

Αναφορά Εργαστηρίου 1

Οργάνωση Υπολογιστών

Ομάδα Εργασίας: Κριθαράκης Εμμανουήλ, Φωτάκης Τζανής

Σύντομη περιγραφή

Το εργαστήριο χωρίστηκε σε 2 τμήματα. Στο πρώτο οφείλαμε να υλοποιήσουμε μια ALU ,η οποία εκτελεί συγκεκριμένες πράξεις (10 στο πλήθος) και να εξάγουμε το τελικό αποτέλεσμα καθώς και αλλά 3 σήματα εξόδου ονόματι Zero, Cout (Carry Out) και Onf (Overflow). Στο δεύτερο μέρος οφείλαμε να σχεδιάσουμε ένα register file, κάνοντας χρήση 32 registers 32 bit έκαστος, με τρεις θύρες δύο για ανάγνωση και μια για εγγραφή.

Επισκόπηση

Μέρος Α: A.L.U.

Στο πρώτο μέρος της εργαστηριακής άσκησης ζητήθηκε μια ALU (arithmetic logic unit) η οποία έπρεπε να εκτελεί 10 ξεχωριστές αριθμητικές πράξεις. Συγκεκριμένα, ανάλογα με το control unit (η είσοδος opcode) δίνονταν εντολή για την εκτέλεση μιας εκ των δέκα πράξεων. Οι πράξεις αυτές ήταν οι εξής:

Κωδικός	Πράξη
0000	Πρόσθεση
0001	Αφαίρεση
0010	Λογικό “ΚΑΙ”
0011	Λογικό “Η”
0100	Αντιστροφή του A
1000	Αριθμητική ολίσθηση δεξιά κατά 1 θέση
1001	Λογική ολίσθηση δεξιά κατά 1 θέση
1010	Λογική ολίσθηση αριστερά κατά 1 θέση
1100	Κυκλική ολίσθηση αριστερά κατά 1 θέση
1101	Κυκλική ολίσθηση δεξιά κατά 1 θέση

Η υλοποίηση της παραπάνω ALU υλοποιήθηκε από τις γνώσεις μας στην προχωρημένη λογική σχεδίαση. Αναλυτικότερα, κάθε πράξη αποτέλεσε ένα διαφορετικό module, τα οποία και συνενώθηκαν στο τέλος στο top module όπου και πήραν τις κατάλληλες τιμές τους οι τιμές εξόδων: output(32 bits),Zero(1 bit),Cout(1 bit) και Ovf(1 bit).Ως είσοδοι δόθηκαν τα εξής σήματα : A(32 bits),B(32 bits) και Op(4-bits).Η διασύνδεση στο top module ακολούθησε την λογική που παρουσιάζεται στο παρακάτω διάγραμμα. Συγκεκριμένα ανάλογα την τιμή του opcode που δίνονταν ως τιμή στο control ενός 16 προς 1 πολυπλέκτη προέκυψε η έξοδος για κάθε ένα από τα 32 bits της εξόδου του αποτελέσματος (output).Η επιλογή των 32 πολυπλεκτών (16 προς 1) βασίστηκε πάνω στην λογική πώς δεδομένου του κωδικού πράξης θα μπορώ άμεσα να αποδώσω σε οποιοδήποτε από τα 32 bits εξόδου το αποτέλεσμα του. Επιπρόσθετα,η επιλογή των 16 bits για εύρος εισόδου πολυπλέκτη επιλέχθηκε για την κάλυψη των 10 διαφορετικών πράξεων των 2 τελεστών.(οι 6 υπόλοιπες θέσεις συμπληρώθηκαν με το λογικό μηδέν).

Σήματα εξόδου

Output(32 bits):Προέκυψε όπως προαναφέρθηκε στην προηγούμενη παράγραφο.

Zero(1 bit): Εφόσον η έξοδος μας ήταν μηδέν τότε το Zero ισούταν με 1 αλλιώς με 0.

