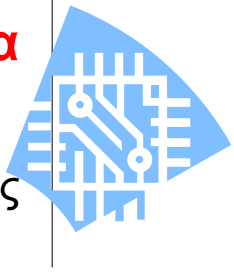
Ετερογενή Υπολογιστικά Συστήματα

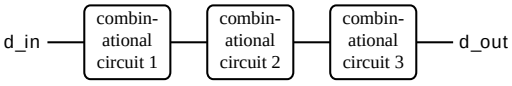
VHDL - Παραδείγματα

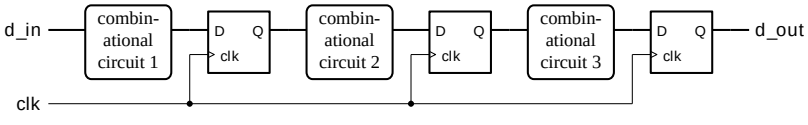


Μ. Ψαράκης

Μηχανισμός διοχέτευσης (pipeline)

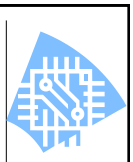
- Συνολική καθυστέρηση = Delay1 + Delay2 + Delay3
- Διάστημα μεταξύ δεδομένων > Συνολική καθυστέρηση





- Περίοδος ρολογιού = max(Delay1, Delay2, Delay3)
- Συνολική καθυστέρηση = 3 × περίοδος ρολογιού
- Διάστημα μεταξύ δεδομένων = 1 περίοδος ρολογιού

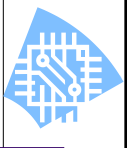
Παράδειγμα διοχέτευσης



- Υπολογισμός του μέσου όρου τριών ροών δεδομένων
- Νέα δεδομένα σε κάθε ακμή του ρολογιού

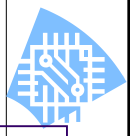
```
library ieee;
use ieee.std_logic_1164.all;

entity average_pipeline is
  port ( clk     : in std_logic;
         a, b, c : in integer;
         avg     : out integer);
end entity average_pipeline;
```


Παράδειγμα διοχέτευσης (συν.)



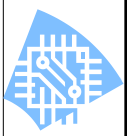
```
architecture rtl of average_pipeline is
  signal a_plus_b, sum, sum_div_3 : integer;
  signal saved_a_plus_b,
         saved_c, saved_sum : integer;

begin
  a_plus_b <= a + b;
  reg1 : process (clk) is
  begin
    if rising_edge(clk) then
      saved_a_plus_b <= a_plus_b;
      saved_c <= c;
    end if;
  end process reg1;
...
```

Παράδειγμα διοχέτευσης (συν.)



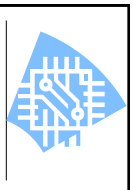
```
  sum <= saved_a_plus_b + saved_c;

  reg2 : process (clk) is
  begin
    if rising_edge(clk) then
      saved_sum <= sum;
    end if;
  end process reg2;

  sum_div_3 <= saved_sum/3;

  reg3 : process (clk) is
  begin
    if rising_edge(clk) then
      avg <= sum_div_3;
    end if;
  end process reg3;
end architecture average_pipeline;
```


Διαδρομές Δεδομένων και Έλεγχος



- Τα ψηφιακά συστήματα εκτελούν ακολουθίες λειτουργιών σε κωδικοποιημένα δεδομένα
- *Διαδρομή Δεδομένων (Datapath)*
  - Συνδυαστικά κυκλώματα για τις λειτουργίες
  - Καταχωρητές για την αποθήκευση των ενδιάμεσων αποτελεσμάτων
- *Τμήμα Ελέγχου (Control)*: ακολουθία ελέγχου
  - Παράγει *σήματα ελέγχου (control signals)*
    - Επιλέγει τις λειτουργίες που θα εκτελεστούν
    - Ενεργοποιεί τους καταχωρητές στις σωστές στιγμές
  - Χρησιμοποιεί *σήματα κατάστασης (status signals)* από τη διαδρομή δεδομένων

