PROJECT ΕΞΑΜΗΝΟΥ

Ψηφιακά Συστήματα Hardware σε Χαμηλά Επίπεδα

Λογικής ΙΙ

Δήμητρα Ελένη Λεούδη
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Εαρινό Εξάμηνο 2024/2025

ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

Περιεχόμενα

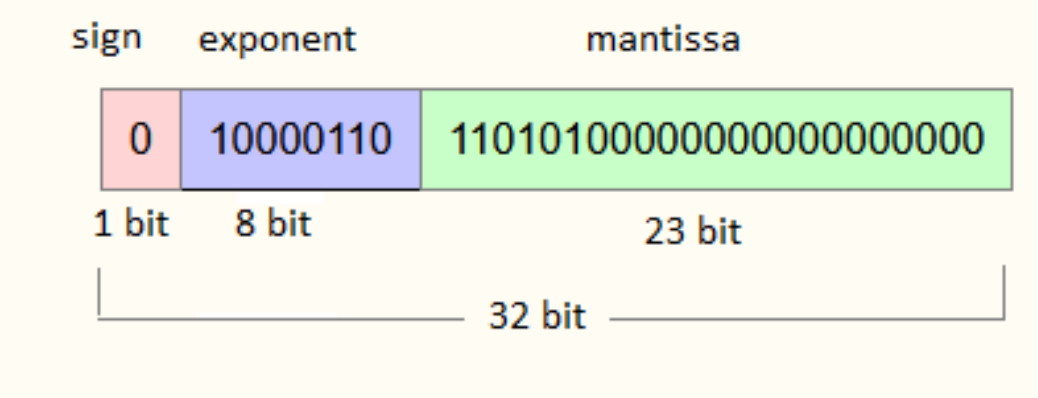
Εισαγωγή

Στο πλαίσιο αυτής της εργασίας, στόχος είναι η υλοποίηση ενός πολλαπλασιαστή κινητής υποδιαστολής (floating point multiplier) σε γλώσσα SystemVerilog. Η υλοποίηση αφορά αριθμούς κινητής υποδιαστολής μονής ακρίβειας (32-*bit*), σύμφωνα με τις προδιαγραφές του προτύπου IEEE 754 Single Precision.

Ο σχεδιασμός του πολλαπλασιαστή διαχωρίζεται σε τρία βασικά στάδια, το καθένα από τα οποία υλοποιήθηκε ως ξεχωριστό module:

- το στάδιο κανονικοποίησης,
- το στάδιο στρογγυλοποίησης,
- και το στάδιο διαχείρισης εξαιρέσεων.

Για την επαλήθευση της λειτουργικότητας και της ορθότητας του πολλαπλασιαστή, αναπτύχθηκαν επιπλέον testbenches και assertions, τα οποία θα παρουσιαστούν και θα αναλυθούν λεπτομερώς στο πλαίσιο αυτής της αναφοράς.



Main Module - fp_mult.sv

Το κύριο module fp_mult υλοποιεί έναν πολλαπλασιαστή κινητής υποδιαστολής (single precision, 32-bit) σύμφωνα με το πρότυπο $IEEE$ 754. Σκοπός του είναι να λάβει δύο αριθμούς κινητής υποδιαστολής και να υπολογίσει το γινόμενό τους με σωστή διαχείριση πρόσημου, εκθέτη, *mantissa*, στρογγυλοποίησης και εξαιρέσεων.

1. Inputs-Outputs

Οι κύριες είσοδοι του *module* είναι:

- a, b: Δύο 32-*bit* αριθμοί κινητής υποδιαστολής μονής ακρίβειας.
- rnd: 3-*bit* σήμα επιλογής τρόπου στρογγυλοποίησης rounding mode).
- clk, rst: Σήματα ρολογιού και επαναφοράς.

Ενώ οι έξοδοι είναι:

- z: 32-*bit* αποτέλεσμα του πολλαπλασιασμού των a, b.
- status: 8-*bit* *flags* κατάστασης που ενημερώνουν για εξαιρέσεις όπως overflow, underflow, inexact κτλ.

