Γλώσσες Περιγραφής Υλικού

Σύνταξη κώδικα για λογική σύνθεση

Νικόλαος Καββαδίας
nkavv@physics.auth.gr
nkavv@uop.gr

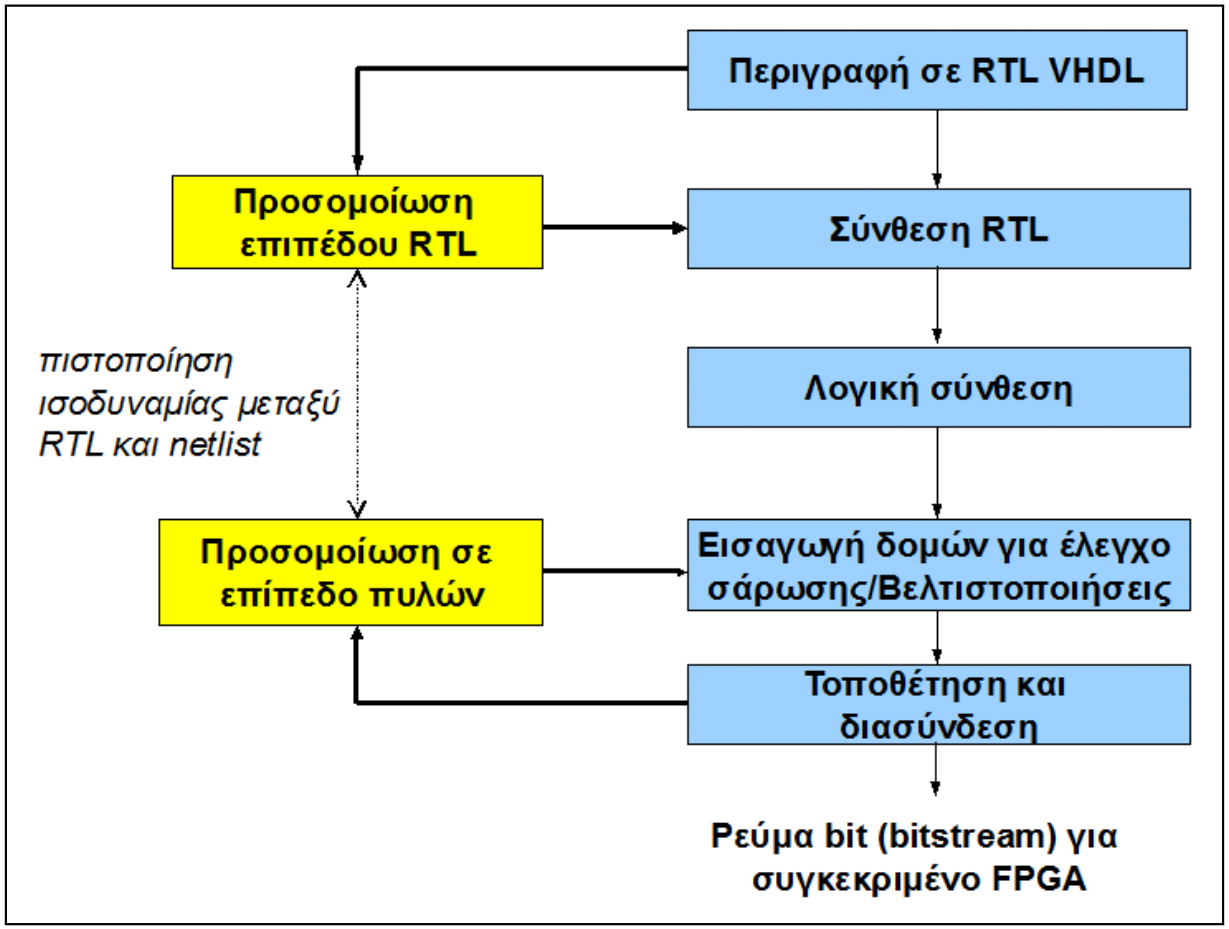
28 Απριλίου 2009

Σκιαγράφηση της διάλεξης

- Σύνταξη κώδικα για λογική σύνθεση
  - Λογική σύνθεση και κανόνες σύνταξης συνθέσιμων περιγραφών
  - Βασικά κυκλώματα για διαδρόμους δεδομένων
  - Επικοινωνία μεταξύ διεργασιών
  - Μνήμη μόνο ανάγνωσης (ROM)
  - Μνήμη τυχαίας προσπέλασης (RAM)
  - Αρχείο καταχωρητών (register file)

Η ροή της διαδικασίας λογικής σύνθεσης

- Η συγκεκριμένη ροή εστιάζει στις τεχνολογίες FPGA



Κανόνες για τη σύνταξη συνθέσιμων περιγραφών (1)

1. Χρήση ενός σήματος ρολογιού
2. Αποθήκευση τιμών στο κύκλωμα σε καταχωρητές ή μνήμες
3. Ανάθεση μιας τιμής ανά σήμα σε κάθε κύκλο ρολογιού
4. Χρησιμοποίηση μόνο σύγχρονης επανατοποθέτησης (synchronous reset)
5. Χρήση μόνο ακμοπυροδότησης στα flip-flop
6. Να μην παράγονται νέα σήματα χρονισμού με βάση το εξωτερικό ρολόι, αλλά αντί αυτού να χρησιμοποιούνται σήματα επίτρεψης/φόρτωσης για την επιλεκτική ενεργοποίηση κάποιας υπομονάδας

Κανόνες για τη σύνταξη συνθέσιμων περιγραφών (2)

