ARISTOTLE UNIVERSITY OF THESSALONIKI

Aristotle University of Thessaloniki
Department of Electrical and Computer Engineering

Καράτης Δημήτριος 10775, *karatisd@ece.auth.gr*

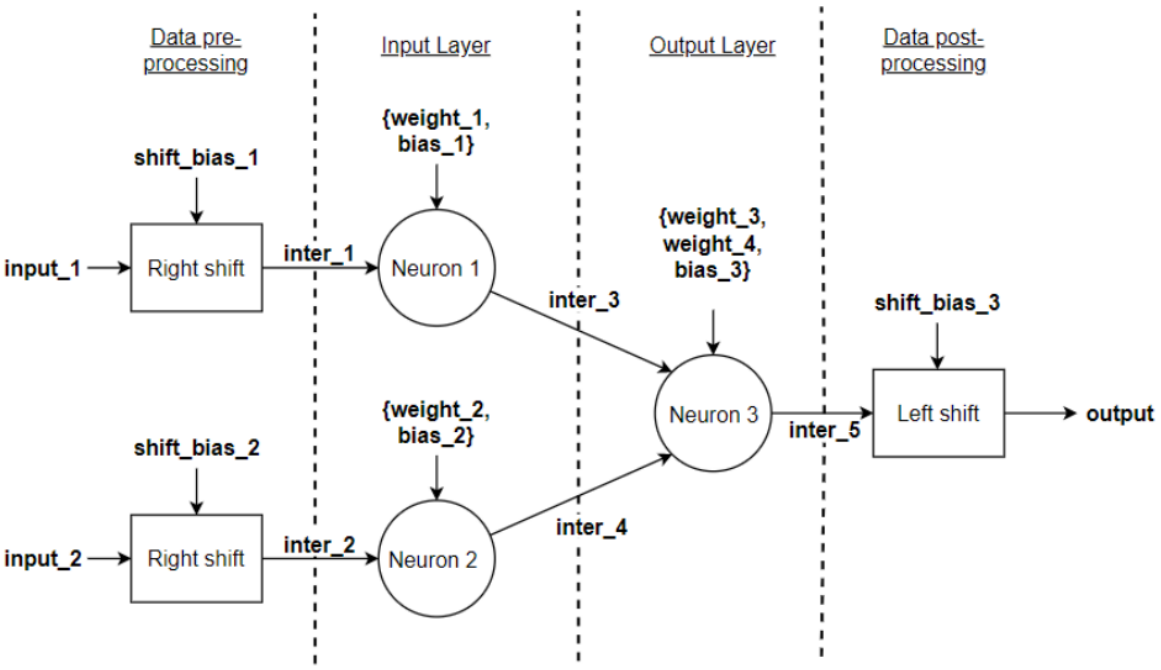
ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ HARDWARE ΣΕ ΧΑΜΗΛΑ ΕΠΙΠΕΔΑ ΛΟΓΙΚΗΣ Ι

Εργασία (2025-2026)

Περιεχόμενα

1 Εισαγωγή

Στο πλαίσιο της παρούσας εργασίας, υλοποιείται ο σχεδιασμός και η επαλήθευση ενός ψηφιακού συστήματος επιτάχυνσης νευρωνικών δικτύων (AI Accelerator). Το σύστημα βασίζεται σε μια αρχιτεκτονική επεξεργασίας δεδομένων που εκτελεί αριθμητικές και λογικές πράξεις σε περιβάλλον σταδίων, χρησιμοποιώντας μια κεντρική Μονάδα Ελέγχου (Control Unit) βασισμένη σε Μηχανή Πεπερασμένων Καταστάσεων (FSM).



Σχήμα 1: Απλό νευρωνικό σύστημα με τρεις νευρώνες.

Η εργασία αναπτύσσεται κλιμακωτά μέσω των εξής επιμέρους ενοτήτων:

- **Υλοποίηση Arithmetic Logic Unit (ALU):** Σχεδιάζεται μια μονάδα 32-bit υπεύθυνη για την εκτέλεση των βασικών πράξεων, όπως πρόσθεση, αφαίρεση, και προσημασμένο πολλαπλασιασμό. Η ALU ενσωματώνει μηχανισμούς ανίχνευσης μηδενικού αποτελέσματος (zero) και υπερχείλισης (overflow), εξασφαλίζοντας την εγκυρότητα των υπολογισμών.
- **Σχεδίαση Ψηφιακής Αριθμομηχανής (Calculator):** Δημιουργείται ένα σύγχρονο κύκλωμα που αξιοποιεί την ALU για την ενημέρωση ενός συσσωρευτή (accumulator) 16-bit. Η μονάδα ελέγχεται από εξωτερικά πλήκτρα εισόδου, ενώ η λογική επιλογής των πράξεων υλοποιείται μέσω δομικής Verilog (structural verilog).
- **Ανάπτυξη Αρχείου Καταχωρητών (Register File):** Υλοποιείται μια μονάδα αποθήκευσης 16 καταχωρητών των 32 bits. Το αρχείο αυτό είναι απαραίτητο για τη διατήρηση των βαρών (weights) και των

πολώσεων (biases) του δικτύου, υποστηρίζοντας πολλαπλές θύρες ταυτόχρονης ανάγνωσης για την παραλληλοποίηση των πράξεων.

- **Σύνθεση Νευρωνικού Συστήματος (Neural Network):** Το τελικό στάδιο περιλαμβάνει τη σύνθεση των παραπάνω μονάδων σε ένα ολοκληρωμένο σύστημα τριών νευρώνων. Το σύστημα εκτελεί πράξεις MAC (Multiply and Accumulate), επεξεργάζεται τα δεδομένα εισόδου και εξόδου μέσω ολισθήσεων (pre/post-processing) και ελέγχεται από ένα FSM 7 καταστάσεων.

Σημαντική παράμετρος της σχεδίασης είναι η διαχείριση της υπερχείλισης (overflow) σε οποιοδήποτε στάδιο, η οποία οδηγεί το σύστημα σε άμεση μετάβαση σε κατάσταση αδράνειας (IDLE) και στην απόδοση του μέγιστου δυνατού θετικού αριθμού στην έξοδο. Η επαλήθευση της σωστής λειτουργίας πραγματοποιείται μέσω testbench και σύγκρισης με μαθηματικό μοντέλο αναφοράς.

1.1 Περιβάλλον Ανάπτυξης και Ροή Εργασίας

Η σχεδίαση και η επαλήθευση του συστήματος βασίστηκαν σε μια ολοκληρωμένη ροή εργασίας με χρήση εργαλείων ανοιχτού κώδικα. Συγκεκριμένα:

- **Συγγραφή και Διαχείριση:** Χρησιμοποιήθηκε ο επεξεργαστής κειμένου Visual Studio Code με το extension TerosHDL, το οποίο επέτρεψε την αυτόματη παραγωγή τεκμηρίωσης (module documentation previews) και την οπτική ανάλυση της ιεραρχίας των modules.
- **Μεταγλώττιση (Compilation):** Για τη σύνταξη και τον έλεγχο ορθότητας του κώδικα χρησιμοποιήθηκε ο compiler Icarus Verilog, ο οποίος εκτέλεσε τη μεταγλώττιση και παρήγαγε τα απαραίτητα αρχεία για την προσομοίωση.
- **Προσομοίωση και Ανάλυση:** Η εκτέλεση της προσομοίωσης έγινε μέσω του εργαλείου VVP (μέρος του Icarus Verilog toolset), ενώ η ανάλυση των διαγραμμάτων χρονισμού πραγματοποιήθηκε με χρήση του VS Code extension: WaveTrace, επιτρέποντας την γρήγορη επαλήθευση της λογικής συμπεριφοράς των σημάτων στο περιβάλλον Linux που δούλεψα.

1.2 Παράδειγμα Μεταγλώττισης (Compilation) της μονάδας ALU

Ενδεικτικά, η διαδικασία παραγωγής του εκτελέσιμου αρχείου προσομοίωσης για την μονάδα ALU (και κατ΄ επέκταση και για τις υπόλοιπες) πραγματοποιήθηκε μέσω του τερματικού με την ακόλουθη σειρά εντολών:

- **Δημιουργία Εκτελέσιμου**: `iverilog -o alu_sim alu_tb.v alu.v`
- **Εκτέλεση Προσομοίωσης**: `vvp alu_sim`

Η πρώτη εντολή συνθέτει τα απαραίτητα αρχεία πηγαίου κώδικα (`.v`) σε ένα αρχείο εξόδου με όνομα `alu_sim`, ενώ η δεύτερη εκκινεί τον προσομοιωτή VVP για την παραγωγή των αποτελεσμάτων και του αρχείου κυματομορφών (`.vcd`).

Αξίζει να σημειωθεί ότι δεν απαιτείται εκ νέου μεταγλώττιση των αρχείων, καθώς μαζί με τον πηγαίο κώδικα (`.v`) παρατίθενται και τα αντίστοιχα έτοιμα αρχεία προσομοίωσης (πχ. `alu_sim, calc_sim`, κλπ.) καθώς και τα αρχεία κυματομορφών (`.vcd`) για μεγαλύτερη ευκολία στην αξιολόγηση.

