18/4/2019

1ο Εργαστήριο Οργάνωσης Υπολογιστών
ΣΧΕΔΙΑΣΗ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ ΜΟΝΑΔΑΣ ΑΡΙΘΜΗΤΙΚΩΝ ΚΑΙ ΛΟΓΙΚΩΝ ΠΡΑΞΕΩΝ (ALU) ΚΑΙ ΑΡΧΕΙΟΥ ΚΑΤΑΧΩΡΗΤΩΝ (REGISTER FILE)
22/2/2019

Ομάδα LAB31239632

Αλέξανδρος Πουπάκης 2016030061
Βαγγέλης Κιούλος 2016030056

Σκοπός εργαστηριακής άσκησης

Στη παρούσα εργαστηριακή άσκηση έπρεπε να σχεδιάσουμε αρχικά ένα συνδυαστικής λογικής ALU που να υποστηρίζει τις πράξεις +, -, and, or, not, arithmetic shift right, logic shift right, logic shift left καθώς και αριστερόστροφο/δεξιόστροφο rotate. Έπειτα, έπρεπε να υλοποιήσουμε ένα register file με 32 καταχωρητές πλάτους 32 bit.

Προεργασία

Για το ALU, σχεδιάσαμε ένα behavioural module στο οποίο περιγράφουμε την λειτουργία του, τις πράξεις που υλοποιεί καθώς και τις καθυστερήσεις των σημάτων εξόδου. Ορίσαμε δυο ασύγχρονα processes όπου στο πρώτο εκτελούμε την πράξη που αντιστοιχεί στο κάθε Op-code και στο δεύτερο περνάμε το αποτέλεσμα της πράξης στο σήμα εξόδου του datapath και υπολογίζουμε τα Zero, Cout και ovf flags. Στο δεύτερο process προστίθενται και οι καθυστερήσεις των σημάτων.

Για το Register File, αρχικά σχεδιάσαμε τον αποκωδικοποιητή καθώς και το Register 32-bit. Στο behavioural module του αποκωδικοποιητή γράφουμε έναν «1» στη θέση των 32 bit που αντιστοιχεί στον δυαδικό αριθμό της εισόδου. Με ένα for loop διατρέχουμε όλα τα bit εξόδου και τα θέτουμε «1» εάν η θέση είναι ίση με τον αριθμό της δυαδικής ακολουθίας εισόδου και «0» αλλιώς.

Στο behavioural module του Register, κατασκευάσαμε ένα σύγχρονο process κατά τα γνωστά από προηγούμενα μαθήματα και περνάμε στην έξοδο την είσοδο εάν βρισκόμαστε στη θετική ακμή του ρολογιού και το WE (write enable) είναι «1».

Τέλος, τα δυο παραπάνω modules συνδυάστηκαν στο structural module του Register File (top module) όπου με ένα for generate παράχθηκαν τα 32 instances του register και διασυνδέθηκαν με τον αποκωδικοποιητή και τα υπόλοιπα σήματα.

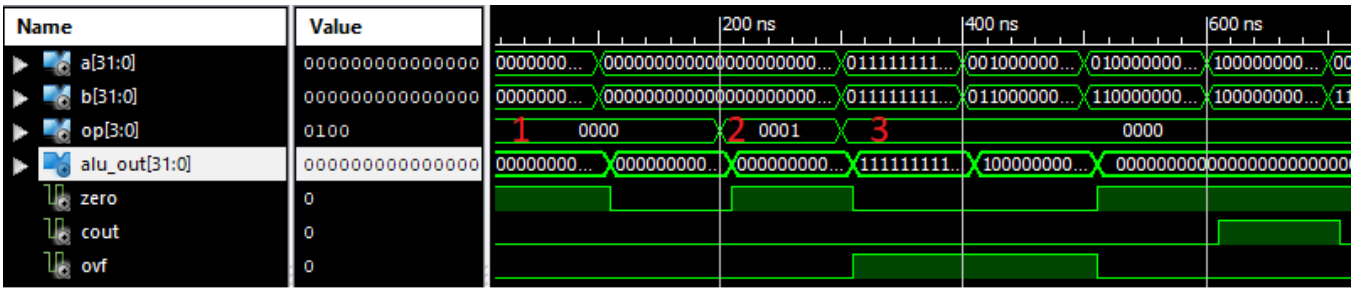
Περιγραφή

Για το ALU χρησιμοποιήθηκε η βιβλιοθήκη IEEE.NUMERIC_STD για την πρόσθεση και αφαίρεση ακεραίων. Οι bitwise πράξεις και τα rotate έγιναν μέσω δεικτοδότησης του STD_LOGIC_VECTOR των τελεστέων και ενδιάμεσων σημάτων. Για την παραγωγή των σημάτων Zero, Cout και ovf χρησιμοποιήθηκαν οι κατάλληλες συνθήκες.

