Ψηφιακά Συστήματα **HW** σε Χαμηλά Επίπεδα Λογικής **I**

Αναφορά Εργαστηριακών Ασκήσεων
Εξοικείωσης με τη Verilog

Ζωίδης Βασίλειος
ΑΕΜ: **10652**

Ιανουάριος 2026

Contents

1 Εισαγωγή

Η παρούσα αναφορά περιγράφει την υλοποίηση τεσσάρων εργαστηριακών ασκήσεων σε γλώσσα περιγραφής υλικού Verilog, με τελικό στόχο τη σχεδίαση ενός απλού AI accelerator, καθώς και τις κρίσιμες σχεδιαστικές αποφάσεις που ελήφθησαν κατά τη διάρκεια της ανάπτυξης. Οι ασκήσεις έχουν σχεδιαστεί με προοδευτικό τρόπο, ώστε κάθε επόμενη άσκηση να βασίζεται και να επεκτείνει τη λειτουργικότητα των προηγούμενων, δημιουργώντας ένα ολοκληρωμένο σύστημα ψηφιακής επεξεργασίας.

1.1 Δομή και Περιεχόμενο Ασκήσεων

Οι τέσσερις ασκήσεις περιλαμβάνουν:

1. **Αριθμητική/Λογική Μονάδα (ALU) 32-bit:** Σχεδίαση μιας πλήρους ALU που υποστηρίζει 12 διαφορετικές λειτουργίες (αριθμητικές, λογικές, και ολισθήσεις), με ενσωματωμένο μηχανισμό ανίχνευσης υπερχείλισης.

2. **Αριθμομηχανή βασισμένη στην ALU:** Υλοποίηση ενός λειτουργικού συστήματος αριθμομηχανής που αξιοποιεί την ALU της πρώτης άσκησης. Περιλαμβάνει structural encoder για την κωδικοποίηση των εισόδων ελέγχου, καθώς και ολοκληρωμένο testbench για επαλήθευση της ορθής λειτουργίας.

3. **Register File με Πολλαπλές Θύρες I/O:** Δημιουργία ενός καταχωρητή 16×32-bit με δυνατότητα σύγχρονης ανάγνωσης και εγγραφής μέσω πολλαπλών θυρών. Υλοποιήθηκε με τεχνική data forwarding για την αποφυγή data hazards και τη βελτίωση της απόδοσης.

4. **Σχεδίαση Νευρωνικού Δικτύου:** Σχεδίαση ενός πλήρους συστήματος νευρωνικού δικτύου που ενσωματώνει όλα τα προηγούμενα modules. Υλοποιήθηκε με Moore FSM για καθαρό συγχρονισμό, περιλαμβάνει Multiply-Accumulate Unit (MAC) για αποδοτικούς υπολογισμούς, και διαχειρίζεται το Register File με τεχνική prefetching για την αντιμετώπιση της σύγχρονης ανάγνωσης.

1.2 Μεθοδολογία και Εργαλεία

Η ανάπτυξη και προσομοίωση όλων των modules πραγματοποιήθηκε στην πλατφόρμα **EDA Playground** με χρήση του προσομοιωτή **Icarus Verilog 12.0**. Για την οπτικοποίηση και ανάλυση των κυματομορφών χρησιμοποιήθηκε το ενσωματωμένο εργαλείο **EPWave**, το οποίο επέτρεψε την αναλυτική επαλήθευση της χρονικής συμπεριφοράς των σημάτων. Κάθε άσκηση συνοδεύεται από εκτενή testbenches που επαληθεύουν τη σωστή λειτουργία σε διάφορα σενάρια εισόδου. Η σχεδίαση ακολουθεί τις βέλτιστες πρακτικές RTL (Register Transfer Level) design και συμμορφώνεται με το πρότυπο IEEE 1364-2005 για τη γλώσσα Verilog.

2 Άσκηση 1: Αριθμητική/Λογική Μονάδα (ALU)

2.1 Περιγραφή

Η ALU είναι ένα συνδυαστικό κύκλωμα 32-bit που υλοποιεί δώδεκα διαφορετικές λειτουργίες, κατηγοριοποιημένες σε τέσσερεις ομάδες: λογικές πράξεις, αριθμητικές πράξεις και πράξεις λογικής κι αριθμητικής ολίσθησης.

Table 1: Πίνακας λειτουργιών ALU

alu_op	**Κατηγορία - Λειτουργία**
1000	Λογική - AND
1001	Λογική - OR
1010	Λογική - NOR
1011	Λογική - NAND
1100	Λογική - XOR
0100	Προσημασμένη - Πρόσθεση
0101	Προσημασμένη - Αφαίρεση
0110	Προσημασμένος - Πολλαπλασιασμός
0000	Λογική ολίσθηση - Δεξιά
0001	Λογική ολίσθηση - Αριστερά
0010	Αριθμητική ολίσθηση - Δεξιά
0011	Αριθμητική ολίσθηση - Αριστερά

2.2 Αρχιτεκτονική

Η ALU σχεδιάστηκε ως συνδυαστικό κύκλωμα με τις ακόλουθες θύρες εισόδου/εξόδου:

Table 2: Θύρες του module ALU

Θύρα	**Κατεύθυνση**	**Πλάτος**	**Περιγραφή**
`op1`	Είσοδος	32	Τελεστής 1 σε συμπλήρωμα ως προς 2
`op2`	Είσοδος	32	Τελεστής 2 σε συμπλήρωμα ως προς 2
`alu_op`	Είσοδος	4	Κωδικός επιλογής λειτουργίας
`result`	Έξοδος	32	Αποτέλεσμα της λειτουργίας
`zero`	Έξοδος	1	Σημαία μηδενικού αποτελέσματος
`ovf`	Έξοδος	1	Σημαία υπερχείλισης

2.2.1 Λειτουργίες ALU

Λογικές Πράξεις: Οι λογικές πράξεις (AND, OR, NOR, NAND, XOR) εκτελούνται bit-wise μεταξύ των δύο τελεστέων. Δεν παράγουν υπερχείλιση, συνεπώς το σήμα `ovf` παραμένει πάντα 0.

