

Σχεδίαση Ψηφιακών Κυκλωμάτων

Η γλώσσα περιγραφής υλικού VHDL - Μέρος II

Νικόλαος Καββαδίας
nkavn@uop.gr

08 Δεκεμβρίου 2010

Σκιαγράφηση της διάλεξης

- Σύνταξη κώδικα για λογική σύνθεση
- Σχεδίαση μνημών ROM και RAM
- Δομές ελέγχου/επαλήθευσης λειτουργίας των κυκλωμάτων
- Μηχανές πεπερασμένων καταστάσεων
- Μη προγραμματιζόμενοι επεξεργαστές

Απαριθμητοί τύποι δεδομένων (enumerated data types)

- Ο χρήστης ορίζει τη λίστα των επιτρεπόμενων τιμών που μπορούν να ανατεθούν σε ένα αντικείμενο που δηλώνεται με τον συγκεκριμένο τύπο
- Απαριθμητός τύπος δεδομένων χαρακτηριστικός για μηχανές πεπερασμένων καταστάσεων (FSM)

```
TYPE fsm_state is (idle, forward, backward, stop);  
...  
signal current_state := IDLE;
```

- Απαριθμητός τύπος δεδομένων που καταγράφει τα επιτρεπόμενα χρώματα στο πρότυπο TELETEXT

```
TYPE rgb3 is (black, blue, green, cyan, red, magenta, yellow, white);
```

- Τύπος δεδομένων για την υλοποίηση λογικής 4 επιπέδων κατά Verilog

```
TYPE verilog_mv14 is ('0', '1', 'X', 'Z');
```

Σύνθετοι τύποι: Πίνακες

- Πίνακας: συλλογή από αντικείμενα του ίδιου τύπου
- Δήλωση για τον ορισμό ενός νέου τύπου πίνακα και δήλωση SIGNAL αυτού του τύπου:

```
TYPE <array name> IS ARRAY specification OF <data type>;  
SIGNAL <signal name>: <array type> [:= <initial value>;
```

- Παραδείγματα

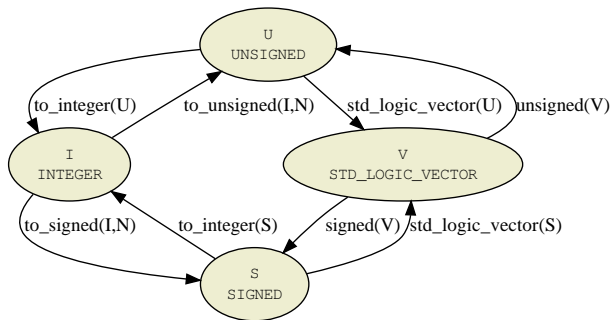
```
TYPE image is ARRAY (0 to 31) of byte; -- 1Dx1D  
TYPE matrix2D is ARRAY (0 to 3, 1 downto 0) OF STD_LOGIC;  
...  
SIGNAL x: image; -- an 1Dx1D signal  
SIGNAL y: matrix2D;  
-- Initialization  
... := "0001"; -- 1D  
... := ('0','0','0','1'); -- 1D  
... := (('0','1','1','1'),('1','1','1','0')); -- 1Dx1D or 2D  
-- Assignments  
x(0) <= y(1)(2); -- 1Dx1D  
x(0) <= v(1,2); -- 2D  
x <= y(0); -- entire row  
x(3 downto 1) <= y(1)(4 downto 2);  
x(3 downto 0) <= (3 => '1', 2 => '0', others => '0');
```

Σύνοψη των συναρτήσεων μετατροπής τύπου του πακέτου numeric_std

- Δήλωση για τη χρήση του πακέτου

```
LIBRARY ieee;  
USE ieee.numeric_std.all;
```

- Γραφική αναπαράσταση των επιτρεπόμενων μετατροπών



Συναρτήσεις μετατροπής του πακέτου std_logic_arith

- Στο std_logic_arith μπορούμε να βρούμε τις συναρτήσεις μετατροπής
conv_integer, conv_unsigned, conv_signed, conv_std_logic_vector