Το Zero μπορούσε να παρατηρηθεί σε όλες τις πράξεις.

Ovf(1 bit): Εφόσον στην έξοδος υπήρχε overflow τότε το Ovf ισούταν με 1 αλλιώς με 0.

Το Ovf μπορούσε να παρατηρηθεί μόνο στις πράξεις της πρόσθεσης και αφαίρεσης.

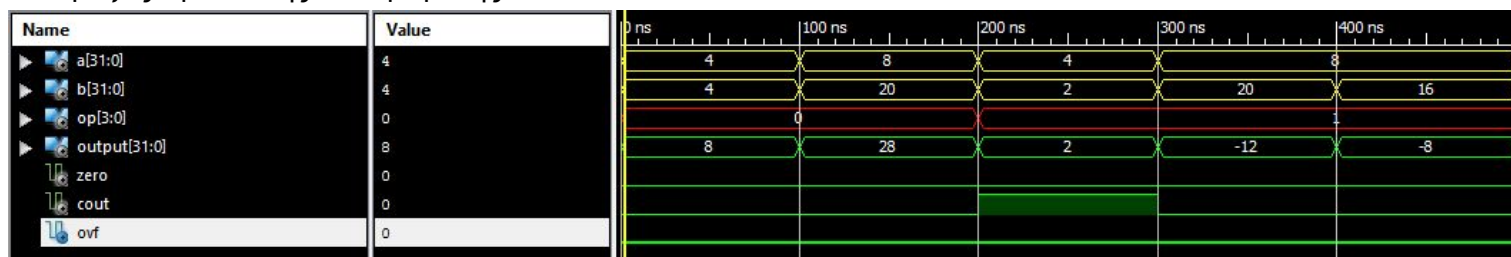
Cout(1 bit) : Εφόσον η έξοδος μας ήταν μηδέν τότε το Cout ισούταν με 1 αλλιώς με 0.

Το Cout μπορούσε να παρατηρηθεί μόνο στις πράξεις της πρόσθεσης και αφαίρεσης.








Simulation








Στις παρακάτω εικόνες αποδίδουμε τα αποτελέσματα του simulation για κάθε διαφορετική πράξη:

Για πράξεις πρόσθεσης και αφαίρεσης:

















Για overflow στην πράξη της πρόσθεσης:

Name	Value
 a[31:0]	10000000000000000000000000000000
 b[31:0]	10000000000000000000000000000000
 op[3:0]	0000
 output[31:0]	00000000000000000000000000000000
 zero	1
 cout	1
 ovf	1

Name	Value
 a[31:0]	01000000000000000000000000000100
 b[31:0]	01000000000000000000000000000100
 op[3:0]	0000
 output[31:0]	10000000000000000000000000001000
 zero	0
 cout	0
 ovf	1

Για overflow στην πράξη της αφαίρεσης:

Name	Value
 a[31:0]	10000000000000000000000000000000
 b[31:0]	01111111111111111111111111111111
 op[3:0]	0001
 output[31:0]	00000000000000000000000000000001
 zero	0
 cout	1
 ovf	1

Name	Value
 a[31:0]	0100000000000000000000000000100
 b[31:0]	10111111111111111111111111111111
 op[3:0]	0001
 output[31:0]	1000000000000000000000000000101
 zero	0
 cout	0
 ovf	1

Για λογικό “H”:

[illegible]

Για λογικό “ΚΑΙ”:

[illegible]

Για αντιστροφή του A:

Name	Value
a[31:0]	000001110000000011100000000000100
b[31:0]	00000000000000000000000000000010
op[3:0]	0100
output[31:0]	111110001111110001111111111011
zero	0
cout	0
ovf	0

Για αριθμητική ολίσθηση δεξιά κατά 1 θέση:

[illegible]

Για λογική ολίσθηση δεξιά κατά 1 θέση:

[illegible]

Για λογική ολίσθηση αριστερά κατά 1 θέση:

[illegible]

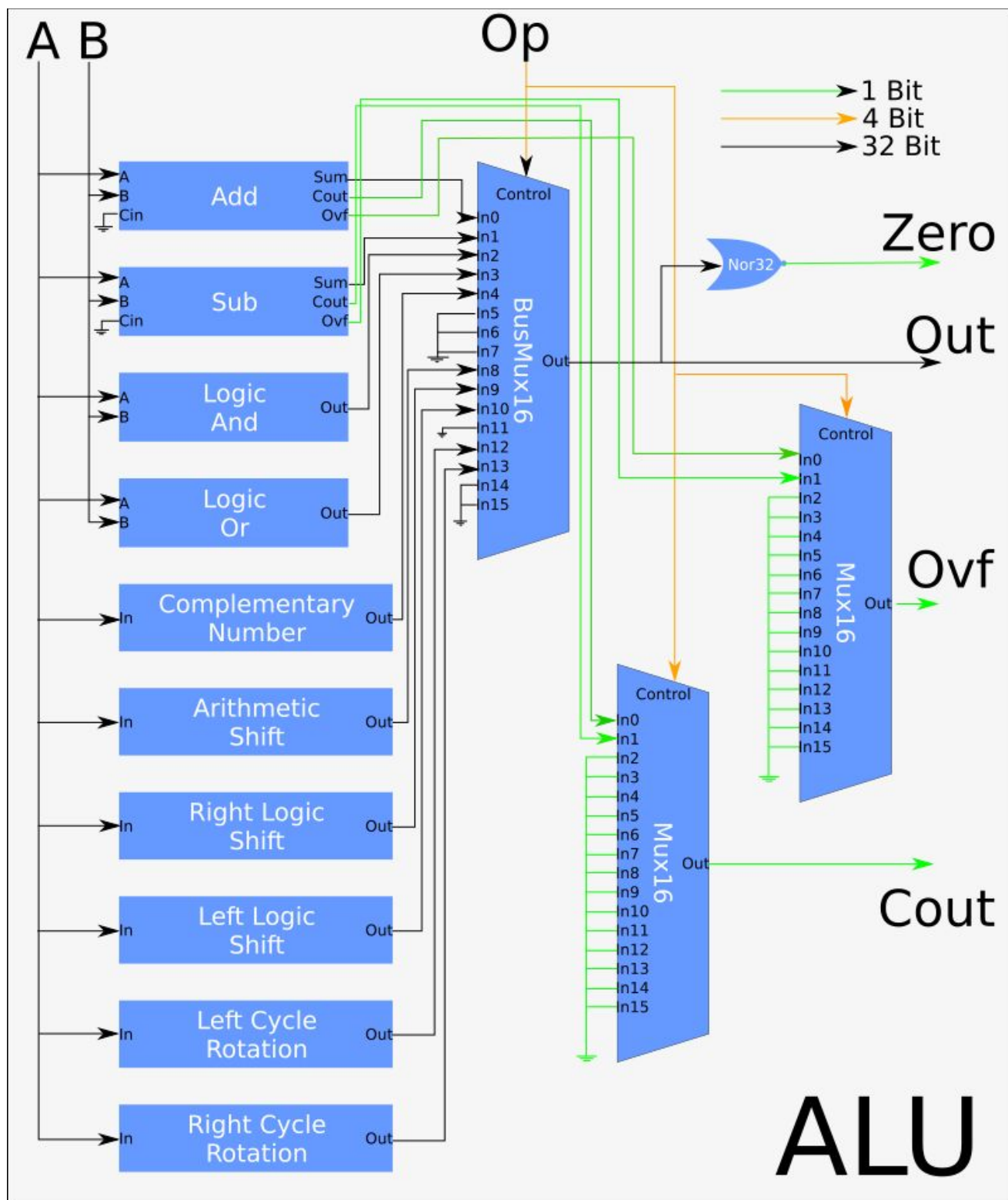
Για κυκλική ολίσθηση αριστερά κατά 1 θέση:

[illegible]

Για κυκλική ολίσθηση δεξιά κατά 1 θέση:

[illegible]

(Οι παραπάνω εικόνες αποδίδονται μόνο με το name και value για να είναι εμφανές το αποτέλεσμα στο 32bits αποτέλεσμα.) Ακολουθεί η σχηματική αναπαράσταση του κυκλώματος της ALU.