Αριθμητικές Πράξεις: Οι αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμός) εκτελούνται σε αριθμούς συμπληρώματος ως προς 2 και περιλαμβάνουν μηχανισμό ανίχνευσης υπερχείλισης.

Πράξεις Ολίσθησης: Οι πράξεις ολίσθησης μετακινούν τα bits του op1 κατά τον αριθμό θέσεων που καθορίζεται από το op2. Πάρθηκε η απόφαση να χρησιμοποιούνται μόνο **τα 5 λιγότερο σημαντικά bits** του op2 (bits 4:0), επιτρέποντας ολισθήσεις από 0 έως 31 θέσεις, καθώς για έναν καταχωρητή 32-bit, ολίσθηση πέρα από 31 θέσεις δεν έχει πρακτική σημασία. Αναφορικά με κενά bits που δημιουργούνται, αυτά κατά περίπτωση:

- **Λογική ολίσθηση (SRL & SLL):** Τα κενά bits συμπληρώνονται με 0.
- **Αριθμητική ολίσθηση δεξιά (SRA & SLA):** Τα κενά bits συμπληρώνονται έτσι ώστε να διατηρείται το πρόσημο του op1 (sign extension).

2.3 Ανίχνευση Υπερχείλισης

Η υπερχείλιση συμβαίνει όταν το αποτέλεσμα μιας αριθμητικής πράξης δεν μπορεί να αναπαρασταθεί σωστά σε 32 bits συμπληρώματος ως προς 2. Στην αναπαράσταση συμπληρώματος ως προς 2, το bit 31 (MSB) αποτελεί το **bit προσήμου**: 0 για θετικούς αριθμούς, 1 για αρνητικούς.

2.3.1 Πρόσθεση

Η υπερχείλιση στην πρόσθεση ανιχνεύεται με την παρακάτω λογική, καθώς υπερχείλιση συμβαίνει μόνο όταν προσθέτουμε δύο αριθμούς με το **ίδιο πρόσημο** και το αποτέλεσμα έχει **διαφορετικό πρόσημο**:

```
ovf = (op1[31] == op2[31]) && (result[31] != op1[31]);
```

Listing 1: Έλεγχος υπερχείλισης για τη πρόσθεση

Περιπτώσεις:

- Θετικός + Θετικός = Αρνητικός → Υπερχείλιση
- Αρνητικός + Αρνητικός = Θετικός → Υπερχείλιση
- Θετικός + Αρνητικός = Οποιοδήποτε → Όχι υπερχείλιση (το άθροισμα είναι πάντα εντός ορίων)

Παράδειγμα:
0x7FFFFFFF (max θετικός: $2^{31}-1$) + 0x00000001 = 0x80000000 (max αρνητικός: -2^{31}) ⇒ op1[31]=0, op2[31]=0, result[31]=1 ⇒ ovf=1

2.3.2 Αφαίρεση

Η υπερχείλιση στην αφαίρεση ανιχνεύεται με την παρακάτω λογική, καθώς υπερχείλιση συμβαίνει όταν αφαιρούμε αριθμούς με **διαφορετικό πρόσημο** και το αποτέλεσμα έχει το **ίδιο πρόσημο με τον αφαιρετέο** (op2):

```
ovf = (op1[31] != op2[31]) && (result[31] == op2[31]);
```

Listing 2: Έλεγχος υπερχείλισης για την αφαίρεση

Περιπτώσεις:

- Θετικός - Αρνητικός = Αρνητικός → Υπερχείλιση
- Αρνητικός - Θετικός = Θετικός → Υπερχείλιση
- Θετικός - Θετικός = Οποιοδήποτε → Όχι υπερχείλιση

Παράδειγμα:
`0x7FFFFFFF` (max θετικός) - `0x80000000` (max αρνητικός) = `0xFFFFFFFF` (αρνητικός)
⇒ `op1[31]=0`, `op2[31]=1`, `result[31]=1` ⇒ `ovf=1`

2.3.3 Πολλαπλασιασμός

Ο πολλαπλασιασμός δύο αριθμών 32-bit παράγει αποτέλεσμα 64-bit. Η υπερχείλιση ανιχνεύεται ελέγχοντας αν τα υψηλότερα 33 bits (bits 63:31) είναι έγκυρη επέκταση προσήμου του bit 31:

```
ovf = (mult_result[63:31] != {33{mult_result[31]}}) &&
      (mult_result[63:31] != 33'b0);
```

Listing 3: Έλεγχος υπερχείλισης για τον πολλαπλασιασμό

Περιπτώσεις: Για να χωρέσει το αποτέλεσμα σε 32 bits, τα bits 63:31 πρέπει να είναι:

- Όλα 1 (αν το bit 31 είναι 1, για αρνητικούς αριθμούς), ή
- Όλα 0 (για θετικούς αριθμούς που χωρούν σε 32 bits)

Αν καμία από τις δύο συνθήκες δεν ισχύει, το αποτέλεσμα είναι πολύ μεγάλο ή πολύ μικρό για να αναπαρασταθεί σε 32 bits.

3 Άσκηση 2: Αριθμομηχανή (Calculator)

3.1 Περιγραφή

Σε αυτή την άσκηση σχεδιάστηκε ένα κύκλωμα αριθμομηχανής (calc.v) που χρησιμοποιεί την ALU της Άσκησης 1. Το κύκλωμα διατηρεί μια τρέχουσα τιμή σε έναν συσσωρευτή (accumulator) 16-bit και επιτρέπει στον χρήστη να εκτελεί ορισμένες αριθμητικές και λογικές πράξεις μέσω της ALU. Οι λειτουργίες της αριθμομηχανής επιλέγονται μέσω τριών κουμπιών (btnl, btnr, btnd) και το αποτέλεσμα εμφανίζεται στην έξοδο led (16-bit).

