# Γλώσσες Περιγραφής Υλικού

## Δομές ελέγχου/επαλήθευσης λειτουργίας των κυκλωμάτων

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

5 Μαΐου 2009

# Σκιαγράφηση της διάλεξης

- Δομές ελέγχου/επαλήθευσης λειτουργίας των κυκλωμάτων
  - Η εντολή ASSERT
  - Κύκλωμα ελέγχου/επαλήθευσης (testbench)
  - Παραγωγή διανυσμάτων αναφοράς
  - Επαλήθευση κυκλώματος ως προς διανύσματα αναφοράς
  - Αρχεία στη VHDL
  - Μηχανισμοί εισόδου/εξόδου και τα πακέτα `STD.STANDARD` και `STD.TEXTIO`

# Η εντολή ASSERT (1)

- Η εντολή ASSERT χρησιμοποιείται για την επιστροφή μηνυμάτων στο τερματικό κατά την προσομοίωση
- Με την ASSERT ελέγχεται μία συνθήκη: π.χ. αν ένα SIGNAL έχει λάβει συγκεκριμένη αριθμητική τιμή κάποια στιγμή κατά τη διάρκεια της προσομοίωσης
- Η ASSERT αποτελείται από τρία τμήματα
  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

☣ Η εντολή ASSERT δεν είναι συνθέσιμη και η χρήση της περιορίζεται αποκλειστικά σε testbenches

# Η εντολή ASSERT (2)

- Σύνταξη της εντολής ASSERT

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

- Τα επίπεδα σοβαρότητας είναι: σημείωση (NOTE), προειδοποίηση (WARNING), σφάλμα (ERROR), ή αποτυχία (FAILURE)
- Το μήνυμα επιστρέφεται όταν η τιμή της συνθήκης είναι FALSE
- Παράδειγμα εντολής ASSERT

```
signal a: STD_LOGIC_VECTOR(7 downto 0);
signal b: STD_LOGIC_VECTOR(8 downto 2);
...
ASSERT (a'LENGTH = b'LENGTH)
  REPORT "Error: vectors do not have the same length!"
  SEVERITY FAILURE;
```

# Επίπεδα σοβαρότητας σε μια εντολή ASSERT

Ο παρακάτω πίνακας συνοψίζει το ρόλο των επίπεδων σοβαρότητας

| **Επίπεδο σοβαρότητας** | **Χρήση** |
|---|---|
| `NOTE` | Γενική πληροφορία για την κατάσταση μέρους του κυκλώματος |
| `WARNING` | Ενημέρωση για πιθανά προβλήματα κάτω από ορισμένες συνθήκες λειτουργίας του κυκλώματος |
| `ERROR` | Ενημέρωση για συνθήκες λειτουργίας του κυκλώματος που μπορεί να προκαλέσουν σφάλματα |
| `FAILURE` | Ενημέρωση για συνθήκες λειτουργίας με ενδεχόμενα καταστροφικά αποτελέσματα |

# Αρχεία στη VHDL

- Ο τύπος ΑΡΧΕΙΟΥ (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)
- Η δήλωση ενός αρχείου επιτρέπει σε κώδικα VHDL να γίνει κάποια χρήση του αρχείου
- Ένα αρχείο μπορεί να ανοιχτεί για ανάγνωση ή εγγραφή
- Από το πρότυπο VHDL-93 και έπειτα, οι λειτουργίες ανοίγματος και κλεισίματος ενός αρχείου πραγματοποιούνται με τις διαδικασίες `FILE_OPEN()` και `FILE_CLOSE()`
- Στη βιβλιοθήκη STD
  - Το πακέτο STANDARD ορίζει βασικές ρουτίνες για είσοδο και έξοδο (I/O) αντικειμένων των βασικών τύπων της VHDL
  - Το πακέτο TEXTIO ορίζει πιο ισχυρές ρουτίνες για το χειρισμό αρχείων κειμένου

# Ορισμοί για το χειρισμό αρχείων στη VHDL (1)

- Ορισμός του αρχείου κειμένου (πακέτο TEXTIO)

```
type LINE is access STRING; -- A LINE is a pointer to a STRING value.
-- Predefined operators for type LINE: "=", "/="
type TEXT is file of STRING;
```

- Τα προκαθορισμένα αρχεία κειμένου INPUT και OUTPUT (πακέτο TEXTIO)

```
-- Standard text files:
file INPUT: TEXT open READ_MODE is "STD_INPUT";
file OUTPUT: TEXT open WRITE_MODE is "STD_OUTPUT";
```

- Τρόποι ανοίγματος ενός αρχείου (πακέτο STANDARD)

```
type FILE_OPEN_KIND is (
  READ_MODE,     -- Resulting access mode is read-only.
  WRITE_MODE,    -- Resulting access mode is write-only.
  APPEND_MODE); -- Resulting access mode is write-only; information
                 -- is appended to the end of the existing file.
```