Μέρος B: Register File

Το μέρος αυτό έχει σκοπό την δημιουργία ενός Αρχείου Καταχωρητών, εκείνων δηλαδή που χρησιμοποιούνται από τον επεξεργαστή για την αποθήκευση αλλά και ανάκτηση δεδομένων προς επεξεργασία μέσω των βασικών εντολών μηχανής.

Για την σχεδίαση της register file χρησιμοποιήθηκαν οι εξής είσοδοι:

Ard1(5 bits):Διεύθυνση πρώτου καταχωρητή για ανάγνωση

Ard2(5 bits):Διεύθυνση δεύτερου καταχωρητή για ανάγνωση

Awr(5 bits):Διεύθυνση καταχωρητή για εγγραφή

Din(32 bits):Δεδομένα για εγγραφή

WrEn(1 bit):Ενεργοποίηση Εγγραφής καταχωρητή

Clk(1 bit):Πολύ

Για έξοδο:

Dout1(32 bits):Δεδομένα πρώτου καταχωρητή προς ανάγνωση με διεύθυνση Ard1

Dout2(32 bits):Δεδομένα δεύτερου καταχωρητή προς ανάγνωση με διεύθυνση Ard2

Κατά την υλοποίηση χρησιμοποιήθηκαν οι γνώσεις μας από την προχωρημένη λογική σχεδίαση.

Συγκεκριμένα,πραγματοποιήθηκαν τα modules: **Decoder**, **Register32**, **BusMux32**, **Comparator** .Τα modules αυτά συνδέθηκαν όλα στο top_module ονόματι RegisterFile στο οποίο τόσο οι ενδιάμεσοι παραγόμενοι έξοδοι των modules όσο και οι είσοδοι στο top module χρησιμοποιήθηκαν για την παραγωγή του αρχείου καταχωρητών.

