Γλώσσες Περιγραφής Υλικού

Υποδειγματική εργασία

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

19 Μαΐου 2009

Σκιαγράφηση της διάλεξης

- Παρουσίαση υποδειγματικής εργασίας
  - Γενικός οδηγός και συμβουλές για την πραγματοποίηση της εξαμηνιαίας εργασίας για το μάθημα
  - Κατανόηση του ζητούμενου - Εξαγωγή προδιαγραφών
  - Ανάπτυξη του κυκλώματος από προδιαγραφές
  - Σχεδιασμός του κυκλώματος
  - Προσομοίωση και έλεγχος ορθής λειτουργίας
  - Οργάνωση της αναφοράς για την εργασία

Αντικείμενο μιας εργασίας - Τιτλοδότηση

- Τίτλος της εργασίας:
  - Ένας τίτλος είναι επαρκώς περιγραφικός, όσο το δυνατόν πιο σύντομος και δεν παραπλανεί, π.χ.:
    Σχεδιασμός μιας μονάδας ελέγχου φωλιασμένων βρόχων με μηδενική καθυστέρηση
- Ανάλυση της πληροφορίας του τίτλου του παραδείγματος
  - *Σχεδιασμός*: η εργασία περιγράφει το σχέδιο ενός κυκλώματος
  - *μονάδας ελέγχου φωλιασμένων βρόχων*: το είδος του κυκλώματος και σε τι εξυπηρετεί η λειτουργία του
  - *μηδενική καθυστέρηση*: ιδιαίτερο χαρακτηριστικό-ιδιότητα του κυκλώματος που σχεδιάστηκε
- Ανάπτυξη και σχεδιασμός
  - Ανάπτυξη: μέρος της εργασίας είναι και ο προσδιορισμός των προδιαγραφών λειτουργίας του κυκλώματος
  - Σχεδιασμός: ζητούμενο είναι μόνο η αναπαραγωγή του κυκλώματος από αναλυτική περιγραφή με κάποιον τρόπο

Ανάπτυξη προδιαγραφών του κυκλώματος

- Πότε η ανάπτυξη των προδιαγραφών λειτουργίας ενός κυκλώματος είναι απαραίτητη
  - Όταν δεν δίνεται σχηματικό διάγραμμα του κυκλώματος
  - Όταν δεν δίνεται διάγραμμα αλγοριθμικής ροής
- Πολλές φορές ζητείται να εξαχθούν από αλγοριθμική περιγραφή υπό μορφή πηγαίου κώδικα σε διαδικαστική γλώσσα προγραμματισμού (π.χ. ANSI C)
- Προδιαγραφές ενός κυκλώματος
  - Είσοδος και έξοδος: διεπαφή (interface)
  - Προσδοκώμενο αποτέλεσμα
  - Απαιτήσεις ταχύτητας επεξεργασίας ή χρονισμού (μέγιστος χρόνος για τη διεκπεραίωση της επίλυσης του προβλήματος)
  - Απαίτηση για μέγιστη επιφάνεια υλικού (π.χ. η συνολική επιφάνεια ενός διαθέσιμου FPGA)
  - Εκτελέσιμη περιγραφή του κυκλώματος (για προσομοίωση)
  - Τεκμηρίωση του κατασκευαστή (datasheet)

Υποδειγματική εργασία: Εισαγωγή (1)

- Τίτλος
  - Σχεδιασμός μιας μονάδας ελέγχου φωλιασμένων βρόχων με μηδενική καθυστέρηση
- Υπόβαθρο του προβλήματος
  - Βρόχος (loop): ακολουθία εντολών (σε επίπεδο λογισμικού ενός μικροεπεξεργαστή) ή μικρολειτουργιών οι οποίες εμπεριέχονται σε μία δομή επανάληψης
  - Οι επαναλήψεις καθορίζονται από τρεις παραμέτρους: την αρχική τιμή (initial), την τελική τιμή (final) και την τιμή βήματος (step)
  - Η τρέχουσα επανάληψη καθορίζεται από το δείκτη βρόχου (loop index)
  - Δύο ή περισσότεροι βρόχοι μπορεί να είναι φωλιασμένοι (ο ένας να περικλείει πλήρως τον άλλο)

Υποδειγματική εργασία: Εισαγωγή (2)

- Υπόβαθρο του προβλήματος (συνέχεια)
  - Για την εκτέλεση ενός βρόχου σε έναν μικροεπεξεργαστή, χρειάζονται οι ακόλουθες λειτουργίες:
    1. Εκτέλεση όλων των χρήσιμων υπολογιστικά λειτουργιών (περικλειόμενες εντολές στη δομή επανάληψης)
    2. Αύξηση του δείκτη για τη δεικτοδότηση της επόμενης επανάληψης
    3. Σύγκριση του δείκτη με την παράμετρο final
    4. Άλμα στην αρχή του βρόχου εφόσον αυτή δεν είναι η τελευταία επανάληψη

Περιγραφές βρόχων

- Βρόχος FOR στην C

```
for (i=0; i<10; i++) {
  /* statements */ }
```

- Κώδικας συμβολομεταφραστή (για έναν τυπικό RISC) για την περίπτωση του απλού βρόχου

