ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
&
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
&
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μετατροπή κώδικα Matlab σε γλώσσα προγραμματισμού C και η χρήση του CCC για παραγωγή Hardware απο αυτό.

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΔΟΎΛΑΛΑ ΝΙΚΟΛΑΟΥ

Α.Ε.Μ. 70

Επιβλέπων : ΔΟΣΗΣ ΜΙΧΑΗΛ

Καστοριά Μάιος 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
&
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
&
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μετατροπή κώδικα Matlab σε γλώσσα προγραμματισμού C και η χρήση του CCC για παραγωγή Hardware απο αυτό.

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΔΟΎΛΑΛΑ ΝΙΚΟΛΑΟΥ

Α.Ε.Μ. 70

Επιβλέπων : ΔΟΣΗΣ ΜΙΧΑΗΛ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26 Μαΐου 2023

.....	.....	.....
Ον/μο Μέλους	Ον/μο Μέλους	Ον/μο Μέλους

Καστοριά Μάιος 2023

Ευχαριστίες

Με την παρούσα διπλωματική εργασία ολοκληρώνονται οι σπουδές μου στο πρόγραμμα μεταπτυχιακών σπουδών «Προηγμένες Τεχνολογίες Πληροφορικής και Υπηρεσίες» του Πανεπιστημίου Δυτικής Μακεδονίας.

Στις σπουδές μου ήταν καθοριστική η συμβολή των καθηγητών μου στα γνωστικά αντικείμενα που παρακολούθησα και οφείλω να εκφράσω τις ευχαριστίες μου για τις γνώσεις που μου μετέφεραν και την συμβολή τους γενικότερα στην ολοκλήρωση των σπουδών μου.

Ιδιαίτερα θέλω να ευχαριστήσω τον καθηγητή μου και επιβλέποντα την παρούσα διπλωματική εργασία, κο Μιχάλη Δόση για την επιστημονική και συμβουλευτική καθοδήγηση που μου προσέφερε σε όλα τα στάδια εκπόνησης της εργασίας με τις εύστοχες και πολύ εποικοδομητικές παρατηρήσεις του.

Τέλος, οφείλω να ευχαριστήσω την οικογένειά μου, για τη συμπαράσταση και την υπομονή τους.

ΠΕΡΙΛΗΨΗ

Η εργασία αφορά την μετατροπή κώδικα Matlab σε γλώσσα προγραμματισμού C, και η εισαγωγή του στο πρόγραμμα CCC για την παραγωγή hardware από αυτό. Ο αρχικός κώδικας, (Matlab), υλοποιούσε συνάρτηση η οποία αποτελεί μέρος αλγόριθμου προσομοίωσης κοπτικού εργαλείου. Σκοπός της συνάρτησης είναι να υπολογιστούν οι δυνάμεις οι οποίες ασκούνται στο κάθε άτομο, η επιτάχυνση του κάθε ατόμου που προκύπτει από αυτές τις δυνάμεις και η τελική θέση του κάθε ατόμου για το επόμενο χρονικό βήμα μέσω διπλής ολοκλήρωσης. Ουσιαστικά πρόκειται για έναν αλγόριθμο υπολογισμού δυνάμεων μοριακής δυναμικής με χρήση High Level Synthesis (HLS). Το HLS αποτελεί μια διαδικασία σχεδίασης υλικού με την χρήση γλώσσας υψηλού επιπέδου, όπου γίνεται μετατροπή της γλώσσας υψηλού επιπέδου σε γλώσσα περιγραφής υλικού (HDL), και έχει ως αποτέλεσμα την μείωση του χρόνου, της πολυπλοκότητας και του κόστους σχεδιασμού. Η διαδικασία μετατροπής σε γλώσσα περιγραφής υλικού γίνεται μέσω του προγράμματος CCC, και γίνεται σε δυο στάδια. Στο πρώτο στάδιο έχουμε την μετατροπή από γλώσσα προγραμματισμού C σε ADA, και στο δεύτερο στάδιο την μετατροπή του ADA σε HDL. Το CCC είναι ένα εργαλείο βασισμένο σε τυπικές μεθόδους (Formal Methods), και δίνει την δυνατότητα να δημιουργηθεί Hardware το οποίο είναι ειδικού σκοπού, από κώδικα υψηλού επιπέδου. Με αυτόν τον τρόπο μειώνεται ο χρόνος σχεδίασης και υλοποίησης κατά τάξης μεγέθους. Δέχεται ως είσοδο ANSI C ή ADA και παράγει Verilog ή VHDL. Αξίζει να σημειώσουμε τέλος ότι προσφέρεται δωρεάν μέσω εργονομικής ιστοσελίδας για ερευνητική, πειραματική και ακαδημαϊκή αποκλειστικά χρήση.

Λέξεις κλειδιά : CCC, HLS (High Level Synthesis), HDL, Μοριακή δυναμική

Πίνακας Περιεχομένων

Κατάλογος εικόνων

Εισαγωγή

Η Μοριακή Δυναμική είναι μια υπολογιστική μέθοδος για την προσομοίωση συστημάτων που αποτελούνται από ένα πλήθος ατόμων και χρησιμοποιείται ευρέως για τον υπολογισμό των ιδιοτήτων και της συμπεριφοράς διαφόρων ειδών υλικών, την αλληλεπίδρασή τους με άλλα σώματα υπό διάφορες συνθήκες κτλ.

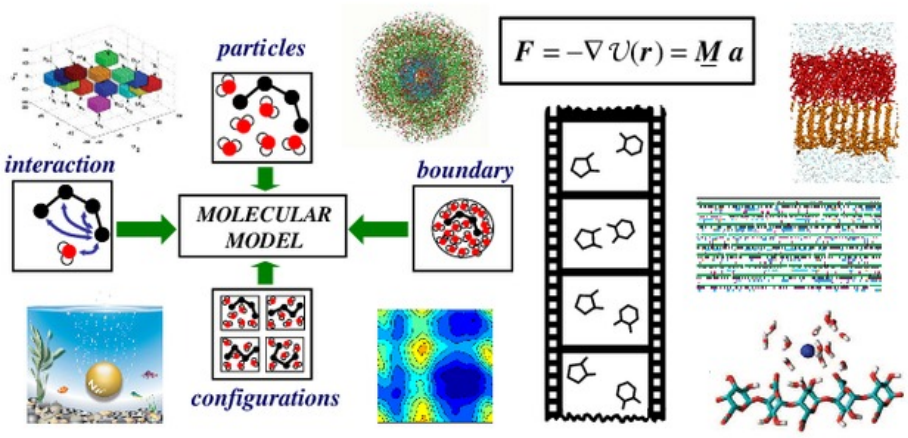
Ωστόσο, με την αυξανόμενη πολυπλοκότητα αυτών των συστημάτων, οι παραδοσιακές προσομοιώσεις μοριακής δυναμικής μπορούν να γίνουν υπολογιστικά ακριβές και χρονοβόρες. Τα ενσωματωμένα συστήματα, τα οποία ενσωματώνουν υλικό και λογισμικό σε μια ενιαία συσκευή, προσφέρουν μια πιθανή λύση σε αυτό το πρόβλημα [1]. Χρησιμοποιώντας σύνθεση υψηλού επιπέδου (HLS) για τον σχεδιασμό και τη βελτιστοποίηση προσομοιώσεων μοριακής δυναμικής για ενσωματωμένα συστήματα, οι ερευνητές μπορούν να επιτύχουν σημαντική επιτάχυνση και ενεργειακή απόδοση, καθιστώντας δυνατή την προσομοίωση μεγαλύτερων και πιο περίπλοκων συστημάτων σε πραγματικό χρόνο [2].

Η HLS είναι μια μεθοδολογία σχεδιασμού που επιτρέπει στους προγραμματιστές να καθορίσουν τη συμπεριφορά ενός συστήματος σε μια γλώσσα προγραμματισμού υψηλού επιπέδου, η οποία στη συνέχεια συντίθεται αυτόματα σε λογική υλικού. Χρησιμοποιώντας η HLS για την υλοποίηση προσομοιώσεων μοριακής δυναμικής, οι ερευνητές μπορούν να επωφεληθούν από τον παραλληλισμό και την απόδοση του υλικού, ενώ παράλληλα επωφελούνται από την ευελιξία και την ευκολία χρήσης του λογισμικού [3]. Αυτή η προσέγγιση μπορεί επίσης να επιτρέψει στους ερευνητές να ενσωματώσουν ανάδραση σε πραγματικό χρόνο από φυσικούς αισθητήρες στις προσομοιώσεις τους, καθιστώντας δυνατή τη δημιουργία συστημάτων ελέγχου κλειστού βρόχου που μπορούν να ανταποκριθούν στις αλλαγές του περιβάλλοντος [4].

Παρά τα πιθανά οφέλη από τη χρήση του HLS για την εφαρμογή προσομοιώσεων μοριακής δυναμικής σε ενσωματωμένα συστήματα, υπάρχουν επίσης σημαντικές προκλήσεις που πρέπει να ξεπεραστούν. Αυτά περιλαμβάνουν τη σχεδίαση αποτελεσματικών αλγορίθμων που μπορούν να εκμεταλλευτούν τους διαθέσιμους πόρους υλικού στα ενσωματωμένα συστήματα, καθώς και τη διασφάλιση ότι οι προσομοιώσεις παραμένουν ακριβείς και αξιόπιστες ενόψει του αναπόφευκτου θορύβου και της μεταβλητότητας των συστημάτων του πραγματικού κόσμου. Ωστόσο, με την πρόοδο στα εργαλεία και τις τεχνικές HLS καθώς και η αυξανόμενη διαθεσιμότητα ισχυρών ενσωματωμένων υπολογιστικών πλατφορμών, η μοριακή δυναμική στα ενσωματωμένα συστήματα γίνεται όλο και πιο πολλά υποσχόμενος τομέας έρευνας τόσο για ακαδημαϊκές όσο και για βιομηχανικές εφαρμογές [5].

Μοριακή Δυναμική

Η μοριακή δυναμική είναι μια ισχυρή τεχνική υπολογιστικής προσομοίωσης, που έχει χρησιμοποιηθεί εκτενώς για τη μελέτη των φυσικών ιδιοτήτων των μορίων και των ατόμων. Οι προσομοιώσεις μοριακής δυναμικής βασίζονται στην κλασική μηχανική, η οποία περιγράφει την κίνηση και την αλληλεπίδραση των σωματιδίων σε ένα σύστημα [6]. Επιλύοντας τις εξισώσεις κίνησης για αυτά τα σωματίδια, οι προσομοιώσεις μοριακής δυναμικής μπορούν να προβλέψουν τη συμπεριφορά και την εξέλιξη ενός συστήματος με την πάροδο του χρόνου.



Εικόνα 1: Μοντέλο μοριακής Δυναμικής [7]

Αρχές της Μοριακής Δυναμικής

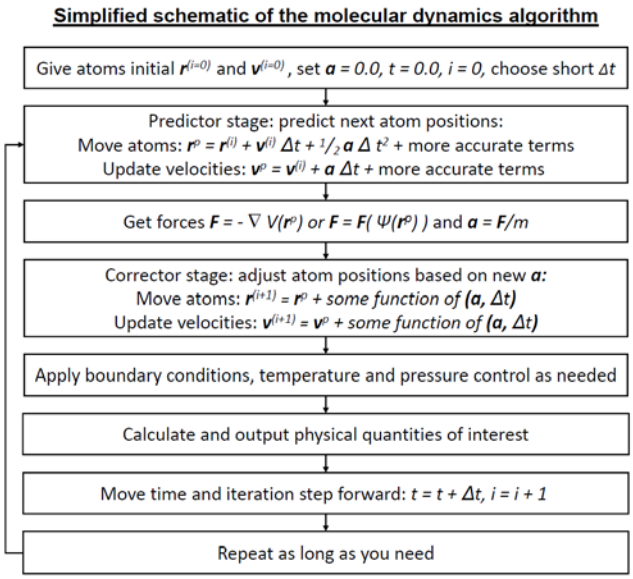
Οι αρχές της μοριακής δυναμικής βασίζονται στην κλασική μηχανική, η οποία είναι ένας κλάδος της φυσικής που ασχολείται με την κίνηση των αντικειμένων υπό την επίδραση δυνάμεων. Στις προσομοιώσεις μοριακής δυναμικής, τα άτομα και τα μόρια ενός συστήματος αντιπροσωπεύονται ως σωματίδια με καθορισμένες φυσικές ιδιότητες όπως μάζα, φορτίο και μέγεθος. Αυτά τα σωματίδια στη συνέχεια υπόκεινται στην κλασική μηχανική, η οποία διέπει τις κινήσεις και τις αλληλεπιδράσεις τους μεταξύ τους [8]. Οι αλληλεπιδράσεις μεταξύ των σωματιδίων περιγράφονται από διατομικά ή διαμοριακά δυναμικά, τα οποία μπορούν να προκύψουν από πειραματικά δεδομένα ή από κβαντομηχανικούς υπολογισμούς. Ενσωματώνοντας τις εξισώσεις κίνησης για αυτά τα σωματίδια, οι προσομοιώσεις μοριακής δυναμικής μπορούν να προβλέψουν τις θέσεις, τις ταχύτητες και τις ενέργειες των σωματιδίων στο σύστημα ανά πάσα στιγμή [9].

Εφαρμογές της Μοριακής Δυναμικής

Οι προσομοιώσεις μοριακής δυναμικής έχουν χρησιμοποιηθεί εκτενώς σε διάφορους τομείς όπως η χημεία, η βιοφυσική, η επιστήμη των υλικών και η μηχανική. Στη χημεία, οι προσομοιώσεις μοριακής δυναμικής μπορούν να χρησιμοποιηθούν για τη μελέτη των ιδιοτήτων υγρών, αερίων και στερεών, καθώς και χημικών αντιδράσεων[10]. Για παράδειγμα, οι προσομοιώσεις μοριακής δυναμικής μπορούν να χρησιμοποιηθούν για την πρόβλεψη των ρυθμών και των μηχανισμών των χημικών αντιδράσεων και για τη μελέτη των ιδιοτήτων των καταλυτών και των ενδιάμεσων αντιδράσεων. Στη βιοφυσική, οι προσομοιώσεις MD χρησιμοποιούνται για τη μελέτη της δομής και της λειτουργίας βιομορίων όπως οι πρωτεΐνες, τα νουκλεϊκά οξέα και οι μεμβράνες [11]. Για παράδειγμα, οι προσομοιώσεις μοριακής δυναμικής μπορούν να χρησιμοποιηθούν για την πρόβλεψη των διαμορφωτικών αλλαγών των πρωτεϊνών και της δέσμευσης των προσδεμάτων σε πρωτεΐνες. Στην επιστήμη των υλικών, οι προσομοιώσεις μοριακής δυναμικής χρησιμοποιούνται για τη μελέτη των ιδιοτήτων υλικών όπως τα πολυμερή, τα κράματα και τα κεραμικά. Για παράδειγμα, οι προσομοιώσεις μοριακής δυναμικής μπορούν να χρησιμοποιηθούν για την πρόβλεψη των μηχανικών ιδιοτήτων των υλικών υπό διαφορετικές συνθήκες και για τη μελέτη της διάχυσης των μορίων στα υλικά[12]. Στη μηχανική, οι προσομοιώσεις μοριακής δυναμικής χρησιμοποιούνται για τη μελέτη των ιδιοτήτων ρευστών και στερεών, καθώς και της συμπεριφοράς συστημάτων όπως εναλλάκτες θερμότητας, κινητήρες και τουρμπίνες [13]. Για παράδειγμα, οι προσομοιώσεις μοριακής δυναμικής μπορούν να χρησιμοποιηθούν για τη βελτιστοποίηση του σχεδιασμού των εναλλάκτη θερμότητας και για την πρόβλεψη της απόδοσης των κινητήρων.

Περιγραφή λειτουργίας αλγόριθμου μοριακής δυναμικής

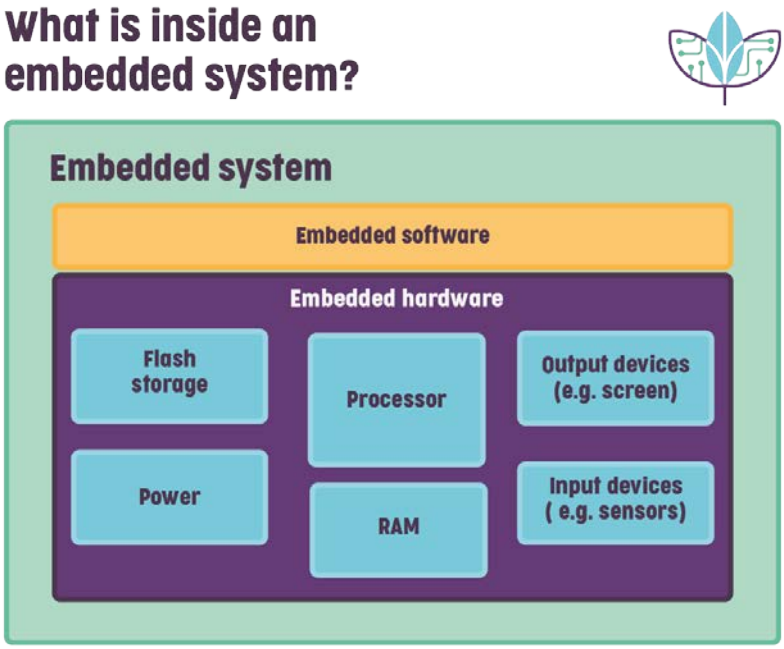
Σε έναν κώδικα Μοριακής Δυναμικής πρώτα δημιουργείται η γεωμετρία του συστήματος, ορίζοντας τις αρχικές θέσεις των ατόμων του συστήματος και στη συνέχεια ορίζεται η συμπεριφορά των διαφόρων στοιχείων που μπορεί να απαρτίζουν το σύστημα αυτό μέσω μιας συνάρτησης που περιγράφει τη δυναμική τους ενέργεια[14]. Έπειτα ορίζονται οι αρχικές και οριακές συνθήκες του συστήματος, όπως ταχύτητες και επιταχύνσεις και τέλος γίνεται επαναληπτικά ο προσδιορισμός της τροχιάς κάθε ατόμου του συστήματος με βάση τις δυνάμεις που του ασκούνται από τα γειτονικά του, μέσω επίλυσης του δεύτερου νόμου του Νεύτωνα για κάθε άτομο. Προκειμένου να υπολογιστούν οι δυνάμεις που ασκούνται σε κάθε άτομο, απαιτείται αρχικά ο προσδιορισμός των γειτονικών ατόμων κάθε ατόμου του συστήματος που επιτρέπεται να κινηθεί χωρίς περιορισμό [12]. Έπειτα, ανάλογα με τη μορφή της συνάρτησης της δυναμικής ενέργειας, υπολογίζονται οι δυνάμεις αλληλεπίδρασης για ζεύγη ατόμων ή περισσότερα από δύο άτομα και στο τέλος αθροίζονται οι δυνάμεις που ασκούνται σε κάθε άτομο από τα γειτονικά του [13]. Η διαδικασία αυτή αποτελεί το πιο χρονοβόρο κομμάτι του αλγορίθμου Μοριακής Δυναμικής και είναι ανάγκη να χρησιμοποιηθεί κάποια μέθοδος που θα επιταχύνει τη διαδικασία αυτή ώστε να είναι δυνατή η προσομοίωση συστημάτων με μεγάλο αριθμό ατόμων σε λογικό χρόνο.



Εικόνα 2: Γενική μορφή ενός αλγόριθμου μοριακής δυναμικής [15]

Ενσωματωμένα συστήματα

Τα ενσωματωμένα συστήματα έχουν γίνει ουσιαστικό μέρος της σύγχρονης τεχνολογίας. Ένα ενσωματωμένο σύστημα είναι ένα σύστημα υπολογιστή που έχει σχεδιαστεί για να εκτελεί μια συγκεκριμένη εργασία. Είναι ένας συνδυασμός υλικού και λογισμικού, ο οποίος είναι ενσωματωμένος σε ένα μεγαλύτερο σύστημα. Τα ενσωματωμένα συστήματα χρησιμοποιούνται σε διάφορους τομείς όπως αυτοκίνητα, ιατρικές συσκευές, ηλεκτρονικά είδη ευρείας κατανάλωσης και πολλά άλλα. Σε αυτό το δοκίμιο, θα συζητήσουμε τη σημασία των ενσωματωμένων συστημάτων, τα πλεονεκτήματα και τις εφαρμογές τους [4].



Εικόνα 3: Μέρη ενός ενσωματωμένου συστήματος [16]

Εξέλιξη των ενσωματωμένων συστημάτων

Τα ενσωματωμένα συστήματα είναι πανταχού παρόντα στη σύγχρονη τεχνολογία, αλλά η προέλευσή τους μπορεί να ανιχνευθεί στις πρώτες ημέρες της χρήσης υπολογιστών. Η ανάπτυξη των ενσωματωμένων συστημάτων ήταν μια σταδιακή εξέλιξη, που διαμορφώθηκε από τις προόδους στο υλικό και το λογισμικό υπολογιστών [17].

Τα πρώτα ενσωματωμένα συστήματα ήταν απλές ηλεκτρομηχανικές συσκευές όπως θερμοστάτες, ρολόγια και αριθμομηχανές. Αυτές οι συσκευές ελέγχονταν από μηχανικούς διακόπτες και δεν ήταν προγραμματιζόμενες. Το πρώτο ηλεκτρονικό ενσωματωμένο σύστημα εισήχθη τη δεκαετία του 1960 με την ανάπτυξη του υπολογιστή καθοδήγησης Apollo. Ο υπολογιστής σχεδιάστηκε για να πλοηγεί το διαστημόπλοιο Apollo στο φεγγάρι και πίσω [18].

Στη δεκαετία του 1970 εισήχθησαν οι μικροεπεξεργαστές, οι οποίοι επέτρεψαν τη δημιουργία προγραμματιζόμενων ενσωματωμένων συστημάτων. Ο Intel 4004, που παρουσιάστηκε το 1971,

ήταν ο πρώτος μικροεπεξεργαστής. Χρησιμοποιήθηκε στην αριθμομηχανή Busicom και άνοιξε το δρόμο για την ανάπτυξη των μικροελεγκτών. Αυτοί παρουσιάστηκαν στη δεκαετία του 1980 και οποίοι συνδύαζαν έναν μικροεπεξεργαστή με τη μνήμη, την είσοδο/έξοδο και άλλα περιφερειακά σε ένα μόνο τσιπ. Ο 8051, που παρουσιάστηκε από την Intel το 1980, ήταν ένας από τους πρώτους μικροελεγκτές και παραμένει δημοφιλής μέχρι σήμερα [16].

Στη δεκαετία του 1990, τα ενσωματωμένα συστήματα έγιναν πιο περίπλοκα, με την εισαγωγή των λειτουργικών συστημάτων σε πραγματικό χρόνο (RTOS) και των δυνατοτήτων δικτύωσης. Το Διαδίκτυο των Πραγμάτων (IoT) προέκυψε ως μια νέα ιδέα, η οποία συνέδεσε τα ενσωματωμένα συστήματα με το διαδίκτυο [16]. Στη δεκαετία του 2000, η ανάπτυξη ενσωματωμένων συστημάτων καθοδηγήθηκε από τις απαιτήσεις των φορητών υπολογιστών και την ανάγκη για ενεργειακά αποδοτικά συστήματα. Η ARM, μια βρετανική εταιρεία, αναδείχθηκε ως κυρίαρχος παίκτης στον τομέα των ενσωματωμένων συστημάτων με τους επεξεργαστές χαμηλής ισχύος και υψηλής απόδοσης.

Σήμερα, τα ενσωματωμένα συστήματα είναι πανταχού παρόντα, τροφοδοτώντας τα πάντα, από αυτοκίνητα και ιατρικές συσκευές μέχρι smartphone και έξυπνες οικιακές συσκευές. Ο τομέας των ενσωματωμένων συστημάτων εξελίσσεται ταχέως, με προόδους στη μηχανική μάθηση, την τεχνητή νοημοσύνη και άλλες τεχνολογίες. Τα ενσωματωμένα συστήματα γίνονται πιο έξυπνα και αυτόνομα, με την ικανότητα να μαθαίνουν από δεδομένα και να λαμβάνουν αποφάσεις χωρίς ανθρώπινη παρέμβαση. Η ανάπτυξη του IoT έχει δημιουργήσει νέες ευκαιρίες για ενσωματωμένα συστήματα, δίνοντάς τους τη δυνατότητα να συνδέονται με το cloud και άλλες συσκευές.

Εικόνα 4: Παραδείγματα χρήσης ενσωματωμένων συστημάτων [19]

Σημασία των ενσωματωμένων συστημάτων

Τα ενσωματωμένα συστήματα είναι ζωτικής σημασίας σε διάφορους κλάδους, επειδή μπορούν να εκτελέσουν μια συγκεκριμένη εργασία πιο αποτελεσματικά και αξιόπιστα από έναν υπολογιστή γενικής χρήσης. Στην αυτοκινητοβιομηχανία, τα ενσωματωμένα συστήματα χρησιμοποιούνται για τον έλεγχο του κινητήρα, των συστημάτων πέδησης και ανάρτησης. Σε ιατρικές συσκευές, τα ενσωματωμένα συστήματα χρησιμοποιούνται για την παρακολούθηση των ζωτικών σημείων των ασθενών, όπως ο καρδιακός ρυθμός, η αρτηριακή πίεση και τα επίπεδα οξυγόνου. Στα ηλεκτρονικά είδη ευρείας κατανάλωσης, τα ενσωματωμένα συστήματα χρησιμοποιούνται σε κινητά τηλέφωνα, τηλεοράσεις και έξυπνες οικιακές συσκευές [20].

Πλεονεκτήματα των ενσωματωμένων συστημάτων

Τα πλεονεκτήματα των ενσωματωμένων συστημάτων είναι πολλά. Έχουν σχεδιαστεί για να εκτελούν συγκεκριμένες εργασίες με ελάχιστη παρέμβαση του χρήστη, γεγονός που μειώνει τον φόρτο εργασίας του χρήστη. Τα ενσωματωμένα συστήματα έχουν χαμηλή κατανάλωση ενέργειας, γεγονός που τα καθιστά κατάλληλα για συσκευές που λειτουργούν με μπαταρίες. Είναι επίσης συμπαγείς, γεγονός που τα καθιστά κατάλληλα για συσκευές με περιορισμένο χώρο. Η δυνατότητα επεξεργασίας σε πραγματικό χρόνο των ενσωματωμένων συστημάτων τα καθιστά ιδανικά για κρίσιμες εφαρμογές όπως ιατρικές συσκευές και αεροδιαστημικά συστήματα [21].

Εφαρμογές ενσωματωμένων συστημάτων

Τα ενσωματωμένα συστήματα έχουν ένα ευρύ φάσμα εφαρμογών. Στην αυτοκινητοβιομηχανία, τα ενσωματωμένα συστήματα χρησιμοποιούνται σε μονάδες ελέγχου κινητήρα, συστήματα αντιμπλοκαρίσματος πέδησης και συστήματα αερόσακων. Στον ιατρικό τομέα, τα ενσωματωμένα συστήματα χρησιμοποιούνται σε βηματοδότες, αντλίες ινσουλίνης και συσκευές παρακολούθησης γλυκόζης αίματος. Στη βιομηχανία ηλεκτρονικών ειδών ευρείας κατανάλωσης, τα ενσωματωμένα συστήματα χρησιμοποιούνται σε κινητά τηλέφωνα, έξυπνα ρολόγια και έξυπνες οικιακές συσκευές. Στην αεροδιαστημική βιομηχανία, τα ενσωματωμένα συστήματα χρησιμοποιούνται σε συστήματα πλοήγησης και ελέγχου αεροσκαφών [21].

Ενσωματωμένα συστήματα στη μοριακή δυναμική

Μια βασική πρόκληση για την εκτέλεση προσομοιώσεων μοριακής δυναμικής ενσωματωμένων συστημάτων είναι το υπολογιστικό κόστος. Αυτά τα συστήματα είναι συνήθως μεγάλα και πολύπλοκα, και απαιτούν σημαντική ποσότητα υπολογιστικής ισχύος για να μοντελοποιήσουν με ακρίβεια τη συμπεριφορά τους [22]. Για την αντιμετώπιση αυτής της πρόκλησης, οι ερευνητές έχουν αναπτύξει έναν αριθμό εξειδικευμένων τεχνικών για την προσομοίωση

ενσωματωμένων συστημάτων, όπως μοντέλα με χονδρόκοκκο, τα οποία απλοποιούν την περιγραφή του συστήματος ομαδοποιώντας τα άτομα σε μεγαλύτερα σωματίδια [23].

Μια άλλη πρόκληση είναι η ακριβής περιγραφή των αλληλεπιδράσεων μεταξύ των ενσωματωμένων μορίων και του γύρω περιβάλλοντος. Σε πολλές περιπτώσεις, οι αλληλεπιδράσεις μεταξύ των μορίων είναι πολύπλοκες και δύσκολο να μοντελοποιηθούν με ακρίβεια [24]. Για να ξεπεράσουν αυτή την πρόκληση, οι ερευνητές έχουν αναπτύξει μια ποικιλία πεδίων δύναμης, τα οποία είναι μαθηματικά μοντέλα που περιγράφουν τις αλληλεπιδράσεις μεταξύ ατόμων και μορίων. Αυτά τα πεδία δύναμης βελτιώνονται διαρκώς και βελτιώνονται για την καλύτερη καταγραφή της συμπεριφοράς των ενσωματωμένων συστημάτων [25].

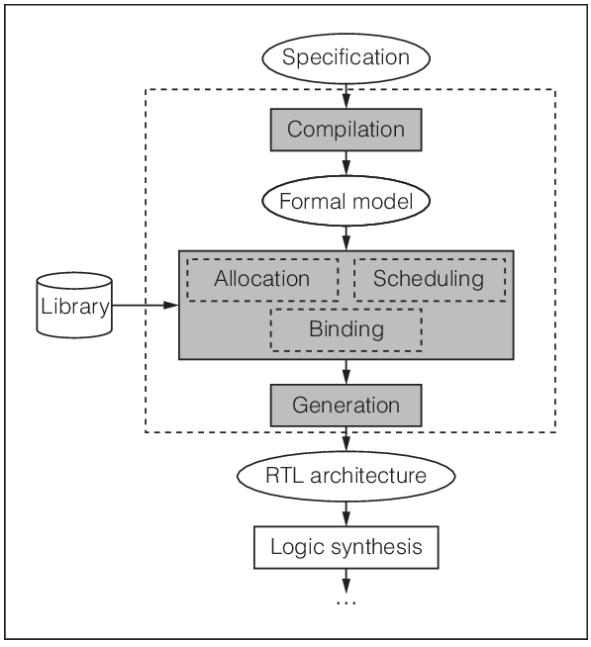
Παρά αυτές τις προκλήσεις, οι προσομοιώσεις MD των ενσωματωμένων συστημάτων έχουν ήδη αποφέρει πληθώρα νέων γνώσεων σχετικά με τη συμπεριφορά αυτών των πολύπλοκων συστημάτων [23]. Για παράδειγμα, προσομοιώσεις MD έχουν χρησιμοποιηθεί για τη μελέτη της συμπεριφοράς των μορίων φαρμάκου που είναι ενσωματωμένα σε διπλές στοιβάδες λιπιδίων, παρέχοντας νέες γνώσεις σχετικά με τους μηχανισμούς χορήγησης φαρμάκων. Ομοίως, προσομοιώσεις MD έχουν χρησιμοποιηθεί για τη μελέτη της συμπεριφοράς των πολυμερών που είναι ενσωματωμένα στα ηλεκτρόδια της μπαταρίας, ρίχνοντας φως στις πολύπλοκες ηλεκτροχημικές αντιδράσεις που συμβαίνουν κατά τη φόρτιση και την εκφόρτιση της μπαταρίας [6].

High Level Synthesis

Το High level Synthesis αποτελεί μια διαδικασία σχεδίασης υλικού, στην οποία ο σχεδιαστής εισάγει σε ένα εργαλείο έναν κώδικα σε μια γλώσσα προγραμματισμού υψηλού επιπέδου, και αυτό παράγει τον κατάλληλο κώδικα σε γλώσσα περιγραφής υλικού (Hardware Description Language, HDL) [26]. Η HLS διαδραματίζει κρίσιμο ρόλο στη σχεδίαση πολύπλοκων συστημάτων, ειδικά συστημάτων επεξεργασίας ψηφιακών σημάτων (DSP), επειδή μειώνει σημαντικά τον χρόνο σχεδιασμού, την πολυπλοκότητα του σχεδιασμού και το κόστος σχεδίασης. Αυτό το δοκίμιο θα συζητήσει την έννοια της σύνθεσης υψηλού επιπέδου, τα οφέλη, τις προκλήσεις και τις μελλοντικές προοπτικές της [27].

Πλεονεκτήματα της HLS

Η σύνθεση υψηλού επιπέδου προσφέρει πολλά οφέλη στους σχεδιαστές. Πρώτον, η HLS επιτρέπει την ταχεία εξερεύνηση σχεδιασμού και βελτιστοποίηση, παρέχοντας ένα ευέλικτο περιβάλλον σχεδιασμού. Οι σχεδιαστές μπορούν εύκολα να πειραματιστούν με διαφορετικούς αλγόριθμους, τύπους δεδομένων και αρχιτεκτονικές για να βελτιστοποιήσουν τον σχεδιασμό τους. Δεύτερον, η HLS παρέχει ένα υψηλό επίπεδο αφαίρεσης που επιτρέπει στους σχεδιαστές να επικεντρωθούν στη λειτουργική συμπεριφορά του συστήματος και όχι στις λεπτομέρειες υλοποίησης χαμηλού επιπέδου. Αυτό απλοποιεί τη διαδικασία σχεδιασμού και μειώνει τον χρόνο σχεδιασμού. Τρίτον, η HLS επιτρέπει την αυτόματη παραγωγή βελτιστοποιημένου υλικού, το οποίο έχει ως αποτέλεσμα βελτιωμένη απόδοση, μειωμένη κατανάλωση ενέργειας και μειωμένο κόστος σχεδίασης [28] [29].



Εικόνα 5: Διάγραμμα ροής για HLS [27]

Προκλήσεις της HLS

Παρά τα οφέλη της HLS, η σύνθεση υψηλού επιπέδου θέτει επίσης αρκετές προκλήσεις. Πρώτον, η διαδικασία αντιστοίχισης αφαιρέσεων υψηλού επιπέδου σε υλικό χαμηλού επιπέδου είναι μια πολύπλοκη εργασία που απαιτεί εξειδίκευση τόσο στον σχεδιασμό υλικού όσο και στο λογισμικό. Δεύτερον, η ποιότητα του υλικού που προκύπτει εξαρτάται σε μεγάλο βαθμό από την ποιότητα της περιγραφής υψηλού επιπέδου [30]. Τυχόν σφάλματα ή παραλείψεις στην περιγραφή υψηλού επιπέδου, μπορεί να οδηγήσουν σε σημαντικά σφάλματα σχεδιασμού. Τρίτον, τα εργαλεία που χρησιμοποιούνται για τη σύνθεση υψηλού επιπέδου, βρίσκονται ακόμη στα αρχικά στάδια ανάπτυξης και απαιτούν σημαντική βελτίωση όσον αφορά την ακρίβεια, την απόδοση και την ευκολία χρήσης [31].

Οι ιστορία της HLS

Η έννοια της σύνθεσης υψηλού επιπέδου όπως είναι γνωστή σήμερα, εμφανίστηκε στη δεκαετία του 1980, όταν οι ερευνητές άρχισαν να αναπτύσσουν εργαλεία για την αυτόματη μετάφραση των περιγραφών υψηλού επιπέδου συστημάτων σε υλοποιήσεις RTL [32]. Το 1996, ο αναπτύχθηκε ένα εργαλείο που ονομάζεται Language for Instruction Set Architecture (LISA) που μπορούσε να δημιουργήσει αυτόματα περιγραφές RTL μικροεπεξεργαστών από περιγραφές υψηλού επιπέδου [33], [34].

Στις αρχές της δεκαετίας του 1990, έγιναν διαθέσιμα αρκετά εμπορικά εργαλεία για σύνθεση υψηλού επιπέδου, συμπεριλαμβανομένου του Behavioral Compiler από το Cadence και του Behavioral Compiler από το Synopsys. Αυτά τα εργαλεία επέτρεψαν στους σχεδιαστές να δημιουργήσουν υλοποιήσεις RTL πολύπλοκων συστημάτων σε ένα κλάσμα του χρόνου που θα χρειαζόταν χρησιμοποιώντας παραδοσιακές μεθόδους σχεδιασμού [35].

Πρόοδος στη HLS

Από την εμφάνιση της σύνθεσης υψηλού επιπέδου στη δεκαετία του 1980, έχουν σημειωθεί σημαντικές πρόοδοι στην τεχνολογία. Στη δεκαετία του 2000, οι ερευνητές άρχισαν να εξερευνούν τη χρήση σύνθεσης υψηλού επιπέδου για συστήματα DSP, η οποία οδήγησε στην ανάπτυξη εργαλείων ειδικά σχεδιασμένων για τη σύνθεση DSP, όπως το Synphony C Compiler από τη Synopsys [36].