Παράδειγμα: Μιγαδικός Πολλαπλασιαστής



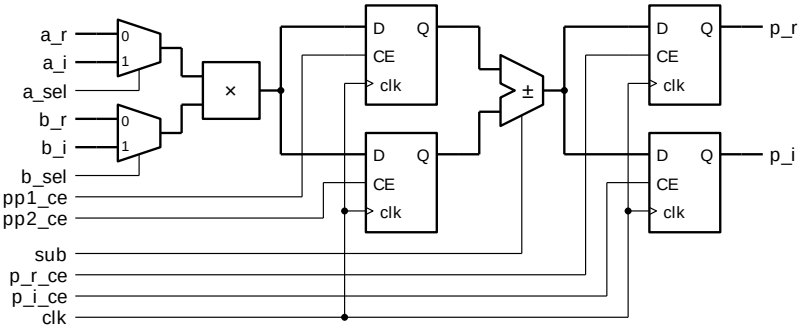
- Καρτεσιανή μορφή, σταθερή υποδιαστολή
  - τελεστέοι: 4 bit πριν, 12 μετά τη δυαδική υποδ.
  - αποτέλεσμα: 8 bit πριν, 24 μετά τη δυαδική υποδ.
- Με έμφαση στη μείωση της επιφάνειας (area)

$$a = a_r + ja_i \qquad b = b_r + jb_i$$

$$p = ab = p_r + jp_i = (a_r b_r - a_i b_i) + j(a_r b_i + a_i b_r)$$

- 4 πολλαπλασιασμοί, 1 πρόσθεση, 1 αφαίρεση
  - Ακολουθιακή εκτέλεση χρησιμοποιώντας 1 πολλαπλασιαστή, 1 αθροιστή/αφαιρέτη

Διαδρομή Δεδομένων

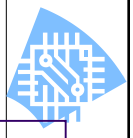


Μιγαδικός Πολλαπλασιαστής στην VHDL

```
library ieee; use ieee.std_logic_1164.all, ieee.fixed_pkg.all;
entity multiplier is
  port ( clk, reset : in std_logic;
         input_rdy : in std_logic;
         a_r, a_i, b_r, b_i : in sfixed(3 downto -12);
         p_r, p_i : out sfixed(7 downto -24) );
end entity multiplier;
```

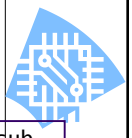
```
architecture rtl of multiplier is
  signal a_sel, b_sel, pp1_ce, pp2_ce,
         sub, p_r_ce, p_i_ce : std_logic; -- control signals
  signal a_operand, b_operand : sfixed(3 downto -12);
  signal pp, pp1, pp2, sum : sfixed(7 downto -24);
  ...
begin
```


Μιγαδικός Πολλαπλασιαστής στην VHDL



```
  a_operand <= a_r when a_sel = '0' else a_i;  -- mux
  b_operand <= b_r when b_sel = '0' else b_i;  -- mux
  pp <= a_operand * b_operand;  -- multiplier
  pp1_reg : process (clk) is  -- partial product register 1
  begin
    if rising_edge(clk) then
      if pp1_ce = '1' then
        pp1 <= pp;
      end if;
    end if;
  end process pp1_reg;
  pp2_reg : process (clk) is -- partial product register 2
  begin
    if rising_edge(clk) then
      if pp2_ce = '1' then
        pp2 <= pp;
      end if;
    end if;
  end process pp2_reg;
```

Μιγαδικός Πολλαπλασιαστής στην VHDL

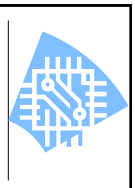


```
  sum <= pp1 + pp2 when sub = '0' else pp1 - pp2;  -- add/sub
  p_r_reg : process (clk) is  -- result real-part register
  begin
    if rising_edge(clk) then
      if p_r_ce = '1' then
        p_r <= sum;
      end if;
    end if;
  end process p_r_reg;
  p_i_reg : process (clk) is  -- result imag-part register
  begin
    if rising_edge(clk) then
      if p_i_ce = '1' then
        p_i <= sum;
      end if;
    end if;
  end process p_i_reg;
  ...  -- control circuit
end architecture rtl;
```