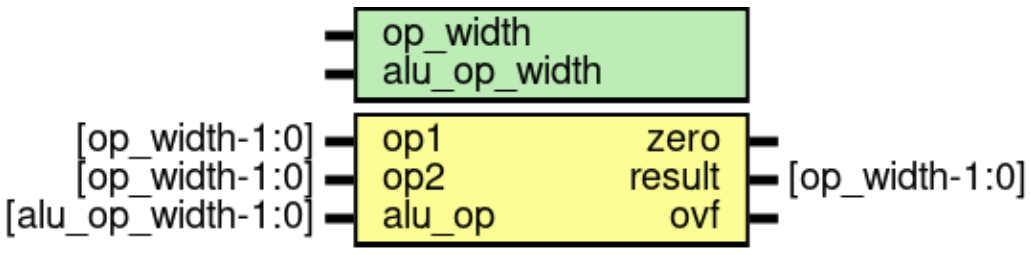
2 Μονάδα ALU (Άσκηση 1)

Σκοπός της Άσκησης 1 είναι η σχεδίαση μιας Αριθμητικής και Λογικής Μονάδας (Arithmetic Logic Unit - ALU) πλάτους 32-bit. Η μονάδα αυτή αποτελεί τον πυρήνα επεξεργασίας του συστήματος, καθώς είναι υπεύθυνη για την εκτέλεση όλων των αριθμητικών πράξεων (πρόσθεση, αφαίρεση, πολλαπλασιασμός), των λογικών πράξεων (AND, OR, XOR κ.λπ.) και των λειτουργιών ολίσθησης (shifting). Η συγκεκριμένη ΑΛΥ σχεδιάστηκε ως ένα αμιγώς συνδυαστικό κύκλωμα, το οποίο θα αποτελέσει το βασικό δομικό στοιχείο για την υλοποίηση της αριθμομηχανής στην Άσκηση 2, καθώς και των μονάδων MAC (Multiply and Accumulate) του νευρωνικού επιταχυντή στην Άσκηση 4.

2.1 Αρχιτεκτονική Σχεδίασης και Μεθοδολογία

Η ALU σχεδιάστηκε με παραμετρικό εύρος δεδομένων (32-bit). Χρησιμοποιήθηκε μια δομή `case` η οποία λειτουργεί ως πολυπλέκτης, επιλέγοντας την έξοδο ανάλογα με το σήμα `alu_op` (`alu.v`).

- **Αριθμητικές Πράξεις:** Υλοποιήθηκαν οι ADD, SUB και MUL με χρήση της βιβλιοθήκης `$signed` για σωστό χειρισμό αριθμών συμπληρώματος ως προς 2.
- **Υπερχείλιση (Overflow):** Για την πρόσθεση και αφαίρεση, η υπερχείλιση ανιχνεύεται από τη σύγκριση των MSB των εισόδων και του αποτελέσματος. Στον πολλαπλασιασμό, ελέγχεται αν το αποτέλεσμα παραμένει εντός των 32-bit ορίων.
- **Ολισθήσεις:** Υλοποιήθηκαν λογικές και αριθμητικές ολισθήσεις (SRL, SLL, SRA, SLA).



Σχήμα 2: Σχηματική αναπαράσταση της μονάδας `ALU`.
(Black Box diagram με την βοήθεια του εργαλείου TerosHDL στο VS Code)

2.2 Προσομοίωση και Επαλήθευση Λειτουργίας

2.2.1 Testbench και Σενάρια Ελέγχου

Για την επαλήθευση της ορθότητας της σχεδίασης αναπτύχθηκε ένα *testbench* (`alu_tb.v`), το οποίο προσομοιώνει τη λειτουργία της μονάδας υπό διαφορετικές συνθήκες φορτίου. Η επαλήθευση δεν περιορίστηκε σε απλούς υπολογισμούς, αλλά επικεντρώθηκε σε κρίσιμα σενάρια λειτουργίας:

- **Αριθμητική Ακρίβεια:** Ελέγχθηκαν οι βασικές πράξεις (ADD, SUB, MUL) με προσημασμένους αριθμούς για την επιβεβαίωση της σωστής υλοποίησης του συμπληρώματος ως προς 2.
- **Ανίχνευση Υπερχείλισης (Overflow):** Δοκιμάστηκαν οριακές τιμές (π.χ. `32'h7FFFFFFF + 1`) ώστε να επιβεβαιωθεί ότι το σήμα `ovf` ενεργοποιείται σωστά όταν το αποτέλεσμα υπερβαίνει το εύρος αναπαράστασης των 32-bit.
- **Λογικές Πράξεις και Ολισθήσεις:** Ελέγχθηκαν σύνθετες λογικές πύλες (NAND) καθώς και η αριθμητική ολίσθηση δεξιά (SRA). Στην περίπτωση της SRA, δόθηκε έμφαση στη διατήρηση του προσήμου (sign extension) κατά τη μετατόπιση αρνητικών αριθμών.
- **Αυτοματοποιημένος Έλεγχος:** Η χρήση της οδηγίας `$display` επέτρεψε την άμεση καταγραφή των αποτελεσμάτων στο τερματικό (terminal), διευκολύνοντας τη σύγκριση των εξαγόμενων τιμών με τις αναμενόμενες θεωρητικές τιμές.

2.2.2 Ανάλυση Εξόδου Προσομοίωσης

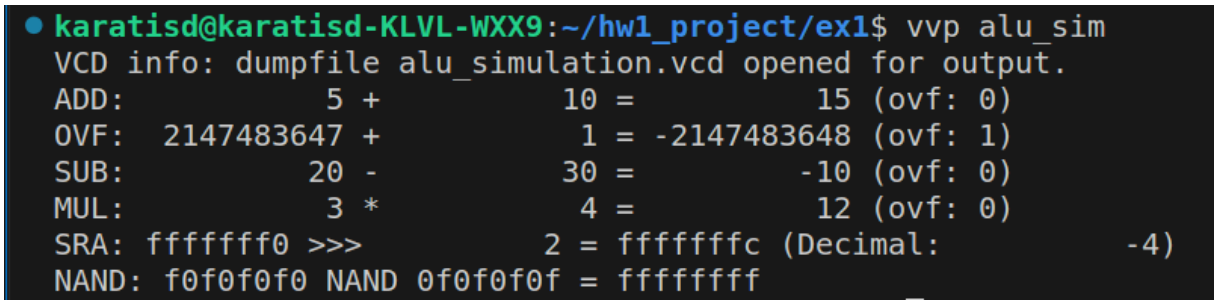
Κατά την εκτέλεση της προσομοίωσης (με χρήση της εντολής `vvp`), ελήφθησαν τα εξής αποτελέσματα:

Πράξη	**Είσοδος A**	**Είσοδος B**	**Αποτέλεσμα**	**Overflow**
ADD	5	10	15	0
ADD	2147483647	1	-2147483648	1
SUB	20	30	-10	0
MUL	3	4	12	0
SRA	fffffff0	2	fffffffc	-
NAND	f0f0f0f0	0f0f0f0f	ffffffff	-

Πίνακας 1: Αποτελέσματα Επαλήθευσης ALU

.

- **Πρόσθεση (ADD):** Η ALU υπολόγισε ορθά το άθροισμα $(5 + 10 = 15)$ χωρίς υπερχείλιση $(ovf = 0)$.



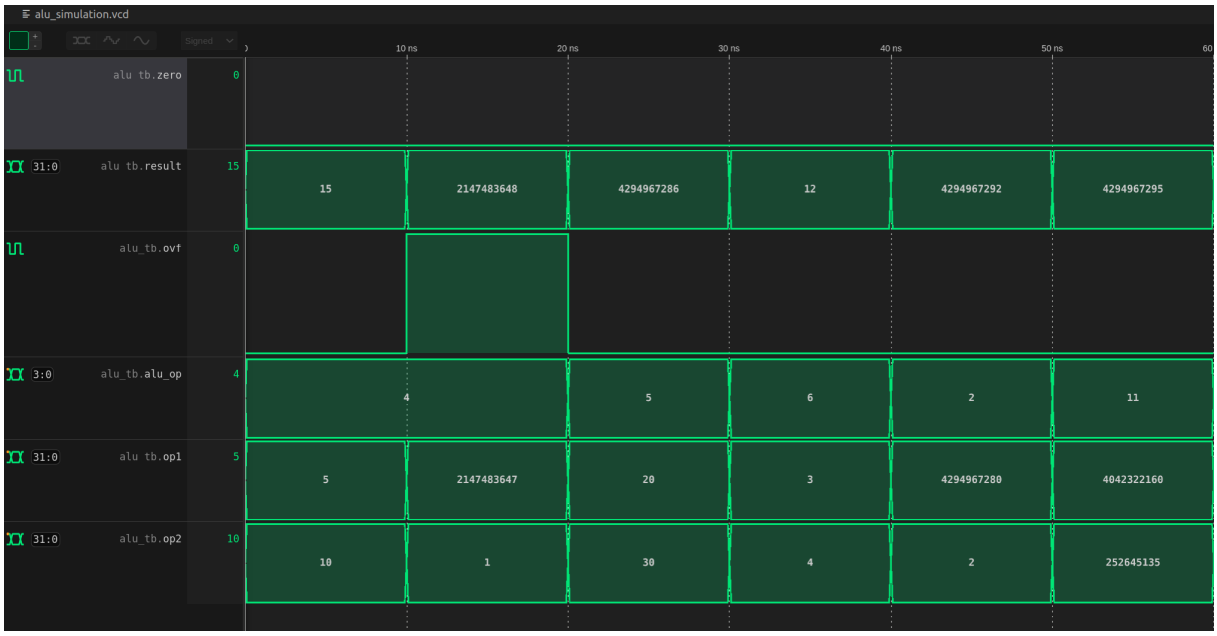
Σχήμα 3: Αποτελέσματα Terminal.

