Γλώσσες Περιγραφής Υλικού

Δομές ακολουθιακού και συντρέχοντος κώδικα

Νικόλαος Καββαδίας
nkavv@physics.auth.gr

24 Μαρτίου 2009

Σκιαγράφηση της διάλεξης

- Συντρέχων και ακολουθιακός κώδικας
  - Ανάθεση σε ΜΕΤΑΒΛΗΤΗ (VARIABLE) και ΣΗΜΑ (SIGNAL)
  - Διαφορές μεταξύ VARIABLE και SIGNAL
  - Ακολουθιακός κώδικας: ΔΙΕΡΓΑΣΙΑ (PROCESS)
  - Εντολή χρονισμού WAIT και σύγχρονα κυκλώματα
  - Δομές επιλογής: IF και CASE-WHEN
  - Δομές επανάληψης: LOOP, σχήματα FOR και WHILE, εντολές NEXT, EXIT
  - Συντρέχων κώδικας: εντολές WHEN-ELSE, WITH-SELECT
  - Παραδείγματα σχεδιασμού κυκλωμάτων: απαριθμητής ψηφίου, καταχωρητής, πολυπλέκτης, τρισταθής απομονωτής, αθροιστές, αριθμητική-λογική μονάδα (ALU)

Ανάθεση σε VARIABLE

- Η ανάθεση σε VARIABLE αντικαθιστά την τρέχουσα τιμή της με μια νέα τιμή η οποία προσδιορίζεται από μια αριθμητική έκφραση
- Σύνταξη μιας ανάθεσης:
  `identifier := expression;`
- Παραδείγματα:

```
ix := 'a'; -- assignment of a character value
a := 1.0; -- assignment of REAL number "1.0"
y := "0000";
```

- Οι μεταβλητές δεν είναι ορατές έξω από μια PROCESS. Έχουν τοπική εμβέλεια μέσα σε PROCESS, FUNCTION, ή PROCEDURE

Ανάθεση σε SIGNAL

- Τα ΣΗΜΑΤΑ (SIGNALS) προσφέρουν επικοινωνία μεταξύ διαφορετικών PROCESS και στιγμιότυπων συστατικών (COMPONENT instances)
- Ένα SIGNAL μπορεί να ανατεθεί σε μια VARIABLE και το αντίστροφο
- Σύνταξη μιας ανάθεσης: `identifier <= expression;`
- Παράδειγμα: επικοινωνία μεταξύ δύο στιγμιότυπων COMPONENT:

```
entity compare is
  port (A,B: in  BIT; C: out BIT);
end compare;

architecture structure of compare is
component my_xor                    -- component interfaces
  port (x,y: in  BIT; z: out BIT); -- for "my_xor" and "my_not"
end component;
component my_not
  port (x: in BIT; z: out BIT);
end component;
signal i: BIT;
begin
U0: my_xor port map (x => A, y => B, z => i);
U1: my_not port map (x => i, z => C);
end structural;
```

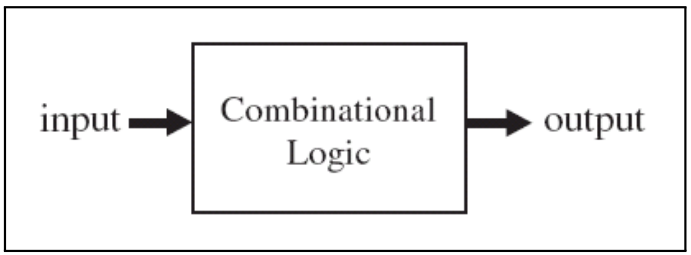
ΣΗΜΑ (SIGNAL) έναντι ΜΕΤΑΒΛΗΤΗΣ (VARIABLE)