3.2 Αρχιτεκτονική

Το calc module υλοποιήθηκε ως ακολουθιακό κύκλωμα με τις ακόλουθες θύρες εισόδου/εξόδου:

Table 3: Θύρες του module calc

Θύρα	Κατεύθυνση	Πλάτος	Περιγραφή
`clk`	Είσοδος	1	Σήμα ρολογιού
`btnc`	Είσοδος	1	Κεντρικό πλήκτρο - εκτέλεση πράξης
`btnac`	Είσοδος	1	Πλήκτρο εκκαθάρισης (all clear)
`btnl`	Είσοδος	1	Αριστερό πλήκτρο (επιλογής)
`btnr`	Είσοδος	1	Δεξί πλήκτρο (επιλογής)
`btnd`	Είσοδος	1	Κάτω πλήκτρο (επιλογής)
`sw`	Είσοδος	16	Διακόπτες εισαγωγής δεδομένων
`led`	Έξοδος	16	LED εξόδου του accumulator

3.2.1 Συσσωρευτής (Accumulator)

Ο accumulator είναι ένας σύγχρονος εσωτερικός καταχωρητής 16-bit με τις εξής ιδιότητες:

- Συνδεδεμένος με την είσοδο ρολογιού (`clk`)
- Σύγχρονος μηδενισμός με το πάτημα του `btnac`
- Ενημερώνεται με τα 16 χαμηλότερα bits του αποτελέσματος της ALU όταν πατηθεί το `btnc`
- Η τιμή του εμφανίζεται στα LED

```
always @(posedge clk) begin
    if (btnac)
        accumulator <= 16'b0;            // Synchronous Reset
    else if (btnc)
        accumulator <= alu_result[15:0];
end
```

Listing 4: Συμπεριφορά Καταχωρητή Accumulator

3.2.2 Επέκταση Προσήμου (Sign Extension)

Επειδή η ALU δέχεται τελεστές 32-bit και ο accumulator/switches είναι 16-bit, εφαρμόζεται επέκταση προσήμου (sign extension). Το MSB (bit 15) επαναλαμβάνεται 16 φορές για τα υψηλότερα bits:

```
assign op1_extended={{16{accumulator[15]}},accumulator}; //acc -> 32-bit
assign op2_extended={{16{sw[15]}},sw};                   // sw -> 32-bit
```

Listing 5: Επέκταση προσήμου του MSB

3.3 Encoder (calc_enc.v)

Ο encoder (calc_enc.v) υλοποιήθηκε σε **structural Verilog** χρησιμοποιώντας βασικές πύλες (AND, OR, NOT, XOR) σύμφωνα με τα Σχήματα 2-5 της εκφώνησης. Οι λογικές εξισώσεις για κάθε bit του alu_op είναι:

$$(\Sigma\chi.\ 2): \quad \text{alu\_op}[0] = (\overline{\text{btnl}} \cdot \text{btnd}) + ((\text{btnl} \cdot \text{btnr}) \cdot \overline{\text{btnd}}) \tag{1}$$

$$(\Sigma\chi.\ 3): \quad \text{alu\_op}[1] = \text{btnl} \cdot (\overline{\text{btnr}} + \overline{\text{btnd}}) \tag{2}$$

$$(\Sigma\chi.\ 4): \quad \text{alu\_op}[2] = (\overline{\text{btnl}} \cdot \text{btnr}) + (\text{btnl} \cdot \overline{(\text{btnr} \oplus \text{btnd})}) \tag{3}$$

$$(\Sigma\chi.\ 5): \quad \text{alu\_op}[3] = (\text{btnl} \cdot \text{btnr}) + (\text{btnl} \cdot \text{btnd}) \tag{4}$$

Ο πίνακας αλήθειας που προκύπτει από αυτές τις εξισώσεις:

btnl	btnr	btnd	alu_op	Λειτουργία
0	0	0	0000	SRL
0	0	1	0001	SLL
0	1	0	0100	ADD
0	1	1	0101	SUB
1	0	0	0110	MULT
1	0	1	1010	NOR
1	1	0	1011	NAND
1	1	1	1100	XOR

3.4 Αποτελέσματα Testbench

Η καλή λειτουργία της αριθμομηχανής επιβεβαιώθηκε αποτυπώνοντας τον έλεγχο της εκφώνησης στο calc_tb.v:

Table 4: Αποτελέσματα Προσομοίωσης (Calculator Testbench)

#	**Input** (btnl,btnr,btnd)	**Switches** (input)	**Function** (in ALU)	**Expected Result**	**Actual Result**	**Test Result**
1	btnac	xxxx	Reset	0x0000	0x0000	**PASS**
2	0, 1, 0	0x285a	ADD	0x285a	0x285a	**PASS**
3	1, 1, 1	0x04c8	XOR	0x2c92	0x2c92	**PASS**
4	0, 0, 0	0x0005	SRL	0x0164	0x0164	**PASS**
5	1, 0, 1	0xa085	NOR	0x5e1a	0x5e1a	**PASS**
6	1, 0, 0	0x07fe	MULT	0x13cc	0x13cc	**PASS**
7	0, 0, 1	0x0004	SLL	0x3cc0	0x3cc0	**PASS**
8	1, 1, 0	0xfa65	NAND	0xc7bf	0xc7bf	**PASS**
9	0, 1, 1	0xb2e4	SUB	0x14db	0x14db	**PASS**
Συνολικό Αποτέλεσμα: 9/9 PASS						

3.5 Κυματομορφές Προσομοίωσης

Τα παραπάνω μπορούμε να διαπιστώσουμε και στις κυματομορφές από την προσομοίωση του testbench (Figure 1). Παρατηρούμε:

- Η τιμή της εξόδου (`led`) ενημερώνεται -σύγχρονα- στην ανερχόμενη ακμή του `clk` όταν το `btnc` είναι ενεργό.
- Το `btnac` λειτουργεί ως reset πάλι σύγχρονα.
- Οι διαδοχικές τιμές αντιστοιχούν στα αναμενόμενα αποτελέσματα: `0x0` → `0x285a` → `0x2c92` → `0x0164` → `0x5e1a` → `0x13cc` → `0x3cc0` → `0xc7bf` → `0x14db`