- **Υπερχείλιση (OVF):** Προσθέτοντας 1 στον μέγιστο θετικό αριθμό $2^{31} - 1$ (7FFFFFFF), το αποτέλεσμα έγινε -2^{31} (80000000) λόγω του περιορισμού των 32 bits. Η ALU ανίχνευσε σωστά την αλλαγή προσήμου και ενεργοποίησε τη σημαία `ovf`.
- **Αφαίρεση (SUB):** Η πράξη $20 - 30$ παρήγαγε το αρνητικό αποτέλεσμα -10, αποδεικνύοντας τη σωστή χρήση του συμπληρώματος ως προς 2.
- **Πολλαπλασιασμός (MUL):** Επιβεβαιώθηκε η λειτουργία της ALU με το γινόμενο $3 \times 4 = 12$.
- **Αριθμητική Ολίσθηση (SRA):** Ο αριθμός -16 (`fffffff0`) ολισθήθηκε δεξιά κατά 2 θέσεις. Το αποτέλεσμα `fffffffc` (-4) δείχνει ότι έγινε *sign extension* (διατήρηση του bit προσήμου), κάτι που είναι απαραίτητο για προσημασμένες πράξεις.
- **Λογικό NAND:** Η πράξη εκτελέστηκε bit-προς-bit, επιστρέφοντας την τιμή `ffffffff`, επιβεβαιώνοντας τη λογική πύλη.

2.2.3 Ανάλυση Κυματομορφών Προσομοίωσης

Στις κυματομορφές που παρήχθησαν με τη χρήση του εργαλείου WaveTrace ενδεικτικά παρατηρούμε:

1. **Χρόνος 10ns-20ns:** Η πρόσθεση του 1 στον μέγιστο θετικό αριθμό (7FFFFFFF) οδηγεί σε υπέρβαση του ορίου των 32-bit. Αυτό έχει ως αποτέλεσμα το αποτέλεσμα να ερμηνεύεται ως αρνητικό (λόγω του προσημασμένου συστήματος συμπληρώματος ως προς 2), γεγονός που ενεργοποιεί σωστά το σήμα υπερχείλισης (`ovf`).
2. **Χρόνος 40ns-50ns:** Η αριθμητική δεξιά ολίσθηση (`SRA`) του -16 (`fffffff0`) κατά 2 θέσεις διατηρεί το πρόσημο (σιγν εξτενσιον), δίνοντας αποτέλεσμα -4 (`fffffffc`).



Σχήμα 4: Αποτελέσματα Κυματομορφών του Testbench με χρήση του εργαλείου WaveTrace στο VS Code.

2.3 Συμπεράσματα

Η ALU ανταποκρίνεται πλήρως στις προδιαγραφές της άσκησης. Η χρήση του `$signed` εξασφάλισε τη σωστή λειτουργία των αριθμητικών πράξεων, ενώ το testbench επιβεβαίωσε την ορθή ανίχνευση υπερχείλισης και τη λειτουργία των shifts.

Κώδικες για το ερώτημα αυτό μπορούν να βρεθούν στα αρχεία `alu.v` και `alu_tb.v`, με την αντίστοιχη προσομοίωση να βρίσκεται για ευκολία στο αρχείο με όνομα `alu_simulation.vcd`.

3 Μονάδα CALCULATOR (Άσκηση 2)

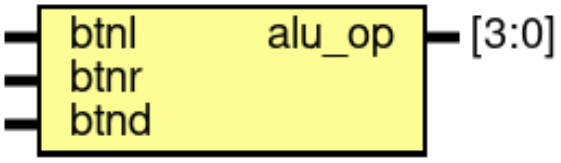
Αντικείμενο της δεύτερης άσκησης είναι η σύνθεση μιας ολοκληρωμένης ψηφιακής αριθμομηχανής 16-bit. Η σχεδίαση βασίζεται στην επαναχρησιμοποίηση της ALU που αναπτύχθηκε στην Άσκηση 1 και την ενσωμάτωσή της σε ένα σύστημα που περιλαμβάνει έναν κωδικοποιητή λειτουργιών, έναν συσσωρευτή (accumulator) και κυκλώματα επέκτασης προσήμου (Sign Extension).

3.1 Αρχιτεκτονική Σχεδίασης και Μεθοδολογία

3.1.1 Δομικός Κωδικοποιητής (calc_enc.v)

Η μονάδα `calc_enc` υλοποιήθηκε με τη μέθοδο της δομικής περιγραφής (structural modeling), χρησιμοποιώντας αποκλειστικά βασικές λογικές πύλες (`not`, `and`, `or`, `xor`). Ο σκοπός της είναι η μετατροπή των σημάτων εισόδου από τα πλήκτρα `btnl`, `btnr` και `btnd` στον 4-bit κωδικό ελέγχου `alu_op`.

- **Υλοποίηση:** Κάθε bit της εξόδου (`alu_op[3:0]`) αντιστοιχεί σε ένα συγκεκριμένο συνδυαστικό κύκλωμα βάσει των Σχημάτων 2 έως 5 της εκφώνησης.



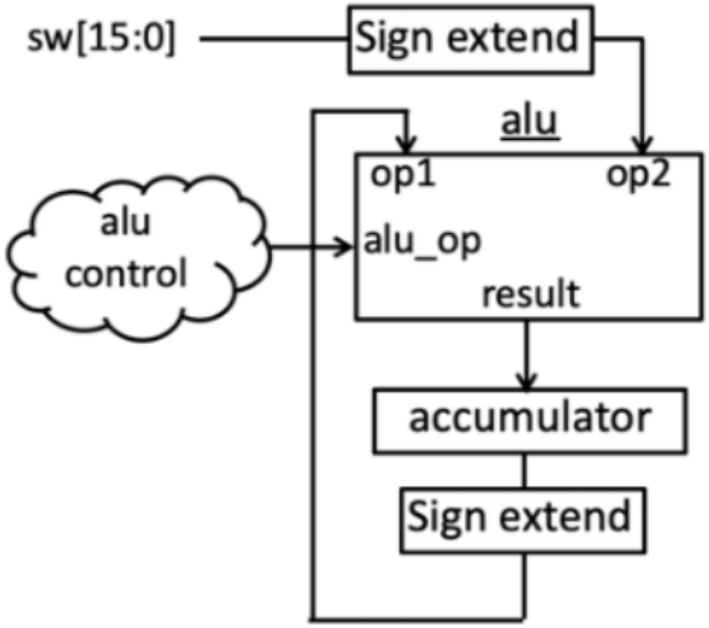
Σχήμα 5: Σχηματική αναπαράσταση της μονάδας `CALC_ENC`.
(Black Box diagram με την βοήθεια του εργαλείου TerosHDL στο VS Code)

3.1.2 Κεντρική Μονάδα Αριθμομηχανής (calc.v)

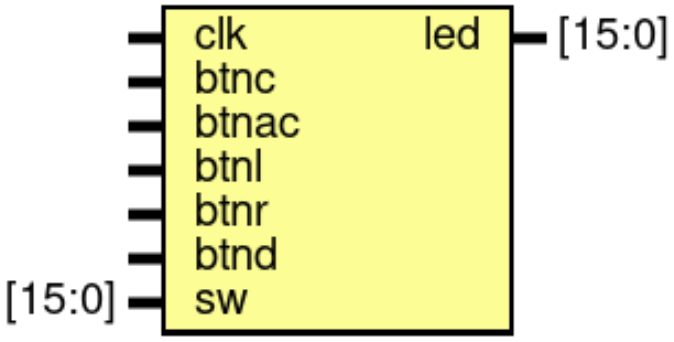
Η μονάδα `calc` αποτελεί το ανώτερο επίπεδο (top-level) της σχεδίασης. Η αρχιτεκτονική της ακολουθεί το διάγραμμα ροής του Σχήματος 6.

1. **Επέκταση Προσήμου (Sign Extension):** Προκειμένου να διασφαλιστεί η συμβατότητα με την 32-bit ALU, οι 16-bit είσοδοι (διακόπτες `sw` και `accumulator`) επεκτείνονται σε 32-bit με χρήση concatenation, αναπαράγοντας το bit προσήμου.
2. **Συσσωρευτής (Accumulator):** Υλοποιήθηκε ένας καταχωρητής 16-bit ο οποίος:
   - Μηδενίζεται σύγχρονα με το πλήκτρο `btnac`.
   - Ενημερώνεται σύγχρονα με το πλήκτρο `btnc`, αποθηκεύοντας τα 16 λιγότερο σημαντικά bit (LSB) του αποτελέσματος της ALU.

3. **Ιεραρχική Σύνδεση:** Η ALU και ο κωδικοποιητής ενσωματώνονται ως sub-modules (`alu_inst`, `encoder_inst`), δημιουργώντας ένα κλειστό σύστημα ανάδρασης (feedback loop).



Σχήμα 6: Διάγραμμα ροής της αριθμομηχανής.



