Γλώσσες Περιγραφής Υλικού

Μη προγραμματιζόμενοι επεξεργαστές

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

26 Μαΐου 2009

Σκιαγράφηση της διάλεξης

- Μη προγραμματιζόμενοι επεξεργαστές
  - Υλοποίηση με ξεχωριστό χειριστή ελέγχου (controller) και χειριστή δεδομένων (datapath)
  - Υλοποίηση με μηχανές πεπερασμένων καταστάσεων με χειριστή δεδομένων (FSMD: Finite-State Machine with Datapath)
  - Εξομοίωση της συμπεριφοράς ενός μη προγραμματιζόμενου επεξεργαστή από μη συνθέσιμο αλγοριθμικό μοντέλο
- Πλήρες παράδειγμα: επεξεργαστής για τον υπολογισμό του μέγιστου κοινού διαιρέτη (GCD) δύο θετικών ακεραίων αριθμών

Μη προγραμματιζόμενοι επεξεργαστές

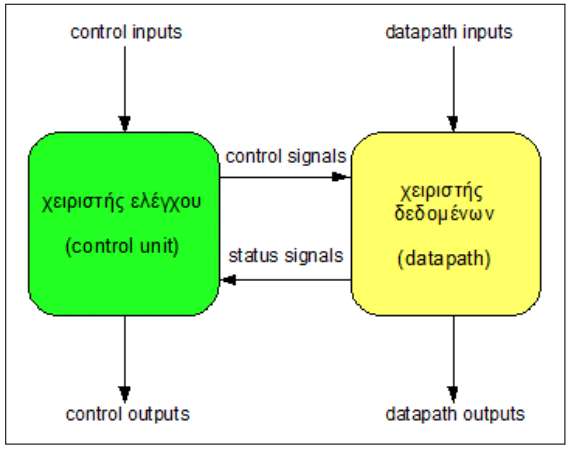
- Μη προγραμματιζόμενοι επεξεργαστές είναι εκείνα τα κυκλώματα τα οποία έχουν σχεδιαστεί έτσι ώστε να μπορούν να επιλύσουν ένα μόνο πρόβλημα
- Η μη προγραμματισιμότητα οφείλεται στον τρόπο σχεδιασμού των μηχανισμών ελέγχου των διαδικασιών επεξεργασίας δεδομένων που συμβαίνουν στον επεξεργαστή
- Η λογική ελέγχου στον επεξεργαστή είναι καλωδιωμένη (hardwired) και γενικά δεν μπορεί να τροποποιηθεί μετά την υλοποίηση του επεξεργαστή
- Ένας μη προγραμματιζόμενος επεξεργαστής αποτελείται από το χειριστή ελέγχου (controller ή control unit) και το χειριστή δεδομένων (datapath)

Η οργάνωση ενός μη-προγραμματιζόμενου επεξεργαστή (1)

- Ο χειριστής ελέγχου παράγει σήματα ελέγχου για την δρομολόγηση των μηχανισμών που λαμβάνουν χώρα στον χειριστή δεδομένων
- Στο χειριστή δεδομένων πραγματοποιείται η επεξεργασία δεδομένων με τα αρχικά δεδομένα να λαμβάνονται από κάποια εξωτερική πηγή ή από στοιχεία αποθήκευσης (π.χ. μνήμη ROM)
- 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 (2)

- Γενικό σχηματικό διάγραμμα ενός μη προγραμματιζόμενου επεξεργαστή



Μη προγραμματιζόμενοι επεξεργαστές με αρχιτεκτονική FSMD

- Η αρχιτεκτονική FSMD (Finite-State Machine with Datapath αποτελεί ένα είδος περιγραφής μη προγραμματιζόμενων επεξεργαστών στην οποία οι καταστάσεις του FSM το οποίο λειτουργεί ως λογική ελέγχου ενσωματώνει τους μηχανισμούς του χειριστή δεδομένων
- Ένα FSMD μπορεί να υλοποιήσει πλήρως τη συμπεριφορά ενός επεξεργαστή σε επίπεδο RTL