2. Υπολογισμός πρόσημου

Το πρόσημο του γινομένου υπολογίζεται με XOR των *bit* πρόσημου των εισόδων, δηλαδή:

```
1 assign sign = a[31] ^ b[31];
```

3. Υπολογισμός εκθέτη

Ο εκθέτης του αποτελέσματος προκύπτει ως:

```
1 assign exp_add = ({2'b00, a[30:23]}+{2'b00, b[30:23]})-10'd127;
```

όπου το *bias* για *single precision* είναι 127.
Η χρήση 10-*bit*, αντί για 8 στους εκθέτες αποτρέπει $overflow$ κατά την πρόσθεση.

4. Υπολογισμός *mantissa*

Οι *mantissa* των δύο αριθμών πολλαπλασιάζονται αφού προηγουμένως έχουν προσαυξηθεί με το *leading − one* που υποδηλώνει την κανονικοποιημένη μορφή:

```
1 assign mantissa_mult = {1'b1, a[22:0]}*{1'b1, b[22:0]};
```

Το αποτέλεσμα είναι 48-*bit* για να διατηρηθεί η ακρίβεια πριν τη στρογγυλοποίηση.

5. Κανονικοποίηση

Στην συνέχεια αρχικοποιείται το normalize_mult, δηλαδή το στάδιο της στρογγυλοποίησης

6. Pipeline stage

Το επόμενο σημαντικό στάδιο του κυκλώματος είναι η υλοποίηση *pipeline* μετά το στάδιο της κανονικοποίησης. Αυτό το στάδιο απαιτείται για να συγχρονιστούν τα ενδιάμεσα σήματα και να εξασφαλιστεί η ομαλή ροή δεδομένων προς τα επόμενα στάδια. Όλα τα σχετικά σήματα αποθηκεύονται σε *pipeline* καταχωρητές με τη βοήθεια ενός μπλοκ always_ff, το οποίο ενεργοποιείται στην ανερχόμενη ακμή του ρολογιού και περιλαμβάνει ενεργό χαμηλό σήμα επαναφοράς. Σε περίπτωση επαναφοράς, τα καταχωρημένα σήματα μηδενίζονται. Αντίθετα, κατά την κανονική λειτουργία, οι τιμές τους ενημερώνονται με τα αντίστοιχα αποτελέσματα της κανονικοποίησης, αλλά και με άλλες σημαντικές πληροφορίες, όπως τα σήματα εισόδου a, b και η επιλογή τρόπου στρογγυλοποίησης round_mode. Επίσης, το σήμα mantissa_round_input δημιουργείται κατάλληλα με το leading one και την κανονικοποιημένη μαντισσα. Με αυτόν τον τρόπο επιτυγχάνεται αποδοτική υλοποίηση pipeline, επιτρέποντας την εκκίνηση νέων πράξεων πριν ολοκληρωθούν οι προηγούμενες, γεγονός που ενισχύει την απόδοση του πολλαπλασιαστή.

Το στάδιο λειτουργεί ως εξής:

```
always_ff @(posedge clk or negedge rst) begin
    if(!rst) begin
        sign_pipe <= 1'b0;
        exp_pipe <= 10'b0;
        mant_pipe <= 23'b0;
        a_pipe <= 32'b0;
        b_pipe <= 32'b0;
        guard_pipe <= 1'b0;
        sticky_pipe <= 1'b0;
        round_mode_pipe <= 3'b0;
        mantissa_round_input <= 24'b0;
    end
    else begin
        sign_pipe <= a[31] ^ b[31];
        exp_pipe <= exp_norm;
        mant_pipe <= mant_norm;
        a_pipe <= a;
        b_pipe <= b;
        mantissa_round_input <= {1'b1, mant_norm};
        guard_pipe <= guard;
        sticky_pipe <= sticky;
        round_mode_pipe <= round_mode;
    end
end
```

7. Στρογγυλοποίηση

Μετά γίνεται αρχικοποίηση του στάδιου της στρογγυλοποίησης, δηλαδή του round_mult.