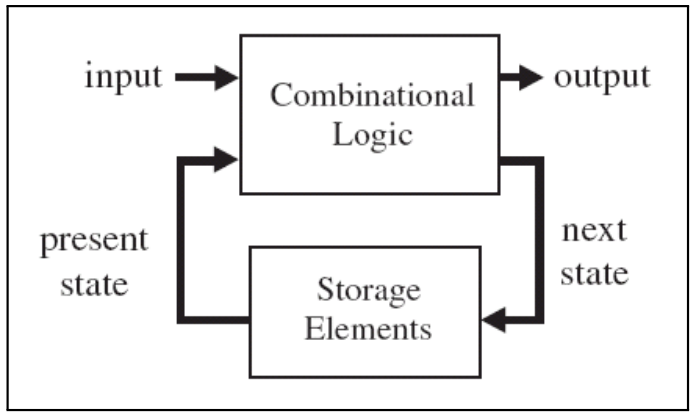
- Συχνά είναι δύσκολη η επιλογή ανάμεσα σε ένα αντικείμενο τύπου SIGNAL και σε ένα τύπου VARIABLE
- Βασική διαφορά είναι ότι η απόδοση τιμής σε μία VARIABLE είναι άμεση, ενώ για ένα SIGNAL αυτό συμβαίνει μετά την ολοκλήρωση εκτέλεσης της αντίστοιχης διεργασίας

	SIGNAL	VARIABLE
Απόδοση τιμής	<=	:=
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

Συνδυαστική και ακολουθιακή λογική

- Θεμελιώδεις τρόποι οργάνωσης των ψηφιακών κυκλωμάτων: συνδυαστική (combinational) και ακολουθιακή λογική (sequential)
- **Συνδυαστική λογική**: η έξοδος του κυκλώματος εξαρτάται αποκλειστικά από τις τρέχουσες εισόδους
- **Ακολουθιακή λογική**: η έξοδος του κυκλώματος εξαρτάται από τις τρέχουσες εισόδους και την τρέχουσα κατάσταση





Συντρέχων και ακολουθιακός κώδικας

- Στην 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 στην 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 (για την GENERATE περισσότερα στη 'Σύνταξη παραμετρικών περιγραφών')
- Μία διεργασία αποτελεί τμήμα συντρέχοντος κώδικα (μία διεργασία εκτελείται παράλληλα σε σχέση με τυχόν άλλες διεργασίες)

Ακολουθιακός κώδικας

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

ΔΙΕΡΓΑΣΙΑ (PROCESS)

- Η PROCESS προσφέρει τη δυνατότητα σχεδιασμού ακολουθιακού κώδικα, χρησιμοποιώντας τεχνικές από τον διαδικαστικό προγραμματισμό (ANSI C, Pascal)
- Σύνταξη μιας PROCESS:

```
[process_label:]
  process [(sensitivity list)]
    subprogram_declaration or subprogram_body
    type_declaration
    subtype_declaration
    constant_declaration
    variable_declaration
    file_declaration
    alias_declaration
    attribute_declaration
    attribute_specification
    use_clause
  begin
    sequential_statements
  end process [process_label];
```

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;
```

Λίστα ευαισθησίας (sensitivity list)

- Η λίστα ευαισθησίας αποτελεί κατάλογο εισόδων και SIGNAL για μεταβολές των οποίων μία PROCESS υποχρεούται να αναμένει
- ☞ Σε μια λίστα ευαισθησίας δηλώνονται ΟΛΕΣ οι είσοδοι ή σήματα που πρέπει να διαβαστούν στην PROCESS
- Παράδειγμα 1:

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

- Παράδειγμα 2: βρείτε το λάθος

```
process (a)
begin
  if (a = '1') then
    temp <= not (temp);
  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 ή 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

Εντολή WAIT

- Η εντολή WAIT προσφέρει τη δυνατότητα ενεργοποίησης των υπολογισμών σε μια PROCESS σε εξάρτηση με τη δραστηριότητα κάποιου ή κάποιων σημάτων
- Όταν χρησιμοποιείται η WAIT, δεν γίνεται χρήση λίστας ευαισθησίας
- Τρεις τύποι της WAIT: WAIT ON, WAIT UNTIL και WAIT FOR
- Σύνταξη της WAIT:

```
wait [on signal_name {, signal_name}]
             [until conditional_expression]
             [for time_expression];
```

- Παραδείγματα
  - Αναστέλλει την εκτέλεση μέχρι να συμβεί αλλαγή στο σήμα a ή στο b

```
wait on a, b;
```