Σχήμα 7: Σχηματική αναπαράσταση της μονάδας `CALC`.
(Black Box diagram με την βοήθεια του εργαλείου TerosHDL στο VS Code)

3.2 Προσομοίωση και Επαλήθευση Λειτουργίας

3.2.1 Testbench και Σενάρια Ελέγχου

Για την επαλήθευση της ορθότητας της αριθμομηχανής χρησιμοποιήθηκε το αρχείο `calc_tb.v`, το οποίο υλοποιεί 9 διαδοχικές δοκιμές που καλύπτουν το πλήρες εύρος των υποστηριζόμενων λειτουργιών. Η στρατηγική ελέγχου βασίστηκε στα εξής σημεία:

- **Διαδοχικοί Υπολογισμοί:** Το testbench ελέγχει την ικανότητα του συσσωρευτή (accumulator) να διατηρεί το αποτέλεσμα μιας πράξης και να το χρησιμοποιεί ως είσοδο για την επόμενη, προσομοιώνοντας την πραγματική χρήση μιας αριθμομηχανής.
- **Έλεγχος Χρονισμού:** Επαληθεύεται ότι η ενημέρωση της τιμής των LED γίνεται σύγχρονα με την ακμή του ρολογιού μόνο όταν πατηθεί το κεντρικό πλήκτρο (`btnc`), ενώ η εκκαθάριση (Reset) μέσω του `btnac` μηδενίζει άμεσα την τρέχουσα τιμή.
- **Επαλήθευση Πράξεων:** Το σενάριο περιλαμβάνει αριθμητικές πράξεις (ADD, SUB, MULT), λογικές (NAND, NOR, XOR) και πράξεις ολίσθησης (SLL, SRL), συγκρίνοντας την έξοδο με προϋπολογισμένες τιμές αναφοράς σε δεκαεξαδική μορφή.

3.2.2 Ανάλυση Εξόδου Προσομοίωσης

Η έξοδος του εργαλείου vvp επιβεβαιώνει την απόλυτη ταύτιση των αποτελεσμάτων με τις αναμενόμενες τιμές:

Βήμα	**Πράξη**	**Είσοδος (sw)**	**Έξοδος (LED)**	**Αναμενόμενο**
1	Reset	-	`0000`	`0000`
2	ADD	`285a`	`285a`	`285a`
3	XOR	`04c8`	`2c92`	`2c92`
4	SRL	`0005`	`0164`	`0164`
5	NOR	`a085`	`5e1a`	`5e1a`
6	MUL	`07fe`	`13cc`	`13cc`
7	SLL	`0004`	`3cc0`	`3cc0`
8	NAND	`fa65`	`c7bf`	`c7bf`
9	SUB	`b2e4`	`14db`	`14db`

Πίνακας 2: Αποτελέσματα Επαλήθευσης Αριθμομηχανής.

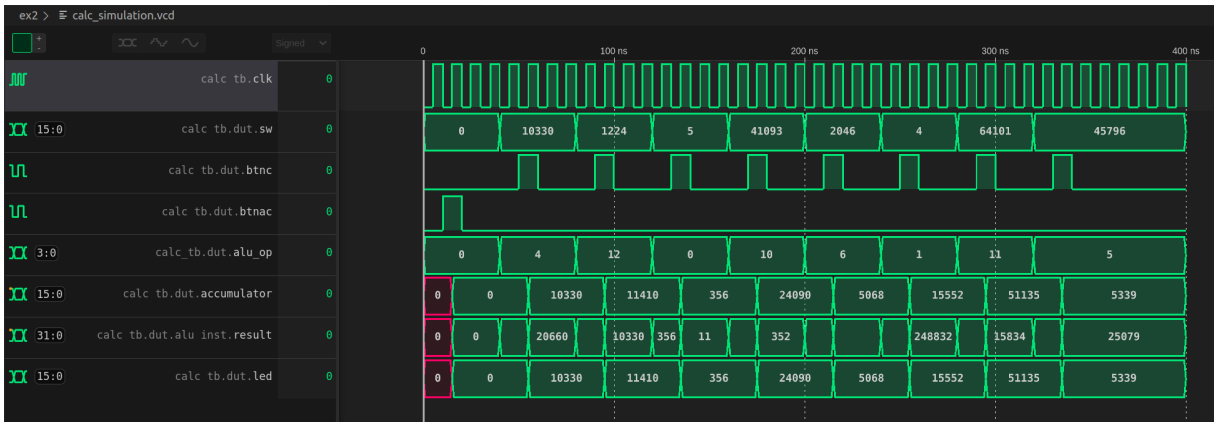
```
karatisd@karatisd-KLVL-WXX9:~/hw1_project/ex2$ vvp calc_sim
VCD info: dumpfile calc_simulation.vcd opened for output.
1. Reset: LED=0000 (Expected: 0000)
2. ADD: LED=285a (Expected: 285a)
3. XOR: LED=2c92 (Expected: 2c92)
4. SRL: LED=0164 (Expected: 0164)
5. NOR: LED=5e1a (Expected: 5e1a)
6. MUL: LED=13cc (Expected: 13cc)
7. SLL: LED=3cc0 (Expected: 3cc0)
8. NAND: LED=c7bf (Expected: c7bf)
9. SUB: LED=14db (Expected: 14db)
```

Σχήμα 8: Αποτελέσματα Επαλήθευσης Αριθμομηχανής μέσω Τερματικού (Terminal).

3.2.3 Ανάλυση Κυματομορφών Προσομοίωσης

Η επαλήθευση της σχεδίασης ολοκληρώθηκε με την ανάλυση των κυματομορφών από το αρχείο `calc_simulation.vcd`. Στο διάγραμμα χρονισμού παρατηρούνται τα εξής κρίσιμα σημεία:

- **Λειτουργία Reset:** Στην αρχή της προσομοίωσης, η ενεργοποίηση του σήματος `btnac` μηδενίζει ακαριαία τον συσσωρευτή (`accumulator`), όπως φαίνεται από τη μετάβαση στην τιμή 0. Ταυτόχρονα, τα LED καθαρίζουν, προετοιμάζοντας το σύστημα για την πρώτη πράξη.
- **Δυναμικός Υπολογισμός και Feedback:** Παρατηρείται ότι η έξοδος της ALU (`alu_inst.result`) μεταβάλλεται συνδυαστικά μόλις αλλάξουν οι διακόπτες `sw` ή ο κωδικός `alu_op`. Ωστόσο, η τιμή αυτή περνά στον συσσωρευτή μόνο κατά την ακμή ανόδου του ρολογιού (`clk`) όταν το σήμα `btnc` είναι ενεργό (high).
- **Παράδειγμα Διαδοχικών Πράξεων:**
  1. Στο πρώτο βήμα μετά το reset, με `sw = 10330` και `alu_op = 4` (ADD), ο συσσωρευτής φορτώνει την τιμή 10330.
  2. Στο επόμενο βήμα, η νέα είσοδος `sw = 1224` προστίθεται στην προηγούμενη τιμή του συσσωρευτή (feedback loop), δίνοντας το αποτέλεσμα 11410.
- **Συγχρονισμός:** Η σταθερότητα των σημάτων `led` και `accumulator` ακριβώς μετά την ακμή του `btnc` επιβεβαιώνει ότι η ιεραρχική σύνδεση των modules λειτουργεί με βάση τις προδιαγραφές.



Σχήμα 9: Αποτελέσματα Κυματομορφών του Testbench με χρήση του εργαλείου WaveTrace στο VS Code.

3.3 Συμπεράσματα

Η επιτυχής ολοκλήρωση της Άσκησης 2 αποδεικνύει την ορθή λειτουργία της ιεραρχικής σχεδίασης.

- Ο δομικός κωδικοποιητής μετέφερε σωστά τις εντολές του χρήστη στην ALU.
- Η λογική του συσσωρευτή επέτρεψε τη διαδοχική εκτέλεση πράξεων πάνω στο προηγούμενο αποτέλεσμα, επιβεβαιώνοντας τη σωστή διασύνδεση του συστήματος.
- Η ακριβής ταύτιση των τιμών επαληθεύει την ορθότητα της ALU στα 32-bit και του γενικότερου συστήματος.

Κώδικες για το ερώτημα αυτό μπορούν να βρεθούν στα αρχεία `calc.v`, `calc_enc.v` και `calc_tb.v`, με την αντίστοιχη προσομοίωση να βρίσκεται για ευκολία στο αρχείο με όνομα `calc_simulation.vcd`.

4 Μονάδα REGISTER FILE (Άσκηση 3)

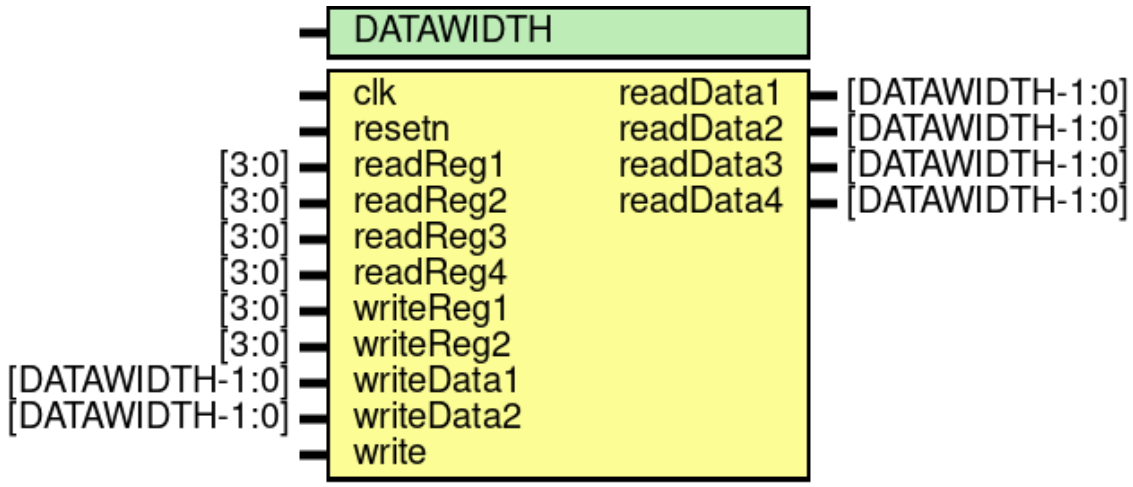
Σκοπός της Άσκησης 3 είναι η σχεδίαση ενός αρχείου καταχωρητών (Register File) πλάτους 32-bit και βάθους 16 θέσεων. Η συγκεκριμένη μονάδα αποτελεί κρίσιμο στοιχείο του AI accelerator της Άσκησης 4, καθώς είναι υπεύθυνη για την αποθήκευση και την ταχεία ανάκτηση των βαρών (weights) και των πολώσεων (biases) του νευρωνικού δικτύου.