8. Έλεγχος $overflow$ και $underflow$

Με βάση το αποτέλεσμα του εκθέτη μετά το στάδιο της στρογγυλοποίησης, γίνεται έλεγχος για $overflow$ ή $underflow$, δηλαδή ελέγχεται αν ο εκθέτης

φτάνει στο μέγιστο ή αν μηδενίζεται. Αυτές οι πληροφορίες που θα προκύψουν, θα περαστούν στο στάδιο διαχείρισης εξαιρέσεων. Ο έλεγχος γίνεται ως εξής:

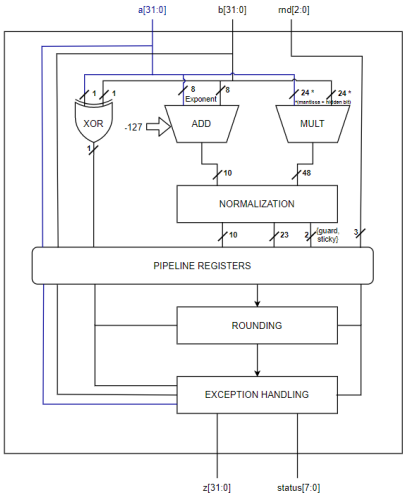
```
always_comb begin

    if (round_exponent[7:0] == 8'b1111_1111)
                overflow = 1;
    else
                overflow = 0;

    if (round_exponent[7:0] == 8'b0000_0000)
                underflow = 1;
    else
                underflow = 0;
end
```

9. Διαχείριση εξαιρέσεων

Τέλος αρχικοποιείται και το στάδιο στο οποίο γίνεται η διαχείρηση των εξεραίσεων, δηλαδή το exceptional_mult.



Normalization Module - normalize_mult.sv

Το module normalize_mult είναι υπεύθυνο για το στάδιο της κανονικοποίησης, δηλαδή την κανονικοποίηση του αποτελέσματος του πολλαπλασιασμού της mantissa, του εκθέτη, καθώς και για τον υπολογισμό των *guard* και *sticky bits*. Στόχος του είναι να φέρει την *mantissa* σε κανονικοποιημένη μορφή, δηλαδή να διατηρεί ένα μη μηδενικό δυαδικό ψηφίο πριν την υποδιαστολή, και να προσαρμόσει αναλόγως τον εκθέτη.

Είσοδοι:

- P (48-*bit*): το αποτέλεσμα του πολλαπλασιασμού των *mantissas*
- S (10-*bit*): το άθροισμα των εκθετών μείον το *bias*

Έξοδοι:

- norm_mantissa (23-*bit*): η κανονικοποιημένη *mantissa*
- norm_exponent (10-*bit*): ο κανονικοποιημένος εκθέτης
- guard: το guard bit
- sticky: το sticky bit

Η κανονικοποίηση εξαρτάται από το MSB του αποτελέσματος $P[47]$. Το δυαδικό σημείο τοποθετείται μεταξύ των $P[46]$ και $P[45]$. Αν το $P[47]$ είναι 1, αυτό σημαίνει ότι υπάρχει «υπερχείλιση» στην *mantissa* και απαιτείται μετατόπιση του δυαδικού σημείου αριστερά κατά 1 θέση και ταυτόχρονη αύξηση του εκθέτη κατά 1. Αν είναι 0, δεν απαιτείται μετατόπιση και ο εκθέτης παραμένει ως έχει.

Τα *guard* και *sticky bits* παίζουν κρίσιμο ρόλο στη στρογγυλοποίηση, για αυτό φροντίσουμε να τα κρατήσουμε από αυτό το στάδιο και να τα περάσουμε ως εισόδους στο επόμενο.

- Το *guard bit* είναι το επόμενο δυαδικό ψηφίο αμέσως μετά την 23η θέση της *mantissa*.
- Το *sticky bit* είναι το λογικό OR όλων των υπολειπόμενων *bit* μετά το *guard bit*.

6. Round Module - round_mult.sv

Το round_mult είναι υπεύθυνο για την υλοποίηση της στρογγυλοποίησης της mantissa, ανάλογα με το σήμα λειτουργίας round_mode. Οι είσοδοι περιλαμβάνουν την mantissa έτοιμη για στρογγυλοποίηση, τα σήματα guard και sticky, καθώς και το πρόσημο του αριθμού. Οι έξοδοι είναι η mantissa μετά τη στρογγυλοποίηση, το πρόσημο και το σήμα inexact που δηλώνει αν το αποτέλεσμα ήταν ακριβές.

