Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών

ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ ΣΕ ΧΑΜΗΛΑ ΕΠΙΠΕΔΑ ΛΟΓΙΚΗΣ ΙΙ

ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ

Εαρινό εξάμηνο 2023/24

Δεϊρμεντζόγλου Ιωάννης
Α.Ε.Μ.: 10015
Email: deirmentz@ece.auth.gr

Περιεχόμενα

1. Εισαγωγή

Στην παρούσα εργασία στο πλαίσιο του μαθήματος αντικείμενο είναι η υλοποίηση ενός πολλαπλασιαστή κινητής υποδιαστολής με βάση το πρότυπο IEEE-754.

Η ”floating-point” αριθμητική χρησιμοποιείται ευρέως σε πολλούς τομείς, και ειδικά για επιστημονικούς υπολογισμούς καθώς και για επεξεργασία σημάτων. Με το πέρασμα των χρόνων έχουν αναπτυχθεί πολλές προσεγγίσεις σε αριθμούς. Η ”floating-point” αριθμητική είναι η πιο αποτελεσματική αναπαράσταση πραγματικών αριθμών, και ειδικά στον τομέα των υπολογιστών, γιατί δίνει τη δυνατότητα να αναπαρασταθούν πολύ μικροί και πολλοί μεγάλοι αριθμοί. Ο όρος ”floating-point” αναφέρεται στο γεγονός ότι η υποδιαστολή ενός αριθμού μπορεί να τοποθετηθεί σε οποιαδήποτε θέση του αριθμού. Η μετατόπιση της υποδιαστολής μεταβάλλει και τον εκθέτη που περιλαμβάνεται στον αριθμό.

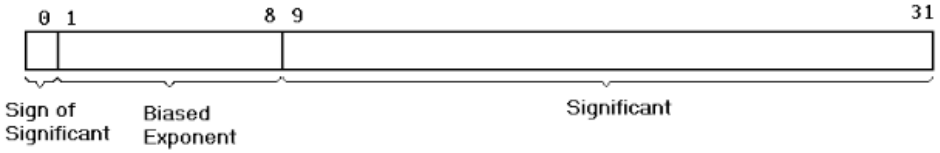
Η σημαντικότερη αναπαράσταση αριθμητικής κινητής υποδιαστολής ορίζεται στην IEEE Standard 754. Το standard αυτό ,έχει υιοθετηθεί και χρησιμοποιείται ευρέως.

Όλοι οι αριθμοί κινητής υποδιαστολής αποτελούνται από τρία μέρη:

1. Το πρόσημο – **sign :** Υποδεικνύει εάν ο αριθμός είναι θετικός (sign = 0) ή αρνητικός (sign = 1).

2. Τον εκθέτη – **exponent** : Αυτή είναι η δύναμη του 2 (δυαδική) με την οποία πολλαπλασιάζεται η mantissa για να ληφθεί η πραγματική τιμή. Η τιμή του εκθέτη αποθηκεύεται στα bit 1 έως 8 . Η παράσταση που χρησιμοποιείται για τον εκθέτη , είναι γνωστή και ως biased αναπαράσταση : μια σταθερή τιμή , ορισμένη ως bias σταθερά, αφαιρείται από το πεδίο τιμών του εκθέτη , για να πάρει ο εκθέτης την πραγματική του τιμή . Σ ’αυτήν την περίπτωση , το πεδίο των 8 bits αποδίδει τους αριθμούς μεταξύ 0 και 255 . Με bias σταθερά ίση με 128 , οι τιμές του εκθέτη βρίσκονται στο διάστημα από –128 έως +127 .

3. Το κλάσμα – **mantissa** : Το «κλασματικό» τμήμα του αριθμού με το πιο αριστερό ψηφίο της mantissa θα πρέπει να είναι το 1. Επομένως δεν χρειάζεται να το αποθηκευτεί στον υπολογιστή . Για αυτό τον λόγο το πεδίο των 23 bit αποθηκεύει μια 24-bit mantissa με μια τιμή μεταξύ του 0,5 και του 1,0 .Τα μεγέθη του εκθέτη και της mantissa μπορεί να διαφέρουν ανάλογα με την ακρίβεια που χρησιμοποιείται.



Σχήμα 1.1 : Format των floating point numbers

2. Normalization Module

Το 1ο module που πρέπει να υλοποιηθεί για τον floating point multiplier είναι το normalization module που είναι υπεύθυνο για την κανονικοποίηση της μάντισσας του αριθμού κινητής υποδιαστολής και τον υπολογισμό των guard και sticky bits. Το guard_bit είναι ένα bit που χρησιμοποιείται στις αποφάσεις στρογγυλοποίησης κατά τη διάρκεια των πράξεων κινητής υποδιαστολής. Το sticky_bit είναι ένα bit που υποδεικνύει αν κάποιο από τα bits χαμηλότερης ακρίβειας είναι ενεργοποιημένα, χρησιμοποιούμενο στις αποφάσεις στρογγυλοποίησης.

Οι είσοδοι του module είναι το αποτέλεσμα πολλαπλασιασμού μήκους 48 bit **mult_result_P** και το άθροισμα εκθετών μήκους 10 bit από το οποίο αφαιρείται το bias. Οι έξοδοι της μονάδας είναι τα **sticky bit** και **guard bit**, η κανονικοποιημένη mantissa μήκους 23 bit **norm_mantissa** και ο κανονικοποιημένος εκθέτης 10 bit **norm_exponent.**

Με ένα always_comb block και με μια unique if/ else συνθήκη ελέγχεται το MSB της mult_mantissa (input) και ανάλογα με την τιμή του υπολογίζονται η norm-mantissa, ο norm_exponent και το guard bit όπως υποδεικνύεται από το figure 6 της εκφώνησης. Στο ίδιο procedural block στην συνέχεια, υπολογίζεται και το sticky bit με έναν for βρόγχο. Η μεταβλητή size_st χρησιμοποιείται για να καθορίσει το εύρος των bits που θα ελεγχθούν, το οποίο είναι 22 αν το MSB είναι 0, ή 23 αν το MSB είναι 1. Ο κώδικας διατρέχει τα bits από 0 μέχρι το size_st και χρησιμοποιεί το bitwise OR για να θέσει το sticky bit αν κάποιο από τα bits εντός αυτού του εύρους είναι 1.

3. Rounding Module

Το 2ο module που απαιτείται να υλοποιηθεί είναι το rounding module, το οποίο είναι υπεύθυνο για τη στρογγυλοποίηση της μάντισσας μετά την κανονικοποίηση με βάση την είσοδο ‘round’.