Το πρόβλημα του μέγιστου κοινού διαιρέτη δύο αριθμών

- Δεχόμαστε ότι: $gcd(n, 0) = gcd(0, n) = gcd(0, 0) = 0$
- 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 την μέτρηση δύο ευθυγράμμων 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

- Ο αλγόριθμος υπολογισμού του Μέγιστου Κοινού Διαιρέτη δύο θετικών ακεραίων

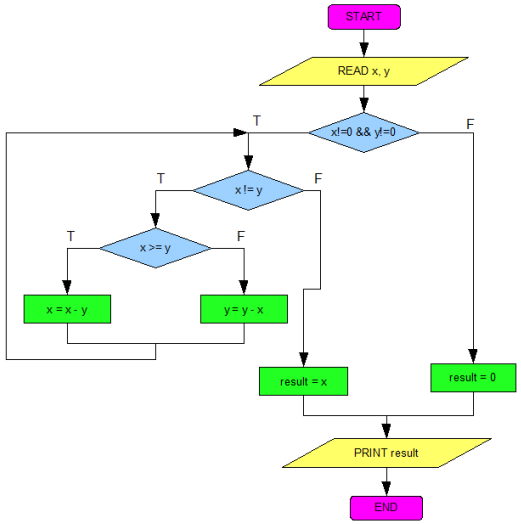
```
int gcd(int a, int b)
{
  int result;
  int x, y;

  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);
}
```

Αλγοριθμικό διάγραμμα ροής για τον αλγόριθμο GCD



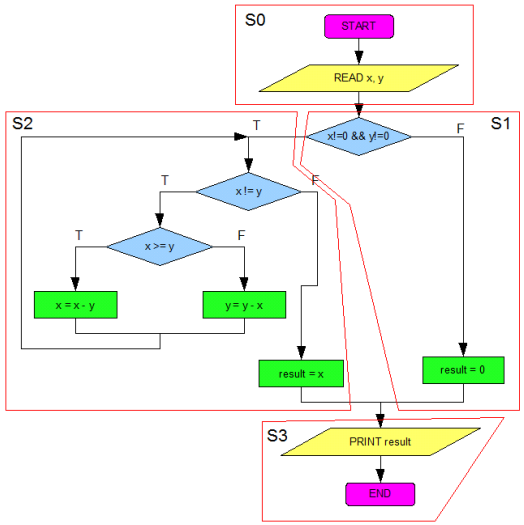
Αριθμητικό παράδειγμα υπολογισμού του GCD

- 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 τετριμμένο 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: $gcd(196, 42) = 14$

Διαχωρισμός του διαγράμματος ροής σε καταστάσεις



Τεχνικές περιγραφής ενός μη προγραμματιζόμενου επεξεργαστή GCD

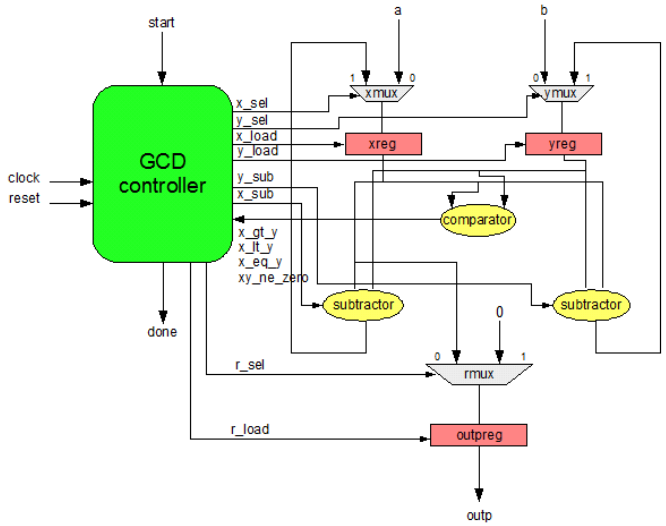
- Μη συνθέσιμη περιγραφή
  - Περιγραφή της συμπεριφοράς του επεξεργαστή σε αλγοριθμικό επίπεδο
  - Χρήση δομών υψηλού επιπέδου όπως δομές επανάληψης `for ... loop` και `while ... loop`
  - Συχνά χρησιμοποιείται ως τμήμα ενός testbench για την εξομοίωση της συμπεριφοράς του κυκλώματος και την εξαγωγή εξόδων αναφοράς (reference vectors)
- Συνθέσιμη περιγραφή
  - Υλοποίηση με χειριστή ελέγχου (τύπου FSM) και χειριστή δεδομένων (datapath)
  - Υλοποίηση με αρχιτεκτονική FSMD