Η λογική στρογγυλοποίησης καθορίζεται με βάση το round_mode. Για παράδειγμα, στην περίπτωση IEEE_near, αν το guard *bit* είναι 1 και ταυτόχρονα είτε το sticky είτε το τελευταίο *bit* της mantissa είναι 1, τότε προστίθεται 1 στη mantissa. Το απόσπασμα υλοποίησης του είναι το εξής:

```
1 if (pipe_guard && (pipe_sticky  mantissa_extended[0])) begin
2     mantissa_extended = mantissa_extended + 1;
3 end
```

Αντίστοιχα, για άλλες τιμές του round_mode (IEEE_zero, IEEE_pinf, IEEE_ninf, near_up, away_zero), οι συνθήκες στρογγυλοποίησης αλλάζουν ώστε να προσαρμοστούν στις απαιτήσεις κάθε περίπτωσης.

Το αποτέλεσμα της στρογγυλοποίησης αποθηκεύεται στο round_mantissa, ενώ το inexact υπολογίζεται ως το λογικό OR των pipe_guard και pipe_sticky, υποδεικνύοντας ότι το αποτέλεσμα δεν ήταν ακριβές.

Με αυτόν τον τρόπο, το στάδιο στρογγυλοποίησης ολοκληρώνει την προετοιμασία του τελικού αποτελέσματος πριν από την είσοδο στο στάδιο διαχείρισης εξαιρέσεων.

Name	Values	Description
round	IEEE_near	IEEE round to nearest, even. Round to the nearest representable value, if both are equally near, output the result with an even significand.
	IEEE_zero	IEEE round towards zero.
	IEEE_pinf	IEEE round to +Infinity.
	IEEE_ninf	IEEE round to -Infinity.
	near_up	Round to the nearest representable value, if both are equally near, output the result closer to +Infinity.
	away_zero	Round away from zero.

Exception Handling Module - exception_mult.sv

Το exception_mult *module* είναι υπεύθυνο για τη διαχείριση των εξαιρέσεων και την τελική παραγωγή του αποτελέσματος z. Με βάση τα σήματα underflow, overflow και inexact, καθώς και τις τιμές των εισόδων a, b και z_calc, το *module* προσδιορίζει εάν το τελικό αποτέλεσμα πρέπει να είναι μηδέν, άπειρο ή κανονικός αριθμός, ακολουθώντας τις προδιαγραφές του προτύπου IEEE-754, με μερικές εξαιρέσεις, όπως ορίζονται από την εκφώνηση.

Είσοδοι:

- a, b (32-*bit*): Οι αριθμοί εισόδου κινητής υποδιαστολής.
- z_calc (32-*bit*): Το αποτέλεσμα του προηγούμενου σταδίου, έτοιμο προς έλεγχο.
- $underflow$, $overflow$ (1-*bit*): Σήματα που υποδεικνύουν υποχείλιση ή υπερχείλιση.
- inexact (1-*bit*): Σήμα που δηλώνει ανακριβές αποτέλεσμα.
- round_mode (3-*bit*): Τρόπος στρογγυλοποίησης.

Έξοδοι:

- z (32-*bit*): Το τελικό αποτέλεσμα σε μορφή IEEE-754.
- zero_f, inf_f, nan_f, tiny_f, huge_f, inexact_f (1-*bit*): Σήματα κατάστασης που υποδεικνύουν ειδικές περιπτώσεις ή εξαιρέσεις.

Η λειτουργία του *module* βασίζεται σε δύο βοηθητικές συναρτήσεις:

- num_interp: Ερμηνεύει τον τύπο ενός αριθμού (π.χ. μηδέν, άπειρο, κανονικός αριθμός) με βάση το πεδίο εκθέτη.
- z_num: Επιστρέφει την κατάλληλη δυαδική αναπαράσταση για έναν τύπο αριθμού.