7. Σε μια λίστα ευαισθησίας αναφέρονται όλα τα σήματα ή είσοδοι οι οποίες 'διαβάζονται' μέσα στη διεργασία
8. Σε μια λίστα ευαισθησίας μιας αποκλειστικά σύγχρονης διεργασίας επιτρέπεται να περιληφθεί μόνο το σήμα ρολογιού (`clk`)
9. Για την τρέχουσα και επόμενη κατάσταση ενός FSM, να χρησιμοποιείται απαριθμητός τύπος δεδομένων
10. Σε ένα κύκλωμα θα πρέπει να γίνεται ανάθεση σε όλα τα σήματα εξόδου για όλες τις περιπτώσεις λειτουργίας για την αποφυγή δημιουργίας ανεπιθύμητων μανδαλωτών
11. Επιτρέπεται η αρχική ανάθεση σε σήμα για την κάλυψη όλων των πιθανών περιπτώσεων
12. Να μην χρησιμοποιούνται οι τιμές `'X'` και `'Z'` ενός σήματος για τον έλεγχο περιπτώσεων (δήλωση WHEN σε μια CASE)

Ανάθεση τιμής για κάθε πιθανή περίπτωση

- ΛΑΘΟΣ

```
process (state, input)
begin
  case state is
    when S1 =>
      if (input = '1') then
        output <= '0';
      end if;
    when S2 =>
      output <= '1';
  end case;
end process;
```

- ΣΩΣΤΟ

```
process (state, input)
begin
  case state is
    when S1 =>
      if (input = '1') then
        output <= '0';
      else
        output <= '1';
      end if;
    when S2 =>
      output <= '1';
  end case;
end process;
```

Χρήση τιμών προεπιλογής για την αρχική ανάθεση σήματος

- Χρήση τιμής προεπιλογής για την έξοδο `output` στο προηγούμενο παράδειγμα

```
process (state, input)
begin
  output <= '1';
  case state is
    when S1 =>
      if (input = '1') then
        output <= '0';
      end if;
    end case;
end process;
```

Λανθασμένος και ορθός έλεγχος περιπτώσεων σε μία εντολή CASE/WHEN

- ΛΑΘΟΣ

```
  process (A,B,Sel) begin
    case Sel is
      when "00" =>
        Res <= A + B;
      when "01" =>
        Res <= A + (not B) + 1;
      when "1X" =>
        Res <= A and B;
      when "1Z" =>
        Res <= A or B;
      when others =>
        Res <= "XX";
    end case;
end process;
```

- ΣΩΣΤΟ

```
  process (A,B,Sel) begin
    case Sel is
      when "00" =>
        Res <= A + B;
      when "01" =>
        Res <= A + (not B) + 1;
      when "10" =>
        Res <= A and B;
      when "11" =>
        Res <= A or B;
      when others =>
        Res <= "XX";
    end case;
end process;
```

Παραδείγματα κυκλωμάτων που χρησιμοποιούνται σε διαδρόμους δεδομένων

- Αποκωδικοποιητής
- Κωδικοποιητής προτεραιότητας
- Καταχωρητής ολίσθησης
- Πολλαπλασιαστής πολλαπλών κύκλων

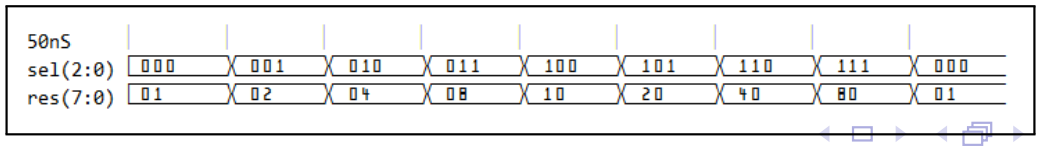
Αποκωδικοποιητής 3-σε-8 (3-to-8 decoder)

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

entity dec3_8 is
  port (
    sel : in  unsigned(2 downto 0);
    res : out unsigned(7 downto 0)
  );
end dec3_8;

architecture comb of dec3_8 is
begin
  res <= "00000001" when sel = "000" else
         "00000010" when sel = "001" else
         "00000100" when sel = "010" else
         "00001000" when sel = "011" else
         "00010000" when sel = "100" else
         "00100000" when sel = "101" else
         "01000000" when sel = "110" else "10000000";
end comb;
```

Διάγραμμα χρονισμού του κυκλώματος



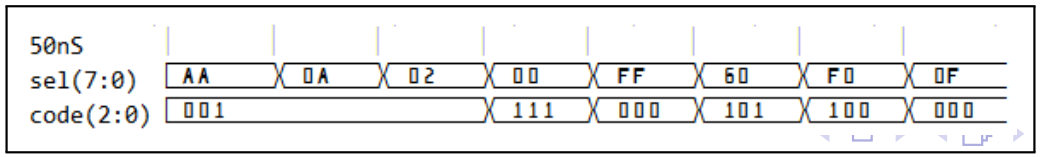
Κωδικοποιητής προτεραιότητας (priority encoder)

```
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;
```

Διάγραμμα χρονισμού του κυκλώματος



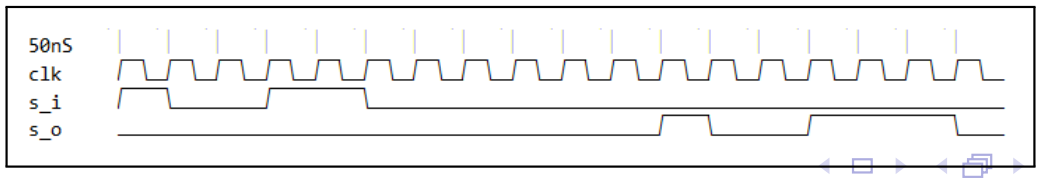
Καταχωρητής ολίσθησης των 8-bit με σειριακή είσοδο και έξοδο

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

entity shifter is
  port (
    clk, s_i : in std_logic;
    s_o      : out std_logic
  );
  end shifter;

architecture impl of shifter is
  signal tmp : std_logic_vector(7 downto 0) := "00000000";
begin
  process (clk)
  begin
    if rising_edge(clk) then
      tmp <= tmp(6 downto 0) & s_i;
    end if;
  end process;
  s_o <= tmp(7);
end impl;
```