- Περίπτωση 1

```
  MOVE   R1, #0
loop_start:
  ...
  ADD    R1, R1, #1
  SEQI   R2, R1, #10
  BEQZ   R2, loop_start
```

- Περίπτωση 2

```
  MOVE   R1, #0
  MOVE   R2, #10
loop_start:
  ...
  INC    R1
  JEQ    R1, R2, loop_start
```

- Διπλά φωλιασμένοι βρόχοι FOR στην C

```
for (i=0; i<10; i++) {
  for (j=1; j<8; j+=2) {
  /* statements */ } }
```

Γενική περίπτωση των πλήρως φωλιασμένων βρόχων

- Πλήρως φωλιασμένοι βρόχοι: δομή φωλιασμένων βρόχων με χρήσιμες υπολογιστικά λειτουργίες (statements) μόνο στον εσώτερο βρόχο (inner loop)
- Κώδικας C για n πλήρως φωλιασμένους βρόχους

```
for (index1=initial1; index1<=final1; index1+=step1) {
  for (index2=initial2; index2<=final2; index2+=step2) {
  ...
    for (indexn=initialn; indexn<=finaln; indexn+=stepn) {
      statement1;
      statement2;
      ...
      statementm;
    }
  }
}
```

Ζητούμενο της εργασίας

- Να σχεδιαστεί μονάδα ελέγχου πλήρως φωλιασμένων βρόχων με συγκεκριμένα χαρακτηριστικά
- Χαρακτηριστικά
  - Προϋποθέσεις
    - Η αποθήκευση των παραμέτρων βρόχου (αρχική, τελική και τιμή βήματος) γίνονται εξωτερικά της μονάδας
    - Οι τιμές των παραμέτρων βρόχου μπορεί να μεταβάλλονται μόνον εκτός της δομής βρόχων
  - Περιορισμοί
    - Η μεταβολή ενός δείκτη μπορεί να γίνει μόνο με την πρόσθεση ενός βήματος και όχι με άλλη μαθηματική έκφραση
    - Η αρχική τιμή δείκτη είναι ίση με μηδέν
  - Δυνατότητες
    - Να μπορεί να σχεδιαστεί κύκλωμα για οποιοδήποτε n (συνολικός αριθμός βρόχων) ζητηθεί
    - Με την ολοκλήρωση της τελικής επανάληψης του εσώτερου βρόχου, να γίνεται ταυτόχρονη επαναρχικοποίηση όλων των βρόχων

Εισαγωγή στο πρόβλημα

- Εισαγωγή για το αντικείμενο της εργασίας και προαιρετικά σύνοψη της εργασίας

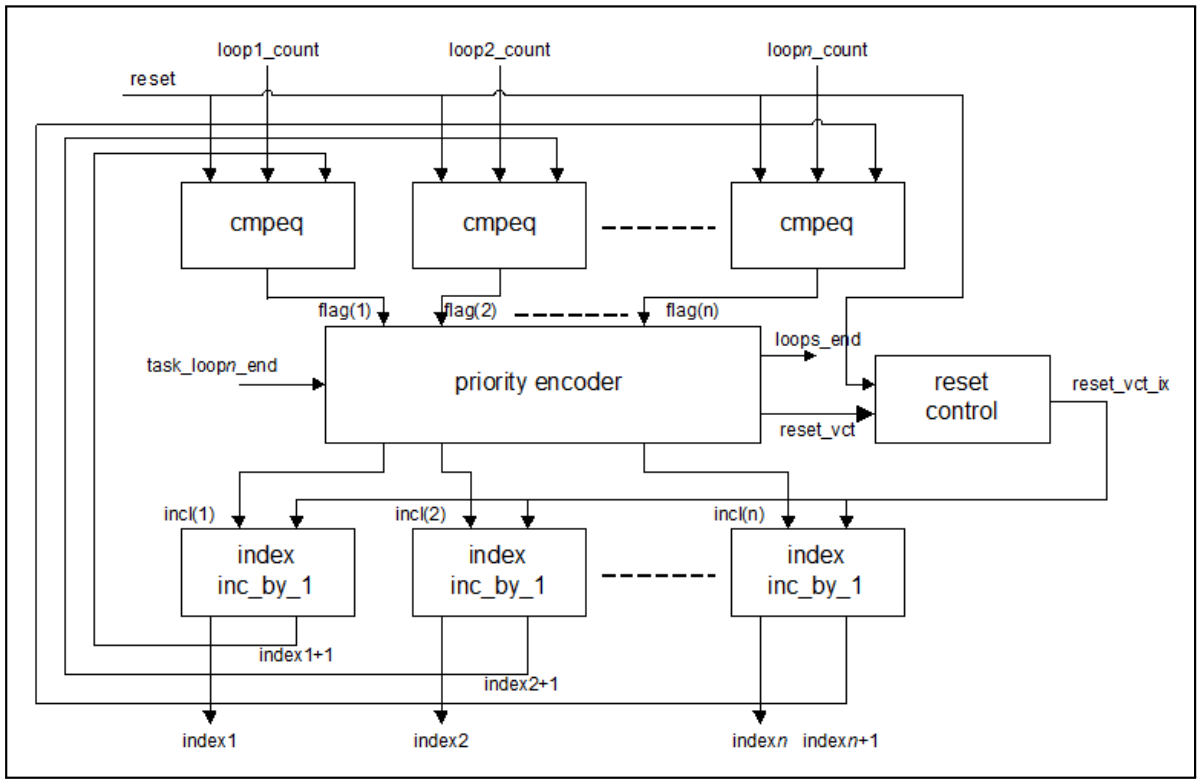
Στην εργασία αυτή, περιγράφεται [η ανάπτυξη και] ο σχεδιασμός μιας μονάδας για την εκτέλεση πλήρως φωλιασμένων βρόχων στο υλικό. Η μονάδα η οποία αναπτύχθηκε ονομάζεται HWLU: Hardware Looping Unit και ο σχεδιασμός της έγινε στη γλώσσα περιγραφής υλικού VHDL. Κύρια χρήση της μονάδας αυτής είναι η ενίσχυση της λογικής ελέγχου δομών βρόχων σε μη προγραμματιζόμενους επεξεργαστές, με την υλοποίηση των λειτουργιών βρόχου (δεικτοδότηση και έλεγχος της διαδοχής των επαναλήψεων) στο υλικό.

