Σειρά Ασκήσεων 9:
Δεύτερη Γνωριμία με τη Γλώσσα Verilog

Προθεσμία έως Παρασκευή 26 Μαρτίου 2010, ώρα 23:59

[**Up** - Table of Contents] [printer version - **PDF**]
[**Prev** - 8. Verilog Intro. 1] [10. Processor Datapath - **Next**]

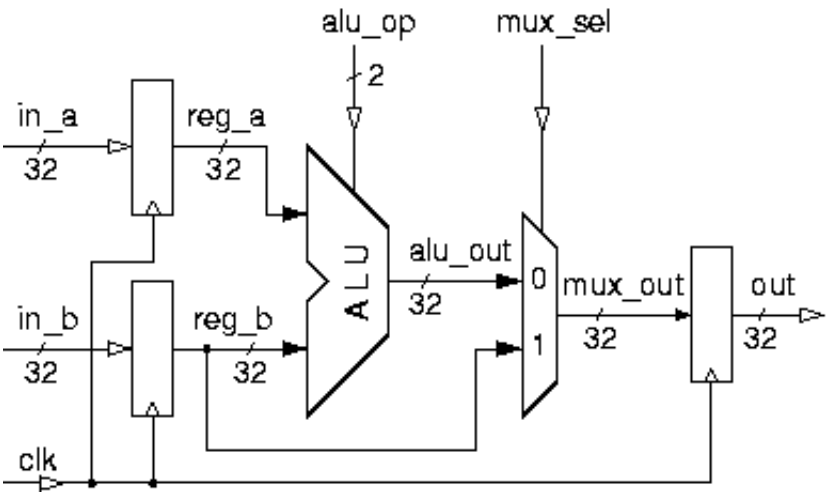
Ο σκοπός αυτής της σειράς ασκήσεων είναι η παρουσίαση ορισμένων επιπλέον δυνατοτήτων της γλώσσας Verilog και η περαιτέρω εξοικείωση σας με τα συναφή εργαλεία. Συγκεκριμένα θα χρησιμοποιήσουμε:

- Συνδυαστικά και ακολουθιακά στοιχεία από τη βιβλιοθήκη του μαθήματος (**`lib9_mux2, lib9_mux4, lib9_alu, lib9_reg`**).
- Σταθερές και σήματα πολλών bits.
- Ρολόϊ.

Οι επιμέρους άσκησεις της σειράς 9 περιλαμβάνουν:

- Μελέτη ενός κυκλώματος και της περιγραφής του σε Verilog που δίδονται.
- Προσομοίωση του κώδικα με το ModelSim
- Επέκταση του κώδικα για να περιλαμβάνει νέα κυκλώματα και επαναπροσομοίωση και έλεγχο εξόδων.

Άσκηση 9.1: Περιγραφή Κυκλώματος σε Verilog



Το κύκλωμα με το οποίο θα ξεκινήσουμε φαίνεται στο σχήμα δίπλα. Τα άσπρα βέλη χρησιμοποιούνται για να δείξουν ότι τα σύρματα αυτά είναι είσοδοι ή έξοδοι απο το σύστημα μας. Το σύστημα διαβάζει τις εισόδους **`in_a`** και **`in_b`** σε κάθε κύκλο ρολογιού **`clk`**, υπολογίζει την πράξη μεταξύ τους --όπως ορίζει το **`alu_op`**-- και τέλος αποθηκεύει στον καταχωρητή εξόδου είτε το αποτέλεσμα αυτό ή την είσοδο **`in_b`** (από τον καταχωρητή εισόδου). Ο κώδικας που περιγράφει το κύκλωμα αυτό σε Verilog είναι ο εξής:

```
`timescale 1ns/1ps
module ask9a (out, in_a, in_b, alu_op, mux_sel, clk);
   output [31:0] out;          // data output
   input  [31:0] in_a, in_b;   // data inputs
   input   [1:0] alu_op;       // control inputs
   input         mux_sel;
   input  clk;                 // clock

   // declare internal signals:
   wire [31:0] reg_a, reg_b, alu_out, mux_out;

   // input registers:
   lib9_reg #32 r0 (reg_a, in_a, clk);
   lib9_reg #32 r1 (reg_b, in_b, clk);

   // ALU:
   lib9_alu #32 alu0 (alu_out, reg_a, reg_b, alu_op);

   // mux:
```

```
    lib9_mux2 #32 m0 (mux_out, alu_out, reg_b, mux_sel);

    // output register:
    lib9_reg #32 r2 (out, mux_out, clk);
endmodule
```