Ακολουθία Ελέγχου του Πολλαπλασιαστή

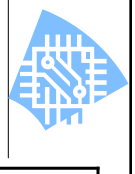
- Πρώτη υλοποίηση
  1. a_r * b_r → pp1_reg
  2. a_i * b_i → pp2_reg
  3. pp1 – pp2 → p_r_reg
  4. a_r * b_i → pp1_reg
  5. a_i * b_r → pp2_reg
  6. pp1 + pp2 → p_i_reg
- Αποφεύγει τη διένεξη των πόρων
- Διαρκεί 6 κύκλους ρολογιού

Ακολουθία Ελέγχου του Πολλαπλασιαστή



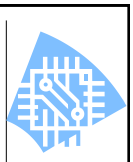
- Βελτιωμένη υλοποίηση
  1. a_r * b_r → pp1_reg
  2. a_i * b_i → pp2_reg
  3. pp1 – pp2 → p_r_reg
     a_r * b_i → pp1_reg
  4. a_i * b_r → pp2_reg
  5. pp1 + pp2 → p_i_reg
- Συγχωνεύει τα βήματα όπου δεν υπάρχει διένεξη των πόρων
- Διαρκεί 5 κύκλους ρολογιού

Σήματα Ελέγχου του Πολλαπλασιαστή



Βήμα	a_sel	b_sel	pp1_ce	pp2_ce	sub	p_r_ce	p_i_ce
1	0	0	1	0	–	0	0
2	1	1	0	1	–	0	0
3	0	1	1	0	1	1	0
4	1	0	0	1	–	0	0
5	–	–	0	0	0	0	1

Finite State Machine: Έλεγχος Πολλαπλασιαστή

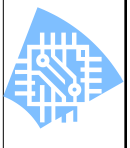


- Μία κατάσταση ανά βήμα
- Ξεχωριστή κατάσταση αδράνειας (idle);
  - Αναμένει έως ότου input_rdy = '1'
  - Έπειτα, προχωράει στα βήματα 1, 2, ...
  - Αλλά αυτό σπαταλάει έναν κύκλο!
- Χρησιμοποιεί το βήμα 1 ως κατάσταση αδράνειας
  - Επαναλαμβάνει το βήμα 1 εάν input_rdy≠'1'
  - Διαφορετικά προχωράει στο βήμα 2
- Συνάρτηση εξόδου
  - Moore ή Mealy;

Συνάρτηση μετάβασης

current_ state	input_ rdy	next_ state
step1	0	step1
step1	1	step2
step2	–	step3
step3	–	step4
step4	–	step5
step5	–	step1

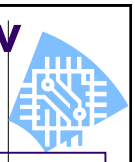
FSM στην VHDL



- Χρησιμοποιούμε έναν τύπο απαρίθμησης για τις τιμές των καταστάσεων
  - αφηρημένο, έτσι αποφεύγουμε να προδιαγράψουμε την κωδικοποίηση

```
type multiplier_state is (step1, step2, step3, step4, step5);
signal current_state, next_state : multiplier_state;
...
```

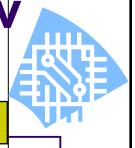
Έλεγχος Πολλαπλασιαστή στην VHDL



```
state_reg : process (clk, reset) is
begin
  if reset = '1' then
    current_state <= step1;
  elsif rising_edge(clk) then
    current_state <= next_state;
  end if;
end process state_reg;
```

```
next_state_logic : process
  (current_state, input_rdy) is
begin
  case current_state is
    when step1 =>
      if input_rdy = '0' then
        next_state <= step1;
      else
        next_state <= step2;
      end if;
    when step2 =>
      next_state <= step3;
    when step3 =>
      next_state <= step4;
    when step4 =>
      next_state <= step5;
    when step5 =>
      next_state <= step1;
  end case;
end process next_state_logic;
```

