Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Αρχιτεκτονική Προηγμένων Υπολογιστών και Επιταχυντών 2024-2025
Αναφορά 1ου Εργαστηρίου

Ομάδα 3
Γαλάνη Δήμητρα, ΑΕΜ 10331
Σπανόπουλος Γρηγόριος, ΑΕΜ 10459

Διδάσκων: Παπαευσταθίου Ιωάννης

20 Δεκεμβρίου 2024

Εισαγωγή στο Vitis - ALU

Σε πρώτη φάση υλοποιούμε μια απλή αριθμητική μονάδα ALU με τον κώδικα που δίνεται από την εκφώνηση του εργαστηρίου. Στην κυματομορφή βλέπουμε ότι η πράξη της πρόσθεσης, της αφαίρεσης και του πολλαπλασιασμού γίνεται σε έναν κύκλο και στην έξοδο φαίνεται κατευθείαν το αποτέλεσμα. Ενώ στον επόμενο κύκλο ρολογιού σηκώνεται το σήμα ap_done και το σήμα ap_ready ενώ η έξοδος γίνεται απροσδιόριστη. Αφού κατέβουν τα δύο σήματα η ALU είναι έτοιμη να δεχτεί νέα νούμερα, το οποίο συμβαίνει στον επόμενο κύκλο ρολογιού. Οπότε βλέπουμε ότι κάθε δύο κύκλους μπορούμε να πάρουμε αποτέλεσμα, με την προϋπόθεση ότι οι πράξεις εκτελούνται σε έναν κύκλο. Στην περίπτωση της διαίρεσης το αποτέλεσμα υπολογίζεται και φαίνεται στην έξοδο μετά από 35 κύκλους.

Ερώτημα 1

Αρχικά υλοποιήσαμε έναν multiplier χωρίς να κάνουμε βελτιστοποίηση για top layer και ένα testbench που να ελέγχει εάν το αποτέλεσμα που δίνει ο multiplier θα είναι σωστό.

Ερώτημα 2 & Ερώτημα 3

Συνθέσαμε την σχεδίαση (C Synthesis) για $l_m = l_n = l_p = 6$ και προσομοιώσαμε τον κώδικα για επαλήθευση και πήραμε τα παρακάτω αποτελέσματα.

Estimated clock period:	6.304ns
Worst case latency:	131091
Number of DSP48E used:	32
Number of BRAMs used:	0
Number of FFs used:	1491
Number of LUTs used:	4420

Πίνακας 1: Αποτελέσματα Σύνθεσης

Total Execution Time:	1311095 ns
Min latency:	131.089 μs
Avg. latency:	131.089 μs
Max latency:	131.089 μs

Πίνακας 2: Αποτελέσματα Cosimulation

Ερώτημα 4

Κάναμε διάφορες δοκιμές με pragmas και συνδυασμούς τους καθώς και δοκιμάσαμε να τα τοποθετήσουμε σε διαφορετικά σημεία του προγράμματος.

Pipeline

Αρχικά, εισάγαμε το PIPELINE pragma με Initiation Interval (II) = 1 εντός του $2^{ου}$ loop, ώστε να ξεκινάει νέο iteration του loop έναν κύκλο αφού έχει γίνει issue το προηγούμενο iteration. Δηλαδή στην πράξη, να ξεκινάει να επεξεργάζεται ένα νέο στοιχείο του πίνακα κάθε φορά. Επιλέξαμε να μην βάλουμε το pipeline εντός του $3^{ου}$ loop καθώς τότε ο ταυτόχρονος υπολογισμός είναι μόνο στο επίπεδο ενός στοιχείου του τελικού πίνακα, δηλαδή χρειάζεται να γίνουν 64 προσθέσεις και πολλαπλασιασμοί (σύνολο 68 κύκλοι latency) για να ξεκινήσει ο υπολογισμός ενός επόμενου στοιχείου. Ωστόσο, με το να είναι το $2^{ο}$ loop pipelined, υπήρχε II violation(βλ. Εικόνα 1), αφού τα δεδομένα δεν ήταν προσβάσιμα παράλληλα, με αποτέλεσμα να χρειάζεται μεγαλύτερο latency.

Array Partition

Οπότε για να μπορούν να φορτωθούν παράλληλα δεδομένα χρησιμοποιήσαμε το pragma ARRAY_PARTITION. Επιλέξαμε να διαμερίσουμε τον πίνακα A κατά στήλες, οπότε προκύπτουν m nx1 πίνακες-στήλες, ενώ διαμερίσαμε τον πίνακα B κατά γραμμές, και άρα προέκυψαν m 1xp πίνακες-γραμμές. Με αυτόν τον τρόπο, γίνονται παράλληλα access στην μνήμη σε διαφορετικά μπλοκ οπότε είναι εφικτός ο πολλαπλασιασμός των πινάκων, χωρίς να περιμένουμε τα στοιχεία να φορτωθούν ένα-ένα από την μνήμη.