4.1 Αρχιτεκτονική Σχεδίασης και Μεθοδολογία

4.1.1 Διεπαφή Μονάδας (Module Interface)

Το Register File σχεδιάστηκε για να υποστηρίζει υψηλό βαθμό παραλληλίας, διαθέτοντας:

- **4 Θύρες Ανάγνωσης (Read Ports):** Επιτρέπουν την ταυτόχρονη ανάκτηση τεσσάρων διαφορετικών παραμέτρων (`readData1-4`).
- **2 Θύρες Εγγραφής (Write Ports):** Επιτρέπουν την παράλληλη ενημέρωση δύο καταχωρητών (`writeData1-2`) στον ίδιο κύκλο ρολογιού.
- **Παραμετροποίηση:** Χρήση της παραμέτρου `DATAWIDTH` με προεπιλεγμένη τιμή 32-bit.



Σχήμα 10: Σχηματική αναπαράσταση της μονάδας `REG FILE`.
(Black Box diagram με την βοήθεια του εργαλείου TerosHDL στο VS Code)

4.1.2 Μεθοδολογία Υλοποίησης

Για τη σχεδίαση της μονάδας `regfile.v`, δόθηκε ιδιαίτερη βαρύτητα στη βελτιστοποίηση του χρονισμού και στη διαχείριση της ροής δεδομένων, ώστε να εξασφαλιστεί η μέγιστη απόδοση του AI accelerator της επόμενης άσκησης. Οι βασικοί πυλώνες της υλοποίησης συνοψίζονται παρακάτω:

- **Σύγχρονη Εγγραφή:** Η αποθήκευση των δεδομένων στους εσωτερικούς καταχωρητές (`registers`) πραγματοποιείται αποκλειστικά στην ακμή ανόδου του ρολογιού (`posedge clk`). Η επιλογή αυτή διασφαλίζει τη σταθερότητα του συστήματος, καθώς αποτρέπει την εγγραφή ενδιάμεσων τιμών και εγγυάται ότι οι αλλαγές στη μνήμη γίνονται με απόλυτο συγχρονισμό με το υπόλοιπο ψηφιακό σύστημα.
- **Ασύγχρονη Ανάγνωση και Στρατηγική Σχεδίασης:** Για την ανάγνωση των δεδομένων υιοθετήθηκε ασύγχρονη (συνδυαστική) λογική μέσω εντολών `assign`. Η συγκεκριμένη προσέγγιση επιλέχθηκε έναντι της σύγχρονης για τους εξής κρίσιμους λόγους:
  - **Μείωση Καθυστέρησης (Latency):** Επιτρέπει στον AI accelerator (Άσκηση 4) να ανακτά δεδομένα, όπως βάρη και εισόδους, ακαριαία μόλις τεθεί η διεύθυνση ανάγνωσης στις θύρες `readReg`. Με αυτόν τον τρόπο, τα δεδομένα είναι διαθέσιμα στις εισόδους της ALU μέσα στον ίδιο κύκλο ρολογιού.
  - **Απλοποίηση Μηχανής Καταστάσεων (FSM):** Η άμεση διαθεσιμότητα των δεδομένων επιτρέπει την εκτέλεση σύνθετων πράξεων Multiply and Accumulate (MAC) χωρίς την ανάγκη προσθήκης καταστάσεων αναμονής (wait states) στην FSM, κάνοντας τον επιταχυντή πιο αποδοτικό.
- **Εσωτερική Προώθηση (Internal Forwarding):** Ενσωματώθηκε ειδική λογική προτεραιότητας (`forwarding path`) η οποία συγκρίνει σε πραγματικό χρόνο τις διευθύνσεις ανάγνωσης και εγγραφής. Εάν αυτές συμπίπτουν και το σήμα `write` είναι ενεργό, η έξοδος οδηγείται απευθείας από το σήμα `writeData`. Η τεχνική αυτή επιλύει συγκρούσεις δεδομένων, διασφαλίζοντας ότι η μονάδα παρέχει πάντα την πιο πρόσφατη τιμή, ακόμη και αν αυτή δεν έχει προλάβει να αποθηκευτεί μόνιμα στη μνήμη.
- **Ασύγχρονο Reset:** Χρησιμοποιήθηκε σήμα `resetn` (active low), το οποίο επιβάλλει τον άμεσο μηδενισμό και των 16 καταχωρητών. Η ασύγχρονη φύση του ρεσετ εγγυάται ότι το σύστημα θα μεταβεί σε μια γνωστή και έγκυρη αρχική κατάσταση αμέσως μετά την ενεργοποίηση ή την επανεκκίνηση, ανεξάρτητα από την παρουσία ή την κατάσταση του ρολογιού.

4.2 Προσομοίωση και Επαλήθευση Λειτουργίας

4.2.1 Testbench και Σενάρια Ελέγχου

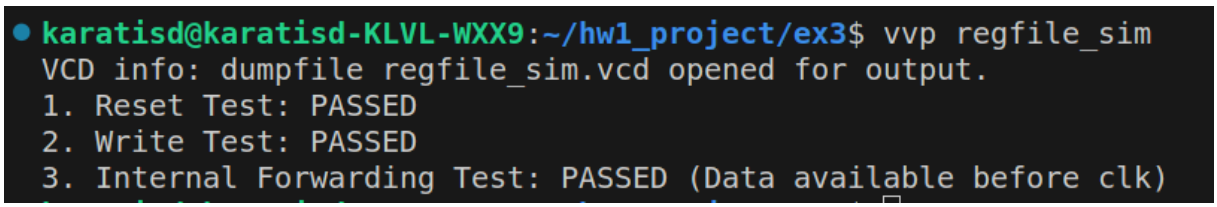
Για την επικύρωση της σχεδίασης, αναπτύχθηκε το αρχείο `regfile_tb.v`, το οποίο εκτελεί τρία κρίσιμα σενάρια ελέγχου:

1. **Reset Test:** Επιβεβαίωση ότι η μνήμη αρχικοποιείται στο μηδέν ανεξάρτητα από το ρολόι.
2. **Write Test:** Έλεγχος της ικανότητας ταυτόχρονης εγγραφής σε δύο διαφορετικούς καταχωρητές.
3. **Forwarding Test:** Έλεγχος της ασύγχρονης απόκρισης όταν η διεύθυνση ανάγνωσης συμπίπτει με τη διεύθυνση εγγραφής.

4.2.2 Ανάλυση Εξόδου Προσομοίωσης

Η έξοδος του τερματικού (terminal) επιβεβαιώνει την επιτυχή ολοκλήρωση όλων των δοκιμών:

```
1.  Reset Test:  PASSED
2.  Write Test:  PASSED
3.  Internal Forwarding Test:  PASSED (Data available before
clk)
```



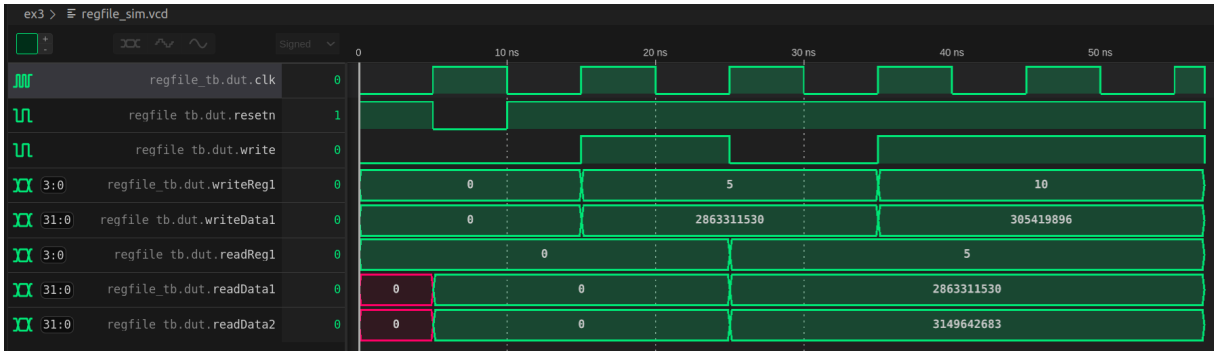
Σχήμα 11: Αποτελέσματα Επαλήθευσης Αρχείου Καταχωρητών μέσω Τερματικού (Terminal).

4.2.3 Ανάλυση Κυματομορφών Προσομοίωσης

Η ανάλυση των παρακάτω κυματομορφών επιβεβαιώνει την ορθή λειτουργία της μονάδας `regfile` στις εξής κρίσιμες φάσεις:

- **Ασύγχρονη Επαναφορά (Reset):** Στο χρονικό διάστημα μεταξύ 5ns και 10ns, παρατηρείται ότι το σήμα `resetn` μεταβαίνει σε χαμηλή λογική στάθμη (active-low). Αυτό έχει ως άμεσο αποτέλεσμα τον μηδενισμό των εξόδων `readData1` και `readData2`, αποδεικνύοντας ότι η αρχικοποίηση των καταχωρητών λειτουργεί ασύγχρονα, ανεξάρτητα από την ακμή του ρολογιού.