Διάγραμμα χρονισμού του κυκλώματος



Πολλαπλασιαστής πολλών κύκλων διαστάσεων $N \times N$

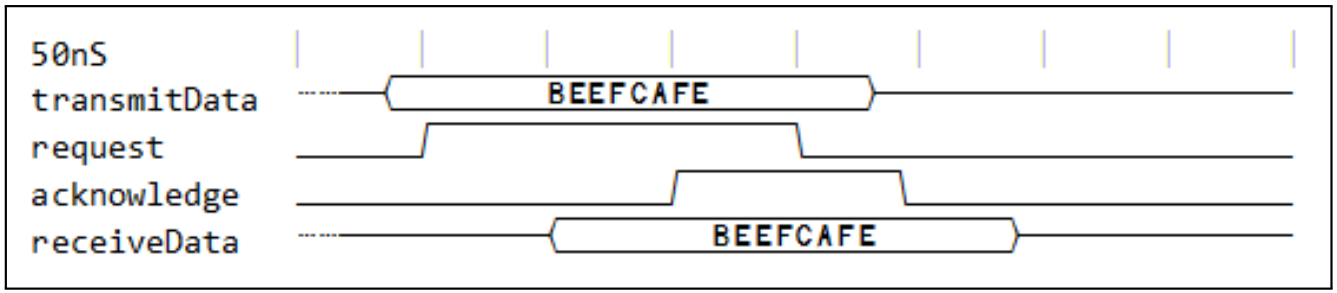
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity multiplier is
  generic (N: integer := 32);
  port (
    a,b : in std_logic_vector(N-1 downto 0);
    m   : out std_logic_vector(2*N-1 downto 0)
  );
end multiplier;

architecture behavioral of multiplier is
begin
  process(a,b)
    variable bReg: std_logic_vector(2*N-1 downto 0);
    variable aReg: std_logic_vector(N-1 downto 0);
  begin
    aReg:=a;
    bReg:=(2*N-1 downto 0 => '0') & b;
    for index in 1 to N loop
      if (bReg(0)= '1') then
        bReg(2*N-1 downto N) := bReg(2*N-1 downto N) + aReg(N-1 downto 0);
      end if;
      bReg(2*N-1 downto 0):= '0' & bReg(2*N-1 downto 1);
    end loop;
    m <= bReg;
  end process;
end behavioral;
```

Επικοινωνία μεταξύ διεργασιών (1)

- Η επικοινωνία (ανταλλαγή πληροφορίας) μεταξύ δύο διεργασιών γίνεται με τη βοήθεια σημάτων
- Για την αναμονή όσον αφορά τη μεταβολή της τιμής ενός σήματος η οποία γίνεται εκτός διεργασίας, χρήσιμη είναι η εντολή `WAIT UNTIL`
- Στο παρακάτω διάγραμμα χρονισμού δίνεται ένα παράδειγμα ασύγχρονης επικοινωνίας τύπου 'αίτησης και γνωστοποίησης' (request and acknowledgement)



Επικοινωνία μεταξύ διεργασιών (2)

```
 entity handshake is
  port(
    inputData: in std_logic_vector(31 downto 0)
  );
  end handshake;

architecture protimpl of handshake is
  signal transmitData: std_logic_vector(31 downto 0);
  signal request, acknowledge: std_logic;
begin
```

```
producer: process
begin
  wait until inputData'EVENT;
  transmitData <= inputData;
  request <= '1';
  wait until (acknowledge = '1');
  request <= '0';
  wait until (acknowledge = '0');
end process;
```