Κλίση συνάρτησης	Περιγραφή
conv_integer(param)	Μετατρέπει μια παράμετρο <i>param</i> τύπου INTEGER, UNSIGNED, SIGNED ή STD_ULOGIC σε μια τιμή τύπου INTEGER
conv_unsigned(param,b)	Μετατρέπει μια παράμετρο <i>param</i> τύπου INTEGER, UNSIGNED, SIGNED ή STD_ULOGIC σε μια τιμή τύπου UNSIGNED με μέγεθος <i>b</i> bit
conv_signed(param,b)	Μετατρέπει μια παράμετρο <i>param</i> τύπου INTEGER, UNSIGNED, SIGNED ή STD_ULOGIC σε μια τιμή τύπου SIGNED με μέγεθος <i>b</i> bit
conv_std_logic_vector(param,b)	Μετατρέπει μια παράμετρο <i>param</i> τύπου INTEGER, UNSIGNED, SIGNED ή STD_ULOGIC σε μια τιμή τύπου STD_LOGIC_VECTOR με μέγεθος <i>b</i> bit

Το πακέτο std_logic_unsigned ορίζει υπερφορτωμένες εκδοχές των πρώτων τριών συναρτήσεων μετατροπής για δεδομένα τύπου STD_LOGIC_VECTOR.

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) – υπάρχουν εξαιρέσεις (FSMs)
- 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)

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**

i 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**)

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

```
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;
```

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
- 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)
- Τρόπος εγγραφής READ FIRST: Τα περιεχόμενα της διευθυνσιοδοτούμενης θέσης μνήμης εμφανίζονται στην έξοδο. Τα δεδομένα εισόδου γράφονται στην ίδια θέση (ανάγνωση πριν την εγγραφή)

RAM με ασύγχρονη ανάγνωση

```
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 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;
```

ASSERT

- Η ASSERT είναι μία μη συνθέσιμη εντολή που χρησιμοποιείται για την επιστροφή μηνυμάτων στο τερματικό κατά την προσομοίωση
 - 1 Τμήμα συνθήκης (condition)
 - 2 Τμήμα αναφοράς μηνύματος που προσδιορίζεται από τη λέξη κλειδί REPORT
 - 3 Τμήμα σοβαρότητας στο οποίο γίνεται δήλωση της επίδρασης που έχει η μη ικανοποίηση της συνθήκης στη συνέχεια της προσομοίωσης. Σημειώνεται με τη λέξη κλειδί SEVERITY

```
ASSERT <condition>  
  [REPORT "<message>"]  
  [SEVERITY <severity level>];  
END <package name>;
```

- Τα επίπεδα σοβαρότητας είναι: σημείωση (NOTE), προειδοποίηση (WARNING), σφάλμα (ERROR), ή αποτυχία (FAILURE)

Αρχεία στη VHDL

- Ο τύπος APXEIOY (FILE) προσφέρει ένα βολικό τρόπο για την επικοινωνία μιας περιγραφής VHDL με το περιβάλλον του μηχανήματος-ξενιστή (ο υπολογιστής στον οποίο γίνεται η ανάπτυξη και ο έλεγχος λειτουργίας της περιγραφής)

```
type LINE is access STRING; -- A LINE is a pointer to a STRING value.
type TEXT is file of STRING;
```

- Διαδικασίες για το χειρισμό αρχείων κειμένου (TEXTIO)

```
procedure FILE_OPEN (file F: TEXT; External_Name; in STRING;
                    Open_Kind: in FILE_OPEN_KIND := READ_MODE);
procedure FILE_OPEN (Status: out FILE_OPEN_STATUS; file F: TEXT;
                    External_Name: in STRING;
                    Open_Kind: in FILE_OPEN_KIND := READ_MODE);
procedure FILE_CLOSE (file F: TEXT);
function ENDFILE (file F: TEXT) return BOOLEAN;
procedure READLINE (file F: TEXT; L: inout LINE);
procedure WRITELINE (file F: TEXT; L: inout LINE);
procedure READ (file F: TEXT; VALUE: out STRING);
procedure WRITE (file F: TEXT; VALUE: in STRING);
procedure HREAD(L:inout LINE; VALUE:out STD_LOGIC_VECTOR; GOOD: out BOOLEAN);
procedure HWRITE(L:inout LINE; VALUE:in STD_LOGIC_VECTOR;
                JUSTIFIED:in SIDE := RIGHT; FIELD:in WIDTH := 0);
```