Τα τελευταία χρόνια, υπάρχει αυξανόμενο ενδιαφέρον για τη χρήση σύνθεσης υψηλού επιπέδου για τον σχεδιασμό συστημάτων μηχανικής μάθησης. Οι ερευνητές έχουν αναπτύξει εργαλεία που μπορούν να δημιουργήσουν αυτόματα περιγραφές RTL νευρωνικών δικτύων από περιγραφές υψηλού επιπέδου, όπως το Tensor Comprehension Compiler από το Facebook.

Οι μελλοντικές προοπτικές σύνθεσης υψηλού επιπέδου είναι ελπιδοφόρες. Πρώτον, η αυξανόμενη πολυπλοκότητα των συστημάτων και η ανάγκη για γρήγορο και αποτελεσματικό σχεδιασμό θα συνεχίσει να οδηγεί τη ζήτηση για σύνθεση υψηλού επιπέδου. Δεύτερον, η συνεχιζόμενη έρευνα στον τομέα της σύνθεσης υψηλού επιπέδου, οδηγεί στην ανάπτυξη πιο ακριβών και αποτελεσματικών εργαλείων. Τρίτον, η ενοποίηση της σύνθεσης υψηλού επιπέδου με άλλα εργαλεία σχεδιασμού, όπως η επαλήθευση και η βελτιστοποίηση, θα εξορθολογίσει περαιτέρω τη διαδικασία σχεδιασμού και θα βελτιώσει την ποιότητα του υλικού που προκύπτει.

Εργαλεία για HLS

Υπάρχουν πολλά εμπορικά εργαλεία HLS διαθέσιμα στην αγορά σήμερα, όπως το Vivado HLS από τη Xilinx, το Catapult HLS από την Mentor Graphics και το LegUp από τη LegUp Computing. Αυτά τα εργαλεία προσφέρουν ένα ευρύ φάσμα δυνατοτήτων και λειτουργιών, όπως υποστήριξη για διαφορετικές γλώσσες προγραμματισμού υψηλού επιπέδου, βελτιστοποίηση για απόδοση, κατανάλωση ενέργειας και περιοχή και αυτόματη δημιουργία κώδικα RTL.

Το Vivado HLS είναι ένα από τα πιο δημοφιλή εργαλεία HLS και χρησιμοποιείται ευρέως στη βιομηχανία. Υποστηρίζει γλώσσες προγραμματισμού C και C++ και προσφέρει μια σειρά επιλογών βελτιστοποίησης, συμπεριλαμβανομένης της διοχέτευσης, του ξετυλίγματος βρόχου και της κατάτμησης μνήμης. Το Vivado HLS διαθέτει επίσης μια φιλική προς το χρήστη διεπαφή που επιτρέπει στους σχεδιαστές να δοκιμάσουν εύκολα τη σχεδίασή τους και να προσαρμόσουν τις ρυθμίσεις βελτιστοποίησης για να ανταποκρίνονται στις απαιτήσεις σχεδιασμού τους.

Το Catapult HLS είναι ένα άλλο ευρέως χρησιμοποιούμενο εργαλείο HLS που υποστηρίζει γλώσσες προγραμματισμού C και SystemC. Προσφέρει προηγμένες επιλογές βελτιστοποίησης, συμπεριλαμβανομένης της δυνατότητας εκτέλεσης μετασχηματισμών βρόχου υψηλού επιπέδου και της δυνατότητας εκτέλεσης βελτιστοποιήσεων σε επίπεδο εργασιών. Το Catapult HLS υποστηρίζει επίσης τη συν-σχεδίαση υλικού-λογισμικού, επιτρέποντας στους σχεδιαστές να ενσωματώνουν στοιχεία λογισμικού με τον σχεδιασμό του υλικού τους.

Το LegUp είναι ένα νεότερο εργαλείο HLS που κερδίζει δημοτικότητα λόγω της υποστήριξής του για σχέδια υλικού ανοιχτού κώδικα. Το LegUp υποστηρίζει τη χρήση της υποδομής μεταγλωττιστή LLVM και μπορεί να δημιουργήσει αυτόματα κώδικα RTL από υψηλού επιπέδου LLVM IR (Intermediate Representations). Το LegUp προσφέρει επίσης μια σειρά επιλογών βελτιστοποίησης, συμπεριλαμβανομένης της διοχέτευσης βρόχου, του ξετυλίγματος βρόχου και της βελτιστοποίησης μνήμης.

Εκτός από τα εμπορικά εργαλεία HLS, υπάρχουν επίσης διαθέσιμα πολλά εργαλεία ανοιχτού κώδικα HLS, όπως η Βιβλιοθήκη Zynq Open-Source Vivado (ZOVIL) που βασίζεται στο LLVM και το εργαλείο Σύνθεσης υψηλού επιπέδου που βασίζεται σε Eclipse (Eclipse HLS). Αυτά τα εργαλεία χρησιμοποιούνται συχνά για ερευνητικούς και εκπαιδευτικούς σκοπούς, καθώς προσφέρουν μια εναλλακτική λύση χαμηλότερου κόστους σε σχέση με τα εμπορικά εργαλεία [29].

Το HLS εργαλείο CCC

Στην παρούσα εργασία για την πραγματοποίηση του HLS χρησιμοποιήθηκε το εργαλείο CCC (Cubed-C).

Το εργαλείο αυτό αποτελεί ένα εργαλείο βασισμένο σε Τυπικές μεθόδους (Formal Methods) και δίνει την δυνατότητα να παραχθεί Hardware το οποίο είναι ειδικού σκοπού, από κώδικα υψηλού επιπέδου [37].Με αυτό το τρόπο μειώνεται ο χρόνος κατά τάξη μεγέθους που χρειάζεται για την τελειοποίηση των προδιαγραφών, της σχεδίασης, της υλοποίησης και τελικά της επιβεβαίωσης ορθής λειτουργίας[37].

Το εργαλείο δέχεται ως είσοδο κώδικα υψηλού επιπέδου είτε σε γλώσσα ANSI C είτε σε γλώσσα ADA, και το μετατρέπει σε γλώσσα HDL (VHDL/Verilog)[37]–[39]. Η μετατροπή αυτή γίνεται σε δύο στάδια. Αρχικά, ο κώδικας C μετατρέπεται σε γλώσσα ADA, η οποία στην συνέχεια μετατρέπεται, λαμβάνοντας υπόψιν τις επιλογές του χρήστη, στην προτιμητέα γλώσσα HDL [40], [41].

Το CCC προσφέρεται δωρεάν, μέσω εργονομικής ιστοσελίδας για ερευνητική, πειραματική και ακαδημαϊκή αποκλειστικά χρήση.

Μετατροπή C σε ADA

Η μετατροπή του αρχικού κώδικα C σε γλώσσα ADA, πραγματοποιείται σε δύο στάδια, στο frontend, το οποίο παράγει μια ενδιάμεση μορφή, και στο backend το οποίο παράγει τον τελικό κώδικα σε ADA [39]. Το frontend του εργαλείου παρομοιάζει το frontend ενός C compiler. Περιέχει ένα λεξικό, συντακτικό και σημασιολογικό αναλυτή για τον εισαγόμενο κώδικα, ο οποίος αναλύει την ορθότητα και τη δυνατότητα του εργαλείου για να πραγματοποιήσει την μετατροπή [41]. Επιπλέον, πραγματοποιεί τις απαραίτητες βελτιστοποιήσεις έτσι ώστε το αποτέλεσμα να έχει την καλύτερη απόδοση. Πιο συγκεκριμένα, πραγματοποιεί τις ακόλουθες βελτιστοποιήσεις [41], [42]:

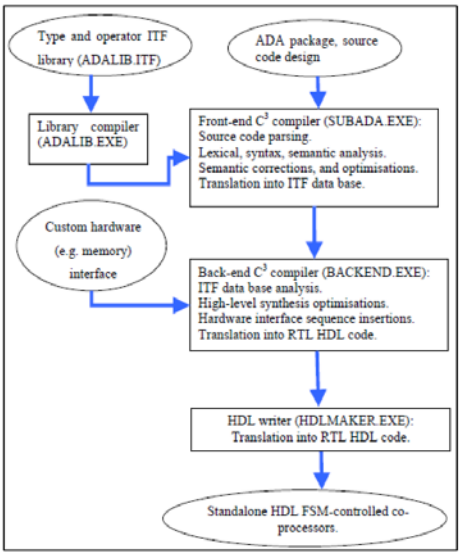
1. Loop unrolling
2. Loop pipelining
3. Function inlining
4. Code motion
5. Variable placement optimization
6. Complex structure variable optimization
7. Scalar and small structure variable optimization.

Επιπλέον, εν αντιθέσει με πολλά από τα εργαλεία HLS τα οποία υπάρχουν στην αγορά, υποστηρίζει σχεδόν το σύνολο της γλώσσας C, συμπεριλαμβανόμενων loop (for, while) αλλά και δεικτών. Στην περίπτωση των δεικτών, το εργαλείο τοποθετεί τις μεταβλητές αυτές σε εξωτερική μνήμη, δίνοντας με αυτό το τρόπο την δυνατότητα για δυναμικά μεγέθη μεταβλητών [42].

Ο αρχικός κώδικας μέσω του frontend compiler, ο οποίος πραγματοποιεί και τις βελτιστοποιήσεις, μετατρέπεται σε μια ενδιάμεση μορφή του τύπου Abstract syntax tree (AST). Κατόπιν το ίδιο αυτό AST μετατρέπεται σε κώδικα ADA χρησιμοποιώντας τα semantics της συγκεκριμένης γλώσσας χρησιμοποιώντας τον backend compiler [41].

Μετατροπή ADA σε HDL

Την ίδια λογική με την μετατροπή της γλώσσας C σε ADA, χρησιμοποιεί και το κύριο σκέλος του HLS εργαλείου. Χρησιμοποιεί έναν frontend compiler, ο οποίος μετατρέπει τον κώδικα σε μια ενδιάμεση μορφή, και ακολούθως ο backend compiler μετατρέπει αυτή την ενδιάμεση μορφή σε κώδικα HDL[37], [38], [43].



Εικόνα 6: Η μορφή και ροή του εργαλείου CCC [44]

Ο frontend compiler του εργαλείου CCC δέχεται ως είσοδο τον κώδικα σε γλώσσα ADA, και τον μετατρέπει σε μία ενδιάμεση μορφή, η οποία αποτελείται από λογικές προτάσεις, η οποία ονομάζεται Formal Intermediate Format (FIF). Κάθε απλή ή σύνθετη εντολή, αναλύεται και μεταφράζεται σε πράξεις τριών μεταβλητών (δυο εισόδου και μια εξόδου), οι οποίες αποτελούν το FIF. Επιπλέον, όπως και στην προηγούμενη μετατροπή, πραγματοποιεί την συντακτική και λεκτική ανάλυση του εισαγόμενου κώδικα για σφάλματα [37], [38]. Επιπλέον, δίνεται η δυνατότητα στον χρήστη να επεκτείνει τις δυνατότητες και τις μορφές που δέχεται ο frontend compiler, χρησιμοποιώντας επιπλέον βιβλιοθήκες, τις οποίες έχει συντάξει σε FIF, και ενσωματώνοντάς τες μέσω του compiler για τις βιβλιοθήκες [38]. Τέτοιες δυνατότητες αποτελούν, για παράδειγμα, η υποστήριξη αριθμητικών πράξεων κινητής υποδιαστολής, οι οποίες δεν είναι εξ ορισμού υποστηριζόμενες [45].

Ο backend compiler, δέχεται σαν είσοδο το FIF και παράγει τον τελικό κώδικα σε γλώσσα HDL. Για να το πραγματοποιήσει αυτό, χρησιμοποιεί κατηγορήματα (predicates) της γλώσσας Prolog με επίσημες λογικές συνθήκες (συνθήκες Horn) για την ανάλυση του FIF. Το τελικό αποτέλεσμα είναι ένα αρχείο σε γλώσσα HDL, το οποίο αποτελείται από FSM [37], [38].

Το τελικό αποτέλεσμα, βελτιστοποιείται χρησιμοποιώντας τον αλγόριθμο Parallel Abstract Resource Constrained Scheduling (PARCS). Ο αλγόριθμος αυτός δύναται να βελτιστοποιήσει σημαντικά το τελικό αποτέλεσμα, καθώς μειώνει τις καταστάσεις της παραγόμενης FSM. Η μείωση των καταστάσεων, πραγματοποιείται χρησιμοποιώντας formal τεχνικές όπως ο λογικός προγραμματισμός, οι σχέσεις RDF υποκείμενου κατηγορήματος αντικειμένου αλλά και επιβεβαίωσης μέσω σχημάτων XML [43]. Η βελτιστοποίηση υλοποιείται με την συγχώνευση των καταστάσεων οι οποίες δεν παρουσιάζουν εξαρτήσεις οπότε οι υπολογισμοί που γίνονται σε αυτές μπορούν να παραλληλοποίηθούν [37].

```
1. start with the initial schedule (inc. the
   special external port operations)
2. Current PARCS state <- 1
3. Get the 1st state and make it the current
   state
4. Get the next state
5. See if the operations of the next state do
   not have dependencies with the current
   state
6. If no dependencies then absorb the next
   state into the current PARCS state; If
   there are dependencies then absorb the
   current state into the PARCS state, store
   the PARCS state, PARCS state <- PARCS state
   + 1; make next state the current state
7. If next state = conditional then call the
   conditional (true/false branch) processing
   predicates, else continue
8. If there are more states to process then
   goto step 4, otherwise finalize the current
   PARCS state and terminate
```

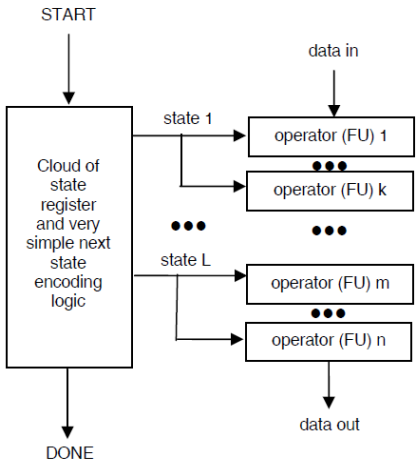
Εικόνα 7: Ψευδοκώδικας του Αλγόριθμου PARCS [38]

Επιπλέον, δίνεται στον χρήστη η παραμετροποίηση του παραγόμενου κώδικα μέσω μιας σειράς παραμέτρων. Οι παράμετροι αυτοί είναι [38], [40]:

1. Παραγόμενη HDL (Verilog, VHDL-93)
2. Παραγόμενη μικροαρχιτεκτονική (Μόνο FSM/FSM και Datapath)
3. Περιορισμοί σε πόρους για τον αλγόριθμο PARCS
4. Συναρτήσεις οι οποίες μεταφράζονται απευθείας σε HDL (π.χ. συνδυαστική λογική)
5. Αρχεία επιλογών επικοινωνίας με μνήμες (Διεπαφή μνήμης, Πρωτόκολλο, μέγεθος κ.α.)

Ακόμα, ο backend compiler δημιουργεί ένα αρχείο το οποίο περιέχει στατιστικά για την χρήση του, όπως ο χρόνος εκτέλεσης, το πλήθος των καταστάσεων της FSM του κάθε αρχείου καθώς και τις παραμέτρους με τις οποίες εκτελέστηκε [38].

Τέλος, για κάθε συνάρτηση του αρχικού κώδικα, δημιουργείται ένα αρχείο το οποίο αποτελείται από μια συγκεκριμένη διεπαφή για την βέλτιστη και εύκολη ενσωμάτωσή του σε ένα σύστημα. Η διεπαφή αυτή αποτελείται από τα σήματα : start, busy, done και results_read [38], [40].



Εικόνα 8: Παραγόμενη μικροαρχιτεκτονική του CCC (FSM και Datapath) [38]

Μέσω αυτού του τρόπου, καθώς όλα τα στάδια αποτελούν μέρος μιας formal μεθοδολογίας, γίνεται εύκολη η επιβεβαίωση ορθής λειτουργίας του κώδικα σε διαφορετικά επίπεδα. Ξεκινώντας από το υψηλότερο επίπεδο και επιβεβαιώνοντας την ορθή λειτουργία του αρχικού κώδικα, μπορεί ο χρήστης να είναι σίγουρος για την ορθή λειτουργία του τελικού κώδικα σε HDL, αποφεύγοντας τη χρονοβόρα επιβεβαίωση σε χαμηλό επίπεδο [38], [44].

Υλοποίηση

Ο αρχικός κώδικας σε γλώσσα προγραμματισμού του προγράμματος MATLAB, παρουσιάζεται στο αντίστοιχο εδάφιο του Παραρτήματος.

Αποτελείται από μία συνάρτηση η οποία αποτελεί μέρος ενός αλγόριθμου προσομοίωσης συστημάτων νανοκατεργασιών που αποτελούνται από ένα κοπτικό εργαλείο και ένα τεμάχιο. Σκοπός της συνάρτησης είναι να υπολογιστούν:

1. Οι δυνάμεις οι οποίες ασκούνται στο κάθε άτομο.
2. Η επιτάχυνση του κάθε ατόμου που προκύπτει από αυτές τις δυνάμεις.
3. Η τελική θέση του κάθε ατόμου για το επόμενο χρονικό βήμα μέσω διπλής ολοκλήρωσης.

Περιγραφή Λειτουργικότητας του Κώδικα

Η συνάρτηση υπολογίζει τις δυνάμεις μεταξύ των σωματιδίων χρησιμοποιώντας το δυναμικό Lennard-Jones και ενημερώνει το διάνυσμα επιτάχυνσης για κάθε σωματίδιο ανάλογα. Διασχίζει κάθε ζεύγος σωματιδίων και υπολογίζει την απόσταση μεταξύ τους, χρησιμοποιώντας την απόσταση για να προσδιορίσει εάν πρέπει να εφαρμοστεί το δυναμικό Lennard-Jones.

Η συνάρτηση ελέγχει επίσης τον τύπο των δύο σωματιδίων και εφαρμόζει διαφορετικές συναρτήσεις δυναμικού ανάλογα με τους τύπους τους. Στη συνέχεια, η συνάρτηση ενημερώνει το διάνυσμα επιτάχυνσης για κάθε σωματίδιο με βάση τις διασωματιδιακές δυνάμεις. Τέλος, η συνάρτηση ενημερώνει το διάνυσμα επιτάχυνσης για σωματίδια τύπου 1 και 4 με τις προηγούμενες τιμές επιτάχυνσής τους.

Το δυναμικό Lennard-Jones είναι ένα μαθηματικό μοντέλο που περιγράφει την αλληλεπίδραση μεταξύ ζευγών ουδέτερων ατόμων ή μορίων. Χρησιμοποιείται ευρέως σε μοριακές προσομοιώσεις και μοντελοποίηση ρευστών και πήρε το όνομά του από τον John Lennard-Jones, ο οποίος το εισήγαγε το 1924. Το δυναμικό δίνεται από τον τύπο:

$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

όπου r είναι η απόσταση μεταξύ των δύο σωματιδίων, ε είναι το βάθος του πηγαδιού δυναμικού και σ είναι η απόσταση στην οποία η δυναμική ενέργεια είναι μηδέν. Το δυναμικό έχει ελάχιστο στο $r = 2^{\left(\frac{1}{6}\right)}\sigma$, που αντιστοιχεί στην απόσταση ισορροπίας μεταξύ των δύο σωματιδίων, και έχει απωστική συνιστώσα σε μικρές αποστάσεις (r < σ) και ελκτική συνιστώσα σε μεγαλύτερες αποστάσεις (r > σ). Το δυναμικό Lennard-Jones χρησιμοποιείται ευρέως σε προσομοιώσεις μοριακής δυναμικής για τη μοντελοποίηση των αλληλεπιδράσεων μεταξύ ατόμων ή μορίων όπως και στην παρούσα εργασία.

Περιγραφή Εισόδων-Εξόδων Κώδικα

Η συνάρτηση παίρνει σαν ορίσματα δύο πολυδιάστατους πίνακες (raold, mr), έναν μονοδιάστατο (mtype) και μία μεταβλητή rCut.

Ο πίνακας raold περιέχει την τιμή της κάθε συνιστώσας της επιτάχυνσης για το κάθε άτομο στο τρέχον χρονικό βήμα. Το μέγεθος του πίνακα καθορίζεται από τις σταθερές NDIM και nMol. Η σταθερά NDIM, έχει την τιμή 3, και περιγράφει τον αριθμό των διαστάσεων στις οποίες γίνονται οι υπολογισμοί. Η σταθερά nMol, περιέχει τον αριθμό των ατόμων που περιέχει ο αρχικός πίνακας και για τα οποία θα γίνουν οι υπολογισμοί.

Ο πίνακας mr έχει τις ίδιο μέγεθος με τον πίνακα raold. Σε αυτόν περιέχεται η θέση του κάθε ατόμου μέσα στον χώρο.

Ο τρίτος πίνακας εισόδου περιγράφει το είδος κάθε ατόμου, δηλαδή αν ανήκει στο κοπτικό εργαλείο ή σε κάποια ζώνη ατόμων του τεμαχίου.

Τέλος, σαν είσοδος εισάγεται και η απόσταση αποκοπής (rCut). Καθώς λόγω της μορφής της συνάρτησης της δυναμικής ενέργειας, όταν η απόσταση μεταξύ δύο ατόμων είναι μεγάλη, η ενέργεια είναι ουσιαστικά μηδέν. Η απόσταση αποκοπής χρησιμοποιείται για να αποφεύγονται περιττοί υπολογισμοί δυνάμεων μεταξύ ατόμων που βρίσκονται σε μεγάλες αποστάσεις, διότι προηγείται κατάλληλος έλεγχος των αποστάσεων μεταξύ των ατόμων πριν γίνει ο υπολογισμός του δυναμικού Lennard-Jones.

Το αποτέλεσμα της συνάρτησης είναι ο πίνακας ra, ο οποίος περιέχει την τιμή της συνιστώσας επιτάχυνσης του κάθε ατόμου στο επόμενο χρονικό βήμα, και κατά τη διάρκεια της προσομοίωσης ανατροφοδοτείται ως ο πίνακας raold του επόμενου χρονικού βήματος.

Περιγραφή Λειτουργικότητας

Αρχικά, αρχικοποιείται ο πίνακας της επιτάχυνσης και ένας πίνακας που αποθηκεύει τις αποστάσεις μεταξύ των ατόμων χρησιμοποιώντας μηδενικά.

Στη συνέχεια, εκτελείται το κύριο μέρος της συνάρτησης το οποίο υπολογίζει τις δυνάμεις που ασκούνται σε κάθε άτομο από τα άτομα με τα οποία αλληλοεπιδρά.

Το κύριο μέρος της συνάρτησης αποτελείται από ένα διπλό loop το οποίο τρέχει για κάθε άτομο που βρίσκεται μέσα στον πίνακα. Αρχικά, υπολογίζεται η απόσταση του ατόμου αναφοράς (j1) με τα άλλα άτομα του συστήματος (j2), ελέγχοντας ένα ζεύγος ατόμων τη φορά. Γνωρίζοντας τις συντεταγμένες του κάθε ατόμου, ο υπολογισμός της απόστασης δίνεται από τον τύπο:

$$d = \left(\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}\right)$$

Εάν αυτή η απόσταση είναι μικρότερη από την απόσταση αποκοπής (rCut) τότε εκτελείται ο υπολογισμός δυνάμεων, αλλιώς δεν γίνεται υπολογισμός, γιατί το δυναμικό Lennard-Jones

μεταξύ αυτών των ατόμων θα είναι μηδενικό. Η σύγκριση αυτή γίνεται πάνω στα τετράγωνα της απόστασης αποκοπής και της πραγματικής απόστασης (rr).

Για τις περιπτώσεις που η απόσταση είναι μικρότερη από την απόσταση αποκοπής, γίνεται έλεγχος για το εάν το ζεύγος ατόμων είναι του ίδιου υλικού ή διαφορετικών υλικών. Εάν είναι από διαφορετικό υλικό, και το ένα από τα δύο ανήκει στο κοπτικό εργαλείο (mtype=1), τότε ο υπολογισμός του δυναμικού Lennard-Jones γίνεται με τον τύπο:

$$F = 0{,}55207 * \left(-e^{-(3{,}42*(d-2{,}4965))}\right) + e^{-(1{,}71*(d-2{,}4965))}$$

Ενώ στις περιπτώσεις που κανένα από τα δύο δεν ανήκει στο κοπτικό εργαλείο τότε το δυναμικό Lennard-Jones υπολογίζεται από τον τύπο:

$$F = 0{,}93186504 * \left(e^{-(2{,}7176*(d-2{,}78))}\right) - e^{-(1{,}3588*(d-2{,}78))}$$

Εάν και τα δύο άτομα ανήκουν στο κοπτικό εργαλείο τότε το δυναμικό Lennard-Jones το οποίο ασκείται μεταξύ τους θεωρείται 0.

Στη συνέχεια και αφού έχει υπολογιστεί το δυναμικό Lennard-Jones που ασκείται μεταξύ των δύο ατόμων, γίνεται υπολογισμός της επιτάχυνσης που προκύπτει από το δυναμικό Lennard-Jones, σε κάθε διάσταση. Έτσι, για το άτομο αναφοράς, η νέα επιτάχυνση είναι σε κάθε διάσταση i είναι:

$$\alpha_i' = \alpha_i + \frac{F * d_i}{r}$$

Ενώ αντίστοιχα του ατόμου με το οποίο συσχετίζεται:

$$\alpha_i' = \alpha_i - \frac{F * d_i}{r}$$

Έτσι, με το πέρας του διπλού loop, Έχουμε στον πίνακα ra την υπολογισμένη επιτάχυνση για κάθε άτομο, σε κάθε διάσταση.

Στο τέλος της συνάρτησης, γίνεται ο έλεγχος εάν το άτομο ανήκει σε οριακή συνθήκη ή αποτελεί άτομο του κοπτικού εργαλείου. Εάν κάποιο άτομο ανήκει σε οριακή συνθήκη ή στο κοπτικό εργαλείο η επιτάχυνσή του ορίζεται ίση με την επιτάχυνση στο προηγούμενο χρονικό βήμα. Αυτό συμβαίνει γιατί τα άτομα που έχουν το ρόλο της οριακής συνθήκης τύπου πάκτωσης (mtype=4), προκειμένου να μη μετακινείται το τεμάχιο που κατεργαζόμαστε στο χώρο, πρέπει να παραμείνουν ακίνητα χωρίς να επηρεαστούν από τις δυνάμεις που τους ασκούνται. Όσον αφορά τα άτομα που ανήκουν στο κοπτικό εργαλείο (mtype=1), επειδή το κοπτικό εργαλείο θεωρείται απαραμόρφωτο στερεό σώμα δε χρειάζεται να υπολογίζονται για τα άτομά του νέες τιμές επιτάχυνσης σε κάθε βήμα.

Μετατροπή κώδικα από Matlab σε C

Για να καταστεί δυνατή η εισαγωγή του κώδικά της συνάρτησης στο εργαλείο HLS, αρχικά χρειαζόταν η μετατροπή του στη γλώσσα προγραμματισμού C. Λόγω του ότι ο κώδικας που θα

εισαγόταν στο εργαλείο HLS αποτελούσε μια συνάρτηση, για τις ανάγκες του ελέγχου σωστής λειτουργίας του, έπρεπε να φτιαχτούν οι βοηθητικές συναρτήσεις καθώς και μια κύρια η οποία θα καλούσε τον υπό μετατροπή κώδικα.

Οι βοηθητικές συναρτήσεις που υλοποιήθηκαν για να καταστεί δυνατός ο έλεγχος, αποτελούταν από μια συνάρτηση η οποία λαμβάνει τα δεδομένα εισόδου σε μορφή αρχείου txt, μια συνάρτηση η οποία τυπώνει τα αποτελέσματα σε μορφή αρχείου txt, και μια συνάρτηση η οποία τυπώνει τους πίνακες που δημιουργήθηκαν στην κονσόλα.

Δεσμεύτηκε μνήμη για 2 πίνακες διαστάσεων NDIM*nMol, οι οποίοι θα αντιπροσώπευαν τον πίνακα επιταχύνσεων του προηγούμενου χρονικού βήματος(raold) και τον πίνακα συντεταγμένων του κάθε ατόμου (coordinates), καθώς και ένας πίνακας διάστασης nMol, ο οποίος θα περιέχει τον τύπο των ατόμων (mtype).

Στη συνέχεια, καλείται η συνάρτηση loadFile, η οποία παίρνει σαν ορίσματα εισόδου το αρχείο που περιέχει τις απαραίτητες πληροφορίες καθώς και τους πίνακες coordinates και mtype οι οποίοι θα αρχικοποιηθούν με τις τιμές οι οποίες θα διαβαστούν από το αρχείο. Το αρχείο εισόδου περιέχει σε κάθε σειρά τις απαραίτητες πληροφορίες για κάθε άτομο με τη μορφή:

id coordinate_x coordinate_y coordinate_z mtype

έχοντας μόνο ένα κενό ανάμεσα στις τιμές.

Στη συνάρτηση loadFile, το αρχείο διαβάζεται χρησιμοποιώντας τη συνάρτηση fscanf με όρισμα διαβάσματος μια συμβολοσειρά. Η συνάρτηση αυτή έχει την ιδιότητα να διαβάζει χαρακτήρες μέχρι να βρει κενό ή αλλαγή γραμμής. Με αυτό τον τρόπο και αυξάνοντας μια μεταβλητή i σε κάθε συμβολοσειρά που διαβάζεται, γνωρίζουμε ότι κάθε πρώτο διάβασμα πρέπει να αγνοηθεί, το $2^{ο}$, $3^{ο}$ και $4^{ο}$ αποτελούν τις συντεταγμένες ενώ το $5^{ο}$ αποτελεί τον τύπο του ατόμου. Για να μπορέσουμε να μετατρέψουμε τη συμβολοσειρά, η οποία ανακτήθηκε από το αρχείο, σε αριθμό χρησιμοποιείται η συνάρτηση atof η οποία λαμβάνει σαν όρισμα μια συμβολοσειρά και επιστρέφει σε μορφή κινητής υποδιαστολής τον αριθμό που αναπαρίσταται. Για τον τύπο του ατόμου χρησιμοποιείται η συνάρτηση atoi η οποία αντίστοιχα λαμβάνει συμβολοσειρά σαν είσοδο αλλά επιστρέφει ακέραιο αριθμό.

Έπειτα, υλοποιήθηκε μια συνάρτηση η οποία γράφει τα αποτελέσματα των επιταχύνσεων σε ένα αρχείο. Η συνάρτηση αυτή λαμβάνει σαν είσοδο τον πίνακα που περιέχει τις επιταχύνσεις και τις γράφει σε ένα αρχείο, έτσι ώστε να μπορέσουμε να επιβεβαιώσουμε τα αποτελέσματα της συνάρτησης.

Στη συνέχεια, για τη μετατροπή του αρχικού κώδικα έγινε μεταφορά γραμμή προς γραμμή του αρχικού κώδικα της Matlab.

Έγινε επιβεβαίωση της σωστής λειτουργίας χρησιμοποιώντας τις βοηθητικές συναρτήσεις οι οποίες παρουσιάστηκαν.

Έπειτα, απομονώθηκε η συνάρτηση η οποία θα μετατρεπόταν μέσω HLS, και τροφοδοτήθηκε σαν είσοδος στο εργαλείο το οποίο βρίσκεται στο:

CUSTOM COPROCESSOR ON THE WEB (teiwm.gr) (http://ccc.kastoria.teiwm.gr/ccc/)

Το εργαλείο δεν μπόρεσε να μετατρέψει τον κώδικα που του τροφοδοτήθηκε χρησιμοποιώντας HLS, καθώς μια ιδιαιτερότητα του εργαλείου που χρησιμοποιήθηκε για HLS είναι η απουσία μονάδων κινητής υποδιαστολής.

Για αυτό τον λόγο, όλοι οι αριθμοί που θα χρησιμοποιούταν στη συνάρτηση που θέλαμε να μετατρέψουμε, έπρεπε να είναι αριθμοί σταθερής υποδιαστολής (fixed point). Έτσι οι αριθμοί, κατά τη διάρκεια της φόρτωσης από το αρχείο και πριν αποθηκευτούν στον πίνακα συντεταγμένων μετατρεπόταν από floating point σε fixed point. Για τη μετατροπή αυτή, δημιουργήθηκε η συνάρτηση fixed_to_float η οποία λαμβάνει σαν είσοδο έναν float αριθμό και επιστρέφει έναν αριθμό fixed point ο οποίος μπορεί να αποθηκευτεί στη μορφή ενός integer.

Έχοντας γεμίσει τον πίνακα των συντεταγμένων με την κατάλληλη είσοδο, στο επόμενο βήμα, δεδομένου ότι οι πράξεις με fixed point αριθμούς δεν είναι ίδιες με αυτές για integers ή floating point, δημιουργήσαμε τις πράξεις τις οποίες χρειαζόμασταν σε νέες συναρτήσεις της C. Οι συναρτήσεις αυτές έπρεπε να είναι synthesizable από το εργαλείο HLS και να παράγουν σωστά αποτελέσματα.

Υλοποιήθηκαν για αριθμούς fixed point οι ακόλουθες πράξεις:

- Πολλαπλασιασμός
- Διαίρεση
- Τετραγωνική Ρίζα
- Δύναμη
- Exponential

Για το exponential χρησιμοποιήθηκε μια προσέγγιση με σειρά taylor ανεπτυγμένη σε 14 όρους, ενώ για την τετραγωνική ρίζα χρησιμοποιήθηκε μια μέθοδος η οποία βασίζεται στη διαίρεση ψηφίο προς ψηφίο, υπολογίζοντας σε κάθε iteration τα Most Significant bits. Για τον υπολογισμό της δύναμης χρησιμοποιήθηκαν η συνάρτηση διαίρεσης και πολλαπλασιασμού.

Μετά την υλοποίηση των συναρτήσεων για τις πράξεις, έγινε επιβεβαίωση των αποτελεσμάτων. Τα αποτελέσματα με πράξεις σταθερής υποδιαστολής δεν ήταν τα αναμενόμενα, οπότε για κάποιες πράξεις οι οποίες ήταν σταθερές, υπολογίστηκε η τιμή τους και οι συναρτήσεις υπολογισμού του δυναμικού Lennard-Jones τροποποιήθηκαν με σκοπό να περιέχουν τις λιγότερες δυνατές μεταβαλλόμενες πράξεις.

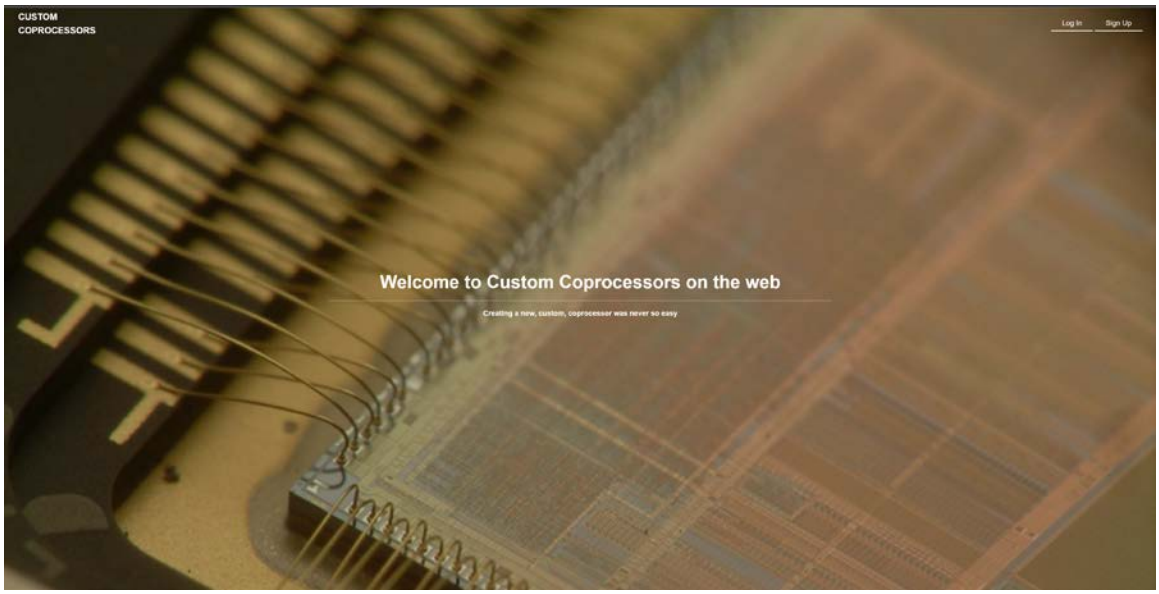
Στη συνέχεια, έγινε ξανά επιβεβαίωση των αποτελεσμάτων και σύγκριση με αυτών του αρχικού κώδικα σε Matlab, όπου παρατηρήθηκε μερική ταύτιση των αποτελεσμάτων. Για τη διόρθωση της μερικής ταύτισης, τροποποιήθηκε η ακρίβεια των πράξεων και πιο συγκεκριμένα της συνάρτησης υπολογισμού του exponential. Στη σύγκριση μετά και τις τελευταίες τροποποιήσεις, παρατηρήθηκε ότι τα αποτελέσματα ταυτιζόταν σε επίπεδο $5^{ου}$ δεκαδικού ψηφίου, το οποίο κρίθηκε αποδεκτό.

