Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Υποχρεωτική εργασία 2024

Γαλάνη Δήμητρα
ΑΕΜ 10331

Μάθημα: Ψηφιακά Συστήματα HW σε Χαμηλά Επίπεδα Λογικής ΙΙ
Διδάσκων: Παυλίδης Βασίλειος
Αναπληρωτής Καθηγητής Α.Π.Θ.

16 Ιουνίου 2024

Περιεχόμενα

1 Normalization module

Αρχικά, υλοποιείται ένα module για την κανονικοποίηση των αριθμών κινητής υποδιαστολής. Σκοπός του module είναι να φέρει τον αριθμό κινητής υποδιαστολής στην μορφή $1.M$, δηλαδή να έχει ο αριθμός ακριβώς ένα μη μηδενικό ψηφίο πριν την υποδιαστολή. Κατά την διαδικασία αυτή πρέπει και ο εκθέτης να μεταβάλλεται ανάλογα, δηλαδή με την μετατόπιση του αριθμού προς τα δεξιά να αυξάνεται κατά ένα.

Το module έχει τις παρακάτω εισόδους:

- **P:** διάνυσμα 48-bit τύπου logic με το αποτέλεσμα του πολλαπλασιασμού.
- **exponent_sum:** διάνυσμα 10-bit τύπου logic με το αποτέλεσμα της πρόσθεση των δύο εκθετών.

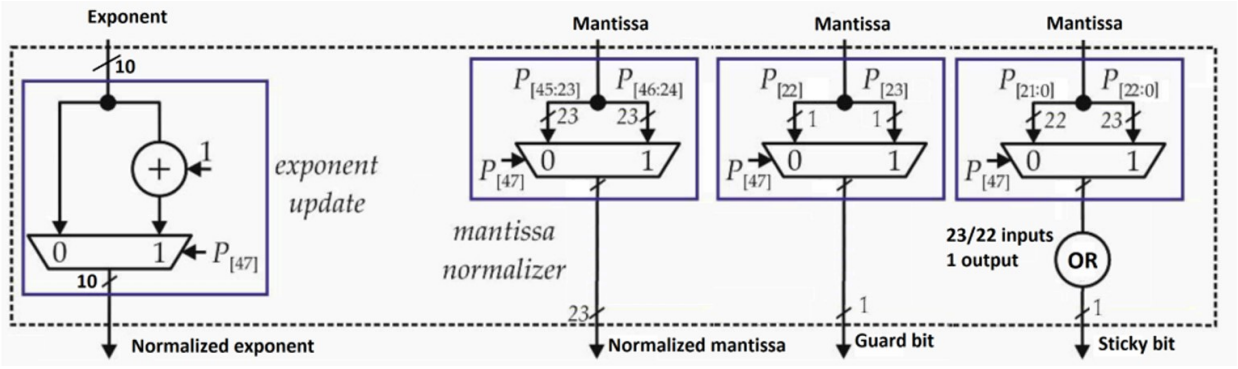
Και εξόδους:

- **guard_bit:** τιμή bit τύπου logic, με το bit που βρίσκεται στην 24η θέση μετά την υποδιαστολή, το πρώτο bit που πρέπει να κοπεί κατά την κανονικοποίηση λόγω περιορισμένου χώρου.
- **sticky_bit:** τιμή bit τύπου logic με το αποτέλεσμα της bitwise-OR πράξης των υπόλοιπων bit μετά το guard bit. Χρησιμοποιείται στο rounding module για να γνωρίζουμε εάν υπάρχουν ή όχι κάποια bit που κόβονται λόγω περιορισμένου χώρου.
- **normalized_mantissa:** διάνυσμα 23-bit τύπου logic με την κανονικοποιημένη mantissa.
- **normalized_exponent:** διάνυσμα 10-bit τύπου logic με τον κανονικοποιημένο εκθέτη.

Θεωρείται ότι οι αριθμοί που πολλαπλασιάστηκαν ήταν στην μορφή $1.M$, οπότε το γινόμενο τους θα είναι είτε στην μορφή $1.M$ είτε στην μορφή $10.M$ είτε στην μορφή $11.M$. Η κανονικοποίηση εξαρτάται από το Most Significant Bit, το 48ο bit, το οποίο θα περιέχει είτε:

- μονάδα, δηλαδή το γινόμενο θα είναι στην μορφή $10.M$ ή $11.M$, και τότε πρέπει να γίνει μετατόπιση της υποδιαστολής προς τα δεξιά κατά ένα bit και αύξηση του εκθέτη κατά μία μονάδα, είτε
- μηδενικό, και τότε δεν χρειάζεται κανονικοποίηση γιατί είναι ήδη κανονικοποιημένο.

Σε κάθε μία από αυτές τις περιπτώσεις ορίζονται τα υπόλοιπα μεγέθη (normalized_mantissa, guard_bit, sticky_source) κατάλληλα με βάση τα bits που τους αναλογούν, ακολουθώντας την εικόνα 6 της εκφώνησης.



Σχήμα 1: Εικόνα που δείχνει τα bit που πρέπει να έχει κάθε μέγεθος ανάλογα με το MSB.

2 Rounding module

Στην συνέχεια, υλοποιείται ένα module το οποίο παίρνει σαν είσοδο, την έξοδο του normalization module. Σκοπός του είναι να στρογγυλοποιήσει την κανονικοποιημένη mantissa με βάση τον κανόνα στρογγυλοποίησης που έχει επιλεχθεί.

Το module έχει τις παρακάτω εισόδους:

- **mantissa_in:** διάνυσμα 23-bit τύπου logic με την κανονικοποιημένη mantissa.
- **guard_bit:** τιμή bit τύπου logic, με το bit που βρίσκεται στην $23^{η}$ θέση μετά την υποδιαστολή.
- **sticky_bit:** τιμή bit τύπου logic με το αποτέλεσμα της bitwise-OR πράξης των υπόλοιπων bit μετά το guard bit.
- **sign:** το πρόσημο του αριθμό, το αποτέλεσμα του λογικού XOR των δύο προσήμων των αριθμών εισόδου. Το πρόσημο υπολογίζεται στο main module.
- **round:** διάνυσμα 3-bit τύπου logic που καθορίζει τον κανόνα στρογγυλοποίησης.

Και εξόδους:

- **result:** διάνυσμα 25-bit τύπου logic με το αποτέλεσμα, το οποίο μπορεί να έχει υποστεί overflow. Το πρώτο bit είναι το overflowed bit. Εάν είναι 1 αυτό σημαίνει ότι θα υπάρχει overflow και αυτός ο άσσος θα είναι το leading one ενώ εάν είναι 0 το leading one είναι το result[23]. Το τι θα γίνει με το overflowed bit αποφασίζεται σε άλλο module.
- **inexact:** σήμα που εκφράζει εάν το αποτέλεσμα που πολλαπλασιασμού είναι ακριβές ή όχι. Για να είναι ακριβές ένα αποτέλεσμα πρέπει ούτε το guard bit ούτε το sticky bit να είναι 1 και σε αυτήν την περίπτωση έχω $inexact = 0$.

Αρχικά έχει οριστεί ένα enumeration για τον τύπο του rounding mode round_t. Στο module αυτό ανάλογα με το round input ελέγχεται αν θα υπάρξει αύξηση στην mantissa κατά 1 ή όχι, εάν χρειάζεται να πάει στην επόμενη τιμή ή όχι. Στην συνέχεια θα αναλυθούν οι κανόνες στρογγυλοποίησης.

