Report: Floating Point Multiplier

Παναγιώτης Κούτρης

pkoutris@ece.auth.gr (10671)

Contents

1 Εισαγωγή

Η εργασία αφορά την υλοποίηση και επαλήθευση ενός πολλαπλασιαστή κινητής υποδιαστολής μονής ακρίβειας (IEEE-754) σε SystemVerilog.

Αναπτύχθηκαν τέσσερις επιμέρους μονάδες: `fp_mult`, `normalize_mult`, `round_mult` και `exception_mult`. Η λειτουργία τους ελέγχθηκε με `testbench` και **SystemVerilog Assertions (SVA)**. Η προσομοίωση έγινε με χρήση του εργαλείου `Questa - Intel FPGA Edition`.

2 Multiplier

2.1 normalize_mult.sv

Η μονάδα `normalize_mult` δέχεται ως είσοδο το γινόμενο δύο σημαντικών, 48-bit, καθώς και έναν ενδιάμεσο εκθέτη 10-bit. Σκοπός της μονάδας είναι να ευθυγραμμίσει τη mantissa ώστε να έχει τη σωστή μορφή για τα επόμενα στάδια του πολλαπλασιαστή.

Αν το πιο σημαντικό bit του γινομένου είναι 1, τότε το αποτέλεσμα μετατοπίζεται προς τα δεξιά και ο εκθέτης αυξάνεται κατά 1. Αντίθετα, αν το bit αυτό είναι 0, δεν πραγματοποιείται μετατόπιση και ο εκθέτης παραμένει ίδιος.

Η έξοδος `mantissa_norm` είναι 23-bit και αποτελεί την κανονικοποιημένη mantissa. Παράλληλα, υπολογίζονται δύο επιπλέον bits:

- Το `guard_bit`, δηλαδή το bit αμέσως μετά την mantissa,
- Το `sticky_bit`, δηλαδή το λογικό OR όλων των υπολοίπων bits, που χρησιμοποιείται αργότερα για σκοπούς στρογγυλοποίησης.

2.2 round_mult.sv

Η μονάδα `round_mult` εφαρμόζει στρογγυλοποίηση στο αποτέλεσμα της κανονικοποιημένης mantissa με βάση το επιλεγμένο είδος στρογγυλοποίησης. Δέχεται ως είσοδο:

- τη mantissa των 24-bit,
- το `guard_bit` και το `sticky_bit`,
- το προσημασμένο αποτέλεσμα (`calculated_sign`),
- και τον τύπο στρογγυλοποίησης από την `enum round_mode`.

Η στρογγυλοποίηση γίνεται προσθέτοντας 1 στο λιγότερο σημαντικό bit της mantissa εάν πληρούνται οι αντίστοιχες συνθήκες ανά περίπτωση. Ο προσδιορισμός του `round_up` βασίζεται στον τύπο στρογγυλοποίησης και τα σήματα `guard`, `sticky`, καθώς και στο πρόσημο.

Το αποτέλεσμα είναι 25-bit, ώστε να καλύπτεται ενδεχόμενο overflow. Παράλληλα υπολογίζεται το `inexact` bit, το οποίο είναι 1 όταν υπάρχει απώλεια ακρίβειας λόγω των επιπλέον bits (`guard` ή `sticky`).

Η μονάδα κάνει χρήση του `package round_pkg`, στο οποίο ορίζονται οι υποστηριζόμενοι τρόποι στρογγυλοποίησης με `typedef enum`:

- `IEEE_near` – προς τον πλησιέστερο αριθμό (tie to even),
- `IEEE_zero` – προς το μηδέν,
- `IEEE_pinf` – προς το $+\infty$,
- `IEEE_ninf` – προς το $-\infty$,
- `near_up` – απλή στρογγυλοποίηση προς τον επόμενο αριθμό,
- `away_zero` – στρογγυλοποίηση μακριά από το μηδέν.

2.3 exception_mult.sv

Η μονάδα `exception_mult` διαχειρίζεται περιπτώσεις του πολλαπλασιασμού κινητής υποδιαστολής και παράγει την τελική έξοδο `z`, καθώς και έξι σήματα κατάστασης: `zero_f`, `inf_f`, `nan_f`, `tiny_f`, `huge_f`, `inexact_f`.