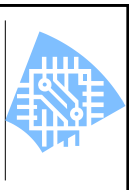

Έλεγχος Πολλαπλασιαστή στην VHDL



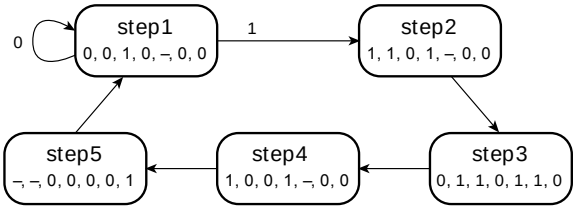
FSM τύπου Moore

```
output_logic : process (current_state) is
begin
  case current_state is
    when step1 =>
      a_sel <= '0'; b_sel <= '0'; pp1_ce <= '1'; pp2_ce <= '0';
      sub <= '0'; p_r_ce <= '0'; p_i_ce <= '0';
    when step2 =>
      a_sel <= '1'; b_sel <= '1'; pp1_ce <= '0'; pp2_ce <= '1';
      sub <= '0'; p_r_ce <= '0'; p_i_ce <= '0';
    when step3 =>
      a_sel <= '0'; b_sel <= '1'; pp1_ce <= '1'; pp2_ce <= '0';
      sub <= '1'; p_r_ce <= '1'; p_i_ce <= '0';
    when step4 =>
      a_sel <= '1'; b_sel <= '0'; pp1_ce <= '0'; pp2_ce <= '1';
      sub <= '0'; p_r_ce <= '0'; p_i_ce <= '0';
    when step5 =>
      a_sel <= '0'; b_sel <= '0'; pp1_ce <= '0'; pp2_ce <= '0';
      sub <= '0'; p_r_ce <= '0'; p_i_ce <= '1';
  end case;
end process output_logic;
```

Διάγραμμα Ελέγχου του Πολλαπλασιαστή



- Είσοδος: input_rdy
- Έξοδοι:
  - a_sel, b_sel, pp1_ce, pp2_ce, sub, p_r_ce, p_i_ce

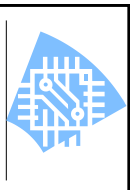


Μνήμες στην VHDL

- Η αποθήκευση της RAM αναπαρίσταται από ένα σήμα πίνακα

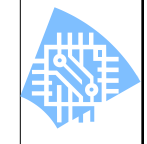
```
type RAM_4Kx16 is array (0 to 4095) of std_logic_vector(15 downto 0);
signal data_RAM : RAM_4Kx16;
...
data_RAM_flow_through : process (clk) is
begin
  if rising_edge(clk) then
    if en = '1' then
      if wr = '1' then
        data_RAM(to_integer(a)) <= d_in;  d_out <= d_in;
      else
        d_out <= RAM(to_integer(a));
      end if;
    end if;
  end if;
end process data_RAM_flow_through;
```

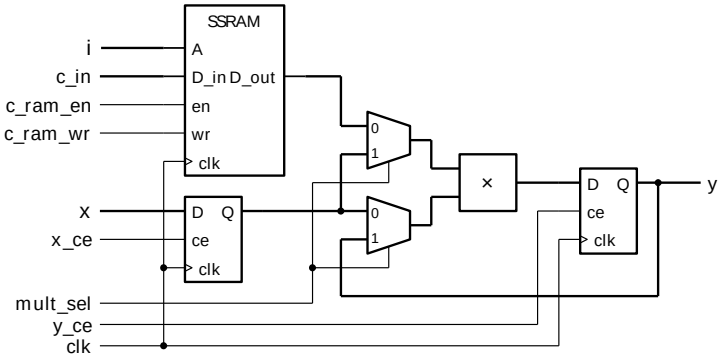
Παράδειγμα: Πολ/στής Συντελεστών



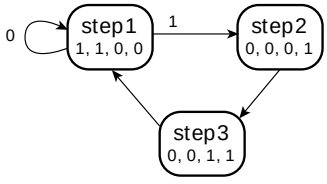
- Υπολογίστε τη συνάρτηση $y = c_i \times x^2$
  - Ο συντελεστής είναι αποθηκευμένος σε μια SSRAM
    - ακέραιος δείκτης των 12 bit για το i
  - x, y, c_i προσημασμένες τιμές σταθερής υποδιαστολής των 20 bit
    - 8 bit πριν και 12 bit μετά τη δυαδική υποδιαστολή
  - Είσοδος *start*: δηλώνει την άφιξη νέων τιμών στις εισόδους x και i
  - Χρησιμοποιείστε έναν απλό πολλαπλασιαστή
    - Πολλαπλασιάστε $c_i \times x \times x$