```
consumer: process
  variable receiveData:
    std_logic_vector(31 downto 0);
begin
  wait until (request = '1');
  receiveData := transmitData;
  acknowledge <= '1';
  wait until (request = '0');
  acknowledge <= '0';
end process;
```

Σχεδίαση μνήμης

- Τα κυκλώματα μνήμης προσφέρουν τη δυνατότητα σε ένα σύστημα να αποθηκεύει δεδομένα και αποτελέσματα ώστε να μπορεί να τα επαναχρησιμοποιήσει σε νέους υπολογισμούς
- Βασικές δομές μνήμης
  - Μνήμη μόνο ανάγνωσης (ROM)
  - Μνήμη τυχαίας προσπέλασης (RAM) για ανάγνωση και εγγραφή
- Δομές μνήμης που ανάγονται στις βασικές
  - Πίνακας αναζήτησης (LUT: Look-Up Table): συνήθως αναφέρεται σε μνήμη ROM ή RAM της οποίας τα περιεχόμενα δεν μεταβάλλονται, η οποία είναι ασύγχρονης ανάγνωσης
  - Αρχείο καταχωρητών (register file): μνήμη RAM με πολλαπλές θύρες εισόδου (εγγραφής) ή/και εξόδου (ανάγνωσης)

Βασικά στοιχεία στο σχεδιασμό κυκλωμάτων μνήμης

- Τρόποι ανάγνωσης
  - Ασύγχρονη ανάγνωση: αποτελέσματα διαθέσιμα στον ίδιο κύκλο στον οποίο διευθυνσιοδοτήθηκαν με κάποια συνδυαστική χρονική καθυστέρηση
  - Σύγχρονη ανάγνωση: αποτελέσματα διαθέσιμα στον επόμενο κύκλο ρολογιού
- Σήματα επίτρεψης

RAM Επίτρεψη ανάγνωσης (read enable ή `re`)

RAM Επίτρεψη εγγραφής (write enable ή `we`)

RAM,ROM Επίτρεψη εξόδου (output enable ή `oe`)

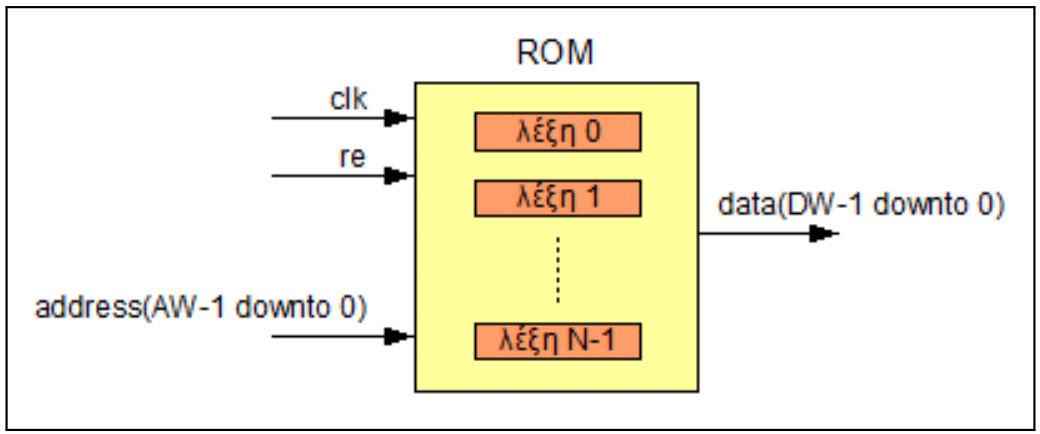
- Παράμετροι
  - Αριθμός θέσεων (καταχωρήσεων): `N` ή `NR`
  - Εύρος λέξης διεύθυνσης (address width): `AW`
  - Εύρος λέξης δεδομένων (data width): `DW`
  - Η παράμετρος AW ορισμένες φορές υπολογίζεται από την `NR` μέσω της έκφρασης: $AW = \lceil log_2(NR) \rceil$
  - Αριθμός θυρών εισόδου (`NWP`) και εξόδου (`NRP`)

Μνήμη μόνο ανάγνωσης (ROM) (1)

- Μία ROM διαθέτει τουλάχιστον μία είσοδο για τη διευθυνσιοδότηση (`address`) και μία έξοδο για την ανάγνωση των δεδομένων από τη συγκεκριμένη θέση στη μνήμη (data)
- Μπορεί να διαθέτει είσοδο ρολογιού (`clk`) και επίτρεψη ανάγνωσης (`re`)
- Τα περιεχόμενα της ROM υλοποιούνται είτε ως CONSTANT του κατάλληλου τύπου είτε ως αποκωδικοποιητής του σήματος `address`
- Όταν `re = '0'`, τα δεδομένα στην έξοδο επιλέγουμε είτε να παραμένουν αμετάβλητα είτε να μην οδηγούνται (υψηλή αντίσταση)
- Η πολλαπλή ανάγνωση από μία θέση στη μνήμη ταυτόχρονα δεν δημιουργεί πρόβλημα διαμάχης (conflict)

Μνήμη μόνο ανάγνωσης (ROM) (2)

Διεπαφή της ROM



Σύγχρονη μνήμη ROM των 8-bit με 16 θέσεις και χρήση CONSTANT (1)

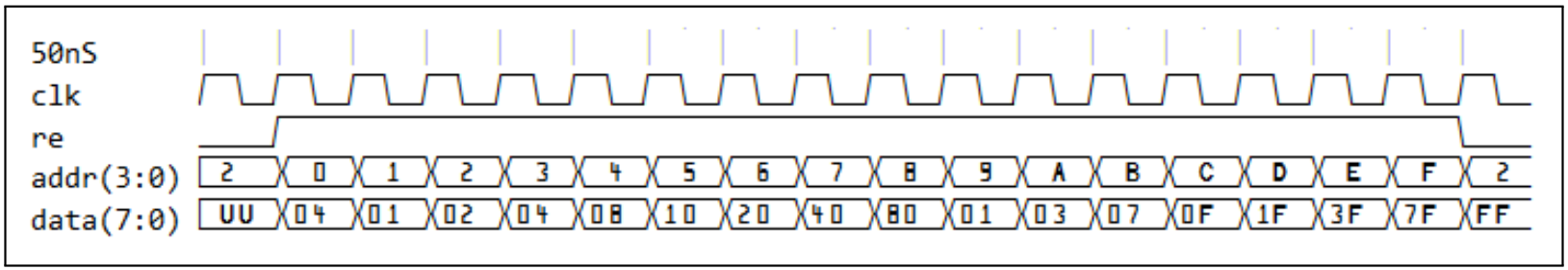
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity rom_16_8 is
  port (
    clk, re : in std_logic;
    addr    : in std_logic_vector(3 downto 0);
    data    : out std_logic_vector(7 downto 0)
  );
end rom_16_8;

architecture impl of rom_16_8 is
  type rom_type is array (0 to 15) of std_logic_vector(7 downto 0);
  constant ROM : rom_type :=
  (X"01", X"02", X"04", X"08", X"10", X"20", X"40", X"80",
   X"01", X"03", X"07", X"0F", X"1F", X"3F", X"7F", X"FF");
begin
  process (clk)
  begin
    if (clk='1' and clk'EVENT) then
      if (re = '1') then
        data <= ROM(conv_integer(addr));
      end if;
    end if;
  end process;
end impl;
```

Σύγχρονη μνήμη ROM των 8-bit με 16 θέσεις και χρήση CONSTANT (2)

Διάγραμμα χρονισμού



Ασύγχρονη μνήμη ROM των 8-bit με 16 θέσεις και αποκωδικοποίηση διεύθυνσης

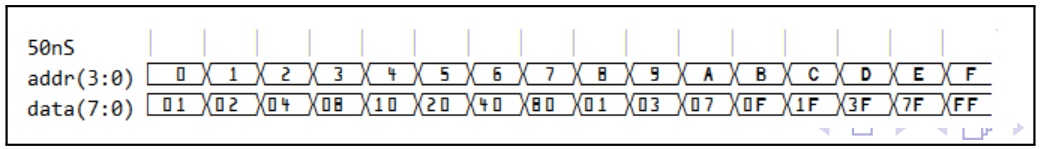
```
library ieee;
use ieee.std_logic_1164.all;

entity rom_16_8 is
  port (
    addr : in std_logic_vector(3 downto 0);
    data : out std_logic_vector(7 downto 0)
  );
end rom_16_8;

architecture impl of rom_16_8 is
begin
  process (addr)
  begin
```

```
    case addr is
      when "0000" => data <= X"01";
      when "0001" => data <= X"02";
      when "0010" => data <= X"04";
      when "0011" => data <= X"08";
      when "0100" => data <= X"10";
      when "0101" => data <= X"20";
      when "0110" => data <= X"40";
      when "0111" => data <= X"80";
      when "1000" => data <= X"01";
      when "1001" => data <= X"03";
      when "1010" => data <= X"07";
      when "1011" => data <= X"0F";
      when "1100" => data <= X"1F";
      when "1101" => data <= X"3F";
      when "1110" => data <= X"7F";
      when "1111" => data <= X"FF";
    end case;
  end process;
end impl;
```

