Γλώσσες Περιγραφής Υλικού

Σύνταξη παραμετρικών περιγραφών

Νικόλαος Καββαδίας
nkavv@physics.auth.gr

7 Απριλίου 2009

Σκιαγράφηση της διάλεξης

- Σύνταξη παραμετρικών περιγραφών
  - Βιβλιοθήκες και πακέτα (libraries and packages)
  - Υποπρογράμματα (subprograms)
    - Συναρτήσεις (functions)
    - Διαδικασίες (procedures)
  - Η εντολή GENERATE
    - Σχήμα FOR ... GENERATE
    - Σχήμα IF ... GENERATE
  - Παραδείγματα σχεδιασμού παραμετρικών κυκλωμάτων:
    1. ολισθητής διανύσματος
    2. αθροιστής ριπής κρατουμένου (ripple-carry adder)
    3. γεννήτρια ισοτιμίας (parity generator)
    4. ανιχνευτής ισοτιμίας (parity detector)
    5. γενικευμένος πολυπλέκτης n-σε-1
    6. απλός διάδρομος δεδομένων

Η έννοια του πακέτου (PACKAGE) στη VHDL

- Το πακέτο στη VHDL αποτελεί ένα μέσο για την οργάνωση τμημάτων κώδικα τα οποία είναι χρήσιμο να είναι διαθέσιμα σε περισσότερα του ενός κυκλώματα
- Σε ένα πακέτο συνηθίζεται να τοποθετούνται συχνά χρησιμοποιούμενα τμήματα κώδικα:
  - Συστατικά κυκλωμάτων ώστε να είναι ορατά από άλλα κυκλώματα χωρίς την συμπερίληψη της δήλωσής τους στην περιοχή δηλώσεων των ARCHITECTURE
  - Συναρτήσεις και διαδικασίες
  - Ορισμοί τύπων και υποτύπων δεδομένων
  - Σταθερές
  - Σήματα καθολικής εμβέλειας αλλά όχι για χρήση σε συνθέσιμο κώδικα
- Ένα πακέτο μεταγλωττίζεται στην αντίστοιχή του βιβλιοθήκη (LIBRARY)

Σύνταξη και χρήση ενός πακέτου

- Για ένα πακέτο προαιρετικά μπορεί να συνταχθεί σώμα πακέτου (PACKAGE BODY) στο οποίο καταγράφονται οι περιγραφές τυχόν συναρτήσεων και διαδικασιών που έχουν δηλωθεί στην περιοχή δηλώσεων (statements) του πακέτου
- Σύνταξη ενός πακέτου

```
PACKAGE <package name> IS
  statements
END <package name>;

[PACKAGE BODY <package name> IS
  implementation of functions and procedures
END <package name>;]
```

- Δήλωση χρήσης ενός πακέτου

```
USE <library name>.<package name>.<package parts>;
```

Παράδειγμα περιγραφής ενός πακέτου

- Πακέτο με δηλώσεις τύπων, σταθεράς και δήλωση και περιγραφή συνάρτησης

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE my_package IS
  TYPE state IS (st1, st2, st3, st4);
  TYPE color IS (red, green, blue);
  CONSTANT all_ones : STD_LOGIC_VECTOR(7 downto 0) := "11111111";
  FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN;
END my_package;

PACKAGE BODY my_package IS
  FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN BOOLEAN IS
  BEGIN
    RETURN (s'EVENT and s='1');
  END positive_edge;
END my_package;
```


Περισσότερα για τη δήλωση χρήσης (USE) ενός πακέτου

- Τα πακέτα πρέπει να είναι ορατά στην εμβέλεια του κώδικα όπου πρόκειται να χρησιμοποιηθούν κάποια από τα περιεχόμενά τους
  - Με την εντολή USE δηλώνεται η διαθεσιμότητα ενός πακέτου σε κάποια ENTITY, ARCHITECTURE ή ακόμη και σε άλλα πακέτα που βασίζονται σε αυτό