- **Σύγχρονη Εγγραφή:** Στο σημείο των 20ns, με το σήμα `write` ενεργοποιημένο, ξεκινά η διαδικασία εγγραφής στη διεύθυνση `writeReg1 = 5`. Η τιμή 2863311530 (που αντιστοιχεί στην δεκαεξαδική τιμή `0xAAAAAAAA`) αποθηκεύεται επιτυχώς στον καταχωρητή κατά την ακμή ανόδου του ρολογιού.
- **Ασύγχρονη Ανάγνωση και Internal Forwarding:** Η πλέον κρίσιμη παρατήρηση γίνεται ακριβώς στα 25ns. Μόλις η διεύθυνση ανάγνωσης `readReg1` μεταβληθεί στην τιμή 5, η έξοδος `readData1` ενημερώνεται **ακαριαία** στην τιμή 2863311530, πριν από την έλευση της επόμενης ακμής ανόδου του ρολογιού (30ns).
- **Συμπεράσματα:** Η άμεση αυτή απόκριση επαληθεύει:
  1. Την επιτυχή υλοποίηση της **ασύγχρονης ανάγνωσης**, η οποία εξαλείφει την καθυστέρηση ενός κύκλου ρολογιού (zero latency read).
  2. Τη σωστή λειτουργία του **Internal Forwarding**, καθώς η έξοδος τροφοδοτείται απευθείας από το σήμα εγγραφής όταν οι διευθύνσεις συμπίπτουν, εξασφαλίζοντας ότι η μονάδα παρέχει πάντα την πιο πρόσφατη πληροφορία.



Σχήμα 12: Αποτελέσματα Κυματομορφών του Testbench με χρήση του εργαλείου WaveTrace στο VS Code.

4.3 Συμπεράσματα

Η υλοποίηση του Register File κρίνεται απόλυτα επιτυχής και σύμφωνη με τις προδιαγραφές του project. Η υιοθέτηση της ασύγχρονης ανάγνωσης και του internal forwarding διασφαλίζει ότι ο AI accelerator θα μπορεί να εκτελεί πράξεις Multiply and Accumulate (MAC) με τη μέγιστη δυνατή ταχύτητα, αποφεύγοντας περιττούς κύκλους αναμονής για την ανάγνωση των βαρών.

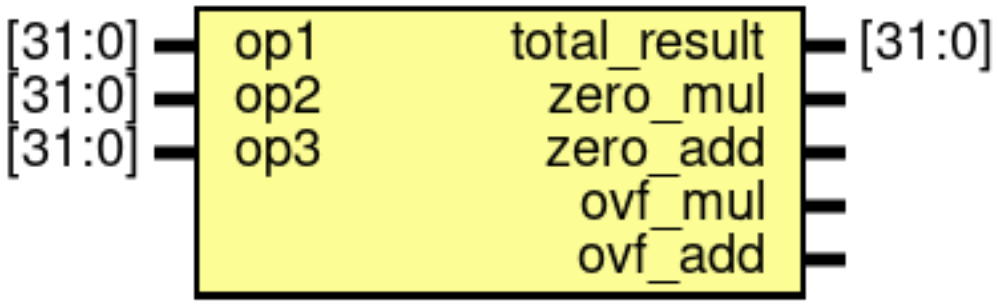
Κώδικες για το ερώτημα αυτό μπορούν να βρεθούν στα αρχεία `regfile.v` και `regfile_tb.v`, με την αντίστοιχη προσομοίωση να βρίσκεται για ευκολία στο αρχείο με όνομα `regfile_simulation.vcd`.

5 Σύνθεση AI ACCELERATOR (Άσκηση 4)

Στο πλαίσιο της τέταρτης και τελικής άσκησης, υλοποιείται το ολοκληρωμένο νευρωνικό σύστημα. Η διαδικασία ξεκινά με τη σχεδίαση της υπολογιστικής μονάδας MAC (Multiply and Accumulate), η οποία αποτελεί το βασικό στοιχείο επεξεργασίας κάθε νευρώνα. Στη συνέχεια, οι μονάδες αυτές συνδυάζονται με το Register File και τη ROM υπό τον έλεγχο μιας κεντρικής Μηχανής Πεπερασμένων Καταστάσεων (FSM).

5.1 Μονάδα MAC (mac_unit.v)

Η μονάδα MAC σχεδιάστηκε για να εκτελεί τη βασική πράξη ενός νευρώνα: $Result = (Input \times Weight) + Bias$. Η υλοποίηση βασίζεται στη διασύνδεση δύο μονάδων ALU από την Άσκηση 1, διατηρώντας την πλήρη προσημασμένη ακρίβεια 32-bit.



Σχήμα 13: Σχηματική αναπαράσταση της μονάδας `MAC UNIT`.
(Black Box diagram με την βοήθεια του εργαλείου TerosHDL στο VS Code)

5.1.1 Αρχιτεκτονική Σχεδίασης και Μεθοδολογία

Η αρχιτεκτονική της μονάδας `mac_unit` ακολουθεί μια σειριακή ροή δεδομένων:

- **Στάδιο Πολλαπλασιασμού (ALU 1):** Η πρώτη ALU λαμβάνει τις εισόδους op1 (δεδομένα) και op2 (βάρος). Χρησιμοποιώντας τον κωδικό λειτουργίας `4'b0110`, εκτελεί προσημασμένο πολλαπλασιασμό και παράγει το ενδιάμεσο γινόμενο `mul_res`.
- **Στάδιο Πρόσθεσης (ALU 2):** Η δεύτερη ALU λαμβάνει το `mul_res` και την είσοδο op3 (πόλωση/bias). Με τον κωδικό `4'b0100`, εκτελεί την πρόσθεση για την παραγωγή του τελικού αποτελέσματος `total_result`.
- **Διαχείριση Σημάτων Κατάστασης (Flags):** Η μονάδα εξάγει ξεχωριστά σήματα υπερχείλισης (`ovf_mul, ovf_add`) και μηδενισμού (`zero_mul, zero_add`) για κάθε στάδιο. Αυτό επιτρέπει στην κεντρική

μονάδα ελέγχου να ανιχνεύει σφάλματα υπολογισμού σε οποιοδήποτε σημείο της διαδικασίας.

5.2 Προσομοίωση και Επαλήθευση Μονάδας MAC

5.2.1 Testbench και Σενάρια Ελέγχου

Για την επαλήθευση της μονάδας αναπτύχθηκε το testbench `mac_unit_tb.v`, το οποίο υλοποίησε στοχευμένα σενάρια για τον έλεγχο της ορθότητας της μονάδας:

- **Βασική Λειτουργία:** Υπολογισμός θετικών αριθμών $(2 \times 3) + 4 = 10$.
- **Προσημασμένη Αριθμητική:** Έλεγχος με αρνητικούς αριθμούς $(-5 \times 4) + 10 = -10$, επιβεβαιώνοντας τη σωστή διατήρηση του προσήμου μέσω των ALUs.
- **Ανίχνευση Υπερχείλισης:** Δοκιμή με πολύ μεγάλες τιμές όπου το γινόμενο υπερβαίνει τα 32-bit, ενεργοποιώντας το σήμα `ovf_mul`.
- **Έλεγχος Μηδενισμού:** Επιβεβαίωση ενεργοποίησης του `zero_mul` όταν μία από τις εισόδους του πολλαπλασιασμού είναι μηδέν.

5.2.2 Ανάλυση Εξόδου Προσομοίωσης

Τα αποτελέσματα που εξήχθησαν από τον Icarus Verilog compiler επιβεβαιώνουν την ακρίβεια της μονάδας:

Έλεγχος	Πράξη (op1 * op2 + op3)	Αποτέλεσμα	Zero_mul	Ovf_mul
Test 1	(2 * 3) + 4	10	0	0
Test 2	(-5 * 4) + 10	-10	0	0
Test 3	(1073741824 * 4) + 1	1	0	1
Test 4	(0 * 100) + 50	50	1	0

Πίνακας 3: Αποτελέσματα Επαλήθευσης Μονάδας MAC.

```
karatisd@karatisd-KLVL-WXX9:~/hw1_project/ex4$ vvp mac_sim
VCD info: dumpfile mac_unit_simulation.vcd opened for output.
--- Starting MAC Unit Test ---
Test 1: (          2 *           3) +           4 =          10 | Zero_mul: 0
Test 2: (         -5 *           4) +          10 =         -10 | Zero_mul: 0
Test 3: ( 1073741824 *           4) +           1 =           1 | Ovf_mul: 1
Test 4: (          0 *         100) +          50 =          50 | Zero_mul: 1
--- MAC Unit Test Completed ---
```

Σχήμα 14: Αποτελέσματα Επαλήθευσης Μονάδας MAC μέσω Τερματικού.

Κώδικες για την μονάδα αυτή βρίσκονται στα αρχεία `mac_unit.v` και `mac_unit_tb.v`, ενώ προσομοίωση περιλαμβάνεται στο αρχείο `mac_unit_simulation.vcd`.

5.3 Κεντρική Μονάδα Νευρωνικού Δικτύου (nn.v)

Η κεντρική μονάδα `NN` αποτελεί το ανώτερο επίπεδο της σχεδίασης, ενσωματώνοντας όλες τις προηγούμενες μονάδες (ALU, Register File, MAC Unit) και τη μνήμη ROM που δίνεται έτοιμη για ευκολία. Η λειτουργία της διέπεται από μια Μηχανή Πεπερασμένων Καταστάσεων (FSM) 7 σταδίων, σχεδιασμένη να εκτελεί τη ροή δεδομένων ενός νευρωνικού δικτύου τριών νευρώνων.