Το αποτέλεσμα κάθε πράξης μεταξύ των mantissa γενικά αποθηκεύεται σε ένα μεγαλύτερου μήκους καταχωρητή (μνήμης) . Όταν το αποτέλεσμα ‘επιστρέφει’ σε format κινητής υποδιαστολής θα πρέπει να απαλλαχθεί από τα επιπλέον ψηφία . Στην εργασία χρησιμοποιείται η είσοδος round και ανάλογα με την τιμή της υποδεικνύει την μέθοδο rounding. Το πρότυπο IEEE-754 προτείνει κάποιες μεθόδους στρογγυλοποίησης. Οι 3 πρώτες στρογγυλοποιούν στην κοντινότερη τιμή ενώ οι υπόλοιπες τρεις είναι κατευθυνόμενες. Η στρογγυλοποίηση στο κοντινότερο ζυγό (round to nearest even) επικρατεί μεταξύ των μεθόδων και επιλέγεται ως η προκαθορισμένη μέθοδος στην πλειοψηφία των μονάδων κινητής υποδιαστολής (Floating Point Units (FPU)) καθώς οι στρογγυλοποιήσεις που δίνει είναι αρκετά κοντά στο πραγματικό αποτέλεσμα.

Στρογγυλοποιήσεις στον κοντινότερο αριθμό :

- Στρογγυλοποίηση στον κοντινότερο, ισοψηφία στο ζυγό (round to nearest, ties to even ή round to nearest even). Η μέθοδος αυτή στρογγυλοποιεί το αποτέλεσμα της πράξης στον κοντινότερο αριθμό που μπορεί να αναπαρασταθεί. Αν το αποτέλεσμα είναι ακριβώς στη μέση, τότε επιλέγεται ο αριθμός που είναι ζυγός.

- Στρογγυλοποίηση στον κοντινότερο, ισοψηφία στο μακριά από το μηδέν (round to nearest, ties away from zero). Η μέθοδος αυτή στρογγυλοποιεί το αποτέλεσμα της πράξης στον κοντινότερο αριθμό που μπορεί να αναπαρασταθεί. Αν το αποτέλεσμα είναι ακριβώς στη μέση, τότε επιλέγεται ο αριθμός που είναι πιο μακριά από το μηδέν.

- Στρογγυλοποίηση προς τα πάνω όταν είναι κοντά (near_up). Αυτό σημαίνει ότι αν η τελική τιμή είναι κοντά σε μια ακέραια τιμή, τότε θα στρογγυλοποιηθεί προς την επόμενη μεγαλύτερη ακέραια τιμή.

Κατευθυντικές στρογγυλοποιήσεις :

- Προς το μηδέν (IEEE round to zero) – IEEE_zero

- Προς το θετικό άπειρο (IEEE round to positive infinity) – IEEE_pinf

- Προς το αρνητικό άπειρο (IEEE round to negative infinity) – IEEE_ninf

Οι είσοδοι του module είναι : το πρόσημο του αριθμού - **sign** μήκους 1 bit, η είσοδος **round** μήκους 3 bit που αναφέρθηκε προηγουμένως και είναι η παράμετρος που καθορίζει τον τρόπο στρογγυλοποίησης, ο κανονικοποιημένος

εκθέτης - **norm_exponent** μήκους 10 bits, το **guard** και το **sticky** bit που υπολογίζεται στο normalization module και η κανονικοποιημένη mantissa – **norm_mantissa** μήκους 23 bit (χωρίς το leading 1). Από την άλλη, οι έξοδοι θα είναι ο εκθέτης μετα την στρογγυλοποίηση – **post_exponent** μήκους 10 bits, η mantissa μετα το rounding (24 bits)- **post_mantissa** και το **inexact bit** που δείχνει αν το αποτέλεσμα της πράξης είναι ακριβές ή όχι. Σημειώνεται πως όπως και στο προηγούμενο module όλες οι είσοδοι και έξοδοι είναι τύπου logic.

Αρχικά, δημιουργείται ένα enumerate type που χρησιμοποιείται για να αποθηκεύσει την τιμή του τρόπου στρογγυλοποίησης, μετατρέποντας την τιμή round στον τύπο round_input. Η μετατροπή αυτή γίνεται χρησιμοποιώντας casting round_input'(round), που σημαίνει ότι η τιμή του round μετατρέπεται στον enumerated τύπο round_input. Τα labels του round_input ουσιαστικά περιέχουν τον πίνακα figure 6 της εκφώνησης με την παρακάτω αντιστοίχιση :

- IEEE_zero = 3'b001 : Στρογγυλοποίηση προς το μηδέν.
- IEEE_pinf = 3'b010 : Στρογγυλοποίηση προς το συν άπειρο.
- IEEE_ninf = 3'b011 : Στρογγυλοποίηση προς το πλην άπειρο.
- IEEE_near = 3'b100 : Στρογγυλοποίηση προς τον πλησιέστερο ακέραιο.
- away_zero = 3'b101 : Στρογγυλοποίηση μακριά από το μηδέν.
- near_up = 3'b110 : Στρογγυλοποίηση προς τα πάνω.

Με ένα always_comb block υλοποιείται το υπόλοιπο κομμάτι του module. Αρχικά, το inexact_bit υπολογίζεται ως το NOT του OR των sticky_bit και guard_bit. Αυτό δείχνει αν το αποτέλεσμα της στρογγυλοποίησης είναι ακριβές ή όχι. Αν οποιοδήποτε από τα sticky_bit ή guard_bit είναι 1, τότε το αποτέλεσμα δεν είναι ακριβές. Στην συνέχεια, ο υπολογισμός του round_bit γίνεται βάσει της τιμής του round_v και της κατάστασης των sign, guard_bit, και sticky_bit. Χρησιμοποιείται μια δομή unique case με το round_v για να προσδιοριστεί η κατάλληλη τιμή του round_bit. Πιο συγκεκριμένα :

- IEEE_zero: Στρογγυλοποίηση προς το μηδέν, οπότε round_bit = 0.
- IEEE_pinf: Στρογγυλοποίηση προς το συν άπειρο. Αν το πρόσημο (sign) είναι 0 (δηλαδή, ο αριθμός είναι θετικός), τότε round_bit = 1, αλλιώς round_bit = 0
- IEEE_ninf: Στρογγυλοποίηση προς το πλην άπειρο. Αν το πρόσημο (sign) είναι 0, τότε round_bit = 0, αλλιώς round_bit = 1.
- away_zero: Στρογγυλοποίηση μακριά από το μηδέν, οπότε round_bit = 1
- IEEE_near και near_up: Στρογγυλοποίηση προς τον πλησιέστερο ακέραιο και στρογγυλοποίηση προς τα πάνω. Αν το guard_bit είναι 0, τότε round_bit = 0, αλλιώς round_bit = 1.

Η post_round_mantissa υπολογίζεται προσθέτοντας την τιμή του round_bit στη post_mantissa. Στην συνεχεια, άν το bit 24 της post_round_mantissa είναι 1, αυτό σημαίνει ότι η mantissa έχει υπερχειλίσει και απαιτείται επιπλέον

κανονικοποίηση. Σε αυτή την περίπτωση, η mantissa μετατοπίζεται δεξιά κατά μία θέση και ο εκθέτης αυξάνεται κατά 1. Αν δεν υπάρχει υπερχείλιση, mantissa και ο εκθέτης παραμένουν αμετάβλητοι.