```
-- all the package contents are accessible
USE work.my_package.all;
-- only the function named "positive_edge" is accessible
USE work.my_package.positive_edge;
```


Υποπρογράμματα

- Τα υποπρογράμματα στη VHDL χρησιμοποποιούνται με παρόμοιο τρόπο σε σχέση με τις συμβατικές γλώσσες διαδικαστικού προγραμματισμού
- Επιτρέπουν τη χρήση επαναλαμβανόμενων τμημάτων κώδικα χωρίς να χρειάζεται αυτά να επαναγραφούν
- Με τη χρήση υποπρογραμμάτων, εκτεταμένες δομές κώδικα διαιρούνται σε μικρότερα τμήματα κώδικα τα οποία είναι πιο εύκολα στη διαχείρισή τους (δομημένος προγραμματισμός)
- Η VHDL παρέχει συναρτήσεις (functions) και διαδικασίες (procedures)

ΣΥΝΑΡΤΗΣΗ (FUNCTION)

- Οι συναρτήσεις αποτελούν είδος υποπρογράμματος το οποίο επιστρέφει (όταν καλείται) μία μοναδική τιμή
- Οι συναρτήσεις καλούνται από εκφράσεις
- Δεν μπορούν να τροποποιήσουν τις παραμέτρους εισόδου τους
- Η χρήση εντολής RETURN για την επιστροφή τιμής είναι υποχρεωτική
- Σύνταξη:

```
FUNCTION <function name> [<parameter list>] RETURN <type> IS
  [statements]
BEGIN
  sequential statements
END <function name>;
```

- Η λίστα παραμέτρων (parameter list) μπορεί να περιλαμβάνει:

```
[CONSTANT] <constant name> : <constant type>;
or
SIGNAL <signal name> : <signal type>;
```

Παραδείγματα συναρτήσεων (1)

- Παράδειγμα: Συνάρτηση `conv_integer`

```
FUNCTION conv_integer(SIGNAL vector: STD_LOGIC_VECTOR) RETURN INTEGER IS
  VARIABLE result: INTEGER RANGE 0 to 2**vector'LENGTH-1;
BEGIN
  IF (vector(vector'HIGH)='1') THEN
    result := 1;
  ELSE
    result := 0;
  END IF;
  FOR i IN (vector'HIGH-1) DOWNTO (vector'LOW) LOOP
    result := result * 2;
    IF (vector(i)='1') THEN
      result := result + 1;
   END IF;
  END LOOP;
  RETURN result;
END conv_integer;
...
SIGNAL y: INTEGER;
SIGNAL a: STD_LOGIC_VECTOR(7 downto 0);
...
y <= conv_integer(a);
```


Παραδείγματα συναρτήσεων (2)

- Παράδειγμα: Συνάρτηση $\lceil log_2 \rceil$
- Μια συνάρτηση λογαρίθμου ως προς 2 τύπου ceiling (στρογγυλοποίηση αποτελέσματος στον πλησιέστερο μεγαλύτερο ακέραιο) είναι ιδιαίτερα χρήσιμη για την περιγραφή εύρους bit διευθύνσεων σε σχέση με το συνολικό αριθμό των διευθυνσιοδοτούμενων θέσεων, με μία μόνο παράμετρο
- Παραδείγματα χρήσης: αποκωδικοποιητές, κωδικοποιητές, γενικευμένοι πολυπλέκτες

```
function LOG2C(input: INTEGER) return INTEGER is
  variable temp,log: INTEGER;
  begin
    log := 0;
    temp := 1;
    for i in 0 to input loop
      if temp < input then
        log := log + 1;
        temp := temp * 2;
      end if;
    end loop;
    return (log);
end function LOG2C;
```


ΔΙΑΔΙΚΑΣΙΑ (PROCEDURE)

- Συντάσσονται με παρόμοιο τρόπο με τις ΣΥΝΑΡΤΗΣΕΙΣ
- Καλούνται από δηλώσεις στον κώδικα (statements)
- Μια διαδικασία μπορεί να χρησιμοποιηθεί ως εντολή από μόνη της
- Επιτρέπεται να τροποποιούν τις τιμές των παραμέτρων τους
- Δεν απαιτείται να περιλαμβάνουν εντολή RETURN
- Οι παράμετροι μπορεί να έχουν κατευθυντικότητα (IN, OUT, INOUT) και μπορεί να είναι οποιαδήποτε αντικείμενα (SIGNAL, VARIABLE, CONSTANT)
- Σύνταξη:

```
PROCEDURE <procedure name> [<parameter list>] IS
  [statements]
BEGIN
  sequential statements
END <procedure name>;
```

- Η λίστα παραμέτρων (parameter list) μπορεί να περιλαμβάνει:

```
[CONSTANT] <constant name> : <constant type>;
SIGNAL <signal name> : <direction> <signal type>;
VARIABLE <variable name> : <direction> <variable type>;
```


Παραδείγματα διαδικασιών

- Παράδειγμα: Διαδικασία `sort`

```
PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0 to limit;
  SIGNAL min, max: OUT INTEGER RANGE 0 to limit) IS
BEGIN
  IF (in1 >= in2) THEN
    max <= in1;
    min <= in2;
  ELSE
    max <= in2;
    min <= in1;
  END IF;
END sort;
...
PROCESS (en)
BEGIN
  IF (en = '1') THEN
    sort (inp1, inp2, outp1, outp2);
  END IF;
END PROCESS;
...
```

Πακέτο για αριθμητική με μιγαδικούς αριθμούς (complex number arithmetic): Δήλωση PACKAGE

```
PACKAGE complex_type IS
  CONSTANT re : INTEGER := 0;
  CONSTANT im : INTEGER := 1;
  TYPE complex IS ARRAY (NATURAL RANGE re TO im) OF REAL;
  FUNCTION "+" (a, b : complex) RETURN complex;
  FUNCTION "-" (a, b : complex) RETURN complex;
  FUNCTION "*" (a, b : complex) RETURN complex;
  FUNCTION "/" (a, b : complex) RETURN complex;
END complex_type;
```


Πακέτο για αριθμητική με μιγαδικούς (complex number arithmetic): Τελεστές '+' και '-'

```
PACKAGE BODY complex_type IS

  FUNCTION "+" (a, b : complex) RETURN complex IS
    VARIABLE t : complex;
    BEGIN
      T(re) := a(re) + b(re);
      T(im) := a(im) + b(im);
      RETURN t;
  END "+";

  FUNCTION "-" (a, b : complex) RETURN complex IS
    VARIABLE t : complex;
    BEGIN
      T(re) := a(re) - b(re);
      T(im) := a(im) - b(im);
      RETURN t;
  END "-";
```


Πακέτο για αριθμητική με μιγαδικούς (complex number arithmetic): Τελεστές '*' και '/'

```
  FUNCTION "*" (a, b : complex) RETURN complex IS
    VARIABLE t : complex;
    BEGIN
      t(re) := a(re) * b(re) - a(im) * b(im);
      t(im) := a(re) * b(im) + b(re) * a(im);
      RETURN t;
  END "*";

  FUNCTION "/" (a, b : complex) RETURN complex IS
    VARIABLE i : real;
    VARIABLE t : complex;
    BEGIN
      t(re) := a(re) * b(re) + a(im) * b(im);
      t(im) := b(re) * a(im) - a(re) * b(im);
      i := b(re)**2 + b(im)**2;
      t(re) := t(re) / i;
      t(im) := t(im) / i;
      RETURN t;
  END "/";

END complex_type;
```


Η εντολή GENERATE

- Η εντολή GENERATE παρέχει τη δυνατότητα περιγραφής επαναλαμβανόμενων κυκλωματικών δομών με συμπαγή τρόπο καταλαμβάνοντας μικρή έκταση κώδικα
- Διευκολύνει την περιγραφή δομών που παρουσιάζουν κανονικότητα στο υλικό όπως αθροιστές ριπής κρατουμένου, αρχεία καταχωρητών (register files), και δένδρα πυλών XOR (δομή σύγκρισης)
- Οποιαδήποτε εντολή που επιτρέπεται σε συντρέχοντα κώδικα στη VHDL μπορεί να συμπεριληφθεί σε μια εντολή GENERATE
- Είναι εφικτή η περιγραφή φωλιασμένων δομών από GENERATE
- Ειδικότερα, μια εντολή GENERATE μπορεί να περικλείει στιγμιότυπα συστατικών
- Μια εντολή GENERATE μπορεί να εκφραστεί με δύο διαφορετικά σχήματα
  - Σχήμα FOR (FOR ... GENERATE)
  - Σχήμα IF (IF ... GENERATE)

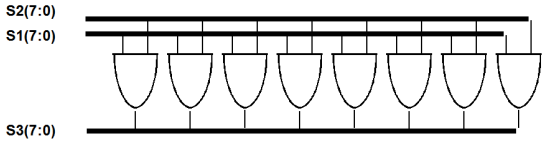
Το σχήμα FOR της εντολής GENERATE

- Για το σχήμα FOR της εντολής GENERATE
  - Η παράμετρος δεικτοδότησης του FOR δεν πρέπει να χρησιμοποιείται εκτός της FOR ... GENERATE
  - Ο βρόχος δεν τερματίζεται πρόωρα
  - Σύνταξη:

```
<label> : FOR parameter_specification GENERATE
  [declaration_statements]
BEGIN
  concurrent_statements
END GENERATE <label>;
```


Παράδειγμα απλού κυκλώματος με FOR ... GENERATE

- Το απλό κύκλωμα του σχήματος αποτελεί ένα διάνυσμα από πύλες AND 2 εισόδων στο οποίο εφαρμόζονται δύο είσοδοι τύπου STD_LOGIC_VECTOR των 8-bit
- Σχηματικό διάγραμμα



- Αρχιτεκτονική του κυκλώματος

```
ARCHITECTURE test_forgen OF test IS
  SIGNAL S1, S2, S3: BIT_VECTOR(7 DOWNTO 0);
BEGIN
  G1 : FOR N IN 7 DOWNTO 0 GENERATE
    and_array : and_gate
      PORT MAP (
        in1  => S1(N),
        in2  => S2(N),
        out1 => S3(N)
      );
  END GENERATE G1;
END test_forgen;
```


Το σχήμα IF της εντολής GENERATE

- Για το σχήμα IF της εντολής GENERATE
  - Επιτρέπει την υπό συνθήκη δημιουργία υλικού
  - Δεν επιτρέπεται η χρήση των ELSE και ELSIF τμημάτων μιας εντολής IF
  - Σύνταξη:

```
<label> : IF parameter_specification GENERATE
  [declaration_statements]
BEGIN
  concurrent_statements
END GENERATE <label>;
```


Παράδειγμα απλού κυκλώματος με IF ... GENERATE

- Στο παράδειγμα αυτό η εντολή IF ... GENERATE χρησιμοποιείται ένθετη στην εξωτερική FOR ... GENERATE
- Σε σχέση με το αρχικό κύκλωμα, η πύλη AND για τα ψηφία υψηλότερης σημαντικότητας (MSB) αντικαθίσταται από μία XOR
- Αρχιτεκτονική του κυκλώματος

```
ARCHITECTURE test_ifgen OF test
  SIGNAL S1, S2, S3: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
  G1 : FOR N IN 7 DOWNTO 0 GENERATE
    G2 : IF (N = 7) GENERATE
      S3(N) <= S1(N) xor S2(N);
    END GENERATE G2;

    G3 : IF (N < 7) GENERATE
      S3(N) <= S1(N) and S2(N);
    END GENERATE G3;
  END GENERATE G1;
END test_ifgen;
```

Κυκλώματα με παραμετρικές περιγραφές

1. Ολισθητής διανύσματος
2. Αθροιστής ριπής κρατουμένου (ripple-carry adder)
3. Γεννήτρια ισοτιμίας (parity generator)
4. Ανιχνευτής ισοτιμίας (parity detector)
5. Γενικευμένος πολυπλέκτης n-σε-1
6. Απλός διάδρομος δεδομένων

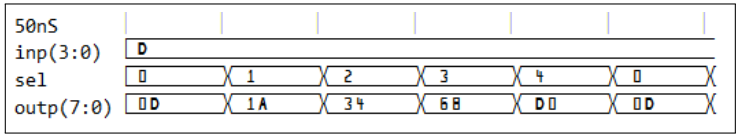
Ολισθητής διανύσματος για αριστερή λογική ολίσθηση (1)

- Κύκλωμα ολίσθησης του διανύσματος εισόδου `inp` κατά έναν αριθμό θέσεων
- Ο αριθμός θέσεων της ολίσθησης καθορίζεται από την είσοδο `sel` που είναι τύπου INTEGER

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

entity vshifter is
  port (
    inp  : in std_logic_vector(3 downto 0);
    sel  : in integer range 0 to 4;
    outp : out std_logic_vector(7 downto 0)
  );
end vshifter;

architecture leftshift of vshifter is
  subtype vector is std_logic_vector(7 downto 0);
  type matrix is array (4 downto 0) of vector;
  signal rows: matrix;
begin
  rows(0) <= "0000" & inp;
  G1: for i in 1 to 4 generate
    rows(i) <= rows(i-1)(6 downto 0) & '0';
  end generate G1;
  outp <= rows(sel);
end leftshift;
```