  - Αναστέλλει την εκτέλεση μέχρι να ικανοποιηθεί η συνθήκη x > 10

```
wait until x > 10;
```

  - Αναστέλλει την εκτέλεση για χρονικό διάστημα 10 ns

```
wait for 10 ns;
```

Εντολές για την επιβολή συγχρονισμού σε σύγχρονα κυκλώματα

- Στα σύγχρονα κυκλώματα οι υπολογισμοί νέων τιμών (και επόμενης κατάστασης) πραγματοποιούνται με τη μεταβολή του σήματος ρολογιού
- Εναλλακτικοί τρόποι επιβολής συγχρονισμού στην VHDL:
  - Με δήλωση του `clk` στη λίστα ευαισθησίας
    `if (clk'EVENT and clk='1')` ή `if (clk'EVENT and clk='0')`
    για θετική και αρνητική ακμοπυροδότηση, αντίστοιχα
  - Με χρήση εντολής WAIT χωρίς δήλωση του `clk` στη λίστα ευαισθησίας
    `wait on (clk'EVENT and clk='1')` ή `wait on (clk'EVENT and clk='0')`
    για θετική και αρνητική ακμοπυροδότηση, αντίστοιχα
  - Με χρήση των προκαθορισμένων συναρτήσεων RISING_EDGE και FALLING_EDGE
    `if rising_edge(clk)` ή `if falling_edge(clk)`
    για θετική και αρνητική ακμοπυροδότηση, αντίστοιχα

Δομές ελέγχου σε ακολουθιακό κώδικα: εντολή IF (1)

- Η εντολή IF αποτελεί τη θεμελιώδη δομή για την εκτέλεση κώδικα υπό συνθήκη
- Έχει την ίδια σημασιολογία αλλά διαφορετική σύνταξη από την if της ANSI C
- Σύνταξη της IF:

```
if condition then
  sequence_of_statements
[elsif condition then
  sequence_of_statements]
[else
  sequence of statements]
end if;
```

Δομές ελέγχου σε ακολουθιακό κώδικα: εντολή IF (2)

- Υπάρχουν τρεις τύποι της εντολής IF: `if ...then`, `if ...then ...else` και `if ...then ...elsif`
- Παραδείγματα

```
if ... then
  s1;
  s2;
  ...
  sn;
end if;
```

```
if ... then
  s1;
  s2;
  ...
  sm;
else
  sn;
  ...
  sk;
end if;
```

```
if ... then
  s1;
  s2;
  s3;
elsif ... then
  s4;
  s5;
elsif ... then
  s6;
else
  s7;
  ...
  sn;
end if;
```

Δομές ελέγχου σε ακολουθιακό κώδικα: εντολή CASE (1)

- Η εντολή CASE αποτελεί μία χρήσιμη εντολή για την περιγραφή δομών αποκωδικοποίησης (decoding) μέσα σε τμήματα ακολουθιακού κώδικα
- Η γενική σύνταξη της CASE:

```
case expression is
  when choices => sequence_of_statements
  ...
end case;
```

- Με τις δηλώσεις WHEN γίνεται προσδιορισμός τιμής, μεμονωμένων τιμών ή περιοχής τιμών για τις οποίες θέλουμε να πραγματοποιηθούν οι δηλώσεις του δεύτερου μέλους

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

Δομές ελέγχου σε ακολουθιακό κώδικα: εντολή CASE (2)

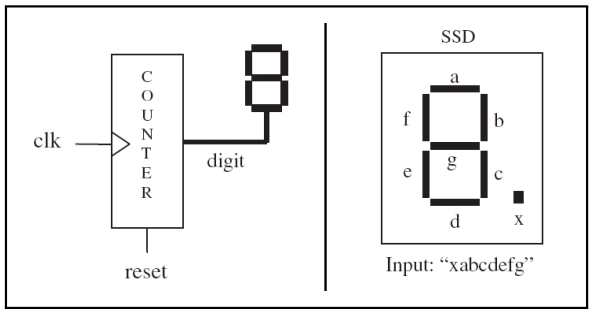
- Όταν δεν θέλουμε να υλοποιηθεί κάποια εντολή, χρησιμοποιούμε τη λέξη-κλειδί `NULL` όπως για παράδειγμα: `when others => NULL;`
- Απλό παράδειγμα εντολής CASE:

```
CASE control IS
  WHEN "00" => x <= a;
               y <= b;
  WHEN "01" => x <= b;
               y <= c;
  WHEN "10" => NULL;
  WHEN others => x <= "0000";
                 y <= "ZZZZ";
end CASE;
```