Τέλος, οι βοηθητικές συναρτήσεις για τις πράξεις καθώς και ο κώδικας της συνάρτησης υπολογισμού των επιταχύνσεων, δόθηκε ως είσοδος στο εργαλείο HLS.

Χρήση του Εργαλείου HLS

Το εργαλείο HLS που χρησιμοποιήθηκε βρίσκεται στον ιστότοπο : http://ccc.kastoria.teiwm.gr/ccc/

Στην αρχική σελίδα ο χρήστης εισάγει τα στοιχεία του λογαριασμού του για να εισέλθει στο περιβάλλον του εργαλείου.



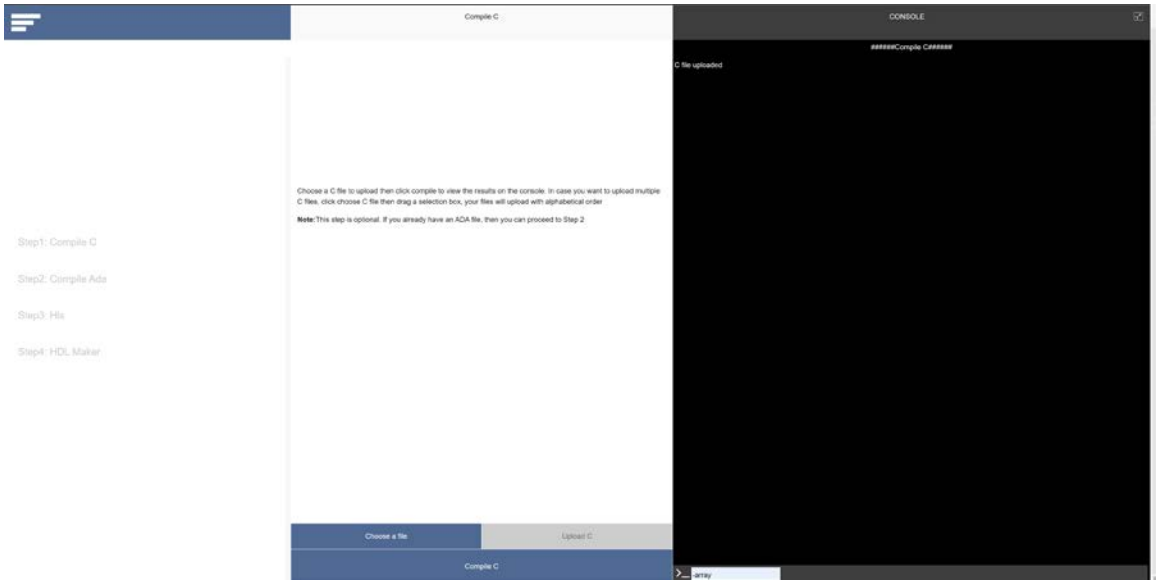
Εικόνα 9: Αρχική σελίδα εργαλείου HLS

Έπειτα ο χρήστης βρίσκεται στο περιβάλλον στο οποίο μπορεί να εισάγει τον κώδικα C για τη μετατροπή του. Μπορεί να παρατηρηθεί ότι η ροή της μετατροπής είναι από κώδικα C σε κώδικα ADA και ακολούθως σε κώδικα HDL. Στη σελίδα αυτήν, μετά το ανέβασμα του κώδικα μας δίνεται η δυνατότητα να εισαχθούν παράμετροι για τη μετατροπή του σε ADA. Δεδομένου ότι η συνάρτηση προς μετατροπή χρησιμοποιεί arrays από αριθμούς, ήταν απαραίτητη η παράμετρος «-array» για τη μετατροπή. Στην επόμενη σελίδα, δεν χρειάστηκε η εισαγωγή κάποιου αρχείου ADA ή αρχείου μνήμης οπότε έγινε αμέσως το compilation του παραχθέντος κώδικα σε ADA. Η επόμενη ιστοσελίδα η οποία εμφανίζεται περιέχει τα αποτελέσματα του compilation και δίνει τη δυνατότητα εισαγωγής επιπλέον παραμέτρων και αρχείων για το εργαλείο HLS, που πραγματοποιεί τη μετατροπή του κώδικα ADA σε HDL. Στη σελίδα αυτή δεν εισήχθησαν κάποια αρχεία ή παράμετροι, οπότε καλέστηκε το εργαλείο HLS. Το αποτέλεσμα του παρουσιάζεται στην Εικόνα 5, όπου μπορεί να παρατηρηθεί επιπλέον ότι για την παραγωγή του τελικού κώδικα παρουσιάζονται οι δυνατότητες:

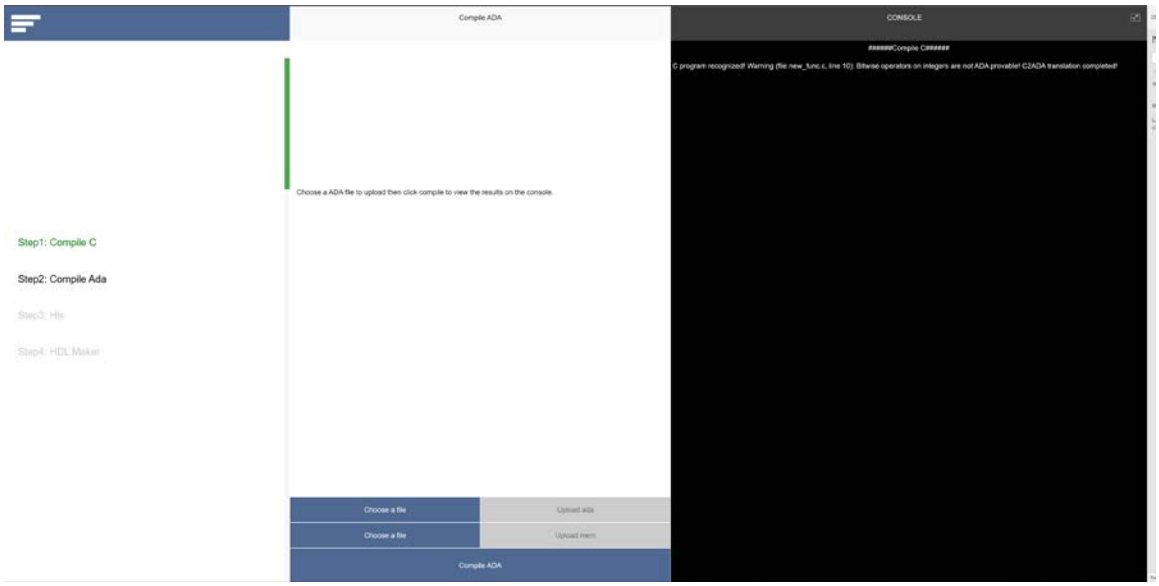
- Επιλογή προσομοιωτή με ακρίβεια κύκλων ρολογιού (Ναι/Όχι),
- Επιλογή παραγόμενης γλώσσας HDL (VHDL/Verilog)
- Επιλογή τρόπου υλοποίησης (Massively-parallel(mp)/Datapath+FSM(dp))

- Επιλογή τρόπου reset (Σύγχρονο/Ασύγχρονο)
- Έλεγχος του παραγόμενου αρχείου με το πρόσθετο Checkstyle για την υλοποίηση χρησιμοποιώντας προκαθορισμένους κανόνες

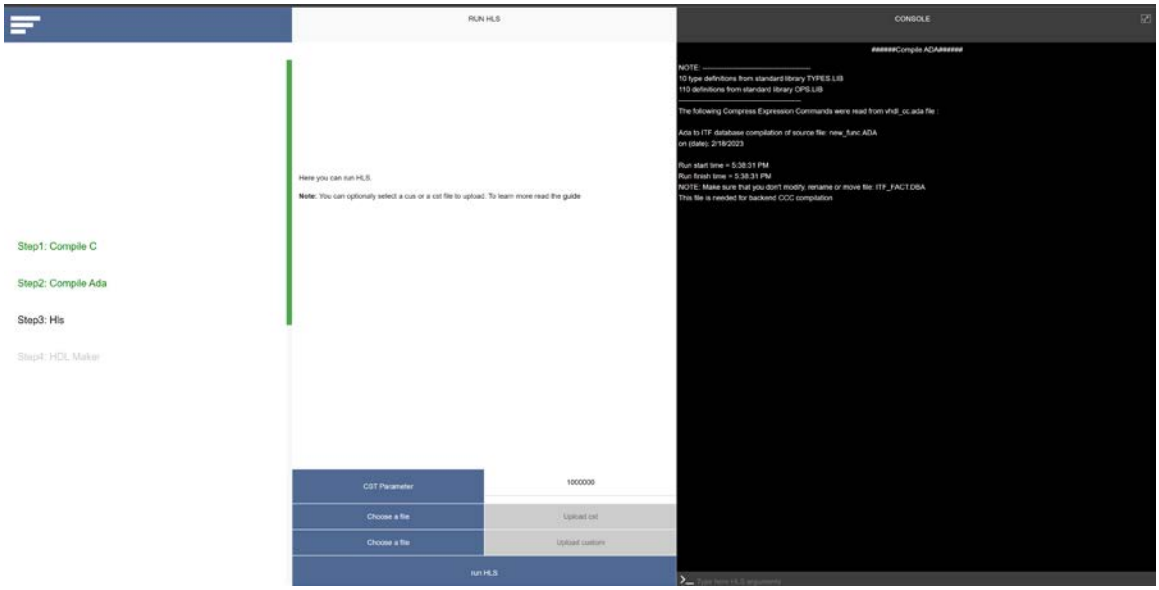
Μετά την επιλογή των κατάλληλων παραμέτρων γίνεται η παραγωγή του κώδικα στη γλώσσα HDL, με τις παραμέτρους που επιλέχθηκαν.



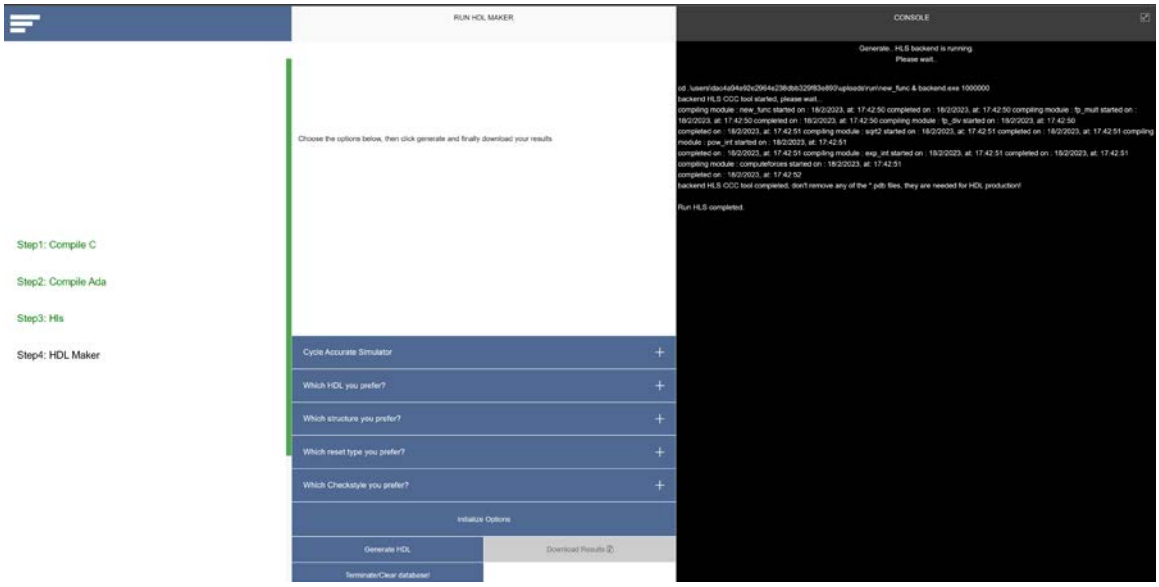
Εικόνα 10: Σελίδα εισαγωγής αρχείου C



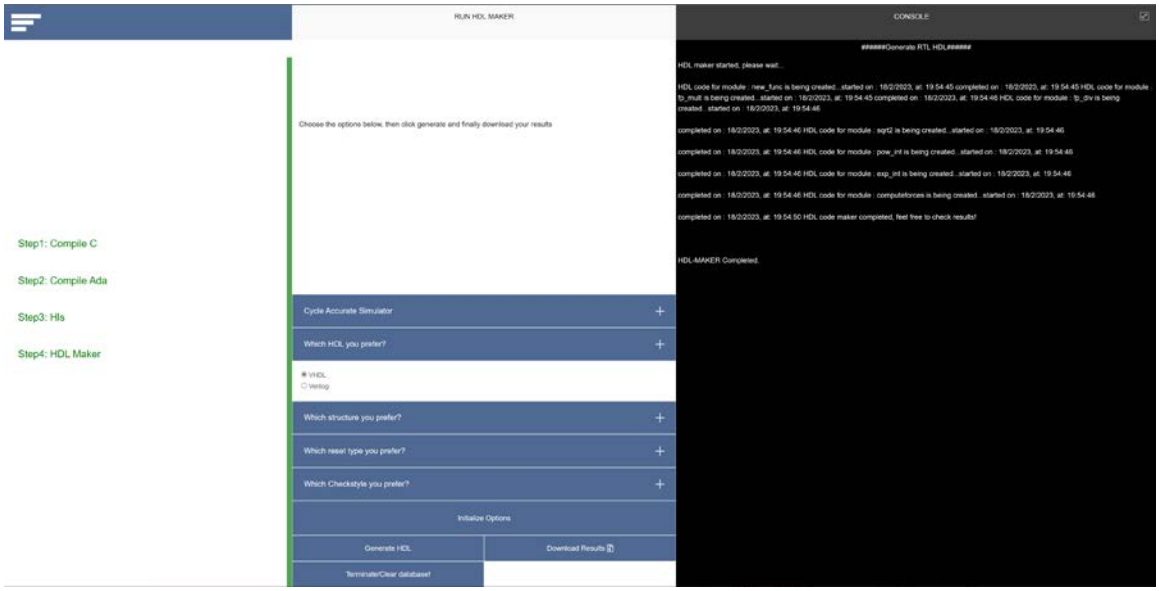
Εικόνα 11: Σελίδα μετά τη μετατροπή του αρχείου C



Εικόνα 12: Αποτελέσματα compilation ADA και σελίδα HLS



Εικόνα 13: Αποτελέσματα εργαλείου HLS



Εικόνα 14: Αποτελέσματα εργαλείου HDLMaker

Αποτελέσματα χρήσης του εργαλείου

Χρησιμοποιώντας τις παραμέτρους:

- Cycle Accurate Simulator: No
- HDL: VHDL
- Structure: Datapath
- Reset Type: Async
- Check style: Check

Τα αποτελέσματα χρήσης του εργαλείου ήταν 11 module, τα οποία ήταν:

1. Sqrt2.vhd
2. Sqrt2_parcs.vhd
3. Pow_int.vhd
4. Pow_int_parcs.vhd
5. Fp_mult.vhd
6. Fp_mult_parcs.vhd
7. Fp_div.vhd
8. Fp_div_parcs.vhd
9. Exp_int.vhd
10. Exp_int_parcs.vhd
11. Computeforces.vhd
12. Computeforces_parcs.vhd

Για κάθε συνάρτηση του αρχείου C, το εργαλείο παρήγαγε ένα αρχείο χωρίς την βελτιστοποίηση PARCS και ένα με την βελτιστοποίηση το οποίο αποτελούταν από λιγότερες καταστάσεις στην παραγόμενη FSM.

Κάθε αρχείο αποτελούταν από μια FSM η οποία αποτελούσε το control logic και κάποια datapath operations για τη λογική.

Επιλέγοντας τη γλώσσα Verilog, το εργαλείο δεν μπορούσε να βγάλει αποτελέσματα. Το ίδιο αποτέλεσμα υπήρχε και όταν προσπαθούσαμε να παράγουμε μέσω του εργαλείου τον Cycle Accurate Simulator.

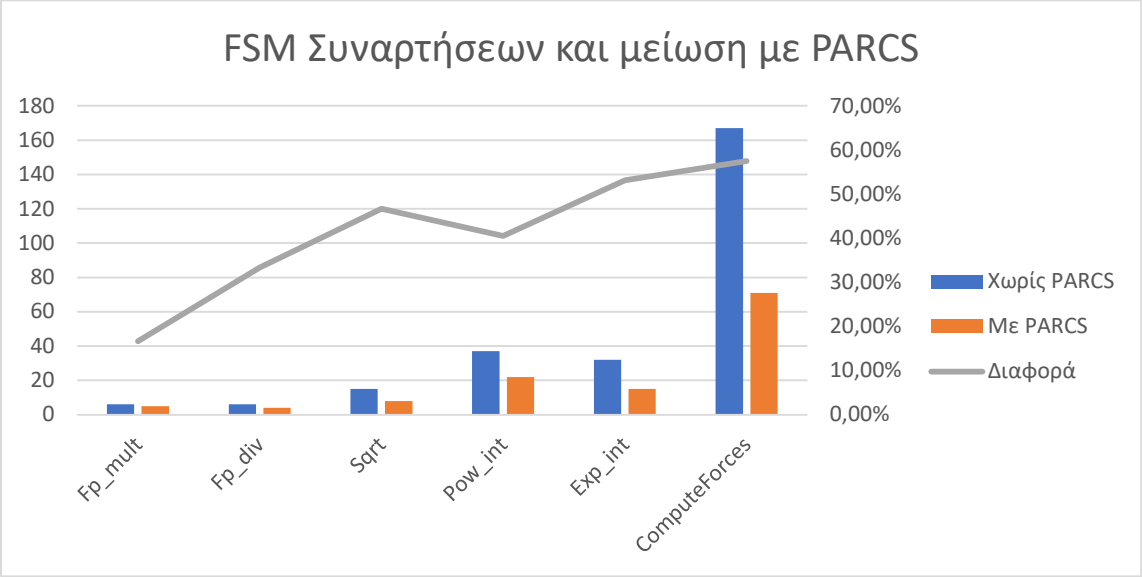
Αξιολόγηση παραγόμενου κώδικα

Από τα αρχεία τα οποία παρήχθησαν από το εργαλείο CCC, μπορεί να πραγματοποιηθεί αξιολόγηση της ταχύτητας τους.

Στον Πίνακα 1 καθώς και στην Εικόνα 15 μπορούμε να παρατηρήσουμε τα ακόλουθα αποτελέσματα όσον αφορά το πλήθος των καταστάσεων τις οποίες έχει το κάθε αρχείο, από την αναφορά του backend compiler.

Πίνακας 1: FSM Συναρτήσεων και μείωση με PARCS

Συνάρτηση	Χωρίς PARCS	Με PARCS	Διαφορά
Fp_mult	6	5	-16,67%
Fp_div	6	4	-33,33%
Sqrt	15	8	-46,67%
Pow_int	37	22	-40,54%
Exp_int	32	15	-53,13%
Compute_forces	167	71	-57,49%



Εικόνα 15: FSM Συναρτήσεων και μείωση με PARCS

Από τα αποτελέσματα μπορούμε να παρατηρήσουμε τον χρονισμό της κάθε συνάρτησης, ο οποίος αντιστοιχεί στις καταστάσεις τις οποίες έχει αλλά και τις κλήσεις σε άλλες συναρτήσεις.

Χωρίς την βελτιστοποίηση PARCS, ο συνολικός χρόνος που χρειάζεται η συνάρτηση για να εκτελεστεί είναι 167 κύκλοι, καθώς ενσωματώνει τις υπόλοιπες συναρτήσεις μέσα στο computeforces. Χρησιμοποιώντας την βελτιστοποίηση PARCS οι χρόνοι εκτέλεσης της κάθε συνάρτησης είναι:

Πίνακας 2: Χρόνος εκτέλεσης αλγορίθμου

Συνάρτηση	Κλήσεις σε άλλες συναρτήσεις	Παράλληλες συναρτήσεις	Συνολικός χρόνος
fp_div	-	-	5 κύκλοι
fp_mult	-	-	4 κύκλοι
Sqrt	-	-	8 κύκλοι
pow_int	2 x fp_div, 1 x fp_mult	-	22+2*5+3=35 κύκλοι
exp_int	1 x pow_int, 1 x fp_div	-	15+35+5=55 κύκλοι
computeforces	1 x sqrt, 4 x fp_mult, 6 x pow_int, 2 x exp_int, 6 x fp_div	3 pow_int (state 20), sqrt – pow_int (state 22), mult – pow_int – exp_int – 2 x fp_div (state 35), pow_int – fp_mult – exp_int – 2 x fp_div (state 41), fp_mult – fp_div (state 47), fp_mult – fp_div (state 51)	71+35+35+35+35+5+5=221 κύκλοι

Ο κώδικας που παράγεται είναι εύκολο να διαβαστεί και να γίνει κατανοητός καθώς περιέχει σχόλια, ενώ και τα ονόματα των εισόδων, εξόδων και μεταβλητών είναι περιγραφικά. Η παραγόμενη FSM αλλά και το Datapath είναι κατανοητή.

Ως προς την αξιολόγηση της, δεδομένης της παραλληλίας των πράξεων αλλά και της υλοποίησης με FSM, κρίνεται ότι ο κώδικας είναι πολύ αποτελεσματικός για υλοποίηση σε FPGA, καθώς περιλαμβάνει ισορροπία σε registers και λογική.

Συμπεράσματα

Οι αλγόριθμοι μοριακής δυναμικής απαιτούν αρκετούς πόρους, τόσο σε υπολογιστική ισχύ, όσο και σε μνήμη. Χρησιμοποιώντας την επιτάχυνση που μπορούν να προσφέρουν τα ενσωματωμένα συστήματα, τέτοιοι αλγόριθμοι μπορούν να παρουσιάσουν αποτελέσματα σε πολύ μικρότερο χρόνο. Τα ενσωματωμένα συστήματα όμως, απαιτούν αρκετό χρόνο έτσι ώστε να αναπτυχθούν, να επιβεβαιωθεί η ορθή τους λειτουργία και τελικά να μπορέσουν να παράγουν τα αναμενόμενα αποτελέσματα.

Έτσι, κρίνεται αναγκαίος ένας τρόπος ώστε να επιταχυνθεί και αυτή η λειτουργία. Χρησιμοποιώντας την τεχνολογία HLS, γίνεται πιο γρήγορη η ανάπτυξη τέτοιων αλγορίθμων, με την μικρότερη δυνατή επιβεβαίωση της ορθής λειτουργίας. Επιπλέον, η ανάπτυξη τέτοιων συστημάτων με τη χρήση HLS, μπορεί να γίνει από άτομα τα οποία δεν έχουν γνώσεις λογικής σχεδίασης, παρά μόνο προγραμματισμού υψηλού επιπέδου.

Στην παρούσα εργασία, παρατηρήθηκε ότι η ανάπτυξη πολύπλοκων αλγορίθμων, ως IP, κατέστη δυνατή γνωρίζοντας μόνο τις ιδιαιτερότητες του προγράμματος HLS που χρησιμοποιήθηκε. Έτσι, έγινε πολύ γρήγορη η μεταφορά ενός αλγόριθμου μοριακής δυναμικής από γλώσσα MATLAB, σε γλώσσα προγραμματισμού C, και στη συνέχεια σε γλώσσα HDL.

Η επιβεβαίωση ορθής λειτουργίας έγινε στη γλώσσα προγραμματισμού C, και έχοντας εμπιστοσύνη στο πρόγραμμα HLS, μπορεί να καταστεί σχεδόν σίγουρο, ότι ο αλγόριθμος θα συμπεριφέρεται εξίσου καλά και στο Υλικό.

Βιβλιογραφία

[1] I. Parnassos *et al.*, "A programming model and runtime system for approximation-aware heterogeneous computing," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–4. doi: 10.23919/FPL.2017.8056774.

[2] "Cover Image, Volume 37, Issue 27: Journal of Computational Chemistry," *J. Comput. Chem.*, vol. 37, no. 27, pp. i–i, Oct. 2016, doi: 10.1002/jcc.24490.

[3] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *J. Comput. Phys.*, vol. 117, no. 1, pp. 1–19, Mar. 1995, doi: 10.1006/jcph.1995.1039.

[4] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: a cyber-physical systems approach*, Second edition. Cambridge, Massachuetts: MIT Press, 2017.

[5] A. Correia, R. Andrade, and P. Moreira, "Energy-efficient molecular dynamics using FPGAs," *2019 IEEE Int. Symp. Circuits Syst. ISCAS*, pp. 1–5, 2019.

[6] M. A. Khan, M. Chiu, and M. C. Herbordt, "FPGA-Accelerated Molecular Dynamics," in *High-Performance Computing Using FPGAs*, W. Vanderbauwhede and K. Benkrid, Eds., New York, NY: Springer New York, 2013, pp. 105–135. doi: 10.1007/978-1-4614-1791-0_4.

[7] "Homepage - CSMS." https://csms.ethz.ch/ (accessed Feb. 22, 2023).

[8] D. Frenkel and B. Smit, *Understanding molecular simulation: from algorithms to applications*, 2nd ed. in Computational science series, no. 1. San Diego: Academic Press, 2002.

[9] D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, 2nd ed. Cambridge University Press, 2004. doi: 10.1017/CBO9780511816581.

[10] M. P. Allen, D. J. Tildesley, and D. J. Tildesley, *Computer simulation of liquids*, Reprinted. in Oxford science publications. Oxford: Clarendon Pr, 2009.

[11] A. Hospital, J. R. Goñi, M. Orozco, and J. L. Gelpi, "Molecular dynamics simulations: advances and applications," *Adv. Appl. Bioinforma. Chem.*, vol. 8, pp. 37–47, Nov. 2015, doi: 10.2147/AABC.S70333.

[12] A. R. Leach, *Molecular modelling: principles and applications*, 2nd ed. Harlow, England ; New York: Prentice Hall, 2001.

[13] T. Schlick, *Molecular modeling and simulation: an interdisciplinary guide*, 2nd ed. in Interdisciplinary applied mathematics, no. v. 21. New York: Springer, 2010.

[14] K. Karandashev and J. Vanicek, "A combined on-the-fly/interpolation procedure for evaluating energy values needed in molecular simulations," *J. Chem. Phys.*, vol. 151, no. 17, p. 174116, Nov. 2019, doi: 10.1063/1.5124469.

[15] "Molecular dynamics," *Wikipedia*. Jan. 29, 2023. Accessed: Feb. 22, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Molecular_dynamics&oldid=1136322033

[16] V. Creative and Bluefruit, "A brief history of embedded operating systems," *Bluefruit Software*, Sep. 06, 2022. https://www.bluefruit.co.uk/training/a-brief-history-of-embedded-operating-systems/ (accessed Feb. 22, 2023).

[17] "Introduction to Embedded Systems." http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C1_EmbeddedSystemsShapeTheWorld.htm (accessed Feb. 22, 2023).

[18] "What is an Embedded System?," *IoT Agenda*. https://www.techtarget.com/iotagenda/definition/embedded-system (accessed Feb. 22, 2023).

[19] "Embedded Systems Trends and Technologies," *ARC Advisory Group*, Nov. 30, 2018. https://www.arcweb.com/blog/embedded-systems-trends-technologies (accessed Feb. 22, 2023).

[20] Administrator, “Embedded System and Its Real Time Applications,” *Electronics Hub*, Oct. 12, 2017. https://www.electronicshub.org/embedded-system-real-time-applications/ (accessed Feb. 22, 2023).
[21] L. Micco, F. Vargas, and P. Fierens, “A Literature Review on Embedded Systems,” *IEEE Lat. Am. Trans.*, vol. 18, pp. 188–205, Feb. 2020, doi: 10.1109/TLA.2020.9085271.
[22] C. Yang *et al.*, “Fully Integrated On-FPGA Molecular Dynamics Simulations.” arXiv, May 13, 2019. doi: 10.48550/arXiv.1905.05359.
[23] M. Schaffner and L. Benini, “On the Feasibility of FPGA Acceleration of Molecular Dynamics Simulations.” arXiv, Aug. 08, 2018. doi: 10.48550/arXiv.1808.04201.
[24] J. Cong, Z. Fang, H. Kianinejad, and P. Wei, “Revisiting FPGA Acceleration of Molecular Dynamics Simulation with Dynamic Data Flow Behavior in High-Level Synthesis.” arXiv, Nov. 10, 2016. doi: 10.48550/arXiv.1611.04474.
[25] D. Jones *et al.*, “Accelerators for Classical Molecular Dynamics Simulations of Biomolecules,” *J. Chem. Theory Comput.*, vol. 18, no. 7, pp. 4047–4069, Jul. 2022, doi: 10.1021/acs.jctc.1c01214.
[26] Ó. Lucía, E. Monmasson, D. Navarro, L. A. Barragán, I. Urriza, and J. I. Artigas, “Chapter 29 - Modern Control Architectures and Implementation,” in *Control of Power Electronic Converters and Systems*, F. Blaabjerg, Ed., Academic Press, 2018, pp. 477–502. doi: 10.1016/B978-0-12-816136-4.00030-0.
[27] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, “An Introduction to High-Level Synthesis,” *IEEE Des. Test Comput.*, vol. 26, no. 4, pp. 8–17, Jul. 2009, doi: 10.1109/MDT.2009.69.
[28] D. D. Gajski and L. Ramachandran, “Introduction to high-level synthesis,” *IEEE Des. Test Comput.*, vol. 11, no. 4, pp. 44–54, 1994, doi: 10.1109/54.329454.
[29] “Benefits of High-Level Synthesis • Vitis High-Level Synthesis User Guide (UG1399) • Reader • Documentation Portal.” https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Benefits-of-High-Level-Synthesis (accessed Feb. 22, 2023).
[30] K. Rupnow, Y. Liang, Y. Li, and D. Chen, “A study of high-level synthesis: Promises and challenges,” Oct. 2011, pp. 1102–1105. doi: 10.1109/ASICON.2011.6157401.
[31] N. Pundir, F. Farahmandi, and M. Tehranipoor, “Secure High-Level Synthesis: Challenges and Solutions,” in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, Apr. 2021, pp. 164–171. doi: 10.1109/ISQED51717.2021.9424365.
[32] “Behavioral Synthesis,” *Semiconductor Engineering*. https://semiengineering.com/knowledge_centers/eda-design/methodologies-and-flows/behavioral-synthesis/ (accessed Feb. 22, 2023).
[33] R. Camposano, R. A. Bergamaschi, C. E. Haynes, M. Payer, and S. M. Wu, “The IBM High-Level Synthesis System,” in *High-Level VLSI Synthesis*, R. Camposano and W. Wolf, Eds., in The Springer International Series in Engineering and Computer Science. Boston, MA: Springer US, 1991, pp. 79–104. doi: 10.1007/978-1-4615-3966-7_4.
[34] V. Zivojnovic, S. Pees, C. Schläger, and H. Meyr, “LISA bridges gaps in high-tech languages,” *Electron. Eng. Times*, Oct. 1996.
[35] B. Bailey, “The Evolution Of High-Level Synthesis,” *Semiconductor Engineering*, Aug. 27, 2020. https://semiengineering.com/the-evolution-of-high-level-synthesis/ (accessed Feb. 22, 2023).
[36] “Synopsys Introduces Synphony High Level Synthesis.” https://news.synopsys.com/index.php?s=20295&item=123096 (accessed Feb. 22, 2023).

[37] M. F. Dossis, T. Themelis, and L. Markopoulos, “A Web Service to Generate Program Coprocessors,” in *2009 Fourth International Workshop on Semantic Media Adaptation and Personalization*, Dec. 2009, pp. 121–128. doi: 10.1109/SMAP.2009.15.
[38] M. F. Dossis, “Automatic Generation of Massively Parallel Hardware from Control-Intensive Sequential Programs,” in *2010 IEEE Computer Society Annual Symposium on VLSI*, Jul. 2010, pp. 98–103. doi: 10.1109/ISVLSI.2010.40.
[39] M. Dossis and G. Dimitriou, “Are HLS Tools Healthy? The C-Cubed Project,” *Eng. Technol. Appl. Sci. Res.*, vol. 5, Jan. 2015, doi: 10.5281/zenodo.16989.
[40] M. Dossis, “Custom Options for Custom Processors,” in *Proceedings of the 1st International Virtual Scientific Conference*, Slovakia, Zilina, 2013.
[41] G. Dimitriou, G. Chatzianastasiou, A. Tsakyridis, G. Stamoulis, and M. Dossis, “Source-Level Compiler Optimizations for High-Level Synthesis,” in *Proceedings of the SouthEast European Design Automation, Computer Engineering, Computer Networks and Social Media Conference*, in SEEDA-CECNSM ’16. New York, NY, USA: Association for Computing Machinery, Sep. 2016, pp. 11–18. doi: 10.1145/2984393.2984406.
[42] G. Dimitriou, M. Dossis, and G. Stamoulis, “Global and Pointer Variables in High-Level Synthesis,” in *2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, Sep. 2020, pp. 1–6. doi: 10.1109/SEEDA-CECNSM49515.2020.9221802.
[43] M. Dossis, “High-Level Synthesis: A Practical Perspective,” *Adv. Robot. Autom.*, vol. 03, no. 03, 2013, doi: 10.4172/2168-9695.1000123.
[44] D. Amanatidis and M. Dossis, “High Level Synthesis of CART,” in *2019 4th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, Sep. 2019, pp. 1–5. doi: 10.1109/SEEDA-CECNSM.2019.8908494.
[45] M. F. Dossis and G. D. V. Hados, “Numerical Block High-Level Synthesis,” 2014. Accessed: Mar. 22, 2023. [Online]. Available: https://www.semanticscholar.org/paper/Numerical-Block-High-Level-Synthesis-Dossis-Hados/f1653ec8d0fa13f0d553e1dbcf497d59d7b2c866

Παράρτημα

Αρχικός Κώδικας σε γλώσσα Matlab

```
function [ra] = ComputeForces(rCut,tool,part_b,raold,mr,mtype)

NDIM=10000;nMol=3;

partbound=part_b;
rrCut = rCut^2;
ac = zeros(NDIM, nMol);
ra = zeros(NDIM, nMol);
dr = zeros(NDIM,1);

uSum = 0;
virSum = 0;

for j1 = 1:(nMol - 1)
    for j2 = (j1 + 1):(nMol - 1)
```

```
        for k=1:NDIM
           dr(k,1) = mr(k,j1) - mr(k,j2);
        end
        rr = (sqrt(dr(1,1)^2+dr(2,1)^2+dr(3,1)^2))^2;
        rri=1/rr;
        rri3=rri*rri*rri;
        r = sqrt(dr(1,1)^2+dr(2,1)^2+dr(3,1)^2);

        if (rr < rrCut)

          if ((mtype(1,j1) == 1) && (mtype(1,j2) ~= 1)) || ((mtype(j1) ~=1) && (mtype(j2) == 1))

             fcVal = 0.55207 * (-exp(-3.42 * (r - 2.4965)) + exp(-1.71 * (r -2.4965)));

             if mtype(1,j1) == 1
                ac(:,j1) = ac(:,j1) + fcVal .* dr(:,1) / r;
             end
             if mtype(1,j2) == 1
                ac(:,j2) = ac(:,j2) + fcVal .* dr(:,1) / r;
             end

          elseif ((mtype(1,j1) ~= 1) && (mtype(1,j2) ~= 1))
             fcVal = 0.93186504 * (exp(-2.7176 * (r - 2.78)) -exp(-1.3588 * (r - 2.78)));

          else
              fcVal = 0;
          end
          for m=1:NDIM
             ra(m,j1) = ra(m,j1) + fcVal .* dr(m,1) / r;
             ra(m,j2) = ra(m,j2) - fcVal .* dr(m,1) / r;
          end
          uSum = uSum + 4 * rri3 * (rri3 - 1) + 1;
          virSum = virSum + fcVal * rr;
         end
    end
end
for m=1:NDIM
  for mm=1:nMol
     if (partbound(mm,1)==1)
       ra(m, mm) = raold(m, mm);
     end
     if (tool(mm,1)==1)
        ra(m, mm) = raold(m, mm);
     end
  end
end
```