Μη συνθέσιμη περιγραφή του GCD

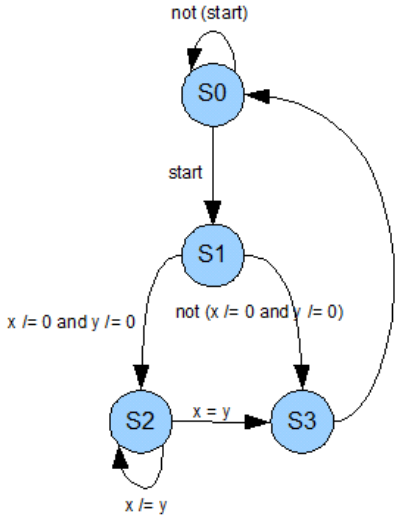
- Έστω A, B οι είσοδοι δεδομένων και Y η έξοδος με την τιμή του GCD
- Υλοποίηση σε μία process

```
GCD: process
  variable A,B,Y: integer range 0 to 65535;
begin
  ...
  if (A /= 0 and B /= 0) then
    while (B /= 0) loop
      if (A >= B) then
        A := A - B;
      else
        B := B - A;
      end if;
    end loop;
  else
    A := 0;
  end if;
  Y := A;
  ...
end process GCD;
```


Το συνολικό κύκλωμα του επεξεργαστή GCD



Διάγραμμα καταστάσεων για το FSM του χειριστή ελέγχου



Περιγραφή του FSM (1)

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

entity gcd_controller is
  port (
    clock : in std_logic;
    reset : in std_logic;
    start : in std_logic;
    xy_ne_zero : in std_logic;
    x_gt_y : in std_logic;
    x_lt_y : in std_logic;
    x_eq_y : in std_logic;
    x_load : out std_logic;
    y_load : out std_logic;
    r_load : out std_logic;
    x_sel : out std_logic;
    y_sel : out std_logic;
    r_sel : out std_logic;
    x_sub : out std_logic;
    y_sub : out std_logic;
    done  : out std_logic
  );
end gcd_controller;
```

```
architecture fsm of gcd_controller is
  type state_type is (s0,s1,s2,s3);
  signal current_state, next_state:
    state_type;
begin
  process1: process (clock, reset)
  begin
  if (reset = '1') then
    current_state <= s0;
  elsif (clock = '1' and clock'EVENT) then
    current_state <= next_state;
  end if;
  end process process1;