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

- Δεκαδικός απαριθμητής ενός ψηφίου (μετρά $0 \rightarrow 9$) και μετατρέπει τους δυαδικά κωδικοποιημένους δεκαδικούς (BCD: Binary Coded Decimal) σε μορφή κατάλληλη για απεικόνιση σε οθόνη επτά τμημάτων (SSD: Seven Segment Display)
- Το κύκλωμα διασυνδέεται με την οθόνη 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;
```

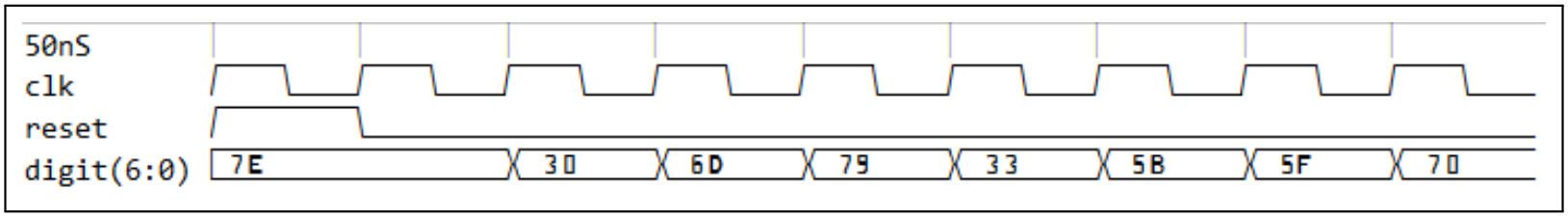


Παράδειγμα περιγραφής κυκλώματος: Απαριθμητής ψηφίου με έξοδο σε οθόνη επτά τμημάτων (2)

```
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 => NULL;
    end case;
  end process;
end rtl;
```

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

Διάγραμμα χρονισμού για το κύκλωμα του απαριθμητή ψηφίου



Δομές επανάληψης: εντολή LOOP

- Η εντολή LOOP προσφέρει έναν βολικό τρόπο για την περιγραφή επαναληπτικών κυκλωματικών δομών
- Η γενική σύνταξη της LOOP:

```
[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`
- Η ακολουθία των εντολών που περικλείεται σε μια LOOP θα εκτελεστεί ΜΗΔΕΝ ή περισσότερες φορές

Τα σχήματα επανάληψης FOR και WHILE

- Παράδειγμα υπολογισμού τετραγώνων ακεραίων

Περιγραφή με FOR

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

Περιγραφή με WHILE

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

- ☞ Η εντολή LOOP επιτρέπεται μόνο μέσα σε PROCESS
- ☞ Η μεταβλητή δείκτη `i` είναι προκαθορισμένη και δεν χρειάζεται να δηλωθεί με VARIABLE

Δομές επανάληψης: Η εντολή NEXT

- Η εντολή NEXT χρησιμοποιείται για την παράλειψη βημάτων του βρόχου
- Σύνταξη της NEXT:

```
next [loop_label] [when condition];
```

- Στο παράδειγμα που ακολουθεί η εντολή NEXT προκαλεί την παράλειψη μιας επανάληψης όταν `i = skip`

```
FOR i IN 0 to 15 LOOP
  NEXT WHEN i = skip;
  (...)
END LOOP;
```

- Χρήση παρόμοια με την `continue` στην ANSI C

Η εντολή EXIT

- Η εντολή EXIT χρησιμοποιείται για τον πρόωρο τερματισμό της εκτέλεσης του βρόχου
- Σύνταξη της EXIT:

```
exit [loop_label] [when condition];
```