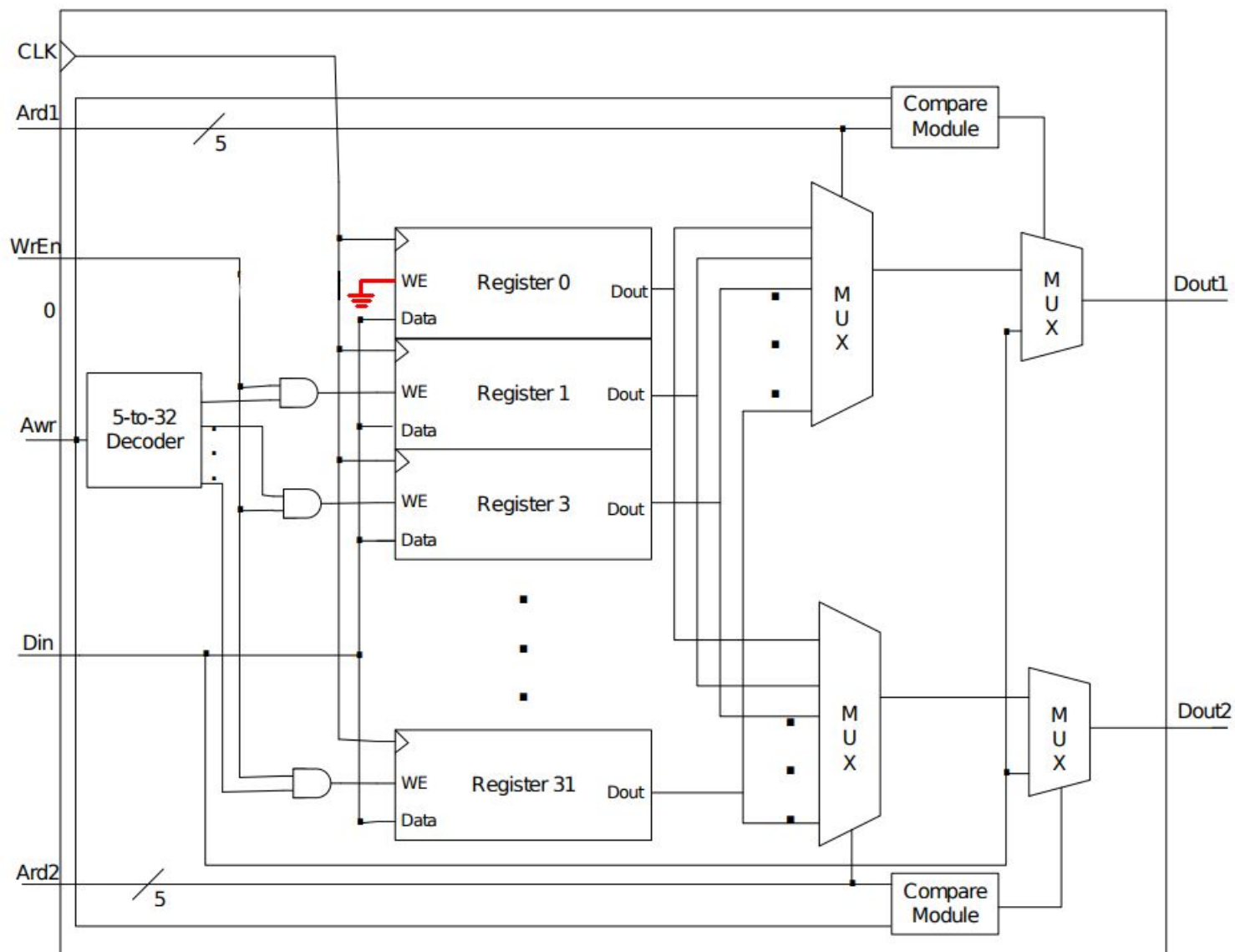
Ο ζητούμενος 32bit register αποτελείται όπως έχουμε διδαχθεί από τα προηγούμενα έτη των σπουδών μας από ένα D-FlipFlop και έναν πολυπλέκτη δύο εισόδων και μίας εξόδου για κάθε bit του register. Η χρήση, λοιπόν, 32 από εκείνους μας συνθέτει το βασικό μέσω αποθήκευσης πληροφορίας του επεξεργαστή. Το Αρχείο Καταχωρητών, παρόλαυτα, ως εξάρτημα διαθέτει και άλλα modules όπως προαναφέρθηκε τα οποία κυρίως καθορίζουν την ορθή λειτουργία της ανάγνωσης και της εγγραφής.

Για την διευκόλυνση της ανάπτυξης του απαιτούμενου κώδικα για την δημιουργία του εξαρτήματος αυτού χρησιμοποιήθηκαν δομές επανάληψης (κυρίως για την δημιουργία και διασύνδεση των 32 registers και των 32 σημάτων Write-Enable (WrEn) προς αυτούς. Επίσης, έγινε η δημιουργία ενός πακέτου το οποίο περιέχει κάποιους δικούς μας τύπους εισόδων και εξόδων, πιο συγκεκριμένα πίνακες από std_logic_vector(31 downto 0) κυρίως για να γίνεται πιο εύκολη η διασύνδεση μεταξύ των διαφόρων εξαρτημάτων.