2.1 IEEE_near

```
1 IEEE_near:round_increment = (guard_bit && (sticky_bit || mantissa_in[0]));
```

Σε αυτόν τον κανόνα ο αριθμός στρογγυλοποιείται στην πιο κοντινή τιμή που μπορεί να απεικονιστεί ως floating point αριθμός. Εάν απέχει εξίσου από τις δύο πιο κοντινές επιλέγεται ο αριθμός του οποίου το LSB είναι ζυγός αριθμός δηλαδή 0(even significand).

Πιο συγκεκριμένα, σύμφωνα με τον κανόνα πρώτα εξετάζεται το guard bit το οποίο είναι το πιο σημαντικό bit που κόβεται. Και προκύπτουν οι παρακάτω περιπτώσεις:

- Αν το guard bit είναι 0 τότε ο αριθμός βρίσκεται πιο κοντά στον προηγούμενο αριθμό που μπορεί να απεικονιστεί σε floating point μορφή, ανεξάρτητα από το τι είναι το sticky bit, οπότε δεν χρειάζεται αύξηση. Για αυτό το λόγο η λογική έκφραση έχει σαν αποτέλεσμα το λογικό 0. Π.χ. στο $1.x|0x$ οι τιμές των bit είναι: $guard\_bit = 0$, $sticky\_bit = x$, $mantissa[0] = x$.
- Αν το guard bit είναι 1 και το sticky bit είναι και αυτό 1 τότε ο αριθμός με παρόμοια λογική βρίσκεται ακριβώς στην μέση των δύο πιο κοντινών αριθμών που μπορούν να απεικονιστούν σε floating point μορφή, οπότε η mantissa χρειάζεται αύξηση κατά 1. Για αυτό το λόγο η λογική έκφραση έχει σαν αποτέλεσμα το λογικό 1. Π.χ. στο $1.x|11$ οι τιμές των bit είναι: $guard\_bit = 1$, $sticky\_bit = 1$, $mantissa[0] = x$.
- Αν το guard bit είναι 1 και το sticky bit είναι 0 αυτό σημαίνει ότι ο αριθμός βρίσκεται ακριβώς ανάμεσα στους δύο πιο κοντινούς αριθμούς που μπορούν να απεικονιστούν σε floating point μορφή. Σε αυτήν την περίπτωση προκύπτουν δύο σενάρια ανάλογα με το LSB της mantissa. Εάν είναι:

- 0, σημαίνει ότι ο προηγούμενος, του αριθμού που εξετάζεται, είναι ζυγός καθώς τελειώνει σε 0 και ο επόμενος από τον αριθμό που εξετάζεται είναι μονός. Δηλαδή το $1.00|10$ βρίσκεται ανάμεσα στο 1.00 και στο 1.01. Σε αυτήν την περίπτωση ο αριθμός δεν χρειάζεται αύξηση καθώς στρογγυλοποιείται στον προηγούμενο του, διώχνοντας απλώς τα περιττά bit. Για αυτό το λόγο η λογική έκφραση έχει σαν αποτέλεσμα το λογικό 0. Στο παράδειγμα $1.00|10$ οι τιμές των bit είναι: $guard\_bit = 1$, $sticky\_bit = 0$, $mantissa[0] = 0$.
- 1, σημαίνει ότι ο προηγούμενος, του αριθμού που εξετάζεται, είναι μονός καθώς τελειώνει σε 1 και ο επόμενος από τον αριθμό που εξετάζεται είναι ζυγός. Δηλαδή το $1.01|10$ βρίσκεται ανάμεσα στο 1.01 και στο 1.10. Σε αυτήν την περίπτωση ο αριθμός χρειάζεται αύξηση καθώς στρογγυλοποιείται στον επόμενο του, στον αριθμό δηλαδή με το ζυγό LSB, προσθέτοντας μια μονάδα στην mantissa. Για αυτό το λόγο η λογική έκφραση έχει σαν αποτέλεσμα το λογικό 1. Στο παράδειγμα $1.01|10$ οι τιμές των bit είναι: $guard\_bit = 1$, $sticky\_bit = 0$, $mantissa[0] = 1$.

2.2 IEEE_zero

```
IEEE_zero: round_increment = 1'b0;
```

Σε αυτόν τον κανόνα ο αριθμός στρογγυλοποιείται στον αριθμό πιο κοντά στο μηδέν. Αυτό επιτυγχάνεται απλώς αποκόπτοντας τα bit που δεν χωράνε στο διάνυσμα, δηλαδή το guard bit και τα sticky bits. Όταν αποκόπτονται τα bits ο αριθμός στρογγυλοποιείται προς το 0 τόσο για τους θετικούς αριθμούς όσο και για τους αρνητικούς αριθμούς. Για παράδειγμα κατά την στρογγυλοποίηση των παρακάτω αριθμών στο πρώτο bit μετά την υποδιαστολή:

- $1.101 = 1.625$, αποκόπτοντας τα περιττά bit προκύπτει: $1.1 = 1.5$ το οποίο είναι πράγματι πιο κοντά στο μηδέν.
- $-1.101 = -1.625$, αποκόπτοντας τα περιττά bit προκύπτει: $-1.1 = -1.5$ το οποίο είναι πράγματι πιο κοντά στο μηδέν.

2.3 IEEE_pinf

```
IEEE_pinf: round_increment = (sign == 1'b0) && (guard_bit || sticky_bit);
```

Σε αυτόν τον κανόνα ο αριθμός στρογγυλοποιείται στον αριθμό που βρίσκεται πιο κοντά στο $+\infty$. Αυτό επιτυγχάνεται αυξάνοντας πάντα στους θετικούς αριθμούς όταν η τιμή τουλάχιστον ενός από τα guard bit και sticky bit είναι μονάδα. Για τους αρνητικούς αριθμούς δεν χρειάζεται ποτέ αύξηση της mantissas καθώς κάτι τέτοιο θα σήμαινε την αύξηση του αριθμού προς τον αρνητικό άξονα.

2.4 IEEE_ninf

```
IEEE_ninf: round_increment = (sign == 1'b1) && (guard_bit || sticky_bit);
```

Σε αυτόν τον κανόνα ο αριθμός στρογγυλοποιείται στον αριθμό που βρίσκεται πιο κοντά στο $-\infty$. Σε αναλογία με τον πάνω κανόνα στρογγυλοποίησης σε αυτήν την περίπτωση πραγματοποιείται αύξηση πάντα στους αρνητικούς αριθμούς όταν η τιμή τουλάχιστον ενός από τα guard bit και sticky bit είναι μονάδα.

2.5 near_up

```
near_up: round_increment = guard_bit && (sign == 1'b0 || sticky_bit);
```

Ο κανόνας αυτός δεν είναι μέρος των κανόνων στρογγυλοποίησης που έχουν οριστεί από το IEEE. Σύμφωνα με την εκφώνηση της εργασίας σκοπός αυτού του κανόνα είναι να στρογγυλοποιήσει τον αριθμό, με παρόμοιο τρόπο με τον κανόνα IEEE_near, στην πιο κοντινή τιμή που μπορεί να απεικονιστεί ως floating point αριθμός. Η διαφορά τους βρίσκεται στην περίπτωση που ο αριθμός απέχει εξίσου από

τις δύο πιο κοντινές τιμές. Τότε επιλέγεται ο αριθμός που βρίσκεται πιο κοντά στο $+\infty$. Σε αυτόν τον κανόνα κοιτάμε το guard bit.