- Στο παράδειγμα που ακολουθεί η εντολή EXIT προκαλεί μια οριστική έξοδο από τον βρόχο. Η εκτέλεση του βρόχου ολοκληρώνεται όταν η τιμή του διανύσματος data γίνει διαφορετική από μηδέν

```
FOR i IN 31 DOWNTO 0 LOOP
  CASE data(i) IS
    WHEN '0' => count := count + 1;
    WHEN OTHERS => EXIT;
  END CASE;
END LOOP;
```

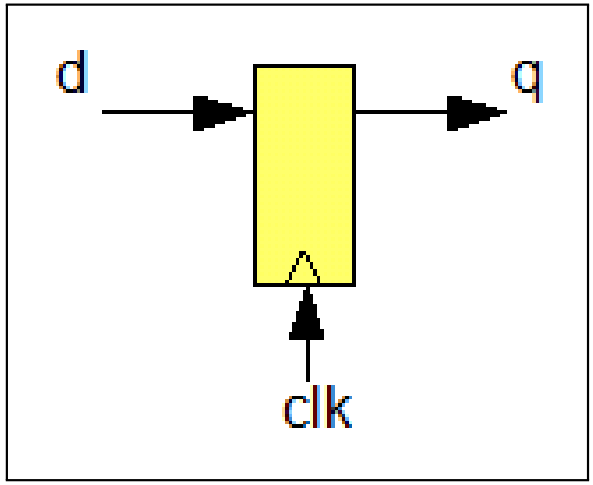
- Χρήση παρόμοια με την `break` στην ANSI C

Απλά ακολουθιακά κυκλώματα: Καταχωρητής ενός bit

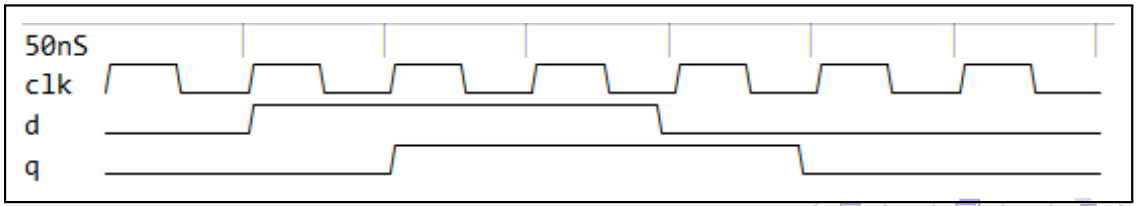
- Περιγραφή D-type flip-flop του 1-bit

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

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

architecture rtl of dreg is
begin
  process (clk, d)
  begin
    if (clk'event and clk = '1') then
      q <= d;
    end if;
  end process;
end rtl;
```

- Σχηματικό διάγραμμα



- Διάγραμμα χρονισμού του κυκλώματος



Καταχωρητής με ασύγχρονη και σύγχρονη επαναφορά (reset)

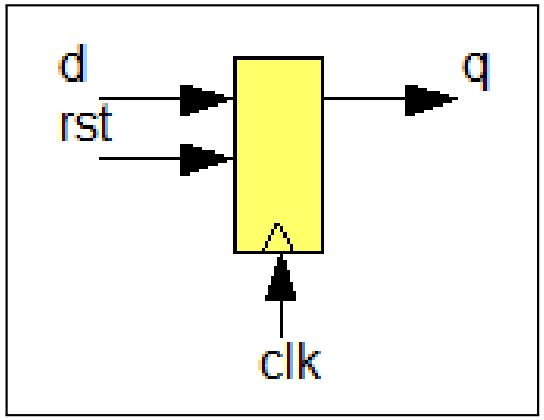
Ασύγχρονη επαναφορά

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

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

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

Σύγχρονη επαναφορά