# Ορισμοί για το χειρισμό αρχείων στη VHDL (2)

- Τύπος κατάστασης ενός αρχείου (πακέτο STANDARD)

```
type FILE_OPEN_STATUS is (
  OPEN_OK,      -- File open was successful.
  STATUS_ERROR, -- File object was already open.
  NAME_ERROR,   -- External file not found or inaccessible.
  MODE_ERROR);  -- Could not open file with requested access mode.
```

- 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);
procedure READ (file F: TEXT; VALUE: out STRING);
procedure WRITE (file F: TEXT; VALUE: in STRING);
function ENDFILE (file F: TEXT) return BOOLEAN;
```

- Ο τύπος SIDE και ο υποτύπος WIDTH

```
type SIDE is (RIGHT, LEFT); -- For justifying output data within fields.
-- Predefined operators for the SIDE type:
-- "=", "/=", "<", "<=", ">", ">="
subtype WIDTH is natural;
```

# Ρουτίνες εισόδου (ανάγνωσης) για αρχεία κειμένου (πακέτο TEXTIO)

```
procedure READLINE (file F: TEXT; L: inout LINE);
procedure READ (L: inout LINE; VALUE: out BIT; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out BIT);
procedure READ (L: inout LINE; VALUE: out BIT_VECTOR; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out BIT_VECTOR);
procedure READ (L: inout LINE; VALUE: out BOOLEAN; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out CHARACTER; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out CHARACTER);
procedure READ (L: inout LINE; VALUE: out INTEGER; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out INTEGER);
procedure READ (L: inout LINE; VALUE: out REAL; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out REAL);
procedure READ (L: inout LINE; VALUE: out STRING; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out STRING);
procedure READ (L: inout LINE; VALUE: out TIME; GOOD: out BOOLEAN);
procedure READ (L: inout LINE; VALUE: out TIME);
```

# Ρουτίνες εξόδου (εγγραφής) για αρχεία κειμένου (πακέτο TEXTIO)

```
procedure WRITELINE (file F: TEXT; L: inout LINE);
procedure WRITE (L: inout LINE; VALUE: in BIT;
                 JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in BIT_VECTOR;
                 JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in BOOLEAN;
                 JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in CHARACTER;
                 JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in INTEGER;
                 JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in REAL;
                 JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0;
                 DIGITS: in NATURAL:= 0);
procedure WRITE (L: inout LINE; VALUE: in STRING;
                 JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0);
procedure WRITE (L: inout LINE; VALUE: in TIME;
                 JUSTIFIED: in SIDE:= RIGHT; FIELD: in WIDTH := 0;
                 UNIT: in TIME:= ns);
```

# Ρουτίνες εισόδου και εξόδου από το μη τυποποιημένο πακέτο IEEE.STD_LOGIC_TEXTIO (1)

- Στο πακέτο STD_LOGIC_TEXTIO περιγράφονται ορισμένες χρήσιμες συναρτήσεις για την είσοδο και έξοδο αντικειμένων του τύπου STD_LOGIC_VECTOR
- Περιγράφονται επίσης και συναρτήσεις για την είσοδο και έξοδο αριθμών δεκαεξαδικής βάσης (hexadecimal)
- ℹ Το πακέτο αυτό συνήθως είναι μεταγλωττισμένο στη βιβλιοθήκη IEEE χωρίς όμως να ανήκει σε αυτήν