Η μονάδα βασίζεται σε έναν τύπο `enum` με τις κατηγορίες `ZERO`, `INF`, `NORM`, `MIN_NORM`, `MAX_NORM`, καθώς και δύο βοηθητικές συναρτήσεις:

- `num_interp`: κατηγοριοποιεί 32-bit αριθμούς σε μία από τις παραπάνω κλάσεις.
- `z_num`: επιστρέφει το αντίστοιχο 31-bit σώμα του αριθμού, ανάλογα με την κλάση.

Αρχικά ανιχνεύονται περιπτώσεις συνδυασμών των εισόδων `a` και `b` με βάση τα corner cases που περιγράφονται στον Πίνακα 4. Παραδείγματα περιλαμβάνουν το `zero × inf` που οδηγεί σε `+inf`, ή το `zero × norm` που δίνει `zero`.

Σε περίπτωση overflow, η λογική έχει ως εξής:

- Για στρογγυλοποίηση προς άπειρο (`IEEE_near`, `near_up`, `away_zero`) η έξοδος `z` λαμβάνει τιμή `INF` και ενεργοποιείται το `inf_f`.
- Για στρογγυλοποίηση προς το μηδέν (`IEEE_zero`), το αποτέλεσμα περιορίζεται στο `MAX_NORM`.
- Για στρογγυλοποίηση προς `+inf` ή `inf`, η επιλογή εξαρτάται από το πρόσημο. Αν το πρόσημο συμφωνεί με την κατεύθυνση, τότε παράγεται `INF`, αλλιώς `MAX_NORM`.

Αντίστοιχα, σε περίπτωση underflow:

- Για στρογγυλοποίηση προς το μηδέν ή χωρίς κατεύθυνση (`IEEE_zero`, `IEEE_near`, `near_up`) παράγεται `ZERO`.
- Για στρογγυλοποίηση προς άπειρο, το αποτέλεσμα είναι είτε `ZERO` είτε `MIN_NORM` ανάλογα με το πρόσημο.
- Για `away_zero`, η έξοδος γίνεται `MIN_NORM`.

Σε κάθε άλλη περίπτωση η έξοδος `z` λαμβάνει την τιμή του `z_calc`.

2.4 fp_mult.sv

Η `fp_mult` αποτελεί την κύρια μονάδα του πολλαπλασιαστή κινητής υποδιαστολής, στην οποία συνδυάζονται όλα τα επιμέρους υποσυστήματα. Η μονάδα είναι χρονισμένη και περιλαμβάνει ενδιάμεσο pipeline στάδιο. Οι είσοδοι είναι οι αριθμοί `a`, `b`, το σήμα στρογγυλοποίησης `rnd`, και τα σήματα ρολογιού `clk` και ασύγχρονου reset `rst`. Οι έξοδοι είναι το αποτέλεσμα `z` και η σημαία κατάστασης `status`.

Η επεξεργασία πραγματοποιείται σε έξι στάδια:

1. **Ανάλυση και Προετοιμασία (Unpack)**: Εξάγονται το πρόσημο, οι εκθέτες και οι mantissas των εισόδων. Οι mantissas μετατρέπονται σε 24-bit (με προσθήκη λανθάνοντος άσου) και πολλαπλασιάζονται παράγοντας 48-bit αποτέλεσμα `P`. Ο εκθέτης υπολογίζεται ως άθροισμα των δύο μείον το bias (127).

2. **Κανονικοποίηση** (`normalize_mult`): Το αποτέλεσμα `P` κανονικοποιείται και προκύπτει mantissa 23-bit, εκθέτης 10-bit, καθώς και τα βοηθητικά `guard_bit` και `sticky_bit`.

3. **Pipeline Register**: Όλα τα απαραίτητα ενδιάμεσα σήματα αποθηκεύονται σε καταχωρητές που συγχρονίζονται με το ρολόι. Η στρογγυλοποίηση `rnd` μετατρέπεται από 3-bit σε τύπο `round_mode`.