Τελικός Κώδικας σε γλώσσα C

```
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int   fp_mult        ( int, int);
int   fp_div         ( int, int);
int   sqrt2          ( int );
int   pow_int        ( int, int);
int   exp_int        ( int );
int*  ComputeForces  ( int*, int*, int*);
int   float_to_fixed ( float );
```

```
float fixed_to_float ( int );
int   loadFile       ( char*, int*, int*);
int   writeOutFile   ( int*);
void  printTables    ( int*, int* );


#define DEBUG 0

#define nMol 210    // O arithmos twn grammwn tou arxeiou
#define NDIM 3      // O arithmos twn syntetagmenwn pou periexei to arxeio
#define rCut  10   // To threshold gia ton ypologismo twn epitaxynsewn

#define FP_SHIFT 16          // oi theseis pou prepei na metakinithoun ta dedomena gia na exoume
tosa dekadika psifia ( den exei dokimastei me allo notation)
#define FP_BASE  (1<<FP_SHIFT)  // to 1 se 15.16 notation


/*************************************************************
*  H synartisi pou kanei ton pollaplasiasmo 2 fixed point   *
*  arithmwn. Xrisimopoioume long long giati to apotelesma   *
*  tou pollaplasiasmou dyo arithmwn 32 bit einai 64 bit     *
*************************************************************/
int fp_mult(int numa, int numb){

  long long int mult = (((long long)numa*(long long)numb)>>FP_SHIFT);

  int mult_int = ((int)(mult&(0x00000000ffffffff)));
  return mult_int;

}

/*************************************************************
*  H synartisi pou kanei tink diairesi 2 fixed point        *
*  arithmwn. Xrisimopoioume long long gia megaliteri        *
*  akriveia tou apootelesmatos. shiftaroume stin arxi ta    *
*  dedomena kata toses theseis oso to megethos twn dedomenwn*
*  mas kai meta toses theseis aristera oso i diafora me ta  *
*  dekadika mas psifia                                      *
*************************************************************/
int fp_div(int numa,int numb){

  long long div = (((((long long)numa)<<32))/((long long)numb))>>(32-FP_SHIFT);

  int div_int = ((int)(div&(0x00000000ffffffff)));

  return div;
}

/*************************************************************
*  H synartisi pou ypologizei tin tetragoniki riza fixed    *
*  point arithmou se 16.15 notation                         *
*  (https://github.com/chmike/fpsqrt/blob/master/fpsqrt.c)  *
*  Greatly loses accuracy for numbers > 5000                *
*************************************************************/
int sqrt2(int number){

    int t, q, b, r;
    r = number;
    b = 0x40000000;
    q = 0;
    while( b > 0x40 )
    {
        t = q + b;
        if( r >= t )
        {
            r -= t;
            q = t + b; // equivalent to q += 2*b
        }
```

```
        r <<= 1;
        b >>= 1;
    }
    q >>= 8;
    return q;
}


/*************************************************************
*  H synartisi pou kanei ton ypologismo dynamis se fixed    *
*  point arithmetic. Dexetai mono akeraies dynameis         *
*************************************************************/
int pow_int ( int num , int pow){

  int number;
  int divider = 1;

  if ( pow == 0){
    number = FP_BASE;
  } else if (pow>0) {
    number  = num;
  } else {
    number = fp_div(1*FP_BASE,num);
  }

  while( pow>1 || pow<-1 ) {
    if (pow>0) { number = fp_mult(number,num); pow--; }
    else if ( pow<0 ) { number = fp_div(number,num); pow++; }
#if DEBUG
    printf("pow : %d number : %d, num : %d\n",pow, number,num);
#endif
  }

  return number;

}

/*****************************************************************
*  H synartisi pou kanei ton ypologismo tou e^x xrisimopoiontas *
*  tin seira Maclaurin gia ton ypologismo e^x = sum(x^i/i!)     *
*  opou oso megalytero ton i toso megaliteri i akriveia         *
*****************************************************************/
int exp_int(int pow){
  int num = FP_BASE;
  int fact = 1,neg=0;

  if (pow < (-905412) ){ // e^(-13.8155) < 10^(-6)
    return 0;
  }

  if (pow < 0) {
    neg=1;
    pow = -pow;
  }
  for(int i=1 ;i<14;i++){
    fact = fact*i;
    num = num + pow_int(pow,i)/fact;
  }
  if (neg==1){
    num=fp_div(FP_BASE,num);
  }
#if DEBUG
  printf("e^%f = %f\n",pow,num);
#endif
  return num;
}

/*************************************************************
```

```
*  H synartisi pou kanei ton metasximatismo twn dedomenwn   *
* Metaferthike apo tin MATLAB se C kwdika                   *
* To raold egine 0 kathws den yparxoun arxikes times kai    *
* to dianysma tou ra ftiaxnetai edw mesa gia na pernaei apo *
* to c2ada                                                  *
************************************************************/
int* ComputeForces(int * raold, int* mr, int* mtype)
{

  int i,j,k,m;

  int fcVal=0;
  int dr[NDIM];
  int r,uSum = 0,virSum=0;

   int *ra;            // Ftiaxnoume tis metavlites tis opoies tha valoume to raold kai to ra
[ telika apotelesmata ] tou arxeiou

   ra     = (int*)malloc((nMol*NDIM)*sizeof(int)); // Desmevoume mnimi gia tis nMol seires tou
pinaka pou perilamvanoun tis telikes epitaxynseis

  for(i=0;i<nMol;i++){
    for(j=0;j<NDIM;j++) {
      ra[i+ j*210]=0;          // Epeidi to raold xrisimopoieitai gia ton ypologismo twn ra prepei
na einai arxikopoiimeno. Opote edw to arxikopoioume sto 0
    }
  }

  for (i = 0; i < nMol-1; i++) {

    for(j=i+1;j<nMol-1;j++){

      for (k = 0; k < NDIM; k++) {
           dr[k]  =  *(mr+i+k*210)  -  *(mr+j+k*210);  // Ypologismos tis diaforas metaxy twn
syntetagmenwn
      }

       r  =  sqrt2(pow_int(dr[0],2)  +  pow_int(dr[1],2)  +  pow_int(dr[2],2)); // Ypologismos tis
apostasis twn dyo simeiwn ston trisdiastato xwro

      int test =0;

       if  (pow_int(r,2)  <  pow_int(rCut*FP_BASE,2))  {  // An  to  tetragwno  tis  apostasis  einai
megalitero tou tetragwnou tou threshold[rCut]

        if (((*(mtype+i) == 1) && (*(mtype+j) != 1)) || ((*(mtype+i)!= 1) && (*(mtype+j) == 1)))
{  // An kapoio apo ta dyo einai typou 1 kai to allo den einai
        test =1;
          int e_171 = exp_int(fp_mult(112066,r)); // e^(1.71*r)
            fcVal  =  fp_div(2588082,e_171)-  pow_int(fp_div(3481242,e_171),2);  // Ypologismos  tou
fcVal
                    }    else    if    ((*(mtype+i)    !=    1)    &&    (*(mtype+j)    !=    1))
{                                  // An kai ta dyo einai typou 1
        test =2;
          int e_135 = exp_int(fp_mult(89050,r)); // e^(1.3588*r)
             fcVal  =  pow_int(fp_div(2764958,e_135),2)  -  fp_div(2669099,e_135);                //
Ypologismos tou fcVal
                                                                                  }                 else
{                                                                                          // An kanena
den einai typou 1
                                                                                    fcVal               =
0;                                                                                      // Tote to fcVal
einai 0
        }

        for (k = 0; k < NDIM; k++) {
           ra[i+k*210] += fp_mult(fcVal , fp_div(dr[k],r));  // i epitaxynsi tou prwtou simeiou
ayxanetai kata Fc*(xi-xj)/r
```

```
          ra[j+k*210] -= fp_mult(fcVal , fp_div(dr[k],r));  // i epitaxynsi tou deyterou simeiou
meionetai kata Fc*(xi-xj)/r
        }

      }
    }
  }

#if DEBUG>0
  printf("uSum : %f , virSum : %f",uSum,virSum);
#endif

  for (m = 0; m < NDIM; m++) {
    for (j = 0; j < nMol; j++) {
      if (*(mtype+j) == 4 || *(mtype+j) == 1) {
          ra[j+m*nMol] = raold[j+m*nMol];         // An o typos einai 4 h 1 tote i epitaxynsi
pairneitin timi raold
      }
    }
  }

  return ra;

}

/*************************************************************
*     Metatropi floating point to fixed point arithmetic    *
*************************************************************/
int float_to_fixed ( float f ){

  float ffx = (f * (float)FP_BASE); // Gia na metatrepsoume ena floating point se fixed point
arkei na to pollaplasiasoume me tin akriveia (p.x. 1 -> 1*2^16)
                                    // Ara tha exoume me ayto to tropo sta 16 MSB to akeraio
meros  kai sta 15 LSB to dekadiko
  return (int)ffx;

}

/*************************************************************
*     Metatropi fixed point to floating point arithmetic    *
*************************************************************/
float fixed_to_float (int ffx ){

  float f = (((float)ffx))/((float)FP_BASE); // Gia na metatrepsoume ena fixed point se floatinf
point diairoume me tin vasi (2^16)
  return f;
}

/*************************************************************
*  H synartisi pou pairnei ta dedomena apo to arxio eisodou *
*************************************************************/
int loadFile(char* fileName,int * coordinates,int * mtype){

  FILE* file;       // H metavliti me ton pointer tou arxeiou
  char line[200];   // Thewroume oti kathe grammi tha exei megisto mikos 200 xaraktires. An einai
megaliteri allazoume ayto to noumero giati alliws tha xasoume dedomena
  int i=0;

  file = fopen(fileName,"r"); // Anoigoume to arxeio gia na to diavasoume

  if ( file == 0 ){           // An den mporoume na anoixoume to arxeio
    printf("error");          // Typwnoume minima lathous
    return -1;                // kai epistrefoume stin kyria sinartisi me lathos
  }


  while (fscanf(file,"%s",line)!=EOF){     // Diavazoume gia kathe simvoloseira tou arxeio . to
fscanf me dedomeno %s diavazei mexri na vrei keno i allagi grammis
```

```
// Kathe dedomeno tou arxeiou eisodou tha exei ta stoixeia grammi*5+stili . p.x. i prwti stili einai 0,5,10,15
// I deyteri stili 1,6,11 , i triti 2,7,12, kok. Etsi sto apo katw switch vazoume ta dedomena sto katalilo stoixeio
// me vasi to ypoloipo tis diairesis me to 5. ton arithmo grammis ton agnooume kai grafoume mono ta ypoloipa stoixeia
// stin antistoixi metavliti , coordinates gia tis syntetagmenes kai mtype gia ton typo.

    switch(i%5){
      case 1:
        coordinates[i/5 + 0*210] = float_to_fixed(atof(line)) ; // H synartisi atof metatrepei mia symvoloseira se float arithmo
        break;
      case 2:
        coordinates[i/5 + 1*210] = float_to_fixed(atof(line));
        break;
      case 3:
        coordinates[i/5 + 2*210] = float_to_fixed(atof(line));
        break;
      case 4:
        mtype[i/5] = atoi(line);
        break;
      default:
        break;
    }
    i++;

  }

  fclose(file); // Kleinoume to arxeio pou diavasame

  return 0;     // Epistrefoume 0 se epityxes fortwma twn dedomenwn

}

/*****************************************************
*  H synartisi pou grafei to teliko arxeio epitaxynsewn *
*****************************************************/
int writeOutFile(int * ra){

  FILE* outfile;    // H metavliti me ton pointer tou arxeiou

  outfile = fopen("results.txt","w"); // Anoigoume to arxeio results.txt gia na grapsoume

  if( outfile == 0 ){                       // An den mporoume na anoixoume to arxeio gia grapsimo
    printf("error : Cannot open output File");        // tote typwnoume to provlima
    return -1;                              // kai epistrefoume stin kyria sinartisi me lathos
  }

  int i;
  for(i=0;i<nMol;i++){
      fprintf(outfile,"%d  %f  %f  %f\n",i+1,fixed_to_float(ra[i  +  0*210]),fixed_to_float(ra[i+1*210]),fixed_to_float(ra[i+ 2*210])); // Grafoume se kathe seira tou arxeiou ta dedomena : i ra[i][0] ra[i][1] ra[i][2]
  }

  fclose(outfile);    // Kleinoume to arxeio pou grapsame

  return 0;   // Epistrefoume 0 oti ola pigan kala

}

/*****************************************************************************
* I synartisi typwnei 2 pinakes [ enan nMol x NDIM kai enan nMol x 1 seira seira *
*                      Kathe seira periexei ta dedomena:                      *
*         coordinates[i][0] coordinates[i][2] coordinates[i][2] *(mtype+i)         *
*****************************************************************************/
```

```
void printTables(int * coordinates,int * mtype){

  int i;
  for(i=0;i<nMol;i++){
        printf("%d   x:   %f   y:   %f   z:   %f   mtype:   %d\n",i,fixed_to_float(coordinates[i+
0*210]),fixed_to_float(coordinates[i+  1*210]),fixed_to_float(coordinates[i+  2*210]),*(mtype+i));
// Typwnetai kathe seira twn dedomenwn
  }

}

/***************************************
*     H kyria synatisi tou programmatos   *
***************************************/

int main(int argc, char *argv[]) {

  // Ftiaxnoume tis metavlites tis opoies tha apothikeysoume ta periexomena tou arxeiou

  int * coordinates;
  int * mtype;
  int * raold;

  // Desmevoume tin mnimi pou xreiazetai gia aytes tis metavlites.
  mtype = (int*)malloc(nMol*sizeof(int)); // Desmevoume mnimi gia enan pinaka nMol x 1

  coordinates = (int*)malloc((nMol*NDIM)*sizeof(int)); // Desmevoume mnimi gia tis nMol seires tou
pinaka pou perilamvanoun tis syntetagmenes
   raold  =  (int*)malloc((nMol*NDIM)*sizeof(int));  //  Desmevoume  mnimi  gia  tis  nMol  seires  tou
pinaka pou perilamvanoun tis syntetagmenes
  memset(raold,0,(nMol*NDIM)*sizeof(int));

  int i=0;
  // for(i=0;i<nMol;i++){
  //   coordinates[i] = (int*)malloc(NDIM*sizeof(int));  // Desmevoume NDIM stiles gia kathe seira
tou pinaka dimiourgwntas stin mnimi ena pinaka megethous nMol x NDIM
  // }

  if(loadFile("input.txt",coordinates,mtype) != 0 ){  //  i Synartisi gia na fortwsoume to arxeio
apo to arxeio input.txt
    printf("Error loading file");                       //  An yparxei provlima stin fortwsi tou
arxeiou typwnoume sfalma
    exit(-1);                                          //  Kai termatizoume to programma
  }

#if DEBUG>0                              //  Sto arxeio .h yparxei to define DEBUG. An to thesoume
ekei ws 1 tote
  printTables(coordinates,mtype);        //  Typwnontai ta periexomenta twn pinakwn pou periexoun
tis syntetagmenes kai ta mtypes
#endif                                   //  To DEBUG isxyei mexri edw

   int*  ra  =  ComputeForces(  raold,  coordinates,  mtype);    //  H  synartisi  tis  Matlab  opws
metaferthike stin C

#if DEBUG>0                        //  Antistoixa opws stis seires 177-179
  printTables(ra,mtype);           //  Typwnontai ta periexomena twn metavlitwn ra kai mtype
#endif

  writeOutFile(ra);       // Grafoume ta periexomenta tis metavlitis pou periexei tis epitaxynseis
se arxeio

}
```

Παραγόμενος Κώδικας σε γλώσσα VHDL

Fixed Point Square root

Without PARCS optimization

```
------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::: performed on design module: 'sqrt2'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:46 ____ :15

--::: The C-cubed compiler, back-end version: CCC_be_6 ::::--
--::::::: Copyright(c) 2007-2020, by Michael F. Dossis  ::::::::--
------------------------------------------------------------


  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;

  PACKAGE new_func IS

   TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

   TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

  END new_func;

  PACKAGE BODY new_func IS

  END new_func;









  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
```

```
  USE IEEE.std_logic_signed.ALL;
  LIBRARY WORK;
  USE WORK.new_func.ALL;

  ENTITY sqrt2 IS
   port(
        clock, reset, start, results_read : IN std_logic;
        number : IN  std_logic_vector(31 DOWNTO 0);

        sqrt2 : OUT  std_logic_vector(31 DOWNTO 0);
        done, busy : OUT std_logic
       );
  END sqrt2 ;


  ARCHITECTURE rtl OF sqrt2 IS

   SIGNAL done_int : std_logic;

   TYPE states_type IS (state_16,
                        state_15, state_14, state_13, state_12, state_11,
                        state_10, state_9, state_8, state_7, state_6,
                        state_5, state_4, state_3, state_2, state_1,
                        state_0);
   SIGNAL state : states_type;  -- this stores the current and next state of the circuit

   SIGNAL v002_t :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v003_q :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v004_b :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v005_r :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(1073741824,
32));  -- integer constants are converted into std_logic
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(0, 32));  --
integer constants are converted into std_logic
   CONSTANT const3 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(64, 32));  -
- integer constants are converted into std_logic
   SIGNAL var1 : std_logic;
   SIGNAL var2 : std_logic;
   CONSTANT const4 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2, 32));  --
integer constants are converted into std_logic
   CONSTANT const5 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(256,
32));  -- integer constants are converted into std_logic

     -- now the datapath (input/output) signal declarations follow --
```

```
  SIGNAL greater32_1_out : std_logic;
  SIGNAL greater32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL greater32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL greaterequal32_1_out : std_logic;
  SIGNAL greaterequal32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL greaterequal32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL minus32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL minus32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL minus32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL div32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL div32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL div32_1_in2 : std_logic_vector(31 DOWNTO 0);

BEGIN

 done <= done_int;

  --- FSM and controller loging (plus massively-parallel ops) ---
  fsm_core : PROCESS (clock, reset)
      VARIABLE var1_conditional_variable :  std_logic;
      VARIABLE var2_conditional_variable :  std_logic;
   BEGIN
    IF reset = '0' THEN
     done_int <= '0';
     busy <= '0';
     state <= state_0;
     sqrt2 <= (OTHERS => '0');
     v002_t <= (OTHERS => '0');
     v003_q <= (OTHERS => '0');
     v004_b <= (OTHERS => '0');
     v005_r <= (OTHERS => '0');
     var1 <= '0';
     var2 <= '0';


    ELSIF clock = '1' AND clock'EVENT THEN

      CASE state IS
```

```
        WHEN state_0 =>
         IF results_read = '1' THEN done_int <= '0'; END IF;
         IF start = '1' THEN
          IF done_int = '0' OR results_read = '1' THEN
           busy <= '1';   -- processing started
           state <= state_1;
          END IF;
         ELSE
          state <= state_0;
         END IF ;

        WHEN state_1 =>
          state <= state_2;
          v005_r(31 DOWNTO 0) <= number(31 DOWNTO 0) ;

        WHEN state_2 =>
          state <= state_3;
          v004_b(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

        WHEN state_3 =>
          state <= state_4;
          v003_q(31 DOWNTO 0) <= const2(31 DOWNTO 0) ;

        WHEN state_4 =>
          state <= state_5;
          var1 <= greater32_1_out;


        WHEN state_5 =>
        IF var1 = '1' THEN
           state <= state_6;
          ELSE
           state <= state_14;
          END IF;

        WHEN state_6 =>
          state <= state_7;
          v002_t <= plus32_1_out;

        WHEN state_7 =>
          state <= state_8;
          var2 <= greaterequal32_1_out;


        WHEN state_8 =>
```

```
        IF var2 = '1' THEN
           state <= state_9;
          ELSE
           state <= state_11;
          END IF;

        WHEN state_9 =>
          state <= state_10;
          v005_r <= minus32_1_out;

        WHEN state_10 =>
          state <= state_11;
          v003_q <= plus32_1_out;

        WHEN state_11 =>
          state <= state_12;
          v005_r <= mult32_1_out;

        WHEN state_12 =>
          state <= state_13;
          v004_b <= div32_1_out;

        WHEN state_13 =>
          state <= state_4;


        WHEN state_14 =>
          state <= state_15;
          v003_q <= mult32_1_out;

        WHEN state_15 =>
         state <= state_0;
         done_int <= '1';   -- processing finished, results are ready !!!
         busy <= '0';

       WHEN OTHERS => state <= state_0;
                      done_int <= '0';
                      busy <= '0';

      END CASE ;

   END IF ;
  END PROCESS fsm_core ;
```

```
--  Datapath operators (Functional Units) --
  greater32_1_out <= '1' WHEN greater32_1_in1 > greater32_1_in2 ELSE '0';
  plus32_1_out <= plus32_1_in1 + plus32_1_in2;
  greaterequal32_1_out <= '1' WHEN greaterequal32_1_in1 >= greaterequal32_1_in2 ELSE '0';
  minus32_1_out <= minus32_1_in1 - minus32_1_in2;
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  division_div32_1_out:
   WITH state SELECT
   div32_1_out <= conv_std_logic_vector(CONV_INTEGER(v004_b) / CONV_INTEGER(const4), 32) WHEN
state_12,
               (OTHERS => '0') WHEN OTHERS;


-- now the data routing --

-- now the greater-than comparisons --
  routing_greater32_1_in1:
   WITH state SELECT
   greater32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v004_b), 32) WHEN state_4,
                      (OTHERS => '0') WHEN OTHERS;
  routing_greater32_1_in2:
   WITH state SELECT
   greater32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const3), 32) WHEN state_4,
                      (OTHERS => '0') WHEN OTHERS;


-- now the additions --
  routing_plus32_1_in1:
   WITH state SELECT
   plus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v003_q), 32) WHEN state_6,
                     conv_std_logic_vector(CONV_INTEGER(v002_t), 32) WHEN state_10,
                      (OTHERS => '0') WHEN OTHERS;
  routing_plus32_1_in2:
   WITH state SELECT
   plus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v004_b), 32) WHEN state_6,
                     conv_std_logic_vector(CONV_INTEGER(v004_b), 32) WHEN state_10,
                      (OTHERS => '0') WHEN OTHERS;


-- now the greater-equality comparisons --
  routing_greaterequal32_1_in1:
   WITH state SELECT
   greaterequal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v005_r), 32) WHEN state_7,
                      (OTHERS => '0') WHEN OTHERS;
```

```
  routing_greaterequal32_1_in2:
   WITH state SELECT
   greaterequal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v002_t), 32) WHEN state_7,
                       (OTHERS => '0') WHEN OTHERS;


-- now the subtractions --
  routing_minus32_1_in1:
   WITH state SELECT
   minus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v005_r), 32) WHEN state_9,
                       (OTHERS => '0') WHEN OTHERS;
  routing_minus32_1_in2:
   WITH state SELECT
   minus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v002_t), 32) WHEN state_9,
                       (OTHERS => '0') WHEN OTHERS;


-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v005_r), 32) WHEN state_11,
                      conv_std_logic_vector(CONV_INTEGER(v003_q), 32) WHEN state_14,
                       (OTHERS => '0') WHEN OTHERS;
  routing_mult32_1_in2:
   WITH state SELECT
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_11,
                      conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_14,
                       (OTHERS => '0') WHEN OTHERS;



  END  rtl ;
```

With PARCS optimization

```
------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::::::: Optimiser engine used: 'PARCS' ::::::::::::::--

--:::::::::: performed on design module: 'sqrt2'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:46 ____ :21
```

```
--::: The C-cubed compiler, back-end version: CCC_be_6 :::--
--::::::: Copyright(c) 2007-2020, by Michael F. Dossis  :::::::--
-----------------------------------------------------------------


 LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
 USE IEEE.std_logic_arith.ALL;
 USE IEEE.std_logic_signed.ALL;

 PACKAGE new_func IS

  TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

  TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

 END new_func;

 PACKAGE BODY new_func IS

 END new_func;



 LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
 USE IEEE.std_logic_arith.ALL;
 USE IEEE.std_logic_signed.ALL;
 LIBRARY WORK;
 USE WORK.new_func.ALL;

 ENTITY sqrt2 IS
  port(
       clock, reset, start, results_read : IN std_logic;
       number : IN  std_logic_vector(31 DOWNTO 0);
       sqrt2 : OUT  std_logic_vector(31 DOWNTO 0);
       done, busy : OUT std_logic
      );
 END sqrt2 ;


 ARCHITECTURE rtl OF sqrt2 IS

  SIGNAL done_int : std_logic;
```

```
   TYPE states_type IS (state_8,
                        state_7, state_6, state_5, state_4, state_3,
                        state_2, state_1, state_0);
   SIGNAL state : states_type;  -- this stores the current and next state of the circuit

   SIGNAL v002_t :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v003_q :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v004_b :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v005_r :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(1073741824,
32));  -- integer constants are converted into std_logic
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(0, 32));  --
integer constants are converted into std_logic
   CONSTANT const3 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(64, 32));  -
- integer constants are converted into std_logic
   SIGNAL var1 : std_logic;
   SIGNAL var2 : std_logic;
   CONSTANT const4 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2, 32));  --
integer constants are converted into std_logic
   CONSTANT const5 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(256,
32));  -- integer constants are converted into std_logic

     -- now the datapath (input/output) signal declarations follow --

   SIGNAL greater32_1_out : std_logic;
   SIGNAL greater32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL greater32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL greaterequal32_1_out : std_logic;
   SIGNAL greaterequal32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL greaterequal32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL minus32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL minus32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL minus32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_in2 : std_logic_vector(31 DOWNTO 0);

  BEGIN
```

```
done <= done_int;

 --- FSM and controller loging (plus massively-parallel ops) ---
 fsm_core : PROCESS (clock, reset)
     VARIABLE var1_conditional_variable :  std_logic;
     VARIABLE var2_conditional_variable :  std_logic;
  BEGIN
   IF reset = '0' THEN
    done_int <= '0';
    busy <= '0';
    state <= state_0;
    sqrt2 <= (OTHERS => '0');
    v002_t <= (OTHERS => '0');
    v003_q <= (OTHERS => '0');
    v004_b <= (OTHERS => '0');
    v005_r <= (OTHERS => '0');
    var1 <= '0';
    var2 <= '0';


   ELSIF clock = '1' AND clock'EVENT THEN

     CASE state IS

       WHEN state_0 =>
        IF results_read = '1' THEN done_int <= '0'; END IF;
        IF start = '1' THEN
         IF done_int = '0' OR results_read = '1' THEN
          busy <= '1';   -- processing started
          state <= state_1;
         END IF;
        ELSE
         state <= state_0;
        END IF ;

       WHEN state_1 =>
         v003_q(31 DOWNTO 0) <= const2(31 DOWNTO 0) ;
         v004_b(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
         v005_r(31 DOWNTO 0) <= number(31 DOWNTO 0) ;
         state <= state_2;
       WHEN state_2 =>
         IF v004_b > const3 THEN var1 <= '1'; ELSE var1 <= '0'; END IF;
         state <= state_3;
       WHEN state_3 =>
         IF var1 = '1' THEN
```

```
          state <= state_4;
          ELSE
          state <= state_7;
          END IF;
          IF var1 = '1' THEN
          v002_t <= plus32_1_out;
          ELSE
          END IF;
        WHEN state_4 =>
          IF v005_r >= v002_t THEN var2 <= '1'; ELSE var2 <= '0'; END IF;
          state <= state_5;
        WHEN state_5 =>
          IF var2 = '1' THEN
          state <= state_6;
          ELSE
          state <= state_6;
          END IF;
          IF var2 = '1' THEN
          v005_r <= minus32_1_out;
          v003_q <= plus32_1_out;
          ELSE
          END IF;
        WHEN state_6 =>
          v005_r <= conv_std_logic_vector(CONV_INTEGER(v005_r) * CONV_INTEGER(const4), 32);
          v004_b <= conv_std_logic_vector(CONV_INTEGER(v004_b) / CONV_INTEGER(const4), 32);

          state <= state_2;
        WHEN state_7 =>
          v003_q <= conv_std_logic_vector(CONV_INTEGER(v003_q) * CONV_INTEGER(const5), 32);
          state <= state_8;
        WHEN state_8 =>
          sqrt2(31 DOWNTO 0) <= v003_q(31 DOWNTO 0) ;
          state <= state_0;
          done_int <= '1';   -- processing finished, results are ready !!!
          busy <= '0';

       WHEN OTHERS => state <= state_0;
                      done_int <= '0';
                      busy <= '0';

      END CASE ;

    END IF ;
  END PROCESS fsm_core ;
```

```
--  Datapath operators (Functional Units) --
  greater32_1_out <= '1' WHEN greater32_1_in1 > greater32_1_in2 ELSE '0';
  plus32_1_out <= plus32_1_in1 + plus32_1_in2;
  greaterequal32_1_out <= '1' WHEN greaterequal32_1_in1 >= greaterequal32_1_in2 ELSE '0';
  minus32_1_out <= minus32_1_in1 - minus32_1_in2;
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  division_div32_1_out:
   WITH state SELECT
   div32_1_out <= conv_std_logic_vector(CONV_INTEGER(v004_b) / CONV_INTEGER(const4), 32) WHEN
state_6,
               (OTHERS => '0') WHEN OTHERS;


-- now the data routing --

-- now the greater-than comparisons --
  routing_greater32_1_in1:
   WITH state SELECT
   greater32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v004_b), 32) WHEN state_2,
                      (OTHERS => '0') WHEN OTHERS;
  routing_greater32_1_in2:
   WITH state SELECT
   greater32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const3), 32) WHEN state_2,
                      (OTHERS => '0') WHEN OTHERS;


-- now the additions --
  routing_plus32_1_in1:
   WITH state SELECT
   plus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v003_q), 32) WHEN state_3,
                    conv_std_logic_vector(CONV_INTEGER(v002_t), 32) WHEN state_5,
                     (OTHERS => '0') WHEN OTHERS;
  routing_plus32_1_in2:
   WITH state SELECT
   plus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v004_b), 32) WHEN state_3,
                    conv_std_logic_vector(CONV_INTEGER(v004_b), 32) WHEN state_5,
                     (OTHERS => '0') WHEN OTHERS;


-- now the greater-equality comparisons --
  routing_greaterequal32_1_in1:
```

```
   WITH state SELECT
   greaterequal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v005_r), 32) WHEN state_4,
                       (OTHERS => '0') WHEN OTHERS;
  routing_greaterequal32_1_in2:
   WITH state SELECT
   greaterequal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v002_t), 32) WHEN state_4,
                       (OTHERS => '0') WHEN OTHERS;


-- now the subtractions --
  routing_minus32_1_in1:
   WITH state SELECT
   minus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v005_r), 32) WHEN state_5,
                       (OTHERS => '0') WHEN OTHERS;
  routing_minus32_1_in2:
   WITH state SELECT
   minus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v002_t), 32) WHEN state_5,
                       (OTHERS => '0') WHEN OTHERS;


-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v005_r), 32) WHEN state_6,
                      conv_std_logic_vector(CONV_INTEGER(v003_q), 32) WHEN state_7,
                       (OTHERS => '0') WHEN OTHERS;
  routing_mult32_1_in2:
   WITH state SELECT
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_6,
                      conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_7,
                       (OTHERS => '0') WHEN OTHERS;



  END  rtl ;
```

Power of Fixed Point

Without PARCS optimization

```
------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::: performed on design module: 'pow_int'
```

```
------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:46 ____ :29

--::: The C-cubed compiler, back-end version: CCC_be_6 :::--
--::::::: Copyright(c) 2007-2020, by Michael F. Dossis  :::::::--
-----------------------------------------------------------------


  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;

  PACKAGE new_func IS

   TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

   TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

  END new_func;

  PACKAGE BODY new_func IS

  END new_func;



  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;
  LIBRARY WORK;
  USE WORK.new_func.ALL;

  ENTITY pow_int IS
   port(
        clock, reset, start, results_read : IN std_logic;
        fp_div_results_read :  OUT std_logic;
        fp_div_start :  OUT std_logic;
        fp_div_done :  IN std_logic;
        fp_div_busy :  IN std_logic;
        fp_mult_results_read :  OUT std_logic;
        fp_mult_start :  OUT std_logic;
        fp_mult_done :  IN std_logic;
```

```
      fp_mult_busy :  IN std_logic;
      num : IN  std_logic_vector(31 DOWNTO 0);

      pow : IN  std_logic_vector(31 DOWNTO 0);

      pow_int : OUT  std_logic_vector(31 DOWNTO 0);

      fp_div_numa : OUT  std_logic_vector(31 DOWNTO 0);

      fp_div_numb : OUT  std_logic_vector(31 DOWNTO 0);

      fp_div_fp_div : IN  std_logic_vector(31 DOWNTO 0);

      fp_mult_numa : OUT  std_logic_vector(31 DOWNTO 0);

      fp_mult_numb : OUT  std_logic_vector(31 DOWNTO 0);

      fp_mult_fp_mult : IN  std_logic_vector(31 DOWNTO 0);

      done, busy : OUT std_logic
     );
END pow_int ;


ARCHITECTURE rtl OF pow_int IS

 SIGNAL done_int : std_logic;

 TYPE states_type IS (state_38,
                      state_37, state_36, state_35, state_34, state_33,
                      state_32, state_31, state_30, state_29, state_28,
                      state_27, state_26, state_25, state_24, state_23,
                      state_22, state_21, state_20, state_19, state_18,
                      state_17, state_16, state_15, state_14, state_13,
                      state_12, state_11, state_10, state_9, state_8,
                      state_7, state_6, state_5, state_4, state_3,
                      state_2, state_1, state_0);
 SIGNAL state : states_type;  -- this stores the current and next state of the circuit

 SIGNAL temporary_pow :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v003_number :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v004_divider :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary000 : std_logic;
 SIGNAL temporary001 : std_logic;
 SIGNAL temporary002 :  std_logic_vector(31 DOWNTO 0);
```

```
   SIGNAL temporary003 : std_logic;
   CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(1, 32));  --
integer constants are converted into std_logic
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(0, 32));  --
integer constants are converted into std_logic
   SIGNAL var1 : std_logic;
   CONSTANT const3 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(65536,
32));  -- integer constants are converted into std_logic
   SIGNAL var2 : std_logic;
   CONSTANT const4 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(-1, 32));  -
- integer constants are converted into std_logic
   SIGNAL var3 : std_logic;
   SIGNAL var4 : std_logic;
   SIGNAL fp_div_results_read_int :  std_logic;
   SIGNAL fp_div_start_int :  std_logic;
   SIGNAL fp_mult_results_read_int :  std_logic;
   SIGNAL fp_mult_start_int :  std_logic;

     -- now the datapath (input/output) signal declarations follow --

   SIGNAL equal32_1_out : std_logic;
   SIGNAL equal32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL equal32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL greater32_1_out : std_logic;
   SIGNAL greater32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL greater32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL not1_1_out : std_logic;
   SIGNAL not1_1_in2 : std_logic;
   SIGNAL less32_1_out : std_logic;
   SIGNAL less32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL less32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL minus32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL minus32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL minus32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_in2 : std_logic_vector(31 DOWNTO 0);

  BEGIN

   done <= done_int;
   fp_div_start <= fp_div_start_int;
   fp_div_results_read <= fp_div_results_read_int;
   fp_mult_start <= fp_mult_start_int;
   fp_mult_results_read <= fp_mult_results_read_int;
```

```
--- FSM and controller loging (plus massively-parallel ops) ---
fsm_core : PROCESS (clock, reset)
    VARIABLE var1_conditional_variable :  std_logic;
    VARIABLE var2_conditional_variable :  std_logic;
    VARIABLE temporary000_conditional_variable :  std_logic;
    VARIABLE var3_conditional_variable :  std_logic;
    VARIABLE var4_conditional_variable :  std_logic;
 BEGIN
  IF reset = '0' THEN
   done_int <= '0';
   busy <= '0';
   state <= state_0;
   pow_int <= (OTHERS => '0');
   temporary_pow <= (OTHERS => '0');
   v003_number <= (OTHERS => '0');
   v004_divider <= (OTHERS => '0');
   temporary000 <= '0';
   temporary001 <= '0';
   temporary002 <= (OTHERS => '0');
   temporary003 <= '0';
   var1 <= '0';
   var2 <= '0';
   var3 <= '0';
   var4 <= '0';
   fp_div_numa <= (OTHERS => '0');
   fp_div_numb <= (OTHERS => '0');
   fp_mult_numa <= (OTHERS => '0');
   fp_mult_numb <= (OTHERS => '0');
   fp_div_results_read_int <= '0';
   fp_div_start_int <= '0';
   fp_mult_results_read_int <= '0';
   fp_mult_start_int <= '0';


  ELSIF clock = '1' AND clock'EVENT THEN

    CASE state IS

      WHEN state_0 =>
       IF results_read = '1' THEN done_int <= '0'; END IF;
       IF start = '1' THEN
        IF done_int = '0' OR results_read = '1' THEN
         busy <= '1';    -- processing started
         state <= state_1;
```

```
    END IF;
   ELSE
    state <= state_0;
   END IF ;

  WHEN state_1 =>
    state <= state_2;
    temporary_pow(31 DOWNTO 0) <= pow(31 DOWNTO 0) ;

  WHEN state_2 =>
    state <= state_3;
    v004_divider(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

  WHEN state_3 =>
    state <= state_4;
    var1 <= equal32_1_out;


  WHEN state_4 =>
  IF var1 = '1' THEN
     state <= state_5;
    ELSE
     state <= state_7;
    END IF;

  WHEN state_5 =>
    state <= state_6;
    v003_number(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

  WHEN state_6 =>
    state <= state_12;


  WHEN state_7 =>
    state <= state_8;
    var2 <= greater32_1_out;


  WHEN state_8 =>
  IF var2 = '1' THEN
     state <= state_9;
    ELSE
     state <= state_11;
    END IF;
```

```
        WHEN state_9 =>
          state <= state_10;
          v003_number(31 DOWNTO 0) <= num(31 DOWNTO 0) ;

        WHEN state_10 =>
          state <= state_12;


        WHEN state_11 =>
          state <= state_12;


        WHEN state_12 =>
          state <= state_13;
          temporary001 <= greater32_1_out;

        WHEN state_13 =>
          state <= state_14;
          temporary000 <= not1_1_out;


        WHEN state_14 =>
        IF temporary000 = '1' THEN
           state <= state_15;
          ELSE
           state <= state_18;
          END IF;

        WHEN state_15 =>
          state <= state_16;
          temporary003 <= less32_1_out;

        WHEN state_16 =>
          state <= state_17;
          temporary000 <= temporary003 ;

        WHEN state_17 =>
          state <= state_19;


        WHEN state_18 =>
          state <= state_19;
          temporary000 <= not1_1_out;
```

```
        WHEN state_19 =>
        IF temporary000 = '1' THEN
           state <= state_20;
          ELSE
           state <= state_37;
          END IF;

        WHEN state_20 =>
          state <= state_21;
          var3 <= greater32_1_out;


        WHEN state_21 =>
        IF var3 = '1' THEN
           state <= state_22;
          ELSE
           state <= state_25;
          END IF;

        WHEN state_22 =>
          state <= state_23;


        WHEN state_23 =>
          state <= state_24;
          temporary_pow <= minus32_1_out;

        WHEN state_24 =>
          state <= state_29;


        WHEN state_25 =>
          state <= state_26;
          var4 <= less32_1_out;


        WHEN state_26 =>
        IF var4 = '1' THEN
           state <= state_27;
          ELSE
           state <= state_29;
          END IF;

        WHEN state_27 =>
          state <= state_28;
```

```
WHEN state_28 =>
  state <= state_29;
  temporary_pow <= plus32_1_out;

WHEN state_29 =>
  state <= state_30;
  temporary001 <= greater32_1_out;

WHEN state_30 =>
  state <= state_31;
  temporary000 <= not1_1_out;


WHEN state_31 =>
IF temporary000 = '1' THEN
   state <= state_32;
  ELSE
   state <= state_35;
  END IF;

WHEN state_32 =>
  state <= state_33;
  temporary003 <= less32_1_out;

WHEN state_33 =>
  state <= state_34;
  temporary000 <= temporary003 ;

WHEN state_34 =>
  state <= state_36;


WHEN state_35 =>
  state <= state_36;
  temporary000 <= not1_1_out;

WHEN state_36 =>
  state <= state_19;


WHEN state_37 =>
 state <= state_0;
 done_int <= '1';   -- processing finished, results are ready !!!
```

```
          busy <= '0';

        WHEN OTHERS => state <= state_0;
                       done_int <= '0';
                       busy <= '0';

       END CASE ;

     END IF ;
    END PROCESS fsm_core ;



--  Datapath operators (Functional Units) --
  equal32_1_out <= '1' WHEN equal32_1_in1 = equal32_1_in2 ELSE '0';
  greater32_1_out <= '1' WHEN greater32_1_in1 > greater32_1_in2 ELSE '0';
  not1_1_out <= not not1_1_in2;
  less32_1_out <= '1' WHEN less32_1_in1 < less32_1_in2 ELSE '0';
  minus32_1_out <= minus32_1_in1 - minus32_1_in2;
  plus32_1_out <= plus32_1_in1 + plus32_1_in2;


-- now the data routing --

-- now the equality comparisons --
  routing_equal32_1_in1:
   WITH state SELECT
   equal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_3,
                    (OTHERS => '0') WHEN OTHERS;
  routing_equal32_1_in2:
   WITH state SELECT
   equal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const2), 32) WHEN state_3,
                    (OTHERS => '0') WHEN OTHERS;


-- now the greater-than comparisons --
  routing_greater32_1_in1:
   WITH state SELECT
   greater32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_7,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_12,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_20,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_29,
                       (OTHERS => '0') WHEN OTHERS;
  routing_greater32_1_in2:
   WITH state SELECT
```

```
   greater32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const2), 32) WHEN state_7,
                      conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_12,
                      conv_std_logic_vector(CONV_INTEGER(const2), 32) WHEN state_20,
                      conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_29,
                       (OTHERS => '0') WHEN OTHERS;



-- now the less-than comparisons --
  routing_less32_1_in1:
   WITH state SELECT
   less32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_15,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_25,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_32,
                       (OTHERS => '0') WHEN OTHERS;
  routing_less32_1_in2:
   WITH state SELECT
   less32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary002), 32) WHEN state_15,
                      conv_std_logic_vector(CONV_INTEGER(const2), 32) WHEN state_25,
                      conv_std_logic_vector(CONV_INTEGER(temporary002), 32) WHEN state_32,
                       (OTHERS => '0') WHEN OTHERS;


-- now the subtractions --
  routing_minus32_1_in1:
   WITH state SELECT
   minus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_23,
                       (OTHERS => '0') WHEN OTHERS;
  routing_minus32_1_in2:
   WITH state SELECT
   minus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_23,
                       (OTHERS => '0') WHEN OTHERS;


-- now the additions --
  routing_plus32_1_in1:
   WITH state SELECT
   plus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_28,
                       (OTHERS => '0') WHEN OTHERS;
  routing_plus32_1_in2:
   WITH state SELECT
   plus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_28,
                       (OTHERS => '0') WHEN OTHERS;
```

```
  END  rtl ;
```