4. Exception Handling Module

Το 3ο κατά σειρά module που πρέπει να υλοποιηθεί είναι υπεύθυνο για τον χειρισμό εξαιρέσεων, όπως ο πολλαπλασιασμός δύο μεγάλων αριθμών που οδηγούν σε υπερχείλιση ή ο πολλαπλασιασμός των άπειρων.

Πέρα από τη στρογγυλοποίηση, η αριθμητική κινητής υποδιαστολής επιβάλλει και την ύπαρξη τυποποίησης στο χειρισμό εξαιρέσεων (exception handling), δηλαδή αριθμητικών πράξεων που εμφανίζουν κάποια ιδιαιτερότητα. Το πρότυπο προβλέπει συγκεκριμένες εξαιρέσεις, οι οποίες σηματοδοτούνται στα περισσότερα κυκλώματα κινητής υποδιαστολής από κατάλληλες σημαίες καταστάσεις (status flags). Κάποιες από τις εξαιρέσεις αυτές είναι η διαίρεση με το μηδέν (division by zero), η μη έγκυρη πράξη (invalid operation) που προκύπτει, για παράδειγμα, κατά τον υπολογισμό τετραγωνικής ρίζας αρνητικού αριθμού. ΄Άλλες εξαιρέσεις αποτελούν η υπερχείλιση (overflow) και η υποχείληση (underflow) που προκύπτουν όταν το αποτέλεσμα είναι πολύ μεγάλο, ή πολύ μικρό αντίστοιχα, για να αναπαρασταθεί με ακρίβεια. Στην περίπτωση της υπερχείλισης, ως αποτέλεσμα επιστρέφεται κάποιο άπειρο ή ο μέγιστος, σε απόλυτη τιμή, κανονικοποιημένος αριθμός. Αντίστοιχα, στην περίπτωση της υποχείλισης επιστρέφεται μηδέν, ο ελάχιστος, σε απόλυτη τιμή, κανονικοποιημένος αριθμός, ή κάποιος denormal. ΄Ενα ακόμα status flag που ορίζει το πρότυπο IEEE-754 είναι αυτό του ανακριβούς αποτελέσματος (inexact) που σηματοδοτεί ότι το τελικό αποτέλεσμα διαφέρει από αυτό που προκύπτει από την αριθμητική πράξη άπειρης ακρίβειας.

Οι είσοδοι του module παρουσιαζονται παρακάτω:

- **a, b**: Δύο 32-bit ακέραιοι αριθμοί κινητής υποδιαστολής που θα πολλαπλασιαστούν.
- **z_calc** (32 bit): Το αποτέλεσμα του πολλαπλασιασμού πριν την επεξεργασία των εξαιρέσεων.
- **overflow** (1 bit): Σήμα που δηλώνει υπερχείλιση.
- **underflow** (1 bit): Σήμα που δηλώνει υποχείλιση.
- **inexact_bit** (1 bit): Σήμα που δηλώνει αν το αποτέλεσμα δεν είναι ακριβές.
- **round** (3 bit): Η μέθοδος στρογγυλοποίησης που θα χρησιμοποιηθεί.
- **sign** (1 bit): Το πρόσημο του αποτελέσματος.

Ακολουθουν οι έξοδοι :

- **z** (32 bit): Το τελικό αποτέλεσμα μετά την επεξεργασία των εξαιρέσεων.

- **zero_f, inf_f, nan_f, tiny_f, huge_f, inexact_f** (1 bit): Flags κατάστασης που δηλώνουν αν το αποτέλεσμα είναι μηδέν, άπειρο, NaN, μικρό, μεγάλο ή μη ακριβές αντίστοιχα.

Συναρτηση num_interp

Αρχικά, στο module υλοποιείται η συναρτηση num_interp όπως ζητείται από την εκφώνηση. Η συναρτηση δέχεται ως είσοδο ένα 32-bit σήμα (είσοδοι a, b, z_calc) και χωρίζει την είσοδο σε δυο μέρη: τον εκθέτη και την mantissa. Στην συνεχεια, με if/else δομή:

- Αν ο εκθέτης είναι 0, επιστρέφει ZERO, προσδιορίζοντας ότι ο αριθμός είναι μηδέν.
- Αν ο εκθέτης είναι όλα 1, επιστρέφει INF, προσδιορίζοντας ότι ο αριθμός είναι άπειρος.
- Αν ο εκθέτης είναι 254 (11111110 στο δυαδικό) και η mantissa είναι όλα 1, επιστρέφει MAX_NORM, προσδιορίζοντας ότι ο αριθμός είναι ο μέγιστος κανονικοποιημένος αριθμός.
- Αν ο εκθέτης είναι 1 (00000001 στο δυαδικό) και η mantissa είναι όλα μηδέν, επιστρέφει MIN_NORM, προσδιορίζοντας ότι ο αριθμός είναι ο ελάχιστος κανονικοποιημένος αριθμός.
- Σε όλες τις άλλες περιπτώσεις, επιστρέφει NORM, προσδιορίζοντας ότι ο αριθμός είναι κανονικός (δηλαδή όχι μηδέν, άπειρος, μέγιστος ή ελάχιστος κανονικοποιημένος).

Συναρτηση interp_t

Στην συνεχεια, στο module υλοποιείται η συναρτηση interp_t όπως ζητείται από την εκφώνηση. Η συναρτηση δέχεται ως είσοδο ένα enum interp_t και επιστρέφει ένα 32-bit σήμα που αντιστοιχεί στην τιμή του enum της εισόδου. Στην συνεχεια, με if/else δομή:

- Αν η τιμή είναι INF, επιστρέφει έναν αριθμό με εκθέτη όλα 1 και mantissa όλα 0.
- Αν η τιμή είναι MAX_NORM, επιστρέφει έναν αριθμό με εκθέτη 254 (11111110) και mantissa όλα 1.
- Αν η τιμή είναι MIN_NORM, επιστρέφει τον ελάχιστο κανονικοποιημένο αριθμό, δηλαδή έναν αριθμό με εκθέτη 1 (00000001) και mantissa όλα 0.
- Σε όλες τις άλλες περιπτώσεις, επιστρέφει μηδέν.