4. **Στρογγυλοποίηση** (`round_mult`): Η mantissa στρογγυλοποιείται με βάση το σήμα `round_mode`, λαμβάνοντας υπόψη τα bits `guard` και `sticky`. Η έξοδος έχει μήκος 25 bits και παράγεται επίσης το σήμα `inexact`.

5. **Post Rounding**: Αν υπάρχει υπερχείλιση στο MSB της mantissa, γίνεται επιπλέον δεξιά μετατόπιση και προσαύξηση του εκθέτη. Ο εκθέτης περιορίζεται στα όρια [0, 255] και υπολογίζεται το προσωρινό αποτέλεσμα `z_calc`, καθώς και οι σημαίες `overflow` και `underflow`.

6. **Διαχείριση Εξαιρέσεων** (`exception_mult`): Εξετάζονται ειδικές περιπτώσεις και ενεργοποιούνται τα κατάλληλα σήματα κατάστασης. Η τελική έξοδος `z` λαμβάνει είτε την τιμή `z_calc` είτε κάποια τροποποιημένη μορφή, ανάλογα με το σενάριο.

Τέλος, το σήμα `status` αποτελείται από 8 bits, εκ των οποίων χρησιμοποιούνται τα 6 για τα σήματα εξόδου `zero_f`, `inf_f`, `nan_f`, `tiny_f`, `huge_f`, και `inexact_f`. Τα δύο ανώτερα bits παραμένουν μηδενικά.

3 Testbenches

3.1 testbench.sv

Η μονάδα `testbench` χρησιμοποιείται για την επαλήθευση της ορθής λειτουργίας του πολλαπλασιαστή `fp_mult_top` μέσω αυτόματης δημιουργίας και ελέγχου τυχαίων δεδομένων εισόδου. Έγιναν μερικές πολύ μικρές αλλαγές στο δοσμένο `fp_mult_top` που φαίνονται μέσω αντίστοιχων comments σε αυτό, και γι' αυτό δεν θα μπούμε σε λεπτομέρειες. Παρατίθεται στον φάκελο `exercise2`.

Η προσομοίωση λειτουργεί χρονισμένα με περίοδο 10ns και ακολουθεί δομή με καθυστέρηση 4 κύκλων μεταξύ εισαγωγής και παραγωγής αποτελέσματος.

Πιο συγκεκριμένα:

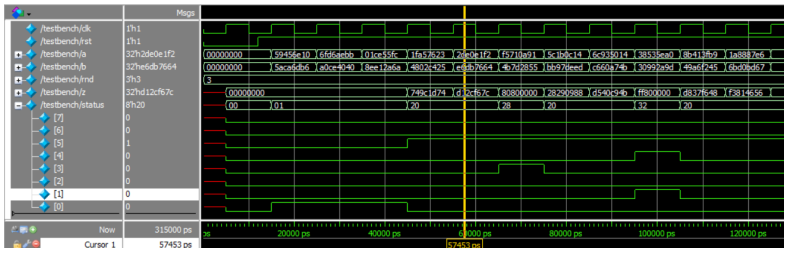
- Για κάθε κύκλο, δημιουργούνται δύο τυχαίοι αριθμοί κινητής υποδιαστολής 32-bit (`a`, `b`) μέσω της συνάρτησης `gen_valid_float()`.
- Το προσδοκώμενο αποτέλεσμα `expected_z` υπολογίζεται μέσω της συνάρτησης `multiplication()` που μας παρέχεται.
- Τα δεδομένα αποθηκεύονται σε πίνακα `test_queue`, ο οποίος περιέχει για κάθε δοκιμή:
  - τα δεδομένα εισόδου `a`, `b`,
  - το αναμενόμενο αποτέλεσμα,
  - τον τύπο στρογγυλοποίησης (`round_str`),
  - την περιγραφή του τεστ και
  - τον κύκλο στον οποίο αναμένεται το αποτέλεσμα.
- Κατά την άφιξη του κύκλου εξόδου, συγκρίνεται το αποτέλεσμα του κυκλώματος `z` με το αναμενόμενο αποτέλεσμα και καταγράφεται επιτυχία ή σφάλμα.