```
-- Read and Write procedures for STD_LOGIC_VECTOR
procedure READ(L:inout LINE; VALUE:out STD_LOGIC_VECTOR);
procedure READ(L:inout LINE; VALUE:out STD_LOGIC_VECTOR; GOOD: out BOOLEAN);
procedure WRITE(L:inout LINE; VALUE:in STD_LOGIC_VECTOR;
                JUSTIFIED:in SIDE := RIGHT; FIELD:in WIDTH := 0);
```

# Ρουτίνες εισόδου και εξόδου από το μη τυποποιημένο πακέτο IEEE.STD_LOGIC_TEXTIO (2)

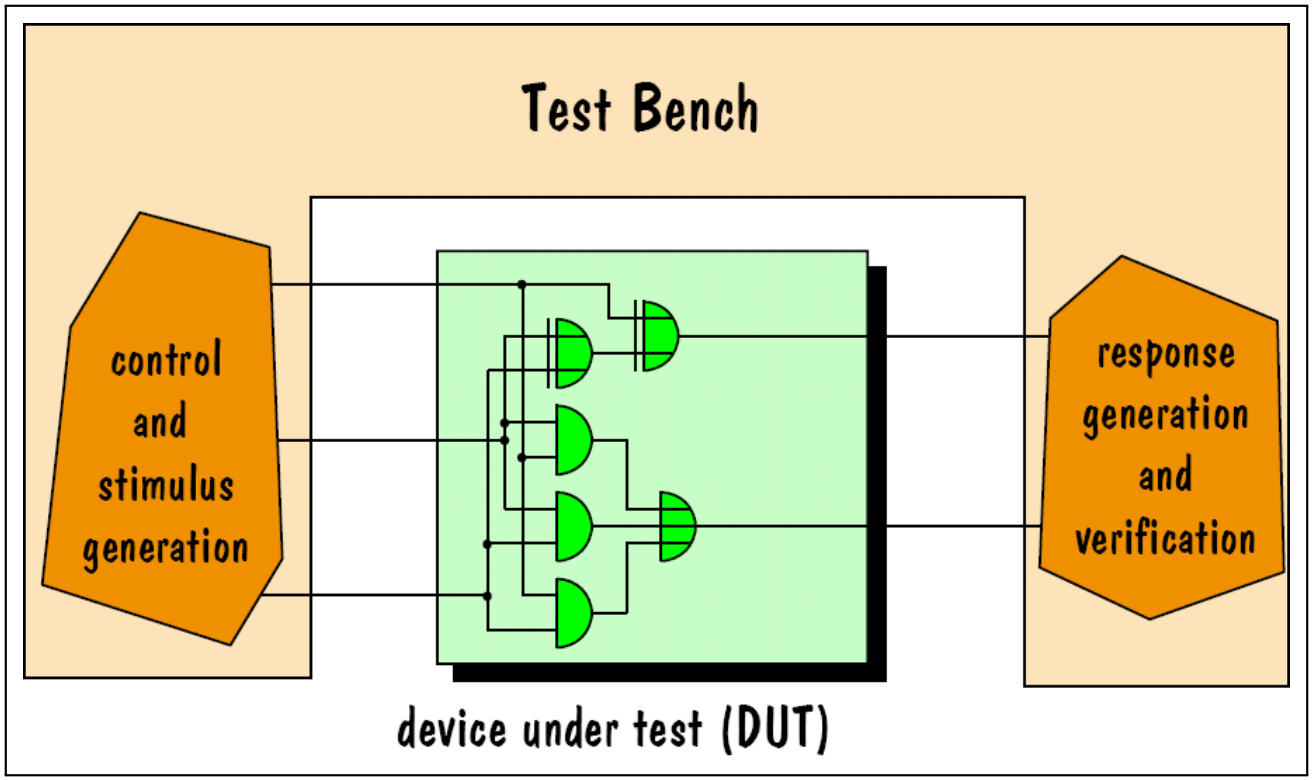
```
procedure HREAD(L:inout LINE; VALUE:out STD_ULOGIC_VECTOR);
procedure HREAD(L:inout LINE; VALUE:out STD_ULOGIC_VECTOR; GOOD: out BOOLEAN);
procedure HWRITE(L:inout LINE; VALUE:in STD_ULOGIC_VECTOR;
                 JUSTIFIED:in SIDE := RIGHT; FIELD:in WIDTH := 0);
procedure HREAD(L:inout LINE; VALUE:out STD_LOGIC_VECTOR);
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);
```

# Περιγραφές ελέγχου/επαλήθευσης λειτουργίας (testbenches)

- Σε μία σχεδίαση, το testbench αντιστοιχεί στο υψηλότερο επίπεδο ιεραρχίας
- Η entity ενός testbench δεν περιλαμβάνει καμία δήλωση θύρας, μπορεί όμως να περιλαμβάνει generics
- Στο testbench, δηλώνεται το `COMPONENT` του συνολικού κυκλώματος
- Ένα testbench μπορεί να χρησιμοποιηθεί και για τα εξής:
  - Παραγωγή σημάτων διέγερσης για την προσομοίωση
  - Την υλοποίηση των μηχανισμών του κυκλώματος, αλλά σε επίπεδο behavioral για τη δημιουργία διανυσμάτων αναφοράς (reference vectors)
  - Την παράλληλη (μαζί με την εφαρμογή εισόδων στο εξεταζόμενο κύκλωμα) υλοποίηση του σε επίπεδο behavioral για την άμεση σύγκριση των αναμενόμενων με τις πραγματικές τιμές εξόδου κατά την προσομοίωση

# Λειτουργία ενός testbench

- Το testbench αποτελεί ένα εικονικό κύκλωμα το οποίο εφαρμόζει εισόδους προς (διέγερση: stimulus) και λαμβάνει εξόδους (απόκριση: response) από το πραγματικό κύκλωμα



# Οργάνωση της περιγραφής ενός testbench

- Τυπική οργάνωση ενός αρχείου testbench