With PARCS optimization

```
------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::::::: Optimiser engine used: 'PARCS' ::::::::::::::--

--:::::::::: performed on design module: 'pow_int'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:46 ____ :38

--::: The C-cubed compiler, back-end version: CCC_be_6 ::::--
--:::::: Copyright(c) 2007-2020, by Michael F. Dossis  ::::::::--
------------------------------------------------------------


  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;

  PACKAGE new_func IS

   TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

   TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

  END new_func;

  PACKAGE BODY new_func IS

  END new_func;



  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;
  LIBRARY WORK;
  USE WORK.new_func.ALL;
```

```
ENTITY pow_int IS
 port(
      clock, reset, start, results_read : IN std_logic;
      fp_div_results_read :  OUT std_logic;
      fp_div_start :  OUT std_logic;
      fp_div_done :  IN std_logic;
      fp_div_busy :  IN std_logic;
      fp_mult_results_read :  OUT std_logic;
      fp_mult_start :  OUT std_logic;
      fp_mult_done :  IN std_logic;
      fp_mult_busy :  IN std_logic;
      num : IN  std_logic_vector(31 DOWNTO 0);
      pow : IN  std_logic_vector(31 DOWNTO 0);
      pow_int : OUT  std_logic_vector(31 DOWNTO 0);
      fp_div_numa : OUT  std_logic_vector(31 DOWNTO 0);
      fp_div_numb : OUT  std_logic_vector(31 DOWNTO 0);
      fp_div_fp_div : IN  std_logic_vector(31 DOWNTO 0);
      fp_mult_numa : OUT  std_logic_vector(31 DOWNTO 0);
      fp_mult_numb : OUT  std_logic_vector(31 DOWNTO 0);
      fp_mult_fp_mult : IN  std_logic_vector(31 DOWNTO 0);
      done, busy : OUT std_logic
     );
END pow_int ;


ARCHITECTURE rtl OF pow_int IS

 SIGNAL done_int : std_logic;

 TYPE states_type IS (state_22,
                      state_21, state_20, state_19, state_18, state_17,
                      state_16, state_15, state_14, state_13, state_12,
                      state_11, state_10, state_9, state_8, state_7,
                      state_6, state_5, state_4, state_3, state_2,
                      state_1, state_0);
 SIGNAL state : states_type;  -- this stores the current and next state of the circuit

 SIGNAL temporary_pow :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v003_number :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v004_divider :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary000 : std_logic;
 SIGNAL temporary001 : std_logic;
 SIGNAL temporary002 :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary003 : std_logic;
```

```
   CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(1, 32));  --
integer constants are converted into std_logic
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(0, 32));  --
integer constants are converted into std_logic
   SIGNAL var1 : std_logic;
   CONSTANT const3 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(65536,
32));  -- integer constants are converted into std_logic
   SIGNAL var2 : std_logic;
   CONSTANT const4 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(-1, 32));  -
- integer constants are converted into std_logic
   SIGNAL var3 : std_logic;
   SIGNAL var4 : std_logic;
   SIGNAL fp_div_results_read_int : std_logic;
   SIGNAL fp_div_start_int : std_logic;
   SIGNAL fp_mult_results_read_int : std_logic;
   SIGNAL fp_mult_start_int : std_logic;

     -- now the datapath (input/output) signal declarations follow --

   SIGNAL equal32_1_out : std_logic;
   SIGNAL equal32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL equal32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL greater32_1_out : std_logic;
   SIGNAL greater32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL greater32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL not1_1_out : std_logic;
   SIGNAL not1_1_in2 : std_logic;
   SIGNAL less32_1_out : std_logic;
   SIGNAL less32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL less32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL minus32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL minus32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL minus32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_in2 : std_logic_vector(31 DOWNTO 0);

  BEGIN

   done <= done_int;
   fp_div_start <= fp_div_start_int;
   fp_div_results_read <= fp_div_results_read_int;
   fp_mult_start <= fp_mult_start_int;
   fp_mult_results_read <= fp_mult_results_read_int;
```

```
--- FSM and controller loging (plus massively-parallel ops) ---
fsm_core : PROCESS (clock, reset)
    VARIABLE var1_conditional_variable :  std_logic;
    VARIABLE var2_conditional_variable :  std_logic;
    VARIABLE temporary000_conditional_variable :  std_logic;
    VARIABLE var3_conditional_variable :  std_logic;
    VARIABLE var4_conditional_variable :  std_logic;
 TYPE STATES_ARRAY IS ARRAY (integer RANGE <>) OF STATES_TYPE;
 VARIABLE state_var : STATES_ARRAY(1 DOWNTO 1);
 BEGIN
  IF reset = '0' THEN
   done_int <= '0';
   busy <= '0';
   state <= state_0;
   pow_int <= (OTHERS => '0');
   temporary_pow <= (OTHERS => '0');
   v003_number <= (OTHERS => '0');
   v004_divider <= (OTHERS => '0');
   temporary000 <= '0';
   temporary001 <= '0';
   temporary002 <= (OTHERS => '0');
   temporary003 <= '0';
   var1 <= '0';
   var2 <= '0';
   var3 <= '0';
   var4 <= '0';

   fp_div_results_read_int <= '0';
   fp_div_start_int <= '0';
   fp_div_numa <= (OTHERS => '0');
   fp_div_numb <= (OTHERS => '0');
   fp_mult_results_read_int <= '0';
   fp_mult_start_int <= '0';
   fp_mult_numa <= (OTHERS => '0');
   fp_mult_numb <= (OTHERS => '0');

  ELSIF clock = '1' AND clock'EVENT THEN

    CASE state IS

      WHEN state_0 =>
       IF results_read = '1' THEN done_int <= '0'; END IF;
       IF start = '1' THEN
        IF done_int = '0' OR results_read = '1' THEN
         busy <= '1';    -- processing started
```

```
    state <= state_1;
   END IF;
  ELSE
   state <= state_0;
  END IF ;

 WHEN state_1 =>
   v004_divider(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
   temporary_pow(31 DOWNTO 0) <= pow(31 DOWNTO 0) ;
   state <= state_2;
 WHEN state_2 =>
   IF temporary_pow = const2 THEN var1 <= '1'; ELSE var1 <= '0'; END IF;
   state <= state_3;
 WHEN state_3 =>
   IF var1 = '1' THEN
   state <= state_7;
   ELSE
   state <= state_4;
   END IF;
   IF var1 = '1' THEN
   v003_number(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

   ELSE
   END IF;
 WHEN state_4 =>
   IF temporary_pow > const2 THEN var2 <= '1'; ELSE var2 <= '0'; END IF;
   state <= state_5;
 WHEN state_5 =>
   IF var2 = '1' THEN
   state <= state_7;
   ELSE
   state <= state_6;
   END IF;
   IF var2 = '1' THEN
   v003_number(31 DOWNTO 0) <= num(31 DOWNTO 0) ;

   ELSE
   END IF;
 WHEN state_6 =>
   ------ this is a call to module : fp_div -----
  fp_div_numa <= const3;
  fp_div_numb <= num;

              IF fp_div_busy = '0' AND fp_div_done = '0' THEN
               IF fp_div_results_read_int = '1' THEN
```

```
                                    fp_div_start_int <= '0';
                                    fp_div_results_read_int <= '0';
                          state <= state_7;
                         ELSE
                          fp_div_start_int <= '1';
                          state <= state_6;
                         END IF;
                        ELSIF fp_div_busy = '1' AND fp_div_start_int = '1' THEN   -- when it begins
                         fp_div_start_int <= '0';
                         state <= state_6;
                        ELSIF fp_div_done = '1'  THEN   -- when it is completed then read the
results
           v003_number <= fp_div_fp_div;
                         fp_div_start_int <= '0';
                         IF fp_div_results_read_int = '0' THEN  -- if it is done then indicate that
the results are read
                          fp_div_results_read_int <= '1';
                          state <= state_6;
                         ELSIF             fp_div_results_read_int = '1'              THEN -- all
calls are synchronized and completed
                                     fp_div_results_read_int <= '0';
                          state <= state_7;
                         END IF;
                        ELSE
                         state <= state_6;
                        END IF;
          WHEN state_7 =>
            IF temporary_pow > const1 THEN temporary001 <= '1'; ELSE temporary001 <= '0'; END IF;
            state <= state_8;
          WHEN state_8 =>
            temporary000 <= not temporary001 ;
            state <= state_9;
          WHEN state_9 =>
            IF temporary000 = '1' THEN
            state <= state_10;
            ELSE
            state <= state_11;
            END IF;
            IF temporary000 = '1' THEN
            temporary003 <= less32_1_out;
            ELSE
            END IF;
          WHEN state_10 =>
            temporary000 <= temporary003 ;
```

```
            state <= state_12;
          WHEN state_11 =>
            temporary000 <= not temporary000 ;
            state <= state_12;
          WHEN state_12 =>
            IF temporary000 = '1' THEN
            state <= state_13;
            ELSE
            state <= state_22;
            END IF;
            IF temporary000 = '1' THEN
            var3 <= greater32_1_out;
            ELSE
            END IF;
          WHEN state_13 =>
            IF var3 = '1' THEN
            state <= state_16;
            ELSE
            state <= state_14;
            END IF;
            IF var3 = '1' THEN

            temporary_pow <= minus32_1_out;

             ------ this is a call to module : fp_mult -----
           fp_mult_numa <= v003_number;
           fp_mult_numb <= num;

                        IF fp_mult_busy = '0' AND fp_mult_done = '0' THEN
                         IF fp_mult_results_read_int = '1' THEN
                                      fp_mult_start_int <= '0';
                                      fp_mult_results_read_int <= '0';
                           state <= state_7;
                          ELSE
                           fp_mult_start_int <= '1';
                           state <= state_13;
                          END IF;
                        ELSIF fp_mult_busy = '1' AND fp_mult_start_int = '1'  THEN   -- when it
begins
                         fp_mult_start_int <= '0';
                         state <= state_13;
                        ELSIF fp_mult_done = '1'  THEN   -- when it is completed then read the
results
            v003_number <= fp_mult_fp_mult;
                         fp_mult_start_int <= '0';
```

```
                        IF fp_mult_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                         fp_mult_results_read_int <= '1';
                         state <= state_13;
                        ELSIF             fp_mult_results_read_int = '1'              THEN --
all calls are synchronized and completed
                                     fp_mult_results_read_int <= '0';
                         state <= state_7;
                        END IF;
                       ELSE
                        state <= state_13;
                       END IF;
            ELSE
            END IF;
          WHEN state_14 =>
            IF temporary_pow < const2 THEN var4 <= '1'; ELSE var4 <= '0'; END IF;
            state <= state_15;
          WHEN state_15 =>
            IF var4 = '1' THEN
            state <= state_16;
            ELSE
            state <= state_16;
            END IF;
            IF var4 = '1' THEN

            temporary_pow <= plus32_1_out;
             ------ this is a call to module : fp_div -----
           fp_div_numa <= v003_number;
           fp_div_numb <= num;

                       IF fp_div_busy = '0' AND fp_div_done = '0' THEN
                        IF fp_div_results_read_int = '1' THEN
                                     fp_div_start_int <= '0';
                                     fp_div_results_read_int <= '0';
                          state <= state_7;
                         ELSE
                          fp_div_start_int <= '1';
                          state <= state_15;
                         END IF;
                       ELSIF fp_div_busy = '1' AND fp_div_start_int = '1'  THEN   -- when it
begins
                        fp_div_start_int <= '0';
                        state <= state_15;
                       ELSIF fp_div_done = '1'   THEN   -- when it is completed then read the
results
```

```
          v003_number <= fp_div_fp_div;
                       fp_div_start_int <= '0';
                       IF fp_div_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                        fp_div_results_read_int <= '1';
                        state <= state_15;
                       ELSIF              fp_div_results_read_int = '1'               THEN --
all calls are synchronized and completed
                                    fp_div_results_read_int <= '0';
                        state <= state_7;
                       END IF;
                      ELSE
                       state <= state_15;
                      END IF;
            ELSE
            END IF;
          WHEN state_16 =>
            IF temporary_pow > const1 THEN temporary001 <= '1'; ELSE temporary001 <= '0'; END IF;
            state <= state_17;
          WHEN state_17 =>
            temporary000 <= not temporary001 ;
            state <= state_18;
          WHEN state_18 =>
            IF temporary000 = '1' THEN
            state <= state_19;
            ELSE
            state <= state_20;
            END IF;
            IF temporary000 = '1' THEN
            temporary003 <= less32_1_out;
            ELSE
            END IF;
          WHEN state_19 =>
            temporary000 <= temporary003 ;

            state <= state_21;
          WHEN state_20 =>
            temporary000 <= not temporary000 ;
            state <= state_21;
          WHEN state_21 =>

            state <= state_12;
          WHEN state_22 =>
            pow_int(31 DOWNTO 0) <= v003_number(31 DOWNTO 0) ;
            state <= state_0;
```

```
            done_int <= '1';    -- processing finished, results are ready !!!
            busy <= '0';

         WHEN OTHERS => state <= state_0;
                        done_int <= '0';
                        busy <= '0';

        END CASE ;

      END IF ;
     END PROCESS fsm_core ;



--  Datapath operators (Functional Units) --
  equal32_1_out <= '1' WHEN equal32_1_in1 = equal32_1_in2 ELSE '0';
  greater32_1_out <= '1' WHEN greater32_1_in1 > greater32_1_in2 ELSE '0';
  not1_1_out <= not not1_1_in2;
  less32_1_out <= '1' WHEN less32_1_in1 < less32_1_in2 ELSE '0';
  minus32_1_out <= minus32_1_in1 - minus32_1_in2;
  plus32_1_out <= plus32_1_in1 + plus32_1_in2;


-- now the data routing --

-- now the equality comparisons --
  routing_equal32_1_in1:
   WITH state SELECT
   equal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_2,
                       (OTHERS => '0') WHEN OTHERS;
  routing_equal32_1_in2:
   WITH state SELECT
   equal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const2), 32) WHEN state_2,
                       (OTHERS => '0') WHEN OTHERS;


-- now the greater-than comparisons --
  routing_greater32_1_in1:
   WITH state SELECT
   greater32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_4,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_7,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_12,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_16,
                       (OTHERS => '0') WHEN OTHERS;
  routing_greater32_1_in2:
```

```
   WITH state SELECT
   greater32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const2), 32) WHEN state_4,
                      conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_7,
                      conv_std_logic_vector(CONV_INTEGER(const2), 32) WHEN state_12,
                      conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_16,
                       (OTHERS => '0') WHEN OTHERS;



-- now the less-than comparisons --
  routing_less32_1_in1:
   WITH state SELECT
   less32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_9,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_14,
                      conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_18,
                       (OTHERS => '0') WHEN OTHERS;
  routing_less32_1_in2:
   WITH state SELECT
   less32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary002), 32) WHEN state_9,
                      conv_std_logic_vector(CONV_INTEGER(const2), 32) WHEN state_14,
                      conv_std_logic_vector(CONV_INTEGER(temporary002), 32) WHEN state_18,
                       (OTHERS => '0') WHEN OTHERS;


-- now the subtractions --
  routing_minus32_1_in1:
   WITH state SELECT
   minus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_13,
                       (OTHERS => '0') WHEN OTHERS;
  routing_minus32_1_in2:
   WITH state SELECT
   minus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_13,
                       (OTHERS => '0') WHEN OTHERS;


-- now the additions --
  routing_plus32_1_in1:
   WITH state SELECT
   plus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_15,
                       (OTHERS => '0') WHEN OTHERS;
  routing_plus32_1_in2:
   WITH state SELECT
   plus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_15,
                       (OTHERS => '0') WHEN OTHERS;
```

```
END  rtl ;
```

Fixed Point Multiplication

Without PARCS optimization

```
------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::: performed on design module: 'fp_mult'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:45 ____ :88

--::: The C-cubed compiler, back-end version: CCC_be_6 ::::--
--:::::: Copyright(c) 2007-2020, by Michael F. Dossis  ::::::::--
------------------------------------------------------------


 LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
 USE IEEE.std_logic_arith.ALL;
 USE IEEE.std_logic_signed.ALL;

 PACKAGE new_func IS

  TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

  TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

 END new_func;

 PACKAGE BODY new_func IS

 END new_func;

 LIBRARY IEEE;

 USE IEEE.std_logic_1164.ALL;
 USE IEEE.std_logic_arith.ALL;
 USE IEEE.std_logic_signed.ALL;
 LIBRARY WORK;
 USE WORK.new_func.ALL;

 ENTITY fp_mult IS
  port(
```

```
        clock, reset, start, results_read : IN std_logic;
        numa : IN  std_logic_vector(31 DOWNTO 0);

        numb : IN  std_logic_vector(31 DOWNTO 0);

        fp_mult : OUT  std_logic_vector(31 DOWNTO 0);

        done, busy : OUT std_logic
       );
  END fp_mult ;


  ARCHITECTURE rtl OF fp_mult IS

   SIGNAL done_int : std_logic;

   TYPE states_type IS (state_7,
                        state_6, state_5, state_4, state_3, state_2,
                        state_1, state_0);
   SIGNAL state : states_type;  -- this stores the current and next state of the circuit

   SIGNAL v003_mult :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v004_mult_int :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary001 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary002 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var1 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(65536,
32));  -- integer constants are converted into std_logic
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(-1, 32));  -
- integer constants are converted into std_logic
   SIGNAL var2 : std_logic;

     -- now the datapath (input/output) signal declarations follow --

   SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL and32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL and32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL and32_1_in2 : std_logic_vector(31 DOWNTO 0);
```

```
BEGIN

 done <= done_int;

  --- FSM and controller loging (plus massively-parallel ops) ---
 fsm_core : PROCESS (clock, reset)
  BEGIN
   IF reset = '0' THEN
    done_int <= '0';
    busy <= '0';
    state <= state_0;
    fp_mult <= (OTHERS => '0');
    v003_mult <= (OTHERS => '0');
    v004_mult_int <= (OTHERS => '0');
    temporary000 <= (OTHERS => '0');
    temporary001 <= (OTHERS => '0');
    temporary002 <= (OTHERS => '0');
    var1 <= (OTHERS => '0');
    var2 <= '0';


   ELSIF clock = '1' AND clock'EVENT THEN

     CASE state IS

       WHEN state_0 =>
        IF results_read = '1' THEN done_int <= '0'; END IF;
        IF start = '1' THEN
         IF done_int = '0' OR results_read = '1' THEN
          busy <= '1';   -- processing started
          state <= state_1;
         END IF;
        ELSE
         state <= state_0;
        END IF ;

       WHEN state_1 =>
         state <= state_2;
         temporary000(31 DOWNTO 0) <= numa(31 DOWNTO 0) ;

       WHEN state_2 =>
         state <= state_3;
         temporary001(31 DOWNTO 0) <= numb(31 DOWNTO 0) ;

       WHEN state_3 =>
```

```
            state <= state_4;
            var1 <= mult32_1_out;

          WHEN state_4 =>
            state <= state_5;
            v003_mult <= div32_1_out;

          WHEN state_5 =>
            state <= state_6;
            v004_mult_int <= and32_1_out;

          WHEN state_6 =>
           state <= state_0;
           done_int <= '1';   -- processing finished, results are ready !!!
           busy <= '0';

         WHEN OTHERS => state <= state_0;
                        done_int <= '0';
                        busy <= '0';

        END CASE ;

      END IF ;
     END PROCESS fsm_core ;



--  Datapath operators (Functional Units) --
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  division_div32_1_out:
   WITH state SELECT
   div32_1_out <= conv_std_logic_vector(CONV_INTEGER(var1) / CONV_INTEGER(const1), 32) WHEN
state_4,
               (OTHERS => '0') WHEN OTHERS;
  and32_1_out <= and32_1_in1 and and32_1_in2;


-- now the data routing --

-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary000), 32) WHEN state_3,
                       (OTHERS => '0') WHEN OTHERS;
```

```
  routing_mult32_1_in2:
   WITH state SELECT
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary001), 32) WHEN state_3,
                       (OTHERS => '0') WHEN OTHERS;



-- now the logical AND --
  routing_and32_1_in1:
   WITH state SELECT
   and32_1_in1 <= v003_mult WHEN state_5,
                       (OTHERS => '0') WHEN OTHERS;
  routing_and32_1_in2:
   WITH state SELECT
   and32_1_in2 <= temporary002 WHEN state_5,
                       (OTHERS => '0') WHEN OTHERS;


  END  rtl ;
```

With PARCS optimization

```
------------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::::::: Optimiser engine used: 'PARCS' ::::::::::::::--

--:::::::::: performed on design module: 'fp_mult'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:45 ____ :98

--::: The C-cubed compiler, back-end version: CCC_be_6 ::::--
--:::::: Copyright(c) 2007-2020, by Michael F. Dossis  ::::::::--
------------------------------------------------------------------


  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;

  PACKAGE new_func IS

   TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);
```

```
   TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

  END new_func;

  PACKAGE BODY new_func IS

  END new_func;



  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;
  LIBRARY WORK;
  USE WORK.new_func.ALL;

  ENTITY fp_mult IS
   port(
        clock, reset, start, results_read : IN std_logic;
        numa : IN  std_logic_vector(31 DOWNTO 0);
        numb : IN  std_logic_vector(31 DOWNTO 0);
        fp_mult : OUT  std_logic_vector(31 DOWNTO 0);
        done, busy : OUT std_logic
       );
  END fp_mult ;


  ARCHITECTURE rtl OF fp_mult IS

   SIGNAL done_int : std_logic;

   TYPE states_type IS (state_5,
                        state_4, state_3, state_2, state_1, state_0);
   SIGNAL state : states_type;  -- this stores the current and next state of the circuit

   SIGNAL v003_mult :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v004_mult_int :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary001 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary002 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var1 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(65536,
32));  -- integer constants are converted into std_logic
```

```
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(-1, 32));  -
- integer constants are converted into std_logic
   SIGNAL var2 : std_logic;

     -- now the datapath (input/output) signal declarations follow --

   SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL and32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL and32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL and32_1_in2 : std_logic_vector(31 DOWNTO 0);

  BEGIN

   done <= done_int;

    --- FSM and controller loging (plus massively-parallel ops) ---
    fsm_core : PROCESS (clock, reset)
     BEGIN
      IF reset = '0' THEN
       done_int <= '0';
       busy <= '0';
       state <= state_0;
       fp_mult <= (OTHERS => '0');
       v003_mult <= (OTHERS => '0');
       v004_mult_int <= (OTHERS => '0');
       temporary000 <= (OTHERS => '0');
       temporary001 <= (OTHERS => '0');
       temporary002 <= (OTHERS => '0');
       var1 <= (OTHERS => '0');
       var2 <= '0';


      ELSIF clock = '1' AND clock'EVENT THEN

        CASE state IS

          WHEN state_0 =>
           IF results_read = '1' THEN done_int <= '0'; END IF;
           IF start = '1' THEN
            IF done_int = '0' OR results_read = '1' THEN
```

```
            busy <= '1';   -- processing started
            state <= state_1;
           END IF;
          ELSE
           state <= state_0;
          END IF ;

         WHEN state_1 =>
           temporary001(31 DOWNTO 0) <= numb(31 DOWNTO 0) ;
           temporary000(31 DOWNTO 0) <= numa(31 DOWNTO 0) ;
           state <= state_2;
         WHEN state_2 =>
           var1 <= conv_std_logic_vector(CONV_INTEGER(temporary000) * CONV_INTEGER(temporary001),
32);
           state <= state_3;
         WHEN state_3 =>
           v003_mult <= conv_std_logic_vector(CONV_INTEGER(var1) / CONV_INTEGER(const1), 32);
           state <= state_4;
         WHEN state_4 =>
           v004_mult_int <= v003_mult and temporary002;
           state <= state_5;
         WHEN state_5 =>
           fp_mult(31 DOWNTO 0) <= v004_mult_int(31 DOWNTO 0) ;
           state <= state_0;
           done_int <= '1';   -- processing finished, results are ready !!!
           busy <= '0';

        WHEN OTHERS => state <= state_0;
                       done_int <= '0';
                       busy <= '0';

       END CASE ;

     END IF ;
    END PROCESS fsm_core ;



--  Datapath operators (Functional Units) --
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  division_div32_1_out:
   WITH state SELECT
   div32_1_out <= conv_std_logic_vector(CONV_INTEGER(var1) / CONV_INTEGER(const1), 32) WHEN
state_3,
```

```
              (OTHERS => '0') WHEN OTHERS;
  and32_1_out <= and32_1_in1 and and32_1_in2;


-- now the data routing --

-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary000), 32) WHEN state_2,
                       (OTHERS => '0') WHEN OTHERS;
  routing_mult32_1_in2:
   WITH state SELECT
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary001), 32) WHEN state_2,
                       (OTHERS => '0') WHEN OTHERS;



-- now the logical AND --
  routing_and32_1_in1:
   WITH state SELECT
   and32_1_in1 <= v003_mult WHEN state_4,
                       (OTHERS => '0') WHEN OTHERS;
  routing_and32_1_in2:
   WITH state SELECT
   and32_1_in2 <= temporary002 WHEN state_4,
                       (OTHERS => '0') WHEN OTHERS;


  END  rtl ;
```

Fixed Point Division

Without PARCS optimization

```
-----------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::: performed on design module: 'fp_div'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:46 ____ :4

--::: The C-cubed compiler, back-end version: CCC_be_6 ::::--
--::::::: Copyright(c) 2007-2020, by Michael F. Dossis  :::::::--
-----------------------------------------------------------------
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_signed.ALL;

PACKAGE new_func IS

 TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

 TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

END new_func;

PACKAGE BODY new_func IS

END new_func;

LIBRARY IEEE;

USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_signed.ALL;
LIBRARY WORK;
USE WORK.new_func.ALL;

ENTITY fp_div IS
 port(
      clock, reset, start, results_read : IN std_logic;
      numa : IN  std_logic_vector(31 DOWNTO 0);

      numb : IN  std_logic_vector(31 DOWNTO 0);

      fp_div : OUT  std_logic_vector(31 DOWNTO 0);

      done, busy : OUT std_logic
     );
END fp_div ;


ARCHITECTURE rtl OF fp_div IS

 SIGNAL done_int : std_logic;
```

```
   TYPE states_type IS (state_7,
                        state_6, state_5, state_4, state_3, state_2,
                        state_1, state_0);
   SIGNAL state : states_type;  -- this stores the current and next state of the circuit

   SIGNAL v003_div :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v004_div_int :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary001 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary002 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(65536,
32));  -- integer constants are converted into std_logic
   SIGNAL var1 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(-1, 32));  -
- integer constants are converted into std_logic
   SIGNAL var2 : std_logic;

     -- now the datapath (input/output) signal declarations follow --

   SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL div32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL and32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL and32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL and32_1_in2 : std_logic_vector(31 DOWNTO 0);

  BEGIN

   done <= done_int;

    --- FSM and controller loging (plus massively-parallel ops) ---
    fsm_core : PROCESS (clock, reset)
     BEGIN
      IF reset = '0' THEN
       done_int <= '0';
       busy <= '0';
       state <= state_0;
       fp_div <= (OTHERS => '0');
       v003_div <= (OTHERS => '0');
       v004_div_int <= (OTHERS => '0');
       temporary000 <= (OTHERS => '0');
       temporary001 <= (OTHERS => '0');
```

```
 temporary002 <= (OTHERS => '0');
 var1 <= (OTHERS => '0');
 var2 <= '0';


ELSIF clock = '1' AND clock'EVENT THEN

  CASE state IS

    WHEN state_0 =>
     IF results_read = '1' THEN done_int <= '0'; END IF;
     IF start = '1' THEN
      IF done_int = '0' OR results_read = '1' THEN
       busy <= '1';   -- processing started
       state <= state_1;
      END IF;
     ELSE
      state <= state_0;
     END IF ;

    WHEN state_1 =>
      state <= state_2;
      temporary000(31 DOWNTO 0) <= numa(31 DOWNTO 0) ;

    WHEN state_2 =>
      state <= state_3;
      temporary001(31 DOWNTO 0) <= numb(31 DOWNTO 0) ;

    WHEN state_3 =>
      state <= state_4;
      var1 <= mult32_1_out;

    WHEN state_4 =>
      state <= state_5;
      v003_div <= div32_1_out;

    WHEN state_5 =>
      state <= state_6;
      v004_div_int <= and32_1_out;

    WHEN state_6 =>
     state <= state_0;
     done_int <= '1';   -- processing finished, results are ready !!!
     busy <= '0';
```

```
        WHEN OTHERS => state <= state_0;
                       done_int <= '0';
                       busy <= '0';

       END CASE ;

     END IF ;
    END PROCESS fsm_core ;



--  Datapath operators (Functional Units) --
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  division_div32_1_out:
   WITH state SELECT
   div32_1_out <= conv_std_logic_vector(CONV_INTEGER(var1) / CONV_INTEGER(temporary001), 32) WHEN
state_4,
              (OTHERS => '0') WHEN OTHERS;
  and32_1_out <= and32_1_in1 and and32_1_in2;


-- now the data routing --

-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary000), 32) WHEN state_3,
                       (OTHERS => '0') WHEN OTHERS;
  routing_mult32_1_in2:
   WITH state SELECT
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_3,
                       (OTHERS => '0') WHEN OTHERS;



-- now the logical AND --
  routing_and32_1_in1:
   WITH state SELECT
   and32_1_in1 <= v003_div WHEN state_5,
                       (OTHERS => '0') WHEN OTHERS;
  routing_and32_1_in2:
   WITH state SELECT
   and32_1_in2 <= temporary002 WHEN state_5,
                       (OTHERS => '0') WHEN OTHERS;
```

```
END  rtl ;
```

With PARCS optimization

```
------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::::::: Optimiser engine used: 'PARCS' ::::::::::::::--

--:::::::::: performed on design module: 'fp_div'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:46 ____ :8

--::: The C-cubed compiler, back-end version: CCC_be_6 ::::--
--::::::: Copyright(c) 2007-2020, by Michael F. Dossis  ::::::::--
------------------------------------------------------------


  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;

  PACKAGE new_func IS

   TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

   TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

  END new_func;

  PACKAGE BODY new_func IS

  END new_func;



  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;
  LIBRARY WORK;
```

```
 USE WORK.new_func.ALL;

 ENTITY fp_div IS
  port(
       clock, reset, start, results_read : IN std_logic;
       numa : IN  std_logic_vector(31 DOWNTO 0);
       numb : IN  std_logic_vector(31 DOWNTO 0);
       fp_div : OUT  std_logic_vector(31 DOWNTO 0);
       done, busy : OUT std_logic
      );
 END fp_div ;


 ARCHITECTURE rtl OF fp_div IS

  SIGNAL done_int : std_logic;

  TYPE states_type IS (state_4,
                       state_3, state_2, state_1, state_0);
  SIGNAL state : states_type;  -- this stores the current and next state of the circuit

  SIGNAL v003_div :  std_logic_vector(31 DOWNTO 0);
  SIGNAL v004_div_int :  std_logic_vector(31 DOWNTO 0);
  SIGNAL temporary000 :  std_logic_vector(31 DOWNTO 0);
  SIGNAL temporary001 :  std_logic_vector(31 DOWNTO 0);
  SIGNAL temporary002 :  std_logic_vector(31 DOWNTO 0);
  CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(65536,
32));  -- integer constants are converted into std_logic
  SIGNAL var1 :  std_logic_vector(31 DOWNTO 0);
  CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(-1, 32));  -
- integer constants are converted into std_logic
  SIGNAL var2 : std_logic;

    -- now the datapath (input/output) signal declarations follow --

  SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL div32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL div32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL div32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL and32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL and32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL and32_1_in2 : std_logic_vector(31 DOWNTO 0);
```

```
BEGIN

 done <= done_int;

  --- FSM and controller loging (plus massively-parallel ops) ---
 fsm_core : PROCESS (clock, reset)
  BEGIN
   IF reset = '0' THEN
    done_int <= '0';
    busy <= '0';
    state <= state_0;
    fp_div <= (OTHERS => '0');
    v003_div <= (OTHERS => '0');
    v004_div_int <= (OTHERS => '0');
    temporary000 <= (OTHERS => '0');
    temporary001 <= (OTHERS => '0');
    temporary002 <= (OTHERS => '0');
    var1 <= (OTHERS => '0');
    var2 <= '0';


   ELSIF clock = '1' AND clock'EVENT THEN

     CASE state IS

       WHEN state_0 =>
        IF results_read = '1' THEN done_int <= '0'; END IF;
        IF start = '1' THEN
         IF done_int = '0' OR results_read = '1' THEN
          busy <= '1';   -- processing started
          state <= state_1;
         END IF;
        ELSE
         state <= state_0;
        END IF ;

       WHEN state_1 =>
         temporary001(31 DOWNTO 0) <= numb(31 DOWNTO 0) ;
         temporary000(31 DOWNTO 0) <= numa(31 DOWNTO 0) ;
         state <= state_2;
       WHEN state_2 =>
         var1 <= conv_std_logic_vector(CONV_INTEGER(temporary000) * CONV_INTEGER(const1), 32);
         state <= state_3;
       WHEN state_3 =>
         v003_div <= conv_std_logic_vector(CONV_INTEGER(var1) / CONV_INTEGER(temporary001), 32);
```

```
            state <= state_4;
          WHEN state_4 =>
            fp_div(31 DOWNTO 0) <= v003_div(31 DOWNTO 0) ;
            v004_div_int <= v003_div and temporary002;
            state <= state_0;
            done_int <= '1';   -- processing finished, results are ready !!!
            busy <= '0';

         WHEN OTHERS => state <= state_0;
                        done_int <= '0';
                        busy <= '0';

        END CASE ;

      END IF ;
     END PROCESS fsm_core ;



--  Datapath operators (Functional Units) --
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  division_div32_1_out:
   WITH state SELECT
   div32_1_out <= conv_std_logic_vector(CONV_INTEGER(var1) / CONV_INTEGER(temporary001), 32) WHEN
state_3,
               (OTHERS => '0') WHEN OTHERS;
  and32_1_out <= and32_1_in1 and and32_1_in2;


-- now the data routing --

-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary000), 32) WHEN state_2,
                       (OTHERS => '0') WHEN OTHERS;
  routing_mult32_1_in2:
   WITH state SELECT
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const1), 32) WHEN state_2,
                       (OTHERS => '0') WHEN OTHERS;



-- now the logical AND --
```

```
  routing_and32_1_in1:
   WITH state SELECT
   and32_1_in1 <= v003_div WHEN state_4,
                       (OTHERS => '0') WHEN OTHERS;
  routing_and32_1_in2:
   WITH state SELECT
   and32_1_in2 <= temporary002 WHEN state_4,
                       (OTHERS => '0') WHEN OTHERS;


  END  rtl ;
```