  process2: process (current_state,
    start, xy_ne_zero,
    x_gt_y, x_lt_y, x_eq_y)
  begin
    x_load <= '0';
    y_load <= '0';
    r_load <= '0';
    x_sel <= '0';
    y_sel <= '0';
    r_sel <= '0';
    x_sub <= '0';
    y_sub <= '0';
    done <= '0';
```

Περιγραφή του FSM (2)

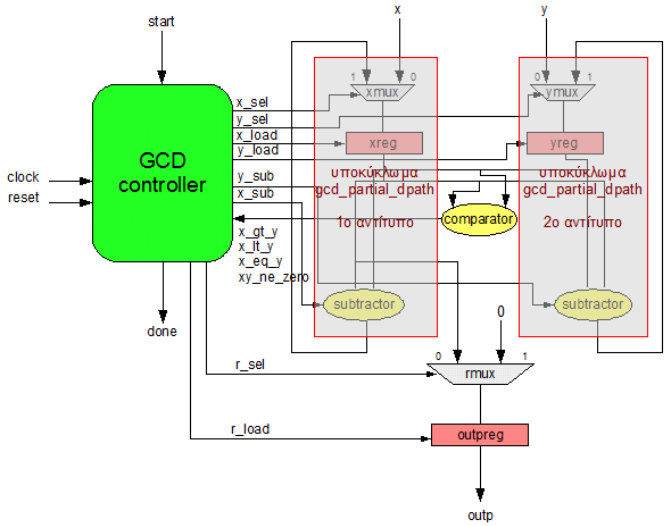
```
case current_state is
  when s0 =>
    if (start = '1') then
      x_load <= '1';
      y_load <= '1';
      next_state <= s1;
    else
      next_state <= s0;
    end if;
  when s1 =>
    if (xy_ne_zero = '1') then
      next_state <= s2;
    else
      r_load <= '1';
      r_sel <= '1';
      next_state <= s3;
    end if;
```

```
  when s2 =>
        if (x_gt_y = '1') then
          x_load <= '1';
          x_sel <= '1';
          x_sub <= '1';
          next_state <= s2;
        elsif (x_lt_y = '1') then
          y_load <= '1';
          y_sel <= '1';
          y_sub <= '1';
          next_state <= s2;
        else
          r_load <= '1';
          next_state <= s3;
        end if;
      when s3 =>
        done <= '1';
        next_state <= s0;
    end case;
  end process process2;
end fsm;
```


Ιεραρχική περιγραφή του χειριστή δεδομένων (1)

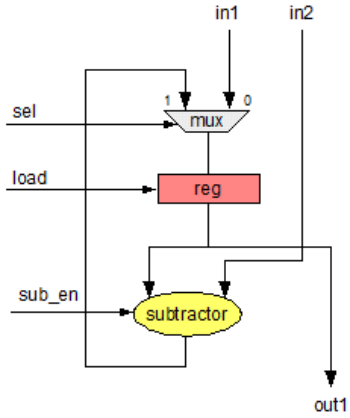
- Ο πολυπλέκτης επιλογής μιας εισόδου (a ή b) του κυκλώματος και του αποτελέσματος της αφαίρεσης $x = x - y$ ή $y = y - x$, ο καταχωρητής x (y) και ο αφαιρέτης εμφανίζονται ως υποκύκλωμα σε δύο αντίτυπα στο κύκλωμα του χειριστή δεδομένων του GCD
- Το υποκύκλωμα αυτό μπορεί να περιγραφεί σε ξεχωριστή entity (gcd_partial_dpath) για την καλύτερη δόμηση της περιγραφής του χειριστή δεδομένων

Ιεραρχική περιγραφή του χειριστή δεδομένων (2)



Το υποκύκλωμα gcd_partial_dpath

- Σχηματικό διάγραμμα του κοινού υποκυκλώματος για τις εισόδους a και b



Περιγραφή του gcd_partial_dpath

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

entity gcd_partial_dpath is
  generic (WIDTH : integer);
  port (
    clock: in std_logic;
    reset: in std_logic;
    sel: in std_logic;
    load: in std_logic;
    sub_en: in std_logic;
    in1: in
      std_logic_vector(WIDTH-1 downto 0);
    in2: in
      std_logic_vector(WIDTH-1 downto 0);
    out1: out
      std_logic_vector(WIDTH-1 downto 0)
  );
end gcd_partial_dpath;

architecture rtl of gcd_partial_dpath is
  signal in1_t, in1_r, res_t :
    std_logic_vector(WIDTH-1 downto 0);
begin
  process (sel, in1, res_t)
  begin
  if (sel = '0') then
    in1_t <= in1;
  else
    in1_t <= res_t;
  end if;
  end process;

  process (clock)
  begin
  if (clock = '1' and clock'EVENT) then
    if (reset = '1') then
      in1_r <= (others => '0');
    else
      if (load = '1') then
        in1_r <= in1_t;
      end if;
    end if;
  end if;
end process;

process (sub_en, in1_r, in2)
begin
if (sub_en = '1') then
  res_t <= in1_r - in2;
else
  res_t <= in1_r;
end if;
end process;
out1 <= in1_r;
end rtl;
```