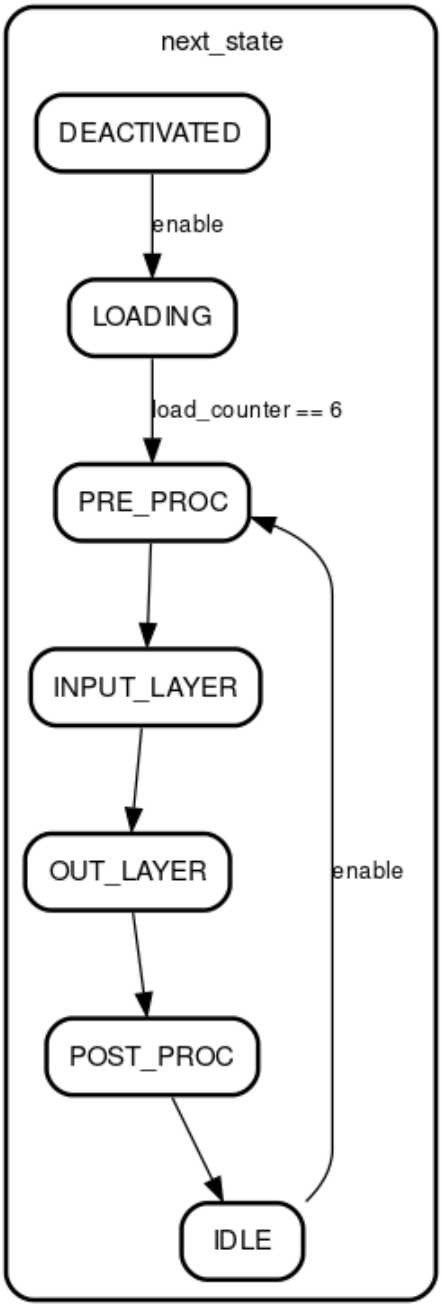
5.3.1 Αρχιτεκτονική και Μηχανή Καταστάσεων (FSM)

Για την υλοποίηση επιλέχθηκε η αρχιτεκτονική **Moore FSM**, καθώς οι έξοδοι ελέγχου εξαρτώνται αποκλειστικά από την τρέχουσα κατάσταση, εξασφαλίζοντας σταθερότητα χρονισμού και αποφυγή συνδυαστικών βρόχων (combinational loops). Οι 7 καταστάσεις της FSM αναλύονται ως εξής:

1. **DEACTIVATED (000):** Κατάσταση αναμονής μετά από reset. Το σύστημα ενεργοποιείται μόνο με το σήμα `enable`.
2. **LOADING (001):** Φόρτωση 10 συνολικά τιμών (βάρη και πολώσεις) από τη ROM στο Register File. Λόγω της υστέρησης ανάγνωσης της ROM, χρησιμοποιήθηκε ένας μετρητής (`load_counter`) που επιτρέπει την εγγραφή των δεδομένων στον επόμενο κύκλο από την αίτησή τους.
3. **PRE_PROC (010):** Παράλληλη προ-επεξεργασία των εισόδων μέσω αριθμητικής ολίσθησης δεξιά (SRA) από τις δύο ALU.
4. **INPUT_LAYER (011):** Παράλληλος υπολογισμός των δύο πρώτων νευρώνων χρησιμοποιώντας τις δύο μονάδες MAC (Neuron 1 & 2).
5. **OUT_LAYER (100):** Σειριακός υπολογισμός του τρίτου νευρώνα (Neuron 3) συνδυάζοντας τα ενδιάμεσα αποτελέσματα.
6. **POST_PROC (101):** Τελική επεξεργασία της εξόδου μέσω αριθμητικής ολίσθησης αριστερά (SLA).
7. **IDLE (110):** Ολοκλήρωση υπολογισμού. Το σύστημα αναμένει νέο `enable` για να ξεκινήσει απευθείας από το στάδιο PRE_PROC, διατηρώντας τα βάρη στη μνήμη.

5.3.2 Σχηματικό Διάγραμμα FSM

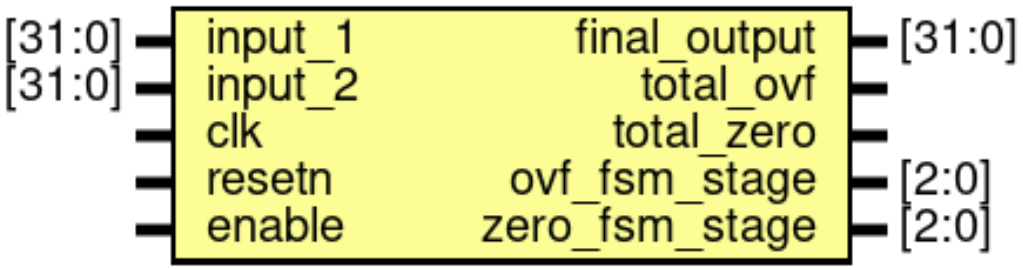
Για την οπτικοποίηση της λογικής ελέγχου του επιταχυντή, παρήχθη το σχηματικό διάγραμμα της Finite State Machine (FSM) μέσω του εργαλείου State Machine Viewer του TerosHDL extension από το αρχείο `nn.v`, που φαίνεται παρακάτω.



Σχήμα 15: Διάγραμμα Καταστάσεων και Μεταβάσεων FSM.

5.3.3 Διαχείριση Υπερχείλισης (Overflow)

Σύμφωνα με τις προδιαγραφές, το σύστημα διαθέτει μηχανισμό άμεσης διακοπής. Σε περίπτωση που ανιχνευθεί overflow σε οποιοδήποτε υπολογιστικό στάδιο, η FSM μεταβαίνει ακαριαία στην κατάσταση IDLE. Η έξοδος `final_output` λαμβάνει την τιμή `32'hFFFFFFFF` (-1 σε προσημασμένη αναπαράσταση), η οποία υποδηλώνει μη έγκυρο αποτέλεσμα λόγω υπέρβασης ορίων.



Σχήμα 16: Σχηματική αναπαράσταση της κεντρικής μονάδας `NN`.
(Black Box diagram με την βοήθεια του εργαλείου TerosHDL στο VS Code)

5.4 Προσομοίωση και Επαλήθευση AI ACCELERATOR

5.4.1 Testbench και Σενάρια Ελέγχου

Η επαλήθευση πραγματοποιήθηκε με το `tb_nn.v`, το οποίο συγκρίνει την έξοδο της μονάδας με ένα μαθηματικό μοντέλο αναφοράς (`nn_model.v`) που δίνεται έτοιμο. Το testbench υλοποιεί 100 διαδοχικές δοκιμές με χρήση της `$urandom_range()`:

- **Συγχρονισμός και Ρολόι:** Το testbench παράγει ένα σήμα ρολογιού με περίοδο 10ns και duty cycle 50%. Η φάση αρχικοποίησης ξεκινά με την ενεργοποίηση του active-low reset (`resetn = 0`) για 20ns, εξασφαλίζοντας ότι η FSM και όλοι οι καταχωρητές του συστήματος εκκινούν από μια γνωστή και έγκυρη κατάσταση.
- **Δομή των 100 Δοκιμών:** Ο έλεγχος υλοποιείται μέσω ενός βρόχου `for` 100 επαναλήψεων, οι οποίες κατανέμονται στοχευμένα για την κάλυψη διαφορετικών σεναρίων:
  1. **Κανονική Λειτουργία (40 Δοκιμές):** Παραγωγή τυχαίων εισόδων στο εύρος $[-4096, 4095]$ μέσω της συνάρτησης `$urandom_range()`. Σε αυτό το εύρος ελέγχεται η αριθμητική ακρίβεια των υπολογισμών υπό κανονικές συνθήκες.
  2. **Θετική Υπερχείλιση (30 Δοκιμές):** Χρήση μεγάλων θετικών τιμών στο εύρος `[32'h3FFFFFFF, 32'h7FFFFFFF]`. Σκοπός είναι η πρόκληση υπερχείλισης και η επιβεβαίωση ότι το κύκλωμα ανταποκρίνεται σωστά, αποδίδοντας την τιμή `32'hFFFFFFFF` στην έξοδο.

3. **Αρνητική Υπερχείλιση (30 Δοκιμές):** Χρήση μεγάλων αρνητικών τιμών στο εύρος `[32'h80000000, 32'hC0000000]`. Επιβεβαιώνεται η ικανότητα της FSM να ανιχνεύει σφάλματα και σε αρνητικές πράξεις.

- **Διαχείριση Καθυστέρησης (Latency):** Λαμβάνοντας υπόψη ότι η συνάρτηση αναφοράς παράγει αποτέλεσμα ακαριαία, ενώ το υλικό απαιτεί χρόνο για τη διαδρομή μέσω των σταδίων της FSM, εισήχθη μια καθυστέρηση αναμονής 10 κύκλων ρολογιού (`repeat (10) @(posedge clk)`) πριν από κάθε σύγκριση.
- **Αναφορά Σφαλμάτων και Σκορ (Logging):** Το testbench επιβλέπει διαρκώς τη σύγκριση `final_output === expected_output`.
  - **Σε περίπτωση σφάλματος:** Εκτυπώνεται αυτόματα η χρονική στιγμή της προσομοίωσης, οι δύο είσοδοι, η τιμή που παρήγαγε το κύκλωμα (Got) και η αναμενόμενη τιμή αναφοράς (Expected).
  - **Τελική Αναφορά:** Με την ολοκλήρωση των δοκιμών, τυπώνεται το συνολικό σκορ, επιτρέποντας την άμεση αξιολόγηση της αξιοπιστίας του συστήματος:

    `Final Score: PASS / Total`

5.4.2 Ανάλυση Εξόδου Προσομοίωσης

Η εκτέλεση της προσομοίωσης παρήγαγε το μέγιστο δυνατό σκορ επιτυχίας, αποδεικνύοντας την αξιοπιστία της σχεδίασης κάτω από όλες τις συνθήκες πίεσης.