Αρχικά, στο module δημιουργείται το enum interp_t με τα labels όπως ζητούνται από την εκφώνηση. Μετά τις συναρτήσεις που αναφέρθηκαν παραπάνω χρησιμοποιείται ένα combinational procedural block. Μέσα στο block

αρχικοποιουνται στο μηδέν τα 6/8 του status σήματος δηλαδή τα zero_f, inf_f, nan_f, tiny_f, huge_f, inexact_f. Έπειτα χρησιμοποιείται η συναρτηση num_interp για να προσδιοριστεί ο τύπος των σημάτων εισόδου. Ανάλογα με τον τύπο τους με μια if/else δομή (με εμφωλευμενες if/else και case εντολές) γίνεται χειρισμός των ακραίων περιπτώσεων. Πιο συγκεκριμένα :

Διαχείριση Ειδικών Περιπτώσεων (Corner Cases):

- **Περίπτωση 1**: Αν a ή b είναι μηδέν **ZERO** και το άλλο είναι άπειρο **INF**, τότε το αποτέλεσμα (z) είναι άπειρο **INF** και τα σήματα inf_f και nan_f γίνονται 1.
- **Περίπτωση 2**: Αν a ή b είναι άπειρο **INF**, τότε το αποτέλεσμα (z) είναι άπειρο **INF** με το πρόσημο να προκύπτει ως εξής: **z[31] = a[31] ^ b[31]**, και το σήμα inf_f γίνεται 1.
- **Περίπτωση 3**: Αν a ή b είναι μηδέν **ZERO**, τότε το αποτέλεσμα (z) είναι μηδέν **ZERO** με το πρόσημο να προκύπτει ως εξής: **z[31] = a[31] ^ b[31]**, και το σήμα zero_f γίνεται 1.
- **Περίπτωση 4**: Αν τόσο a όσο και b είναι κανονικοί **NORMAL: MAX_NORM/MIN_NORM** αριθμοί, τότε έχω τις εξής περιπτώσεις :

  Α. Υπερχείλιση (Overflow): Αν το σήμα overflow είναι 1, τότε το αποτέλεσμα προκύπτει σύμφωνα με την παράμετρο στρογγυλοποίησης (**round**). Χρησιμοποιείται μια case εντολή. Ανάλογα με την τιμή του round, το αποτέλεσμα (z) γίνεται **MAX_NORM** ή **INF** με το σωστό πρόσημο, και τα σήματα inf_f και huge_f γίνονται 1.

  Overflow == 1
  Case → Round
  3'b001: Θέτει το αποτέλεσμα στο MAX_NORM.
  3'b010: Αν το sign είναι 0, θέτει το αποτέλεσμα στο INF και ενεργοποιεί το σήμα inf_f. Αν το sign είναι 1, θέτει το αποτέλεσμα στο MAX_NORM.
  3'b011:Αν το sign είναι 1, θέτει το αποτέλεσμα στο INF και ενεργοποιεί το σήμα inf_f. Αν το sign είναι 0, θέτει το αποτέλεσμα στο MAX_NORM.
  Default: Θέτει το αποτέλεσμα στο INF και ενεργοποιεί το σήμα inf_f.

  Β. Υποχείλιση (Underflow): Ομοίως αν το σήμα underflow είναι 1, τότε το αποτέλεσμα προκύπτει σύμφωνα με την παράμετρο στρογγυλοποίησης (**round**).Ανάλογα με την τιμή του round, το αποτέλεσμα (z) γίνεται **MIN_NORM** ή **ZERO** με το σωστό πρόσημο, και τα σήματα zero_f και tiny_f γίνονται 1.

  Underflow == 1
  Case → Round
  3'b010: Αν το sign είναι 0, θέτει το αποτέλεσμα στο MIN_NORM. Αν το sign είναι 1, θέτει το αποτέλεσμα στο ZERO και ενεργοποιεί το σήμα zero_f.

3'b011: Αν το sign είναι 1, θέτει το αποτέλεσμα στο MIN_NORM. Αν το sign είναι 0, θέτει το αποτέλεσμα στο ZERO και ενεργοποιεί το σήμα zero_f.
3'b101: Θέτει το αποτέλεσμα στο MIN_NORM.
Default: Θέτει το αποτέλεσμα στο ZERO και ενεργοποιεί το σήμα zero_f.

Γ. Κανονική Περίπτωση: Αν δεν υπάρχει ούτε υπερχείλιση ούτε υποχείλιση, το αποτέλεσμα είναι το z_calc, και το σήμα inexact_f να γίνεται 1 αν το αποτέλεσμα δεν είναι ακριβές.

5. Main Module

Το 4ο κατά σειρά module που πρέπει να υλοποιηθεί είναι το main module που απεικονίζεται στα σχήματα της εκφώνησης.

Το βασικό module κάνει instanciate τα module που αναλυθήκαν παραπάνω και έχει τις ακόλουθες εισόδους και εξόδους.

Είσοδοι:

- **a (32 bits)**: Ο 1ος αριθμός κινητής υποδιαστολής που πρόκειται να πολλαπλασιαστεί.
- **b (32 bits)**: Ο 2ος αριθμός κινητής υποδιαστολής που πρόκειται να πολλαπλασιαστεί.
- **round (3 bit)**: Το σήμα εισόδου που καθορίζει τη μέθοδο στρογγυλοποίησης που θα χρησιμοποιηθεί.

Έξοδοι:

- **z (32 bits)**: Το αποτέλεσμα του πολλαπλασιασμού των δύο τιμών κινητής υποδιαστολής.
- **status (8 bits)**: Status flag που δείχνει διάφορες καταστάσεις και εξαιρέσεις κατά τη διάρκεια του πολλαπλασιασμού. Τα bits του είναι τα εξής: **status[0]: zero_f** (Δείχνει αν το αποτέλεσμα είναι μηδέν), **status[1]: inf_f** (Δείχνει αν το αποτέλεσμα είναι άπειρο), **status[2]: nan_f** (Δείχνει αν το αποτέλεσμα είναι NaN), **status[3]: tiny_f** - Δείχνει αν το αποτέλεσμα είναι πολύ μικρό , **status[4]: huge_f** (Δείχνει αν το αποτέλεσμα είναι πολύ μεγάλο), **status[5]: inexact_f** (Δείχνει αν το αποτέλεσμα δεν είναι ακριβές) και status[6] και status[7]: Δεν χρησιμοποιούνται, ορίζονται ως 0.

Αρχικά, γίνεται ο υπολογισμός του προσήμου του αποτελέσματος (sign) που προκύπτει από το XOR των προσημων των εισόδων a και b. Στην συνεχεια, ο υπολογισμός των εκθετών των a και b, οι οποίοι εξάγονται από τα bits [30:23] των αντίστοιχων τιμών. Ο υπολογισμός του αθροίσματος των εκθετών και η αφαίρεση του bias (127) γίνεται για να προκύψει ο τελικός εκθέτης. Ακολουθεί ο υπολογισμός των mantissas των a και b που επεκτείνονται με έναν εμπρόσθιο 1 (leading 1) και πολλαπλασιασμός των mantissas πραγματοποιείται και το αποτέλεσμα αποθηκεύεται στο mult_result_P. Τα στάδια της κανονικοποίησης και της στρογγυλοποίησης πραγματοποιούνται μέσω των nested modules normalize_mult και round_mult. Το module exception_mult χρησιμοποιείται για τη διαχείριση εξαιρέσεων όπως υπερχείλιση, υποχείλιση, NaN, άπειρο, κ.λπ. Τέλος, με ένα combinational procedural block που εξασφαλίζει ότι το αποτέλεσμα του πολλαπλασιασμού (z_calc) υπολογίζεται σωστά, ενώ ταυτόχρονα ρυθμίζονται τα σήματα overflow και underflow με βάση την τιμή του κανονικοποιημένου εκθέτη. Πιο συγκεκριμένα : Αν το bit 9 του κανονικοποιημένου εκθέτη (norm_exponent) είναι 1 ή ο εκθέτης είναι μηδέν, τότε underflow είναι 1 ενώ αν το bit 8 του κανονικοποιημένου εκθέτη είναι 1 ή ο εκθέτης είναι 255, τότε overflow είναι 1.