Επίσης, δοκιμάσαμε και partition με cyclic factor=32 όπου είχαμε λίγο περισσότερα FF (228) και LUT (2350) από όταν χρησιμοποιήσαμε complete partition. Αυτό συμβαίνει γιατί κάποια elements που χρειάζονται σε μια πράξη μπορεί να βρεθούν στο ίδιο μπλοκ και άρα μειώνεται ο παραλληλισμός, οπότε δεν είναι τόσο αποδοτικό.

Αποτελέσματα με pipeline και array Partition

Estimated clock period:	5.358 ns
Number of DSP48E used:	32
Number of BRAMs used:	0
Number of FFs used:	191
Number of LUTs used:	2306

Πίνακας 3: Αποτελέσματα Σύνθεσης

Total Execution Time:	41195 ns
Min latency:	41.010 μs
Avg. latency:	41.010 μs
Max latency:	41.010 μs

Πίνακας 4: Αποτελέσματα Cosimulation

Συνεπώς, η επιτάχυνση με την χρήση pipeline είναι:

$$\alpha = \frac{execution\ \ time\, of\ \ the\ \ initial\ \ design}{execution\ \ time\ \ of\ \ the\ \ final\ \ design} = \frac{1311095}{41195} = 31.83 = 3183\% \tag{1}$$

Unroll

Έπειτα θέλαμε να δούμε την επίδραση που θα έχει το unroll στη σχεδίαση μας. Η υπόθεση μας ήταν ότι, εφόσον θα χωρίσουμε ένα loop σε παράλληλες εκτελέσεις των iterations, η επιτάχυνση μας θα ήταν σημαντικά μεγαλύτερη, ωστόσο σε κάποιο σημείο θα ξεπεραστούν τα διαθέσιμα resources του fpga. Αποφασίσαμε να δοκιμάσουμε πλήρες unroll του εσωτερικού loop, έτσι ώστε οι πράξεις για τον υπολογισμό κάθε στοιχείου του τελικού πίνακα να γίνονται παράλληλα. Φυσικά για να μπορεί να γίνει πρόσβαση σε όλα αυτά τα στοιχεία, κάναμε πλήρες διαμέριση των δύο εισόδων, για τον πίνακα A στην δεύτερη διάσταση ούτως ώστε να μπορούμε να έχουμε ταυτόχρονα πρόσβαση σε μια ολόκληρη σειρά, και αντίστοιχα στο B στην πρώτη διάσταση ώστε να έχουμε ταυτόχρονη πρόσβαση σε μια ολόκληρη στήλη. Για l_n=l_p=l_m=6 (και για μικρότερες διαστάσεις) η υλοποίηση είναι πάρα πολύ αποδοτική, μειώνοντας το latency σε 102 κύκλους. Παράλληλα, το utilization SLR για τα DSPs είναι 89% και για τα LUTs στο 38%. Αυτό σημαίνει ναι μεν ότι η λύση είναι κατά πάσα πιθανότητα υλοποιήσιμη, ωστόσο δεν θα μπορούσε να λειτουργήσει με μεγαλύτερες διαστάσεις. Το finish time είναι 1205ns. Αναλυτικότερα, έχουμε παρακάτω τα ζητούμενα στατιστικά, τα οποία μας παρέχουν και την βέλτιστη λύση της σχεδίασης μας:

Estimated clock period:	5.358 ns
Number of DSP48E used:	2048
Number of BRAMs used:	0
Number of FFs used:	33367
Number of LUTs used:	133279

Πίνακας 5: Αποτελέσματα Σύνθεσης

Total Execution Time:	1205 ns
Min latency:	1.020 μs
Avg. latency:	1.020 μs
Max latency:	1.020 μs

Πίνακας 6: Τελικά Αποτελέσματα Σύνθεσης

Μεταβολή διαστάσεων πινάκων

Όσον αφορά το πως αποκρίνεται το σύστημα με διαφορετικές διαστάσεις, ενώ τα resources ήταν μερικές δεκάδες λιγότερα με την μείωση των διαστάσεων, η διαφορά του execution time ήταν μεγάλη καθώς παρατηρήσαμε ότι υποδιπλασιάζεται όταν μειώνουμε το l_n ή το l_p κατά μια μονάδα. Οπότε όταν μειώνουμε και τα δύο υποτετραπλασιάζεται κ.ο.κ. Αντίστοιχα, όταν τα l_n, l_p γίνονται ίσα με 7, ο χρόνος υπέρτετραπλασιάζεται. Αυτή είναι μια λογική και αναμενόμενη εξέλιξη, καθώς τα n,p έχουν εκθετική σχέση με τα l_n, l_p. Καθώς με το pipeline γίνονται issue κάθε κύκλο νέα στοιχεία, ο αριθμός των κύκλων που χρειάζονται είναι αναλογικός με τον αριθμό των δεδομένων.

Επιτάχυνση της υλοποίησης

Για να βρούμε την επιτάχυνση θα διαιρέσουμε το αρχικό χρόνο εκτέλεσης με τον χρόνο εκτέλεσης που προκύπτει μετά την βελτιστοποίηση της υλοποίησης. Δηλαδή:

$$\alpha = \frac{execution\ \ time\, of\ \ the\ \ initial\ \ design}{execution\ \ time\ \ of\ \ the\ \ final\ \ design} = \frac{1311095}{1205} = 1088 = 108800\% \tag{2}$$