- Εάν $guard\_bit = 1$ και $sticky\_bit = 1$ σημαίνει ότι ο αριθμός βρίσκεται πάνω από την μέση των δύο πιθανών κοντινοτερων αριθμών σε floating point μορφή. Και άρα η mantissa του αριθμού χρειάζεται να αυξηθεί για να μεταβεί στον κοντινότερο αριθμό.
- Εάν $guard\_bit = 1$ και $sticky\_bit = 0$ σημαίνει ότι ο αριθμός βρίσκεται ακριβώς στην μέση των δύο πιο κοντινών αριθμών που μπορούν να απεικονιστούν σε floating point μορφή. Σε αυτήν την περίπτωση πρέπει να στρογγυλοποιηθει στο $+\infty$, οπότε ελέγχεται το πρόσημο του αριθμού. Εάν το πρόσημο είναι 0 δηλαδή ο αριθμός είναι θετικός τότε πρέπει να γίνει αύξηση. Για τους αρνητικούς αριθμούς δεν χρειάζεται αύξηση της mantissas καθώς κάτι τέτοιο θα σήμαινε την αύξηση του αριθμού προς τον αρνητικό άξονα.

2.6 away_zero

```
1 near_up: round_increment = guard_bit && (sign == 1'b0 || sticky_bit);
```

Ο κανόνας αυτός δεν είναι επίσης μέρος των κανόνων στρογγυλοποίησης που έχουν οριστεί από το IEEE. Σύμφωνα με την εκφώνηση της εργασίας σκοπός αυτού του κανόνα είναι να στρογγυλοποιήσει τον αριθμό, στον αριθμό που βρίσκεται πιο μακριά από το μηδέν. Για να επιτευχθεί αυτό, πραγματοποιείται αύξηση εάν υπάρχει είτε guard bit είτε sticky bit. Οπότε αυξάνεται κάθε φορά προς τον θετικό άξονα όταν ο αριθμός είναι θετικός ή προς τον αρνητικό άξονα όταν ο αριθμός είναι αρνητικός.

3 Exception Handling module

Το module αυτό έχει σκοπό την διαχείριση των οριακών περιπτώσεων που προκύπτουν μετά την στρογγυλοποίηση. Όπως για παράδειγμα τις περιπτώσεις που υπάρχει overflow ή ο πολλαπλασιασμός δύο αριθμών οδηγεί στο άπειρο.

Το module έχει τις παρακάτω εισόδους:

- **a:** διάνυσμα 32-bit τύπου logic με τον πρώτο παράγοντα.
- **b:** διάνυσμα 32-bit τύπου logic με τον δεύτερο παράγοντα.
- **z_calc:** διάνυσμα 32-bit τύπου logic με το κανονικοποιημένο και στρογγυλοποιημένο αποτέλεσμα του πολλαπλασιασμού όπως έχει γίνει στο fp_mult module.
- **overflow:** τιμή bit τύπου logic, με ένα bit που καθορίζει εάν ο αριθμός έχει υποστεί overflow.
- **underflow:** τιμή bit τύπου logic, με ένα bit που καθορίζει εάν ο αριθμός έχει υποστεί underflow.
- **inexact:** σήμα που εκφράζει εάν το αποτέλεσμα που πολλαπλασιασμού είναι ακριβές ή όχι, όπως έχει καθοριστεί στο rounding module.
- **round:** διάνυσμα 3-bit τύπου logic που καθορίζει τον κανόνα στρογγυλοποίησης.

Και εξόδους:

- **z:** z, διάνυσμα 32-bit τύπου logic με το αποτέλεσμα του πολλαπλασιασμού εφόσον έχει ελεγχθεί εάν ανήκει σε κάποια ακραία περίπτωση και έχει καθοριστεί ανάλογα.
- **zero_f:** σήμα που λειτουργεί σαν flag το οποίο σηκώνεται όταν ο αριθμός εμπίπτει στην μηδενική περίπτωση.
- **inf_f:** σήμα που λειτουργεί σαν flag το οποίο σηκώνεται όταν ο αριθμός εμπίπτει στην περίπτωση του απείρου.
- **nan_f:** σήμα που λειτουργεί σαν flag το οποίο σηκώνεται όταν ο αριθμός εμπίπτει στην περίπτωση που δεν μπορεί να οριστεί.
- **tiny_f:** σήμα που λειτουργεί σαν flag το οποίο σηκώνεται όταν υπάρχει underflow.
- **huge_f:** σήμα που λειτουργεί σαν flag το οποίο σηκώνεται όταν υπάρχει overflow.
- **inexact_f:** σήμα που λειτουργεί σαν flag το οποίο σηκώνεται όταν το αποτέλεσμα δεν είναι ακριβές.

Αρχικά έχει οριστεί ένα enumeration για τον τύπο του rounding mode round_t και ένα enumeration για την κατηγορία που ανήκει ο αριθμός interp_p. Σε αυτό το module υλοποιούνται στην συνέχεια δύο συναρτήσεις.

3.1 Συνάρτηση num_interp

Σκοπός της συνάρτησης αυτής είναι να επιστρέψει την κατηγορία που ανήκει ο αριθμός, δηλαδή μία μεταβλητή τύπου interp_t. Οι τρεις κατηγορίες που μπορεί να ενταχθεί ο αριθμός σύμφωνα με τον πίνακα 1 της εκφώνησης είναι οι εξής:

- **ZERO:** όταν ο αριθμός είναι μηδενικός ή denormal, δηλαδή όταν ο εκθέτης είναι ίσος με 0.
- **INF:** όταν ο αριθός είναι άπειρος ή απροσδιόριστος (NaN), δηλαδή όταν ο εκθέτης είναι ίσος με $(11111111)_2$ στο δυαδικό σύστημα.
- **NORM:** στις υπόλοιπες περιπτώσεις.

3.2 Συνάρτηση z_num

Η συνάρτηση αυτή κάνει την αντίστροφη δουλειά από την προηγούμενη συνάρτηση. Παίρνει σαν όρισμα μια κατηγορία αριθμού και επιστρέφει ένα διάνυσμα 31-bit με την τιμή(εκθέτης και mantissa) που αντιστοιχεί σε αυτήν την κατηγορία. Παρακάτω παρουσιάζονται οι κατηγορίες και οι τιμές που τους αντιστοιχούν σε δυαδική μορφή.

```
1 ZERO:      z_num = 31'b0;
2 INF:       z_num = 31'b11111111000000000000000000000000;
3 MIN_NORM:  z_num = 31'b00000001000000000000000000000000;
4 MAX_NORM:  z_num = 31'b11111110111111111111111111111111;
```

Οι τιμές αυτές βασίζονται στον πίνακα 2 της εκφώνησης της εργασίας.

3.3 Συνδιασμοί οριακών περιπτώσεων

Στην συνέχεια διαχειρίζονται κατάλληλα οι οριακές περιπτώσεις, σύμφωνα με τα πρότυπα που έχουν οριστεί από το IEEE. Οι οριακές περιπτώσεις χωρίζονται σε κάποιους συνδυασμούς, ανάλογα με τις κατηγορίες που ανήκουν τα a και b.

3.3.1 a: ZERO και b: ZERO | a: ZERO και b: NORM | a: NORM και b: ZERO

Όταν το a και το b είναι μηδενικά ή όταν ένα από τα a και b είναι ZERO και το άλλο είναι NORM, το z παίρνει την τιμή ± 0, ανάλογα με το πρόσημο του αποτελέσματος του πολλαπλασιασμού z_calc. Σε αυτήν την περίπτωση το σήμα zero_f ενεργοποιείται.