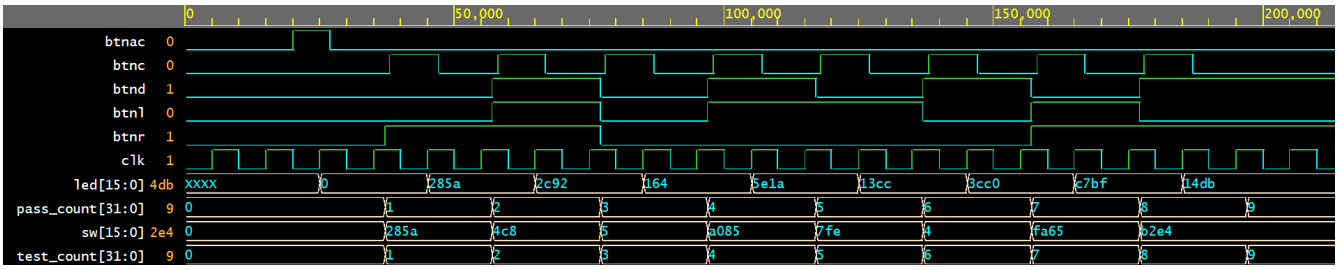

Figure 1: Κυματομορφές προσομοίωσης της αριθμομηχανής. Φαίνεται η εξέλιξη της τιμής του led μετά από κάθε πράξη.

4 Άσκηση 3: Register File

4.1 Περιγραφή

Το αρχείο καταχωρητών (Register File) είναι ένα σημαντικό στοιχείο κάθε ψηφιακού συστήματος. Σε αυτή την άσκηση υλοποιήθηκε ένα Register File 16 καταχωρητών, το οποίο θα χρησιμοποιηθεί στον AI accelerator (Άσκηση 4) για την αποθήκευση των βαρών (weights) και των πολώσεων (biases) του νευρωνικού δικτύου. Το Register File που υλοποιήθηκε υποστηρίζει παραμετροποιήσιμο πλάτος δεδομένων (DATAWIDTH) και ασύγχρονο reset.

4.2 Αρχιτεκτονική

Το regfile module σχεδιάστηκε ως ακολουθιακό κύκλωμα με τις ακόλουθες θύρες εισόδου/εξόδου:

Table 5: Θύρες του module regfile

Θύρα	Κατεύθυνση	Πλάτος	Περιγραφή
`clk`	Είσοδος	1	Σήμα ρολογιού
`resetn`	Είσοδος	1	Σήμα επαναφοράς (active low)
`write`	Είσοδος	1	Σήμα ενεργοποίησης εγγραφής
`readReg1-4`	Είσοδοι	4	Διευθύνσεις ανάγνωσης (4 θύρες)
`writeReg1-2`	Είσοδοι	4	Διευθύνσεις εγγραφής (2 θύρες)
`writeData1-2`	Είσοδοι	DW	Δεδομένα προς εγγραφή
`readData1-4`	Έξοδοι	DW	Δεδομένα ανάγνωσης (4 θύρες)

4.3 Λειτουργία Reset και Read/Write

Η λειτουργία του Register File βασίζεται σε ένα κεντρικό `always` block.

- **Reset**: Όλοι οι καταχωρητές αρχικοποιούνται **ασύγχρονα** στο μηδέν όταν το `resetn` είναι χαμηλό.
- **Read/Write**: Τα δεδομένα διαβάζονται/εγγράφονται στους επιλεγμένους καταχωρητές **σύγχρονα** με την ανερχόμενη ακμή του clk.

```
always @(posedge clk or negedge resetn) begin
    if (!resetn) begin // Asynchronous reset
        for (i = 0; i < 16; i = i + 1) begin
            registers[i] <= {DATAWIDTH{1'b0}}; // Clear all
        end
        readData1 <= {DATAWIDTH{1'b0}};
        ...
    end
    else begin // Synchronous read/write
        if (write) begin
            registers[writeReg1] <= writeData1; // Write
            ...
        end
    end
```

Listing 6: Always block του Register file

4.4 Data Forwarding (Προώθηση Δεδομένων)

Ένα κρίσιμο χαρακτηριστικό της αρχιτεκτονικής είναι η λογική Data Forwarding. Σε περίπτωση που η διεύθυνση ανάγνωσης ταυτίζεται με μια διεύθυνση εγγραφής στον ίδιο κύκλο, πρέπει να δοθεί **προτεραιότητα στην εγγραφή**.

Αντί να διαβαστεί η "παλιά" τιμή του καταχωρητή, στην έξοδο οδηγείται απευθείας η νέα τιμή (`writeData`). Η προτεραιότητα καθορίζεται ως εξής:

1. Forwarding από Write Port 1 (Υψηλότερη προτεραιότητα)
2. Forwarding από Write Port 2 στους επιλεγμένους καταχωρητές
3. Ανάγνωση από Register Array (Κανονική λειτουργία)

```
// Example for Port 1
if (readReg1 == writeReg1)
    readData1 <= writeData1;    // Priority 1
else if (readReg1 == writeReg2)
    readData1 <= writeData2;    // Priority 2
else
    readData1 <= registers[readReg1]; // Normal Read
```

Listing 7: Υλοποίηση Data Forwarding

5 Άσκηση 4: Νευρωνικό Δίκτυο

5.1 Περιγραφή

Σε αυτή την άσκηση υλοποιήθηκε ένας AI Accelerator που εκτελεί ένα απλό νευρωνικό δίκτυο 3 νευρώνων. Το σύστημα ενσωματώνει modules προηγούμενων ασκήσεων, αλλά και νέα modules:

- **ALU** (Άσκηση 1): Για τις πράξεις ολίσθησης (preprocessing/postprocessing).
- **Register File** (Άσκηση 3): Για την αποθήκευση βαρών και πολώσεων.
- **MAC Unit**: Για την εκτέλεση των Multiply / Accumulate σου νευρώνες.
- **ROM**: Για την αρχική φόρτωση των παραμέτρων στο Register File.