Ανάλογα με τον συνδυασμό των τύπων εισόδου (π.χ. zero × zero, zero × inf, norm × norm), το *module*:

- Παράγει απευθείας μηδενικό ή άπειρο αποτέλεσμα.
- Εφαρμόζει ελέγχους overflow/underflow, λαμβάνοντας υπόψη το round_mode, ώστε να αποφασίσει εάν το αποτέλεσμα πρέπει να γίνει inf ή max_norm (σε περίπτωση overflow), ή zero ή min_norm (σε περίπτωση underflow).
- Ενημερώνει τα σήματα κατάστασης όπως zero_f, inf_f, tiny_f, huge_f και inexact_f για τη σωστή ένδειξη των εξαιρέσεων.

Παράδειγμα του τρόπου που ελέγχεται το overflow:

```
if (overflow) begin
    huge_f = 1;
    inexact_f = 1;
    case (round_mode)
        IEEE_near, away_zero: begin
            z = {z_calc[31], z_num(INF)};
            inf_f = 1;
        end
        IEEE_zero: begin
            z = {z_calc[31], z_num(MAX_NORM)};
        end
        // ...
    endcase
end
```

Με αυτόν τον τρόπο, ολοκληρώνεται τη διαδικασία ελέγχου και διαχείρισης των εξαιρέσεων, παράγοντας το τελικό αποτέλεσμα z και τα σήματα κατάστασης που απαιτούνται από το πρότυπο IEEE-754.

Testbench Module - fp_mult_tb.sv

Το fp_mult_tb αποτελεί το *testbench* για την επαλήθευση του floating point multiplier σε επίπεδο συστήματος. Χρησιμοποιεί το *module* fp_mult_top, το οποίο υλοποιεί την αρχιτεκτονική του πολλαπλασιαστή με όλα τα στάδια, και υποστηρίζει δοκιμές σε διαφορετικούς τρόπους στρογγυλοποίησης, καθώς και δοκιμές με *corner cases*.

Κύρια χαρακτηριστικά:

- Το ρολόι clk δημιουργείται με περίοδο $10\,ns$.
- Αρχικοποίηση του reset για συγχρονισμό όλων των καταχωρητών.
- Δοκιμές για όλους τους τρόπους στρογγυλοποίησης (IEEE_near, IEEE_zero, IEEE_pinf, IEEE_ninf, near_up, away_zero).
- Χρήση της εξωτερικής συνάρτησης multiplication για τον υπολογισμό του αναμενόμενου αποτελέσματος expected_z.
- Καθυστέρηση 3 κύκλων για την εξομοίωση του pipeline delay.

Διαχείριση corner cases: Ένας πίνακας corner_case_t περιλαμβάνει ειδικές περιπτώσεις, όπως: NaN, infinity, κανονικούς αριθμούς, υποκανονικούς αριθμούς και μηδενικά. Οι συναρτήσεις corner_case_to_bits και corner_case_to_string παρέχουν τον δυαδικό κωδικό και το αντίστοιχο όνομα για κάθε περίπτωση.

Έλεγχος corner cases: Με διπλούς βρόχους for, όλες οι συνδυαστικές περιπτώσεις ελέγχονται διεξοδικά. Σε περίπτωση NaN, προσαρμόζεται η έξοδος ώστε να συμμορφώνεται με το πρότυπο IEEE-754.

Συνολικά, το *testbench* αυτό διασφαλίζει την πληρότητα του ελέγχου για όλους τους συνδυασμούς εισόδων, στρογγυλοποίησης και corner cases, επικυρώνοντας τη σωστή λειτουργία και την αξιοπιστία του πολλαπλασιαστή κινητής υποδιαστολής.

Assertion Module - test_status_bits.sv

Το test_status_bits *module* περιέχει λογικούς ελέγχους (assertions) που εκτελούνται σε κάθε θετική ακμή του ρολογιού clk. Στόχος του είναι να εντοπίσει πιθανές ασυμβατότητες ή λάθη στα status bits (8-*bit*), τα οποία συνοψίζουν την κατάσταση του πολλαπλασιαστή.

Λειτουργία:

- Ελέγχει αν υπάρχουν παράλληλα ενεργά *bits* που δεν θα έπρεπε να συνυπάρχουν, όπως:
  - zero και infinity
  - zero και invalid
  - zero και tiny
  - infinity και huge, κλπ.
- Αν εντοπιστεί τέτοια περίπτωση, εμφανίζεται μήνυμα λάθους (*error*) με συγκεκριμένη περιγραφή.
- Η λειτουργία εκτελείται μόνο αν το status_bits δεν είναι άγνωστο (*isunknown*).