Fixed Point Exponential

Without PARCS optimization

```
------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::: performed on design module: 'exp_int'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:46 ____ :51

--::: The C-cubed compiler, back-end version: CCC_be_6 ::::--
--::::::: Copyright(c) 2007-2020, by Michael F. Dossis  :::::::--
------------------------------------------------------------


  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;

  PACKAGE new_func IS

   TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

   TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

  END new_func;

  PACKAGE BODY new_func IS

  END new_func;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_signed.ALL;
LIBRARY WORK;
USE WORK.new_func.ALL;

ENTITY exp_int IS
 port(
      clock, reset, start, results_read : IN std_logic;
      pow_int_results_read :  OUT std_logic;
      pow_int_start :  OUT std_logic;
      pow_int_done :  IN std_logic;
      pow_int_busy :  IN std_logic;
      fp_div_results_read :  OUT std_logic;
      fp_div_start :  OUT std_logic;
      fp_div_done :  IN std_logic;
      fp_div_busy :  IN std_logic;
      pow : IN  std_logic_vector(31 DOWNTO 0);

      exp_int : OUT  std_logic_vector(31 DOWNTO 0);

      pow_int_num : OUT  std_logic_vector(31 DOWNTO 0);

      pow_int_pow : OUT  std_logic_vector(31 DOWNTO 0);

      pow_int_pow_int : IN  std_logic_vector(31 DOWNTO 0);

      fp_div_numa : OUT  std_logic_vector(31 DOWNTO 0);

      fp_div_numb : OUT  std_logic_vector(31 DOWNTO 0);

      fp_div_fp_div : IN  std_logic_vector(31 DOWNTO 0);

      done, busy : OUT std_logic
     );
END exp_int ;


ARCHITECTURE rtl OF exp_int IS

 SIGNAL done_int : std_logic;
```

```
   TYPE states_type IS (state_33,
                        state_32, state_31, state_30, state_29, state_28,
                        state_27, state_26, state_25, state_24, state_23,
                        state_22, state_21, state_20, state_19, state_18,
                        state_17, state_16, state_15, state_14, state_13,
                        state_12, state_11, state_10, state_9, state_8,
                        state_7, state_6, state_5, state_4, state_3,
                        state_2, state_1, state_0);
   SIGNAL state : states_type;  -- this stores the current and next state of the circuit

   SIGNAL temporary_pow :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v002_num :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v003_fact :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v004_neg :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v005_i :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary001 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL index000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL tempint000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL funretvalue :  std_logic_vector(31 DOWNTO 0);
   SIGNAL funnoreturn : std_logic;
   CONSTANT const1 :  std_logic := '0';  -- integer constants are converted into std_logic
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(65536,
32));  -- integer constants are converted into std_logic
   CONSTANT const3 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(1, 32));  --
integer constants are converted into std_logic
   CONSTANT const4 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(0, 32));  --
integer constants are converted into std_logic
   CONSTANT const5 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(-905412,
32));  -- integer constants are converted into std_logic
   SIGNAL var1 : std_logic;
   CONSTANT const6 :  std_logic := '0';  -- integer constants are converted into std_logic
   SIGNAL var2 : std_logic;
   CONSTANT const7 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(12, 32));  -
- integer constants are converted into std_logic
   SIGNAL var3 : std_logic;
   SIGNAL var4 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var5 : std_logic;
   SIGNAL pow_int_results_read_int :  std_logic;
   SIGNAL pow_int_start_int :  std_logic;
   SIGNAL fp_div_results_read_int :  std_logic;
   SIGNAL fp_div_start_int :  std_logic;

    -- now the datapath (input/output) signal declarations follow --
```

```
 SIGNAL less32_1_out : std_logic;
 SIGNAL less32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL less32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL minus32_1_out : std_logic_vector(31 DOWNTO 0);
 SIGNAL minus32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL lessequal32_1_out : std_logic;
 SIGNAL lessequal32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL lessequal32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
 SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL div32_1_out : std_logic_vector(31 DOWNTO 0);
 SIGNAL div32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL div32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL plus32_1_out : std_logic_vector(31 DOWNTO 0);
 SIGNAL plus32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL plus32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL plusone32_1_out : std_logic_vector(31 DOWNTO 0);
 SIGNAL plusone32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL plusone32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL equal32_1_out : std_logic;
 SIGNAL equal32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL equal32_1_in2 : std_logic_vector(31 DOWNTO 0);

BEGIN

 done <= done_int;
 pow_int_start <= pow_int_start_int;
 pow_int_results_read <= pow_int_results_read_int;
 fp_div_start <= fp_div_start_int;
 fp_div_results_read <= fp_div_results_read_int;

  --- FSM and controller loging (plus massively-parallel ops) ---
 fsm_core : PROCESS (clock, reset)
     VARIABLE var1_conditional_variable :  std_logic;
     VARIABLE funnoreturn_conditional_variable :  std_logic;
     VARIABLE var2_conditional_variable :  std_logic;
     VARIABLE var3_conditional_variable :  std_logic;
     VARIABLE var5_conditional_variable :  std_logic;
  BEGIN
   IF reset = '0' THEN
    done_int <= '0';
    busy <= '0';
    state <= state_0;
```

```
    exp_int <= (OTHERS => '0');
    temporary_pow <= (OTHERS => '0');
    v002_num <= (OTHERS => '0');
    v003_fact <= (OTHERS => '0');
    v004_neg <= (OTHERS => '0');
    v005_i <= (OTHERS => '0');
    temporary000 <= (OTHERS => '0');
    temporary001 <= (OTHERS => '0');
    index000 <= (OTHERS => '0');
    tempint000 <= (OTHERS => '0');
    funretvalue <= (OTHERS => '0');
    funnoreturn <= '0';
    var1 <= '0';
    var2 <= '0';
    var3 <= '0';
    var4 <= (OTHERS => '0');
    var5 <= '0';
    pow_int_num <= (OTHERS => '0');
    pow_int_pow <= (OTHERS => '0');
    fp_div_numa <= (OTHERS => '0');
    fp_div_numb <= (OTHERS => '0');
    pow_int_results_read_int <= '0';
    pow_int_start_int <= '0';
    fp_div_results_read_int <= '0';
    fp_div_start_int <= '0';


  ELSIF clock = '1' AND clock'EVENT THEN

    CASE state IS

      WHEN state_0 =>
       IF results_read = '1' THEN done_int <= '0'; END IF;
       IF start = '1' THEN
        IF done_int = '0' OR results_read = '1' THEN
         busy <= '1';   -- processing started
         state <= state_1;
        END IF;
       ELSE
        state <= state_0;
       END IF ;

      WHEN state_1 =>
        state <= state_2;
        funnoreturn <= const1 ;
```

```
WHEN state_2 =>
  state <= state_3;
  temporary_pow(31 DOWNTO 0) <= pow(31 DOWNTO 0) ;

WHEN state_3 =>
  state <= state_4;
  v002_num(31 DOWNTO 0) <= const2(31 DOWNTO 0) ;

WHEN state_4 =>
  state <= state_5;
  v003_fact(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

WHEN state_5 =>
  state <= state_6;
  v004_neg(31 DOWNTO 0) <= const4(31 DOWNTO 0) ;

WHEN state_6 =>
  state <= state_7;
  var1 <= less32_1_out;


WHEN state_7 =>
IF var1 = '1' THEN
   state <= state_8;
  ELSE
   state <= state_10;
  END IF;

WHEN state_8 =>
  state <= state_9;
  funretvalue(31 DOWNTO 0) <= const4(31 DOWNTO 0) ;

WHEN state_9 =>
  state <= state_10;
  funnoreturn <= const6 ;


WHEN state_10 =>
IF funnoreturn = '1' THEN
   state <= state_11;
  ELSE
   state <= state_32;
  END IF;
```

```
        WHEN state_11 =>
          state <= state_12;
          var2 <= less32_1_out;


        WHEN state_12 =>
        IF var2 = '1' THEN
           state <= state_13;
          ELSE
           state <= state_15;
          END IF;

        WHEN state_13 =>
          state <= state_14;
          v004_neg(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

        WHEN state_14 =>
          state <= state_15;
          temporary_pow <= minus32_1_out;

        WHEN state_15 =>
          state <= state_16;
          v005_i(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

        WHEN state_16 =>
          state <= state_17;
          v005_i(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

        WHEN state_17 =>
          state <= state_18;
          tempint000(31 DOWNTO 0) <= const7(31 DOWNTO 0) ;

        WHEN state_18 =>
          state <= state_19;
          index000(31 DOWNTO 0) <= const4(31 DOWNTO 0) ;

        WHEN state_19 =>
          state <= state_20;
          var3 <= lessequal32_1_out;


        WHEN state_20 =>
        IF var3 = '1' THEN
           state <= state_21;
          ELSE
```

```
    state <= state_28;
  END IF;

WHEN state_21 =>
  state <= state_22;
  v003_fact <= mult32_1_out;

WHEN state_22 =>
  state <= state_23;


WHEN state_23 =>
  state <= state_24;
  var4 <= div32_1_out;

WHEN state_24 =>
  state <= state_25;
  v002_num <= plus32_1_out;

WHEN state_25 =>
  state <= state_26;
  v005_i <= plus32_1_out;

WHEN state_26 =>
  state <= state_27;
  index000 <= plusone32_1_out;

WHEN state_27 =>
  state <= state_19;


WHEN state_28 =>
  state <= state_29;
  var5 <= equal32_1_out;


WHEN state_29 =>
IF var5 = '1' THEN
   state <= state_30;
  ELSE
   state <= state_31;
  END IF;

WHEN state_30 =>
  state <= state_31;
```

```
          WHEN state_31 =>
            state <= state_32;
            funretvalue(31 DOWNTO 0) <= v002_num(31 DOWNTO 0) ;

          WHEN state_32 =>
           state <= state_0;
           done_int <= '1';    -- processing finished, results are ready !!!
           busy <= '0';

         WHEN OTHERS => state <= state_0;
                        done_int <= '0';
                        busy <= '0';

        END CASE ;

      END IF ;
     END PROCESS fsm_core ;



--  Datapath operators (Functional Units) --
  less32_1_out <= '1' WHEN less32_1_in1 < less32_1_in2 ELSE '0';
  minus32_1_out <=- minus32_1_in2;
  lessequal32_1_out <= '1' WHEN lessequal32_1_in1 <= lessequal32_1_in2 ELSE '0';
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  division_div32_1_out:
   WITH state SELECT
   div32_1_out <= conv_std_logic_vector(CONV_INTEGER(temporary001) / CONV_INTEGER(v003_fact), 32)
WHEN state_23,
               (OTHERS => '0') WHEN OTHERS;
  plus32_1_out <= plus32_1_in1 + plus32_1_in2;
  plusone32_1_out <= plusone32_1_in1 + plusone32_1_in2;
  equal32_1_out <= '1' WHEN equal32_1_in1 = equal32_1_in2 ELSE '0';


-- now the data routing --

-- now the less-than comparisons --
  routing_less32_1_in1:
   WITH state SELECT
   less32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_6,
                       conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_11,
```

```
                      (OTHERS => '0') WHEN OTHERS;
  routing_less32_1_in2:
   WITH state SELECT
   less32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary000), 32) WHEN state_6,
                   conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_11,
                      (OTHERS => '0') WHEN OTHERS;


-- now the subtractions --
  routing_minus32_1_in2:
   WITH state SELECT
   minus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_14,
                      (OTHERS => '0') WHEN OTHERS;


-- now the less-equality comparisons --
  routing_lessequal32_1_in1:
   WITH state SELECT
   lessequal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_19,
                      (OTHERS => '0') WHEN OTHERS;
  routing_lessequal32_1_in2:
   WITH state SELECT
   lessequal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(tempint000), 32) WHEN state_19,
                      (OTHERS => '0') WHEN OTHERS;


-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v003_fact), 32) WHEN state_21,
                      (OTHERS => '0') WHEN OTHERS;
  routing_mult32_1_in2:
   WITH state SELECT
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v005_i), 32) WHEN state_21,
                      (OTHERS => '0') WHEN OTHERS;



-- now the additions --
  routing_plus32_1_in1:
   WITH state SELECT
   plus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v002_num), 32) WHEN state_24,
                   conv_std_logic_vector(CONV_INTEGER(v005_i), 32) WHEN state_25,
                      (OTHERS => '0') WHEN OTHERS;
  routing_plus32_1_in2:
```

```
   WITH state SELECT
   plus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(var4), 32) WHEN state_24,
                   conv_std_logic_vector(CONV_INTEGER(const3), 32) WHEN state_25,
                    (OTHERS => '0') WHEN OTHERS;


-- now the increments by one --
  routing_plusone32_1_in1:
   WITH state SELECT
   plusone32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_26,
                    (OTHERS => '0') WHEN OTHERS;
  routing_plusone32_1_in2:
   WITH state SELECT
   plusone32_1_in2 <= conv_std_logic_vector(1, 32) WHEN state_26,
                    (OTHERS => '0') WHEN OTHERS;


-- now the equality comparisons --
  routing_equal32_1_in1:
   WITH state SELECT
   equal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v004_neg), 32) WHEN state_28,
                    (OTHERS => '0') WHEN OTHERS;
  routing_equal32_1_in2:
   WITH state SELECT
   equal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const3), 32) WHEN state_28,
                    (OTHERS => '0') WHEN OTHERS;


  END  rtl ;
```

With PARCS optimization

```
-----------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::::::: Optimiser engine used: 'PARCS' ::::::::::::::--

--:::::::::: performed on design module: 'exp_int'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:46 ____ :66

--::: The C-cubed compiler, back-end version: CCC_be_6 ::::--
--::::::: Copyright(c) 2007-2020, by Michael F. Dossis  ::::::::--
-----------------------------------------------------------------
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_signed.ALL;

PACKAGE new_func IS

 TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

 TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

END new_func;

PACKAGE BODY new_func IS

END new_func;



LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_signed.ALL;
LIBRARY WORK;
USE WORK.new_func.ALL;

ENTITY exp_int IS
 port(
      clock, reset, start, results_read : IN std_logic;
      pow_int_results_read :  OUT std_logic;
      pow_int_start :  OUT std_logic;
      pow_int_done :  IN std_logic;
      pow_int_busy :  IN std_logic;
      fp_div_results_read :  OUT std_logic;
      fp_div_start :  OUT std_logic;
      fp_div_done :  IN std_logic;
      fp_div_busy :  IN std_logic;
      pow : IN  std_logic_vector(31 DOWNTO 0);
      exp_int : OUT  std_logic_vector(31 DOWNTO 0);
      pow_int_num : OUT  std_logic_vector(31 DOWNTO 0);
      pow_int_pow : OUT  std_logic_vector(31 DOWNTO 0);
      pow_int_pow_int : IN  std_logic_vector(31 DOWNTO 0);
      fp_div_numa : OUT  std_logic_vector(31 DOWNTO 0);
      fp_div_numb : OUT  std_logic_vector(31 DOWNTO 0);
      fp_div_fp_div : IN  std_logic_vector(31 DOWNTO 0);
```

```
       done, busy : OUT std_logic
      );
  END exp_int ;


  ARCHITECTURE rtl OF exp_int IS

   SIGNAL done_int : std_logic;

   TYPE states_type IS (state_15,
                        state_14, state_13, state_12, state_11, state_10,
                        state_9, state_8, state_7, state_6, state_5,
                        state_4, state_3, state_2, state_1, state_0);
   SIGNAL state : states_type;  -- this stores the current and next state of the circuit

   SIGNAL temporary_pow :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v002_num :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v003_fact :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v004_neg :  std_logic_vector(31 DOWNTO 0);
   SIGNAL v005_i :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary001 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL index000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL tempint000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL funretvalue :  std_logic_vector(31 DOWNTO 0);
   SIGNAL funnoreturn : std_logic;
   CONSTANT const1 :  std_logic := '0';  -- integer constants are converted into std_logic
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(65536,
32));  -- integer constants are converted into std_logic
   CONSTANT const3 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(1, 32));  --
integer constants are converted into std_logic
   CONSTANT const4 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(0, 32));  --
integer constants are converted into std_logic
   CONSTANT const5 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(-905412,
32));  -- integer constants are converted into std_logic
   SIGNAL var1 : std_logic;
   CONSTANT const6 :  std_logic := '0';  -- integer constants are converted into std_logic
   SIGNAL var2 : std_logic;
   CONSTANT const7 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(12, 32));  -
- integer constants are converted into std_logic
   SIGNAL var3 : std_logic;
   SIGNAL var4 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var5 : std_logic;
   SIGNAL pow_int_results_read_int : std_logic;
   SIGNAL pow_int_start_int : std_logic;
```

```
  SIGNAL fp_div_results_read_int : std_logic;
  SIGNAL fp_div_start_int : std_logic;

    -- now the datapath (input/output) signal declarations follow --

  SIGNAL less32_1_out : std_logic;
  SIGNAL less32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL less32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL minus32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL minus32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL lessequal32_1_out : std_logic;
  SIGNAL lessequal32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL lessequal32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL div32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL div32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL div32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_2_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_2_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_2_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plusone32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL plusone32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plusone32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL equal32_1_out : std_logic;
  SIGNAL equal32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL equal32_1_in2 : std_logic_vector(31 DOWNTO 0);

BEGIN

  done <= done_int;
  pow_int_start <= pow_int_start_int;
  pow_int_results_read <= pow_int_results_read_int;
  fp_div_start <= fp_div_start_int;
  fp_div_results_read <= fp_div_results_read_int;

   --- FSM and controller loging (plus massively-parallel ops) ---
   fsm_core : PROCESS (clock, reset)
       VARIABLE var1_conditional_variable :  std_logic;
       VARIABLE funnoreturn_conditional_variable :  std_logic;
       VARIABLE var2_conditional_variable :  std_logic;
```

```
    VARIABLE var3_conditional_variable :  std_logic;
    VARIABLE var5_conditional_variable :  std_logic;
TYPE STATES_ARRAY IS ARRAY (integer RANGE <>) OF STATES_TYPE;
VARIABLE state_var : STATES_ARRAY(1 DOWNTO 1);
BEGIN
 IF reset = '0' THEN
  done_int <= '0';
  busy <= '0';
  state <= state_0;
  exp_int <= (OTHERS => '0');
  temporary_pow <= (OTHERS => '0');
  v002_num <= (OTHERS => '0');
  v003_fact <= (OTHERS => '0');
  v004_neg <= (OTHERS => '0');
  v005_i <= (OTHERS => '0');
  temporary000 <= (OTHERS => '0');
  temporary001 <= (OTHERS => '0');
  index000 <= (OTHERS => '0');
  tempint000 <= (OTHERS => '0');
  funretvalue <= (OTHERS => '0');
  funnoreturn <= '0';
  var1 <= '0';
  var2 <= '0';
  var3 <= '0';
  var4 <= (OTHERS => '0');
  var5 <= '0';

  pow_int_results_read_int <= '0';
  pow_int_start_int <= '0';
  pow_int_num <= (OTHERS => '0');
  pow_int_pow <= (OTHERS => '0');
  fp_div_results_read_int <= '0';
  fp_div_start_int <= '0';
  fp_div_numa <= (OTHERS => '0');
  fp_div_numb <= (OTHERS => '0');

 ELSIF clock = '1' AND clock'EVENT THEN

   CASE state IS

     WHEN state_0 =>
      IF results_read = '1' THEN done_int <= '0'; END IF;
      IF start = '1' THEN
       IF done_int = '0' OR results_read = '1' THEN
        busy <= '1';    -- processing started
```

```
    state <= state_1;
   END IF;
  ELSE
   state <= state_0;
  END IF ;

 WHEN state_1 =>
   v004_neg(31 DOWNTO 0) <= const4(31 DOWNTO 0) ;
   v003_fact(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;
   v002_num(31 DOWNTO 0) <= const2(31 DOWNTO 0) ;
   temporary_pow(31 DOWNTO 0) <= pow(31 DOWNTO 0) ;
   funnoreturn <= const1 ;
   state <= state_2;
 WHEN state_2 =>
   IF temporary_pow < temporary000 THEN var1 <= '1'; ELSE var1 <= '0'; END IF;
   state <= state_3;
 WHEN state_3 =>
   IF var1 = '1' THEN
   state <= state_4;
   ELSE
   state <= state_4;
   END IF;
   IF var1 = '1' THEN
   funretvalue(31 DOWNTO 0) <= const4(31 DOWNTO 0) ;
   funnoreturn <= const6 ;
   ELSE
   END IF;
 WHEN state_4 =>
   IF funnoreturn = '1' THEN
   state <= state_5;
   ELSE
   state <= state_15;
   END IF;
   IF funnoreturn = '1' THEN
   var2 <= less32_1_out;
   ELSE
   END IF;
 WHEN state_5 =>
   IF var2 = '1' THEN
   state <= state_6;
   ELSE
   state <= state_6;
   END IF;
   IF var2 = '1' THEN
   v004_neg(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;
```

```
            temporary_pow <= minus32_1_out;
            ELSE
            END IF;
          WHEN state_6 =>
            v005_i(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;
            state <= state_7;
          WHEN state_7 =>
            v005_i(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;
            tempint000(31 DOWNTO 0) <= const7(31 DOWNTO 0) ;
            index000(31 DOWNTO 0) <= const4(31 DOWNTO 0) ;
            state <= state_8;
          WHEN state_8 =>
            IF index000 <= tempint000 THEN var3 <= '1'; ELSE var3 <= '0'; END IF;
            state <= state_9;
          WHEN state_9 =>
            IF var3 = '1' THEN
            state <= state_10;
            ELSE
            state <= state_12;
            END IF;
            IF var3 = '1' THEN
            v003_fact <= mult32_1_out;

             ------ this is a call to module : pow_int -----
           pow_int_num <= temporary_pow;
           pow_int_pow <= v005_i;

                        IF pow_int_busy = '0' AND pow_int_done = '0' THEN
                         IF pow_int_results_read_int = '1' THEN
                                      pow_int_start_int <= '0';
                                      pow_int_results_read_int <= '0';
                           state <= state_4;
                          ELSE
                           pow_int_start_int <= '1';
                           state <= state_9;
                          END IF;
                        ELSIF pow_int_busy = '1' AND pow_int_start_int = '1'  THEN   -- when it
begins
                         pow_int_start_int <= '0';
                         state <= state_9;
                        ELSIF pow_int_done = '1'  THEN   -- when it is completed then read the
results
            temporary001 <= pow_int_pow_int;
                         pow_int_start_int <= '0';
```

```
                        IF pow_int_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                         pow_int_results_read_int <= '1';
                         state <= state_9;
                        ELSIF              pow_int_results_read_int = '1'               THEN --
all calls are synchronized and completed
                                     pow_int_results_read_int <= '0';
                         state <= state_4;
                        END IF;
                       ELSE
                        state <= state_9;
                       END IF;
            ELSE
            END IF;
          WHEN state_10 =>
            var4 <= conv_std_logic_vector(CONV_INTEGER(temporary001) / CONV_INTEGER(v003_fact),
32);
            state <= state_11;
          WHEN state_11 =>
            v002_num <= v002_num + var4;
            v005_i <= v005_i + const3;
            index000 <= index000 + 1;

            state <= state_8;
          WHEN state_12 =>
            IF v004_neg = const3 THEN var5 <= '1'; ELSE var5 <= '0'; END IF;
            state <= state_13;
          WHEN state_13 =>
            IF var5 = '1' THEN
            state <= state_14;
            ELSE
            state <= state_14;
            END IF;
            IF var5 = '1' THEN

             ------ this is a call to module : fp_div -----
           fp_div_numa <= const2;
           fp_div_numb <= v002_num;

                       IF fp_div_busy = '0' AND fp_div_done = '0' THEN
                        IF fp_div_results_read_int = '1' THEN
                                     fp_div_start_int <= '0';
                                     fp_div_results_read_int <= '0';
                          state <= state_4;
                         ELSE
```

```
                          fp_div_start_int <= '1';
                          state <= state_13;
                         END IF;
                       ELSIF fp_div_busy = '1' AND fp_div_start_int = '1'  THEN   -- when it
begins
                        fp_div_start_int <= '0';
                        state <= state_13;
                       ELSIF fp_div_done = '1'   THEN   -- when it is completed then read the
results
           v002_num <= fp_div_fp_div;
                        fp_div_start_int <= '0';
                        IF fp_div_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                         fp_div_results_read_int <= '1';
                         state <= state_13;
                         ELSIF              fp_div_results_read_int = '1'               THEN --
all calls are synchronized and completed
                                      fp_div_results_read_int <= '0';
                         state <= state_4;
                         END IF;
                       ELSE
                        state <= state_13;
                       END IF;
            ELSE
            END IF;
          WHEN state_14 =>
            funretvalue(31 DOWNTO 0) <= v002_num(31 DOWNTO 0) ;
            state <= state_15;
          WHEN state_15 =>
            exp_int(31 DOWNTO 0) <= funretvalue(31 DOWNTO 0) ;
            state <= state_0;
            done_int <= '1';   -- processing finished, results are ready !!!
            busy <= '0';

         WHEN OTHERS => state <= state_0;
                        done_int <= '0';
                        busy <= '0';

        END CASE ;

      END IF ;
     END PROCESS fsm_core ;
```

```
--  Datapath operators (Functional Units) --
  less32_1_out <= '1' WHEN less32_1_in1 < less32_1_in2 ELSE '0';
  minus32_1_out <=- minus32_1_in2;
  lessequal32_1_out <= '1' WHEN lessequal32_1_in1 <= lessequal32_1_in2 ELSE '0';
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  division_div32_1_out:
   WITH state SELECT
   div32_1_out <= conv_std_logic_vector(CONV_INTEGER(temporary001) / CONV_INTEGER(v003_fact), 32)
WHEN state_10,
               (OTHERS => '0') WHEN OTHERS;
  plus32_1_out <= plus32_1_in1 + plus32_1_in2;
  plus32_2_out <= plus32_2_in1 + plus32_2_in2;
  plusone32_1_out <= plusone32_1_in1 + plusone32_1_in2;
  equal32_1_out <= '1' WHEN equal32_1_in1 = equal32_1_in2 ELSE '0';


-- now the data routing --

-- now the less-than comparisons --
  routing_less32_1_in1:
   WITH state SELECT
   less32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_2,
                     conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_4,
                      (OTHERS => '0') WHEN OTHERS;
  routing_less32_1_in2:
   WITH state SELECT
   less32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary000), 32) WHEN state_2,
                     conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_4,
                      (OTHERS => '0') WHEN OTHERS;


-- now the subtractions --
  routing_minus32_1_in2:
   WITH state SELECT
   minus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary_pow), 32) WHEN state_5,
                     (OTHERS => '0') WHEN OTHERS;


-- now the less-equality comparisons --
  routing_lessequal32_1_in1:
   WITH state SELECT
   lessequal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_8,
                     (OTHERS => '0') WHEN OTHERS;
  routing_lessequal32_1_in2:
```

```
   WITH state SELECT
   lessequal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(tempint000), 32) WHEN state_8,
                        (OTHERS => '0') WHEN OTHERS;


-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v003_fact), 32) WHEN state_9,
                        (OTHERS => '0') WHEN OTHERS;
  routing_mult32_1_in2:
   WITH state SELECT
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v005_i), 32) WHEN state_9,
                        (OTHERS => '0') WHEN OTHERS;



-- now the additions --
  routing_plus32_1_in1:
   WITH state SELECT
   plus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v002_num), 32) WHEN state_11,
                        (OTHERS => '0') WHEN OTHERS;
  routing_plus32_1_in2:
   WITH state SELECT
   plus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(var4), 32) WHEN state_11,
                        (OTHERS => '0') WHEN OTHERS;


-- now the additions --
  routing_plus32_2_in1:
   WITH state SELECT
   plus32_2_in1 <= conv_std_logic_vector(CONV_INTEGER(v005_i), 32) WHEN state_11,
                        (OTHERS => '0') WHEN OTHERS;
  routing_plus32_2_in2:
   WITH state SELECT
   plus32_2_in2 <= conv_std_logic_vector(CONV_INTEGER(const3), 32) WHEN state_11,
                        (OTHERS => '0') WHEN OTHERS;


-- now the increments by one --
  routing_plusone32_1_in1:
   WITH state SELECT
   plusone32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_11,
                        (OTHERS => '0') WHEN OTHERS;
  routing_plusone32_1_in2:
```

```
   WITH state SELECT
   plusone32_1_in2 <= conv_std_logic_vector(1, 32) WHEN state_11,
                      (OTHERS => '0') WHEN OTHERS;


-- now the equality comparisons --
  routing_equal32_1_in1:
   WITH state SELECT
   equal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v004_neg), 32) WHEN state_12,
                      (OTHERS => '0') WHEN OTHERS;
  routing_equal32_1_in2:
   WITH state SELECT
   equal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const3), 32) WHEN state_12,
                      (OTHERS => '0') WHEN OTHERS;


  END  rtl ;
```