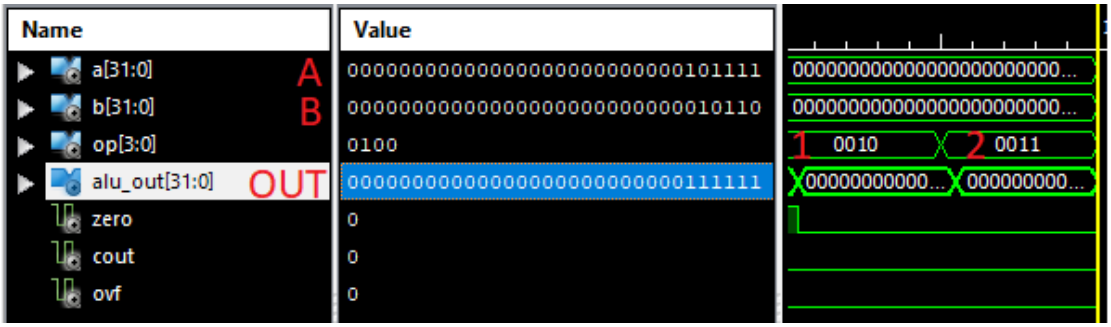
Για το Register File, συνδέσαμε κάθε i-οστο bit του decoder με το WE του αντίστοιχου register μέσω μιας πύλης AND με δεύτερη είσοδο το WrEn. Η εγγραφή δεδομένων γίνεται από το module του κάθε register μέσω ενεργοποίησης του WE του κατάλληλου register. Τα MUX υλοποιούνται σε ένα ξεχωριστό process με 2 for loops τα οποία ελέγχουν εάν το i-οστο register είναι αυτό που περιγράφεται από τις Ard1 και Ard2 αντίστοιχα και περνάνε την έξοδο των register αυτών στο σήμα εξόδου Dout του module. Η συνδεσμολογία των modules για το register file είναι ακριβώς αυτή που περιγράφεται στην εκφώνηση της άσκησης.

Κυματομορφές-Προσομοίωση

ALU



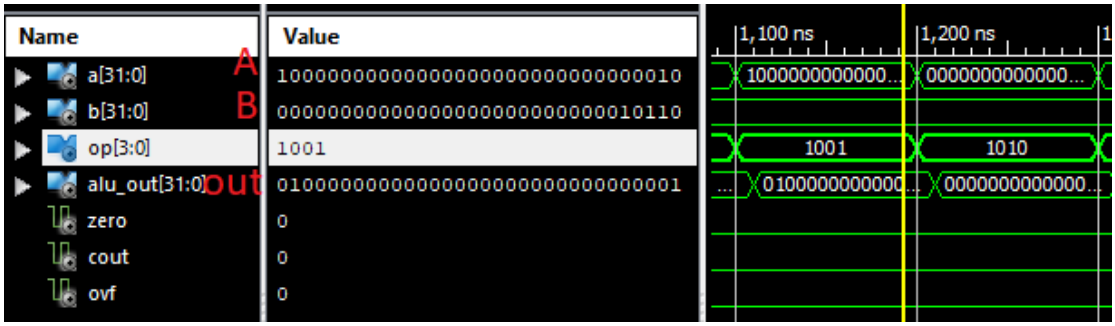
Παραπάνω, παρατηρούμε τις κυματομορφές της προσομοίωσης της ALU. Βλέπουμε ότι όταν το op είναι στο “0000” (νούμερο 1), είμαστε στην πράξη της πρόσθεσης, στην πρώτη περίπτωση, που τα A και B είναι και τα δύο “000...00”, το αποτέλεσμα είναι και αυτό “000...00” και το zero flag είναι '1'. Στην δεύτερη περίπτωση, προσθέτοντας το “000...01” με το “000...01” προκύπτει το σωστό αποτέλεσμα στην έξοδο, δηλαδή “000...010”. Όταν το op είναι στο “0001” (νούμερο 2) είμαστε στην πράξη της αφαίρεσης και βλέπουμε ότι αφαιρώντας το “000...01” από το “000...01” προκύπτει “000...00” στην έξοδο. Στις επόμενες περιπτώσεις (νούμερο 3), προσθέτοντας αρκετά μεγάλα νούμερα, βλέπουμε ότι το ovf flag και το cout flag γίνονται '1' όταν έχουμε υπερχείλιση η κρατούμενο αντίστοιχα.



