Αναφορά 1ου Εργαστηρίου

Ομάδα Εργασίας: LAB31239629
Κονιδάρη Ηρώ A.M. 2012030049
Μάνεσης Αθανάσιος A.M. 2014030061

Σκοπός της Άσκησης

Σκοπός αυτής της άσκησης ήταν η σχεδίαση σε γλώσσα VHDL μιας μονάδας αριθμητικών και λογικών πράξεων (ALU) και ενός αρχείου καταχωρητών και η προσομοίωση της με τα εργαλεία της Xilinx.

Περιγραφή/Υλοποίηση της Άσκησης

Η άσκηση αποτελούνταν από δύο μέρη:

1ο Μέρος:

Στο πρώτο μέρος μας ζητήθηκε να σχεδιάσουμε την μονάδα αριθμητικών και λογικών πράξεων (ALU). Η συγκεκριμένη μονάδα λειτουργεί ως εξής: δέχεται ως **είσοδο δύο 32-bit αριθμούς** σε συμπληρώματα ως προς 2 και ένα **4-bit αριθμό (Op),** ο οποίος προσδιορίζει ποια πράξη πρέπει να γίνει, και παράγει ως **εξόδους ένα 32-bit αριθμό** ο οποίος είναι το αποτέλεσμα της πράξης. Υπάρχουν τρία σήματα που παράγονται επιπλέον: ένα σήμα **Zero** το οποίο υποδηλώνει ότι το αποτέλεσμα είναι μηδέν, ένα σήμα **Cout** το οποίο είναι το κρατούμενο εξόδου στις πράξεις της πρόσθεσης και της αφαίρεσης και το σήμα της υπερχείλισης–**Ovf-** το οποίο ενεργοποιείται όταν υπάρξει υπερχείλιση στις πράξεις της πρόσθεσης και αφαίρεσης. Ουσιαστικά, η ALU εκτελεί ένα process, το οποίο ελέγχει το Op και εκτελεί την ανάλογη πράξη δίνοντας τιμές στα διάφορα σήματα εξόδου.

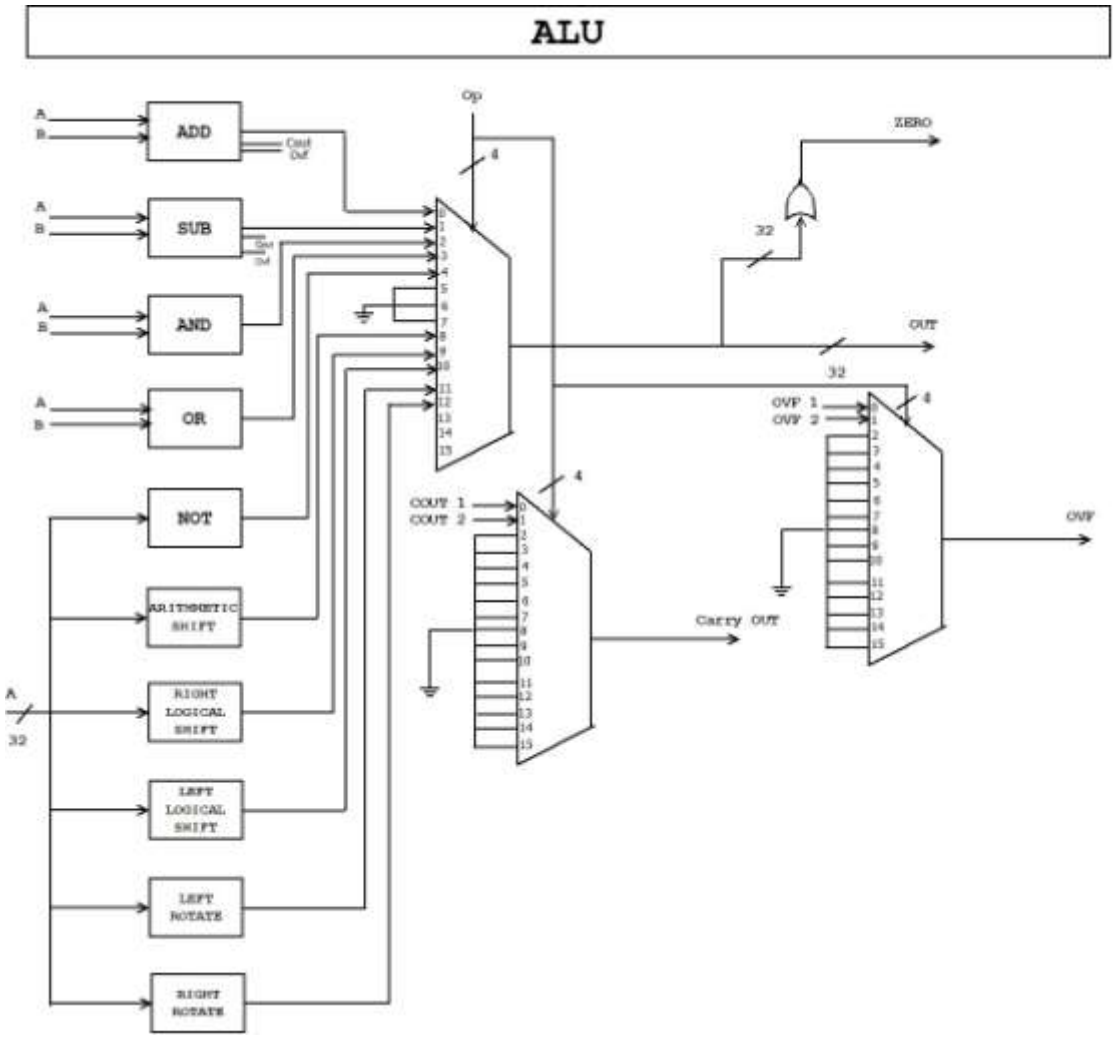
- **Cout** : οι είσοδοί μας A,B ήταν σε συμπλήρωμα ως προς 2 και στην πρόσθεση “προσθέταμε” σε κάθε αριθμό ένα παραπάνω bit ‘0’, το οποίο αν μετά την πράξη γινόταν ‘1’ σήμαινε ότι έχουμε κρατούμενο εξόδου.