```
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;
```

# Παραγωγή σήματος ρολογιού για χρήση σε testbench

- Για τον έλεγχο της λειτουργίας σύγχρονων κυκλωμάτων χρειάζεται η δημιουργία μιας εικονικής γεννήτριας ρολογιού
- Περιγράφεται σε ξεχωριστή διεργασία στο testbench σε σχέση με τη διέγερση των άλλων εισόδων του κυκλώματος
- Σε ορισμένες περιπτώσεις, σε ξεχωριστή διεργασία πραγματοποιείται και η γέννηση του σήματος reset

```
architecture tb_arch of counter is
  component counter
    port (
      clk, reset : in std_logic;
      q : out unsigned(3 downto 0)
    );
  end component;
  ...
  signal clk: std_logic;
  ...
  constant CLK_PERIOD : time := 50 ns;
begin
```

```
  ...
  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;
  ...
end tb_arch;
```

# Οι βασικές λειτουργίες εισόδου και εξόδου κειμένου

- Οι βασικές λειτουργίες με περιεχόμενα αρχείων στη VHDL περιορίζονται στην αφορμάριστη (unformatted) είσοδο και έξοδο χαρακτήρων ASCII προς και από αρχεία, χρησιμοποιώντας τις διαδικασίες του πακέτου `TEXTIO`
- Το πακέτο `TEXTIO` υποστηρίζει τους παρακάτω τύπους δεδομένων
  - `BIT, BIT_VECTOR`
  - `BOOLEAN`
  - `CHARACTER` και πίνακες χαρακτήρων (STRING)
  - `INTEGER` και `REAL`
  - `TIME`

# Το πακέτο STANDARD: Γενικοί τύποι και υποτύποι

```
type BOOLEAN is (FALSE, TRUE);
type BIT is ('0', '1');
-- Predefined operators for the BOOLEAN and BIT types:
-- "and", "or", "nand", "nor", "xor", "xnor", "not"
-- "=", "/=", "<", "<=", ">", ">="

type INTEGER is range -2147483647 to 2147483647;
-- Predefined operators for the INTEGER type:
-- "=", "/=", "<", "<=", ">", ">="
-- unary: "+" , "-", "abs"
-- binary: "+", "-", "*", "/", "mod", "rem", "**"

type REAL is range -1.0E308 to 1.0E308;
-- Predefined operators for the REAL type:
-- "=", "/=", "<", "<=", ">", ">="
-- unary: "+" , "-", "abs"
-- binary: "+", "-", "*", "/", "**"

type BIT_VECTOR is array (NATURAL range <>) of BIT;
-- Predefined operators for the BIT_VECTOR type:
-- "and", "or", "nand", "nor", "xor", "xnor", "not"
-- "sll", "srl", "sla", "sra", "rol", "ror"
-- "=", "/=", "<", "<=", ">", ">="
-- "&" (accepting BIT, BIT_VECTOR in any of the four combinations)

subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
```

# Το πακέτο STANDARD: Οι τύποι TIME και SEVERITY

```
type TIME is range -2147483647 to 2147483647
units
  fs; -- femtosecond
  ps = 1000 fs; -- picosecond
  ns = 1000 ps; -- nanosecond
  us = 1000 ns; -- microsecond
  ms = 1000 us; -- millisecond
  sec = 1000 ms; -- second
  min = 60 sec; -- minute
  hr = 60 min; -- hour
end units;

-- Predefined operators for the TIME type:
-- "=", "/=", "<", "<=", ">", ">="
-- unary: "+", "-", "abs"
-- binary: "+", "-" (accepting TIME)
--         "*" (accepting TIME and INTEGER or TIME and REAL)
--         "/" (accepting TIME and INTEGER or TIME and REAL or TIME and TIME, thus
--              returning an integer with no dimensions)

type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);

-- The predefined operators for this type are as follows:
-- Predefined operators for the SEVERITY type:
-- "=", "/=", "<", "<=", ">", ">="
```

# Το πακέτο STANDARD: Οι τύποι CHARACTER και STRING