3.3.2 a: ZERO και b: INF | a: INF και b: ZERO

Όταν ένα από τα a και b είναι INF και το άλλο είναι ZERO, το z παίρνει την τιμή $+\infty$. Σε αυτήν την περίπτωση το σήμα inf_f και το σήμα nan_f ενεργοποιείται. Αυτή είναι η μόνη περίπτωση που το nan_f μπορεί να ενεργοποιηθεί.

3.3.3 a: NORM και b: INF | a: INF και b: NORM | a: INF και b: INF

Όταν ένα από τα a και b είναι INF και το άλλο είναι NORM ή όταν και τα δύο είναι INF, το z παίρνει την τιμή $\pm\infty$, ανάλογα με το πρόσημο του αποτελέσματος του πολλαπλασιασμού z_calc. Σε αυτήν την περίπτωση το σήμα inf_f ενεργοποιείται.

3.3.4 a: NORM και b: NORM

Όταν το a και το b είναι τύπου NORM, τότε πρέπει να ελεγχθεί η ύπαρξη overflow ή underflow. Στην περίπτωση που υπάρχει overflow, ανάλογα με το κριτήριο στρογγυλοποίησης το z παίρνει την κατάλληλη μορφή. Δηλαδή:

- **IEEE_near:** ο z πάει στο $\pm\infty$ανάλογα με το πρόσημο του z_calc ο οποίος είναι το γινόμενο των δύο αριθμών όπως έχει οριστεί στην fp_mult.
- **IEEE_zero:** ο z πάει στο ±MAX_NORM ανάλογα με το πρόσημο του z_calc ο οποίος είναι το γινόμενο των δύο αριθμών όπως έχει οριστεί στην fp_mult.
- **IEEE_pinf:** ο z πάει στο $+\infty$όταν το πρόσημο του z_calc είναι θετικό και στο - MAX_NORM όταν το πρόσημο του z_calc είναι αρνητικό.
- **IEEE_ninf:** ο z πάει στο +MAX_NORM όταν το πρόσημο του z_calc είναι θετικό και στο - ∞όταν το πρόσημο του z_calc είναι αρνητικό.
- **near_up:** ο z πάει στο $\pm\infty$ανάλογα με το πρόσημο του z_calc.
- **away_zero:** ο z πάει στο $\pm\infty$ανάλογα με το πρόσημο του z_calc.

Στην περίπτωση του overflow το σήμα inexact_f ενεργοποιείται, καθώς και το σήμα huge_f. Το inexact καθώς χάνεται πληροφορία και το huge καθώς ο αριθμός που θα παραχθεί θα είναι πολύ μεγάλος. Επίσης στις περιπτώσεις που ο z πάει στο ∞σηκώνεται επίσης το σήμα inf_f. Με παρόμοια λογική ορίζονται και οι συνθήκες στην περίπτωση που έχουμε underflow:

- **IEEE_near:** ο z πάει στο ± 0 ανάλογα με το πρόσημο του z_calc ο οποίος είναι το γινόμενο των δύο αριθμών όπως έχει οριστεί στην fp_mult.
- **IEEE_zero:** ο z πάει στο ± 0 ανάλογα με το πρόσημο του z_calc ο οποίος είναι το γινόμενο των δύο αριθμών όπως έχει οριστεί στην fp_mult.
- **IEEE_pinf:** ο z πάει στο +MIN_NORM όταν το πρόσημο του z_calc είναι θετικό και στο -0 όταν το πρόσημο του z_calc είναι αρνητικό.
- **IEEE_ninf:** ο z πάει στο +0 όταν το πρόσημο του z_calc είναι θετικό και στο -MIN_NORM όταν το πρόσημο του z_calc είναι αρνητικό.
- **near_up:** ο z πάει στο ± 0 ανάλογα με το πρόσημο του z_calc.
- **away_zero:** ο z πάει στο $\pm$MIN_NORM ανάλογα με το πρόσημο του z_calc.

Στην περίπτωση του underflow το σήμα inexact_f ενεργοποιείται, καθώς και το σήμα tiny_f. Το inexact καθώς χάνεται πληροφορία και το tiny καθώς ο αριθμός που θα παραχθεί θα είναι πολύ μικρός. Επίσης στις περιπτώσεις που ο z πάει στο 0 σηκώνεται επίσης το σήμα zero_f.

Στην περίπτωση που δεν υπάρχει ούτε overflow ούτε underflow ο z παίρνει την τιμή 32-bit του z_calc και το σήμα inexact_f παίρνει την τιμή εισόδου inexact.

4 Main module

Σε αυτό το module που το ονομάζουμε fp_mult, συνδυάζονται τα υπόλοιπα modules και γίνονται οι πράξεις που περιγράφονται στην εκφώνησης της εργασίας. Οι είσοδοι του module είναι οι εξής:

- **a:** διάνυσμα 32-bit τύπου logic με τον πρώτο παράγοντα.
- **b:** διάνυσμα 32-bit τύπου logic με τον δεύτερο παράγοντα.
- **rnd:** διάνυσμα 3-bit τύπου logic που καθορίζει τον κανόνα στρογγυλοποίησης.

Και οι έξοδοι:

- **z:** το τελικό διάνυσμα z 32-bit μετά από το normalization, το rounding και το exception handling module.
- **status:** διάνυσμα 8-bit με τα status flags.

Στο module αυτό υπολογίζονται τα εξής πράγματα:

1. το πρόσημο του z, το οποίο είναι το αποτέλεσμα της λογικής πράξης XOR των προσήμων του a και b.
2. το e, δηλαδή το άθροισμα των εκθετών των δύο παραγόντων μείον το bias, το οποίο στην περίπτωση της ακρίβειας 32-bit float είναι 127.
3. την mantissa του a και του b μαζί με το leading one, η οποία προκύπτει από την συνένωση του leading one με τα 23 τελευταία bit του αριθμού και άρα το τελικό αποτέλεσμα είναι 24-bit με το MSB να είναι το leading one.
4. το prod, το γινόμενο των mant_a και mant_b το οποίο είναι ένα διάνυσμα 48-bit.
5. το addition, ένα σήμα που δείχνει εάν πρέπει να γίνει αύξηση του εκθέτη μετά το rounding με βάση το MSB του mant_round, δηλαδή εάν το MSB είναι 1 αυτό σημαίνει ότι έχουμε overflow στην mantissa και πρέπει να κάνουμε ένα shift right στην mantissa και να υπολογιστεί ο νέος εκθέτης, temp, ο οποίος αυξάνεται κατά 1.
6. το post_rounding_mantissa, που είναι το 24-bit διάνυσμα μετά το shift του εάν έχει γίνει overflow.
7. το overflow, το οποίο είναι το αποτέλεσμα της προσημασμένης σύγκρισης του εκθέτη μετά το πιθανό addition με το 254. Η προσημασμένη σύγκριση είναι απαραίτητη καθώς ο εκθέτης μπορεί να είναι και αρνητικός.
8. το underflow, το οποίο με παρόμοια λογική είναι το αποτέλεσμα της προσημασμένης σύγκρισης του εκθέτη μετά το πιθανό addition με το 1.
9. το status, που είναι η συνένωση των flags της εξόδου του exception handling module.