Ο τύπος στρογγυλοποίησης μπορεί να ρυθμιστεί μέσω της μεταβλητής `rmode` στην αρχή του testbench (σειρά 14 του κώδικα). Εδώ χρησιμοποιείται η επιλογή `IEEE_ninf`, αλλά τα αποτελέσματα δοκιμάστηκαν και είναι ορθά για οποιονδήποτε τύπο στρογγυλοποίησης.

Η ροή εκτέλεσης ολοκληρώνεται αυτόματα μετά το πέρας των δοκιμών, εμφανίζοντας αναφορές σύγκρισης μεταξύ πραγματικών και αναμενόμενων τιμών. Παρακάτω παρουσιάζονται ενδεικτικά screenshots με κάποιες ενδεικτικές κυμματομορφές και τα logs που τυπώθηκαν στο transcript.

Σημείωση: Το αρχείο `mult_pkg.sv` περιλαμβάνει τη συνάρτηση `multiplication()`, η οποία υλοποιεί αριθμητικό πολλαπλασιασμό κινητής υποδιαστολής βάσει της στρογγυλοποίησης, χωρίς pipeline, και χρησιμοποιείται ως σημείο αναφοράς (oracle).

```
# === Starting Random Test ===
# Rounding mode set to: IEEE_ninf
# Cycle 15000: Applied Random Test 0, Round: IEEE_ninf
# Cycle 25000: Applied Random Test 1, Round: IEEE_ninf
# Cycle 35000: Applied Random Test 2, Round: IEEE_ninf
# Cycle 45000: Applied Random Test 3, Round: IEEE_ninf
# Cycle 55000: Applied Random Test 4, Round: IEEE_ninf
# --- Result for Random Test 0, Round: IEEE_ninf ---
# A = 59456e10 (3473224088158208.000000)
# B = 5aca6db6 (28489286873513984.000000)
# Expected z = 749c1d74 (98949669223241047748561126555648.000000)
# Actual   z = 749c1d74 (98949669223241047748561126555648.000000)
# Status     = 00100000
# Match
# Cycle 65000: Applied Random Test 5, Round: IEEE_ninf
# --- Result for Random Test 1, Round: IEEE_ninf ---
# A = 6fd6aebb (13288205655382617361431396352O.000000)
# B = a0ce4040 (-0.000000)
# Expected z = d12cf67c (-46429356032.000000)
# Actual   z = d12cf67c (-46429356032.000000)
# Status     = 00100000
# Match
# Cycle 75000: Applied Random Test 6, Round: IEEE_ninf
# --- Result for Random Test 2, Round: IEEE_ninf ---
# A = 01ce55fc (0.000000)
# B = 8ee12a6a (-0.000000)
# Expected z = 80800000 (-0.000000)
# Actual   z = 80800000 (-0.000000)
# Status     = 00101000
# Match
# Cycle 85000: Applied Random Test 7, Round: IEEE_ninf
# --- Result for Random Test 3, Round: IEEE_ninf ---
# A = 1fa57623 (0.000000)
# B = 4802c425 (133904.578125)
# Expected z = 28290988 (0.000000)
# Actual   z = 28290988 (0.000000)
# Status     = 00100000
# Match
# Cycle 95000: Applied Random Test 8, Round: IEEE ninf
```



3.2 testbench2.sv

Η μονάδα `testbench2` ελέγχει την ορθότητα της υλοποίησης για όλες τα δυνατά συνδυαστικά corner cases, που σχετίζονται με τις 12 προκαθορισμένες κατηγορίες αριθμών κινητής υποδιαστολής. Η επαλήθευση καλύπτει συνολικά **144 συνδυασμούς** (12×12), χρησιμοποιώντας σταθερά προκαθορισμένα πρότυπα 32-bit για κάθε τύπο εισόδου.