Ενδεικτικός έλεγχος:

```
1 assert (!(status_bits[0] & status_bits[1]))
2     else $error("Zero and Infinity bits are both asserted");
```

Με αυτό τον τρόπο, το *module* test_status_bits διασφαλίζει ότι οι σημαίες κατάστασης (status flags) που παράγει ο πολλαπλασιαστής είναι συνεπείς μεταξύ τους και δεν παραβιάζουν τους κανόνες του προτύπου IEEE-754.

Assertion Module - test_status_z_combinations.sv

Το test_status_z_combinations *module* περιέχει ελέγχους ορθότητας για τον συνδυασμό των status_bits και της εξόδου z (καθώς και των εισόδων a και b). Οι έλεγχοι αυτοί διασφαλίζουν ότι η σημαία που τίθεται (π.χ. ZERO, INF, NAN, TINY, HUGE) αντιστοιχεί πράγματι σε σωστή μορφή του z ή στις συνθήκες των εισόδων.

Κύρια σημεία ελέγχου:

- ZERO: Αν το status_bit ZERO είναι ενεργό, το πεδίο εκθέτη του z πρέπει να είναι μηδέν.
- INF: Αν το INF είναι ενεργό, ο εκθέτης του z πρέπει να είναι $8'hFF$.
- NAN: Ο έλεγχος βασίζεται στο αν οι εκθέτες των εισόδων a και b (3 κύκλους πριν) καλύπτουν την περίπτωση παραγωγής NaN.
- HUGE: Το z πρέπει να είναι είτε infinity ($8'hFF$) είτε ο μέγιστος κανονικοποιημένος αριθμός ($8'hFE$ και μαντισσα $0x7FFFFF$).
- TINY: Το z πρέπει να είναι είτε μηδέν ($8'h00$) είτε ο ελάχιστος κανονικοποιημένος ($8'h01$ και *mantissa* μηδενική).

Ενδεικτικό απόσπασμα ελέγχου:

```
1 assert property (@(posedge clk) status_bits[INF] |-> (z[30:23] ==
  ↪ 8'hFF))
2     else $error("ERROR: INF status bit is set but exponent of 'z'
      ↪ is not all ones.");
```

Η χρήση της SystemVerilog sequence nan_condition επιτρέπει τον έλεγχο για NaN βάσει των εισόδων a και b με καθυστέρηση 3 κύκλων. Αυτό διασφαλίζει την ακρίβεια του ελέγχου για πιο σύνθετες εξαρτήσεις.

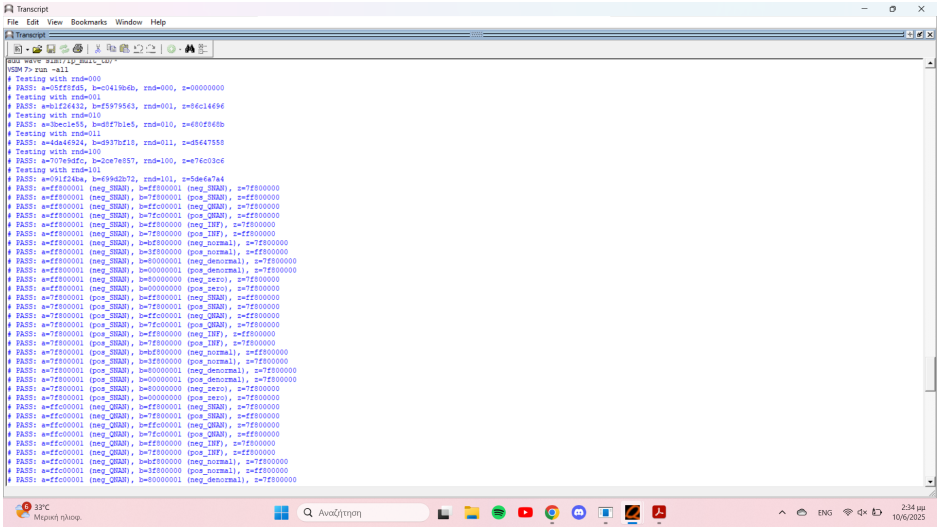
Με τον τρόπο αυτό, το *module* test_status_z_combinations παρέχει μία ολοκληρωμένη πλατφόρμα ελέγχου για την αξιοπιστία και την ακρίβεια της σημασιοδότησης των αποτελεσμάτων του πολλαπλασιαστή κινητής υποδιαστολής.