Ένα πλεονέκτημα της μονάδας είναι οι διαδοχικές τελικές επαναλήψεις σε μια δομή φωλιασμένων βρόχων μπορούν να εκτελεστούν σε ένα μόνο κύκλο μηχανής.

Ενδεικτικά, αναφέρεται ότι η μονάδα μπορεί να χρησιμοποιηθεί σε εφαρμογές πολυδιάστατης επεξεργασίας σήματος (multidimensional signal processing) όπως είναι η κωδικοποίηση στατικής εικόνας και η συμπίεση βίντεο κατά το πρότυπο MPEG-4.

Περιγραφή υψηλού επιπέδου του συνολικού κυκλώματος

- Σχηματικό διάγραμμα του συνολικού κυκλώματος (δομική περιγραφή σε VHDL) χωρίς λεπτομέρειες για την υλοποίηση των υπομονάδων
- Οι διασυνδέσεις των υπομονάδων είναι ορατές (SIGNAL στη VHDL)
- Ορισμένα 'κοινά' σήματα όπως τα clock, reset μπορούν να παραλειφθούν



Ανάλυση της αρχιτεκτονικής και της λειτουργίας του κυκλώματος (1)

- Η ανάλυση της λειτουργίας του κυκλώματος γίνεται σε σχετικά υψηλό επίπεδο, και έχοντας ως αναφορά συνήθως το σχηματικό διάγραμμα του κυκλώματος. Εναλλακτικά μπορεί να γίνει με βάση προδιαγραφή της λειτουργίας του κυκλώματος σε μορφή ψευδοκώδικα ή ANSI C αν και αυτός ο τρόπος δεν εξυπηρετεί κάποιον που έρχεται για πρώτη φορά σε επαφή με το πρόβλημα

Η μονάδα HWLU παρουσιάζεται στο σχήμα ...
Ο απαιτούμενος αριθμός επαναλήψεων για κάθε βρόχο (ίσος με την τελική τιμή μείον ένα ενώ είναι initial0) δίνεται από τις είσοδους loop<i>_count όπου <i> η απαρίθμηση του i-οστού βρόχου.
Οι τιμές των δεικτών βρόχου (index) παραγόνται σε κάθε κύκλο μηχανής. Κατά τον επόμενο κύκλο, της τελικής επανάληψης ενός βρόχου, ο δείκτης επιστρέφει στην αρχική παράμετρο.

Ανάλυση της αρχιτεκτονικής και της λειτουργίας του κυκλώματος (2)

- (συνέχεια)

Ο κωδικοποιητής προτεραιότητας (priority encoder) υλοποιεί ουσιαστικά τη λογική ελέγχου στο κύκλωμα και αποτελεί ένα συνδυαστικό κύκλωμα. Ο κωδικοποιητής προτεραιότητας ανιχνεύει τις εξόδους των συγκριτών ισότητας (cmpeq) οι οποίοι συγκρίνουν τις τιμές loop<i>_count με τις αντίστοιχες index<i> + 1 για τη διαπίστωση του αν πρόκειται για τελική επανάληψη του αντίστοιχου βρόχου ή όχι. Επίσης δέχεται μία είσοδο (task_loop<n>_end), όπου n είναι η απαρίθμηση του εσώτερου βρόχου, η οποία είναι ίση με ΄1΄ όταν ολοκληρώνεται και η εκτέλεση της τελευταίας χρήσιμης λειτουργίας στο βρόχο αυτό. Από τον κωδικοποιητή παράγονται η σημαία τερματισμού της δομής βρόχων (loops_end) και ένα εσωτερικό σήμα reset_vct_ix προς τις μονάδες αύξησης του δείκτη (index inc_by_1).