Testbench

- Το testbench αποτελεί ένα εικονικό κύκλωμα το οποίο εφαρμόζει εισόδους προς (διέγερση) και λαμβάνει εξόδους (απόκριση) από το πραγματικό κύκλωμα
- Η entity ενός testbench δεν περιλαμβάνει καμία δήλωση θύρας, μπορεί όμως να περιλαμβάνει generic
- Στο testbench, δηλώνεται το COMPONENT του συνολικού κυκλώματος

```
ENTITY testbench IS
-- no PORT statement necessary
END testbench;

ARCHITECTURE example IS testbench
    COMPONENT entity_under_test
        PORT(...);
    END COMPONENT;
BEGIN
    Generate_waveforms_for_test;
    Instantiate_component;
    Monitoring_statements;
END example;
```


Διέγερση σημάτων εισόδου από process

```
...
CLK_GEN_PROC: process(clk)
begin
  if (clk = 'U') then
    clk <= '1';
  else
    clk <= not clk after CLK_PERIOD/2;
  end if;
end process CLK_GEN_PROC;

DATA_INPUT: process
  variable ix : integer range 0 to 7;
begin
  in1 <= X"DE"; in2 <= X"AD"; in3 <= X"BE"; in4 <= X"EF";
  sel <= "000"; reset <= '1';
  wait for CLK_PERIOD;
  --
  reset <= '0';
  for i in 0 to 7 loop
    sel <= std_logic_vector(to_unsigned(i,3));
    wait for CLK_PERIOD;
  end loop;
end process DATA_INPUT;
...
```

Παράδειγμα: Εγγραφή αποτελεσμάτων σε αρχείο εξόδου

```
component add ...
signal a, b, sum : std_logic_vector(Dw-1 downto 0);
file output_log : text open write_mode is "add.log";
begin
  UUT : add
    generic map (Dw => Dw)
    port map (a => a, b => b, sum => sum);

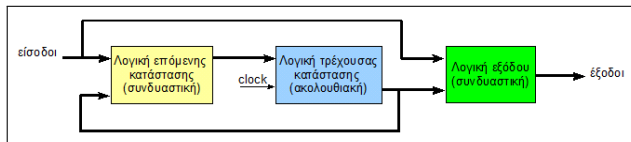
  process
  begin
    a <= X"FF"; b <= X"10"; wait for 10 ns;
    a <= X"10"; b <= X"89"; wait for 10 ns;
  end process;

  output_log_proc: process
    variable out_line : line;
  begin
    write(out_line, NOW, left, 8);
    write(out_line, string'(" a:"), right, 4);
    hwrite(out_line, a, right, 4);
    write(out_line, string'(" b:"), right, 4);
    hwrite(out_line, b, right, 4);
    write(out_line, string'(" sum:"), right, 4);
    hwrite(out_line, sum, right, 4);
    writeline(output_log, out_line);
    wait for 10 ns;
  end process output_log_proc;
...