```
...
process (clk, rst, d)
begin
  if (clk'event and clk = '1') then
    if (rst = '1') then
      temp <= '0';
    else
      temp <= d;
    end if;
  end if;
end process;
q <= temp;
end rtl;
```

Σχηματικό διάγραμμα

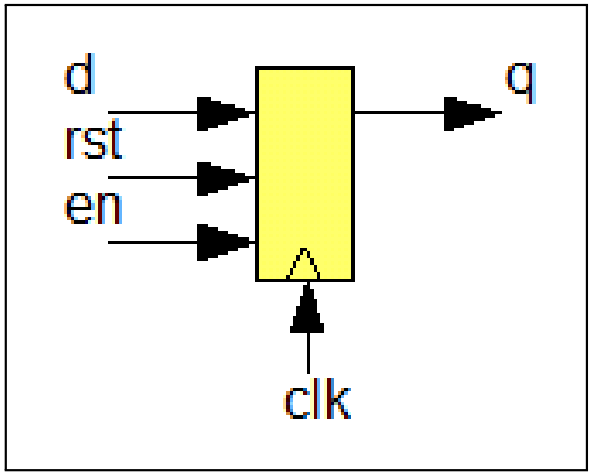


Καταχωρητής με επίτρεψη φόρτωσης (load enable)

Με επίτρεψη φόρτωσης

```
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)
  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



Δομές συντρέχοντος κώδικα: Εντολή WHEN/ELSE

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

```
target <= {expression when condition else} expression;
```

- Παράδειγμα

```
outp <= "000" WHEN (inp='0' OR reset='1') ELSE
        "001" WHEN (ctl='1') ELSE
        "010";
```

Δομές συντρέχοντος κώδικα: Εντολή WITH/SELECT

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

```
WITH expression SELECT
  target <= {expression WHEN choices,} expression;
```

- Παράδειγμα

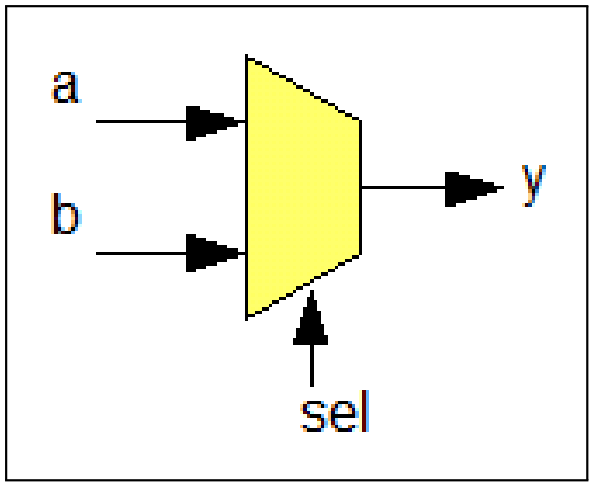
```
WITH control SELECT
  output <= reset WHEN "000",
            set WHEN "111",
            UNAFFECTED WHEN others;
```

Απλά συνδυαστικά κυκλώματα: Πολυπλέκτης 2-σε-1 (1)

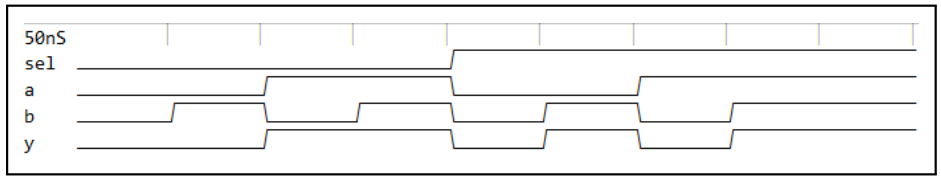
- Υλοποίηση 1-bit πολυπλέκτη 2-σε-1 με εντολή if-then-else

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

entity mux2to1 is
  port (
    sel,a,b : in std_logic;
    y       : out std_logic
  );
end mux2to1;

architecture arch_if of mux2to1 is
begin
  process (sel, a, b)
  begin
    if (sel = '0') then
      y <= a;
    elsif (sel = '1') then
      y <= b;
    else
      y <= 'Z';
    end if;
  end process;
end arch_if;
```

- Σχηματικό διάγραμμα



- Διάγραμμα χρονισμού του κυκλώματος



Απλά συνδυαστικά κυκλώματα: Πολυπλέκτης 2-σε-1 (2)

- Υλοποίηση της mux2to1 με εντολή case