```
type CHARACTER is (
  NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT, FF, CR, SO, SI,
  DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FSP, GSP, RSP, USP,
  ' ', '!', '"', '#', '$', '%', '&', ''', '(', ')', '*', '+', ',', '-', '.', '/',
  '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?',
  '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
  'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\', ']', '^', '_',
  '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
  'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', DEL,
  ...
);
-- Predefined operators for the CHARACTER type:
-- "=", "/=", "<", "<=", ">", ">="

type STRING is array (POSITIVE range <>) of CHARACTER;
-- Predefined operators for the STRING type:
-- "=", "/=", "<", "<=", ">", ">="
-- "&" (accepting CHARACTER, STRING in any of the four combinations)
```

# Χρησιμοποιώντας το πακέτο TEXTIO για είσοδο και έξοδο από αρχεία

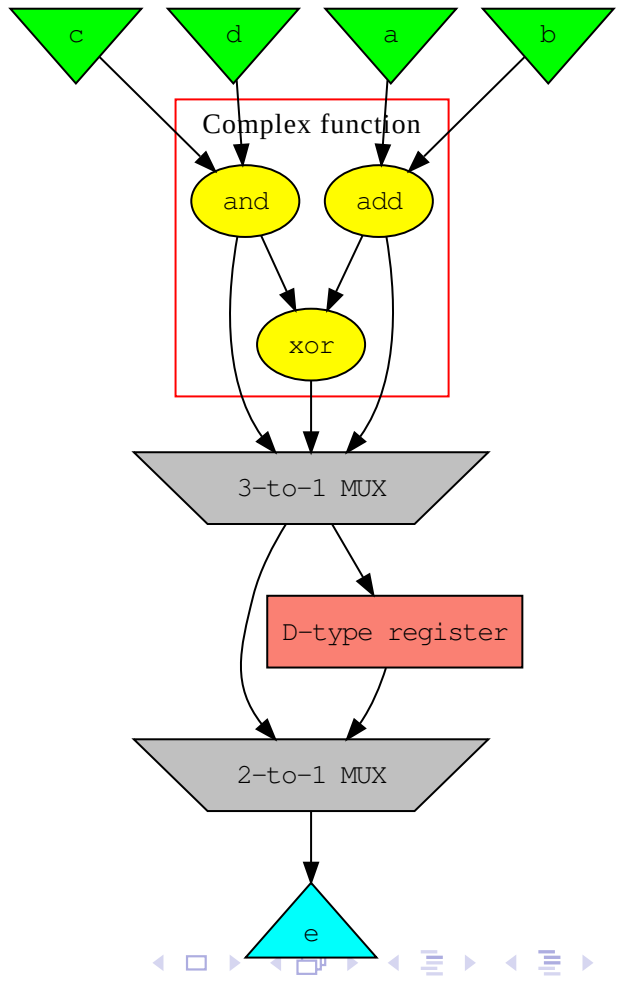
- Ανάγνωση από αρχείο
  1. Η συνάρτηση `READLINE` διαβάζει μία σειρά από το αρχείο και την αποθηκεύει σε μία μεταβλητή τύπου `LINE`
  2. Η συνάρτηση `READ` ανακτά τα δεδομένα από το χώρο προσωρινής αποθήκευσης που προσφέρει η μεταβλητή τύπου `LINE`
- Εγγραφή σε αρχείο
  1. Η `WRITE` γράφει δεδομένα σε μία μεταβλητή `LINE`
  2. Η `WRITELINE` γράφει τα προσωρινά αποθηκευμένα δεδομένα που διατηρούνται στη `LINE`, στο αρχείο
- Οι συναρτήσεις `READ` και `WRITE` διαθέτουν παραμέτρους διαμόρφωσης για το κείμενο εισόδου και εξόδου
  - Εύρος πεδίου
  - Δεξιά ή αριστερή στοίχιση
  - Απεικονιζόμενη μονάδα του χρόνου (για μεταβλητές `TIME`)

# Παράδειγμα 1: Διέγερση σημάτων εισόδου από process (1)

- Η παρακάτω entity αντιστοιχεί σε κύκλωμα διαδρόμου δεδομένων για τον υπολογισμό της έκφρασης `(a + b) XOR (c AND d)` και των μερικών αποτελεσμάτων `a + b` και `c AND d`

```
...
entity datapath is
  port (
    clk, reset : in std_logic;
    in1 : in std_logic_vector(7 downto 0);
    in2 : in std_logic_vector(7 downto 0);
    in3 : in std_logic_vector(7 downto 0);
    in4 : in std_logic_vector(7 downto 0);
    sel : in std_logic_vector(2 downto 0);
    outp : out std_logic_vector(7 downto 0)
  );
end datapath;
```

- Σχηματικό διάγραμμα



# Παράδειγμα 1: Διέγερση σημάτων εισόδου από process (2)

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

entity datapath_tb is
end datapath_tb;

architecture tb_arch of datapath_tb is
  component datapath
    port (
      clk, reset : in std_logic;
      in1, in2, in3, in4 : in std_logic_vector(7 downto 0);
      sel : in std_logic_vector(2 downto 0);
      outp : out std_logic_vector(7 downto 0)
    );
  end component;
  signal clk, reset: std_logic;
  signal in1, in2, in3, in4: std_logic_vector(7 downto 0);
  signal sel: std_logic_vector(2 downto 0);
  signal outp : std_logic_vector(7 downto 0);
  constant CLK_PERIOD : time := 50 ns;
begin
```