```

Δομή ενός FSM

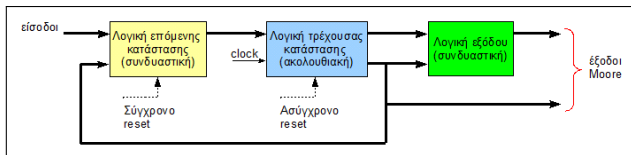
Τυπική οργάνωση ενός FSM



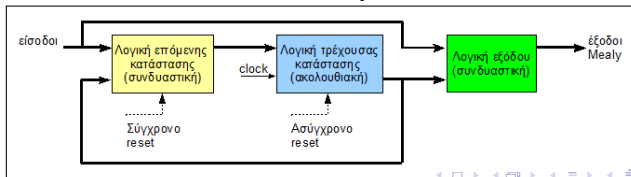
- 1 Λογική τρέχουσας κατάστασης:** Υλοποιείται από καταχωρητή για την αποθήκευση της τρέχουσας κατάστασης του FSM. Η τιμή του αντιπροσωπεύει το συγκεκριμένο στάδιο στο οποίο βρίσκεται η λειτουργία του FSM
- 2 Λογική επόμενης κατάστασης:** Συνδυαστική λογική η οποία παράγει την επόμενη κατάσταση της ακολουθίας. Η επόμενη κατάσταση αποτελεί συνάρτηση των εισόδων του FSM και της τρέχουσας κατάστασης
- 3 Λογική εξόδου:** Συνδυαστική λογική που χρησιμοποιείται για την παραγωγή των σημάτων εξόδου του κυκλώματος. Οι έξοδοι αποτελούν συνάρτηση της εξόδου του καταχωρητή (τρέχουσας) κατάστασης και ΠΙΘΑΝΩΣ των εισόδων του FSM

Κατηγορίες FSM: τύπου Moore και τύπου Mealy

- Στα FSM τύπου Moore οι έξοδοι είναι συνάρτηση μόνο της τρέχουσας κατάστασης
- Οργάνωση ενός FSM τύπου Moore



- Στα FSM τύπου Mealy οι έξοδοι είναι συνάρτηση των εισόδων και της τρέχουσας κατάστασης
- Οργάνωση ενός FSM τύπου Mealy



Κωδικοποίηση της κατάστασης στα FSM

- sequential: σε κάθε κατάσταση ανατίθενται δυαδικοί αριθμοί κατά αύξουσα σειρά
- one-hot: σε κάθε κατάσταση αντιστοιχίζεται ξεχωριστό flip-flop. Σε κάθε κατάσταση ένα μόνο flip-flop έχει την τιμή '1'
- Κωδικοποίηση καθοριζόμενη από το χρήστη

```
constant S1: std_logic_vector(3 downto 0) := "0110";  
constant S2: std_logic_vector(3 downto 0) := "0111";  
constant S3: std_logic_vector(3 downto 0) := "0000";
```

- Κωδικοποίηση καθοριζόμενη από το εργαλείο υλοποίησης (λογικής σύνθεσης)

```
type STATES is (S1, S2, S3, S4);  
signal state : STATES;
```

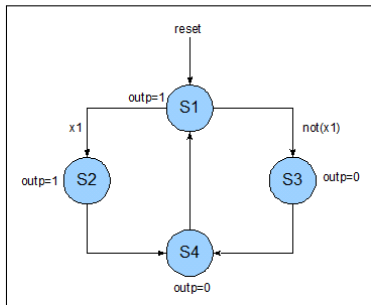
- Άλλες κωδικοποιήσεις: Gray, Johnson, one-cold

Παρατηρήσεις

- Για την αρχικοποίηση του FSM σε μία γνωστή αρχική κατάσταση επιβάλλεται η χρήση ασύγχρονης επανατοποθέτησης (asynchronous reset)
- Γενικά υφίστανται αρκετές τεχνικές για τη σύνταξη της περιγραφής ενός FSM
 - 1 FSM με μία διεργασία (process): Η λογική επόμενης κατάστασης, τρέχουσας κατάστασης και εξόδου σε μία PROCESS
 - 2 FSM με δύο διεργασίες: Η λογική επόμενης κατάστασης και τρέχουσας κατάστασης σε μία PROCESS και η λογική εξόδου σε μία δεύτερη
 - 3 FSM με τρεις διεργασίες: Η λογική επόμενης κατάστασης, τρέχουσας κατάστασης και εξόδου σε ξεχωριστές PROCESS
 - 4 FSM με δύο διεργασίες με τη λογική τρέχουσας κατάστασης σε μία PROCESS και τη λογική επόμενης κατάστασης και εξόδου σε μία δεύτερη PROCESS
 - 5 FSM με αποθηκευμένα σήματα εξόδου