Διάγραμμα χρονισμού



Μνήμη τυχαίας προσπέλασης (RAM)

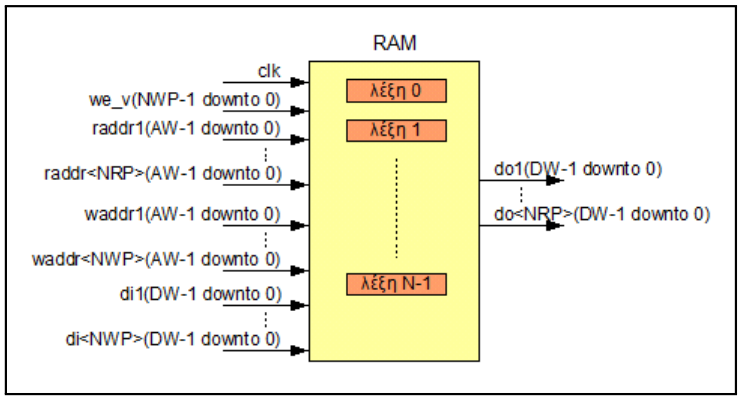
- Μία RAM διαθέτει τουλάχιστον μία είσοδο για τη διευθυνσιοδότηση (`address`) και τουλάχιστον μία θύρα για την ανάγνωση ή/και εγγραφή δεδομένων από και προς συγκεκριμένη θέση στη μνήμη θέση στη μνήμη
- Υποχρεωτικά διαθέτει είσοδο ρολογιού (`clk`) και επίτρεψη εγγραφής (`we`) για κάθε θύρα εγγραφής
- Μπορεί να διαθέτει `reset` είτε για τον καθαρισμό των περιεχομένων όλων των θέσεων
- Τα περιεχόμενα της RAM υλοποιούνται ως SIGNAL
- Μπορεί να οριστεί και επίτρεψη ανάγνωσης θύρας εξόδου
- Οι πολλαπλές αιτήσεις για εγγραφή στην ίδια θέση δημιουργούν πρόβλημα διαμάχης και επιλύονται με κατάλληλη λογική ελέγχου (προτεραιότητα)
- Η ταυτόχρονη εγγραφή και ανάγνωση από μία θέση μνήμης πρέπει να επιλύεται στο υλικό

Διεπαφή και οργάνωση μιας RAM

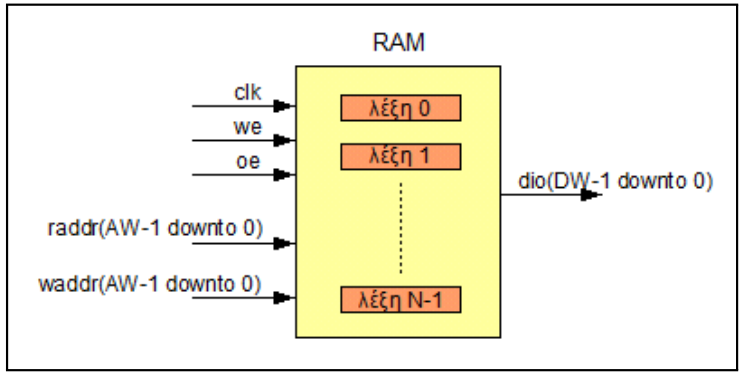
- Μία RAM μπορεί να χρησιμοποιεί ξεχωριστούς διαύλους για την είσοδο και έξοδο δεδομένων ή αμφίδρομο δίαυλο ο οποίος χρησιμοποιείται τόσο για την είσοδο όσο και για την έξοδο δεδομένων (θύρα τύπου inout)
- Διαθέτει τουλάχιστον μία είσοδο για τη διευθυνσιοδότηση
- ☞ Πολλαπλά σήματα επίτρεψης (π.χ. `we_1, we_2, ... we_nwp`) μπορούν να αντικατασταθούν από ένα διάνυσμα επίτρεψης (we_v(NWP-1 downto 0))
- ℹ Σε σύγχρονες τεχνολογίες FPGA, τα ολοκληρωμένα έχουν διαθεσιμότητα ενσωματωμένων μπλοκ μνήμης block RAM

Διεπαφή και οργάνωση μιας RAM (2)

- Διεπαφή RAM με πολλαπλές θύρες ανάγνωσης και εγγραφής



- Διεπαφή RAM με μία αμφίδρομη (δικατευθυντική) θύρα



Συμπεριφορά των θυρών ανάγνωσης σε διαμάχη ανάγνωσης-εγγραφής (1)

- Γενικά δεν είναι ασφαλές να πραγματοποιείται ανάγνωση από μία θέση μνήμης η οποία βρίσκεται σε κατάσταση εγγραφής κατά το συγκεκριμένο κύκλο
- Προκειμένου τη μεγιστοποίηση των επιδόσεων της RAM μπορεί να πραγματοποιηθεί και λειτουργία ανάγνωσης κάτω από κάποιες προϋποθέσεις
- Σε μνήμες σύγχρονης ανάγνωσης καθορίζεται ο 'Τρόπος Εγγραφής' (WRITE MODE) ο οποίος καθορίζει ποια δεδομένα γίνονται διαθέσιμα στην έξοδο