Αποτελέσματα Προσομοίωσης και Κυματομορφές

Σε αυτή την ενότητα παρατίθενται τα αποτελέσματα της προσομοίωσης, καθώς και οι αντίστοιχες κυματομορφές. Τα αποτελέσματα αποδεικνύουν την ορθή λειτουργία του πολλαπλασιαστή, καθώς και την πλήρη κάλυψη όλων των εξαιρέσεων και ακραίων περιπτώσεων (*corner cases*).

1. Έλεγχοι

Η παρακάτω εικόνα δείχνει τα μηνύματα επιτυχίας (PASS) ή αποτυχίας (FAIL) που προκύπτουν κατά την εκτέλεση των testbenches. Παρατηρείται ότι όλες οι συνθήκες επαληθεύτηκαν επιτυχώς:



1) Έλεγχος με τυχαίες τιμές εισόδων: Το testbench δημιουργεί τυχαίες τιμές για τις εισόδους a και b χρησιμοποιώντας τη συνάρτηση $urandom(). Το αποτέλεσμα του πολλαπλασιασμού (z) συγκρίνεται με το αναμενόμενο αποτέλεσμα, το οποίο υπολογίζεται μέσω της συνάρτησης multiplication. Επειδή η multiplication υπολογίζει το αποτέλεσμα ακαριαία (χωρίς καταχωρητές καθυστέρησης), ενώ το fp_mult έχει καθυστέρηση 3 κύκλων, έγινε κατάλληλη ευθυγράμμιση του z μέσω καταχωρητών. Σε περίπτωση αναντιστοιχίας, θα εμφανιζόταν FAIL μήνυμα στο transcript.

2) Έλεγχος με corner cases: Το δεύτερο μέρος του testbench επικεντρώνεται σε όλους τους πιθανούς συνδυασμούς corner cases για τις εισόδους a και b. Για τον σκοπό αυτόν, δημιουργήθηκε μια συνάρτηση που αντιστοιχεί τα δυαδικά μοτίβα των corner cases με περιγραφικές συμβολοσειρές. Αυτές οι corner cases περιλαμβάνουν ειδικούς αριθμούς όπως NaN, infinity, zero, denormal κτλ. Ολοκληρώθηκαν 144 έλεγχοι (12×12 συνδυασμοί), όπως απαιτείται από την εκφώνηση. Κάθε περίπτωση επαληθεύτηκε με την αναμενόμενη τιμή από τη multiplication και επιβεβαιώθηκε με PASS μηνύματα.

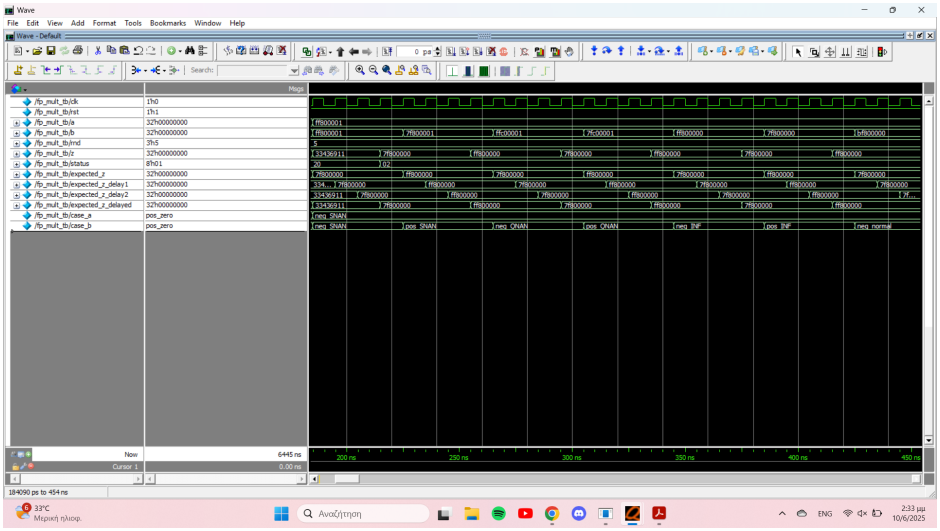
3) Έλεγχοι με Assertions: Κατά τη διάρκεια της εκτέλεσης των testbenches, ενεργοποιήθηκαν τα assertion modules test_status_bits.sv και test_status_z_combinations.sv, τα οποία περιείχαν πολλαπλούς λογικούς ελέγχους για την επαλήθευση της συνέπειας και ορθότητας των σημάτων εξόδου. Σημαντικό είναι ότι:

- Δεν εμφανίστηκαν αποτυχίες, γεγονός που επιβεβαιώνει ότι όλες οι συνθήκες λειτουργίας ικανοποιούνται απόλυτα.
- Παρακολούθησαν κρίσιμα σήματα, όπως τα status bits, τους εκθέτες, τις μαντισσα, και τις εξαιρέσεις NaN, Inf, Zero κτλ.
- Η απουσία οποιουδήποτε μηνύματος σφάλματος (*error*) αποτελεί ισχυρή απόδειξη για την αξιοπιστία της υλοποίησης.

Οι τρεις κατηγορίες ελέγχων διασφαλίζουν την πληρότητα του testbench και την αξιοπιστία του πολλαπλασιαστή. Με βάση τα παραπάνω, ολοκληρώθηκε με επιτυχία η επαλήθευση του πολλαπλασιαστή κινητής υποδιαστολής, ικανοποιώντας όλες τις απαιτήσεις της εκφώνησης.

2. Κυματομορφές σημάτων

Οι επόμενες εικόνες παρουσιάζουν τις κυματομορφές των βασικών σημάτων του *testbench* κατά την εκτέλεση του.



Απεικονίζουν την αλληλουχία σημάτων κατά τη διάρκεια της εξομοίωσης. Συγκεκριμένα:

- Παρατηρείται σωστός χρονισμός του ρολογιού (clk) με περίοδο $10ns$ όπως ορίζεται από την εκφώνηση.
- Τα σήματα εισόδων a και b ενημερώνονται συγχρονισμένα με το ρολόι, εξασφαλίζοντας την ομαλή εκκίνηση κάθε νέας δοκιμής.
- Η έξοδος z (αποτέλεσμα πολλαπλασιασμού) εμφανίζεται με καθυστέρηση 3 κύκλων, επιβεβαιώνοντας τη σωστή λειτουργία του pipeline.
- Τα status bits εμφανίζονται αμέσως μετά την ολοκλήρωση κάθε υπολογισμού, δηλώνοντας τις καταστάσεις όπως zero, inf, nan, huge, tiny, inexact.
- Οι τιμές των expected_z και z συγκλίνουν απόλυτα για κάθε περίπτωση, αποδεικνύοντας την ακρίβεια του πολλαπλασιαστή.

Οι κυματομορφές αυτές αποτελούν ένδειξη ότι όλες οι φάσεις (από την εισαγωγή δεδομένων, μέχρι την παραγωγή της τελικής εξόδου και των status bits) εκτελούνται με απόλυτο συγχρονισμό και ακρίβεια. Επιβεβαιώνεται ότι:

- Δεν υπάρχουν «μεταβατικές καταστάσεις» ή *glitches* στα σήματα εξόδου.
- Η καθυστέρηση λόγω pipeline είναι σταθερή και αναμενόμενη (3 κύκλοι).
- Οι τιμές των *corner cases* αναγνωρίζονται σωστά από τα αντίστοιχα *status flags*.

Κατά συνέπεια, η συμπεριφορά που αποτυπώνεται στις κυματομορφές διαβεβαιώνει την πλήρη και ορθή λειτουργία του πολλαπλασιαστή, τηρώντας απόλυτα τις προδιαγραφές του προτύπου IEEE-754 και της εκφώνησης.