# Παράδειγμα 1: Διέγερση σημάτων εισόδου από process (3)

```
  uut : datapath
    port map (
      clk => clk, reset => reset,
      in1 => in1, in2 => in2,  in3 => in3, in4 => in4,
      sel => sel, outp => outp);

  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;
end tb_arch;
```

# Παράδειγμα 2: Εγγραφή αποτελεσμάτων σε αρχείο εξόδου (1)

- Κύκλωμα παραμετρικού αθροιστή σε επίπεδο RTL

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

entity add is
  generic (
    DW : integer := 8
  );
  port (
    a   : in std_logic_vector(DW-1 downto 0);
    b   : in std_logic_vector(DW-1 downto 0);
    sum : out std_logic_vector(DW-1 downto 0)
  );
end add;

architecture rtl of add is
begin
sum <= a + b;
end rtl;
```

# Παράδειγμα 2: Εγγραφή αποτελεσμάτων σε αρχείο εξόδου (2)

- Περιγραφή του αρχείου testbench

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
use std.textio.all;

entity add_tb is
  generic (Dw: INTEGER:= 8);
end add_tb;

architecture TB_ARCHITECTURE of add_tb is
  -- Component declaration of the tested unit
  component add
    generic (Dw: INTEGER:= 8);
    port (
      a : in std_logic_vector (Dw-1 downto 0);
      b : in std_logic_vector (Dw-1 downto 0);
      sum : out std_logic_vector (Dw-1 downto 0)
    );
  end component;
  -- Stimulus signals
  signal a : std_logic_vector(Dw-1 downto 0);
  signal b : std_logic_vector(Dw-1 downto 0);
  -- Observed signals
  signal sum : std_logic_vector(Dw-1 downto 0);
  --
  file output_log : text open write_mode is "add.log";
```

# Παράδειγμα 2: Εγγραφή αποτελεσμάτων σε αρχείο εξόδου (3)

```
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;
    a <= X"E5";
    b <= X"9A";
    wait for 10 ns;
    a <= X"FD";
    b <= X"01";
    wait for 10 ns;
    a <= X"FE";
    b <= X"07";
    wait for 10 ns;
  end process;
```

```
  output_log_proc: process
  variable out_line : line;
  begin
    write(out_line, NOW, left, 8);
    -- Write values for output signals
    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;
end TB_ARCHITECTURE;
```

# Παράδειγμα 2: Εγγραφή αποτελεσμάτων σε αρχείο εξόδου (4)

- Το παραγόμενο αρχείο εξόδου `add.log`

```
0 ns      a:  00  b:  00 sum:  00
10 ns     a:  FF  b:  10 sum:  0F
20 ns     a:  10  b:  89 sum:  99
30 ns     a:  E5  b:  9A sum:  7F
40 ns     a:  FD  b:  01 sum:  FE
50 ns     a:  FE  b:  07 sum:  05
60 ns     a:  FF  b:  10 sum:  0F
70 ns     a:  10  b:  89 sum:  99
80 ns     a:  E5  b:  9A sum:  7F
90 ns     a:  FD  b:  01 sum:  FE
100 ns    a:  FE  b:  07 sum:  05
```