Συμπεριφορά των θυρών ανάγνωσης σε διαμάχη ανάγνωσης-εγγραφής (2)

- Υπάρχουν τρεις τρόποι εγγραφής: WRITE FIRST (ή Read after Write), READ FIRST (ή Read before Write) και NO CHANGE (ή No Read on Write) οι οποίοι συνοψίζονται στον πίνακα

WRITE FIRST	Τα δεδομένα εισόδου γράφονται στη διευθυνσιοδοτούμενη θέση και ταυτόχρονα εμφανίζονται στην έξοδο (ανάγνωση μετά την εγγραφή)
READ FIRST	Τα περιεχόμενα της διευθυνσιοδοτούμενης θέσης μνήμης εμφανίζονται στην έξοδο. Τα δεδομένα εισόδου γράφονται στην ίδια θέση (ανάγνωση πριν την εγγραφή)
NO CHANGE	Η έξοδος παραμένει αμετάβλητη. Τα δεδομένα εισόδου γράφονται στη διευθυνσιοδοτούμενη θέση

Σκοπιμότητα των διαφορετικών τρόπων εγγραφής

- WRITE FIRST: είναι χρήσιμος στην περίπτωση που ζητείται η ασύγχρονη ανάγνωση των νέων περιεχομένων
- READ FIRST: είναι ο ασφαλέστερος τρόπος εγγραφής για γενικές σχεδιάσεις. Είναι ιδιαίτερα χρήσιμος στο σχεδιασμό κυκλικών ολισθητών και αρχείων καταχωρητών
- NO CHANGE: χρησιμοποιείται σε αρκετές εφαρμογές ψηφιακής επεξεργασίας σήματος όπου χρειάζεται η ενημέρωση πινάκων αναζήτησης με δείγματα κυματομορφών, συντελεστών φίλτρων κ.λ.π. χωρίς επίδραση στην τρέχουσα έξοδο

RAM με ασύγχρονη ανάγνωση (1)

Στα FPGA αυτή η μνήμη υλοποιείται σε λογικά κελιά (distributed RAM)

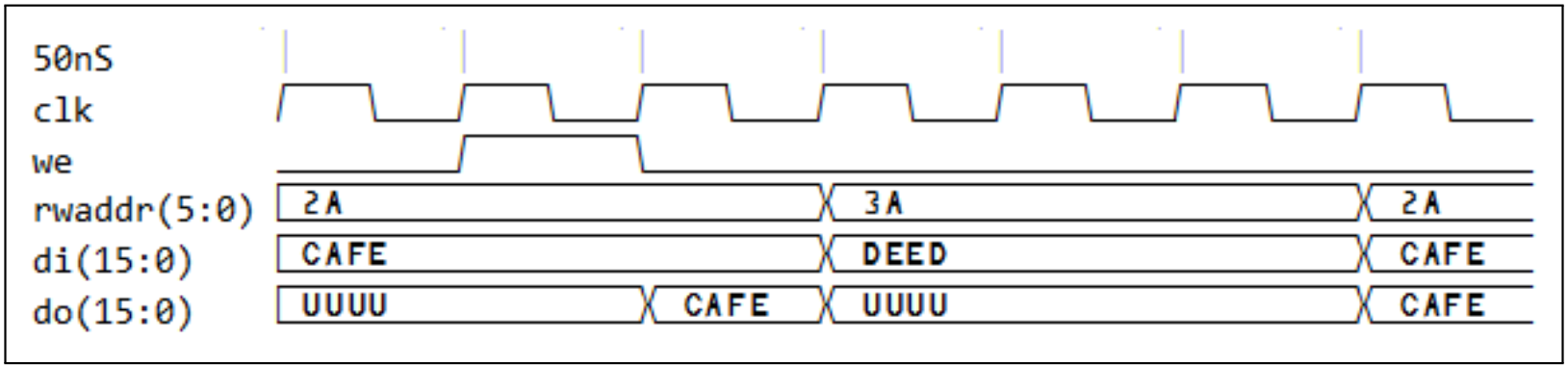
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram_async is
  port (
    clk, we : in std_logic;
    rwaddr : in std_logic_vector(5 downto 0);
    di : in std_logic_vector(15 downto 0);
    do : out std_logic_vector(15 downto 0)
  );
end ram_async;

architecture synth of ram_async is
  type ram_type is array (63 downto 0) of std_logic_vector(15 downto 0);
  signal RAM: ram_type;
begin
  process (clk)
  begin
    if (clk='1' and clk'EVENT) then
      if (we = '1') then
        RAM(conv_integer(rwaddr)) <= di;
      end if;
    end if;
  end process;
  do <= RAM(conv_integer(rwaddr));
end synth;
```

RAM με ασύγχρονη ανάγνωση (2)

Διάγραμμα χρονισμού



RAM με τρόπο εγγραφής READ FIRST

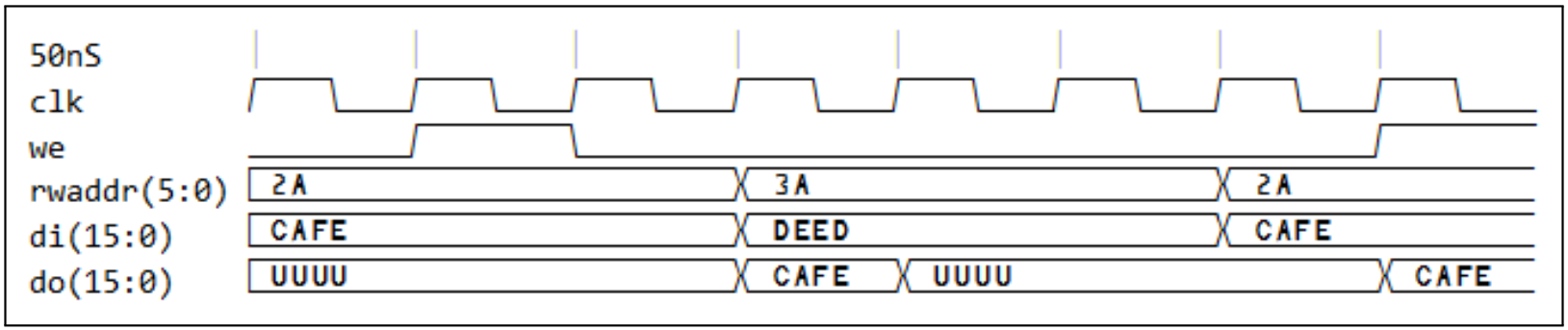
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram_rf is
  port (
    clk, we : in std_logic;
    rwaddr : in std_logic_vector(5 downto 0);
    di : in std_logic_vector(15 downto 0);
    do : out std_logic_vector(15 downto 0)
  );
end ram_rf;

architecture synth of ram_rf is
  type ram_type is array (63 downto 0) of std_logic_vector(15 downto 0);
  signal RAM: ram_type;
begin
  process (clk)
  begin
    if (clk='1' and clk'EVENT) then
      if (we = '1') then
        RAM(conv_integer(rwaddr)) <= di;
      end if;
      do <= RAM(conv_integer(rwaddr));
    end if;
  end process;
end synth;
```