Κατηγορίες που εξετάζονται:

- signaling NaNs (θετικό / αρνητικό)
- quiet NaNs (θετικό / αρνητικό)
- άπειρο (θετικό / αρνητικό)
- κανονικοί αριθμοί (θετικό / αρνητικό)
- υποκανονικοί (subnormals) (θετικό / αρνητικό)
- μηδενικά (θετικό / αρνητικό)

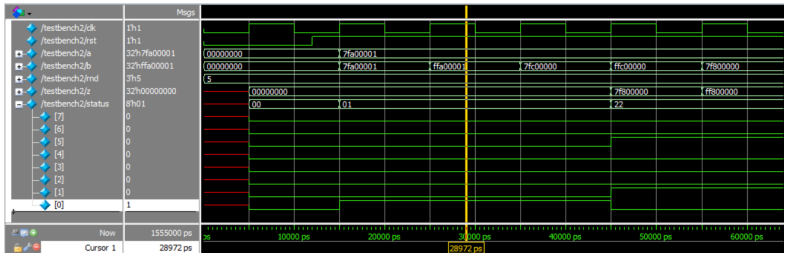
Οι τιμές κωδικοποιούνται μέσω του τύπου `corner_case_t`, ενώ η συνάρτηση `get_pattern()` επιστρέφει το αντίστοιχο δυαδικό πρότυπο ανά κατηγορία.

Για κάθε συνδυασμό:

1. Τα σήματα εισόδου `a` και `b` λαμβάνουν συγκεκριμένο pattern.
2. Υπολογίζεται το αναμενόμενο αποτέλεσμα `expected_z` με τη συνάρτηση `multiplication()` από το πακέτο `mult_pkg`.
3. Το κύκλωμα αξιολογείται και μετά από 4 κύκλους το αποτέλεσμα συγκρίνεται με την αναμενόμενη τιμή.
4. Αν υπάρχει ασυμφωνία, εμφανίζεται σφάλμα μέσω `$error`.

Η επιλεγμένη στρογγυλοποίηση είναι `away_zero`, αλλά μπορεί να τροποποιηθεί με την αντίστοιχη μεταβλητή.Παρακάτω παραθέτουμε υλικό όμοιο με του testbench.sv.

```
# === Starting Corner Case Test ===
# Rounding mode set to: away_zero
# Cycle 15000: Applied Corner Case Test 0: pos_snan x pos_snan
# Cycle 25000: Applied Corner Case Test 1: pos_snan x neg_snan
# Cycle 35000: Applied Corner Case Test 2: pos_snan x pos_qnan
# Cycle 45000: Applied Corner Case Test 3: pos_snan x neg_qnan
# Cycle 55000: Applied Corner Case Test 4: pos_snan x pos_inf
# --- Result for Corner Case Test 0: pos_snan x pos_snan ---
# A = 7fa00001 (nan)
# B = 7fa00001 (nan)
# Expected z = 7f800000 (inf)
# Actual   z = 7f800000 (inf)
# Status     = 00100010
# Match
# Cycle 65000: Applied Corner Case Test 5: pos_snan x neg_inf
# --- Result for Corner Case Test 1: pos_snan x neg_snan ---
# A = 7fa00001 (nan)
# B = ffa00001 (-nan)
# Expected z = ff800000 (-inf)
# Actual   z = ff800000 (-inf)
# Status     = 00100010
# Match
# Cycle 75000: Applied Corner Case Test 6: pos_snan x pos_norm
# --- Result for Corner Case Test 2: pos_snan x pos_qnan ---
# A = 7fa00001 (nan)
# B = 7fc00000 (nan)
# Expected z = 7f800000 (inf)
# Actual   z = 7f800000 (inf)
# Status     = 00100010
# Match
# Cycle 85000: Applied Corner Case Test 7: pos_snan x neg_norm
# --- Result for Corner Case Test 3: pos_snan x neg_qnan ---
# A = 7fa00001 (nan)
# B = ffc00000 (-nan(ind))
# Expected z = ff800000 (-inf)
# Actual   z = ff800000 (-inf)
# Status     = 00100010
# Match
# Cycle 95000: Applied Corner Case Test 8: pos_snan x pos_sub
```



4 Assertions

4.1 test_status_bits.sv

Η μονάδα `test_status_bits` περιέχει **Immediate Assertions** που ελέγχουν την αμοιβαία αποκλειστικότητα μεταξύ συγκεκριμένων bits κατάστασης του σήματος `status`.