Διαδρομή Δεδομένων Πολ/στή

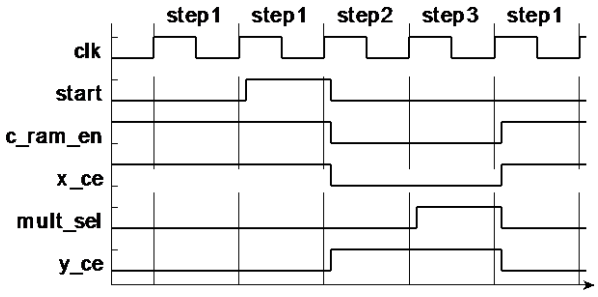




Χρονισμός και Έλεγχος Πολ/στή



Σήματα ελέγχου:
c_ram_en, x_ce, mult_sel, y_ce

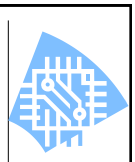


Παράδειγμα: Πολ/στής Συντελεστών

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

entity scaled_square is
  port ( clk, reset : in std_logic;
         start : in std_logic;
         i : in unsigned(11 downto 0);
         c_in, x : in sfixed(7 downto -12);
         y : out sfixed(7 downto -12) );
end entity scaled_square;
```

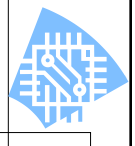
```
architecture rtl of scaled_square is

  signal c_ram_en, c_ram_wr, x_ce, mult_sel, y_ce : std_logic;
  signal c_out, x_out : sfixed(7 downto -12);
  signal y_out : sfixed(7 downto -12);

  type c_array is array (0 to 4095) of sfixed(7 downto -12);
  signal c_RAM : c_array;

  type state is (step1, step2, step3);
  signal current_state, next_state : state;
```

Παράδειγμα: Πολ/στής Συντελεστών



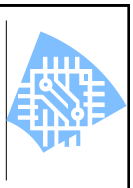
```
begin
  c_ram_wr <= '0';
  c_RAM_flow_through : process (clk) is
  begin
    if rising_edge(clk) then
      if c_ram_en = '1' then
        if c_ram_wr = '1' then
          c_RAM(to_integer(i)) <= c_in;
          c_out <= c_in;
        else
          c_out <= c_RAM(to_integer(i));
        end if;
      end if;
    end if;
  end process c_RAM_flow_through;
```


Παράδειγμα: Πολ/στής Συντελεστών



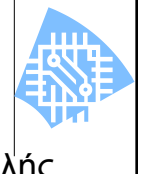
```
  y_reg : process (clk) is
    variable operand1, operand2 : sfixed(7 downto -12);
  begin
    if rising_edge(clk) then
      if y_ce = '1' then
        if mult_sel = '0' then
          operand1 := c_out;  operand2 := x_out;
        else
          operand1 := x_out;  operand2 := y_out;
        end if;
        y_out <= operand1 * operand2;
      end if;
    end if;
  end process y_reg;

  y <= y_out;

  state_reg : process ...
  next_state_logic : process ...
  output_logic : process ...
end architecture rtl;
```

Μνήμες Πολλαπλών Θυρών



- Πολλαπλές συνδέσεις διεύθυνσης, δεδομένων και ελέγχου στις θέσεις αποθήκευσης
- Επιτρέπει ταυτόχρονες προσπελάσεις
  - Αποφεύγει την πολυπλεξία και την ακολουθία ελέγχου
- Σενάριο
  - Ένα σύστημα παράγει, ένα σύστημα καταναλώνει δεδομένα
- Τι θα συμβεί ένα συμβούν ταυτόχρονα δύο εγγραφές σε μία θέση;
  - Το αποτέλεσμα μπορεί να είναι απρόβλεπτο
  - Κάποιες μνήμες πολλαπλών θυρών περιλαμβάνουν κύκλωμα διαιτησίας (arbiter)

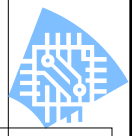
Παράδειγμα: dual-port SSRAM



- Αναπτύξτε το μοντέλο VHDL μίας dual-port (διπλής θύρας) SSRAM μεγέθους 4K x 16 bit
  - Θύρα a1: εγγραφή και ανάγνωση
  - Θύρα a2: μόνο ανάγνωση

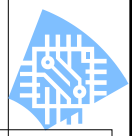
```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity dual_port_SSRAM is
  port ( clk      : in  std_logic;
         en1, wr1 : in  std_logic;
         a1       : in  unsigned(11 downto 0);
         d_in1    : in  std_logic_vector(15 downto 0);
         d_out1   : out std_logic_vector(15 downto 0);
         en2      : in  std_logic;
         a2       : in  unsigned(11 downto 0);
         d_out2   : out std_logic_vector(15 downto 0) );
end entity dual_port_SSRAM;
```

Παράδειγμα: dual-port SSRAM