Το ερώτημα που τίθεται είναι το πώς μπορεί να εξασφαλιστεί η μη εγγραφιμότητα του register 0 αφού αυτός αποτελεί μία σταθερά με τιμή 0. Η απάντηση στο ερώτημα αυτό είναι απλή αφού αρκεί να θέσουμε το σήμα WrEn του register 0 στην τιμή μηδέν ως σταθερά δηλαδή συνδέοντάς το με την γείωση. Μία καλύτερη και οικονομικότερη λύση θα ήταν να μην χρησιμοποιούνταν κανένας register αλλά αντιθέτως την θέση του να έπαιρνε μία σταθερά μηδέν ως είσοδο του πολυπλέκτη, δηλαδή η γείωση του πολυπλέκτη στην είσοδο 0.

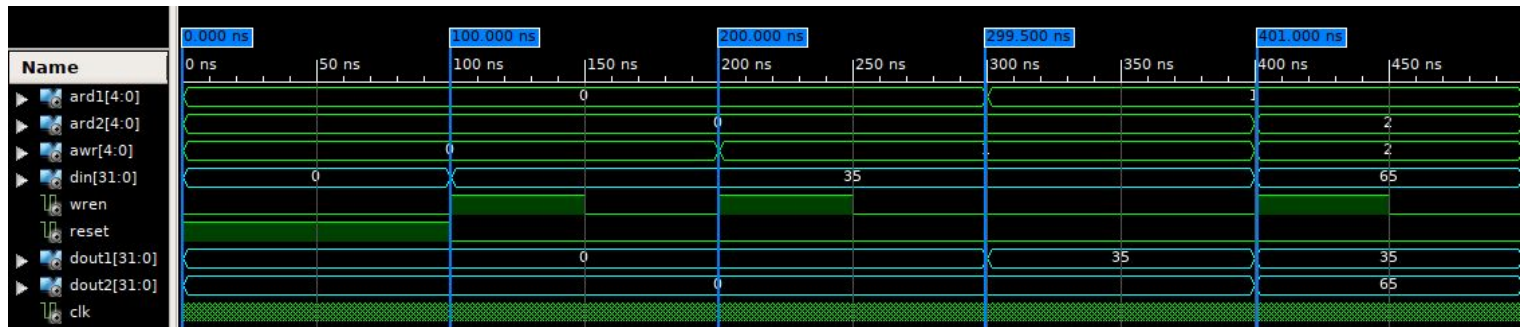
Για την πιθανή επαναχρησιμοποίηση του κώδικα και των διαφόρων εξαρτημάτων, κυρίως του 32 bit Register, του Register File και του D-FlipFlop, έγινε η ενσωμάτωση ενός σήματος reset, ενεργό στο 1, το οποίο κατά την ενεργοποίησή του θέτονται οι εξόδοι των παραπάνω εξαρτημάτων στο μηδέν.

Παρακάτω ακολουθεί το σχεδιάγραμμα του κυκλώματος του Αρχείου Καταχωρητών.



Simulation

Στην παρακάτω εικόνα αποδίδουμε τα αποτελέσματα του simulation του register file:



- 0-100 ns: Γίνεται το reset του κυκλώματος.
- 100-200 ns: Γίνεται μία ανεπιτυχής προσπάθεια να φορτωθεί στον καταχωρητή 0 μία τιμή διαφορετική του μηδενός. Παρατηρούμε ότι σωστά όπως ορίζονται από τις προδιαγραφές η τιμή του δεν αλλάζει.
- 200-300 ns: Φορτώνεται στον καταχωρητή με διεύθυνση 1 η τιμή 35.
- 300-400 ns: Θέτεται η πρώτη διεύθυνση ανάγνωσης στην τιμή 1 και όπως ήταν αναμενόμενο από την πρώτη έξοδο λαμβάνουμε την τιμή που φορτώθηκε στον καταχωρητή 1 στο αμέσως προηγούμενο βήμα.
- 400-500 ns: Η προσομοίωση του κυκλώματος τελειώνει με την ταυτόχρονη φόρτωση και ανάγνωση του ίδιου καταχωρητή της τιμής 65 η οποία και παρουσιάζεται στην έξοδο όπως ζητείται από τις προδιαγραφές.