5.2 Αρχιτεκτονική

Το module **nn** υλοποιήθηκε ως ακολουθιακό κύκλωμα ελεγχόμενο από FSM, με τις ακόλουθες θύρες:

Table 6: Θύρες του module nn

Θύρα	Κατεύθυνση	Πλάτος	Περιγραφή
`clk`	Είσοδος	1	Σήμα ρολογιού
`resetn`	Είσοδος	1	Σήμα επαναφοράς (active low)
`enable`	Είσοδος	1	Σήμα ενεργοποίησης
`input_1`	Είσοδος	32	Πρώτη είσοδος
`input_2`	Είσοδος	32	Δεύτερη είσοδος
`total_ovf`	Έξοδος	1	Ένδειξη υπερχείλισης
`total_zero`	Έξοδος	1	Ένδειξη μηδενικού αποτελέσματος
`ovf_fsm_stage`	Έξοδος	3	Στάδιο υπερχείλισης
`zero_fsm_stage`	Έξοδος	3	Στάδιο μηδενισμού
`final_output`	Έξοδος	32	Τελικό αποτέλεσμα

5.2.1 Αρχιτεκτονική Νευρωνικού

Το νευρωνικό δίκτυο αποτελείται από 3 νευρώνες και υλοποιεί την ακόλουθη λογική:

$$\text{inter}_1 = \text{input}_1 >>> \text{shift\_bias}_1 \tag{5}$$
$$\text{inter}_2 = \text{input}_2 >>> \text{shift\_bias}_2 \tag{6}$$
$$\text{inter}_3 = \text{inter}_1 \times \text{weight}_1 + \text{bias}_1 \tag{7}$$
$$\text{inter}_4 = \text{inter}_2 \times \text{weight}_2 + \text{bias}_2 \tag{8}$$
$$\text{inter}_5 = \text{inter}_3 \times \text{weight}_3 + \text{inter}_4 \times \text{weight}_4 + \text{bias}_3 \tag{9}$$
$$\text{output} = \text{inter}_5 <<< \text{shift\_bias}_3 \tag{10}$$

5.2.2 MAC Unit

Η μονάδα MAC (Multiply and Accumulate) υλοποιεί:

$$\text{result} = (\text{op1} \times \text{op2}) + \text{op3}$$

Αποτελείται από δύο σειριακά συνδεδεμένες ALU:

1. Πρώτη ALU: Πολλαπλασιασμός (op1 × op2)
2. Δεύτερη ALU: Πρόσθεση (result1 + op3)

5.3 Finite State Machine (FSM)

5.3.1 Τύπος FSM: Registered Mealy

Επιλέχθηκε **Registered Mealy FSM** (Σύγχρονη Mealy) για τους εξής λόγους:

- **Εξάρτηση εξόδων από εισόδους:** Οι έξοδοι (`final_output`, `total_ovf`) εξαρτώνται τόσο από την τρέχουσα κατάσταση όσο και από τα σήματα εισόδου (`alu1_ovf`, `alu2_ovf`, `mac1_ovf_*`, `mac2_ovf_*`).
- **Registered outputs:** Επειδή η λογική εξόδων βρίσκεται μέσα σε `always @(posedge clk)`, οι έξοδοι είναι **σύγχρονες**, εξαλείφοντας glitches και εξασφαλίζοντας σταθερές τιμές για ολόκληρο τον κύκλο ρολογιού
- **Καθυστέρηση ενός κύκλου:** Η αλλαγή εξόδων εμφανίζεται στην *επόμενη* ακμή ρολογιού μετά τον εντοπισμό συνθήκης (π.χ. overflow), παρέχοντας πιο εύκολο timing closure

5.3.2 Διάγραμμα Καταστάσεων FSM

Το ακόλουθο διάγραμμα απεικονίζει το FSM 7 καταστάσεων (Figure 2). Πρακτικά το Output Layer (S5) υλοποιήθηκε σε δύο καταστάσεις (έτσι ώστε να εκτελείται σε 2 κυκλους του ρολογιού) για την αποφυγή combinatorial loops (βλ. παρ. 5.3.4).

