

# Ψηφιακά HW 2

## Αναφορά εργασίας

Τσαρναδέλης Αθανάσιος Γρηγόριος  
ΑΕΜ: 10388

## Πίνακας περιεχομένων

---

Παραδοτέα αρχεία .....	3
Main Module – fp_mult.sv .....	3
Normalization module – normalize-mult.sv .....	3
Rounding module – rounding_mult.sv.....	4
Exception Handling Module – exception_mult.sv .....	4
Testbench – fp_mult_tb.sv .....	5
Immediate assertions – test_status_bits.sv .....	6
Concurrent assertions – test_status_z_combinations.sv.....	7

## Παραδοτέα αρχεία

---

Το παραδοτέο zip περιλαμβάνει τα εξής:

- Φάκελος exercise 1
  - fp\_mult.sv
  - normalize\_mult.sv
  - exception\_mult.sv
  - round\_mult.sv
- Φάκελος exercise 2
  - fp\_mult\_tb.sv (περιλαμβάνει τον έλεγχο rounding και corner cases)
- Φάκελος exercise 3
  - test\_status\_bits.sv (Immediate assertions)
  - test\_status\_z\_combinations.sv (Concurrent assertions)

### Main Module – fp\_mult.sv

---

Αρχικά, εντός του αρχείου sv ορίζω ένα Package, με ένα typedef enum, για να κάνω encoding τον τύπο της στρογγυλοποίησης, όπως συστήθηκε και στην περιγραφή της εργασίας. Έχω ένα τύπο για όλα τα πιθανά rounding του Πίνακα 4. Στην συνέχεια το κάνω Import για να μπορώ να το χρησιμοποιήσω στο module που δηλώνεται ακριβώς από κάτω. Συνεχίζω με την υλοποίηση του main module. Δηλώνω τα εσωτερικά σήματα που χρειάζομαι, και υπολογίζω το πρόσημο από τα sign bits των a,b. Επίσης προσθέτω τους εκθέτες και αφαιρώ το bias από το άθροισμα. Συνεχίζω πολλαπλασιάζοντας τα mantissa των a,b με τα 1 που βρίσκονται στην αρχή, και κάνω instantiate το normalization module και το rounding module. Στην συνέχεια κάνω την διαδικασία Post rounding όπως περιεγράφηκε στο rounding module και παράγονται τα σήματα post mantissa και post\_exp. Χρησιμοποιώντας τα δύο τελευταία, καθώς και το πρόσημο που υπολόγισα νωρίτερα συνθέτω το σήμα z\_calc, που είναι το αποτέλεσμα πριν το exception handler. Κάνω instantiate το Module exception\_mult, και συνδέω τα κατάλληλα σήματα. Από την έξοδο του exception handler, χρησιμοποιώ τα flags και τα συνθέτω, ώστε να δημιουργήσω το σήμα status, όπως περιγράφεται στον πίνακα 3.

### Normalization module – normalize-mult.sv

---

Για το normalization module, ορίζω κατάλληλα τις εισόδους και τις εξόδους και υλοποιώ το κύκλωμα όπως περιγράφεται στο Figure 6. Πιο συγκεκριμένα, υπολογίζω τα κανονικοποιημένα exponent και mantissa, καθώς και τα guard και sticky bits. Χρησιμοποιώ ένα always\_comb και ένα if, με δύο περιπτώσεις, αν το P[47] είναι 1 ή 0, όπως περιγράφεται στην εκφώνηση.

## Rounding module – rounding\_mult.sv

---

Υλοποιώ το rounding module, δηλώνοντας κατάλληλα τις εισόδους και εξόδους, καθώς και τα εσωτερικά σήματα. Επίσης κάνω import του package για το rounding που έχω ορίσει. Ορίζω ένα εσωτερικό σήμα, που είναι το mantissa truncated, και υπολογίζω το inexact σύμφωνα με την εκφώνηση. Χρησιμοποιώ ένα case, με όλες τις περιπτώσεις του πίνακα 4, και αυξάνω το προσωρινό mantissa ανάλογα την περίπτωση. Τέλος, θέτω το προσωρινό result ίσο με το σήμα εξόδου result. Το post mantissa έχει περιγραφεί στο main module παραπάνω. Θα επιβεβαιώσω την σωστή λειτουργία του rounding module παρακάτω.

## Exception Handling Module – exception\_mult.sv

---

Ξεκινάω ορίζοντας το exception\_mult με τις σωστές εισόδους και εξόδους και κάνοντας import το package του rounding. Στο main module υπολογίζω τα overflow και underflow όπως ορίζεται στην εκφώνηση, και τα χρησιμοποιώ σαν εισόδους στο exception module. Εσωτερικά ορίζω επίσης ένα typedef enum με τους προκαθορισμένους αριθμούς που θα χρησιμοποιήσω (Zero, Inf, Norm, Min\_Norm, Max\_Norm).