Ολισθητής διανύσματος για αριστερή λογική ολίσθηση (2)

- Διάγραμμα χρονισμού:



Παραμετρικός αθροιστής ριπής κρατουμένου (parameterized ripple-carry adder)

- Η γενική σταθερή N καθορίζει το εύρος bit του αθροιστή
- Για κάθε m στην εντολή GENERATE δημιουργείται ένας πλήρης αθροιστής με διασύνδεση του κρατουμένου εξόδου του στο κρατούμενο εισόδου της επόμενης βαθμίδας

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

entity rca is
  generic (N: integer := 8);
  port (
    a, b : in unsigned(N-1 downto 0);
    cin : in std_logic;
    sum : out unsigned(N-1 downto 0);
    cout : out std_logic
  );
end rca;
```

```
architecture gatelevel of rca is
  signal c : unsigned(N downto 0);
begin
  c(0) <= cin;
  G1: for m in 0 to N-1 generate
    sum(m) <= a(m) xor b(m) xor c(m);
    c(m+1) <= (a(m) and b(m)) or
              (b(m) and c(m)) or
              (a(m) and c(m));
  end generate G1;
  cout <= c(N);
end gatelevel;
```

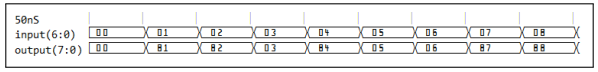
Γενική γεννήτρια ισοτιμίας (parity generator)

Το parity bit είναι `'0'` όταν το πλήθος των `'1'` στη λέξη είναι άρτιο ή `'1'` στην περίπτωση που το πλήθος είναι περιττό, ώστε το προκύπτον διάνυσμα από τη συνένωση εισόδου και ψηφίου ισοτιμίας να έχει πάντα άρτια ισοτιμία

```
entity parity_gen is
  generic (n: integer := 7);
  port (
    input
: in bit_vector(n-1 downto 0);
    output : out bit_vector(n downto 0)
  );
end parity_gen;

architecture rtl of parity_gen is
begin
```

```
  process (input)
  variable temp1: bit;
  variable temp2: bit_vector(n downto 0);
  begin
    temp1 := '0';
    for i in input'RANGE loop
      temp1 := temp1 xor input(i);
      temp2(i) := input(i);
    end loop;
    temp2(n) := temp1;
    output <= temp2;
  end process;
end rtl;
```

Διάγραμμα χρονισμού του κυκλώματος



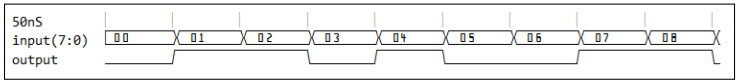
Γενικός ανιχνευτής ισοτιμίας (parity detector)

Ένας ελεγκτής ισοτιμίας επιστρέφει `'0'` στην έξοδο όταν το πλήθος των `'1'` στο διάνυσμα εισόδου είναι άρτιο, και `'1'` όταν είναι περιττό

```
entity parity_det is
  generic (n : integer := 7);
  port (
    input: in bit_vector(n downto 0);
    output: out bit
  );
end parity_det;
```

```
architecture rtl of parity_det is
begin
  process (input)
  variable temp: bit;
  begin
    temp := '0';
    for i in input'RANGE loop
      temp := temp xor input(i);
    end loop;
    output <= temp;
  end process;
end rtl;
```

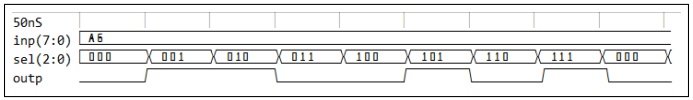
Διάγραμμα χρονισμού του κυκλώματος



Περιγραφή γενικευμένου πολυπλέκτη

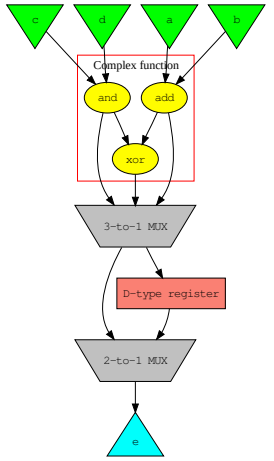
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.various_pkg.all; -- the package where "log2c(<integer>) lives

entity muxnto1 is
  generic (N : integer := 8);
  port (
    inp   : in std_logic_vector(N-1 downto 0);
    sel   : in std_logic_vector(log2c(N)-1 downto 0);
    outp  : out std_logic
  );
end muxnto1;

architecture rtl of muxnto1 is
begin
  outp <= inp(to_integer(unsigned(sel)));
end rtl;
```