```
architecture arch_case of mux2to1 is
begin
  process (sel, a, b)
  begin
    case sel is
      when '0' => y <= a;
      when '1' => y <= b;
      when others => y <= 'Z';
    end case;
  end process;
end arch_case;
```

- Υλοποίηση της mux2to1 με εντολή with-select

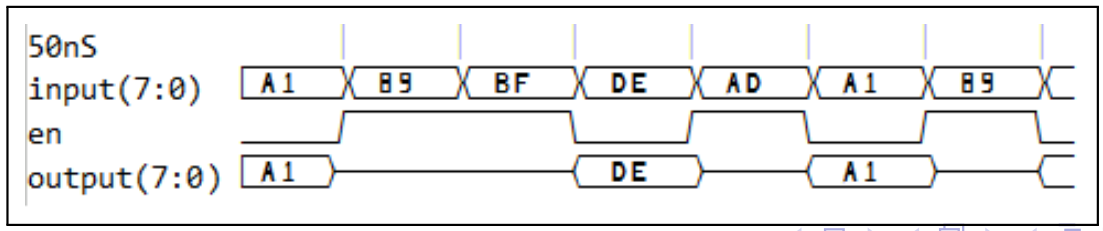
```
architecture arch_withsel of mux2to1 is
begin
  with sel select
    y <= a when '0',
         b when '1',
        'Z' when others;
end arch_withsel;
```

Απλά συνδυαστικά κυκλώματα: Τρισταθής απομονωτής

- Στον τρισταθή απομονωτή η έξοδος ισούται με την είσοδο όταν το σήμα επίτρεψης είναι `en = '0'` αλλιώς η έξοδος οδηγείται σε κατάσταση υψηλής αντίστασης (high impedance state) λόγω της μη οδήγησής της

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

entity tristate is
  port (
    input  : in std_logic_vector(7 downto 0);
    en     : in std_logic;
    output : out std_logic_vector(7 downto 0)
  );
end tristate;

architecture rtl of tristate is
begin
  output <= input when (en = '0') else (others => 'Z');
end rtl;
```

- Διάγραμμα χρονισμού του κυκλώματος

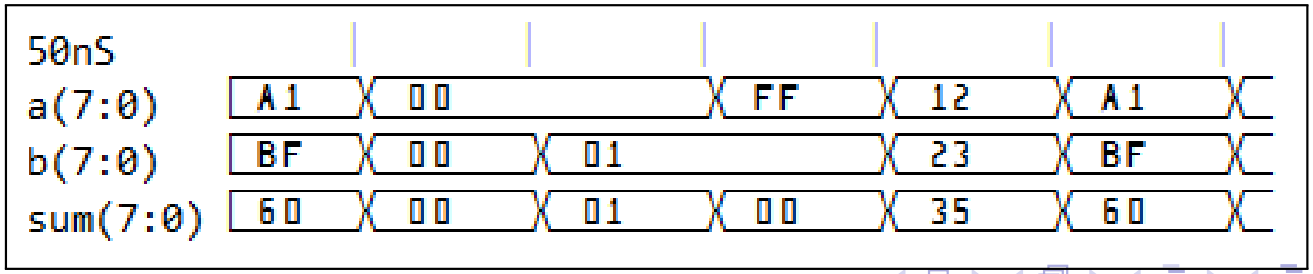


Αθροιστές απρόσημων και προσημασμένων (2's complement) ακεραίων (1)