Στην συνέχεια ορίζω μια συνάρτηση num\_interp, που παίρνει ένα 32bit σήμα και επιστρέφει τον τύπο του σύμφωνα με το πίνακα 1. Αν είναι Inf ή Nan, επιστρέφει Inf, αν είναι Zero ή Denormal επιστρέφει Zero, ενώ αν δεν είναι τίποτα από τα παραπάνω επιστρέφει Norm.

Το αντίστροφο κάνει η επόμενη συνάρτηση z\_num, που παίρνει έναν τύπο typedef enum και επιστρέφει 31bit, το exponent και mantissa δηλαδή, χωρίς το πρόσημο. Επιστρέφει τιμές για τα Zero, Inf, Min\_Norm, Max\_Norm σύμφωνα με τους πίνακες 1 και 2.

Συνεχίζω ορίζοντας ένα always\_comb, εντός του οποίου αρχικοποιώ όλα τα flags στην τιμή 0. Στην συνέχεια ορίζω ένα case με όλες τις πιθανές περιπτώσεις, και ανάλογα την περίπτωση επιστρέφω το κατάλληλο z και θέτω flag αν χρειάζεται. Πχ αν έχω Zero x Zero, τότε σύμφωνα με τον πίνακα 4 πρέπει να επιστρέψω Zero με το πρόσημο της πράξης, άρα επιστρέφω  $z = \{z\_calc[31], z\_num(ZERO)\}$ ; και θέτω `zero_f = 1`.

Συγκεκριμένα για την περίπτωση Norm x Norm, ελέγχω αν έχω overflow ή underflow. Αν έχω overflow, θέτω `huge_f = 1`, `inexact_f = 1` και με βάση το rounding, επιστρέφω Max Norm ή Inf. Όμοια για την περίπτωση που έχω underflow, θέτω `tiny_f = 1`, `inexact_f = 1` και με βάση το rounding, επιστρέφω Min Norm ή Zero.

Αν δεν υπάρχει ούτε overflow ούτε underflow, το z είναι ίσο με το `z_calc` και το `inexact_f` με το `inexact`.

Τέλος ελέγχω αν το `z_calc` είναι Nan, σύμφωνα με τον πίνακα 1, και θέτω το flag ίσο με 1 στην περίπτωση που είναι.

## Testbench – `fp_mult_tb.sv`

---

Για την υλοποίηση του testbench ορίζω τα εσωτερικά σήματα, και κάνω instantiate το `fp_mult_top`, που είναι το DUT. Κάνω bind το DUT με τα modules `test_status_bits` και `test_status_z_combinations`, που αφορούν τα assertions και θα επεξηγηθούν παρακάτω. Ορίζω επίσης ένα typedef enum `corner_case_t` με όλους του τύπους corner cases που θα ελέγξω στο 2<sup>ο</sup> κομμάτι του testbench, όπως αυτοί περιγράφονται στην εκφώνηση, καθώς και μια συνάρτηση `corner_case_to_value`, που παίρνει το corner case και επιστρέφει 32bits της τιμής του. Τέλος, ορίζω ένα array τύπου `corner_case_t` με όλα τα corner cases, που θα χρησιμοποιήσω στο 2<sup>ο</sup> κομμάτι του testbench.

Για το ρολόι, θέτω περίοδο 15 ns.

Σε ένα initial block, θέτω το `clk=0`, και κάνω reset το κύκλωμά μου. Στην συνέχεια χρησιμοποιώ την `urandom`, για να παράξω τυχαίες τιμές για τα `a,b`. Για κάθε rounding mode, ελέγχω αν το αποτέλεσμα είναι ίδιο με το αποτέλεσμα του function multiplication που μας έχει δοθεί και τυπώνω μήνυμα αν το αποτέλεσμα είναι ίδιο ή αν διαφέρει. Παίρνω στο terminal το εξής:

```
VSIM 122> run -all
# IEEE_near Random Test: a= 100634581, b=3225525099
# Test PASSED!
# IEEE_zero Random Test: a=2985452594, b=4120352099
# Test PASSED!
# IEEE_pinf Random Test: a=1005330005, b=3640111589
# Test PASSED!
# IEEE_ninf Random Test: a=1302620452, b=3644309272
# Test PASSED!
# near_up Random Test: a=1887346172, b= 753395799
# Test PASSED!
# away_zero Random Test: a= 153035962, b=1771907954
# Test PASSED!
#
```

Βλέπω ότι το rounding module λειτουργεί σωστά, καθώς περνάει όλα τα test. Ο ίδιος έλεγχος έγινε με πολλά διαφορετικά seed στην `urandom`, ώστε να πάρω διάφορες τιμές για τα `a,b`. Παραπάνω έλεγχοι παραλείπονται από την αναφορά για λόγους συντομίας, αφού η συμπεριφορά του module είναι ίδια.