Με το σήμα reset_vct_ix όταν ένας συγκεκριμένος βρόχος τερματίζει, μηδενίζονται οι εσώτεροι του βρόχοι ενώ ο δείκτης του άμεσα εξώτερου βρόχου αυξάνεται κατά ένα.

Όταν η συνολική δομή βρόχου τερματιστεί το σήμα loops_end γίνεται ένα και στην περίπτωση που η μονάδα HWLU χρησιμοποιείται ως τμήμα ενός μεγαλύτερου συστήματος, ο έλεγχος επιστρέφει στο κυρίως σύστημα.

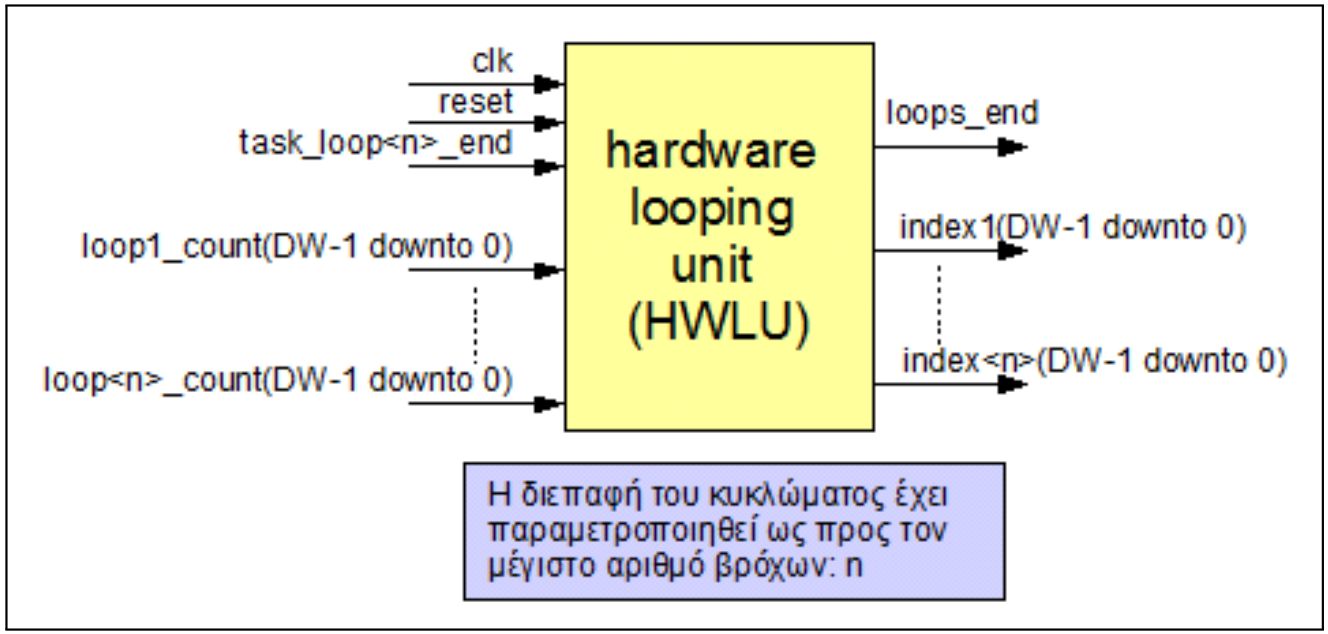
Διεπαφή του κυκλώματος (1)

- Η διεπαφή του κυκλώματος δίνεται με τη μορφή σχηματικού αλλά και ως πίνακας στον οποίο φαίνονται
  - Το εύρος των σημάτων
  - Η κατευθυντικότητα
  - Η περιγραφή του ρόλου τους
- Για το κύκλωμα HWLU:

Θύρα	**Εύρος**	**Κατευθ.**	**Περιγραφή**
`clk`	1	Είσοδος	Σήμα ρολογιού
`reset`	1	Είσοδος	Σήμα επανατοποθέτησης
`task_loop<n>_end`	1	Είσοδος	Σήμα τερματισμού των χρήσιμων υπολογισμών που συμβαίνουν κατά την τρέχουσα επανάληψη του εσώτερου βρόχου
`loop1_count` ... `loop<n>_count`	DW DW	Είσοδος Είσοδος	Αριθμός επαναλήψεων του βρόχου 1 Αριθμός επαναλήψεων του βρόχου n
`loops_end`	1	Έξοδος	Σημαία τερματισμού εκτέλεσης της δομής βρόχου
`index1` ... `index<n>`	DW DW	Έξοδος Έξοδος	Τρέχουσα τιμή δείκτη για το βρόχο 1 Τρέχουσα τιμή δείκτη για το βρόχο n

Διεπαφή του κυκλώματος (2)

- Σημαντικές παρατηρήσεις
  - Οι τιμές δείκτη λαμβάνονται από καταχωρητή
  - Η είσοδος επανατοποθέτησης (reset) χρησιμοποιείται ως σύγχρονη στις μονάδες αύξησης του δείκτη και ως ασύγχρονη για τη λειτουργία reset control
- Σχηματική αναπαράσταση της διεπαφής του κυκλώματος