Ο σκοπός αυτών των assertions είναι να ανιχνεύσουν λογικά σφάλματα κατά τα οποία ενεργοποιούνται ταυτόχρονα flags που, σύμφωνα με την επιθυμητή λειτουργία του πολλαπλασιαστή, δεν θα πρέπει να συνυπάρχουν.

Ελεγχόμενοι συνδυασμοί:

- `zero_f` και `inf_f`
- `zero_f` και `nan_f`
- `inf_f` και `nan_f`
- `zero_f` και `tiny_f`
- `inf_f` και `tiny_f`
- `nan_f` και `tiny_f`
- `zero_f` και `huge_f`
- `inf_f` και `huge_f`
- `nan_f` και `huge_f`
- `tiny_f` και `huge_f`

4.2 test_status_z_combinations.sv

Η μονάδα `test_status_z_combinations` περιλαμβάνει **συγχρονισμένες Concurrent Assertions**, οι οποίες ελέγχουν τη συνέπεια ανάμεσα στα bits κατάστασης (`status`) και τη δυαδική μορφή του αποτελέσματος `z`.

Για τον έλεγχο του `nan_f`, η μονάδα εισάγει buffers καθυστέρησης τριών κύκλων για τους εκθέτες των εισόδων `a` και `b`, ώστε να ληφθεί υπόψη η καθυστέρηση αγωγού (pipeline).

Οι Concurrent Assertions είναι οι εξής:

- `zero_f` ⇒ `ο εκθέτης του z είναι μηδενικός` (`exp_z == 0`)
- `inf_f` ⇒ `ο εκθέτης του z είναι 255` (`exp_z == 255`)
- `nan_f` ⇒ `τρεις κύκλους νωρίτερα: exp_a = 0, exp_b = 255` ή και το αντίστροφο
- `huge_f` ⇒ `το z είναι είτε Inf είτε maxNormal`
- `tiny_f` ⇒ `το z είναι είτε Zero είτε minNormal`

Binding και testing Έπειτα από binding και των 2 assertions και στα 2 testbench και μερικών μικρών διορθώσεων στο `exception_mult` module, περάστηκαν όλα τα test και σταμάτησαν να πετάγονται μηνύματα σφάλματος.Στην υλοποίηση μας δεν πετάγονται μηνύματα επιτυχίας αλλά δεν έχει πρακτική σημασία καθώς στην περίπτωση σφάλματος θα ενημερωθούμε με αντίστοιχο μήνυμα. Το binding έγινε έπειτα από την σειρά `endmodule` του κάθε testbench. Και μερικές φορές χρειάστηκε επανεκκίνηση του περιβάλλοντος για να μην πετάγεται error κατά την εκκίνηση του simulation.

5 Συμπεράσματα

Η εργασία αποτέλεσε μια ολοκληρωμένη εμπειρία σχεδιασμού, υλοποίησης και επαλήθευσης ενός σύγχρονου κυκλώματος κινητής υποδιαστολής. Η υποστήριξη διαφορετικών τρόπων στρογγυλοποίησης, η ακριβής αντιμετώπιση corner cases και η χρήση `SystemVerilog Assertions` συνέβαλαν καθοριστικά στη δομική και λειτουργική αξιοπιστία της υλοποίησης.Το έργο ανέδειξε την πολυπλοκότητα του σχεδιασμού ψηφιακών αριθμητικών μονάδων αλλά και τη χρησιμότητα μιας μεθοδικής και σταδιακής προσέγγισης από την περιγραφή έως και την επαλήθευση.

Σημείωση:

Για οποιοδήποτε τεχνικό πρόβλημα ή σφάλμα κατά το άνοιγμα των αρχείων (π.χ. σε περίπτωση κατεστραμμένου αρχείου ή μη αναγνωρίσιμου module), παρακαλώ μη διστάσετε να επικοινωνήσετε μαζί μου στο email που αναφέρεται στην αρχή της αναφοράς.