- **Overflow**: έχουμε όταν η *πρόσθεση* 2 ομόσημων αριθμών μας δώσει ετερόσημο αποτέλεσμα. Γι αυτό το λόγο στον κώδικά μας ελέγχαμε αν οι δύο είσοδοί μας ήταν ομόσημες και αν το πρόσημο του αποτελέσματος ήταν διαφορετικό τότε είχαμε overflow. Την ίδια μεθοδολογία ακολουθήσαμε και στην αφαίρεση. Σε περιπτώσεις add/sub ετερόσημων αριθμών δεν έχουμε περίπτωση overflow, αφού πάντα θα είμαστε εντός range αναπαράστασης.

- **Zero** : είναι η έξοδος που δημιουργείται αφού περάσουμε τα 32bit εξόδου του πολυπλέκτη της ALU μέσα από μια NOR

- Για τα υπόλοιπα αποτελέσματα σχεδιάσαμε ξεχωριστά modules για την κάθε πράξη και πολυπλέκτες για το control του συστήματος. Αντιστοιχίσαμε την έξοδο κάθε πράξης με τις εισόδους ενός πολυπλέκτη. Έτσι , όλες οι πράξεις γίνονται παράλληλα και ανάλογα με το OP που θα δοθεί θα βγει η επιθυμητή πράξη στην έξοδο.