Αρχική Συνάρτηση Αλγορίθμου

Without PARCS optimization

```
------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::: performed on design module: 'computeforces'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:46 ____ :82

--::: The C-cubed compiler, back-end version: CCC_be_6 ::::--
--::::::: Copyright(c) 2007-2020, by Michael F. Dossis  :::::::--
------------------------------------------------------------


  LIBRARY IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE IEEE.std_logic_signed.ALL;

  PACKAGE new_func IS

   TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

   TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);
```

```
END new_func;

PACKAGE BODY new_func IS

END new_func;



LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_signed.ALL;
LIBRARY WORK;
USE WORK.new_func.ALL;

ENTITY computeforces IS
 port(
      clock, reset, start, results_read : IN std_logic;
      pow_int_results_read :  OUT std_logic;
      pow_int_start :  OUT std_logic;
      pow_int_done :  IN std_logic;
      pow_int_busy :  IN std_logic;
      sqrt2_results_read :  OUT std_logic;
      sqrt2_start :  OUT std_logic;
      sqrt2_done :  IN std_logic;
      sqrt2_busy :  IN std_logic;
      fp_mult_results_read :  OUT std_logic;
      fp_mult_start :  OUT std_logic;
      fp_mult_done :  IN std_logic;
      fp_mult_busy :  IN std_logic;
      exp_int_results_read :  OUT std_logic;
      exp_int_start :  OUT std_logic;
      exp_int_done :  IN std_logic;
      exp_int_busy :  IN std_logic;
      fp_div_results_read :  OUT std_logic;
      fp_div_start :  OUT std_logic;
      fp_div_done :  IN std_logic;
      fp_div_busy :  IN std_logic;
      mr : IN  std_logic_vector(31 DOWNTO 0);

      mtype : IN  std_logic_vector(31 DOWNTO 0);

      computeforces : OUT  std_logic_vector(31 DOWNTO 0);
```

```
      pow_int_num : OUT  std_logic_vector(31 DOWNTO 0);

      pow_int_pow : OUT  std_logic_vector(31 DOWNTO 0);

      pow_int_pow_int : IN  std_logic_vector(31 DOWNTO 0);

      sqrt2_number : OUT  std_logic_vector(31 DOWNTO 0);

      sqrt2_sqrt2 : IN  std_logic_vector(31 DOWNTO 0);

      fp_mult_numa : OUT  std_logic_vector(31 DOWNTO 0);

      fp_mult_numb : OUT  std_logic_vector(31 DOWNTO 0);

      fp_mult_fp_mult : IN  std_logic_vector(31 DOWNTO 0);

      exp_int_pow : OUT  std_logic_vector(31 DOWNTO 0);

      exp_int_exp_int : IN  std_logic_vector(31 DOWNTO 0);

      fp_div_numa : OUT  std_logic_vector(31 DOWNTO 0);

      fp_div_numb : OUT  std_logic_vector(31 DOWNTO 0);

      fp_div_fp_div : IN  std_logic_vector(31 DOWNTO 0);

      done, busy : OUT std_logic
     );
END computeforces ;


ARCHITECTURE rtl OF computeforces IS

 SIGNAL done_int : std_logic;

 TYPE states_type IS (state_168,
                      state_167, state_166, state_165, state_164, state_163,
                      state_162, state_161, state_160, state_159, state_158,
                      state_157, state_156, state_155, state_154, state_153,
                      state_152, state_151, state_150, state_149, state_148,
                      state_147, state_146, state_145, state_144, state_143,
                      state_142, state_141, state_140, state_139, state_138,
                      state_137, state_136, state_135, state_134, state_133,
                      state_132, state_131, state_130, state_129, state_128,
                      state_127, state_126, state_125, state_124, state_123,
```

```
                    state_122, state_121, state_120, state_119, state_118,
                    state_117, state_116, state_115, state_114, state_113,
                    state_112, state_111, state_110, state_109, state_108,
                    state_107, state_106, state_105, state_104, state_103,
                    state_102, state_101, state_100, state_99, state_98,
                    state_97, state_96, state_95, state_94, state_93,
                    state_92, state_91, state_90, state_89, state_88,
                    state_87, state_86, state_85, state_84, state_83,
                    state_82, state_81, state_80, state_79, state_78,
                    state_77, state_76, state_75, state_74, state_73,
                    state_72, state_71, state_70, state_69, state_68,
                    state_67, state_66, state_65, state_64, state_63,
                    state_62, state_61, state_60, state_59, state_58,
                    state_57, state_56, state_55, state_54, state_53,
                    state_52, state_51, state_50, state_49, state_48,
                    state_47, state_46, state_45, state_44, state_43,
                    state_42, state_41, state_40, state_39, state_38,
                    state_37, state_36, state_35, state_34, state_33,
                    state_32, state_31, state_30, state_29, state_28,
                    state_27, state_26, state_25, state_24, state_23,
                    state_22, state_21, state_20, state_19, state_18,
                    state_17, state_16, state_15, state_14, state_13,
                    state_12, state_11, state_10, state_9, state_8,
                    state_7, state_6, state_5, state_4, state_3,
                    state_2, state_1, state_0);
SIGNAL state : states_type;  -- this stores the current and next state of the circuit

SIGNAL temporary_mr :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary_mtype :  std_logic_vector(31 DOWNTO 0);
SIGNAL v003_i :  std_logic_vector(31 DOWNTO 0);
SIGNAL v004_j :  std_logic_vector(31 DOWNTO 0);
SIGNAL v005_k :  std_logic_vector(31 DOWNTO 0);
SIGNAL v006_m :  std_logic_vector(31 DOWNTO 0);
SIGNAL v007_fcval :  std_logic_vector(31 DOWNTO 0);
SIGNAL v008_dr : type000 ;
SIGNAL v009_r :  std_logic_vector(31 DOWNTO 0);
SIGNAL v010_ra : type001 ;
SIGNAL v011_test :  std_logic_vector(31 DOWNTO 0);
SIGNAL v012_e_171 :  std_logic_vector(31 DOWNTO 0);
SIGNAL v013_e_135 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary000 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary001 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary002 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary003 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary004 :  std_logic_vector(31 DOWNTO 0);
```

```
SIGNAL temporary005 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary006 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary007 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary008 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary009 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary010 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary011 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary012 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary013 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary014 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary015 : std_logic;
SIGNAL temporary016 : std_logic;
SIGNAL temporary017 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary018 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary019 : std_logic;
SIGNAL temporary020 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary021 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary022 : std_logic;
SIGNAL temporary023 : std_logic;
SIGNAL temporary024 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary025 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary026 : std_logic;
SIGNAL temporary027 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary028 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary029 : std_logic;
SIGNAL temporary030 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary031 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary032 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary033 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary034 : std_logic;
SIGNAL temporary035 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary036 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary037 : std_logic;
SIGNAL temporary038 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary039 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary040 : std_logic;
SIGNAL temporary041 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary042 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary043 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary044 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary045 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary046 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary047 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary048 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary049 :  std_logic_vector(31 DOWNTO 0);
```

```
   SIGNAL temporary050 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary051 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary052 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary053 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary054 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary055 : std_logic;
   SIGNAL temporary056 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary057 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary058 : std_logic;
   SIGNAL temporary059 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary060 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary061 : std_logic;
   SIGNAL temporary062 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL index000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL tempint000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL index001 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL tempint001 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL index002 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL tempint002 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(0, 32));  --
integer constants are converted into std_logic
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(209,
32));  -- integer constants are converted into std_logic
   SIGNAL var1 : std_logic;
   CONSTANT const3 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2, 32));  --
integer constants are converted into std_logic
   SIGNAL var2 : std_logic;
   CONSTANT const4 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(210,
32));  -- integer constants are converted into std_logic
   SIGNAL var3 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const5 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(1, 32));  --
integer constants are converted into std_logic
   CONSTANT const6 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(208,
32));  -- integer constants are converted into std_logic
   SIGNAL var4 : std_logic;
   SIGNAL var5 : std_logic;
   SIGNAL var6 : std_logic;
   SIGNAL var7 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var8 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var9 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var10 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var11 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const7 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(655360,
32));  -- integer constants are converted into std_logic
   SIGNAL var12 : std_logic;
```

```
   CONSTANT const8 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(112066,
32));  -- integer constants are converted into std_logic
   CONSTANT const9 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2588082,
32));  -- integer constants are converted into std_logic
   CONSTANT const10 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(3481242,
32));  -- integer constants are converted into std_logic
   CONSTANT const11 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(89050,
32));  -- integer constants are converted into std_logic
   CONSTANT const12 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2764958,
32));  -- integer constants are converted into std_logic
   CONSTANT const13 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2669099,
32));  -- integer constants are converted into std_logic
   SIGNAL var13 : std_logic;
   SIGNAL var14 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var15 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var16 : std_logic;
   SIGNAL var17 : std_logic;
   CONSTANT const14 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(4, 32));  -
- integer constants are converted into std_logic
   SIGNAL var18 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL pow_int_results_read_int :  std_logic;
   SIGNAL pow_int_start_int :  std_logic;
   SIGNAL sqrt2_results_read_int :  std_logic;
   SIGNAL sqrt2_start_int :  std_logic;
   SIGNAL fp_mult_results_read_int :  std_logic;
   SIGNAL fp_mult_start_int :  std_logic;
   SIGNAL exp_int_results_read_int :  std_logic;
   SIGNAL exp_int_start_int :  std_logic;
   SIGNAL fp_div_results_read_int :  std_logic;
   SIGNAL fp_div_start_int :  std_logic;

     -- now the datapath (input/output) signal declarations follow --

   SIGNAL lessequal32_1_out : std_logic;
   SIGNAL lessequal32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL lessequal32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plus32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plusone32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL plusone32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL plusone32_1_in2 : std_logic_vector(31 DOWNTO 0);
```

```
 SIGNAL minus32_1_out : std_logic_vector(31 DOWNTO 0);
 SIGNAL minus32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL minus32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL less32_1_out : std_logic;
 SIGNAL less32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL less32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL equal32_1_out : std_logic;
 SIGNAL equal32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL equal32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL notequal32_1_out : std_logic;
 SIGNAL notequal32_1_in1 : std_logic_vector(31 DOWNTO 0);
 SIGNAL notequal32_1_in2 : std_logic_vector(31 DOWNTO 0);
 SIGNAL not1_1_out : std_logic;
 SIGNAL not1_1_in2 : std_logic;

BEGIN

 done <= done_int;
 pow_int_start <= pow_int_start_int;
 pow_int_results_read <= pow_int_results_read_int;
 sqrt2_start <= sqrt2_start_int;
 sqrt2_results_read <= sqrt2_results_read_int;
 fp_mult_start <= fp_mult_start_int;
 fp_mult_results_read <= fp_mult_results_read_int;
 exp_int_start <= exp_int_start_int;
 exp_int_results_read <= exp_int_results_read_int;
 fp_div_start <= fp_div_start_int;
 fp_div_results_read <= fp_div_results_read_int;

  --- FSM and controller loging (plus massively-parallel ops) ---
  fsm_core : PROCESS (clock, reset)
      VARIABLE var1_conditional_variable :  std_logic;
      VARIABLE var2_conditional_variable :  std_logic;
      VARIABLE var4_conditional_variable :  std_logic;
      VARIABLE var5_conditional_variable :  std_logic;
      VARIABLE var6_conditional_variable :  std_logic;
      VARIABLE var12_conditional_variable :  std_logic;
      VARIABLE temporary016_conditional_variable :  std_logic;
      VARIABLE temporary015_conditional_variable :  std_logic;
      VARIABLE temporary023_conditional_variable :  std_logic;
      VARIABLE temporary034_conditional_variable :  std_logic;
      VARIABLE var13_conditional_variable :  std_logic;
      VARIABLE var16_conditional_variable :  std_logic;
      VARIABLE var17_conditional_variable :  std_logic;
      VARIABLE temporary055_conditional_variable :  std_logic;
```

```
BEGIN
 IF reset = '0' THEN
  done_int <= '0';
  busy <= '0';
  state <= state_0;
  computeforces <= (OTHERS => '0');
  temporary_mr <= (OTHERS => '0');
  temporary_mtype <= (OTHERS => '0');
  v003_i <= (OTHERS => '0');
  v004_j <= (OTHERS => '0');
  v005_k <= (OTHERS => '0');
  v006_m <= (OTHERS => '0');
  v007_fcval <= (OTHERS => '0');
  FOR v008_dr_i IN 0 TO 2 LOOP
   v008_dr(v008_dr_i) <= (OTHERS => '0');
  END LOOP;
  v009_r <= (OTHERS => '0');
  FOR v010_ra_i IN 0 TO 629 LOOP
   v010_ra(v010_ra_i) <= (OTHERS => '0');
  END LOOP;
  v011_test <= (OTHERS => '0');
  v012_e_171 <= (OTHERS => '0');
  v013_e_135 <= (OTHERS => '0');
  temporary000 <= (OTHERS => '0');
  temporary001 <= (OTHERS => '0');
  temporary002 <= (OTHERS => '0');
  temporary003 <= (OTHERS => '0');
  temporary004 <= (OTHERS => '0');
  temporary005 <= (OTHERS => '0');
  temporary006 <= (OTHERS => '0');
  temporary007 <= (OTHERS => '0');
  temporary008 <= (OTHERS => '0');
  temporary009 <= (OTHERS => '0');
  temporary010 <= (OTHERS => '0');
  temporary011 <= (OTHERS => '0');
  temporary012 <= (OTHERS => '0');
  temporary013 <= (OTHERS => '0');
  temporary014 <= (OTHERS => '0');
  temporary015 <= '0';
  temporary016 <= '0';
  temporary017 <= (OTHERS => '0');
  temporary018 <= (OTHERS => '0');
  temporary019 <= '0';
  temporary020 <= (OTHERS => '0');
  temporary021 <= (OTHERS => '0');
```

```
        temporary022 <= '0';
        temporary023 <= '0';
        temporary024 <= (OTHERS => '0');
        temporary025 <= (OTHERS => '0');
        temporary026 <= '0';
        temporary027 <= (OTHERS => '0');
        temporary028 <= (OTHERS => '0');
        temporary029 <= '0';
        temporary030 <= (OTHERS => '0');
        temporary031 <= (OTHERS => '0');
        temporary032 <= (OTHERS => '0');
        temporary033 <= (OTHERS => '0');
        temporary034 <= '0';
        temporary035 <= (OTHERS => '0');
        temporary036 <= (OTHERS => '0');
        temporary037 <= '0';
        temporary038 <= (OTHERS => '0');
        temporary039 <= (OTHERS => '0');
        temporary040 <= '0';
        temporary041 <= (OTHERS => '0');
        temporary042 <= (OTHERS => '0');
        temporary043 <= (OTHERS => '0');
        temporary044 <= (OTHERS => '0');
        temporary045 <= (OTHERS => '0');
        temporary046 <= (OTHERS => '0');
        temporary047 <= (OTHERS => '0');
        temporary048 <= (OTHERS => '0');
        temporary049 <= (OTHERS => '0');
        temporary050 <= (OTHERS => '0');
        temporary051 <= (OTHERS => '0');
        temporary052 <= (OTHERS => '0');
        temporary053 <= (OTHERS => '0');
        temporary054 <= (OTHERS => '0');
        temporary055 <= '0';
        temporary056 <= (OTHERS => '0');
        temporary057 <= (OTHERS => '0');
        temporary058 <= '0';
        temporary059 <= (OTHERS => '0');
        temporary060 <= (OTHERS => '0');
        temporary061 <= '0';
        temporary062 <= (OTHERS => '0');
        index000 <= (OTHERS => '0');
        tempint000 <= (OTHERS => '0');
        index001 <= (OTHERS => '0');
        tempint001 <= (OTHERS => '0');
```

```
      index002 <= (OTHERS => '0');
      tempint002 <= (OTHERS => '0');
      var1 <= '0';
      var2 <= '0';
      var3 <= (OTHERS => '0');
      var4 <= '0';
      var5 <= '0';
      var6 <= '0';
      var7 <= (OTHERS => '0');
      var8 <= (OTHERS => '0');
      var9 <= (OTHERS => '0');
      var10 <= (OTHERS => '0');
      var11 <= (OTHERS => '0');
      var12 <= '0';
      var13 <= '0';
      var14 <= (OTHERS => '0');
      var15 <= (OTHERS => '0');
      var16 <= '0';
      var17 <= '0';
      var18 <= (OTHERS => '0');
      pow_int_num <= (OTHERS => '0');
      pow_int_pow <= (OTHERS => '0');
      sqrt2_number <= (OTHERS => '0');
      fp_mult_numa <= (OTHERS => '0');
      fp_mult_numb <= (OTHERS => '0');
      exp_int_pow <= (OTHERS => '0');
      fp_div_numa <= (OTHERS => '0');
      fp_div_numb <= (OTHERS => '0');
      pow_int_results_read_int <= '0';
      pow_int_start_int <= '0';
      sqrt2_results_read_int <= '0';
      sqrt2_start_int <= '0';
      fp_mult_results_read_int <= '0';
      fp_mult_start_int <= '0';
      exp_int_results_read_int <= '0';
      exp_int_start_int <= '0';
      fp_div_results_read_int <= '0';
      fp_div_start_int <= '0';


    ELSIF clock = '1' AND clock'EVENT THEN

      CASE state IS

        WHEN state_0 =>
```

```
 IF results_read = '1' THEN done_int <= '0'; END IF;
 IF start = '1' THEN
  IF done_int = '0' OR results_read = '1' THEN
   busy <= '1';   -- processing started
   state <= state_1;
  END IF;
 ELSE
  state <= state_0;
 END IF ;

WHEN state_1 =>
  state <= state_2;
  temporary_mr(31 DOWNTO 0) <= mr(31 DOWNTO 0) ;

WHEN state_2 =>
  state <= state_3;
  temporary_mtype(31 DOWNTO 0) <= mtype(31 DOWNTO 0) ;

WHEN state_3 =>
  state <= state_4;
  v007_fcval(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

WHEN state_4 =>
  state <= state_5;
  v003_i(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

WHEN state_5 =>
  state <= state_6;
  tempint000(31 DOWNTO 0) <= const2(31 DOWNTO 0) ;

WHEN state_6 =>
  state <= state_7;
  index000(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

WHEN state_7 =>
  state <= state_8;
  var1 <= lessequal32_1_out;


WHEN state_8 =>
IF var1 = '1' THEN
   state <= state_9;
  ELSE
   state <= state_23;
  END IF;
```

```
WHEN state_9 =>
  state <= state_10;
  v004_j(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

WHEN state_10 =>
  state <= state_11;
  tempint001(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

WHEN state_11 =>
  state <= state_12;
  index001(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

WHEN state_12 =>
  state <= state_13;
  var2 <= lessequal32_1_out;


WHEN state_13 =>
IF var2 = '1' THEN
   state <= state_14;
  ELSE
   state <= state_20;
  END IF;

WHEN state_14 =>
  state <= state_15;
  var3 <= mult32_1_out;

WHEN state_15 =>
  state <= state_16;
  temporary000 <= plus32_1_out;

WHEN state_16 =>
  state <= state_17;
  v010_ra(CONV_INTEGER(temporary000)) <= const1;

WHEN state_17 =>
  state <= state_18;
  v004_j <= plus32_1_out;

WHEN state_18 =>
  state <= state_19;
  index001 <= plusone32_1_out;
```

```
        WHEN state_19 =>
          state <= state_12;


        WHEN state_20 =>
          state <= state_21;
          v003_i <= plus32_1_out;

        WHEN state_21 =>
          state <= state_22;
          index000 <= plusone32_1_out;

        WHEN state_22 =>
          state <= state_7;


        WHEN state_23 =>
          state <= state_24;
          v003_i(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

        WHEN state_24 =>
          state <= state_25;
          tempint000(31 DOWNTO 0) <= const6(31 DOWNTO 0) ;

        WHEN state_25 =>
          state <= state_26;
          index000(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

        WHEN state_26 =>
          state <= state_27;
          var4 <= lessequal32_1_out;


        WHEN state_27 =>
        IF var4 = '1' THEN
           state <= state_28;
          ELSE
           state <= state_138;
          END IF;

        WHEN state_28 =>
          state <= state_29;
          v004_j <= plus32_1_out;

        WHEN state_29 =>
```

```
    state <= state_30;
    tempint001 <= minus32_1_out;

  WHEN state_30 =>
    state <= state_31;
    index001(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

  WHEN state_31 =>
    state <= state_32;
    var5 <= lessequal32_1_out;


  WHEN state_32 =>
  IF var5 = '1' THEN
     state <= state_33;
    ELSE
     state <= state_135;
    END IF;

  WHEN state_33 =>
    state <= state_34;
    v005_k(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

  WHEN state_34 =>
    state <= state_35;
    tempint002(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

  WHEN state_35 =>
    state <= state_36;
    index002(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

  WHEN state_36 =>
    state <= state_37;
    var6 <= lessequal32_1_out;


  WHEN state_37 =>
  IF var6 = '1' THEN
     state <= state_38;
    ELSE
     state <= state_49;
    END IF;

  WHEN state_38 =>
    state <= state_39;
```

```
        var7 <= plus32_1_out;

      WHEN state_39 =>
        state <= state_40;
        var8 <= mult32_1_out;

      WHEN state_40 =>
        state <= state_41;
        temporary001 <= plus32_1_out;

      WHEN state_41 =>
        state <= state_42;
        var9 <= plus32_1_out;

      WHEN state_42 =>
        state <= state_43;
        var10 <= mult32_1_out;

      WHEN state_43 =>
        state <= state_44;
        temporary003 <= plus32_1_out;

      WHEN state_44 =>
        state <= state_45;
        temporary005 <= minus32_1_out;

      WHEN state_45 =>
        state <= state_46;
        v008_dr(CONV_INTEGER(v005_k)) <= temporary005;

      WHEN state_46 =>
        state <= state_47;
        v005_k <= plus32_1_out;

      WHEN state_47 =>
        state <= state_48;
        index002 <= plusone32_1_out;

      WHEN state_48 =>
        state <= state_36;


      WHEN state_49 =>
        state <= state_50;
        temporary006 <= v008_dr(CONV_INTEGER(const1));
```

```
WHEN state_50 =>
  state <= state_51;


WHEN state_51 =>
  state <= state_52;
  temporary008 <= v008_dr(CONV_INTEGER(const5));

WHEN state_52 =>
  state <= state_53;


WHEN state_53 =>
  state <= state_54;
  temporary010 <= v008_dr(CONV_INTEGER(const3));

WHEN state_54 =>
  state <= state_55;


WHEN state_55 =>
  state <= state_56;
  var11 <= plus32_1_out;

WHEN state_56 =>
  state <= state_57;
  temporary012 <= plus32_1_out;

WHEN state_57 =>
  state <= state_58;


WHEN state_58 =>
  state <= state_59;
  v011_test(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

WHEN state_59 =>
  state <= state_60;


WHEN state_60 =>
  state <= state_61;
```

```
        WHEN state_61 =>
          state <= state_62;
          var12 <= less32_1_out;


        WHEN state_62 =>
        IF var12 = '1' THEN
           state <= state_63;
          ELSE
           state <= state_132;
          END IF;

        WHEN state_63 =>
          state <= state_64;
          temporary017 <= plus32_1_out;

        WHEN state_64 =>
          state <= state_65;
          temporary019 <= equal32_1_out;

        WHEN state_65 =>
          state <= state_66;
          temporary016 <= temporary019 ;


        WHEN state_66 =>
        IF temporary016 = '1' THEN
           state <= state_67;
          ELSE
           state <= state_70;
          END IF;

        WHEN state_67 =>
          state <= state_68;
          temporary020 <= plus32_1_out;

        WHEN state_68 =>
          state <= state_69;
          temporary022 <= notequal32_1_out;

        WHEN state_69 =>
          state <= state_70;
          temporary016 <= temporary022 ;

        WHEN state_70 =>
```

```
  state <= state_71;
  temporary015 <= not1_1_out;


WHEN state_71 =>
IF temporary015 = '1' THEN
   state <= state_72;
  ELSE
   state <= state_81;
  END IF;

WHEN state_72 =>
  state <= state_73;
  temporary024 <= plus32_1_out;

WHEN state_73 =>
  state <= state_74;
  temporary026 <= notequal32_1_out;

WHEN state_74 =>
  state <= state_75;
  temporary023 <= temporary026 ;


WHEN state_75 =>
IF temporary023 = '1' THEN
   state <= state_76;
  ELSE
   state <= state_79;
  END IF;

WHEN state_76 =>
  state <= state_77;
  temporary027 <= plus32_1_out;

WHEN state_77 =>
  state <= state_78;
  temporary029 <= equal32_1_out;

WHEN state_78 =>
  state <= state_79;
  temporary023 <= temporary029 ;

WHEN state_79 =>
  state <= state_80;
```

```
    temporary015 <= temporary023 ;

  WHEN state_80 =>
    state <= state_82;


  WHEN state_81 =>
    state <= state_82;
    temporary015 <= not1_1_out;


  WHEN state_82 =>
  IF temporary015 = '1' THEN
     state <= state_83;
    ELSE
     state <= state_91;
    END IF;

  WHEN state_83 =>
    state <= state_84;
    v011_test(31 DOWNTO 0) <= const5(31 DOWNTO 0) ;

  WHEN state_84 =>
    state <= state_85;


  WHEN state_85 =>
    state <= state_86;


  WHEN state_86 =>
    state <= state_87;


  WHEN state_87 =>
    state <= state_88;


  WHEN state_88 =>
    state <= state_89;


  WHEN state_89 =>
    state <= state_90;
    v007_fcval <= minus32_1_out;
```

```
WHEN state_90 =>
  state <= state_108;


WHEN state_91 =>
  state <= state_92;
  temporary035 <= plus32_1_out;

WHEN state_92 =>
  state <= state_93;
  temporary037 <= notequal32_1_out;

WHEN state_93 =>
  state <= state_94;
  temporary034 <= temporary037 ;


WHEN state_94 =>
IF temporary034 = '1' THEN
   state <= state_95;
  ELSE
   state <= state_98;
  END IF;

WHEN state_95 =>
  state <= state_96;
  temporary038 <= plus32_1_out;

WHEN state_96 =>
  state <= state_97;
  temporary040 <= notequal32_1_out;

WHEN state_97 =>
  state <= state_98;
  temporary034 <= temporary040 ;


WHEN state_98 =>
IF temporary034 = '1' THEN
   state <= state_99;
  ELSE
   state <= state_107;
  END IF;
```

```
        WHEN state_99 =>
          state <= state_100;
          v011_test(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

        WHEN state_100 =>
          state <= state_101;


        WHEN state_101 =>
          state <= state_102;


        WHEN state_102 =>
          state <= state_103;


        WHEN state_103 =>
          state <= state_104;


        WHEN state_104 =>
          state <= state_105;


        WHEN state_105 =>
          state <= state_106;
          v007_fcval <= minus32_1_out;

        WHEN state_106 =>
          state <= state_108;


        WHEN state_107 =>
          state <= state_108;
          v007_fcval(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

        WHEN state_108 =>
          state <= state_109;
          v005_k(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

        WHEN state_109 =>
          state <= state_110;
          tempint002(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

        WHEN state_110 =>
```

```
    state <= state_111;
    index002(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

  WHEN state_111 =>
    state <= state_112;
    var13 <= lessequal32_1_out;


  WHEN state_112 =>
  IF var13 = '1' THEN
     state <= state_113;
    ELSE
     state <= state_132;
    END IF;

  WHEN state_113 =>
    state <= state_114;
    var14 <= mult32_1_out;

  WHEN state_114 =>
    state <= state_115;
    temporary045 <= plus32_1_out;

  WHEN state_115 =>
    state <= state_116;
    temporary046 <= v008_dr(CONV_INTEGER(v005_k));

  WHEN state_116 =>
    state <= state_117;


  WHEN state_117 =>
    state <= state_118;


  WHEN state_118 =>
    state <= state_119;
    temporary049 <= v010_ra(CONV_INTEGER(temporary045));

  WHEN state_119 =>
    state <= state_120;
    temporary049 <= plus32_1_out;

  WHEN state_120 =>
    state <= state_121;
```

```
    v010_ra(CONV_INTEGER(temporary045)) <= temporary049;

  WHEN state_121 =>
    state <= state_122;
    var15 <= mult32_1_out;

  WHEN state_122 =>
    state <= state_123;
    temporary050 <= plus32_1_out;

  WHEN state_123 =>
    state <= state_124;
    temporary051 <= v008_dr(CONV_INTEGER(v005_k));

  WHEN state_124 =>
    state <= state_125;


  WHEN state_125 =>
    state <= state_126;


  WHEN state_126 =>
    state <= state_127;
    temporary054 <= v010_ra(CONV_INTEGER(temporary050));

  WHEN state_127 =>
    state <= state_128;
    temporary054 <= minus32_1_out;

  WHEN state_128 =>
    state <= state_129;
    v010_ra(CONV_INTEGER(temporary050)) <= temporary054;

  WHEN state_129 =>
    state <= state_130;
    v005_k <= plus32_1_out;

  WHEN state_130 =>
    state <= state_131;
    index002 <= plusone32_1_out;

  WHEN state_131 =>
    state <= state_111;
```

```
WHEN state_132 =>
  state <= state_133;
  v004_j <= plus32_1_out;

WHEN state_133 =>
  state <= state_134;
  index001 <= plusone32_1_out;

WHEN state_134 =>
  state <= state_31;


WHEN state_135 =>
  state <= state_136;
  v003_i <= plus32_1_out;

WHEN state_136 =>
  state <= state_137;
  index000 <= plusone32_1_out;

WHEN state_137 =>
  state <= state_26;


WHEN state_138 =>
  state <= state_139;
  v006_m(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

WHEN state_139 =>
  state <= state_140;
  tempint000(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;

WHEN state_140 =>
  state <= state_141;
  index000(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

WHEN state_141 =>
  state <= state_142;
  var16 <= lessequal32_1_out;


WHEN state_142 =>
IF var16 = '1' THEN
   state <= state_143;
```

```
    ELSE
     state <= state_167;
    END IF;

  WHEN state_143 =>
    state <= state_144;
    v004_j(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

  WHEN state_144 =>
    state <= state_145;
    tempint001(31 DOWNTO 0) <= const2(31 DOWNTO 0) ;

  WHEN state_145 =>
    state <= state_146;
    index001(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;

  WHEN state_146 =>
    state <= state_147;
    var17 <= lessequal32_1_out;


  WHEN state_147 =>
  IF var17 = '1' THEN
     state <= state_148;
    ELSE
     state <= state_164;
    END IF;

  WHEN state_148 =>
    state <= state_149;
    temporary056 <= plus32_1_out;

  WHEN state_149 =>
    state <= state_150;
    temporary058 <= equal32_1_out;

  WHEN state_150 =>
    state <= state_151;
    temporary055 <= not1_1_out;


  WHEN state_151 =>
  IF temporary055 = '1' THEN
     state <= state_152;
    ELSE
```

```
    state <= state_156;
  END IF;

WHEN state_152 =>
  state <= state_153;
  temporary059 <= plus32_1_out;

WHEN state_153 =>
  state <= state_154;
  temporary061 <= equal32_1_out;

WHEN state_154 =>
  state <= state_155;
  temporary055 <= temporary061 ;

WHEN state_155 =>
  state <= state_157;


WHEN state_156 =>
  state <= state_157;
  temporary055 <= not1_1_out;


WHEN state_157 =>
IF temporary055 = '1' THEN
   state <= state_158;
  ELSE
   state <= state_161;
  END IF;

WHEN state_158 =>
  state <= state_159;
  var18 <= mult32_1_out;

WHEN state_159 =>
  state <= state_160;
  temporary062 <= plus32_1_out;

WHEN state_160 =>
  state <= state_161;
  v010_ra(CONV_INTEGER(temporary062)) <= const1;

WHEN state_161 =>
  state <= state_162;
```

```
            v004_j <= plus32_1_out;

          WHEN state_162 =>
            state <= state_163;
            index001 <= plusone32_1_out;

          WHEN state_163 =>
            state <= state_146;


          WHEN state_164 =>
            state <= state_165;
            v006_m <= plus32_1_out;

          WHEN state_165 =>
            state <= state_166;
            index000 <= plusone32_1_out;

          WHEN state_166 =>
            state <= state_141;


          WHEN state_167 =>
           state <= state_0;
           done_int <= '1';    -- processing finished, results are ready !!!
           busy <= '0';

         WHEN OTHERS => state <= state_0;
                        done_int <= '0';
                        busy <= '0';

        END CASE ;

      END IF ;
     END PROCESS fsm_core ;



--  Datapath operators (Functional Units) --
  lessequal32_1_out <= '1' WHEN lessequal32_1_in1 <= lessequal32_1_in2 ELSE '0';
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  plus32_1_out <= plus32_1_in1 + plus32_1_in2;
  plusone32_1_out <= plusone32_1_in1 + plusone32_1_in2;
  minus32_1_out <= minus32_1_in1 - minus32_1_in2;
```

```
  less32_1_out <= '1' WHEN less32_1_in1 < less32_1_in2 ELSE '0';
  equal32_1_out <= '1' WHEN equal32_1_in1 = equal32_1_in2 ELSE '0';
  notequal32_1_out <= '1' WHEN notequal32_1_in1 /= notequal32_1_in2 ELSE '0';
  not1_1_out <= not not1_1_in2;


-- now the data routing --

-- now the less-equality comparisons --
  routing_lessequal32_1_in1:
   WITH state SELECT
   lessequal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_7,
                        conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_12,
                        conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_26,
                        conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_31,
                        conv_std_logic_vector(CONV_INTEGER(index002), 32) WHEN state_36,
                        conv_std_logic_vector(CONV_INTEGER(index002), 32) WHEN state_111,
                        conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_141,
                        conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_146,
                         (OTHERS => '0') WHEN OTHERS;
  routing_lessequal32_1_in2:
   WITH state SELECT
   lessequal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(tempint000), 32) WHEN state_7,
                        conv_std_logic_vector(CONV_INTEGER(tempint001), 32) WHEN state_12,
                        conv_std_logic_vector(CONV_INTEGER(tempint000), 32) WHEN state_26,
                        conv_std_logic_vector(CONV_INTEGER(tempint001), 32) WHEN state_31,
                        conv_std_logic_vector(CONV_INTEGER(tempint002), 32) WHEN state_36,
                        conv_std_logic_vector(CONV_INTEGER(tempint002), 32) WHEN state_111,
                        conv_std_logic_vector(CONV_INTEGER(tempint000), 32) WHEN state_141,
                        conv_std_logic_vector(CONV_INTEGER(tempint001), 32) WHEN state_146,
                         (OTHERS => '0') WHEN OTHERS;


-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_14,
                   conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_39,
                   conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_42,
                   conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_113,
                   conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_121,
                   conv_std_logic_vector(CONV_INTEGER(v006_m), 32) WHEN state_158,
                    (OTHERS => '0') WHEN OTHERS;
  routing_mult32_1_in2:
   WITH state SELECT
```

```
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_14,
                       conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_39,
                       conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_42,
                       conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_113,
                       conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_121,
                       conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_158,
                        (OTHERS => '0') WHEN OTHERS;


-- now the additions --
  routing_plus32_1_in1:
   WITH state SELECT
   plus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_15,
                       conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_17,
                       conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_20,
                       conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_28,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mr), 32) WHEN state_38,
                       conv_std_logic_vector(CONV_INTEGER(var7), 32) WHEN state_40,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mr), 32) WHEN state_41,
                       conv_std_logic_vector(CONV_INTEGER(var9), 32) WHEN state_43,
                       conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_46,
                       conv_std_logic_vector(CONV_INTEGER(temporary007), 32) WHEN state_55,
                       conv_std_logic_vector(CONV_INTEGER(var11), 32) WHEN state_56,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_63,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_67,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_72,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_76,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_91,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_95,
                       conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_114,
                       conv_std_logic_vector(CONV_INTEGER(temporary049), 32) WHEN state_119,
                       conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_122,
                       conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_129,
                       conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_132,
                       conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_135,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_148,
                       conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_152,
                       conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_159,
                       conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_161,
                       conv_std_logic_vector(CONV_INTEGER(v006_m), 32) WHEN state_164,
                        (OTHERS => '0') WHEN OTHERS;
  routing_plus32_1_in2:
   WITH state SELECT
   plus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(var3), 32) WHEN state_15,
                       conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_17,
```

```
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_20,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_28,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_38,
                    conv_std_logic_vector(CONV_INTEGER(var8), 32) WHEN state_40,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_41,
                    conv_std_logic_vector(CONV_INTEGER(var10), 32) WHEN state_43,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_46,
                    conv_std_logic_vector(CONV_INTEGER(temporary009), 32) WHEN state_55,
                    conv_std_logic_vector(CONV_INTEGER(temporary011), 32) WHEN state_56,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_63,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_67,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_72,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_76,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_91,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_95,
                    conv_std_logic_vector(CONV_INTEGER(var14), 32) WHEN state_114,
                    conv_std_logic_vector(CONV_INTEGER(temporary048), 32) WHEN state_119,
                    conv_std_logic_vector(CONV_INTEGER(var15), 32) WHEN state_122,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_129,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_132,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_135,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_148,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_152,
                    conv_std_logic_vector(CONV_INTEGER(var18), 32) WHEN state_159,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_161,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_164,
                     (OTHERS => '0') WHEN OTHERS;


-- now the increments by one --
  routing_plusone32_1_in1:
   WITH state SELECT
   plusone32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_18,
                    conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_21,
                    conv_std_logic_vector(CONV_INTEGER(index002), 32) WHEN state_47,
                    conv_std_logic_vector(CONV_INTEGER(index002), 32) WHEN state_130,
                    conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_133,
                    conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_136,
                    conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_162,
                    conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_165,
                     (OTHERS => '0') WHEN OTHERS;
  routing_plusone32_1_in2:
   WITH state SELECT
   plusone32_1_in2 <= conv_std_logic_vector(1, 32) WHEN state_18,
                    conv_std_logic_vector(2, 32) WHEN state_21,
```

```
                    conv_std_logic_vector(2, 32) WHEN state_47,
                    conv_std_logic_vector(2, 32) WHEN state_130,
                    conv_std_logic_vector(2, 32) WHEN state_133,
                    conv_std_logic_vector(2, 32) WHEN state_136,
                    conv_std_logic_vector(2, 32) WHEN state_162,
                    conv_std_logic_vector(2, 32) WHEN state_165,
                     (OTHERS => '0') WHEN OTHERS;


-- now the subtractions --
  routing_minus32_1_in1:
   WITH state SELECT
   minus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(const6), 32) WHEN state_29,
                    conv_std_logic_vector(CONV_INTEGER(temporary002), 32) WHEN state_44,
                    conv_std_logic_vector(CONV_INTEGER(temporary031), 32) WHEN state_89,
                    conv_std_logic_vector(CONV_INTEGER(temporary043), 32) WHEN state_105,
                    conv_std_logic_vector(CONV_INTEGER(temporary054), 32) WHEN state_127,
                     (OTHERS => '0') WHEN OTHERS;
  routing_minus32_1_in2:
   WITH state SELECT
   minus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_29,
                    conv_std_logic_vector(CONV_INTEGER(temporary004), 32) WHEN state_44,
                    conv_std_logic_vector(CONV_INTEGER(temporary033), 32) WHEN state_89,
                    conv_std_logic_vector(CONV_INTEGER(temporary044), 32) WHEN state_105,
                    conv_std_logic_vector(CONV_INTEGER(temporary053), 32) WHEN state_127,
                     (OTHERS => '0') WHEN OTHERS;


-- now the less-than comparisons --
  routing_less32_1_in1:
   WITH state SELECT
   less32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary013), 32) WHEN state_61,
                    (OTHERS => '0') WHEN OTHERS;
  routing_less32_1_in2:
   WITH state SELECT
   less32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary014), 32) WHEN state_61,
                    (OTHERS => '0') WHEN OTHERS;


-- now the equality comparisons --
  routing_equal32_1_in1:
   WITH state SELECT
   equal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary018), 32) WHEN state_64,
                    conv_std_logic_vector(CONV_INTEGER(temporary028), 32) WHEN state_77,
                    conv_std_logic_vector(CONV_INTEGER(temporary057), 32) WHEN state_149,
```

```
                    conv_std_logic_vector(CONV_INTEGER(temporary060), 32) WHEN state_153,
                     (OTHERS => '0') WHEN OTHERS;
  routing_equal32_1_in2:
   WITH state SELECT
   equal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_64,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_77,
                    conv_std_logic_vector(CONV_INTEGER(const14), 32) WHEN state_149,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_153,
                     (OTHERS => '0') WHEN OTHERS;


-- now the non-equality comparisons --
  routing_notequal32_1_in1:
   WITH state SELECT
   notequal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary021), 32) WHEN state_68,
                    conv_std_logic_vector(CONV_INTEGER(temporary025), 32) WHEN state_73,
                    conv_std_logic_vector(CONV_INTEGER(temporary036), 32) WHEN state_92,
                    conv_std_logic_vector(CONV_INTEGER(temporary039), 32) WHEN state_96,
                     (OTHERS => '0') WHEN OTHERS;
  routing_notequal32_1_in2:
   WITH state SELECT
   notequal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_68,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_73,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_92,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_96,
                     (OTHERS => '0') WHEN OTHERS;



  END  rtl ;
```