Το πακέτο gcd_pkg

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

PACKAGE gcd_pkg IS
component gcd_controller
  port (
    clock : in std_logic;
    reset : in std_logic;
    start : in std_logic;
    xy_ne_zero : in std_logic;
    x_gt_y : in std_logic;
    x_lt_y : in std_logic;
    x_eq_y : in std_logic;
    x_load, y_load : out std_logic;
    r_load : out std_logic;
    x_sel, y_sel, r_sel : out std_logic;
    x_sub, y_sub : out std_logic;
    done : out std_logic
  );
end component;
component gcd_partial_dpath
  generic (WIDTH : integer);
  port (
    clock : in std_logic;
    reset : in std_logic;
    sel, load : in std_logic;
    sub_en : in std_logic;
    in1, in2 : in
      std_logic_vector(WIDTH-1 downto 0);
    out1 : out
      std_logic_vector(WIDTH-1 downto 0)
  );
end component;
component gcd_datapath
  generic (WIDTH : integer);
  port (
    clock  : in std_logic;
    reset  : in std_logic;
    a, b : in
      std_logic_vector(WIDTH-1 downto 0);
    x_load, y_load, r_load : in std_logic;
    x_sel, y_sel, r_sel : in std_logic;
    x_sub, y_sub  : in std_logic;
    xy_ne_zero : out std_logic;
    x_gt_y : out std_logic;
    x_lt_y : out std_logic;
    x_eq_y : out std_logic;
    outp : out
      std_logic_vector(WIDTH-1 downto 0)
  );
end component;
END gcd_pkg;
```


Περιγραφή του gcd_datapath (1)

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

entity gcd_datapath is
  generic (WIDTH : integer);
  port (
    clock  : in std_logic;
    reset  : in std_logic;
    a : in
      std_logic_vector(WIDTH-1 downto 0);
    b : in
      std_logic_vector(WIDTH-1 downto 0);
    x_load : in std_logic;
    y_load : in std_logic;
    r_load : in std_logic;
    x_sel  : in std_logic;
    y_sel  : in std_logic;
    r_sel  : in std_logic;
    x_sub  : in std_logic;
    y_sub  : in std_logic;
    xy_ne_zero : out std_logic;
    x_gt_y : out std_logic;
    x_lt_y : out std_logic;
    x_eq_y : out std_logic;
    outp
: out std_logic_vector(WIDTH-1 downto 0)
  );
end gcd_datapath;

architecture rtl of gcd_datapath is
  signal x, y, res :
    std_logic_vector(WIDTH-1 downto 0);
begin
  x_partial_datapath: gcd_partial_dpath
    generic map (WIDTH => WIDTH)
    port map (
      clock => clock, reset => reset,
      sel => x_sel,
      load => x_load,
      sub_en => x_sub,
      in1 => a, in2 => y,
      out1 => x
    );
```


Περιγραφή του gcd_datapath (2)

```
  y_partial_datapath: gcd_partial_dpath
    generic map (WIDTH => WIDTH)
    port map (
      clock => clock,
      reset => reset,
      sel => y_sel,
      load => y_load,
      sub_en => y_sub,
      in1 => b,
      in2 => x,
      out1 => y
    );
```