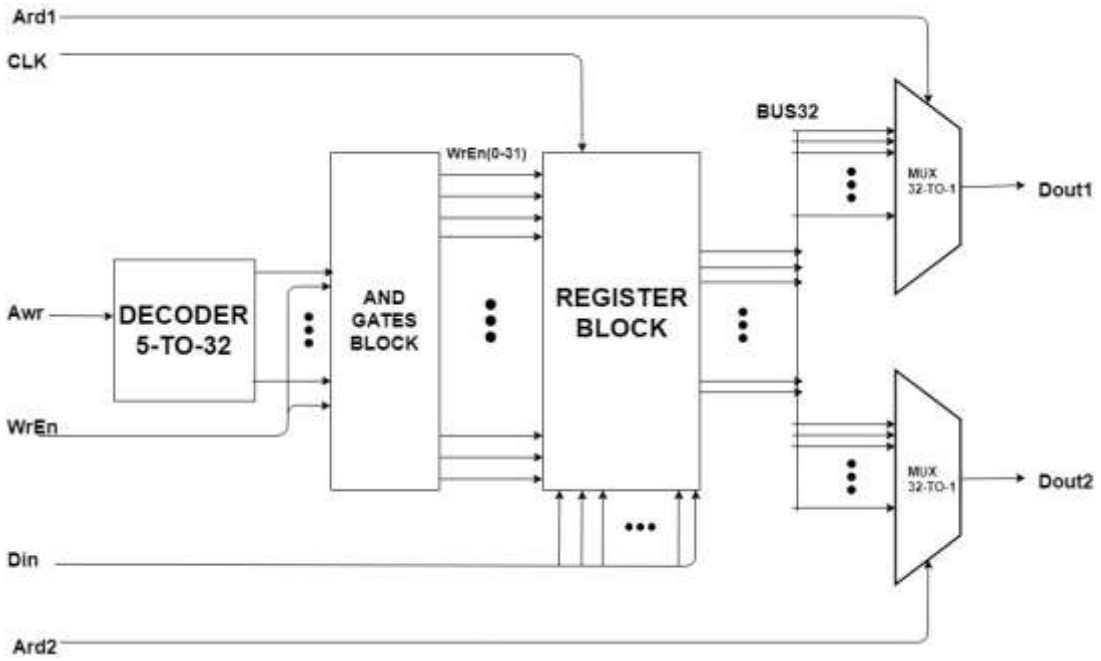
Για τις εξόδους OVF και COUT χρησιμοποιούμε δύο πολυπλέκτες αντίστοιχα. Επειδή περίπτωση OVF και COUT έχουμε μόνο στο ADD και στο SUB, οι υπόλοιπες είσοδοι του πολυπλέκτη οδηγούνται στην γείωση.



2ο Μέρος:

Στο δεύτερο μέρος της άσκησης αρχικά σχεδιάσαμε έναν 32-bit register ο οποίος δεχόταν ως εισόδους τρία σήματα, τα CLK (1bit, ρολόι), DataIn (32-bit, δεδομένα εισόδου) και We (1-bit, write enable, σήμα εγγραφής καταχωρητή), και έβγαζε στην έξοδο ένα σήμα DataOut. Ο καταχωρητής σε κάθε κύκλο του ρολογιού δίνει ως έξοδο την τιμή που ήταν αποθηκευμένη σε αυτόν. Εάν το σήμα We ενεργοποιηθεί τότε η τιμή που είναι αποθηκευμένη στον καταχωρητή αλλάζει και γίνεται αυτή που του δίνεται ως είσοδος. Το σχήμα του καταχωρητή είναι το παρακάτω:

Στην συνέχεια, χρησιμοποιήσαμε 32 καταχωρητές σαν αυτούς που υλοποιήσαμε παραπάνω, 2 πολυπλέκτες 32 προς 1,1 αποκωδικοποιητή 5-bit σε 32-bit για να δημιουργήσουμε το αρχείο καταχωρητών (RF - register file). Η συνδεσμολογία και το block diagram της RF απεικονίζεται παρακάτω:



Ο RF δέχεται ως εισόδους 6 σήματα. Αυτά είναι: το **ρολόι (CLK**), τα **Ard1** και **Ard2** (5 bit το καθένα) τα οποία έχουν την διεύθυνση του πρώτου και δεύτερου καταχωρητή για ανάγνωση καθώς και το **Awr** (5 bit) το οποίο έχει την διεύθυνση του καταχωρητή στον οποίο πρόκειται να γράψουμε. Επιπλέον, έχουμε τα σήματα **Din** (32 bit) που έχει τα δεδομένα προς εγγραφή και **WrEn** (1 bit) που είναι το σήμα ενεργοποίησης εγγραφής του καταχωρητή.