- Μετά το όνομα του module τοποθετούμε τη λίστα των σημάτων τα οποία αποτελούν τη διεπαφή (interface) του module με το υπόλοιπο κύκλωμα. Για κάθε σήμα ορίζουμε αν είναι είναι σήμα εισόδου ή σήμα εξόδου καθώς και το πλάτος του.
- Η γλώσσα Verilog υποστηρίζει και σήματα πολλών bits (vectors). Π.χ. η δήλωση "**`wire [31:0] reg_a, reg_b, ...`**" σημαίνει ότι τα σήματα **reg_a** και **reg_b** έχουν πλάτος 32 bits, το πιο σημαντικό bit το ονομάζουμε bit 31, και το λιγότερο σημαντικό bit το ονομάζουμε bit 0. Μετά από μιά τέτοια δήλωση, αν γράψουμε "**`reg_a[3]`**" σημαίνει το bit 3 από το σήμα **reg_a**, ενώ αν γράψουμε "**`reg[31:27]`**" σημαίνει τα 5 πιό σημαντικά bits του **reg_a**.
- Το κύκλωμα μας αποτελείται από τρία αντίτυπα του module lib9_reg, ένα αντίτυπο του module lib9_mux2, και ένα αντίτυπο του module lib9_alu. Η γλώσσα Verilog υποστηρίζει την παραμετροποίηση των modules, δηλ. ένα module μπορεί να έχει μία ή περισσότερες παραμέτρους στις οποίες δίνουμε τιμή κάθε φορά που δημιουργούμε ένα αντίτυπο από το module. Έτσι, το "#32" μεταξύ του "lib9_reg" (όνομα του module) και του "r0" (όνομα του αντίτυπου) δίνει την τιμή 32 σε μία παράμετρο του module "lib9_reg" από τη βιβλιοθήκη του μαθήματος η οποία δηλώνει το πλάτος του "lib9_reg", δηλαδή πόσα bits έχει αυτός ο καταχωρητής.
- Το module "lib9_alu" είναι μιά απλή αριθμητική/λογική μονάδα. Η είσοδος ελέγχου της, alu_op, καθορίζει την πράξη που κάνει η μονάδα ως εξής:

  00
  : πρόσθεση: out = inA **+** inB.

  01
  : αφαίρεση: out = inA **-** inB.

  10
  : bitwise OR: out = inA **OR** inB.

  11
  : bitwise AND: out = inA **AND** inB.

- Για καλύτερη αναγνωσιμότητα του κώδικα Verilog, συνηθίζουμε να δημιουργούμε τα αντίτυπα των στοιχείων του κυκλώματος με τη σειρά που εμφανίζονται στο κύκλωμα.

Γράψτε το παραπάνω module στο αρχείο "**ask9a.v**" --αυτή είναι η περιγραφή του κυκλώματος. Σε αυτήν την άσκηση, το περιβάλλον ελέγχου (test bench) θα είναι σε ξεχωριστό αρχείο, και δίδεται έτοιμο:

~hy225/10a/verilog/test/test9a.v

Αντιγράψτε το αρχείο αυτό στην περιοχή σας. Η βασική δομή του έχει ως εξής:

```
`timescale 1ns/1ps
`define clk_period 10
`define hold 1

module test;

    reg [31:0] in_a, in_b;
    reg [1:0] alu_op;
    reg mux_sel;
    wire [31:0] out;

    // clock:
    reg clk;
    initial clk = 1;
    always begin
            #(`clk_period / 2)
            clk = ~clk;
    end

    // instantiate the design:
```

```
    ask9a a0 (out, in_a, in_b, alu_op, mux_sel, clk);

    // vectors:
    initial begin
        ...
        @(posedge clk);
        #(`hold);
        in_a    = 3;
        in_b    = 'hFFFF;
        alu_op  = 0;
        mux_sel = 1;
        @(posedge clk);
        ...
    end
endmodule
```