RAM με τρόπο εγγραφής READ FIRST (2)

Διάγραμμα χρονισμού



RAM με τρόπο εγγραφής WRITE FIRST (1)

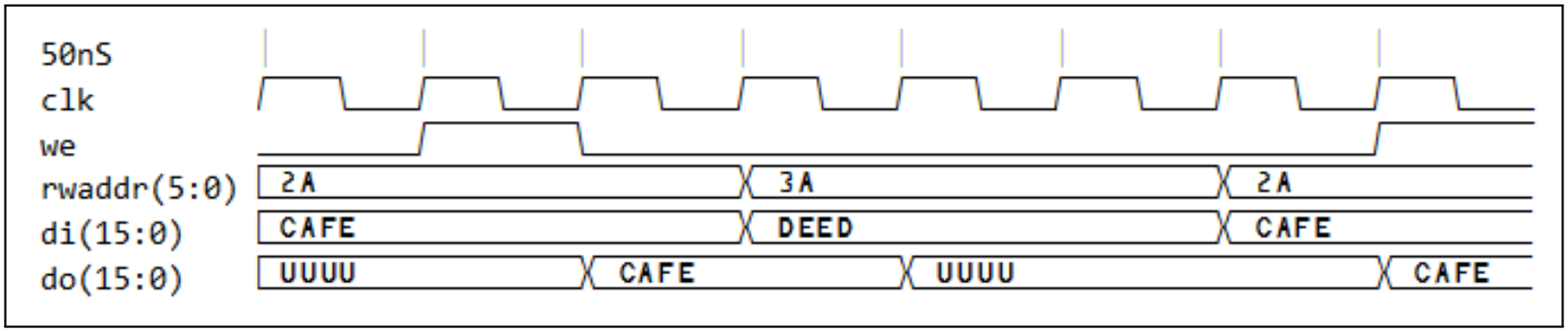
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram_wf is
  port (
    clk, we : in std_logic;
    rwaddr : in std_logic_vector(5 downto 0);
    di : in std_logic_vector(15 downto 0);
    do : out std_logic_vector(15 downto 0)
  );
end ram_wf;

architecture synth of ram_wf is
  type ram_type is array (63 downto 0) of std_logic_vector(15 downto 0);
  signal RAM: ram_type;
begin
  process (clk)
  begin
    if (clk='1' and clk'EVENT) then
      if (we = '1') then
        RAM(conv_integer(rwaddr)) <= di;
        do <= di;
      else
        do <= RAM(conv_integer(rwaddr));
      end if;
    end if;
  end process;
end synth;
```

RAM με τρόπο εγγραφής WRITE FIRST (2)

Διάγραμμα χρονισμού



RAM με τρόπο εγγραφής NO CHANGE (1)

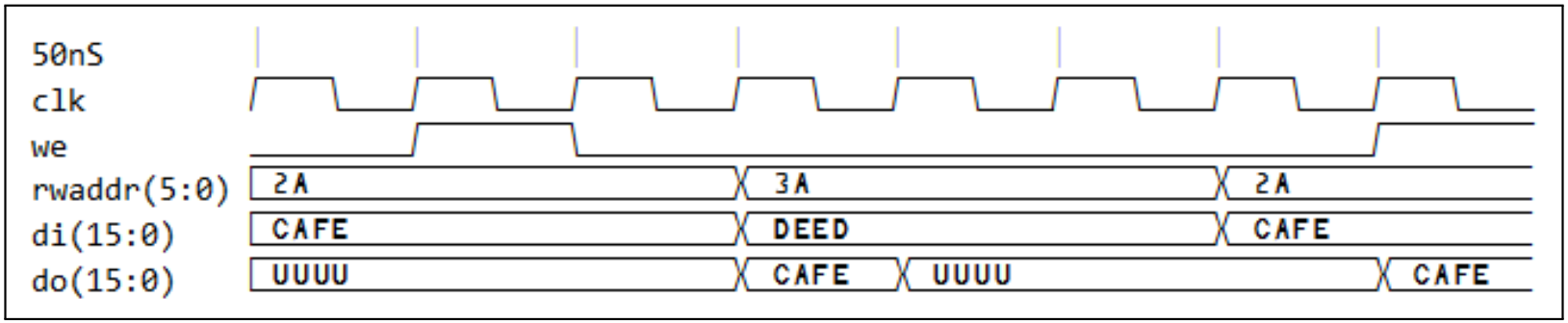
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram_nc is
  port (
    clk, we : in std_logic;
    rwaddr : in std_logic_vector(5 downto 0);
    di : in std_logic_vector(15 downto 0);
    do : out std_logic_vector(15 downto 0)
  );
end ram_nc;

architecture synth of ram_nc is
  type ram_type is array (63 downto 0) of std_logic_vector(15 downto 0);
  signal RAM: ram_type;
begin
  process (clk)
  begin
    if (clk = '1' and clk'EVENT) then
      if (we = '1') then
        RAM(conv_integer(rwaddr)) <= di;
      else
        do <= RAM(conv_integer(rwaddr));
      end if;
    end if;
  end process;
end synth;
```