Το σήμα **Awr** είναι είσοδος στον **αποκωδικοποιητή** (decoder(5 to 32)). Σκοπός αυτού του στοιχείου είναι να μετατρέψει την 5 bit διεύθυνσης του καταχωρητή που θέλουμε να γράψουμε, σε ένα σήμα των 32 bit. Όλα τα bits αυτού του σήματος είναι '0' με εξαίρεση ένα bit το οποίο είναι '1' και σε αυτή τη θέση που βρίσκεται το συγκεκριμένο bit υποδηλώνεται ο καταχωρητής που θα γράψουμε.

Ύστερα καθένα από 32 bits εξόδου του αποκωδικοποιητή, μαζί με την είσοδο **WrEn** μπαίνουν σε μία πύλη AND (32 πύλες συνολικά). Η έξοδος της πύλης πηγαίνει στο **WrEn** του καταχωρητή (**register**) και ανάλογα με την τιμή της ενεργοποιεί τον καταχωρητή για εγγραφή.

Τα σήματα Ard1 και Ard2 χρησιμεύουν ως σήματα ελέγχου(select) στους 2 **πολυπλέκτες 32 προς 1** , οι οποίοι χρησιμοποιούνται για να διαβάζουμε τις τιμές των δύο καταχωρητών, των οποίων οι διευθύνσεις είναι οι τιμές των Ard1 και Ard2. Οι πολυπλέκτες 32 προς 1 δέχονται ως εισόδους τις 32 εξόδους των καταχωρητών και ανάλογα με τα Ard1 και Ard2 μας δίνουν στις εξόδους τους τα δεδομένα των καταχωρητών που θέλουμε.

Για την ευκολότερη διασύνδεση μεταξύ των modules υλοποιήσαμε ένα BUS_PKG όπου δηλώνουμε τύπους σημάτων που είναι arrays. Για παράδειγμα αυτό μας

χρειάστηκε στους πολυπλέκτες εξόδου όπου θέλαμε 32 εισόδους των 32bit. Θα μπορούσαμε να χρησιμοποιήσουμε κάποια δομή generate στο port map για να έχουμε το ίδιο αποτέλεσμα, αλλά η χρήση package είναι πιο απλή.

```
package BUS_PKG is
        type BUS_32     is array (31 downto 0)    of STD_LOGIC_VECTOR (31 downto 0);
        type BUS_16     is array (15 downto 0)    of STD_LOGIC_VECTOR (31 downto 0);
        type BUS_8      is array (7 downto 0)     of STD_LOGIC_VECTOR (31 downto 0);
        type BUS_4      is array (3 downto 0)     of STD_LOGIC_VECTOR (31 downto 0);
        type BUS_2      is array (1 downto 0)     of STD_LOGIC_VECTOR (31 downto 0);
end package;
```

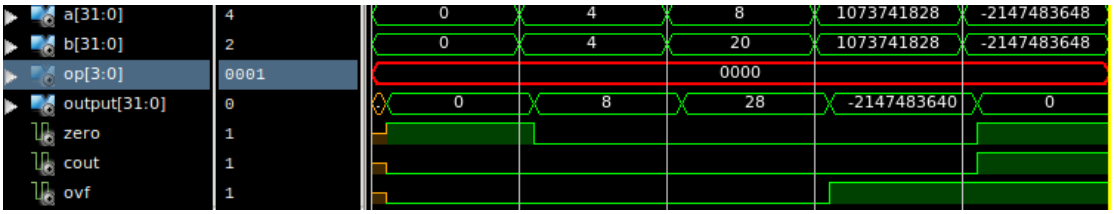
Επειδή θέλουμε ο καταχωρητής **R0** να είναι πάντα **0 (MIPS)**, μπορούσαμε ή να το παραλείψουμε εντελώς και να εκχωρούμε το μηδέν στο input(0) των Mux ή να *απενεργοποιήσουμε* μόνιμα το *WrEn του R0*. Επιλέξαμε την δεύτερη επιλογή, επειδή την θεωρήσαμε σχεδιαστικά ορθότερη.

Κυματομορφές

Εκτός από την ορθή λειτουργία των πράξεων μπορούμε να παρατηρήσουμε και τις καθυστερήσεις των 10ns που προστέθηκαν στην έξοδο.