- Το "**`define**" στη γλώσσα Verilog είναι ανάλογο με το "**#define**" στη γλώσσα C. Με τη βοήθεια της define ορίζουμε την σταθερά "clk_period" που παρακάτω την χρησιμοποιούμε σαν περίοδο του ρολογιού μας. Σε αυτή την άσκηση έχουμε θέσει τη μονάδα χρόνου στο 1 ns, μέσω της βιβλιοθήκης lib9.v που θα χρησιμοποιήσετε. Έτσι, το παραπάνω `define θέτει την περίοδο του ρολογιού σε 10 ns.
- Η εντολή "**always**" σημαίνει: ξεκίνα τη χρονική στιγμή 0 και συνέχισε εκτελώντας επαναληπτικά τις εντολές μέσα στο block που ακολουθεί. Έτσι, το παραπάνω μπλόκ "always" για το ρολόι θα έχει σαν αποτελέσμα να αλλάζει η τιμή του σήματος clk κάθε (`clk_period / 2) ns.
- Η εντολή "**@(*event*)**" σημαίνει: περίμενε μέχρι να συμβεί το *event*. Το *event* μπορεί να είναι είτε μια οποιαδήποτε αλλαγή στην τιμή ενός σήματος "sig", το οποίο το δηλώνουμε σαν "**@(sig)**", ή η θετική ακμή ενός σήματος "sig", το οποίο το δηλώνουμε σαν "**@(posedge sig)**", ή η αρνητική ακμή, "**@(negedge sig)**".
- Αφού έρθει η θετική ακμή του ρολογιού, δηλ. "**@(posedge clk)**", περιμένουμε ακόμα λίγο, δηλ. "**#(`hold)**", πριν αλλάξουμε τις τιμές στα σήματα εισόδου, προκειμένου να μην παραβιάσουμε το "hold time" για τους καταχωρητές του κυκλώματος.
- Σταθερές πολλών bits δηλώνονται με τη μορφή ***[πλάτος]'[βάση][τιμή]***.
    - Το ***[πλάτος]*** είναι μία δεκαδική τιμή που καθορίζει το πλάτος της σταθεράς σε bits, δηλαδή πόσα bits θα έχει η σταθερά. Εάν παραλειφθεί το ***[πλάτος]***, το default εξαρτάται από το μηχάνημα όπου τρέχει η Verilog, αλλά είναι τουλάχιστο 32 bits. Επέκταση του πλάτους γίνεται με zero-filling (εκτός των περιπτώσεων "x" και "z" που δεν θα ασχοληθούμε).
    - Η ***[βάση]*** καθορίζει την αριθμητική βάση στην οποία είναι γραμμένη η ***[τιμή]*** της σταθεράς. Η δεκαδική βάση δηλώνεται με "d", η δυαδική με "b", και η δεκαεξαδική με "h". Εάν παραλειφθεί η ***[βάση]***, το default είναι δεκαδική βάση.

  Έτσι π.χ., η σταθερά "32'hF" έχει πλάτος 32 bits, και έχει την δεκαεξαδική τιμή 0000000F, οπότε μπορεί να γραφεί και ως "32'b1111", ή "32'b00001111", ή "32'd15", ή κλπ.

Θα χρειαστεί επίσης να αντιγράψετε στην περιοχή σας το αρχείο:

~hy225/10a/verilog/lib/lib9.v

που περιέχει τη βιβλιοθήκη των modules του **ask9a**: καταχωρητές (**lib9_reg**), αριθμητική-λογική μονάδα (**lib9_alu**), και πολυπλέκτες (**lib9_mux2**). Τρέξετε τον προσομοιωτή όπως στην άσκηση **8.3**. Ελέγξτε ότι οι έξοδοι έχουν σωστή τιμή με βάση τις εισόδους. Προσοχή στο ποιές είσοδοι περνούν απο καταχωρητή και ποιές όχι!

Άσκηση 9.2: Έλεγχος Κυματομορφών με το ModelSim

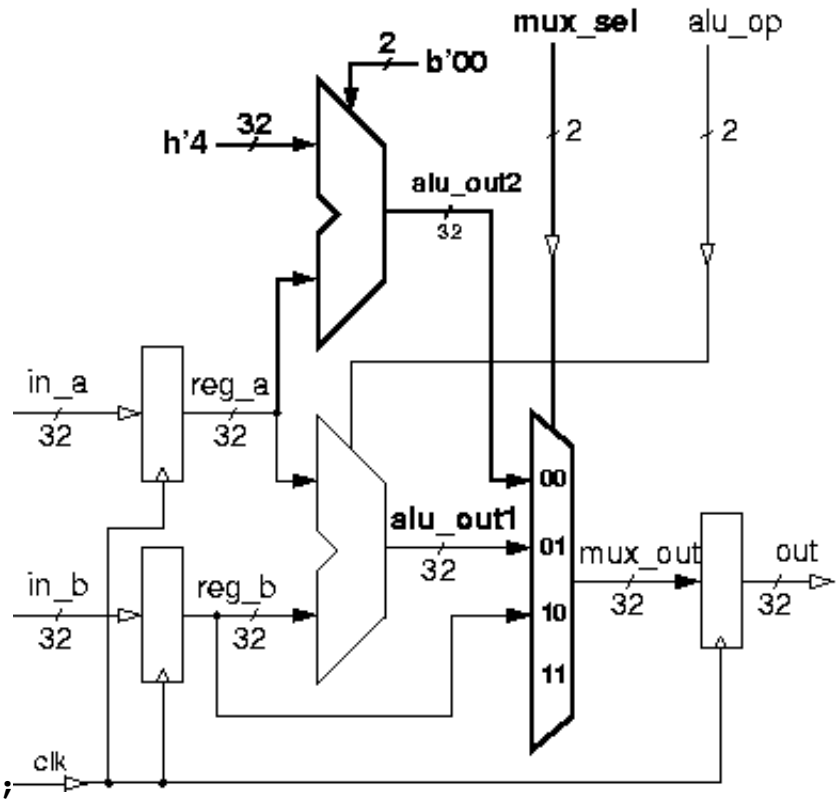
Χρησιμοποιήστε το ModelSim, όπως στην άσκηση **8.4**, γιά να δείτε τις κυματομορφές για τα σήματα.