Επίσης σε αυτό το module γίνονται οι συνδέσεις με τα υπόλοιπα modules. Τα port που συνδέονται όπως έχουν αναλυθεί σε κάθε module. Αξίζει να σημειωθεί ότι:

- το exponent_sum του normalize_mult συνδέεται με το άθροισμα των εκθετών μείον το bias.
- το mantissa_in του round_mult συνδέεται με το concatenation: {1'b1, mant_norm}, δηλαδή με την συνένωση του leading one με την mantissa της εξόδου του normalization module .
- το z_calc του exception_handling συνδέεται με το concatenation:

```
1 .z_calc({sign_z, temp[7:0], post_rounding_mantissa[22:0]})
```

5 Testbench

Παρόλο που στην εκφώνηση απαιτείται ένα testbench που να κάνει και στο εσωτερικό του και τα δύο μέρη, έχουν δημιουργηθεί δύο testbenches αντί για ένα, για λόγους ευκρίνειας για να είναι πιο διακριτά τα αποτελέσματα και στις δύο περιπτώσεις από τον αναγνώστη. Το κάθε testbench εξετάζει το κάθε μέρος που περιγράφεται στην εκφώνηση της εργασίας.

5.1 Τυχαίες τιμές a και b

Στο πρώτο κομμάτι της εκφώνησης ζητείται να ελεγχθεί το αποτέλεσμα του πολλαπλασιαστή που έχει υλοποιηθεί για τυχαίες τιμές. Στο module fp_mult_tb εξετάζεται ακριβώς αυτό. Αρχικά δημιουργείται ένα στιγμιότυπο του module που μας δίνεται για το top level, με το οποίο γίνονται οι κατάλληλες συνδέσεις των ports. Όσον αφορά το ρολόι, ζητείται από την εκφώνηση να έχει περίοδο 15ns, οπότε κάθε 7.5ns ορίζεται η αλλαγή της τιμής του ρολογιού. Στο testbench αυτό έχει δημιουργηθεί ένας πίνακας από strings με τα ονόματα των κανόνων στρογγυλοποίησης, προκειμένου να αλλάζει το rounding στην συνάρτηση multiplication ανάλογα με το j. Με αυτόν τον τρόπο αντιστοιχίζονται οι τιμές που παίρνει το rnd, το οποίο αποτελεί μία απο της εισόδους που δίνουμε στο fp_mult_top, που είναι αριθμοί 3-bit από το 0 μέχρι το 5, με τα string "IEEE_near", "IEEE_zero" κλπ, τα οποία αποτελούν εισόδους του wrapper. Δηλαδή το j, το οποίο κυμαίνεται από το 0 ως το 5, οριοθετεί το κριτήριο που επιλέγεται τόσο σαν αριθμός όσο και σαν string:

```
rnd = j[2:0]; // j is the number of the rounding
// Considering the rounding mode j, chooses the correct string
z_real <= multiplication(rounding_modes_str[j], a, b);
```

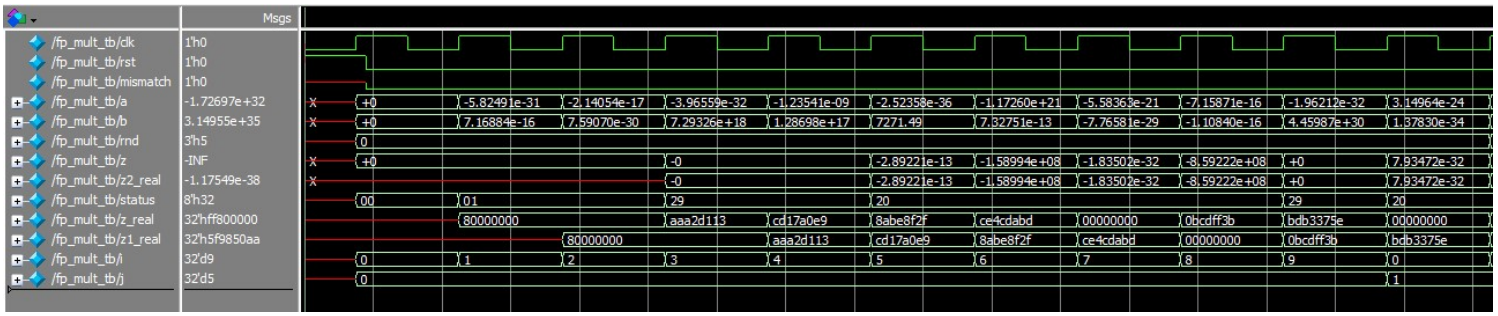
Για κάθε rounding επιλέγεται να ελεγχθούν 10 τυχαία παραδείγματα. Η εναλλαγή των παραδειγμάτων πραγματοποιείται χάρη στην μεταβλητή i. Είναι σαν να έχουμε υλοποιήσει ένα διπλό for με τα j και i.

```
  // Update indices
  if (i < 9) begin // 10 examples of each rounding mode
      i <= i + 1;
  end else if (j < 5) begin // Change rounding mode 6 rounding modes
      i <= 0;
      j <= j + 1;
```

Με την συνάρτηση $urandom δημιουργούνται τυχαίες τιμές 32-bit για το a και b. Και το αποτέλεσμα του πολλαπλασιασμού συγκρίνεται με το αποτέλεσμα της συνάρτησης multiplication που μας δίνεται. Το αποτέλεσμα z το οποίο υπολογίζεται με βάση τα modules που έχουν υλοποιηθεί, υπολογίζεται μετά από δύο κύκλους ρολογιού από την στιγμή που έχουν επιλεγεί τα a και b. Για τον λόγο αυτό προκειμένου να συγκρίνουμε το αποτέλεσμα με το αποτέλεσμα του wrapper κρατάμε το z_real του wrapper για δύο κύκλους χρησιμοποιώντας δύο flip flop, το z1_real και το z2_real, και αναθέτοντας την τιμή του z_real από το ένα στο άλλο με non blocking ανάθεση. Οπότε τελικά η σύγκριση μπορεί να γίνει στο ίδιο ρολόι καθώς η τιμή του z και η τιμή του z2_real ταυτίζονται στο ίδιο ρολόι.

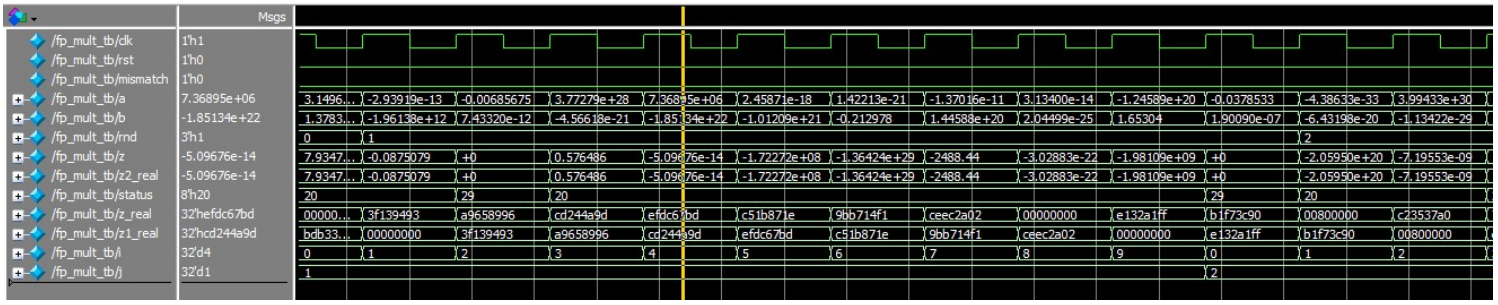
```
  z_real <= multiplication(rounding_modes_str[j], a, b);
  z1_real <= z_real;
  z2_real <= z1_real;
```