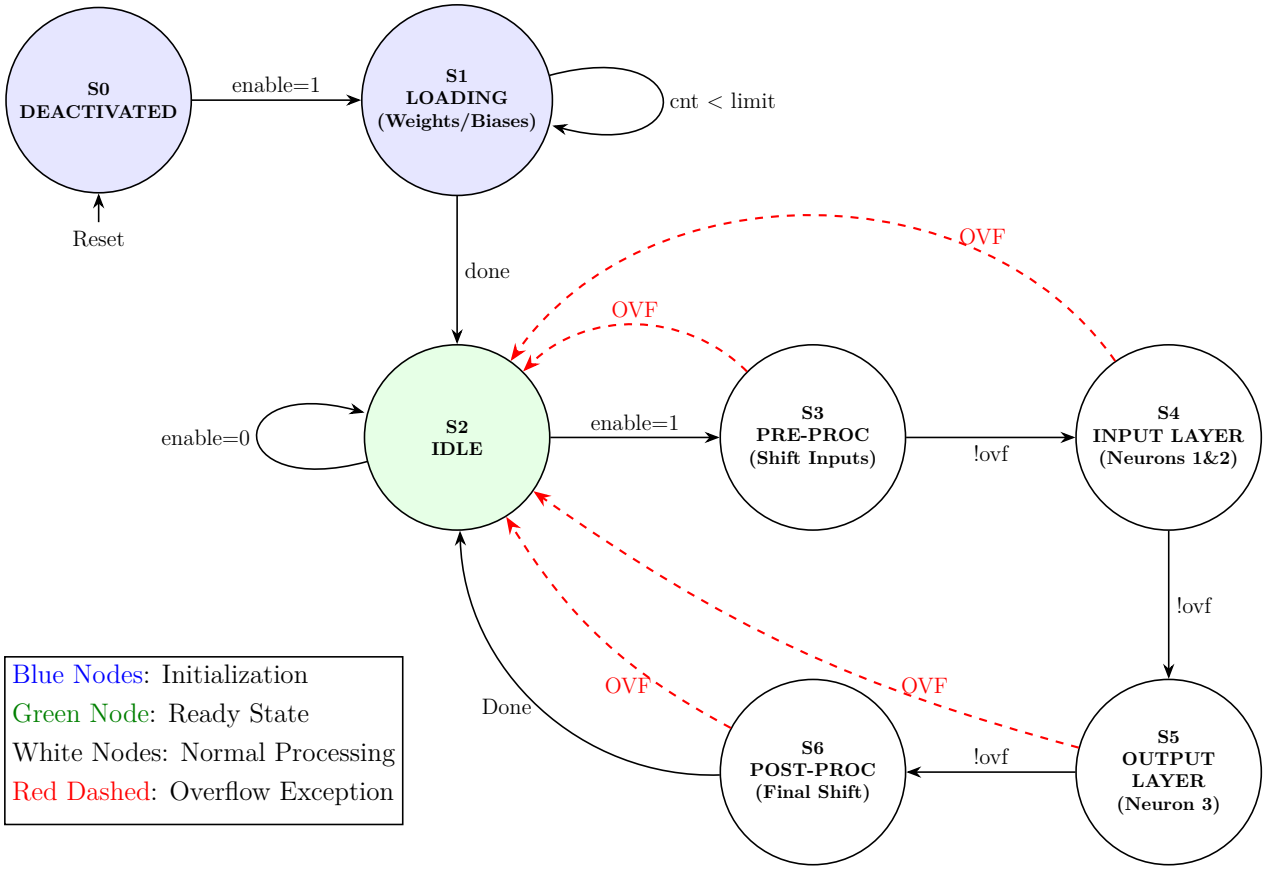

Figure 2: Διάγραμμα καταστάσεων FSM του Neural Network Accelerator.

5.3.3 Περιγραφή Καταστάσεων

Table 7: Καταστάσεις FSM με διαδοχική κωδικοποίηση (8 states)

Κατάσταση	ID	Κωδικός	Περιγραφή
S_DEACTIVATED	S0	000	Αρχική κατάσταση μετά από reset
S_LOADING	S1	001	Φόρτωση βαρών/πολώσεων από τη ROM στο RegFile
S_IDLE	S2	010	Αναμονή για νέες εισόδους (Ready state)
S_PREPROCESS	S3	011	Αριθμητική ολίσθηση δεξιά στις εισόδους
S_INPUT_LAYER	S4	100	Εκτέλεση νευρώνων 1 και 2 (παράλληλα)
S_OUTPUT_LAYER1	S5a	101	Νευρώνας 3 - Πρώτο MAC: inter_3 × w3 + b3
S_OUTPUT_LAYER2	S5b	110	Νευρώνας 3 - Δεύτερο MAC: inter_4 × w4 + mac1_temp
S_POSTPROCESS	S6	111	Αριθμητική ολίσθηση αριστερά στην έξοδο

5.3.4 Σχεδιαστική Απόφαση Υλοποίησης: Διαχωρισμός Output Layer

Παρόλο που η εκφώνηση ζητά 7 στάδια, η πρακτική υλοποίηση στο `nn.v` χρησιμοποιεί **8 καταστάσεις**. Η αιτία εξηγείται παρακάτω.

Το output layer του νευρωνικού δικτύου απαιτεί τον υπολογισμό:

$$\text{inter}_5 = \text{inter}_3 \times \text{weight}_3 + \text{inter}_4 \times \text{weight}_4 + \text{bias}_3 \tag{11}$$

Σύμφωνα με την εκφώνηση αυτή η απαίτηση υλοποιείται ως εξής:

$$\text{mac3} = \text{inter}_3 \times \text{weight}_3 + \text{bias}_3 \quad \text{(MAC1 στο S5a)} \tag{12}$$

$$\text{inter}_5 = \text{inter}_4 \times \text{weight}_4 + \text{mac3} \quad \text{(MAC2 στο S5b)} \tag{13}$$

Ωστόσο: Αν προσπαθήσουμε να εκτελέσουμε και τα δύο MAC σε ένα κύκλο θα δημιουργηθεί **combinatorial loop** διότι:

- Το MAC1 είναι συνδυαστικό κύκλωμα (χωρίς ρολόι)
- Η έξοδος του MAC1 (`mac1_result`) τροφοδοτείται στην είσοδο του MAC2
- Το MAC2 είναι επίσης συνδυαστικό
- Δεν υπάρχει καταχωρητής που να "σπάει" τον βρόχο

```
// Zero-delay loop - Exit 137 (timeout)
mac2_op3 = mac1_result;  // Direct connection
```

Listing 8: Προβληματικός κώδικας (combinatorial loop)

Λύση: Pipelining με Register

Διαχωρίζουμε το output layer σε **δύο καταστάσεις** χρησιμοποιώντας και τις δύο MAC μονάδες σειριακά (όπως αναφέρει και η εκφώνηση):

1. **S_OUTPUT_LAYER1 (S5a):**
   - MAC1 υπολογίζει: `mac1_result = inter_3 × weight_3 + bias_3`
   - Το αποτέλεσμα αποθηκεύεται σε register (`mac1_temp`) στο τέλος του κύκλου
2. **S_OUTPUT_LAYER2 (S5b)**:
   - MAC2 υπολογίζει: `mac2_result = inter_4 × weight_4 + mac1_temp`
   - Χρησιμοποιεί το **registered** αποτέλεσμα (`mac1_temp`) από τον προηγούμενο κύκλο
   - Αποθηκεύει το τελικό αποτέλεσμα στο `inter_5`

5.4 Χειρισμός Υπερχείλισης: Ασυμφωνία στο nn_model.v

Παρατηρήθηκε ότι αν και η εκφώνηση αναφέρει ότι σε περίπτωση overflow η έξοδος πρέπει να είναι ο **"μέγιστος δυνατός θετικός αριθμός"** (δηλαδή το 0x7FFFFFFF), στο παρεχόμενο reference model (nn_model.v) επιστρέφεται η τιμή **0xFFFFFFFF** (γραμμή 95).