- Αφού τρέξει το γραφικό περιβάλλον του προγράμματος, ακολουθήστε ανάλογα βήματα με την άσκηση 8.4 για να δείτε τις κυματομορφές για όλα τα σήματα του test: out, in_a, in_b, alu_op, mux_sel, clk. Ελέγξτε και πάλι τις τιμές τους.
- Αν επιλέξουμε πάνω δεξιά "test" και μετά "a0" (το "a0" είναι το instance name με το οποίο έχουμε δηλώσει το αντίτυπο του module ask9a) μπορούμε να διαλέξουμε και τα εσωτερικά σήματα του "a0". Επιλέξτε τα κατάλληλα σήματα προκειμένου να μετρήσετε την **ελάχιστη και μέγιστη καθυστέρηση** clock-to-output του καταχωρητή, και την ελάχιστη και μέγιστη καθυστέρηση γιά τα συνδυαστικά blocks (alu και mux2). Οι καθυστερήσεις είναι της τάξης των εκατοντάδων picoseconds.
- *Προαιρετική άσκηση*: Βρείτε το critical path του κυκλώματος και, αντίστοιχα, την υψηλότερη συχνότητα

ρολογιού στην οποία δουλεύει το κύκλωμα. Εφαρμόστε την (θέτοντας κατάλληλα το "clk_period") και δείτε ότι δουλεύει, ενώ αν βάλετε λίγο γρηγορότερο ρολόι το κύκλωμα παύει να δουλεύει. Οι χρόνοι setup και hold του καταχωρητή είναι 0.2 ns και 0.1 ns αντίστοιχα.

Άσκηση 9.3: Τροποποίηση του Κυκλώματος

Τροποποιήστε το κύκλωμα που σας δόθηκε παραπάνω, ώστε να γίνει όπως στο σχήμα εδώ. Η καινούρια προσθήκη επιτρέπει το αποτέλεσμα να είναι ίσο με **`(in_a + 4)`**, όταν **`mux_sel == 0`**. Τα στοιχεία που είναι καινούρια, ή που πρέπει να αλλάξουν από το προηγούμενο σχέδιο, είναι απεικονισμένα με παχύτερες γραμμές. Για τον μεγαλύτερο πολυπλέκτη, θα χρειαστεί να αλλάξετε το module **`lib9_mux2`** στο αμέσως μεγαλύτερο, "**`lib9_mux4`**", και να συνδέσετε μια τυχαία τιμή στην τελευταία του είσοδο (π.χ. ένα σύρμα με την τιμή 0). Οι πόρτες του module **`lib9_mux4`** έχουν δηλωθεί στη βιβλιοθήκη, κατ' αναλογία με τις πόρτες του module **`lib9_mux2`**, με τη σειρά: έξοδος, είσοδος00, είσοδος01, είσοδος10, είσοδος11, σήμα ελέγχου. Σύρματα με σταθερές τιμές κανονικά θα έπρεπε να μπορούσατε να ορίσετε π.χ. με:




```
wire [31:0] const_in;
assign const_in = 32'h4;
```

Όμως, λόγω κάποιου bug που σχετίζεται και με τη δική μας (HY-225) βιβλιοθήκη, δηλώστε την επάνω είσοδο της νέας ALU σαν (αρχικοποιημένο) reg:

```
reg [31:0] const_in;
initial const_in = 32'h4;
```

Υλοποιήστε την επέκταση αυτή σε ένα διαφορετικό αρχείο, "**`ask9b.v`**", με όνομα module "**`ask9b`**" αντί "**`ask9a`**", και δοκιμάστε την όπως προηγουμένως, χρησιμοποιώντας όμως το αρχείο "**`~hy225/verilog/test/test9b.v`**".

Παραδώστε (submit), με τον τρόπο των προηγουμένων ασκήσεων, τρία αρχεία: τον κώδικά σας **`ask9b.v`**, ένα χαρακτηριστικό στιγμιότυπο από το ModelSim της άσκησης 9.2, **`wave92.jpg`**, και ένα άλλο από το ModelSim της άσκησης 9.3, **`wave93.jpg`**.

[**Up** - Table of Contents]
[**Prev** - 8. Verilog Intro. 1]
[printer version - **PDF**]
[10. Processor Datapath - **Next**]

Up to the Home Page of CS-225