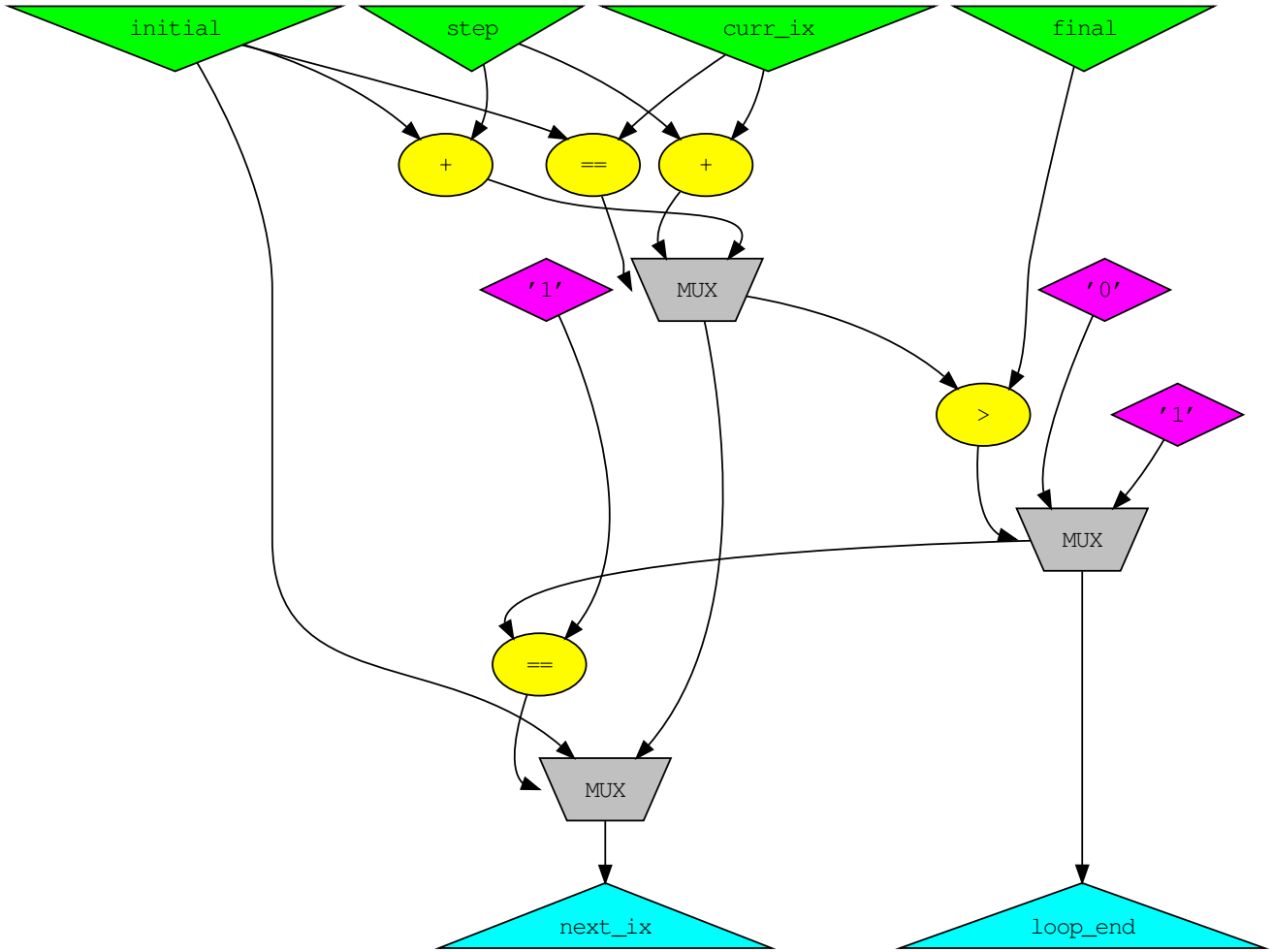
# Παράδειγμα 3: Λήψη διέγερσης και σύγκριση με διανύσματα αναφοράς από αρχείο (1)

- Σχεδιασμός και έλεγχος κυκλώματος το οποίο υλοποιεί τον έλεγχο ενός βρόχου στο υλικό
- Παράγει τον επόμενο δείκτη βρόχου (next_ix) γνωρίζοντας τις παραμέτρους του βρόχου (initial, step, final) και την τρέχουσα τιμή του δείκτη. Επίσης σηματοδοτεί τον τερματισμό του βρόχου (loop_end)
- Το κύκλωμα μπορεί να χρησιμοποιηθεί για τον έλεγχο απλών, μη φωλιασμένων δομών βρόχου της μορφής:

```
for (ix = initial; ix<=final; ix+=step)
{ statements; }
```

# Παράδειγμα 3: Λήψη διέγερσης και σύγκριση με διανύσματα αναφοράς από αρχείο (2)

Σχηματικό διάγραμμα



# Παράδειγμα 3: Λήψη διέγερσης και σύγκριση με διανύσματα αναφοράς από αρχείο (3)

Περιγραφή του κυκλώματος υπολογισμού δείκτη

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

entity hwloop is
  generic (
    DW  : integer := 8
  );
  port (
    initial       : in std_logic_vector(DW-1 downto 0);
    step          : in std_logic_vector(DW-1 downto 0);
    final         : in std_logic_vector(DW-1 downto 0);
    current_index : in std_logic_vector(DW-1 downto 0);
    next_index    : out std_logic_vector(DW-1 downto 0);
    loop_end      : out std_logic
  );
end hwloop;

architecture rtl of hwloop is
  signal add_out    : std_logic_vector(DW-1 downto 0);
  signal loop_end_t : std_logic;
```

# Παράδειγμα 3: Λήψη διέγερσης και σύγκριση με διανύσματα αναφοράς από αρχείο (4)

Περιγραφή του κυκλώματος υπολογισμού δείκτη (συνέχεια)