6. Testbench

Για την υλοποίηση του clock χρησιμοποιήθηκε ένα μπλοκ always για την παραγωγή σήματος ρολογιού με περίοδο 15 ns (χρόνος μετάβασης 7.5 ns).

Μέσα στο testbench υλοποιήθηκε και η συναρτηση conrner_case_generator() με σκοπό να δημιουργεί ανάλογα με την παράμετρο εισόδου corner την ανάλογη corner case. Η είσοδος corner είναι type corner_type δηλαδή ενός enumerated τύπου που δημιουργήθηκε για την αναπαράσταση όλων των ακραίων περιπτώσεων. Επιστρέφει τον κατάλληλο αριθμό. Πιο συγκεκριμένα, τα 12 πιθανά corner cases είναι : positive_zero(=1), negative_zero(=2)positive_inf(=3), negative_inf(=4), positive_snan(=5), negative_snan(=6), positive_qnan(=7), negative_qnan(=8), positive_norm(=9), negative_norm(=10), positive_denorm(=11), negative_denorm(=12).

Γωνιακές Περιπτώσεις

Με μια unique case εντολή με βάση την είσοδο corner η συναρτηση παράγει τον σωστό αριθμό. Σημειώνεται ότι για κάποιες ακραίες περιπτώσεις που είναι διαστήματα αριθμών (όχι μια τιμή συγκεκριμένη όπως πχ. positive Zero) χρησιμοποιείται η $urandom και μια do-while loop.

Η συνάρτηση υποστηρίζει τις ακόλουθες γωνιακές περιπτώσεις:

1. **Θετικό Μηδέν (Positive Zero):**
   corner_sign = 0
   corner_exponent = 00000000
   corner_mantissa = 00000000000000000000000

2. **Αρνητικό Μηδέν (Negative Zero):**
   corner_sign = 1
   corner_exponent = 00000000
   corner_mantissa = 00000000000000000000000

3. **Θετικό Άπειρο (Positive Infinity):**
   corner_sign = 0
   corner_exponent = 11111111
   corner_mantissa = 00000000000000000000000

4. **Αρνητικό Άπειρο (Negative Infinity):**
   corner_sign = 1
   corner_exponent = 11111111
   corner_mantissa = 00000000000000000000000

5. **Θετικό Signaling NaN (Positive Signaling NaN):**
   corner_sign = 0
   corner_exponent = 11111111
   corner_mantissa = 0XXXX...XXXX (τυχαία mantissa όπου το MSB =0)

6. **Αρνητικό Signaling NaN (Negative Signaling NaN):**
   corner_sign = 1
   corner_exponent = 11111111
   corner_mantissa = 0XXXX...XXXX (τυχαία mantissa όπου το MSB =0)

7. **Θετικό Quiet NaN (Positive Quiet NaN):**
   corner_sign = 0
   corner_exponent = 11111111
   corner_mantissa = 1XXXX...XXXX (τυχαία mantissa όπου το MSB =1)

8. **Αρνητικό Quiet NaN (Negative Quiet NaN):**
   corner_sign = 1
   corner_exponent = 11111111
   corner_mantissa = 1XXXX...XXXX (τυχαία mantissa όπου το MSB =1)

9. **Θετικός Denormal Αριθμός (Positive Denormal Number):**
   corner_sign = 0

corner_exponent = 00000000
corner_mantissa = XXXX...XXXX (τυχαία μη μηδενική mantissa)

10. Αρνητικός Denormal Αριθμός (Negative Denormal Number):

corner_sign = 1
corner_exponent = 00000000
corner_mantissa = XXXX...XXXX (τυχαία μη μηδενική mantissa)

11. Θετικός Κανονικός Αριθμός (Positive Normal Number):

corner_sign = 0
corner_exponent = XXXX...XXXX (τυχαίος εκθέτης μεταξύ 00000001 και 11111110)
corner_mantissa = XXXX...XXXX (τυχαία μη μηδενική mantissa)

12. Αρνητικός Κανονικός Αριθμός (Negative Normal Number):

corner_sign = 1
corner_exponent = XXXX...XXXX (τυχαίος εκθέτης μεταξύ 00000001 και 11111110)
corner_mantissa = XXXX...XXXX (τυχαία μη μηδενική mantissa)

Στην συνέχεια, με ένα initial block δημιουργούνται τα 12 corner cases καλώντας την συναρτηση που ζητούνται από την εκφώνηση και γίνεται reset του συστηματος με το σήμα reset. Έπειτα, με ένα always_comb block ελέγχεται εάν τα δύο αποτελέσματα είναι ίδια. Εάν τα αποτελέσματα z και z_check είναι ίσα, η μεταβλητή diaf τίθεται σε 1, δηλώνοντας ότι δεν υπάρχει απόκλιση. Σε διαφορετική περίπτωση, η μεταβλητή diaf τίθεται σε 0 και εμφανίζεται ένα μήνυμα σφάλματος μέσω της συνάρτησης $display.

Τέλος, υλοποιείται ένα always sequential block (always_ff) που εκτελεί τις διάφορες δοκιμές που ζητούνται από την εκφώνηση. Με τη χρήση της συνάρτησης $display, εμφανίζονται οι τιμές των μεταβλητών κατά την κάθε φορά εκτέλεσης, συμπεριλαμβανομένων του ρολογιού (clk), της κατάστασης (status), των μεταβλητών a, b,z και z_check (μαζί με ένα μήνυμα ότι ο πολλαπλασιασμός υπολογίστηκε σωστά ή λάθος) . Για το πρώτο μέρος των δοκιμών εάν η μεταβλητή **part_select** είναι 0, τότε εκτελείται το πρώτο μέρος των δοκιμών, όπου οι μεταβλητές a και b παίρνουν τυχαίες τιμές. Για το δεύτερο μέρος των δοκιμών εάν η **part_select** είναι 1, τότε εκτελείται το δεύτερο μέρος των δοκιμών, όπου οι μεταβλητές a και b παίρνουν τιμές από τα corner cases. Αυτά τα cases διατίθενται μέσω του πίνακα corner_cases. Εάν το σημα reset είναι 1, οι μετρητές counter_a και counter_b τίθενται στο 0 και οι μεταβλητές a και b παίρνουν την τιμή του πρώτου corner case. Σε διαφορετική περίπτωση, οι μετρητές αυξάνονται, και οι μεταβλητές a και b παίρνουν τις

τιμές των corner cases που αντιστοιχούν στις τρέχουσες τιμές των counter_a και counter_b.