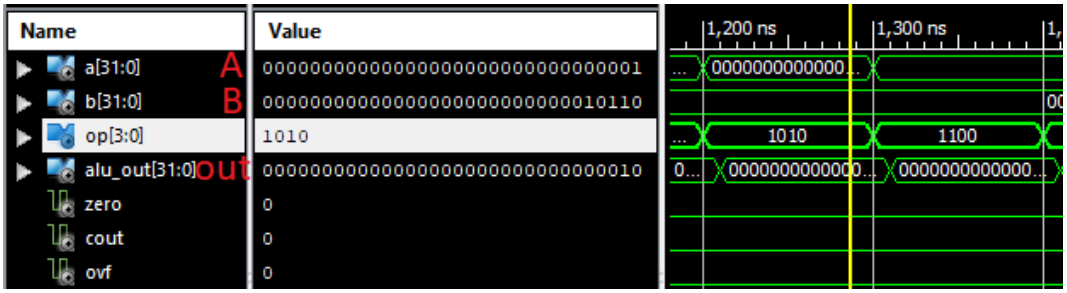
Παραπάνω, παρατηρούμε το αποτέλεσμα της προσομοίωσης (ένδειξη out) όταν το op είναι “0010” (νούμερο 1) όπου έχουμε την λογική πράξη and και όταν το op είναι “0011” (νούμερο 2) όπου έχουμε την λογική πράξη or.

Τέλος, με τα παρακάτω διαγράμματα, αναπαριστάνονται τα αποτελέσματα των ολισθήσεων.

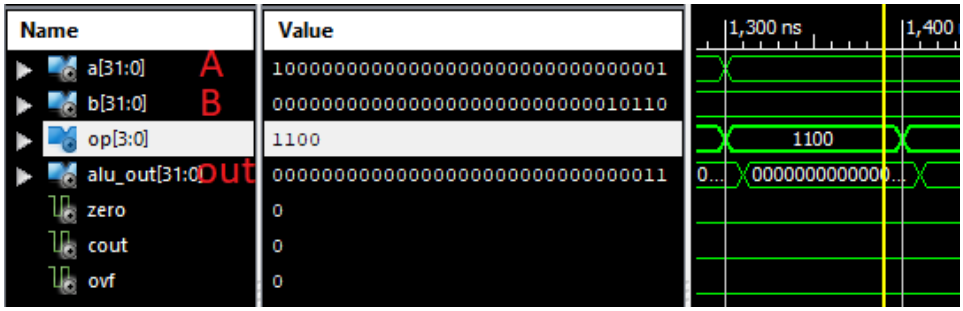
- Όταν το op είναι “1001” έχουμε ολίσθηση προς τα δεξιά με msb '0'. Το αποτέλεσμα της πράξης φαίνεται στην ένδειξη “out”.



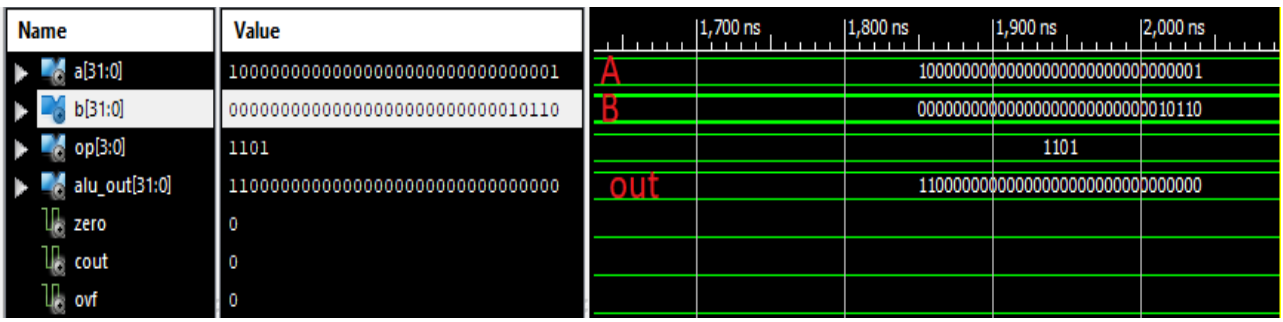
- Όταν το op είναι “1010” έχουμε αριστερή ολίσθηση.