Για να γίνει ο έλεγχος για όλα τα συνεχόμενα ρολόγια ορίζεται ένα σήμα mismatch, το οποίο σηκώνεται όταν το z είναι διαφορετικό από το z2_real. Στις κυματομορφές που παρέχονται παρακάτω παρατηρείται ότι αυτό το σήμα δεν σηκώνεται ποτέ, το οποίο σημαίνει ότι ο πολλαπλασιαστής που υλοποιήθηκε βγάζει σε κάθε τυχαία περίπτωση το ίδιο αποτέλεσμα με τον wrapper. Οπότε η λειτουργία του πολλαπλασιαστή είναι ορθή. Απαιτείται προσοχή κατά την ανάγνωση των κυματομορφών καθώς τα a και b δεν αντιστοιχούν με το z της ίδιας στήλης αλλά με το z που βρίσκει δύο ρολόγια μετά από αυτά, δηλαδή τα a και b δεν είναι ευθυγραμμισμένα με το z.

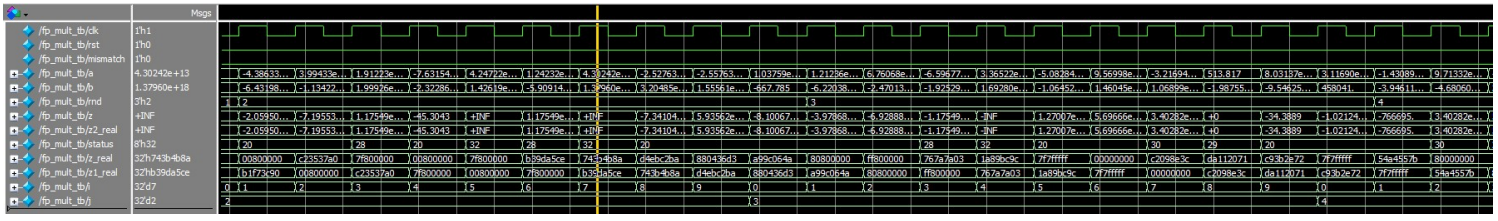


Σχήμα 2: Το πρώτο απόκομμα της κυματομορφής που παράγεται. Το $rnd = 0$ και μπορούμε να μετρήσουμε ότι υπάρχουν 10 τυχαία παραδείγματα μετά την αρχικοποίηση στο 0.

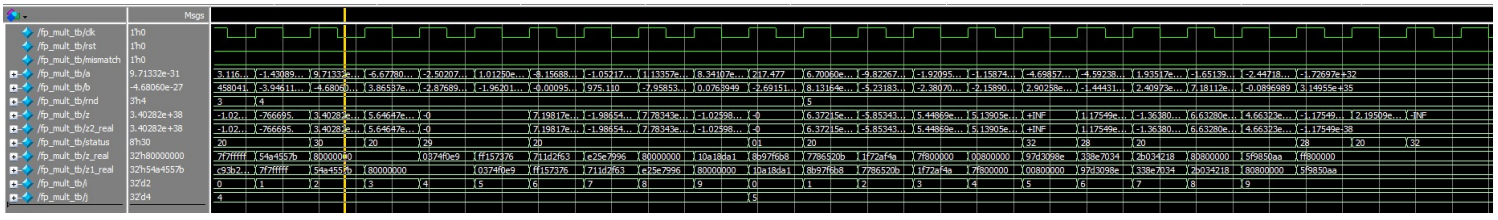
Στο σχήμα φαίνεται κιόλας ότι το z2_real και το z παίρνουν την ίδια τιμή μετά από δύο κύκλους ρολογιού από την στιγμή που επιλέγονται τα πρώτα a και b. Όπως και το ότι δεν υπάρχει κανένα mismatch. Παρακάτω, παρέχονται δύο ακόμα φωτογραφίες με την συνέχεια της κυματομορφής. Δηλαδή τα 10 τυχαία παραδείγματα για τα επόμενα rnd.



Σχήμα 3: Το δεύτερο απόκομμα της κυματομορφής που παράγεται. Φαίνονται τα παραδείγματα για $rnd = 1$.



Σχήμα 4: Το τρίτο απόκομμα της κυματομορφής που παράγεται. Φαίνονται τα παραδείγματα για $rnd = 2$ και $rnd = 3$.



Σχήμα 5: Το τέταρτο απόκομμα της κυματομορφής που παράγεται. Φαίνονται τα παραδείγματα για $rnd = 4$ και $rnd = 5$. Στο τέλος έχουμε αφήσει τουλάχιστον δύο ρολόγια για να ελέγξουμε και το τελευταίο αποτέλεσμα z, χωρίς να αναθέτουμε νέα a και β, καθώς έχουν δοθεί ήδη 10 παραδείγματα.

5.2 Οριακές τιμές a και b

Στο δεύτερο κομμάτι της εκφώνησης ζητείται να ελεγχθεί το αποτέλεσμα του πολλαπλασιαστή που έχει υλοποιηθεί, για όλες τις οριακές τιμές. Στο δεύτερο testbench που υλοποιήθηκε στο module fp_mult_tb_corner εξετάζεται αυτό. Η λογική που ακολουθείται για την υλοποίηση του testbench είναι παρόμοια με την λογική του παραπάνω testbench όποτε δεν θα αναλυθούν οι ίδιες λεπτομέρειες.

Αρχικά ορίζεται ένας τύπος δεδομένων corner_case_t, ο οποίος ορίζεται με enumeration για τις 12 οριακές περιπτώσεις που αναφέρονται στην εκφώνηση. Στην συνέχεια υλοποιείται μια συνάρτηση, η οποία παίρνει σαν είσοδο μια μεταβλητή τύπου corner_case_t και επιστρέφει τον αριθμό σε 32-bit που αντιστοιχεί σε αυτήν την οριακή περίπτωση. Οι τιμές αυτές έχουν οριστεί με την βοήθεια του πίνακα 1 και του πίνακα 2 της εκφώνησης. Η διαφορά του signaling με το quiet NaN είναι ότι το πρώτο έχει στην mantissa τουλάχιστον μία μονάδα εκτός από το MSB, ενώ το MSB της mantissa του quiet NaN είναι 1 και τα υπόλοιπα bit είναι 0.

Στην συνέχεια οι τιμές που επιστρέφει η συνάρτηση, για κάθε ένα από τα corner cases, ανατίθενται σε έναν unpacked array, που ονομάζεται corner_values, ο οποίος αποτελείται από 12 στοιχεία, όπου κάθε στοιχείο είναι ένα packed 32-bit διάνυσμα.