7. Assertions module

Immediate Assertions Module

Το module test_status_bits υλοποιείται με σκοπό να ελεγχθούν ποιοι συνδυασμοί των δύο από τα bit κατάστασης δεν πρέπει ποτέ να κάνουν assert ταυτόχρονα. Το module status_bits_combinations_test έχει τρεις εισόδους: την λογική μεταβλητή 8-bit status, το clock, το σημα reset. Μέσα στο module υλοποιείται ένα always_ff block με ασύγχρονο reset. Εάν το σημα rst είναι 0, δηλαδή αν το σύστημα δεν είναι σε κατάσταση επαναφοράς, τότε εκτελείται ο έλεγχος μέσω της δήλωσης assert. Η συνθήκη του assert ελέγχει ότι οι ακόλουθοι συνδυασμοί των bit κατάστασης δεν ισχύουν: Εάν το status[0] ή το status[3] είναι 1 και ταυτόχρονα κάποιο από τα status[1], status[2], ή status[4] είναι 1. Ή εάν το status[5] είναι ενεργό και ταυτόχρονα κάποιο από τα status[0], status[1], status[2], status[3], ή status[4] είναι 1. Αν η συνθήκη αυτή ικανοποιείται, τότε εμφανίζεται το μήνυμα επιτυχίας status test PASS. Αν η συνθήκη αποτύχει, εμφανίζεται το μήνυμα αποτυχίας status test FAIL.

Concurrent Assertions Module

Το module test_status_z_combinations υλοποιεί τα concurrent assertions που ζητούνται από την εκφώνηση για να ελέγξει τη συμπεριφορά και τις καταστάσεις των σημάτων σε έναν κύκλο ρολογιού. Αυτές οι δηλώσεις χρησιμοποιούνται για να διασφαλίσουν ότι οι συνδυασμοί των bits κατάστασης και τα αποτελέσματα τηρούν ορισμένες ιδιότητες σε σχέση με τα σήματα εισόδου.

Το 1ο property (p1) δηλώνει ότι, σε κάθε θετική ακμή του σήματος ρολογιού, αν το status[0] είναι 1, τότε τα bits 30-23 του z πρέπει να είναι όλα μηδενικά. Ο έλεγχος γίνεται με την δήλωση assertP1: assert property (p1), η οποία εμφανίζει "p1 PASS" αν η ιδιότητα ισχύει και "p1 FAIL" αν δεν ισχύει.

Το 2ο property (p2) δηλώνει ότι, σε κάθε θετική ακμή του σήματος ρολογιού, αν το status[1] είναι 1, τότε τα bits 30-23 του z πρέπει να είναι όλα 1, δηλαδή ίσα με 255. Ο έλεγχος γίνεται με την δήλωση assertP2: assert property (p2), η οποία εμφανίζει "p2 PASS" αν η ιδιότητα ισχύει και "p2 FAIL" αν δεν ισχύει.

Το 3ο property (p3) δηλώνει ότι, σε κάθε θετική ακμή του σήματος ρολογιού, αν το status[2] είναι 1, τότε πρέπει να ήταν αλήθεια πριν από δύο κύκλους (χρήση $past) ότι είτε τα bits 30-23 του a ήταν 255 και του b 0, είτε το αντίστροφο. Ο

έλεγχος γίνεται με την δήλωση assertP3: assert property (p3), η οποία εμφανίζει "p3 PASS" αν η ιδιότητα ισχύει και "p3 FAIL" αν δεν ισχύει.

Το 4ο property (p4) δηλώνει ότι, σε κάθε θετική ακμή του σήματος ρολογιού, αν το status[4] είναι 1, τότε τα bits 30-23 του z πρέπει να είναι είτε 255 είτε 254 και τα bits 22-0 του z πρέπει να είναι 8388607. Ο έλεγχος γίνεται με την δήλωση assertP4: assert property (p4), η οποία εμφανίζει "p4 PASS" αν η ιδιότητα ισχύει και "p4 FAIL" αν δεν ισχύει.

Το 5ο property (p5) δηλώνει ότι, σε κάθε θετική ακμή του σήματος ρολογιού, αν το status[3] είναι 1, τότε τα bits 30-23 του z πρέπει να είναι είτε 0 είτε 1 και τα bits 22-0 του z πρέπει να είναι 0. Ο έλεγχος γίνεται με την δήλωση aP5: assert property (p5), η οποία εμφανίζει "p5 PASS" αν η ιδιότητα ισχύει και "p5 FAIL" αν δεν ισχύει.

8. Αποτελέσματα

Παρακατω παρατίθενται ενδεικτικά κάποια αποτελέσματα από σήματα του testbench. Σημειώνεται ότι η εργασία υλοποιήθηκε στο EDA -playground και το εργαλείο που χρησιμοποιήθηκε είναι το siemens Questa. Επίσης, πέρα από τα ενδεικτικά screenshots που θα επισυναφθούν παρακάτω μαζί με τα υπόλοιπα αρχεία θα υπάρχουν και τα 2 qrun.log αρχεία (για το κάθε part αποτελεσμάτων που ελέγχεται μέσω της μεταβλητής **part_select** του testbench) , για περισσότερη πληρότητα (εμφανίζονται όλα τα αποτελέσματα).