Διάγραμμα μεταγωγής καταστάσεων για ένα απλό FSM 4 καταστάσεων

- Το FSM του παραδείγματος καθορίζεται από:
 - Τέσσερις καταστάσεις: S1, S2, S3, S4
 - Μία είσοδο: x1
 - Μία έξοδο: outp
 - Πέντε περιπτώσεις μετάβασης από κατάσταση σε κατάσταση σύμφωνα με το παρακάτω διάγραμμα μεταγωγής καταστάσεων



Παράδειγμα FSM με μία διεργασία (αποθηκευμένη έξοδος)

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

entity fsm_1 is
port (
  clk, reset, x1 : IN std_logic;
  outp : OUT std_logic
);
end fsm_1;

architecture beh1 of fsm_1 is
  type state_type is (s1,s2,s3,s4);
  signal state: state_type;
begin
  process (clk, reset)
  begin
    if (reset = '1') then
      state <= s1;
      outp <= '1';
    elsif (clk='1' and clk'event) then
      case state is
```

```
when s1 =>
  if (x1 = '1') then
    state <= s2;
    outp <= '1';
  else
    state <= s3;
    outp <= '0';
  end if;
when s2 =>
  state <= s4;
  outp <= '0';
when s3 =>
  state <= s4;
  outp <= '0';
when s4 =>
  state <= s1;
  outp <= '1';
end case;
end if;
end process;
end beh1;
```


Παράδειγμα FSM με τρεις διεργασίες

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

entity fsm_3 is
  port (
    clk, reset, x1 : IN std_logic;
    outp : OUT std_logic
  );
end fsm_3;

architecture beh1 of fsm_3 is
  type state_type is (s1,s2,s3,s4);
  signal current_state, next_state:
    state_type;
begin
  p1: process (clk, reset)
  begin
    if (reset = '1') then
      state <= s1;
    elsif (clk='1' and clk'EVENT) then
      current_state <= next_state;
    end if;
  end process process1;
```

```
  p2 : process (current_state, x1)
  begin
    case current_state is
      when s1 =>
        if (x1 = '1') then
          next_state <= s2;
        else
          next_state <= s3;
        end if;
      when s2 =>
        next_state <= s4;
      when s3 =>
        next_state <= s4;
      when s4 =>
        next_state <= s1;
    end case;
  end process process2;

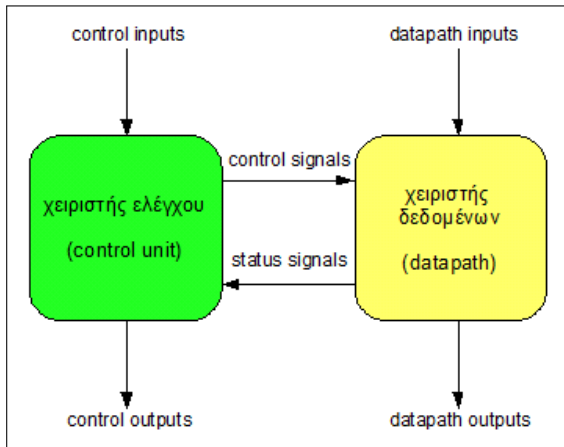
  p3 : process (current_state)
  begin
    case current_state is
      when s1 => outp <= '1';
      when s2 => outp <= '1';
      when s3 => outp <= '0';
      when s4 => outp <= '0';
    end case;
  end process process3;
end beh1;
```

Μη προγραμματιζόμενοι επεξεργαστές

- Μη προγραμματιζόμενοι επεξεργαστές είναι εκείνα τα κυκλώματα τα οποία έχουν σχεδιαστεί έτσι ώστε να μπορούν να επιλύσουν ένα μόνο πρόβλημα
- Ένας μη προγραμματιζόμενος επεξεργαστής αποτελείται από το χειριστή ελέγχου (controller ή control unit) και το χειριστή δεδομένων (datapath)
- Ο χειριστής ελέγχου παράγει σήματα ελέγχου για την δρομολόγηση των μηχανισμών που λαμβάνουν χώρα στο χειριστή δεδομένων
- Ο χειριστής δεδομένων επιστρέφει στο χειριστή ελέγχου σήματα κατάστασης (status signals) τα οποία κατευθύνουν τη μετάβαση ανάμεσα στις εσωτερικές καταστάσεις του χειριστή ελέγχου
- Ο χειριστής ελέγχου υλοποιείται συχνά ως FSM