Παράμετρος Ελέγχου	**Τιμή / Κατάσταση**
Περίοδος Ρολογιού	10 ns (50% duty cycle)
Συνολικές Δοκιμές	100
Επιτυχείς Συγκρίσεις	100
Final Pass Rate	**100 / 100**

Πίνακας 4: Σύνοψη Αποτελεσμάτων Τελικής Επαλήθευσης.

```
karatisd@karatisd-KLVL-WXX9:~/hw1_project/ex4$ vvp nn_sim
WARNING: ./rom.v:13: $readmemb: Standard inconsistency, following 1364-2005.
VCD info: dumpfile nn_simulation.vcd opened for output.
--- Starting 100 Test Cases ---
----------------------------------------
Final Score:         100 /         100 PASS
----------------------------------------
```

Σχήμα 17: Αποτελέσματα Επαλήθευσης Τελικής Μονάδας NN μέσω Τερματικού.

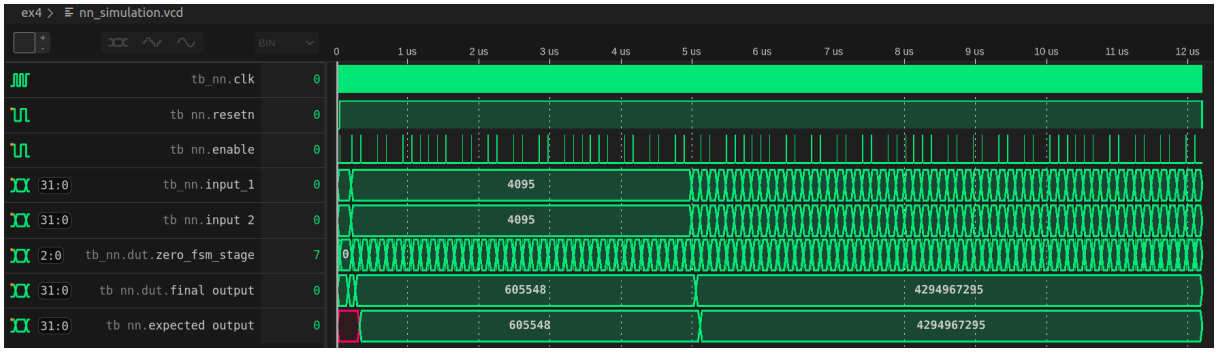
5.4.3 Ανάλυση Κυματομορφών Προσομοίωσης

Λόγω των περιορισμών του εργαλείου οπτικοποίησης (wavetrace), το οποίο επιτρέπει την ταυτόχρονη προβολή έως και 8 σημάτων στην δωρεάν έκδοση, η ανάλυση της προσομοίωσης χωρίστηκε σε δύο διακριτά διαγράμματα κυματομορφών. Η προσέγγιση αυτή επιτρέπει την πλήρη κατανόηση τόσο της εξωτερικής συμπεριφοράς του συστήματος όσο και της εσωτερικής ροής δεδομένων.

Συστημικό Διάγραμμα Χρονισμού (System Overview)

Το πρώτο διάγραμμα (Σχήμα 18) εστιάζει στη διεπαφή του επιταχυντή με το περιβάλλον δοκιμών (testbench).

- **Φάση Αρχικοποίησης:** Παρατηρείται ότι με την ακμή του `resetn`, το σύστημα τίθεται σε γνωστή κατάσταση. Η διαδικασία ξεκινά μόλις το σήμα `enable` ενεργοποιηθεί, δίνοντας το έναυσμα στην FSM για την έναρξη των υπολογισμών.
- **Συγχρονισμός Εισόδων:** Τα σήματα `input_1` και `input_2` μεταβάλλονται δυναμικά, αντιπροσωπεύοντας τα δεδομένα εισόδου που τροφοδοτούνται στον επιταχυντή για επεξεργασία.
- **Επαλήθευση Αποτελεσμάτων:** Είναι εμφανής η απόλυτη ταύτιση μεταξύ του `final_output` (έξοδος κυκλώματος) και του `expected_output` (τιμή αναφοράς). Για παράδειγμα, στην κυματομορφή καταγράφεται η επιτυχής παραγωγή της τιμής 605548 και στη συνέχεια της τιμής 4294967295 (σε περιπτώσεις μεγάλων αριθμών/υπερχείλισης), επιβεβαιώνοντας την αριθμητική ορθότητα του σχεδιασμού.

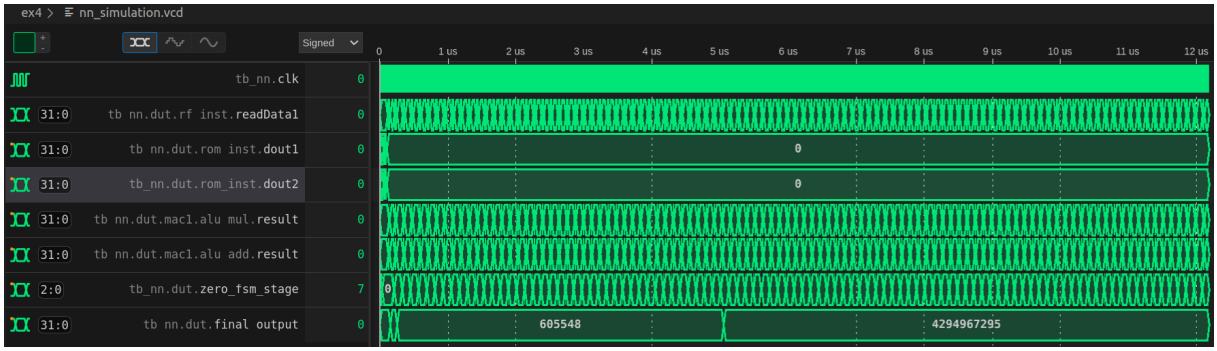


Σχήμα 18: Αποτελέσματα Κυματομορφών του Testbench με χρήση του εργαλείου WaveTrace στο VS Code, για το συνολικό σύστημα.

Εσωτερικό Διάγραμμα Ροής (Internal Data Path)

Το δεύτερο διάγραμμα (Σχήμα 19) αναλύει την αλληλεπίδραση των επιμέρους μονάδων στο εσωτερικό του DUT.

- **Λειτουργία ROM και Register File:** Παρατηρούμε τον παραλληλισμό στην ανάκτηση δεδομένων. Τα σήματα `dout1` και `dout2` από τη ROM παρέχουν τις εισόδους, ενώ ταυτόχρονα το Register File (`readData1`) παρέχει τα απαραίτητα βάρη (weights).
- **Επεξεργασία MAC:** Οι έξοδοι των αριθμητικών μονάδων (`alu_mul.result` και `alu_add.result`) δείχνουν την εξέλιξη του υπολογισμού σε πραγματικό χρόνο. Το γινόμενο υπολογίζεται ακαριαία και προστίθεται στο τρέχον μερικό άθροισμα.
- **Έλεγχος FSM:** Το σήμα `zero_fsm_stage` καθοδηγεί τη ροή. Παρατηρούμε ότι η τελική έξοδος (`final_output`) σταθεροποιείται μόνο όταν η FSM φτάσει στο τελικό στάδιο ολοκλήρωσης (τιμή 7), εξασφαλίζοντας ότι το αποτέλεσμα είναι έγκυρο πριν αναγνωσθεί από το επόμενο στάδιο.



Σχήμα 19: Αποτελέσματα Κυματομορφών του Testbench με χρήση του εργαλείου WaveTrace στο VS Code, για την εσωτερική ροή του συστήματος.

5.5 Συμπεράσματα

Η συνδυαστική ανάλυση των δύο διαγραμμάτων αποδεικνύει ότι ο επιταχυντής λειτουργεί με υψηλή αξιοπιστία. Η χρήση internal forwarding στο Register File και η σωστή διαχείριση των καταστάσεων από την FSM επιτρέπουν την εκτέλεση σύνθετων πράξεων Multiply-Accumulate χωρίς σφάλματα χρονισμού, διατηρώντας την υπολογιστική ακρίβεια σε κάθε περίπτωση.

Οι κώδικες για την τελική μονάδα βρίσκονται στα αρχεία `nn.v` και `tb_nn.v`, ενώ η πλήρης προσομοίωση είναι διαθέσιμη στο αρχείο `nn_simulation.vcd`.

Αναφορές

[1] Υλικό και Διαλέξεις Μαθήματος Ψηφιακά Συστήματα Hardware σε Χαμηλά Επίπεδα Λογικής Ι, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης (ΑΠΘ), Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών.

[2] Icarus Verilog Open-Source Verilog compiler and simulation tool. Available at: `https://steveicarus.github.io/iverilog/`

[3] TerosHDL: Open source toolbox (ASIC/FPGA, VHDL, Verilog, SystemVerilog) and Visual Studio Code extension. Available at: `https://marketplace.visualstudio.com/items?itemName=teros-technology.teroshdl`

[4] WaveTrace: Waveform viewer extension for Visual Studio Code. Available at: `https://marketplace.visualstudio.com/items?itemName=wavetrace.wavetrace`