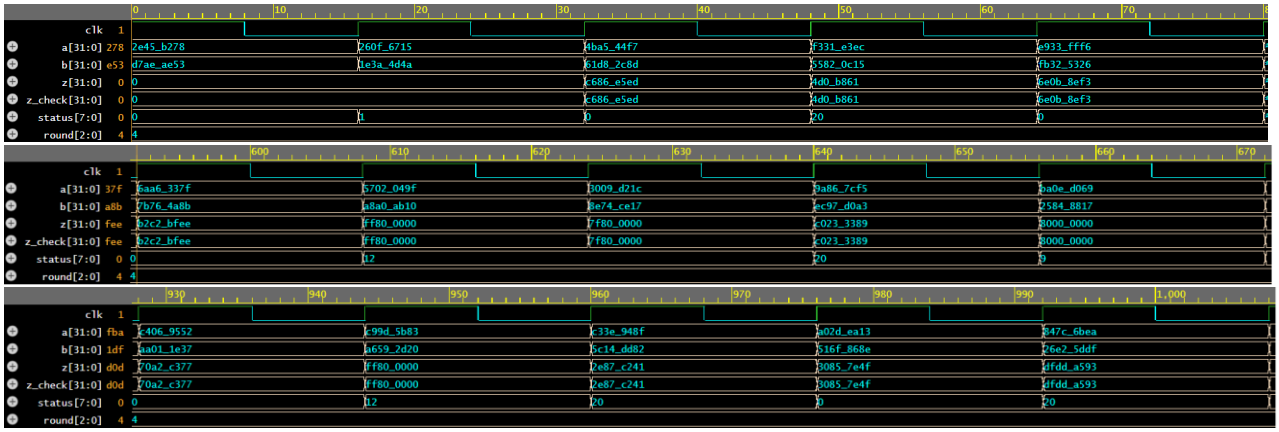
Σημείωση

Για να είναι δυνατό να συγκριθεί η τιμή του z με το αποτέλεσμα που δίνει η συναρτηση της multiplication() της εκφώνησης υλοποιήθηκε το module check_mult() που είναι παρόμοιο με το fp_mult_top που δίνεται. Οι είσοδοι του module είναι οι 2 αριθμοί κινητής υποδιαστολής a, b το σήμα ρολογιού clk, το rst και το round. Η έξοδος είναι το σήμα z_check (logic [31:0]): Το αποτέλεσμα του πολλαπλασιασμού, υπολογισμένο με τη συνάρτηση multiplication. Εσωτερικά, δημιουργείται το string roundString που αναπαριστά τη μέθοδο στρογγυλοποίησης και αρχικοποιείται μέσω ενός initial block. Κατά την αρχικοποίηση, το roundString καθορίζεται με βάση την παράμετρο round: 1: "IEEE_zero" , 2: "IEEE_pinf" ,3: "IEEE_ninf" ,6: "near_up" , 5: "away_zero" , default: "IEEE_near". Μέσα σε ένα always__ff block που ενεργοποιείται με την θετική ακμή του σήματος ρολογιού (posedge clk): Αν το σήμα επαναφοράς (rst) είναι ενεργό (1), τα a_num, b_num, και z_check αρχικοποιουνται σε 0. Αν δεν είναι ενεργό, τα a_num και b_num παίρνουν τις τιμές των εισόδων a και b αντίστοιχα. Το z_check υπολογίζεται με τη συνάρτηση multiplication

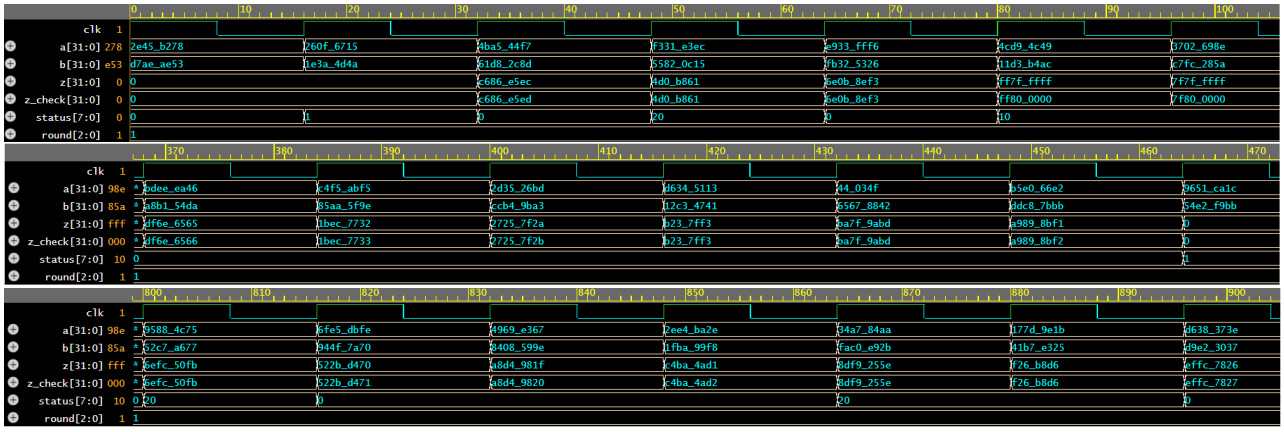
χρησιμοποιώντας το roundString, και τα εσωτερικά σήματα a_num και b_num. Η λειτουργία αυτή διασφαλίζει ότι κάθε φορά που υπάρχει θετική ακμή στο σήμα ρολογιού, εάν το σήμα επαναφοράς δεν είναι ενεργό, οι αριθμοί κινητής υποδιαστολής a και b καταχωρούνται και πολλαπλασιάζονται σύμφωνα με την καθορισμένη μέθοδο στρογγυλοποίησης, αποθηκεύοντας το αποτέλεσμα στο z_check. Το συγκεκριμένο module βρίσκεται στο αρχείο multiplication.sv

Για το 1ο μέρος οπου δημιουργήθηκαν τυχαίοι αριθμοί για τα a,b για τους υπολογισμούς τα αποτελέσματα παρατίθενται παρακάτω.

1ο μέρος/ a,b random values



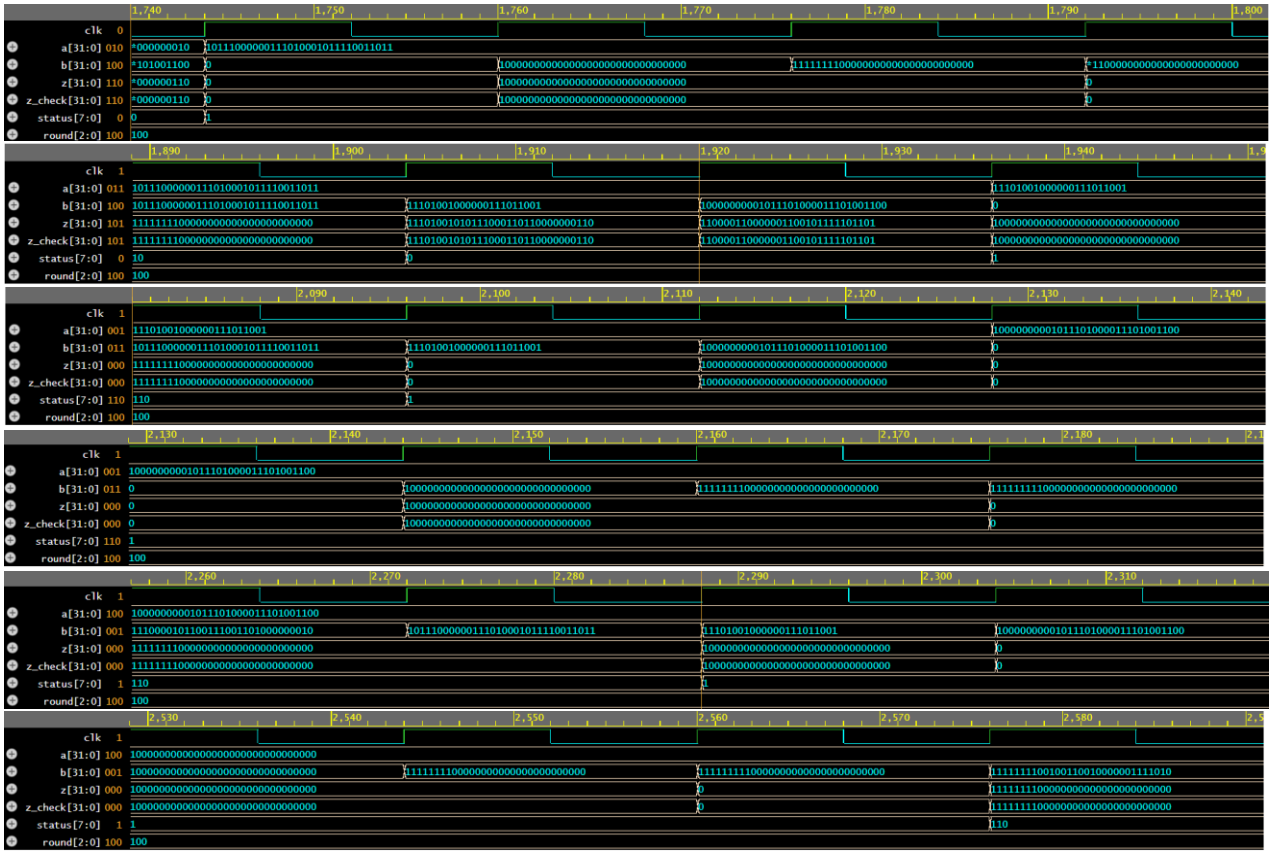
Σχήμα 7.1 : Ενδεικτικά αποτελέσματα προσομοιώσεων για round = "IEEE_near"