Η οργάνωση ενός μη-προγραμματιζόμενου επεξεργαστή

- Γενικό σχηματικό διάγραμμα ενός μη προγραμματιζόμενου επεξεργαστή



Το πρόβλημα του μέγιστου κοινού διαιρέτη δύο αριθμών

- Δεχόμαστε ότι: $\text{gcd}(n, 0) = \text{gcd}(0, n) = \text{gcd}(0, 0) = 0$
- Το ζητούμενο είναι η εύρεση αριθμού m ο οποίος να είναι ο μεγαλύτερος θετικός ακέραιος ο οποίος διαιρεί και τους δύο αριθμούς
- Στα αρχαία Ελληνικά μαθηματικά αντιπροσωπεύει το πρόβλημα εύρεσης κοινής αναφοράς για τη μέτρηση δύο ευθύγραμμων τμημάτων
- Το πρόβλημα του GCD επιλύεται με τον αλγόριθμο του Ευκλείδη

```
unsigned int gcd(unsigned int a, unsigned int b) {  
    assert(a > 0 && b > 0);  
    if (a == b) return a;  
    if (a > b) return gcd(a-b, b);  
    if (b > a) return gcd(a, b-a);  
}
```

- Στον αλγόριθμο του Ευκλείδη, η αναδρομή μπορεί να αποφευχθεί

Ο αλγόριθμος του μέγιστου κοινού διαιρέτη δύο αριθμών (GCD)

- Δεχόμαστε ότι: $\text{gcd}(n, 0) = \text{gcd}(0, n) = \text{gcd}(0, 0) = 0$
- Υλοποίηση σε ANSI C
- Αποφυγή αναδρομής και χρήσης του υπολογισμού ακέραιου υπολοίπου

```
int gcd(int a, int b)
{
    int result;
    int x, y;

    x = a;
    y = b;

    if (x!=0 && y!=0)
    {
        while (x != y)
        {
            if (x >= y)
                x = x - y;
            else
                y = y - x;
        }
    }
}
```

```
    result = x;
}
else
{
    result = 0;
}
return (result);
}

int main()
{
    int result = gcd(196, 42);
    return (result);
}
```

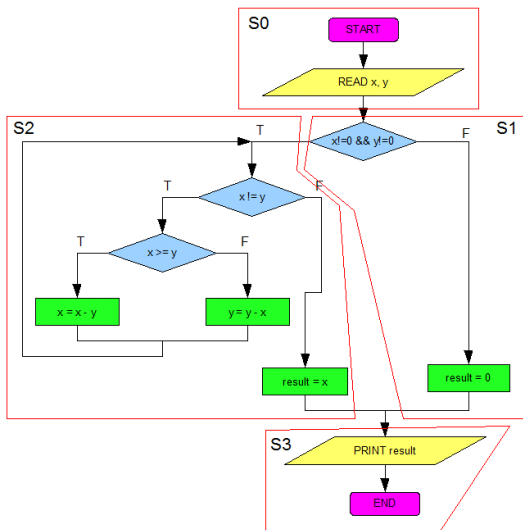
Αριθμητικό παράδειγμα υπολογισμού του GCD

- Ο μικρότερος αριθμός από δύο αριθμούς a, b αφαιρείται από τον μεγαλύτερο σε διαδοχικά βήματα
- Όταν οι δύο αριθμοί γίνουν ίσοι, τότε ισούνται με τον Μέγιστο Κοινό Διαιρέτη τους
- Σε περίπτωση που η διαδικασία φτάσει μέχρι το σημείο που $a = 1$ ή $b = 1$ τότε οι δύο αριθμοί δεν έχουν μη τετραμμένο GCD, δηλαδή μεγαλύτερο του 1
- Παράδειγμα ($a = 196, b = 42$)

