gatelevel;
```