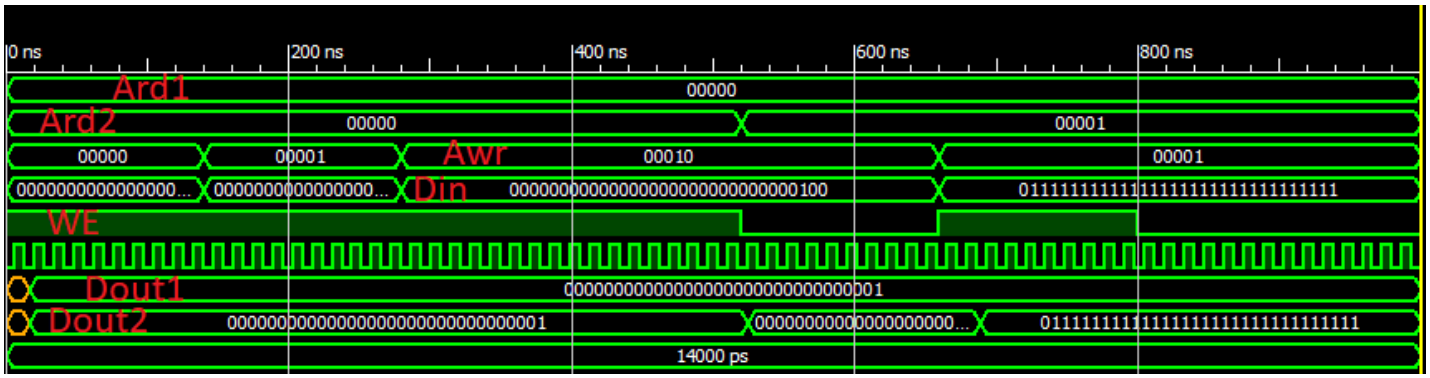
- Όταν το op είναι “1100” έχουμε αριστερή κυκλική ολίσθηση.



- Όταν το op είναι “1101” έχουμε δεξιά κυκλική ολίσθηση.



Register File



Παραπάνω, βλέπουμε τις κυματομορφές της προσομοίωσης από το register file. Παρατηρούμε ότι όταν το we (write enable) είναι '1' και επιλέγουμε τον καταχωρητή “00001”, η τιμή που έχουμε στο Din γράφεται στον καταχωρητή με διεύθυνση “00001”. Εισάγοντας την διεύθυνση του καταχωρητή στην είσοδο Ard1, η αποθηκευμένη τιμή του καταχωρητή εμφανίζεται στο Dout1 και αντίστοιχα εισάγοντας την διεύθυνση του καταχωρητή που θέλουμε στην είσοδο Ard2, εμφανίζουμε την τιμή του καταχωρητή στην έξοδο Dout2. Όταν το we είναι '0', δεν μπορούμε να αποθηκεύσουμε κάποια τιμή στους καταχωρητές, μπορούμε μόνο να διαβάσουμε την τιμή αυτών μέσο των εισόδων Ard1 και Ard2.

Συμπεράσματα

- Χρησιμοποιώντας τις εντολές for και generate, μπορούμε να δημιουργήσουμε πολλά instances κάποιου component με το port map.
- Με την εντολή signed μετατρέπουμε τις εισόδους σε προσημασμένους αριθμούς για να γίνει η αριθμητική πράξη.
- Οι ολισθήσεις μπορούν να γίνουν επικολλώντας bit, σε κάποιο σήμα, με την χρήση του τελεστή &.

Κώδικας

ALU

```
if Op = "0000" then
   temp <= std_logic_vector(resize(signed(A), 33) + resize(signed(B), 33));
elsif Op = "0001" then
   temp <= std_logic_vector(resize(signed(A), 33) - resize(signed(B), 33));
```

Για την υλοποίηση των πράξεων της πρόσθεσης και της αφαίρεσης, στην ALU, χρησιμοποιήσαμε την εντολή signed, για να μετατρέψουμε τις εισόδους από logic vectors σε προσημασμένους αριθμούς και στην συνέχεια μετατρέψαμε το αποτέλεσμα από προσημασμένο αριθμό, σε logic vector.

Register File

```
GEN_RF : for i in 0 to 31 generate
reg : Register32 Port Map (CLK, WE_regI(i), Din, Dout_regI(i));
end generate GEN_RF;
```

Στο παραπάνω κομμάτι κώδικα, χρησιμοποιώντας τις εντολές for και generate, δημιουργούμε με το port map τους 32 καταχωριστές που χρειαζόμαστε.

```
architecture Structural of RegisterFile is
signal Dec_Awr, WE_regI : std_logic_vector (31 downto 0);
type slv_array is array (0 to 31) of std_logic_vector (31 downto 0);
signal Dout_regI : slv_array;
```

Παραπάνω, με την χρήση των εντολών type και is array, δημιουργούμε ένα array 32 θέσεων από logic vectors των 32 bit. Αυτά θα μας χρειαστούν για την δημιουργία του πολυπλέκτη 32 εισόδων.

```
process (Ard1, Ard2, Dout_regI)
begin
   for i in 0 to 31 loop
      if i = to_integer(unsigned(Ard1)) then
         Dout1 <= Dout_regI(i) after 5 ns;
      end if;

      if i = to_integer(unsigned(Ard2)) then
         Dout2 <= Dout_regI(i) after 5 ns;
      end if;
   end loop;
end process;
```

Επιπλέων, με την χρήση του for loop που φαίνεται παραπάνω, υλοποιούμε τους 2 πολυπλέκτες 32 εισόδων που θα χρειαστούμε. Τέλος, με την χρήση του after 5 ns, καθορίζουμε την καθυστέρηση της εξόδου.