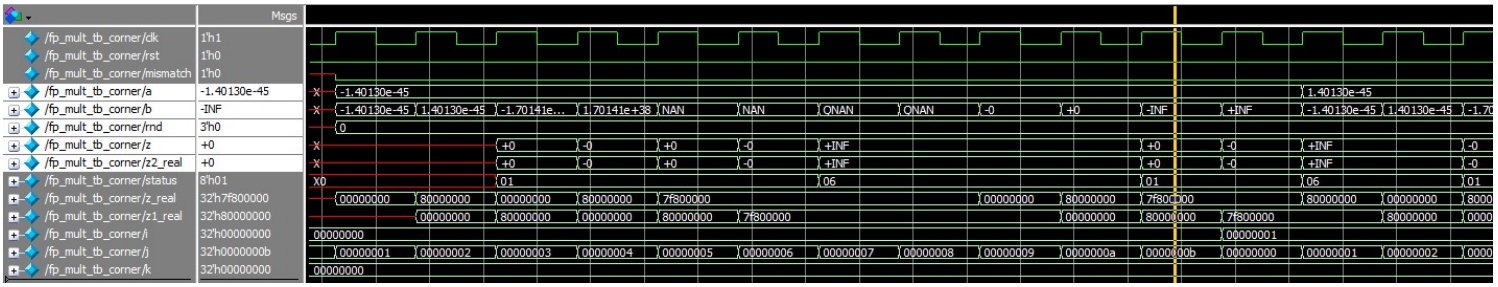
Σε αυτό το testbench αντί για τυχαίες τιμές στο a και στο b ανατίθενται οι οριακές τιμές:

```
// Set a and b values based on current indices
a = corner_values[i];
b = corner_values[j];
```

Σε αυτήν την περίπτωση επειδή θέλουμε να ελέγξουμε και τις 144 περιπτώσεις που προκύπτουν με των συνδυασμό των 12 οριακών περιπτώσεων μεταξύ τους και για τα 6 roundings δημιουργείται ένα τριπλό "for loop" μέσα από τις συνθήκες if, με τα i,j να κυμαίνονται από το 0 ως και το 11 και το k να κυμαίνεται από το 0 ως και το 5. rounding,

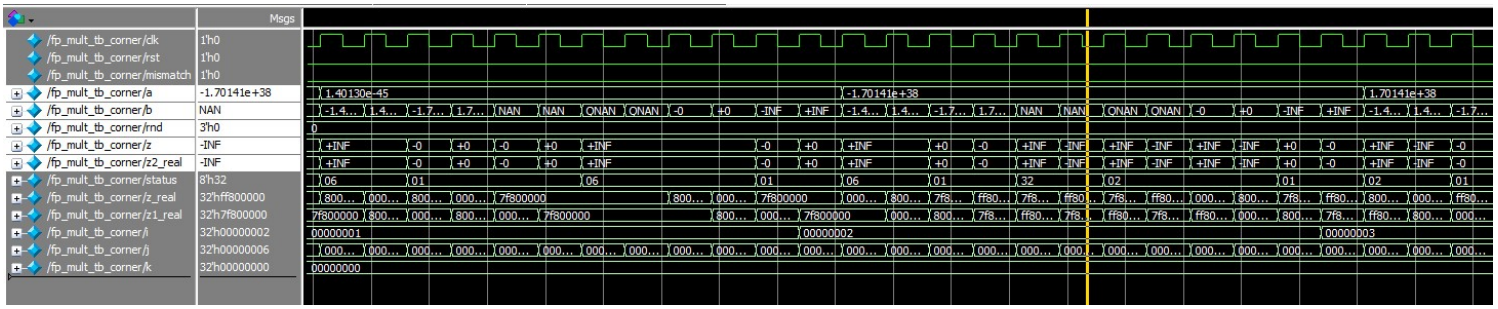
```
      // Update indices
      if (j < 11) begin
        j <= j + 1;
      end else if (i < 11) begin
        j <= 0;
        i <= i + 1;
      end else if (k < 5) begin
        j <= 0;
        i <= 0;
        k <= k + 1;
      end else begin
        #60;
        $finish;
      end
```

Ο έλεγχος του αποτελέσματος z γίνεται με τον ίδιο τρόπο όπως το πρώτο testbench. Παρακάτω παρέχεται η κυματομορφή που προκύπτει από αυτό το testbench, η οποία είναι χωρισμένη σε κομμάτια για καλύτερη ευκρίνεια. Οι εικόνες αυτές παρόλο που είναι χωρισμένες, είναι ενδεικτικές για την επίδειξη της ορθής λειτουργίας του πολλαπλασιαστή και δεν διακρίνονται με ακρίβεια τα αποτελέσματα του κάθε παραδείγματος.

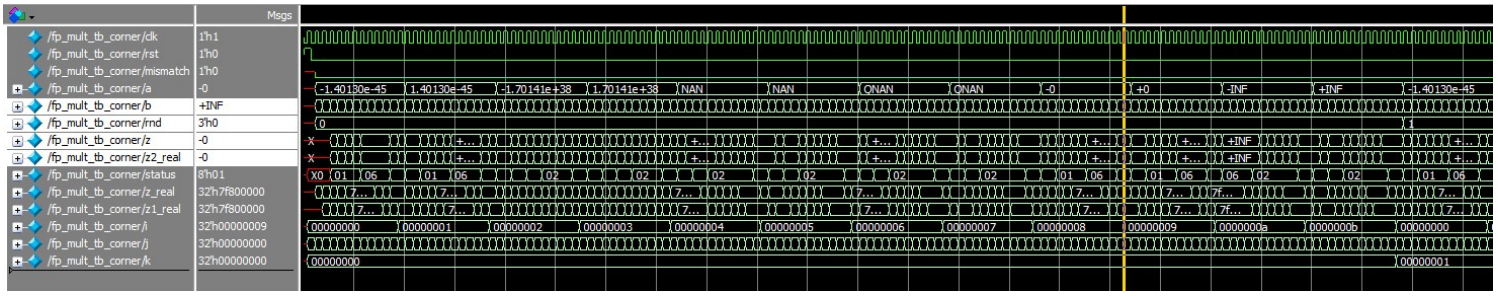


Σχήμα 6: Το πρώτο απόκομμα της κυματομορφής που παράγεται. Φαίνεται ότι ξεκινάει με το a να είναι αρνητικός dernormal και το b να παίρνει όλες τις οριακές τιμές από τον αρνητικό dernormal ως το $+\infty$.

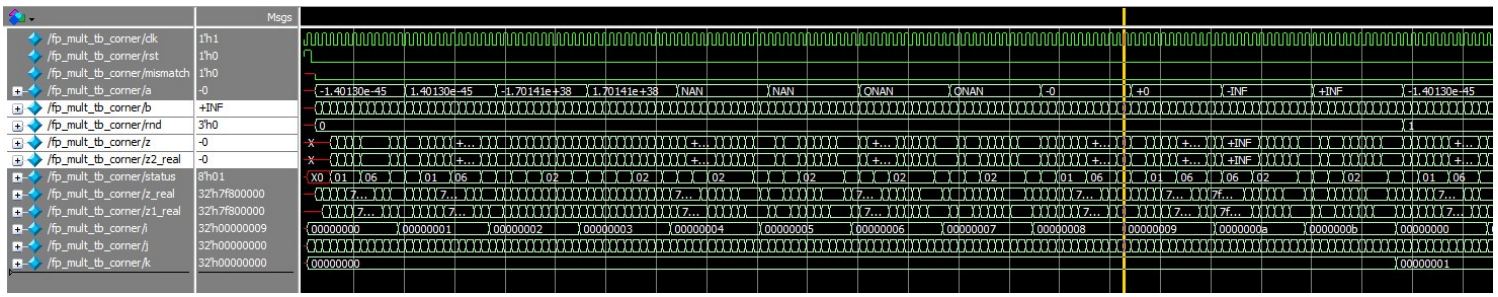
Σε αντιστοιχία με το προηγούμενο διάγραμμα φαίνεται ότι το z2_real και το z παίρνουν τιμή μετά από δύο κύκλους ρολογιού από την στιγμή που επιλέγονται τα πρώτα a και b. Όπως και το ότι δεν υπάρχει κανένα mismatch.