- 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

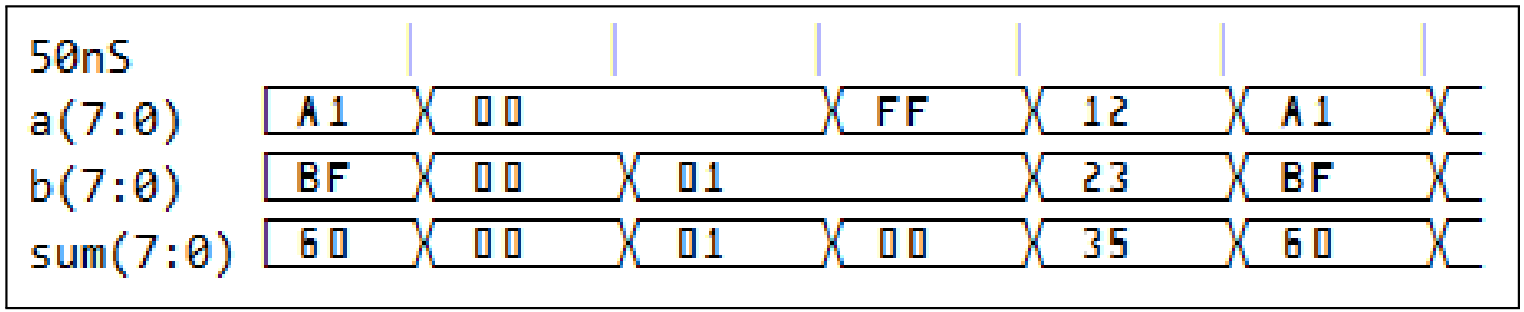


Αθροιστές απρόσημων και προσημασμένων (2's complement) ακεραίων (2)

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

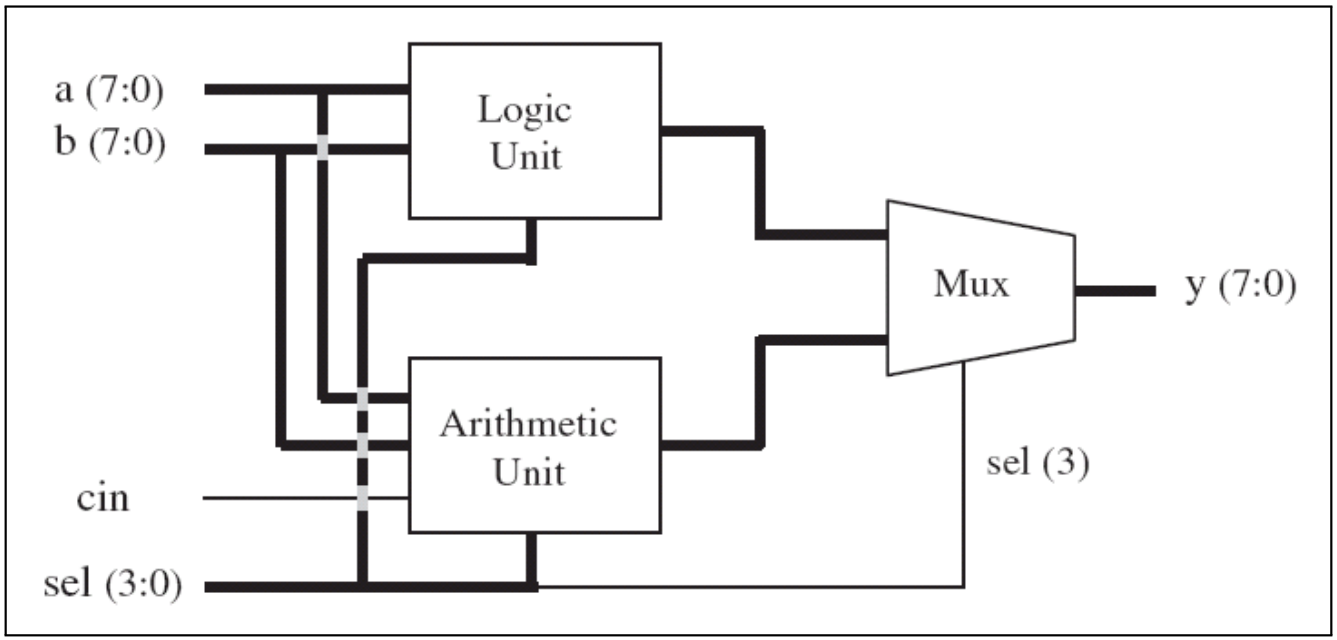
```
...
  signal temp : std_logic_vector(8 downto 0);
begin
  temp <= (a(7) & a) + (b(7) & b);
  sum <= temp(7 downto 0);
end rtl;
```

- 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