Στην συνέχεια θέτω ένα default rounding, και κάνω iterate το array που δήλωσα παραπάνω ώστε να πάρω τους 144 συνδυασμούς corner cases που θέλω. Ελέγχω το αποτέλεσμα που παίρνω και αν είναι διαφορετικό από το αποτέλεσμα του function multiplication, τυπώνω μήνυμα. Από την αναφορά παραλείπονται τα μηνύματα Pass για λόγους συντομίας.

```
# -----Starting corner case testing.-----
#
#
# -----Corner case testing completed.-----
```












Βλέπω ότι τα μηνύματα στην αρχή και το τέλος του Corner case testing δεν περιλαμβάνουν μηνύματα λάθους ενδιάμεσα, και άρα θεωρώ ότι τα corner cases, και άρα ο exception handler λειτουργούν σωστά. Ίδια συμπεριφορά παρατηρείται και για όλους τους υπόλοιπους τύπους rounding, οι οποίοι δοκιμάστηκαν και λειτουργούν όπως στην φωτογραφία.

## Immediate assertions – test\_status\_bits.sv

Σχετικά με τα immediate assertions, τα status bits που δεν πρέπει να είναι 1 ταυτόχρονα είναι:

- Zero και Inf
- Zero και NaN
- Zero και Tiny
- Zero και Huge
- Zero και Inexact
- Inf και Tiny
- Inf και Huge
- Inf και Inexact
- Nan και Tiny
- Nan και Huge
- Nan και Inexact

Γράφω λοιπόν τα κατάλληλα assertions, τα οποία τυπώνουν μήνυμα μόνο σε περίπτωση λάθους. Τρέχω το testbench που περιέγραψα παραπάνω, και έχοντας κάνει bind, βλέπω ότι η έξοδος στο terminal δεν αλλάζει, άρα δεν εκτυπώνεται κανένα μήνυμα λάθους. Επιβεβαιώνω από το assertions tab του προσομοιωτή για τα συγκεκριμένα assertions:

Name	Assertion Type	Language	Enable	Failure Count	Pass Count
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__8	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__9	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__10	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__11	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__12	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__13	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__14	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__15	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__16	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__17	Immediate	SVA	on	0	1
 /fp_mult_tb/DUT/dutbound/#ublk#116954275#7/immed__18	Immediate	SVA	on	0	1






Παρατηρώ ότι τα immediate assertions δεν έχουν failures, αλλά έχουν passes, που σημαίνει ότι έχουν αποτιμηθεί και πέρασαν. Έτσι έχω επιβεβαιώσει με assertions ότι το κύκλωμα για τα flags λειτουργεί σωστά.

## Concurrent assertions – test\_status\_z\_combinations.sv

Ακολουθώ την ίδια διαδικασία με πριν, και γράφω τα assertions που περιγράφονται στην εκφώνηση. Ειδικά για το 3<sup>ο</sup>, γράφω ένα sequence ώστε να πάρω τις τιμές που θέλω πριν από δύο κύκλους ρολογιού, χρησιμοποιώντας την \$past. Ελέγχω την λειτουργία τους στο testbench και παρατηρώ ότι βγάζουν μήνυμα λάθους.

```
# -----Starting corner case testing.-----  
#  
# ** Error: NaN status bit set but exponents of 'a' and 'b' are not in required condition  
# Time: 427500 ps Started: 397500 ps Scope: fp_mult_tb.DUT.dutbound_z File: M:/intelFPGA/HW2TestProject/test_status_z_combinations.sv Line: 21
```

Κοιτάζοντας το tab assertions:

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	#
 /fp_mult_tb/DUT/dutbound_z/assert__0	Concurrent	SVA	on	0	1	
 /fp_mult_tb/DUT/dutbound_z/assert__1	Concurrent	SVA	on	0	1	
 /fp_mult_tb/DUT/dutbound_z/assert__2	Concurrent	SVA	on	24	1	
 /fp_mult_tb/DUT/dutbound_z/assert__3	Concurrent	SVA	on	0	0	
 /fp_mult_tb/DUT/dutbound_z/assert__4	Concurrent	SVA	on	0	0	

Βλέπω πως τα δύο πρώτα assertions περνάνε, το τρίτο κάνει 24 fail, ενώ τα δύο τελευταία δεν κάνουν fail ούτε περνάνε. Σχετικά με το 3<sup>ο</sup>, υπάρχει κάποιο λάθος είτε στον τρόπο που γράφτηκε το assertion είτε στο κύκλωμα. Για τα δύο τελευταία δεν ξέρω την κατάστασή τους, και πιθανόν να μην αποτιμήθηκαν ή να μην πρόλαβαν να περάσουν πριν τερματίσει το testbench.