Σχήμα 7: Το δεύτερο απόκομμα της κυματομορφής που παράγεται. Φαίνονται οι περιπτώσεις στις οποίες $a = +DENORM$ και $a = -NORM$ για $rnd = 0$.



Σχήμα 8: Το τρίτο απόκομμα της κυματομορφής που παράγεται. Φαίνονται όλες οι περιπτώσεις με $rnd = 0$.



Σχήμα 9: Το τέταρτο απόκομμα της κυματομορφής που παράγεται. Φαίνονται όλες οι περιπτώσεις για όλα τα rnd.

6 Assertions

6.1 Immediate Assertions

Στο test_status_bits γίνεται ο έλεγχος για τα status bits. Σύμφωνα με τον πίνακα 3 και την υλοποίηση μέχρι αυτό το σημείο έχει γίνει φανερό ότι κάποια flags του status_bit δεν πρέπει να σηκώνονται ταυτόχρονα. Τα flags αυτά είναι τα:

- Το **zero** και το **inf**.
- Το **zero** και το **nan**.
- Το **zero** και το **huge**.
- Το **inf** και το **tiny**.
- Το **nan** και το **tiny**.
- Το **nan** και το **huge**.
- Το **nan** και το **inexact**.
- Το **tiny** και το **huge**.

Για τον έλεγχο αυτών έχουν οριστεί κάποια assertions. Τα assertions αυτά ελέγχονται σε κάθε θετική ακμή του ρολογιού, δηλαδή κάθε φορά που παράγεται ένα αποτέλεσμα, καθώς είναι επιθυμητό να γίνει έλεγχος για όλα τα αποτελέσματα. Εάν το αποτέλεσμα της παρένθεσης του assert είναι θετική σημαίνει ότι τα δύο status bits δεν γίνονται ταυτόχρονα 1, οπότε εκτυπώνεται "PASS-IMMEDIATE" στο transcript. Σε αντίθετη περίπτωση θα εκτυπώσει "FAIL:" και το αντίστοιχο κριτήριο που δεν ικανοποιήθηκε.

```
zero_infinity:   assert (!(status[0] && status[1])) $display("PASS-IMMEDIATE");
else $display("FAIL:␣Zero␣and␣Infinity␣flags␣both␣asserted.");
```

6.2 Concurrent Assertions

Στο test_status_z_combinations γίνεται ο έλεγχος για την μορφή του z κάθε φορά που σηκώνεται κάποιο flag. Σύμφωνα με τις οδηγίες της εκφώνησης, δημιουργήθηκαν τα κατάλληλα assertions. Τα assertions που ελέγχουν την μορφή του z για τον ίδιο κύκλο ρολογιού έχουν παρόμοια μορφή, όπως το παράδειγμα που ακολουθεί. Έχει χρησιμοποιηθεί ο τελεστής |-> που ορίζει ότι εάν η έφραση πριν από αυτόν, δηλαδή η status[0] είναι true. Τότε και μόνο τότε θα ελέγξει **στον ίδιο κύκλο ρολογιού** εάν ικανοποιείται η έκφραση που ακολουθεί. Οι έλεγχοι αυτοί πραγματοποιούνται στην θετική ακμή του ρολογιού όπως ζητείται από την εκφώνηση καθώς η πρώτη συνθήκη status[0] ελέγχεται στο @(posedge clk).

```
  property zero_exponent_zero;
    @(posedge clk)
      (status[0] |-> (exponent_z === 8'b00000000));
  endproperty
  assert property (zero_exponent_zero) $display("PASS␣--␣CONCURRENT␣--");
  else $display("FAIL:␣Exponent's␣bits␣of␣'z'␣are␣not␣all␣zeros␣when␣'zero'␣is␣asserted."
```

6.3 Assertions test module

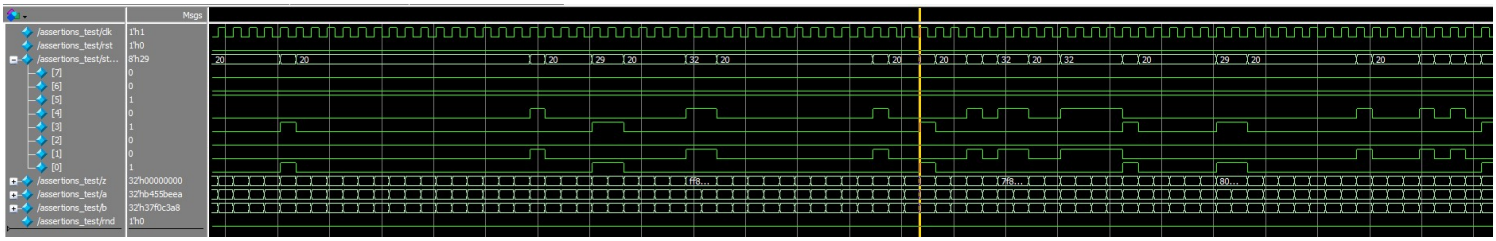
Για να πραγματοποιηθούν όμως οι έλεγχοι αυτοί χρειάζεται να γίνουν πάνω σε ένα DUT(Device Under Test) το οποίο να εναλλάσσει τιμές. Οι τιμές στο DUT εναλλάσσονται τυχαία. Προκειμένου να τρέξουν όντως οι έλεγχοι αυτοί χρησιμοποιούμε την εντολή bind για να ενώσουμε τα δύο modules.

```
    bind mymult test_status_z_combinations test_status_z_combinations_DUT(.clk(clk),
    .status(status), .z(z), .a(a), .b(b));
    bind mymult test_status_bits test_status_bits_DUT(.clk(clk), .status(status));
```

Προσομοιώνοντας το module αυτό του ελέγχου στο transcript εκτυπώνονται τα εξής με τυχαίο τρόπο, ανάλογα με το ποιο κριτήριο ενεργοποιείται και ελέγχεται πρώτο. Προφανώς τα Immediate Assertions εκτυπώνουν πιο συχνά καθώς ελέγχονται όλα στην θετική ακμή του ρολογιού, ενώ τα Concurrent Assertion ελέγχονται στην θετική ακμή του ρολογιού έφοσον ισχύει η πρώτη συνθήκη.

Σχήμα 10: Το Transcript στο οποίο εκτυπώνονται τα $display μια τυχαία χρονική στιγμή.

Παρατηρείται ότι όλα τα assertions εκτυπώνουν PASS το οποίο σημαίνει ότι δεν πραγματοποιείται, επιτυχώς, κάποια από τις ανεπιθύμητες συνθήκες. Και από την κυματομορφή που παράγεται φαίνεται ότι το a και το b παίρνουν συνέχεια τυχαίες τιμές. Εφόσον δεν υπάρχει στο assertions_test module κάποιο $finish, η διαδικασία αυτή συνεχίζει επ' άπειρο μέχρι να γίνει παύση της προσομοίωσης.



Σχήμα 11: Η κυματομορφή που παράγεται όταν εξετάζονται τα assertions.

7 Παράρτημα

Στην εργασία αυτή για την προσομοίωση των modules έχει χρησιμοποιηθεί το εργαλείο Questa - Intel FPGA Starter Edition 2021.2 (Quartus Prime Pro 21.1). Τα αρχεία .sv ήταν σε ένα project κατά το compile και το simulation στο εργαλείο, οπότε δεν χρειαζόταν να γίνουν `include τα άλλα αρχεία που χρειαζόταν στο εκάστοτε module.