RAM με τρόπο εγγραφής NO CHANGE (2)

Διάγραμμα χρονισμού



RAM με ξεχωριστές διευθύνσεις για τη θύρα εισόδου και τη θύρα εξόδου (1)

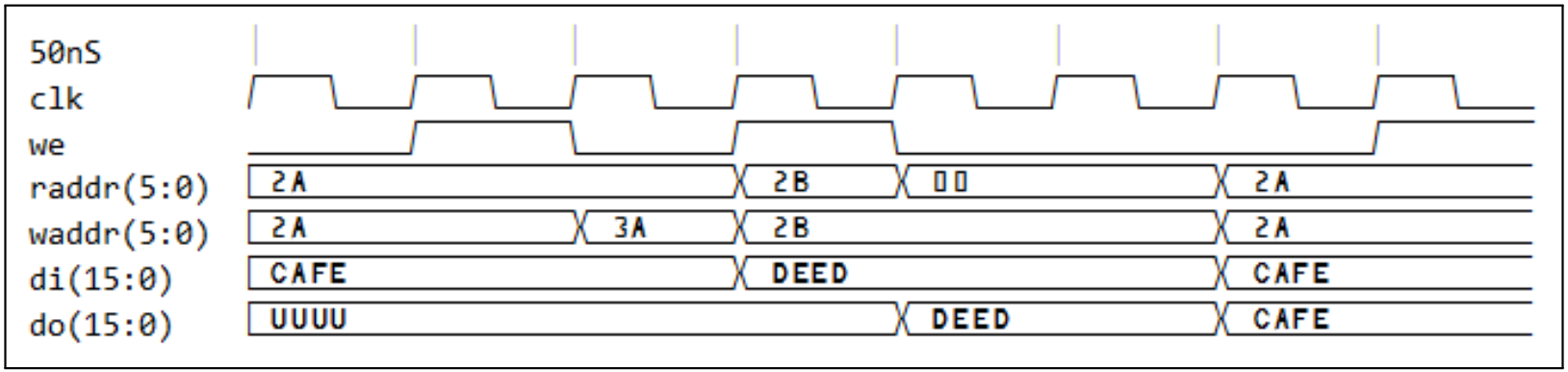
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram_dp_rf is
  port (
    clk, we : in std_logic;
    raddr : in std_logic_vector(5 downto 0);
    waddr : in std_logic_vector(5 downto 0);
    di : in std_logic_vector(15 downto 0);
    do : out std_logic_vector(15 downto 0)
  );
end ram_dp_rf;

architecture synth of ram_dp_rf is
  type ram_type is array (63 downto 0) of std_logic_vector(15 downto 0);
  signal RAM: ram_type;
begin
  process (clk)
  begin
    if (clk='1' and clk'EVENT) then
      if (we = '1') then
        RAM(conv_integer(raddr)) <= di;
      end if;
    end if;
  end process;
  do <= RAM(conv_integer(waddr));
end synth;
```

RAM με ξεχωριστές διευθύνσεις για τη θύρα εισόδου και τη θύρα εξόδου (2)

Διάγραμμα χρονισμού



Αρχείο καταχωρητών (register file) με δύο θύρες εισόδου και μία θύρα εξόδου (1)

Το αρχείο καταχωρητών ελέγχθηκε για: $NRP = 2$, $NWP = 1$, $DW = 8$, και $AW = 4$

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

entity regfile is
  generic (
    NWP : integer :=  1;
    NRP : integer :=  2;
    AW  : integer :=  8;
    DW  : integer := 32
  );
  port (
    clock        : in  std_logic;
    reset        : in  std_logic;
    en           : in  std_logic;
    we_v         : in  std_logic_vector(NWP-1 downto 0);
    waddr_v      : in  std_logic_vector(NWP*AW-1 downto 0);
    raddr_v      : in  std_logic_vector(NRP*AW-1 downto 0);
    input_data_v : in  std_logic_vector(NWP*DW-1 downto 0);
    ram_output_v : out std_logic_vector(NRP*DW-1 downto 0)
  );
end regfile;
```

Αρχείο καταχωρητών (register file) με δύο θύρες εισόδου και μία θύρα εξόδου (2)

```
architecture synth of regfile is
  type mem_type is array ((2**AW-1) downto 0) of
    std_logic_vector(DW-1 downto 0);
  signal ram_name : mem_type := (others => (others => '0'));
begin
  process (clock)
  begin
    if (clock'EVENT and clock = '1') then
      if (en = '1') then
        for i in 0 to NWP-1 loop
          if ((we_v(i) = '1')) then
            ram_name(conv_integer(waddr_v(AW*(i+1)-1 downto AW*i))) <=
            input_data_v(DW*(i+1)-1 downto DW*i);
          end if;
        end loop;
      end if;
    end if;
  end process;
  G_DO_NRP: for i in 0 to NRP-1 generate
   ram_output_v(DW*(i+1)-1 downto DW*i) <=
   ram_name(conv_integer(raddr_v(AW*(i+1)-1 downto AW*i)));
  end generate G_DO_NRP;
end synth;
```

Αρχείο καταχωρητών (register file) με δύο θύρες εισόδου και μία θύρα εξόδου (3)

- Σε μοντέρνα FPGA με άφθονους πόρους ενσωματωμένης μνήμης (block RAM) η υλοποίηση του γενικευμένου αρχείου καταχωρητών με NRP θύρες ανάγνωσης και NWP θύρες εγγραφής απαιτεί τη χρήση $NWP \times NRP$ block RAM
- Διάγραμμα χρονισμού του κυκλώματος