Διάγραμμα χρονισμού του κυκλώματος



Υλοποίηση απλού διαδρόμου δεδομένων (1)

- Έστω ότι θέλουμε να υλοποιήσουμε διάδρομο δεδομένων (datapath) για τον υπολογισμό της έκφρασης `(a + b) XOR (c AND d)`
- Στην έξοδο του κυκλώματος θα βγαίνει το αποτέλεσμα της συνολικής έκφρασης, τα επιμέρους αποτελέσματα `a + b` και `c AND d` και αυτά θα είναι διαθέσιμα είτε άμεσα είτε μέσω καταχωρητή
- Σχηματικό διάγραμμα



Υλοποίηση απλού διαδρόμου δεδομένων (2)

- Το κύκλωμα του `cplxfunc.vhd`

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

entity cplxfunc is
  port (
    a, b, c, d : in std_logic_vector(7 downto 0);
    o1, o2, o3 : out std_logic_vector(7 downto 0)
  );
end cplxfunc;

architecture rtl of cplxfunc is
  signal t1, t3: unsigned(7 downto 0);
begin
  process (a,b,c,d,t1,t3)
  begin
    t3 <= unsigned(c) and unsigned(d);
    t1 <= unsigned(a) + unsigned(b);
    o2 <= std_logic_vector(unsigned(t1) + unsigned(t3));
    o1 <= std_logic_vector(t1);
    o3 <= std_logic_vector(t3);
  end process;
end rtl;
```


Υλοποίηση απλού διαδρόμου δεδομένων (3)

Καταχωρητής

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

entity regn is
  generic (
    N : integer := 16
  );
  port (
    clk : in std_logic;
    reset : in std_logic;
    d : in std_logic_vector(N-1 downto 0);
    q : out std_logic_vector(N-1 downto 0)
  );
end regn;
```

```
architecture rtl of regn is
signal temp:
  std_logic_vector(N-1 downto 0);
begin
  process (clk, reset, d)
  begin
    if (reset = '1') then
      temp <= (others => '0');
    elsif (clk'EVENT and clk='1') then
      temp <= d;
    end if;
  end process;
  q <= temp;
end rtl;
```


Υλοποίηση απλού διαδρόμου δεδομένων (4)

- Το συνολικό κύκλωμα

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

entity datapath is
  port (
    clk, reset : in std_logic;
    in1, in2 : in std_logic_vector(7 downto 0);
    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 datapath;

architecture structural of datapath is
  component cplxfunc is
    port (
      a, b : in std_logic_vector(7 downto 0);
      c, d : in std_logic_vector(7 downto 0);
      o1, o2, o3 : out std_logic_vector(7 downto 0)
    );
  end component;
```


Υλοποίηση απλού διαδρόμου δεδομένων (5)

- Το συνολικό κύκλωμα (συνέχεια - 1)

```
  component regn is
    generic (
      N : integer := 16
    );
    port (
      clk   : in std_logic;
      reset : in std_logic;
      d     : in std_logic_vector(N-1 downto 0);
      q     : out std_logic_vector(N-1 downto 0)
    );
  end component;
  --
  signal x1, x2, x3, x4, x4_r: std_logic_vector(7 downto 0);
begin

  uut1 : cplxfunc
    port map (
      a => in1, b => in2,
      c => in3, d => in4,
      o1 => x1, o2 => x2, o3 => x3
    );
```

Υλοποίηση απλού διαδρόμου δεδομένων (6)

- Το συνολικό κύκλωμα (συνέχεια - 2)

```
G_mux1 : for i in 7 downto 0 generate
  with sel(1 downto 0) select
    x4(i) <= x3(i) when "10",
             x2(i) when "01",
             x1(i) when "00",
             X"00" when others;
end generate G_mux1;

uut3 : regn
  generic map (
    N => 8
  )
```

```
  port map (
    clk   => clk,
    reset => reset,
    d     => x4,
    q     => x4_r
  );

G_mux2 : for i in 7 downto 0 generate
  with sel(2 downto 2) select
    outp(i) <= x4_r(i) when "1",
               x4(i) when others;
end generate G_mux2;
```

- Διάγραμμα χρονισμού του κυκλώματος