Περιγραφή των υπομονάδων του κυκλώματος

- Οι υπομονάδες της HWLU περιγράφονται ξεχωριστά στις δικές τους υποενότητες
- Δίνεται σύντομη περιγραφή λειτουργίας, ένα διάγραμμα βαθμίδων (προαιρετικά) και ο πηγαίος κώδικας σε VHDL
- Παράδειγμα: ο συγκριτής ισότητας cmpeq

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

entity cmpeq is
  generic ( DW : integer := 8 );
  port (
    a, b   : in std_logic_vector(DW-1 downto 0);
    reset  : in std_logic;
    a_eq_b : out std_logic
  );
end cmpeq;

architecture rtl of cmpeq is
begin
  a_eq_b <= '1' when (a = b and reset = '0') else '0';
end rtl;
```

Μονάδα αύξησης του δείκτη

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

entity index_inc is
  generic (
    DW : integer := 8
  );
  port (
    clk             : in std_logic;
    reset           : in std_logic;
    inc_en          : in std_logic;
    index_plus_one, index_out : out
      std_logic_vector(Dw-1 downto 0)
  );
end index_inc;

architecture rtl of index_inc is
component add
...
end component;
component reg_dw
...
end component;
```

```
constant one_dw :
     std_logic_vector(DW-1 downto 0)
  := conv_std_logic_vector(1,DW);
signal index_rin, index_r :
  std_logic_vector(DW-1 downto 0);
begin
  U_adder : add
    generic map (DW => DW)
    port map (
      a => index_r, b => one_dw,
      sum => index_rin
    );

  U_reg : reg
    generic map (DW => DW)
    port map (
      clk => clk, reset => reset,
      load => inc_en,
      d => index_rin,
      q => index_r
    );

  index_out <= index_r;
  index_plus_one <= index_rin;
end rtl;
```

Παραμετρικότητα κυκλωμάτων (1)

- Τα δύο κυκλώματα της HWLU που οφείλουν να είναι παραμετρικά ως προς τον συνολικό αριθμό των υποστηριζόμενων βρόχων (n) είναι:
  - priority_encoder.vhd: ο κωδικοποιητής προτεραιότητας
  - hwlu.vhd: το κύκλωμα HWLU στο οποίο χρησιμοποιούνται τα κατώτερα ιεραρχικά κυκλώματα (υπομονάδες)
- Υλοποίηση της παραμετρικότητας
  - Με τις εντολές FOR...GENERATE και IF...GENERATE επιτυγχάνεται η παραμετρική περιγραφή ενός κυκλώματος εφόσον παρουσιάζει κανονικότητα
  - Με σχεδιασμό γεννήτορα κυκλωμάτων σε κάποια γλώσσα προγραμματισμού όπως η C
  - Γέννηση κυκλώματος για συγκεκριμένη, κάθε φορά, τιμή του n

Παραμετρικότητα κυκλωμάτων (2)

- Τμήμα περιγραφής από το αρχείο genhwlu.c το οποίο παράγει το hwlu.vhd

```
/* Iterate through all loops */
for (i=1; i<=nlp; i++)
{
  /* Generate assignment code */
  fprintf(outfile,
  "\ttemp_loop_count( ((NLP-%d)*DW-1) downto ((NLP-%d)*DW) ) <= loop%d_count;\n",
  i-1, i, i);
}
```

Κωδικοποιητής προτεραιότητας για n=3

```
...
entity priority_encoder is
  generic (
    NLP : integer := 3
  );
  port (
    flag : in
    std_logic_vector(NLP-1 downto 0);
    task_loop3_end : in std_logic;
    incl : out
    std_logic_vector(NLP-1 downto 0);
    reset_vct : out
    std_logic_vector(NLP-1 downto 0);
    loops_end : out std_logic
);
end priority_encoder;

architecture rtl of priority_encoder is
begin
process (flag, task_loop5_end)
begin
  -- if loop2 is terminating:
  -- reset loops 2-0 to initial index
  if (flag(2 downto 0) = "111") then
    incl <= "000";
    reset_vct <= "111";
    loops_end <= '1';
```

```
  -- else if loop1 is terminating:
  -- 1. increment loop2 index
  -- 2. reset loop1 to initial index
  elsif (flag(1 downto 0) = "11") then
    incl <= "100";
    reset_vct <= "011";
    loops_end <= '0';
  -- else if loop0 is terminating:
  -- 1. increment loop1 index
  -- 2. reset loop0 to initial index
  elsif (flag(0 downto 0) = "1") then
    incl <= "010";
    reset_vct <= "001";
    loops_end <= '0';
  -- else increment loop-1 index
  else
    reset_vct <= "000";
    loops_end <= '0';
    if (task_loop3_end = '1') then
      incl <= "001";
    else
      incl <= "000";
    end if;
  end if;
end process;
end rtl;
```

Το συνολικό κύκλωμα: HWLU (1)