Απόφαση: Προσαρμόσαμε το nn.v ώστε να επιστρέφει **0xFFFFFFFF** για να περνάει επιτυχώς το **testbench**, παρόλου που σε 32-bit signed αναπαράσταση, το 0xFFFFFFFF αντιστοιχεί στο -1, όχι στον max positive.

```
case (state)
  //=====================================================================
  // Constants
  //=====================================================================
  localparam [31:0] OVERFLOW_VALUE = 32'hFFFFFFFF; //adapted to nn_model
  localparam [2:0]  NO_OVERFLOW    = 3'b111;
  localparam [2:0]  NO_ZERO        = 3'b111;
endcase
```

Listing 9: Ακολουθούμε το nn_model για να περάσει το testbench

5.5 Register File Pre-fetching

5.5.1 Πρόβλημα: Σύγχρονη Ανάγνωση Register File

Το Register File (Άσκηση 3) υλοποιεί **σύγχρονη ανάγνωση** (registered outputs). Αυτό σημαίνει ότι τα δεδομένα εμφανίζονται στις εξόδους readData **έναν κύκλο μετά** τον ορισμό της διεύθυνσης readReg.

5.5.2 Λύση: Address Pre-fetching

Για να αντιμετωπιστεί αυτή η καθυστέρηση, το FSM εφαρμόζει τεχνική **pre-fetching διευθύνσεων**. Οι διευθύνσεις τίθενται στην **προηγούμενη κατάσταση** από ότι χρειάζονται τα δεδομένα. Αυτή η τεχνική "read-ahead" εξασφαλίζει ότι τα δεδομένα είναι έτοιμα όταν τα χρειάζονται οι ALU/MAC, χωρίς να χρειάζονται επιπλέον κύκλοι αναμονής:

```
case (state)
    // In S2=IDLE we set the addresses for S3=PREPROCESS
    S_IDLE, S_DEACTIVATED: begin
        rf_readReg1 = ADDR_SHIFT_BIAS_1;  // -> data for S_PREPROCESS
        rf_readReg2 = ADDR_SHIFT_BIAS_2;
    end

    // In S3=PREPROCESS we set the addresses for S4=INPUT_LAYER
    S_PREPROCESS: begin
        rf_readReg1 = ADDR_WEIGHT_1;      // -> data for S_INPUT_LAYER
        rf_readReg2 = ADDR_BIAS_1;
        rf_readReg3 = ADDR_WEIGHT_2;
        rf_readReg4 = ADDR_BIAS_2;
    end
    // ... etc.
endcase
```

Listing 10: Pre-fetching διευθύνσεων στο nn.v

5.5.3 Χρονοδιάγραμμα Pre-fetching

Στον παρακάτω πίνακα φαίνεται για κάθε state του FSM (κατά την διάρκεια της κανονικής εκτέλεσης του προγράμματος), τα ποιων διευθύνσεων τα δεδομένα "ζητούνται" απο το Register File και ποια είναι ήδη έτοιμα για επεξεργασία.

Table 8: Χρονοδιάγραμμα Pre-fetching διευθύνσεων RegFile

Τρέχουσα Κατάσταση	**Διευθύνσεις (set)**	**Δεδομένα (available)**
S_IDLE / S_DEACTIVATED	shift_bias_1, shift_bias_2	(Μη έγκυρα)
S_PREPROCESS	weight_1, bias_1, weight_2, bias_2	shift_bias_1, shift_bias_2
S_INPUT_LAYER	weight_3, bias_3	weight_1, bias_1, weight_2, bias_2
S_OUTPUT_LAYER1	weight_4	weight_3, bias_3
S_OUTPUT_LAYER2	shift_bias_3	weight_4
S_POSTPROCESS	–	shift_bias_3

5.6 Testbench Νευρωνικού

Το testbench εκτελεί 100 επαναλήψεις με 3 τεστ ανά επανάληψη, όπου κάθε επανάληψη τεστάρει το νευρωνικό για τυχαία παραγόμενο ζεύγος εισόδων με εύρος τιμών:

1. **Κανονικό εύρος:** [-4096, 4095]
2. **Θετικό overflow:** [MAX_POS/2, MAX_POS]
3. **Αρνητικό overflow:** [MAX_NEG, MAX_NEG/2]

5.6.1 Latency Calibration

Η συνάρτηση αναφοράς `nn_model` υπολογίζει το αναμενόμενο αποτέλεσμα για σύγκριση. Ωστόσο, καθώς αυτό είναι στιγμιαίο, αλλά το κύκλωμα του νευρωνικού έχει καθυστέρηση λόγω του FSM, αν γίνει σύγκριση πολύ νωρίς, το τεστ αποτυγχάνει ακόμα και αν το κύκλωμα λειτουργεί σωστά.

Αφού τα έγκυρα δεδομένα εμφανίζονται στο νευρωνικό ακριβώς 6 κύκλους μετά τον παλμό enable (σύμφωνα με αριθμό καταστάσεων του FSM που διατρέχονται), εφαρμόστηκε το εξείς Latency:

```
// Starting next test:
@(posedge clk);
    enable = 1;
@(posedge clk);
    enable = 0;
// Generating expected result
expected_val = nn_model(in1, in2);
// Wait 6 cycles
repeat(6) @(posedge clk);
// Now the both results are ready and can be compared...
```

5.6.2 Αποτελέσματα Testbench

Τα αποτελέσματα της προσομοίωσης, όπως καταγράφηκαν από το testbench, παρουσιάζονται παρακάτω στο Figure 3.

```
Loading weights...
Weights loaded. Starting tests...

============================================================
FINAL REPORT
============================================================
Total Test Cases: 300
PASSED:           300
FAILED:           0
Success Rate:     300 / 300
RESULT: SUCCESS (All tests passed)
============================================================
```

Figure 3: Στιγμιότυπο κονσόλας από την εκτέλεση του `tb_nn.v`

Όπως φαίνεται από την τελική αναφορά, το κύκλωμα πέρασε επιτυχώς και τις 300 περιπτώσεις ελέγχου (**100% επιτυχία**), επιβεβαιώνοντας την ορθότητα της σχεδίασης του FSM, της διαχείρισης των καταχωρητών και της λογικής ανίχνευσης υπερχείλισης.