Σχήμα 7.2 : Ενδεικτικά αποτελέσματα προσομοιώσεων για round = "IEEE_zero"

2ο μέρος

Για το 2ο μέρος με τα corner cases επιλέχθηκε αντί για δεκαεξαδικο τα αποτελέσματα να είναι στο δυαδικό. Επίσης, για περισσότερη πληρότητα παρατίθενται και κάποια screenshots από το log.



Σχήμα 7.3 : Ενδεικτικά αποτελέσματα προσομοιώσεων για round = "IEEE_near"

```
  0   z result failed
  0  clk=1 status=xxxxxxxx a=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx b=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx z=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx z_check=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  0   z result calculated correct
 16   status test PASS
 16  clk=1 status=00000000 a=00000000000000000000000000000000 b=00000000000000000000000000000000 z=00000000000000000000000000000000 z_check=00000000000000000000000000000000
 32   status test PASS
 32  clk=1 status=00000001 a=00000000000000000000000000000000 b=00000000000000000000000000000000 z=00000000000000000000000000000000 z_check=00000000000000000000000000000000
 32   p1 PASS
 48   status test PASS
 48  clk=1 status=00000001 a=00000000000000000000000000000000 b=10000000000000000000000000000000 z=00000000000000000000000000000000 z_check=00000000000000000000000000000000
 48   p1 PASS
 64   status test PASS
 64  clk=1 status=00000001 a=00000000000000000000000000000000 b=01111111100000000000000000000000 z=00000000000000000000000000000000 z_check=00000000000000000000000000000000
 64   z result calculated correct
 64   p1 PASS
 80   status test PASS
 80  clk=1 status=00000001 a=00000000000000000000000000000000 b=11111111100000000000000000000000 z=10000000000000000000000000000000 z_check=10000000000000000000000000000000
 80   z result calculated correct
 80   p1 PASS
 96   status test PASS
 96  clk=1 status=00000110 a=00000000000000000000000000000000 b=01111111100100110010000001111010 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
 96   p3 PASS
 96   p2 PASS
112   status test PASS
112  clk=1 status=00000110 a=00000000000000000000000000000000 b=11111111101100001101101001111110 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
112   p3 PASS
112   p2 PASS
128   status test PASS
128  clk=1 status=00000110 a=00000000000000000000000000000000 b=01111111110011011110111111011010 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
```

```
1200   status test PASS
1200  clk=1 status=00000110 a=01111111111001101111011111111011010 b=10000000000000000000000000000000 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
1200   p3 PASS
1200   p2 PASS
1216   status test PASS
1216  clk=1 status=00000110 a=01111111111001101111011111111011010 b=01111111100000000000000000000000 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
1216   p3 PASS
1216   p2 PASS
1232   status test PASS
1232  clk=1 status=00000110 a=01111111111001101111011111111011010 b=11111111100000000000000000000000 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
1232   p3 PASS
1232   p2 PASS
1248   status test PASS
1248  clk=1 status=00000010 a=01111111111001101111011111111011010 b=01111111100100110010000001111010 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
1248   z result calculated correct
1248   p2 PASS
1264   status test PASS
1264  clk=1 status=00000010 a=01111111111001101111011111111011010 b=11111111101100001101101001111110 z=11111111100000000000000000000000 z_check=11111111100000000000000000000000
1264   z result calculated correct
1264   p2 PASS
1280   status test PASS
1280  clk=1 status=00000010 a=01111111111001101111011111111011010 b=01111111110011011110111111011010 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
1280   z result calculated correct
1280   p2 PASS
 4720   status test PASS
 4720  clk=1 status=00000110 a=00000000000000000000000000000000 b=11111111101100001101101001111110 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
 4720   p3 PASS
 4720   p2 PASS
 4736   status test PASS
 4736  clk=1 status=00000110 a=00000000000000000000000000000000 b=01111111110011011110111111011010 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
 4736   p3 PASS
 4736   p2 PASS
 4752   status test PASS
 4752  clk=1 status=00000110 a=00000000000000000000000000000000 b=11111111110100011001111001001110 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
 4752   p3 PASS
 4752   p2 PASS
 4768   status test PASS
 4768  clk=1 status=00000110 a=00000000000000000000000000000000 b=01110000101100111001101000000010 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
 4768   p3 PASS
 4768   p2 PASS
 4784   status test PASS
 4784  clk=1 status=00000110 a=00000000000000000000000000000000 b=10111000000111010001011110011011 z=01111111100000000000000000000000 z_check=01111111100000000000000000000000
 4784   z result calculated correct
 4784   p3 PASS
 4784   p2 PASS
 4800   status test PASS
 4800  clk=1 status=00000001 a=00000000000000000000000000000000 b=00000000011101001000000111011001 z=00000000000000000000000000000000 z_check=00000000000000000000000000000000
```

Αποτελέσματα από Log του EDA Playground

Το project υλοποιήθηκε στο περιβάλλον του EDA playground καθώς αντιμετώπισα πρόβλημα και δεν κατάφερα να δουλέψω με το Quartus. Τα log αρχεία και για τα 2 μέρη του testbench για περισσότερη πληρότητα και δυνατότητα σύγκρισης αποτελεσμάτων.

Project

https://www.edaplayground.com/x/qBta