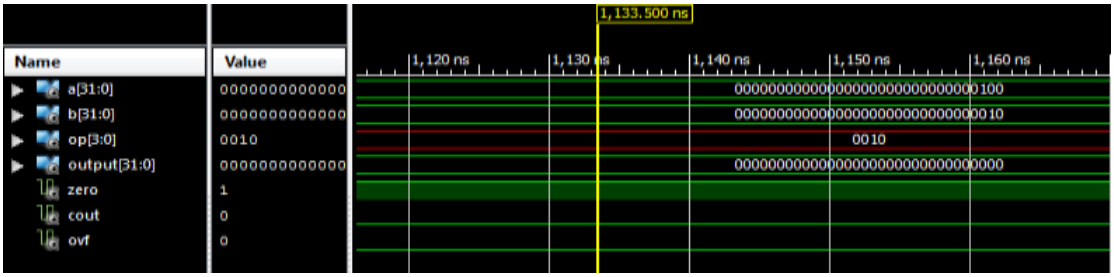
1ο Μέρος:

1. Addition:

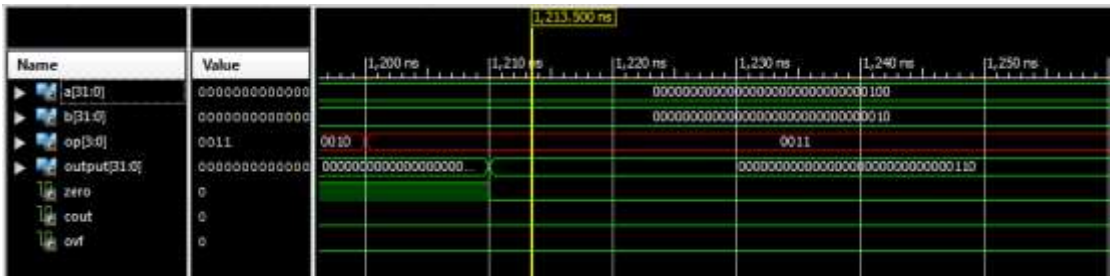


2. Sub:

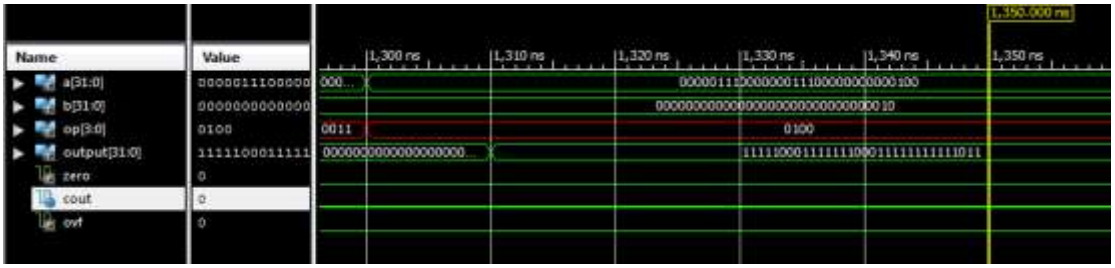
3. AND:



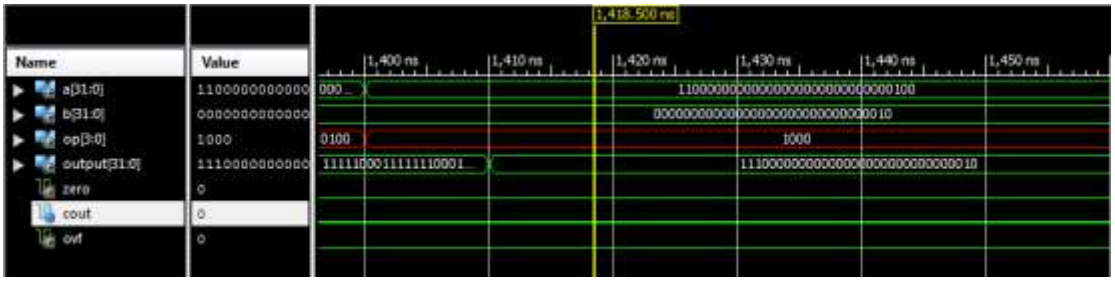
4. OR:



5. NOT:

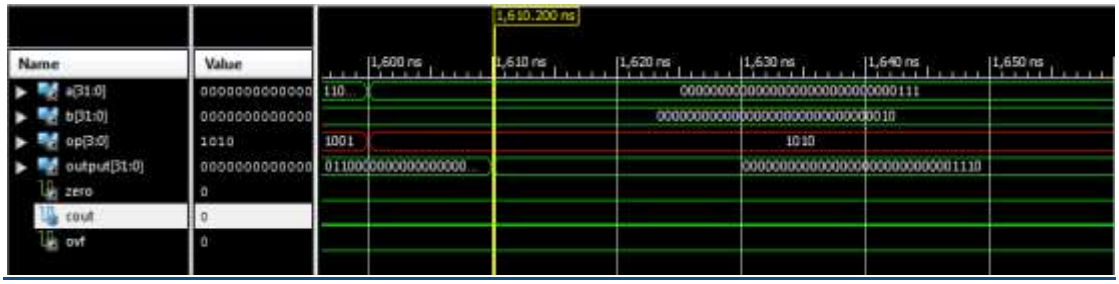


6. RIGHT ARITHMETIC SHIFT:

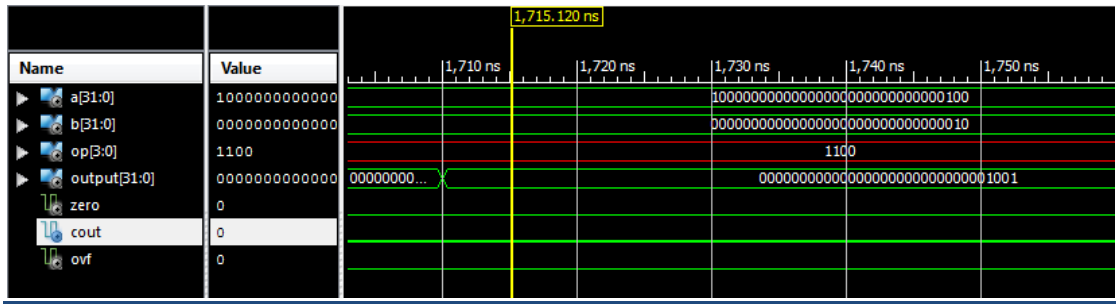


7. RIGHT LOGICAL SHIFT:

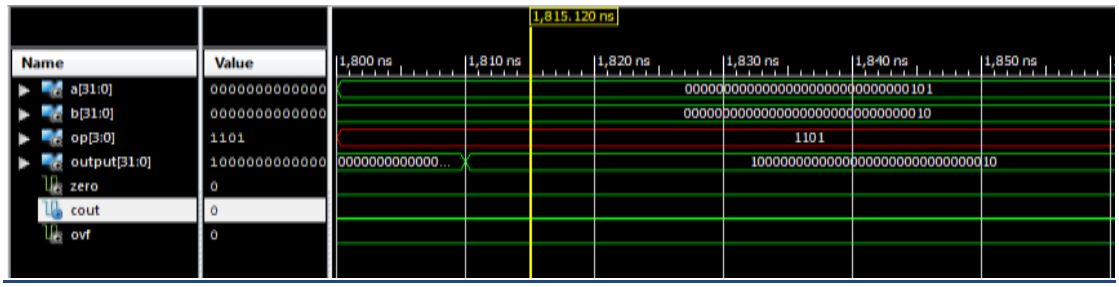
8. LEFT LOGICAL SHIFT:



9. LEFT ROTATE:



10. RIGHT ROTATE:



2ο Μέρος:

WRITE: R1=33
R2=36
R3=39
R1=45
R14=48
R0=51

Όπως φαίνεται στο simulation παρακάτω όλες οι τιμές γράφονται σωστά στους καταχωρητές και μπορούμε να παρατηρήσουμε ότι η έξοδος έχει την καθυστέρηση των 10ns που μας είχε ζητηθεί. Τέλος ο καταχωρητής R0 δεν γράφει την τιμή 51 και είναι πάντα 0.