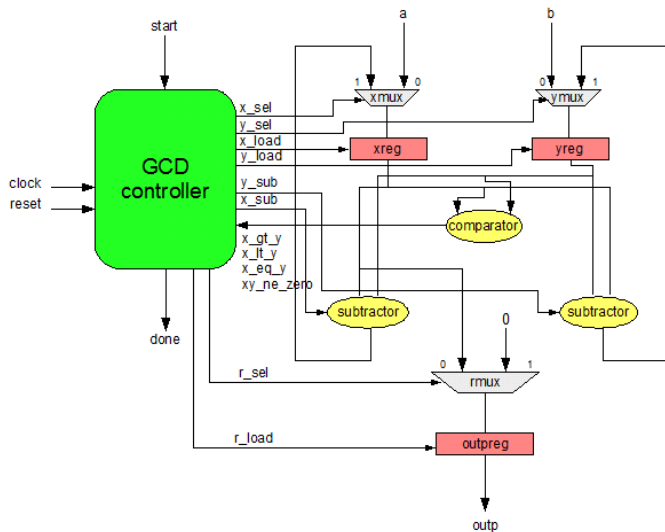
Βήμα	A	B
1	196	42
2	154	42
3	112	42
4	70	42
5	28	42
6	28	14
7	14	14

- Το αποτέλεσμα είναι: $gcd(196, 42) = 14$

Αλγοριθμικό διάγραμμα ροής για τον αλγόριθμο GCD



Το συνολικό κύκλωμα του επεξεργαστή GCD



Περιγραφή της υλοποίησης FSMD του επεξεργαστή GCD (1)

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

entity gcd is
  generic (
    WIDTH : integer
  );
  port (
    clock : in std_logic;
    reset : in std_logic;
    start : in std_logic;
    a      : in std_logic_vector(WIDTH-1 downto 0);
    b      : in std_logic_vector(WIDTH-1 downto 0);
    outp   : out std_logic_vector(WIDTH-1 downto 0);
    done   : out std_logic
  );
end gcd;

architecture fsmd of gcd is
  type state_type is (s0,s1,s2,s3);
  signal state: state_type;
  signal x, y, res : std_logic_vector(WIDTH-1 downto 0);
begin
```

Περιγραφή της υλοποίησης FSMD του επεξεργαστή GCD (2)

```
process (clock, reset)
begin
done <= '0';
--
if (reset = '1') then
state <= s0;
x <= (others => '0');
y <= (others => '0');
res <= (others => '0');
elsif (clock='1' and clock'EVENT) then
case state is
when s0 =>
if (start = '1') then
x <= a;
y <= b;
state <= s1;
else
state <= s0;
end if;
when s1 =>
if (x /= 0 and y /= 0) then
state <= s2;
else
res <= (others => '0');
state <= s3;
end if;
```

Περιγραφή της υλοποίησης FSMD του επεξεργαστή GCD (3)

```
when s2 =>
  if (x > y) then
    x <= x - y;
    state <= s2;
  elsif (x < y) then
    y <= y - x;
    state <= s2;
  else
    res <= x;
    state <= s3;
  end if;
when s3 =>
  done <= '1';
  state <= s0;
end if;
end process;

outp <= res;

end fsmd;
```

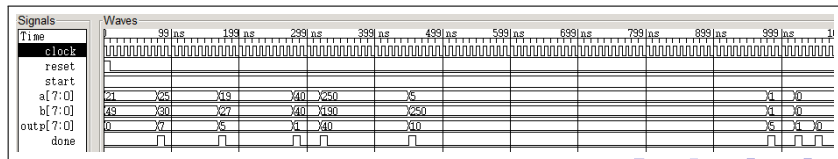
Προσομοίωση του επεξεργαστή GCD

- Δηλώσεις FILE για λήψη εισόδων από αρχείο και εκτύπωση διαγνωστικής εξόδου σε αρχείο