```
architecture synth of dual_port_SSRAM is
  type RAM_4Kx16 is array (0 to 4095)
                    of std_logic_vector(15 downto 0);
  signal data_RAM : RAM_4Kx16;

begin

  read_write_port : process (clk) is
  begin
    if rising_edge(clk) then
      if en1 = '1' then
        if wr1 = '1' then
          data_RAM(to_integer(a1)) <= d_in1;  d_out1 <= d_in1;
        else
          d_out1 <= data_RAM(to_integer(a1));
        end if;
      end if;
    end if;
  end process read_write_port;
...
```


Παράδειγμα: dual-port SSRAM

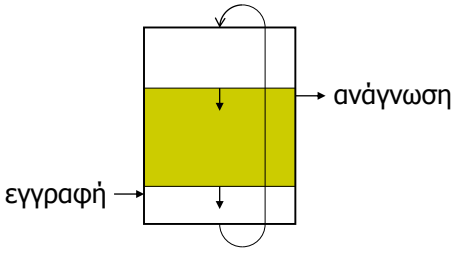


```
...

  read_only_port : process (clk) is
  begin
    if rising_edge(clk) then
      if en2 = '1' then
        d_out2 <= data_RAM(to_integer(a2));
      end if;
    end if;
  end process read_only_port;

end architecture synth;
```

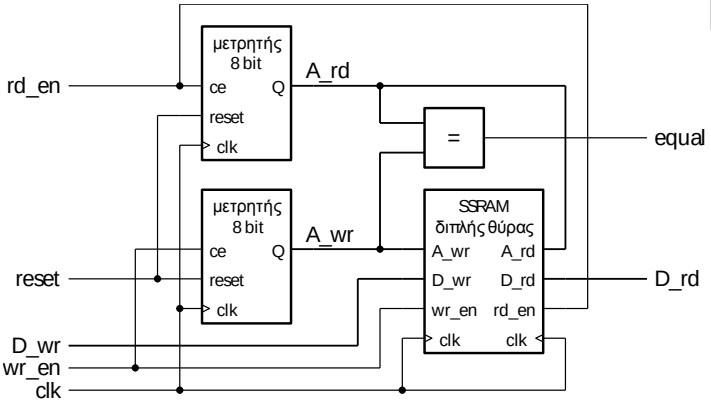
Μνήμες FIFO

- First-In/First-Out (FIFO) προσωρινή μνήμη (buffer)
  - Συνδέει το σύστημα που παράγει με αυτό που καταναλώνει
  - Αποσυνδέει τους ρυθμούς παραγωγής και κατανάλωσης

- Υλοποίηση με RAM διπλής θύρας
  - Κυκλική προσωρινή μνήμη
  - Γεμάτη (full):
    write address = read address
  - Άδεια (empty):
    write address = read address

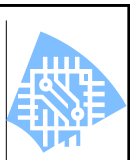


Παράδειγμα: FIFO - Datapath

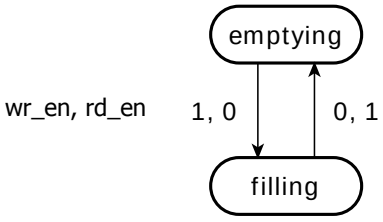


- Equal = γεμάτη ή άδεια
  - Χρειάζεται να διακρίνουμε αυτές τις καταστάσεις — Πώς;

Παράδειγμα: FIFO - Control



- FSM ελέγχου
  - → filling (γεμίζει): εγγραφή χωρίς ταυτόχρονη ανάγνωση
  - → emptying (αδειάζει): ανάγνωση χωρίς ταυτόχρονη εγγραφή
  - αμετάβλητη όταν συμβεί ταυτόχρονη εγγραφή και ανάγνωση



full = filling and equal

empty = emptying and equal