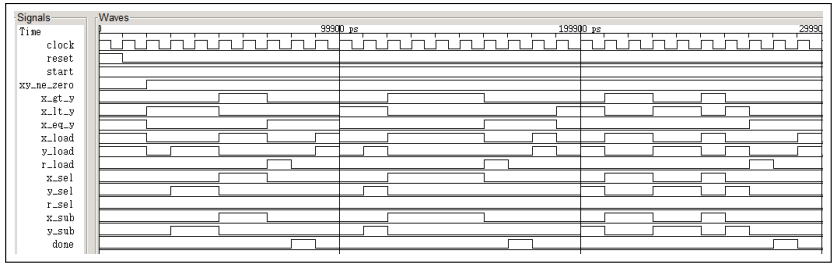
```
  -- res, done reg
  process (clock)
  begin
  if (clock = '1' and clock'EVENT) then
    if (reset = '1') then
      res <= (others => '0');
    else
      if (r_load = '1') then
        if (r_sel = '0') then
          res <= x;
        else
          res <= (others => '0');
        end if;
      end if;
    end if;
  end if;
  end process;

  -- comparator
  x_gt_y <= '1' when (x > y) else '0';
  x_lt_y <= '1' when (x < y) else '0';
  x_eq_y <= '1' when (x = y) else '0';
  xy_ne_zero <= '1' when
    (x /= 0 and y /= 0) else '0';

  outp <= res;
end rtl;
```

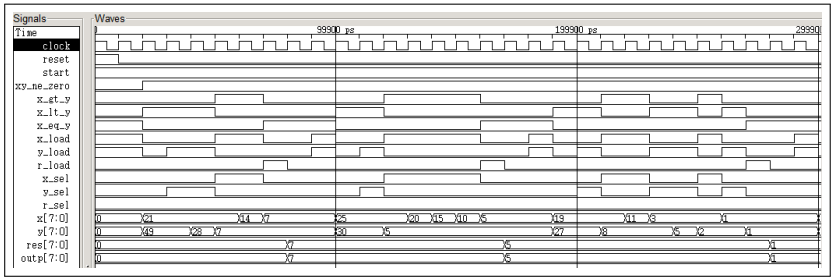
Προσομοίωση του επεξεργαστή GCD (χειριστής ελέγχου)

Χρονικό διάγραμμα του χειριστή ελέγχου (gcd_controller)



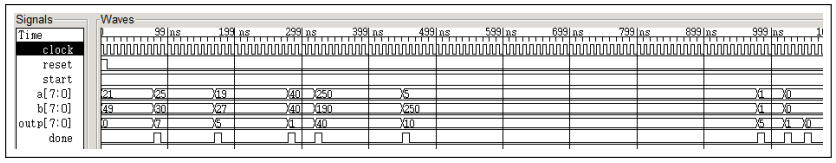
Προσομοίωση του επεξεργαστή GCD (χειριστής δεδομένων)

Χρονικό διάγραμμα του χειριστή δεδομένων (gcd_datapath)

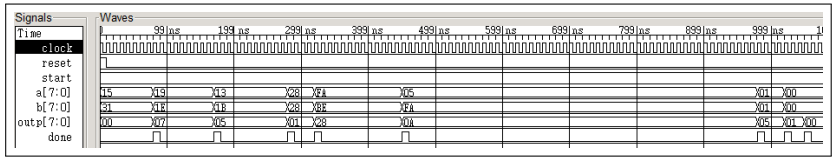


Προσομοίωση του επεξεργαστή GCD (διεπαφή του συνολικού κυκλώματος)

Χρονικό διάγραμμα (τιμές στο δεκαδικό)



Χρονικό διάγραμμα (τιμές στο δεκαεξαδικό)



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 case;
    end if;
  end process;

  outp <= res;

end fsmd;
```


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)

- Δήλωση `SIGNAL` τύπου `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";
```

- Τα περιεχόμενα του αρχείου κειμένου `"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 (3)

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 (4)

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 (5)

- 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
```