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

entity hwlu is
  generic (
    DW  : integer := 8;
    NLP : integer := 3
  );
  port (
    clk             : in std_logic;
    reset           : in std_logic;
    task_loop3_end  : in std_logic;
    loop1_count     : in std_logic_vector(DW-1 downto 0);
    loop2_count     : in std_logic_vector(DW-1 downto 0);
    loop3_count     : in std_logic_vector(DW-1 downto 0);
    index1          : out std_logic_vector(DW-1 downto 0);
    index2          : out std_logic_vector(DW-1 downto 0);
    index3          : out std_logic_vector(DW-1 downto 0);
    loops_end       : out std_logic
  );
end hwlu;

architecture structural of hwlu is
-- Component declarations
component cmpeq
component index_inc
component priority_encoder
...
```

Το συνολικό κύκλωμα: HWLU (2)

```
signal flag : std_logic_vector(NLP-1 downto 0);
signal incl : std_logic_vector(NLP-1 downto 0);
signal temp_loop_count : std_logic_vector(NLP*DW-1 downto 0);
signal temp_index : std_logic_vector(NLP*DW-1 downto 0);
signal temp_index_plus_one : std_logic_vector(NLP*DW-1 downto 0);
signal reset_vct_penc : std_logic_vector(NLP-1 downto 0);
signal reset_vct_ix : std_logic_vector(NLP-1 downto 0);
begin
  temp_loop_count( ((NLP-0)*DW-1) downto ((NLP-1)*DW) ) <= loop1_count;
  temp_loop_count( ((NLP-1)*DW-1) downto ((NLP-2)*DW) ) <= loop2_count;
  temp_loop_count( ((NLP-2)*DW-1) downto ((NLP-3)*DW) ) <= loop3_count;

  GEN_COMPARATORS: for i in 0 to NLP-1 generate
    U_cmp : cmpeq
      generic map (DW => DW)
      port map (
        a => temp_index_plus_one( ((i+1)*DW-1) downto (i*DW) ),
        b => temp_loop_count( ((i+1)*DW-1) downto (i*DW) ),
        reset => reset,
        a_eq_b => flag(i)
      );
end generate GEN_COMPARATORS;
```

Το συνολικό κύκλωμα: HWLU (3)

```
  U_priority_enc : priority_encoder
    generic map (NLP => NLP)
    port map (
      flag => flag, task_loop5_end => task_loop5_end, incl => incl,
      reset_vct => reset_vct_penc,
      loops_end => loops_end
    );

  GEN_RESET_SEL: for i in 0 to NLP-1 generate
    reset_vct_ix(i) <= reset_vct_penc(i) or reset;
  end generate GEN_RESET_SEL;

  GEN_INC_IX: for i in 0 to NLP-1 generate
    U_inc_ix1 : index_inc
      generic map (DW => DW)
      port map (
        clk => clk,
        reset => reset_vct_ix(i),
        inc_en => incl(i),
        index_plus_one => temp_index_plus_one( ((i+1)*DW-1) downto (i*DW) ),
        index_out => temp_index( ((i+1)*DW-1) downto (i*DW) )
    );
  end generate GEN_INC_IX;

  index1 <= temp_index( ((NLP-0)*DW-1) downto ((NLP-1)*DW) );
  index2 <= temp_index( ((NLP-1)*DW-1) downto ((NLP-2)*DW) );
  index3 <= temp_index( ((NLP-2)*DW-1) downto ((NLP-3)*DW) );
end structural;
```

Παραμετρική εκδοχή της HWLU χωρίς την ανάγκη γεννήτορα στην C (1)

```
...
entity hwlu is
  generic (
    DW  : integer := 8;
    NLP : integer := 3
  );
  port (
    clk             : in std_logic;
    reset           : in std_logic;
    task_loop_end   : in std_logic;
    loop_count      : in std_logic_vector(NLP*DW-1 downto 0);
    index           : out std_logic_vector(NLP*DW-1 downto 0);
    loops_end       : out std_logic
  );
end hwlu;
...
  temp_loop_count <= loop_count;

  GEN_COMPARATORS: for i in 0 to NLP-1 generate
    U_cmp : cmpeq
      generic map (DW => DW)
      port map (
        a => temp_index_plus_one( ((i+1)*DW-1) downto (i*DW) ),
        b => temp_loop_count( ((i+1)*DW-1) downto (i*DW) ),
        reset => reset,
        a_eq_b => flag(i)
      );
end generate GEN_COMPARATORS;
```

Παραμετρική εκδοχή της HWLU χωρίς την ανάγκη γεννήτορα στη C (2)

```
  U_priority_enc : priority_encoder
    generic map (NLP => NLP)
    port map (
      flag => flag,
      task_loop_end => task_loop_end,
      incl => incl,
      reset_vct => reset_vct_penc,
      loops_end => loops_end
    );

  GEN_RESET_SEL: for i in 0 to NLP-1 generate
    reset_vct_ix(i) <= reset_vct_penc(i) or reset;
  end generate GEN_RESET_SEL;

  GEN_INC_IX: for i in 0 to NLP-1 generate
    U_inc_ix1 : index_inc
      generic map (DW => DW)
      port map (
        clk => clk,
        reset => reset_vct_ix(i),
        inc_en => incl(i),
        index_plus_one => temp_index_plus_one( ((i+1)*DW-1) downto (i*DW) ),
        index_out => index( ((i+1)*DW-1) downto (i*DW) )
    );
  end generate GEN_INC_IX;
end structural;
```

Testbench για τον έλεγχο της λειτουργίας του κυκλώματος (1)

- Για τον έλεγχο της περίπτωσης του βρόχου:

```
for (i1=0; i1<2; i1++) {
  for (i2=0; i2<6; i2++) {
    for (i3=0; i3<4; i3++) {
    ... } } }
```

- Περιγραφή του testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;

entity hwlu_top_tb is
  generic (
    DW  : integer := 8;
    NLP : integer := 3
  );
end hwloop_top_tb;

architecture tb_arch of hwlu_top_tb is
-- Component declaration of the DUT
component hwlu is
...
```

Testbench για τον έλεγχο της λειτουργίας του κυκλώματος (2)

- Περιγραφή του testbench (συνέχεια)

```
-- Signal declarations
signal clk             : std_logic;
signal reset           : std_logic;
signal task_loop_end   : std_logic;
signal loop1_count     : std_logic_vector(DW-1 downto 0);
signal loop2_count     : std_logic_vector(DW-1 downto 0);
signal loop3_count     : std_logic_vector(DW-1 downto 0);
signal index1          : std_logic_vector(DW-1 downto 0);
signal index2          : std_logic_vector(DW-1 downto 0);
signal index3          : std_logic_vector(DW-1 downto 0);
signal loops_end       : std_logic;
-- Constant declarations
constant CLK_PERIOD : time := 10 ns;
begin
  UUT : hwlu
    generic map (DW => DW, NLP => NLP)
    port map (
      clk => clk,
      reset => reset,
      task_loop_end => task_loop_end,
      loop1_count => loop1_count,
      loop2_count => loop2_count,
      loop3_count => loop3_count,
      index1 => index1, index2 => index2,  index3 => index3,
      loops_end => loops_end
    );
```

Testbench για τον έλεγχο της λειτουργίας του κυκλώματος (3)

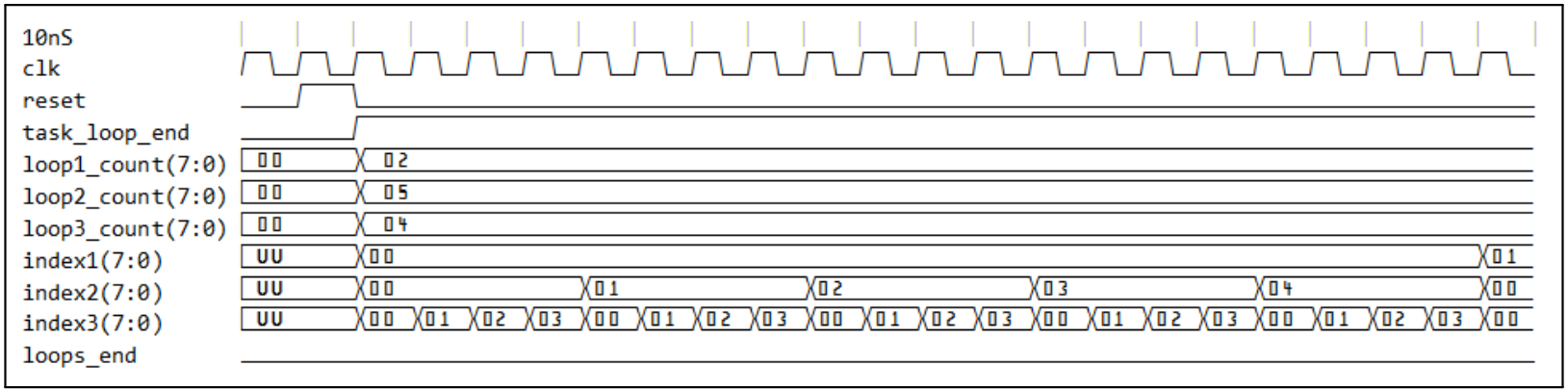
- Περιγραφή του testbench (συνέχεια)