```
begin
process (initial, step, final, current_index, add_out, loop_end_t)
  begin
    if (current_index = initial) then
      add_out <= initial + step;
    else
      add_out <= current_index + step;
    end if;
    if (add_out > final) then
      loop_end_t <= '1';
    else
      loop_end_t <= '0';
    end if;
    if (loop_end_t = '1') then
      next_index <= initial;
    else
      next_index <= add_out;
    end if;
  end process;
  loop_end <= loop_end_t;
end rtl;
```

# Παράδειγμα 3: Λήψη διέγερσης και σύγκριση με διανύσματα αναφοράς από αρχείο (5)

Αρχείο testbench για το κύκλωμα

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_textio.all;
use std.textio.all;

entity hwloop_tb is
  generic (
    DW : INTEGER := 8
);
end hwloop_tb;

architecture tb_arch of hwloop_tb is
...
  constant CLK_PERIOD  : time := 10 ns;
begin
  UUT : hwloop
    generic map ( DW => DW )
    port map (
      initial => initial,
      step => step,
      final => final,
      current_index => current_index,
      next_index => next_index,
      loop_end => loop_end
    );
```

# Παράδειγμα 3: Λήψη διέγερσης και σύγκριση με διανύσματα αναφοράς από αρχείο (6)

Αρχείο testbench για το κύκλωμα (συνέχεια - 1)

```
RW_VECTORS: process
  variable line_in : line;
  file input_vec_file : text open read_mode is "hwloop_ref.vec";
  variable errordet         : boolean;
  variable initial_iv       : std_logic_vector(DW-1 downto 0);
  variable step_iv          : std_logic_vector(DW-1 downto 0);
  variable final_iv         : std_logic_vector(DW-1 downto 0);
  variable current_index_iv : std_logic_vector(DW-1 downto 0);
  variable next_index_ov    : std_logic_vector(DW-1 downto 0);
  variable loop_end_ov      : std_logic;
  variable initial_str      : string(1 to 5) := "init=";
  variable step_str         : string(1 to 5) := "step=";
  variable final_str        : string(1 to 5) := "finl=";
  variable current_index_str: string(1 to 8) := "curr=";
  variable next_index_str   : string(1 to 8) := "next=";
  variable loop_end_str     : string(1 to 9) := "loop_end=";
  variable scolon_str       : character      := ';';
begin
  while not (endfile(input_vec_file)) loop
    readline(input_vec_file, line_in);
    read(line_in,initial_str);
    read(line_in,initial_iv);
    read(line_in,scolon_str);
    read(line_in,step_str);
    read(line_in,step_iv);
    read(line_in,scolon_str);
```

# Παράδειγμα 3: Λήψη διέγερσης και σύγκριση με διανύσματα αναφοράς από αρχείο (7)

Αρχείο testbench για το κύκλωμα (συνέχεια - 2)

```
      read(line_in,final_str);
      read(line_in,final_iv);
      read(line_in,scolon_str);
      read(line_in,current_index_str);
      read(line_in,current_index_iv);
      read(line_in,scolon_str);
      read(line_in,next_index_str);
      read(line_in,next_index_ov);
      read(line_in,scolon_str);
      read(line_in,loop_end_str);
      read(line_in,loop_end_ov);
      read(line_in,scolon_str);
      -- assign input signals from the file vectors
      initial <= initial_iv; step <= step_iv; final <= final_iv;
      current_index <= current_index_iv;
      wait for CLK_PERIOD/2;
      -- Checking actual results with expected results
      if (next_index /= next_index_ov) then
        assert errordet REPORT "next_index: vector mismatch";
      end if;
      if (loop_end /= loop_end_ov) then
        assert errordet REPORT "loop_end: vector mismatch";
      end if;
      wait for CLK_PERIOD/2;
    end loop;
    wait;
  end process RW_VECTORS;
```

# Παράδειγμα 3: Λήψη διέγερσης και σύγκριση με διανύσματα αναφοράς από αρχείο (8)

- Το αρχείο με τα διανύσματα αναφοράς (`hwloop_ref.vec`)

```
init=00000000;step=00000000;finl=00000000;curr=00000000;next=00000000;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00000000;next=00000001;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00000001;next=00000010;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00000010;next=00000011;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00000011;next=00000100;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00000100;next=00000101;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00000101;next=00000110;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00000110;next=00000111;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00000111;next=00001000;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00001000;next=00001001;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00001001;next=00001010;loop_end=0;
init=00000000;step=00000001;finl=00001010;curr=00001010;next=00000000;loop_end=1;
init=00000000;step=00000000;finl=00000000;curr=00000000;next=00000000;loop_end=0;
init=00000001;step=00000010;finl=00000101;curr=00000001;next=00000011;loop_end=0;
init=00000001;step=00000010;finl=00000101;curr=00000011;next=00000101;loop_end=0;
init=00000001;step=00000010;finl=00000101;curr=00000101;next=00000001;loop_end=1;
```