```
file TestDataFile: text open read_mode is "gcd_test_data.txt";  
file ResultsFile: text open write_mode is "gcd_alg_test_results.txt";
```

- Περιεχόμενα του "gcd_test_data.txt" (A, B, result)

```
21 49 7  
25 30 5  
19 27 1  
40 40 40  
250 190 10  
5 250 5  
1 1 1  
0 0 0
```



Έλεγχος ορθής λειτουργίας με testbench (1)

- Τα χαρακτηριστικά του testbench για την επαλήθευση της ορθής συμπεριφοράς του επεξεργαστή GCD
- Λήψη εισόδων από αρχείο με χρήση μεταβλητών
- Εξομοίωση της συμπεριφοράς του επεξεργαστή σε αλγοριθμικό επίπεδο
- Προσθήκη απαριθμητή επιδόσεων (performance counter) για την λήψη του αναλυτικού προφίλ εκτέλεσης του επεξεργαστή GCD για διαφορετικές εισόδους

```
...  
-- Profiling signals  
signal ncycles : integer;  
...  
PROFILING: process(clock, reset, done)  
begin  
  if (reset = '1' or done = '1') then  
    ncycles <= 0;  
  elsif (clock = '1' and clock'EVENT) then  
    ncycles <= ncycles + 1;  
  end if;  
end process PROFILING;
```

Έλεγχος ορθής λειτουργίας με testbench (2)

Αλγοριθμική υλοποίηση και έλεγχος ορθής λειτουργίας

```
GCD_EMUL: process
variable A_v,B_v,Y_v,Y_Ref,temp: integer range 0 to 255;
variable ncycles_v: integer;
variable TestData, BufLine: line;
variable Passed: std_logic := '1';
begin
  while not endfile(TestDataFile) loop
    ---- Read test data from file
    readline(TestDataFile, TestData);
    read(TestData, A_v);
    read(TestData, B_v);
    read(TestData, temp); -- reading the 3rd value (unused here)
    -- Assign inputs
    a <= conv_std_logic_vector(A_v, WIDTH);
    b <= conv_std_logic_vector(B_v, WIDTH);
    ---- Model GCD algorithm
    if (A_v /= 0 and B_v /= 0) then
      while (A_v /= B_v) loop
        if (A_v >= B_v) then
          A_v := A_v - B_v;
        else
          B_v := B_v - A_v;
        end if;
      end loop;
    else
      A_v := 0;
    end if;
```

Έλεγχος ορθής λειτουργίας με testbench (3)

Αλγ. υλοποίηση και έλεγχος ορθής λειτουργίας (συνέχεια)

```
Y_Ref := A_v;
wait until done = '1';
Y_v := conv_integer(outp);
---- Test GCD algorithm
if (Y_v /= Y_Ref) then -- has failed
    Passed := '0';
    write(Bufline, string("GCD Error: A="));
    write(Bufline, A_v);
    write(Bufline, string(" B=")); write(Bufline, B_v);
    write(Bufline, string(" Y=")); write(Bufline, Y_v);
    write(Bufline, string(" Y_Ref=")); write(Bufline, Y_Ref);
    writeline(ResultsFile, Bufline);
else
    ncycles_v := ncycles;
    write(Bufline, string("GCD OK: Number of cycles="));
    write(Bufline, ncycles_v);
    writeline(ResultsFile, Bufline);
end if;
end loop;
if (Passed = '1') then -- has passed
    write(Bufline, string("GCD algorithm test has passed"));
    writeline(ResultsFile, Bufline);
end if;
wait for CLK_PERIOD;
end process GCD_EMUL;
```

Έλεγχος ορθής λειτουργίας με testbench (4)

- Εκτύπωση διαγνωστικής εξόδου στο αρχείο "gcd_alg_test_results.txt"

```
GCD OK: Number of cycles=7  
GCD OK: Number of cycles=8  
GCD OK: Number of cycles=10  
GCD OK: Number of cycles=3  
GCD OK: Number of cycles=12  
GCD OK: Number of cycles=52  
GCD OK: Number of cycles=3  
GCD OK: Number of cycles=2  
GCD algorithm test has passed
```