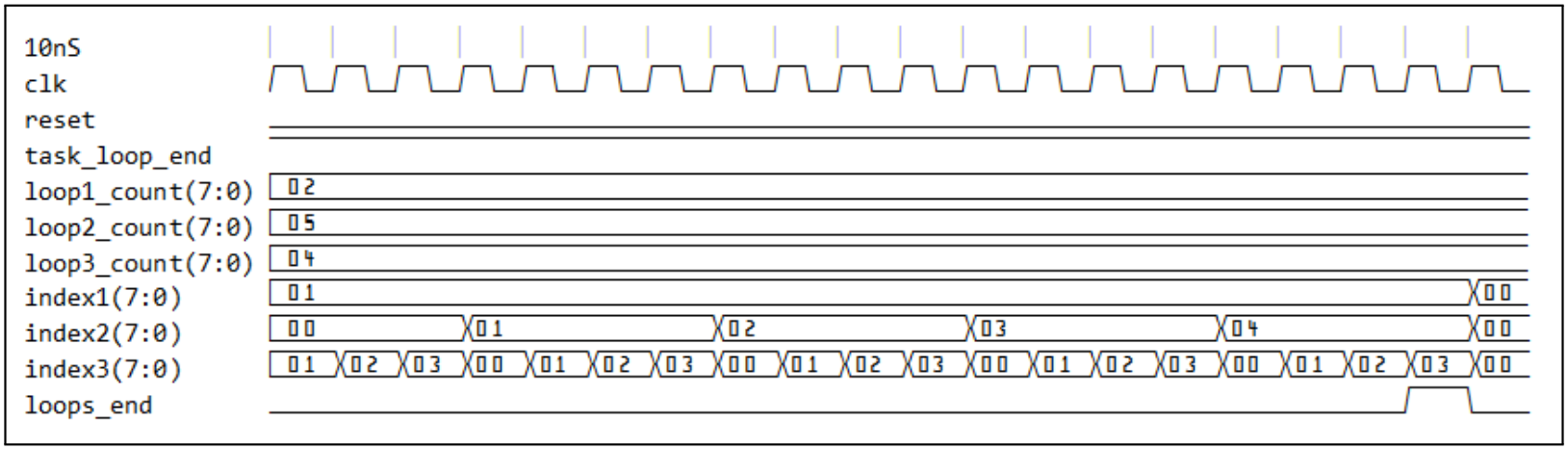
```
CLK_GEN_PROC: process(clk)
begin
  if (clk = 'U') then
    clk <= '1';
  else
    clk <= not clk after CLK_PERIOD/2;
  end if;
end process CLK_GEN_PROC;

DATA_STIM: process
begin
  reset <= '0'; task_loop3_end <= '0';
  loop1_count <= X"00";
  loop2_count <= X"00";
  loop3_count <= X"00";
  wait for CLK_PERIOD;
  --
  reset <= '1';
  wait for CLK_PERIOD;
  --
  reset <= '0'; task_loop3_end <= '1';
  loop1_count <= X"02";
  loop2_count <= X"06";
  loop3_count <= X"04";
  wait for 201*CLK_PERIOD;
end process DATA_STIM;
end tb_arch;
```

Αποτελέσματα από την προσομοίωση του κυκλώματος

Διαγράμματα χρονισμού του κυκλώματος





Οργάνωση της αναφοράς για την εργασία (1)

- Η γραπτή εργασία για τη VHDL έχει τη μορφή τεχνικής αναφοράς (technical report)
- Απαραίτητα στοιχεία
  - Εισαγωγή (σε τι εξυπηρετεί το κύκλωμα)
  - Περιγραφή της αρχιτεκτονικής του κυκλώματος
    - Διεπαφή, σχηματικό διάγραμμα
  - Περιγραφή της λειτουργίας του κυκλώματος
    - Πως παράγονται οι έξοδοι και κάποιες τιμές των εισόδων και για δεδομένη κατάσταση (περιεχόμενα μνημών και καταχωρητών) του κυκλώματος
    - Πως συμμετέχουν οι βασικές υπομονάδες στη λειτουργία του κυκλώματος

Οργάνωση της αναφοράς για την εργασία (2)

- Απαραίτητα στοιχεία (συνέχεια)
  - Περιγραφή όλων των μονάδων και (αν υπάρχει) του αρχείου testbench
    - Κώδικας VHDL
    - Διανύσματα εισόδου/εξόδου (αν υπάρχουν)
  - Κατάλογος αρχείων του σχεδιασμού
  - Παρατηρήσεις (π.χ. περιοχές των τιμών εισόδου) για τη λειτουργία του κυκλώματος
  - Παραρτήματα
    - Γεννήτορες κώδικα VHDL σε C, BASIC, Pascal, Python, Ruby ή ότι άλλο χρησιμοποιήθηκε
    - Σκριπτάκια (scripts) και Makefiles, αρχεία δέσμης (batch files) αν χρησιμοποιήθηκαν

Κατάλογος αρχείων του σχεδιασμού

- Αρχεία για την εργασία σχεδιασμού της μονάδας HWLU

Αρχείο	**Περιγραφή**
`/rtl/add.vhd`	Παραμετρικός αθροιστής
`/rtl/mux2_1.vhd`	Πολυπλέκτης 2-σε-1 για διανύσματα
`/rtl/reg.vhd`	Παραμετρικός καταχωρητής
`/rtl/cmpeq.vhd`	Συγκριτής ισότητας με είσοδο καθαρισμού
`/rtl/index_inc.vhd`	Κύκλωμα αύξησης του δείκτη
`/rtl/prenc_loops3.vhd`	Κωδικοποιητής προτεραιότητας για αριθμό βρόχων $n = 3$
`/rtl/hwlu_loops3.vhd`	Η μονάδα HWLU για $n = 3$
`/test/hwlu_loops3_tb.vhd`	Testbench για τη μονάδα HWLU
`/doc/hwlu.pdf`	Τεχνική αναφορά με την περιγραφή του κυκλώματος
`/sw/genhl.c`	Γεννήτορας της μονάδας HWLU για συγκεκριμένες τιμές του n
`/sw/genpenc.c`	Γεννήτορας του κωδικοποιητή προτεραιότητας για συγκεκριμένες τιμές του n
`/sw/Makefile`	Makefile για τους γεννήτορες κυκλωμάτων