5.7 Κυματομορφές Νευρωνικού Δικτύου

Παρακάτω, παρουσιάζονται οι κυματομορφές από την προσομοίωση του tb nn.v (Figure 4), εστιάζοντας στα πρώτα στάδια της εκτέλεσης (Loading και πρώτες επαναλήψεις ελέγχου). Παρατηρούμε τα εξής στάδια λειτουργίας:

- **Αρχικοποίηση & Φόρτωση (0 - 200ns):** Στην αρχή βλέπουμε την ενεργοποίηση του `resetn` και τον πρώτο παλμό του `enable`. Το σύστημα μεταβαίνει στην κατάσταση `S_LOADING` (δεν φαίνεται το state signal αλλά φαίνεται η καθυστέρηση μέχρι την ετοιμότητα).
- **Κανονική Λειτουργία (First test, 200 - 270ns):** Εφαρμόζονται οι είσοδοι `input_1` και `input_2` και δινεται enable. Μετά από έναν κύκλο το αναμενόμενο αποτέλεσμα (`expected`) εμφανίζεται στην κυματομορφή, ενώ το ίδιο αποτέλεσμα εμφανίζεται στην έξοδο του νευρωνικού (`final_output`) σε 6 κύκλους λόγω του FSM (όπως αναφέραμε στην παράγραφο 5.6.1). Καθώς και οι δύο έξοδοι συμφωνούν στην τιμή `0xfffd5134`, το σήμα `pass_count` αυξάνεται σε 1. Δεν υπάρχει υπερχείλιση (`total_ovf = 0`).
- **Διαχείριση Υπερχείλισης (Second test, 280 - 310ns):** Στο δεύτερο test, παρατηρούμε ότι το σήμα `total_ovf` γίνεται 1 (high). Αυτό υποδεικνύει ότι προέκυψε υπερχείλιση κατά τους υπολογισμούς. Όπως είναι αναμενόμενο, η έξοδος `final_output` οδηγείται στην τιμή `0xffffffff` (Saturation Value), που ταυτίζεται με το `expected` value του μοντέλου αναφοράς, επιβεβαιώνοντας ότι το κύκλωμα διαχειρίζεται σωστά τα overflows και συμμορφώνεται με το μοντέλο.

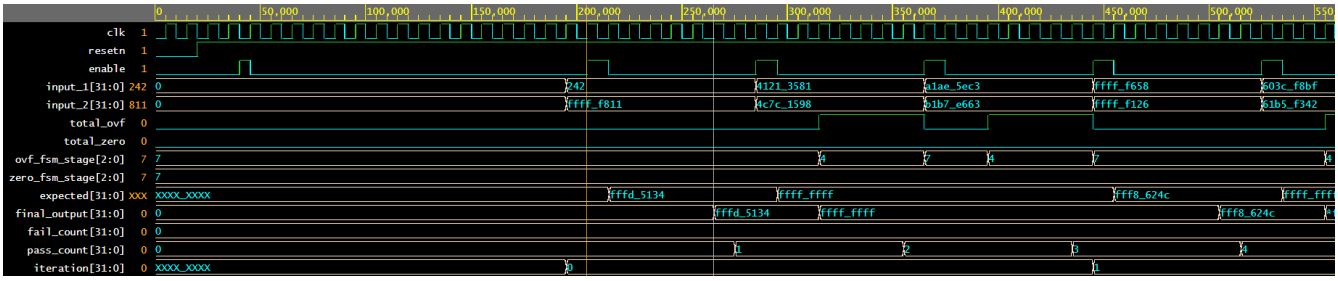

Figure 4: Κυματομορφές προσομοίωσης του Νευρωνικού Δικτύου. Φαίνεται η αρχικοποίηση και οι πρώτες επαναλήψεις ελέγχου (Normal και Overflow cases).

Συνολική Εικόνα: Το σήμα `fail_count` παραμένει σταθερά στο 0, ενώ το `pass_count` αυξάνεται σε κάθε θετικό μέτωπο του ρολογιού μετά την ολοκλήρωση κάθε πράξης, αποδεικνύοντας την ορθή λειτουργία του συστήματος για διαφορετικά σενάρια εισόδου (Figure 5).

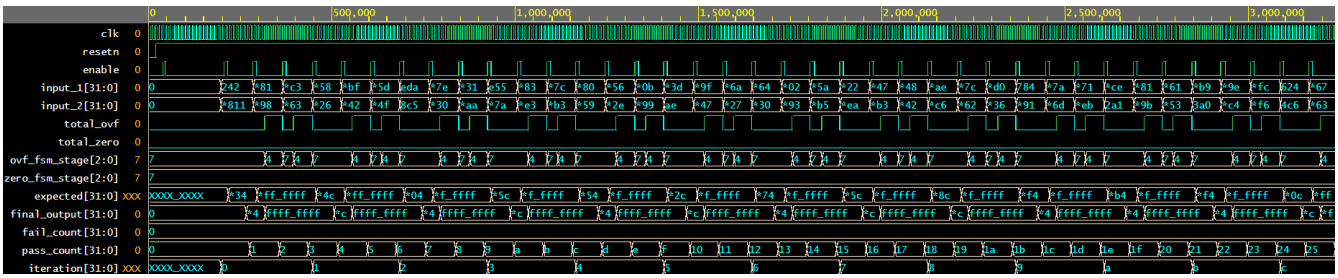

Figure 5: Κυματομορφές προσομοίωσης του Νευρωνικού Δικτύου (Συνολική Εικόνα).

6 Αναφορές

1. Ψηφιακή Σχεδίαση, 6η Έκδοση, Morris Mano and Michael Ciletti
2. IEEE Standard for Verilog Hardware Description Language (IEEE 1364-2005)
3. Playground EDA - Online Verilog Simulator: https://eda-playground.readthedocs.io

Τέλος, σημειώνεται ότι κατά τη διάρκεια εκπόνησης της παρούσας εργασίας χρησιμοποιήθηκε το γλωσσικό μοντέλο Claude Opus 4.5 LLM ως βοηθητικό εργαλείο.