With PARCS optimization

```
------------------------------------------------------------
--:::::::::: C CUBED COMPILATION ->  VHDL RTL MODEL ::::::::::--

--:::::::::::::: Optimiser engine used: 'PARCS' ::::::::::::::--

--:::::::::: performed on design module: 'computeforces'

------------ HDL created on: 18/2/2023
------------ HDL created at: 19:54:49 ____ :40

--::: The C-cubed compiler, back-end version: CCC_be_6 :::--
--::::::: Copyright(c) 2007-2020, by Michael F. Dossis  ::::::::--
```

```
----------------------------------------------------------------


 LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
 USE IEEE.std_logic_arith.ALL;
 USE IEEE.std_logic_signed.ALL;

 PACKAGE new_func IS

  TYPE type000 IS ARRAY (0 TO 2) OF std_logic_vector(31 DOWNTO 0);

  TYPE type001 IS ARRAY (0 TO 629) OF std_logic_vector(31 DOWNTO 0);

 END new_func;

 PACKAGE BODY new_func IS

 END new_func;



 LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
 USE IEEE.std_logic_arith.ALL;
 USE IEEE.std_logic_signed.ALL;
 LIBRARY WORK;
 USE WORK.new_func.ALL;

 ENTITY computeforces IS
  port(
       clock, reset, start, results_read : IN std_logic;
       pow_int_1_results_read : OUT std_logic;
       pow_int_2_results_read : OUT std_logic;
       pow_int_3_results_read : OUT std_logic;
       pow_int_1_start : OUT std_logic;
       pow_int_2_start : OUT std_logic;
       pow_int_3_start : OUT std_logic;
       pow_int_1_done : IN std_logic;
       pow_int_2_done : IN std_logic;
       pow_int_3_done : IN std_logic;
       pow_int_1_busy : IN std_logic;
       pow_int_2_busy : IN std_logic;
       pow_int_3_busy : IN std_logic;
       sqrt2_results_read :  OUT std_logic;
```

```
        sqrt2_start :  OUT std_logic;
        sqrt2_done :  IN std_logic;
        sqrt2_busy :  IN std_logic;
        fp_mult_results_read :  OUT std_logic;
        fp_mult_start :  OUT std_logic;
        fp_mult_done :  IN std_logic;
        fp_mult_busy :  IN std_logic;
        exp_int_results_read :  OUT std_logic;
        exp_int_start :  OUT std_logic;
        exp_int_done :  IN std_logic;
        exp_int_busy :  IN std_logic;
        fp_div_1_results_read : OUT std_logic;
        fp_div_2_results_read : OUT std_logic;
        fp_div_1_start : OUT std_logic;
        fp_div_2_start : OUT std_logic;
        fp_div_1_done : IN std_logic;
        fp_div_2_done : IN std_logic;
        fp_div_1_busy : IN std_logic;
        fp_div_2_busy : IN std_logic;
        mr : IN  std_logic_vector(31 DOWNTO 0);
        mtype : IN  std_logic_vector(31 DOWNTO 0);
        computeforces : OUT  std_logic_vector(31 DOWNTO 0);
        pow_int_1_num : OUT  std_logic_vector(31 DOWNTO 0);
        pow_int_2_num : OUT  std_logic_vector(31 DOWNTO 0);
        pow_int_3_num : OUT  std_logic_vector(31 DOWNTO 0);
        pow_int_1_pow : OUT  std_logic_vector(31 DOWNTO 0);
        pow_int_2_pow : OUT  std_logic_vector(31 DOWNTO 0);
        pow_int_3_pow : OUT  std_logic_vector(31 DOWNTO 0);
        pow_int_1_pow_int : IN  std_logic_vector(31 DOWNTO 0);
        pow_int_2_pow_int : IN  std_logic_vector(31 DOWNTO 0);
        pow_int_3_pow_int : IN  std_logic_vector(31 DOWNTO 0);
        sqrt2_number : OUT  std_logic_vector(31 DOWNTO 0);
        sqrt2_sqrt2 : IN  std_logic_vector(31 DOWNTO 0);
        fp_mult_numa : OUT  std_logic_vector(31 DOWNTO 0);
        fp_mult_numb : OUT  std_logic_vector(31 DOWNTO 0);
        fp_mult_fp_mult : IN  std_logic_vector(31 DOWNTO 0);
        exp_int_pow : OUT  std_logic_vector(31 DOWNTO 0);
        exp_int_exp_int : IN  std_logic_vector(31 DOWNTO 0);
        fp_div_1_numa : OUT  std_logic_vector(31 DOWNTO 0);
        fp_div_2_numa : OUT  std_logic_vector(31 DOWNTO 0);
        fp_div_1_numb : OUT  std_logic_vector(31 DOWNTO 0);
        fp_div_2_numb : OUT  std_logic_vector(31 DOWNTO 0);
        fp_div_1_fp_div : IN  std_logic_vector(31 DOWNTO 0);
        fp_div_2_fp_div : IN  std_logic_vector(31 DOWNTO 0);
        done, busy : OUT std_logic
```

```
    );
END computeforces ;


ARCHITECTURE rtl OF computeforces IS

 SIGNAL done_int : std_logic;

 TYPE states_type IS (state_71,
                      state_70, state_69, state_68, state_67, state_66,
                      state_65, state_64, state_63, state_62, state_61,
                      state_60, state_59, state_58, state_57, state_56,
                      state_55, state_54, state_53, state_52, state_51,
                      state_50, state_49, state_48, state_47, state_46,
                      state_45, state_44, state_43, state_42, state_41,
                      state_40, state_39, state_38, state_37, state_36,
                      state_35, state_34, state_33, state_32, state_31,
                      state_30, state_29, state_28, state_27, state_26,
                      state_25, state_24, state_23, state_22, state_21,
                      state_20, state_19, state_18, state_17, state_16,
                      state_15, state_14, state_13, state_12, state_11,
                      state_10, state_9, state_8, state_7, state_6,
                      state_5, state_4, state_3, state_2, state_1,
                      state_0);
 SIGNAL state : states_type;  -- this stores the current and next state of the circuit

 SIGNAL temporary_mr :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary_mtype :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v003_i :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v004_j :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v005_k :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v006_m :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v007_fcval :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v008_dr : type000 ;
 SIGNAL v009_r :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v010_ra : type001 ;
 SIGNAL v011_test :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v012_e_171 :  std_logic_vector(31 DOWNTO 0);
 SIGNAL v013_e_135 :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary000 :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary001 :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary002 :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary003 :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary004 :  std_logic_vector(31 DOWNTO 0);
 SIGNAL temporary005 :  std_logic_vector(31 DOWNTO 0);
```

```
SIGNAL temporary006 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary007 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary008 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary009 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary010 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary011 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary012 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary013 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary014 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary015 : std_logic;
SIGNAL temporary016 : std_logic;
SIGNAL temporary017 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary018 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary019 : std_logic;
SIGNAL temporary020 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary021 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary022 : std_logic;
SIGNAL temporary023 : std_logic;
SIGNAL temporary024 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary025 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary026 : std_logic;
SIGNAL temporary027 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary028 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary029 : std_logic;
SIGNAL temporary030 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary031 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary032 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary033 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary034 : std_logic;
SIGNAL temporary035 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary036 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary037 : std_logic;
SIGNAL temporary038 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary039 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary040 : std_logic;
SIGNAL temporary041 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary042 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary043 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary044 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary045 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary046 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary047 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary048 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary049 :  std_logic_vector(31 DOWNTO 0);
SIGNAL temporary050 :  std_logic_vector(31 DOWNTO 0);
```

```
   SIGNAL temporary051 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary052 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary053 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary054 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary055 : std_logic;
   SIGNAL temporary056 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary057 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary058 : std_logic;
   SIGNAL temporary059 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary060 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL temporary061 : std_logic;
   SIGNAL temporary062 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL index000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL tempint000 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL index001 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL tempint001 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL index002 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL tempint002 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const1 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(0, 32));  --
integer constants are converted into std_logic
   CONSTANT const2 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(209,
32));  -- integer constants are converted into std_logic
   SIGNAL var1 : std_logic;
   CONSTANT const3 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2, 32));  --
integer constants are converted into std_logic
   SIGNAL var2 : std_logic;
   CONSTANT const4 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(210,
32));  -- integer constants are converted into std_logic
   SIGNAL var3 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const5 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(1, 32));  --
integer constants are converted into std_logic
   CONSTANT const6 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(208,
32));  -- integer constants are converted into std_logic
   SIGNAL var4 : std_logic;
   SIGNAL var5 : std_logic;
   SIGNAL var6 : std_logic;
   SIGNAL var7 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var8 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var9 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var10 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var11 :  std_logic_vector(31 DOWNTO 0);
   CONSTANT const7 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(655360,
32));  -- integer constants are converted into std_logic
   SIGNAL var12 : std_logic;
```

```
   CONSTANT const8 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(112066,
32));  -- integer constants are converted into std_logic
   CONSTANT const9 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2588082,
32));  -- integer constants are converted into std_logic
   CONSTANT const10 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(3481242,
32));  -- integer constants are converted into std_logic
   CONSTANT const11 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(89050,
32));  -- integer constants are converted into std_logic
   CONSTANT const12 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2764958,
32));  -- integer constants are converted into std_logic
   CONSTANT const13 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(2669099,
32));  -- integer constants are converted into std_logic
   SIGNAL var13 : std_logic;
   SIGNAL var14 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var15 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL var16 : std_logic;
   SIGNAL var17 : std_logic;
   CONSTANT const14 :  std_logic_vector(31 DOWNTO 0) := std_logic_vector(conv_unsigned(4, 32));  -
- integer constants are converted into std_logic
   SIGNAL var18 :  std_logic_vector(31 DOWNTO 0);
   SIGNAL pow_int_1_results_read_int : std_logic;
   SIGNAL pow_int_2_results_read_int : std_logic;
   SIGNAL pow_int_3_results_read_int : std_logic;
   SIGNAL pow_int_1_start_int : std_logic;
   SIGNAL pow_int_2_start_int : std_logic;
   SIGNAL pow_int_3_start_int : std_logic;
   SIGNAL sqrt2_results_read_int : std_logic;
   SIGNAL sqrt2_start_int : std_logic;
   SIGNAL fp_mult_results_read_int : std_logic;
   SIGNAL fp_mult_start_int : std_logic;
   SIGNAL exp_int_results_read_int : std_logic;
   SIGNAL exp_int_start_int : std_logic;
   SIGNAL fp_div_1_results_read_int : std_logic;
   SIGNAL fp_div_2_results_read_int : std_logic;
   SIGNAL fp_div_1_start_int : std_logic;
   SIGNAL fp_div_2_start_int : std_logic;

     -- now the datapath (input/output) signal declarations follow --

   SIGNAL lessequal32_1_out : std_logic;
   SIGNAL lessequal32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL lessequal32_1_in2 : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_out : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in1 : std_logic_vector(31 DOWNTO 0);
   SIGNAL mult32_1_in2 : std_logic_vector(31 DOWNTO 0);
```

```
  SIGNAL plus32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plusone32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL plusone32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plusone32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL minus32_1_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL minus32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL minus32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_2_out : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_2_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL plus32_2_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL less32_1_out : std_logic;
  SIGNAL less32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL less32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL equal32_1_out : std_logic;
  SIGNAL equal32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL equal32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL notequal32_1_out : std_logic;
  SIGNAL notequal32_1_in1 : std_logic_vector(31 DOWNTO 0);
  SIGNAL notequal32_1_in2 : std_logic_vector(31 DOWNTO 0);
  SIGNAL not1_1_out : std_logic;
  SIGNAL not1_1_in2 : std_logic;

BEGIN

  done <= done_int;
  pow_int_1_start <= pow_int_1_start_int;
  pow_int_2_start <= pow_int_2_start_int;
  pow_int_3_start <= pow_int_3_start_int;

  pow_int_1_results_read <= pow_int_1_results_read_int;
  pow_int_2_results_read <= pow_int_2_results_read_int;
  pow_int_3_results_read <= pow_int_3_results_read_int;

  sqrt2_start <= sqrt2_start_int;
  sqrt2_results_read <= sqrt2_results_read_int;
  fp_mult_start <= fp_mult_start_int;
  fp_mult_results_read <= fp_mult_results_read_int;
  exp_int_start <= exp_int_start_int;
  exp_int_results_read <= exp_int_results_read_int;
  fp_div_1_start <= fp_div_1_start_int;
  fp_div_2_start <= fp_div_2_start_int;

  fp_div_1_results_read <= fp_div_1_results_read_int;
```

```
fp_div_2_results_read <= fp_div_2_results_read_int;


 --- FSM and controller loging (plus massively-parallel ops) ---
 fsm_core : PROCESS (clock, reset)
     VARIABLE var1_conditional_variable :  std_logic;
     VARIABLE var2_conditional_variable :  std_logic;
     VARIABLE var4_conditional_variable :  std_logic;
     VARIABLE var5_conditional_variable :  std_logic;
     VARIABLE var6_conditional_variable :  std_logic;
     VARIABLE var12_conditional_variable :  std_logic;
     VARIABLE temporary016_conditional_variable :  std_logic;
     VARIABLE temporary015_conditional_variable :  std_logic;
     VARIABLE temporary023_conditional_variable :  std_logic;
     VARIABLE temporary034_conditional_variable :  std_logic;
     VARIABLE var13_conditional_variable :  std_logic;
     VARIABLE var16_conditional_variable :  std_logic;
     VARIABLE var17_conditional_variable :  std_logic;
     VARIABLE temporary055_conditional_variable :  std_logic;
  TYPE STATES_ARRAY IS ARRAY (integer RANGE <>) OF STATES_TYPE;
  VARIABLE state_var : STATES_ARRAY(5 DOWNTO 1);
  BEGIN
   IF reset = '0' THEN
    done_int <= '0';
    busy <= '0';
    state <= state_0;
    computeforces <= (OTHERS => '0');
    temporary_mr <= (OTHERS => '0');
    temporary_mtype <= (OTHERS => '0');
    v003_i <= (OTHERS => '0');
    v004_j <= (OTHERS => '0');
    v005_k <= (OTHERS => '0');
    v006_m <= (OTHERS => '0');
    v007_fcval <= (OTHERS => '0');
    FOR v008_dr_i IN 0 TO 2 LOOP
     v008_dr(v008_dr_i) <= (OTHERS => '0');
    END LOOP;
    v009_r <= (OTHERS => '0');
    FOR v010_ra_i IN 0 TO 629 LOOP
     v010_ra(v010_ra_i) <= (OTHERS => '0');
    END LOOP;
    v011_test <= (OTHERS => '0');
    v012_e_171 <= (OTHERS => '0');
    v013_e_135 <= (OTHERS => '0');
    temporary000 <= (OTHERS => '0');
```

```
    temporary001 <= (OTHERS => '0');
    temporary002 <= (OTHERS => '0');
    temporary003 <= (OTHERS => '0');
    temporary004 <= (OTHERS => '0');
    temporary005 <= (OTHERS => '0');
    temporary006 <= (OTHERS => '0');
    temporary007 <= (OTHERS => '0');
    temporary008 <= (OTHERS => '0');
    temporary009 <= (OTHERS => '0');
    temporary010 <= (OTHERS => '0');
    temporary011 <= (OTHERS => '0');
    temporary012 <= (OTHERS => '0');
    temporary013 <= (OTHERS => '0');
    temporary014 <= (OTHERS => '0');
    temporary015 <= '0';
    temporary016 <= '0';
    temporary017 <= (OTHERS => '0');
    temporary018 <= (OTHERS => '0');
    temporary019 <= '0';
    temporary020 <= (OTHERS => '0');
    temporary021 <= (OTHERS => '0');
    temporary022 <= '0';
    temporary023 <= '0';
    temporary024 <= (OTHERS => '0');
    temporary025 <= (OTHERS => '0');
    temporary026 <= '0';
    temporary027 <= (OTHERS => '0');
    temporary028 <= (OTHERS => '0');
    temporary029 <= '0';
    temporary030 <= (OTHERS => '0');
    temporary031 <= (OTHERS => '0');
    temporary032 <= (OTHERS => '0');
    temporary033 <= (OTHERS => '0');
    temporary034 <= '0';
    temporary035 <= (OTHERS => '0');
    temporary036 <= (OTHERS => '0');
    temporary037 <= '0';
    temporary038 <= (OTHERS => '0');
    temporary039 <= (OTHERS => '0');
    temporary040 <= '0';
    temporary041 <= (OTHERS => '0');
    temporary042 <= (OTHERS => '0');
    temporary043 <= (OTHERS => '0');
    temporary044 <= (OTHERS => '0');
    temporary045 <= (OTHERS => '0');
```

```
        temporary046 <= (OTHERS => '0');
        temporary047 <= (OTHERS => '0');
        temporary048 <= (OTHERS => '0');
        temporary049 <= (OTHERS => '0');
        temporary050 <= (OTHERS => '0');
        temporary051 <= (OTHERS => '0');
        temporary052 <= (OTHERS => '0');
        temporary053 <= (OTHERS => '0');
        temporary054 <= (OTHERS => '0');
        temporary055 <= '0';
        temporary056 <= (OTHERS => '0');
        temporary057 <= (OTHERS => '0');
        temporary058 <= '0';
        temporary059 <= (OTHERS => '0');
        temporary060 <= (OTHERS => '0');
        temporary061 <= '0';
        temporary062 <= (OTHERS => '0');
        index000 <= (OTHERS => '0');
        tempint000 <= (OTHERS => '0');
        index001 <= (OTHERS => '0');
        tempint001 <= (OTHERS => '0');
        index002 <= (OTHERS => '0');
        tempint002 <= (OTHERS => '0');
        var1 <= '0';
        var2 <= '0';
        var3 <= (OTHERS => '0');
        var4 <= '0';
        var5 <= '0';
        var6 <= '0';
        var7 <= (OTHERS => '0');
        var8 <= (OTHERS => '0');
        var9 <= (OTHERS => '0');
        var10 <= (OTHERS => '0');
        var11 <= (OTHERS => '0');
        var12 <= '0';
        var13 <= '0';
        var14 <= (OTHERS => '0');
        var15 <= (OTHERS => '0');
        var16 <= '0';
        var17 <= '0';
        var18 <= (OTHERS => '0');

        pow_int_1_results_read_int <= '0';
        pow_int_1_start_int <= '0';
        pow_int_1_num <= (OTHERS => '0');
```

```
      pow_int_1_pow <= (OTHERS => '0');
      pow_int_2_results_read_int <= '0';
      pow_int_2_start_int <= '0';
      pow_int_2_num <= (OTHERS => '0');
      pow_int_2_pow <= (OTHERS => '0');
      pow_int_3_results_read_int <= '0';
      pow_int_3_start_int <= '0';
      pow_int_3_num <= (OTHERS => '0');
      pow_int_3_pow <= (OTHERS => '0');
      sqrt2_results_read_int <= '0';
      sqrt2_start_int <= '0';
      sqrt2_number <= (OTHERS => '0');
      fp_mult_results_read_int <= '0';
      fp_mult_start_int <= '0';
      fp_mult_numa <= (OTHERS => '0');
      fp_mult_numb <= (OTHERS => '0');
      exp_int_results_read_int <= '0';
      exp_int_start_int <= '0';
      exp_int_pow <= (OTHERS => '0');
      fp_div_1_results_read_int <= '0';
      fp_div_1_start_int <= '0';
      fp_div_1_numa <= (OTHERS => '0');
      fp_div_1_numb <= (OTHERS => '0');
      fp_div_2_results_read_int <= '0';
      fp_div_2_start_int <= '0';
      fp_div_2_numa <= (OTHERS => '0');
      fp_div_2_numb <= (OTHERS => '0');

    ELSIF clock = '1' AND clock'EVENT THEN

      CASE state IS

        WHEN state_0 =>
         IF results_read = '1' THEN done_int <= '0'; END IF;
         IF start = '1' THEN
          IF done_int = '0' OR results_read = '1' THEN
           busy <= '1';   -- processing started
           state <= state_1;
          END IF;
         ELSE
          state <= state_0;
         END IF ;

        WHEN state_1 =>
          index000(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
```

```
    tempint000(31 DOWNTO 0) <= const2(31 DOWNTO 0) ;
    v003_i(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    v007_fcval(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    temporary_mtype(31 DOWNTO 0) <= mtype(31 DOWNTO 0) ;
    temporary_mr(31 DOWNTO 0) <= mr(31 DOWNTO 0) ;
    state <= state_2;
  WHEN state_2 =>
    IF index000 <= tempint000 THEN var1 <= '1'; ELSE var1 <= '0'; END IF;
    state <= state_3;
  WHEN state_3 =>
    IF var1 = '1' THEN
    state <= state_4;
    ELSE
    state <= state_9;
    END IF;
    IF var1 = '1' THEN
    v004_j(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    tempint001(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;
    index001(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    ELSE
    END IF;
  WHEN state_4 =>
    IF index001 <= tempint001 THEN var2 <= '1'; ELSE var2 <= '0'; END IF;
    state <= state_5;
  WHEN state_5 =>
    IF var2 = '1' THEN
    state <= state_6;
    ELSE
    state <= state_8;
    END IF;
    IF var2 = '1' THEN
    var3 <= mult32_1_out;
    ELSE
    END IF;
  WHEN state_6 =>
    temporary000 <= v003_i + var3;
    state <= state_7;
  WHEN state_7 =>
    v010_ra(CONV_INTEGER(temporary000)) <= const1;
    v004_j <= v004_j + const5;
    index001 <= index001 + 1;

    state <= state_4;
  WHEN state_8 =>
    v003_i <= v003_i + const5;
```

```
    index000 <= index000 + 1;

    state <= state_2;
  WHEN state_9 =>
    index000(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    tempint000(31 DOWNTO 0) <= const6(31 DOWNTO 0) ;
    v003_i(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    state <= state_10;
  WHEN state_10 =>
    IF index000 <= tempint000 THEN var4 <= '1'; ELSE var4 <= '0'; END IF;
    state <= state_11;
  WHEN state_11 =>
    IF var4 = '1' THEN
    state <= state_12;
    ELSE
    state <= state_57;
    END IF;
    IF var4 = '1' THEN
    v004_j <= plus32_1_out;
    ELSE
    END IF;
  WHEN state_12 =>
    tempint001 <= const6 - v004_j;
    index001(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    state <= state_13;
  WHEN state_13 =>
    IF index001 <= tempint001 THEN var5 <= '1'; ELSE var5 <= '0'; END IF;
    state <= state_14;
  WHEN state_14 =>
    IF var5 = '1' THEN
    state <= state_15;
    ELSE
    state <= state_56;
    END IF;
    IF var5 = '1' THEN
    v005_k(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    tempint002(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;
    index002(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    ELSE
    END IF;
  WHEN state_15 =>
    IF index002 <= tempint002 THEN var6 <= '1'; ELSE var6 <= '0'; END IF;
    state <= state_16;
  WHEN state_16 =>
    IF var6 = '1' THEN
```

```
    state <= state_17;
    ELSE
    state <= state_20;
    END IF;
    IF var6 = '1' THEN
    var7 <= plus32_1_out;
    var8 <= mult32_1_out;
    ELSE
    END IF;
  WHEN state_17 =>
    temporary001 <= var7 + var8;
    var9 <= temporary_mr + v004_j;
    var10 <= conv_std_logic_vector(CONV_INTEGER(v005_k) * CONV_INTEGER(const4), 32);
    state <= state_18;
  WHEN state_18 =>
    temporary003 <= var9 + var10;
    temporary005 <= temporary002 - temporary004;
    state <= state_19;
  WHEN state_19 =>
    v008_dr(CONV_INTEGER(v005_k)) <= temporary005;
    v005_k <= v005_k + const5;
    index002 <= index002 + 1;

    state <= state_15;
  WHEN state_20 =>
    temporary006 <= v008_dr(CONV_INTEGER(const1));
    ------ this is a call to module : pow_int -----
   pow_int_1_num <= temporary006;
   pow_int_1_pow <= const3;

                IF pow_int_1_busy = '0' AND pow_int_1_done = '0' THEN
                 IF pow_int_1_results_read_int = '1' THEN
                  IF pow_int_1_results_read_int = '1'  AND
                         pow_int_2_results_read_int = '1'  AND
                         pow_int_3_results_read_int = '1'
                  THEN
                   pow_int_1_start_int <= '0';
                         pow_int_2_start_int <= '0';
                         pow_int_3_start_int <= '0';
                   pow_int_1_results_read_int <= '0';
                         pow_int_2_results_read_int <= '0';
                         pow_int_3_results_read_int <= '0';
                   state_var(1) := state_0;
                   done_int <= '1';   -- processing finished, results are ready !!!
                   busy <= '0';
```

```
                        END IF;
                       ELSE
                         pow_int_1_start_int <= '1';
                         state_var(1) := state_20;
                       END IF;
                      ELSIF pow_int_1_busy = '1' AND pow_int_1_start_int = '1' THEN   -- when it
begins
                       pow_int_1_start_int <= '0';
                       state_var(1) := state_20;
                      ELSIF pow_int_1_done = '1'   THEN   -- when it is completed then read the
results
           temporary007 <= pow_int_1_pow_int;
                       pow_int_1_start_int <= '0';
                       IF pow_int_1_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                        pow_int_1_results_read_int <= '1';
                        state_var(1) := state_20;
                       ELSIF pow_int_1_results_read_int = '1'  AND
                                     pow_int_2_results_read_int = '1'  AND
                                     pow_int_3_results_read_int = '1'
                       THEN -- all calls are synchronized and completed
                         pow_int_1_results_read_int <= '0';
                                     pow_int_2_results_read_int <= '0';
                                     pow_int_3_results_read_int <= '0';
                         state_var(1) := state_0;
                         done_int <= '1';   -- processing finished, results are ready !!!
                         busy <= '0';
                        END IF;
                       ELSE
                        state_var(1) := state_20;
                       END IF;
            temporary008 <= v008_dr(CONV_INTEGER(const5));
            ------ this is a call to module : pow_int -----
           pow_int_2_num <= temporary008;
           pow_int_2_pow <= const3;

                       IF pow_int_2_busy = '0' AND pow_int_2_done = '0' THEN
                        IF pow_int_2_results_read_int = '1' THEN
                         IF pow_int_1_results_read_int = '1'  AND
                                pow_int_2_results_read_int = '1'  AND
                                pow_int_3_results_read_int = '1'
                         THEN
                          pow_int_1_start_int <= '0';
                                pow_int_2_start_int <= '0';
                                pow_int_3_start_int <= '0';
```

```
                        pow_int_1_results_read_int <= '0';
                              pow_int_2_results_read_int <= '0';
                              pow_int_3_results_read_int <= '0';
                        state_var(2) := state_var(1);
                        IF state_var(1) = state_0 THEN
                         done_int <= '1';   -- processing finished, results are ready !!!
                         busy <= '0';
                        END IF;
                       END IF;
                      ELSE
                        pow_int_2_start_int <= '1';
                        state_var(2) := state_20;
                      END IF;
                     ELSIF pow_int_2_busy = '1' AND pow_int_2_start_int = '1' THEN   -- when it
begins
                      pow_int_2_start_int <= '0';
                      state_var(2) := state_20;
                     ELSIF pow_int_2_done = '1'   THEN   -- when it is completed then read the
results
           temporary009 <= pow_int_2_pow_int;
                      pow_int_2_start_int <= '0';
                      IF pow_int_2_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                       pow_int_2_results_read_int <= '1';
                       state_var(2) := state_20;
                      ELSIF pow_int_1_results_read_int = '1'  AND
                                    pow_int_2_results_read_int = '1'  AND
                                    pow_int_3_results_read_int = '1'
                      THEN -- all calls are synchronized and completed
                        pow_int_1_results_read_int <= '0';
                                    pow_int_2_results_read_int <= '0';
                                    pow_int_3_results_read_int <= '0';
                        state_var(2) := state_var(1);
                        IF state_var(1) = state_0 THEN
                         done_int <= '1';   -- processing finished, results are ready !!!
                         busy <= '0';
                        END IF;
                       END IF;
                     ELSE
                      state_var(2) := state_20;
                     END IF;
            temporary010 <= v008_dr(CONV_INTEGER(const3));
            ------ this is a call to module : pow_int -----
           pow_int_3_num <= temporary010;
           pow_int_3_pow <= const3;
```

```
                    IF pow_int_3_busy = '0' AND pow_int_3_done = '0' THEN
                     IF pow_int_3_results_read_int = '1' THEN
                      IF pow_int_1_results_read_int = '1'  AND
                             pow_int_2_results_read_int = '1'  AND
                             pow_int_3_results_read_int = '1'
                      THEN
                       pow_int_1_start_int <= '0';
                             pow_int_2_start_int <= '0';
                             pow_int_3_start_int <= '0';
                       pow_int_1_results_read_int <= '0';
                             pow_int_2_results_read_int <= '0';
                             pow_int_3_results_read_int <= '0';
                       state_var(3) := state_var(2);
                       IF state_var(2) = state_0 THEN
                        done_int <= '1';   -- processing finished, results are ready !!!
                        busy <= '0';
                       END IF;
                      END IF;
                     ELSE
                       pow_int_3_start_int <= '1';
                       state_var(3) := state_20;
                     END IF;
                    ELSIF pow_int_3_busy = '1' AND pow_int_3_start_int = '1' THEN   -- when it
begins
                     pow_int_3_start_int <= '0';
                     state_var(3) := state_20;
                    ELSIF pow_int_3_done = '1'   THEN   -- when it is completed then read the
results
           temporary011 <= pow_int_3_pow_int;
                     pow_int_3_start_int <= '0';
                     IF pow_int_3_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                      pow_int_3_results_read_int <= '1';
                      state_var(3) := state_20;
                     ELSIF pow_int_1_results_read_int = '1'  AND
                                   pow_int_2_results_read_int = '1'  AND
                                   pow_int_3_results_read_int = '1'
                     THEN -- all calls are synchronized and completed
                       pow_int_1_results_read_int <= '0';
                                   pow_int_2_results_read_int <= '0';
                                   pow_int_3_results_read_int <= '0';
                       state_var(3) := state_var(2);
                       IF state_var(2) = state_0 THEN
                        done_int <= '1';   -- processing finished, results are ready !!!
```

```
                busy <= '0';
               END IF;
              END IF;
             ELSE
              state_var(3) := state_20;
             END IF;
        state <= state_var(3);
    WHEN state_21 =>
      var11 <= temporary007 + temporary009;
      state <= state_22;
    WHEN state_22 =>
      temporary012 <= var11 + temporary011;
      ------ this is a call to module : sqrt2 -----
     sqrt2_number <= temporary012;

             IF sqrt2_busy = '0' AND sqrt2_done = '0' THEN
              IF sqrt2_results_read_int = '1' THEN
               IF sqrt2_results_read_int = '1'  AND
                     pow_int_1_results_read_int = '1'  AND
                     pow_int_2_results_read_int = '1'
               THEN
                sqrt2_start_int <= '0';
                     pow_int_1_start_int <= '0';
                     pow_int_2_start_int <= '0';
                sqrt2_results_read_int <= '0';
                     pow_int_1_results_read_int <= '0';
                     pow_int_2_results_read_int <= '0';
                state_var(1) := state_0;
                done_int <= '1';   -- processing finished, results are ready !!!
                busy <= '0';
               END IF;
              ELSE
                sqrt2_start_int <= '1';
                state_var(1) := state_22;
              END IF;
             ELSIF sqrt2_busy = '1' AND sqrt2_start_int = '1' THEN   -- when it begins
              sqrt2_start_int <= '0';
              state_var(1) := state_22;
             ELSIF sqrt2_done = '1'   THEN   -- when it is completed then read the
results
      v009_r <= sqrt2_sqrt2;
              sqrt2_start_int <= '0';
              IF sqrt2_results_read_int = '0' THEN  -- if it is done then indicate that
the results are read
               sqrt2_results_read_int <= '1';
```

```
                        state_var(1) := state_22;
                       ELSIF sqrt2_results_read_int = '1'  AND
                                      pow_int_1_results_read_int = '1'  AND
                                      pow_int_2_results_read_int = '1'
                       THEN -- all calls are synchronized and completed
                         sqrt2_results_read_int <= '0';
                                      pow_int_1_results_read_int <= '0';
                                      pow_int_2_results_read_int <= '0';
                         state_var(1) := state_0;
                         done_int <= '1';   -- processing finished, results are ready !!!
                         busy <= '0';
                       END IF;
                      ELSE
                       state_var(1) := state_22;
                      END IF;
           v011_test(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
           ------ this is a call to module : pow_int -----
          pow_int_1_num <= v009_r;
          pow_int_1_pow <= const3;

                      IF pow_int_1_busy = '0' AND pow_int_1_done = '0' THEN
                       IF pow_int_1_results_read_int = '1' THEN
                        IF sqrt2_results_read_int = '1'  AND
                               pow_int_1_results_read_int = '1'  AND
                               pow_int_2_results_read_int = '1'
                        THEN
                         sqrt2_start_int <= '0';
                               pow_int_1_start_int <= '0';
                               pow_int_2_start_int <= '0';
                         sqrt2_results_read_int <= '0';
                               pow_int_1_results_read_int <= '0';
                               pow_int_2_results_read_int <= '0';
                         state_var(2) := state_var(1);
                         IF state_var(1) = state_0 THEN
                          done_int <= '1';   -- processing finished, results are ready !!!
                          busy <= '0';
                         END IF;
                        END IF;
                       ELSE
                         pow_int_1_start_int <= '1';
                         state_var(2) := state_22;
                       END IF;
                      ELSIF pow_int_1_busy = '1' AND pow_int_1_start_int = '1' THEN   -- when it
begins

                       pow_int_1_start_int <= '0';
```

```
                        state_var(2) := state_22;
                       ELSIF pow_int_1_done = '1'   THEN   -- when it is completed then read the
results
           temporary013 <= pow_int_1_pow_int;
                        pow_int_1_start_int <= '0';
                        IF pow_int_1_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                         pow_int_1_results_read_int <= '1';
                         state_var(2) := state_22;
                        ELSIF sqrt2_results_read_int = '1'  AND
                                      pow_int_1_results_read_int = '1'  AND
                                      pow_int_2_results_read_int = '1'
                        THEN -- all calls are synchronized and completed
                          sqrt2_results_read_int <= '0';
                                      pow_int_1_results_read_int <= '0';
                                      pow_int_2_results_read_int <= '0';
                          state_var(2) := state_var(1);
                          IF state_var(1) = state_0 THEN
                           done_int <= '1';   -- processing finished, results are ready !!!
                           busy <= '0';
                          END IF;
                        END IF;
                       ELSE
                        state_var(2) := state_22;
                       END IF;
            ------ this is a call to module : pow_int -----
           pow_int_2_num <= const7;
           pow_int_2_pow <= const3;

                       IF pow_int_2_busy = '0' AND pow_int_2_done = '0' THEN
                        IF pow_int_2_results_read_int = '1' THEN
                         IF sqrt2_results_read_int = '1'  AND
                                pow_int_1_results_read_int = '1'  AND
                                pow_int_2_results_read_int = '1'
                          THEN
                           sqrt2_start_int <= '0';
                                pow_int_1_start_int <= '0';
                                pow_int_2_start_int <= '0';
                           sqrt2_results_read_int <= '0';
                                pow_int_1_results_read_int <= '0';
                                pow_int_2_results_read_int <= '0';
                           state_var(3) := state_var(2);
                           IF state_var(2) = state_0 THEN
                            done_int <= '1';   -- processing finished, results are ready !!!
                            busy <= '0';
```

```
                        END IF;
                       END IF;
                      ELSE
                        pow_int_2_start_int <= '1';
                        state_var(3) := state_22;
                      END IF;
                     ELSIF pow_int_2_busy = '1' AND pow_int_2_start_int = '1' THEN   -- when it
begins
                      pow_int_2_start_int <= '0';
                      state_var(3) := state_22;
                     ELSIF pow_int_2_done = '1'   THEN   -- when it is completed then read the
results
           temporary014 <= pow_int_2_pow_int;
                      pow_int_2_start_int <= '0';
                      IF pow_int_2_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                       pow_int_2_results_read_int <= '1';
                       state_var(3) := state_22;
                      ELSIF sqrt2_results_read_int = '1'  AND
                                    pow_int_1_results_read_int = '1'  AND
                                    pow_int_2_results_read_int = '1'
                      THEN -- all calls are synchronized and completed
                        sqrt2_results_read_int <= '0';
                                    pow_int_1_results_read_int <= '0';
                                    pow_int_2_results_read_int <= '0';
                        state_var(3) := state_var(2);
                        IF state_var(2) = state_0 THEN
                         done_int <= '1';   -- processing finished, results are ready !!!
                         busy <= '0';
                        END IF;
                       END IF;
                     ELSE
                      state_var(3) := state_22;
                     END IF;
                state <= state_var(3);
          WHEN state_23 =>
            IF temporary013 < temporary014 THEN var12 <= '1'; ELSE var12 <= '0'; END IF;
            state <= state_24;
          WHEN state_24 =>
            IF var12 = '1' THEN
            state <= state_25;
            ELSE
            state <= state_55;
            END IF;
            IF var12 = '1' THEN
```

```
    temporary017 <= plus32_1_out;
    temporary019 <= equal32_1_out;
    ELSE
    END IF;
  WHEN state_25 =>
    temporary016 <= temporary019 ;
    state <= state_26;
  WHEN state_26 =>
    IF temporary016 = '1' THEN
    state <= state_27;
    ELSE
    state <= state_28;
    END IF;
    IF temporary016 = '1' THEN
    temporary020 <= plus32_1_out;
    temporary022 <= notequal32_1_out;
    ELSE
    END IF;
  WHEN state_27 =>
    temporary016 <= temporary022 ;
    state <= state_28;
  WHEN state_28 =>
    temporary015 <= not temporary016 ;
    state <= state_29;
  WHEN state_29 =>
    IF temporary015 = '1' THEN
    state <= state_30;
    ELSE
    state <= state_34;
    END IF;
    IF temporary015 = '1' THEN
    temporary024 <= plus32_1_out;
    temporary026 <= notequal32_1_out;
    ELSE
    END IF;
  WHEN state_30 =>
    temporary023 <= temporary026 ;
    state <= state_31;
  WHEN state_31 =>
    IF temporary023 = '1' THEN
    state <= state_32;
    ELSE
    state <= state_33;
    END IF;
    IF temporary023 = '1' THEN
```

```
  temporary027 <= plus32_1_out;
  temporary029 <= equal32_1_out;
  ELSE
  END IF;
WHEN state_32 =>
  temporary023 <= temporary029 ;
  state <= state_33;
WHEN state_33 =>
  temporary015 <= temporary023 ;

  state <= state_35;
WHEN state_34 =>
  temporary015 <= not temporary015 ;
  state <= state_35;
WHEN state_35 =>
  IF temporary015 = '1' THEN
  state <= state_36;
  ELSE
  state <= state_37;
  END IF;
  IF temporary015 = '1' THEN
  v011_test(31 DOWNTO 0) <= const5(31 DOWNTO 0) ;




   ------ this is a call to module : fp_mult -----
 fp_mult_numa <= const8;
 fp_mult_numb <= v009_r;

              IF fp_mult_busy = '0' AND fp_mult_done = '0' THEN
               IF fp_mult_results_read_int = '1' THEN
                IF fp_mult_results_read_int = '1'  AND
                      exp_int_results_read_int = '1'  AND
                      fp_div_1_results_read_int = '1'  AND
                      fp_div_2_results_read_int = '1'  AND
                      pow_int_results_read_int = '1'
                THEN
                 fp_mult_start_int <= '0';
                      exp_int_start_int <= '0';
                      fp_div_1_start_int <= '0';
                      fp_div_2_start_int <= '0';
                      pow_int_start_int <= '0';
                 fp_mult_results_read_int <= '0';
```

```
                              exp_int_results_read_int <= '0';
                              fp_div_1_results_read_int <= '0';
                              fp_div_2_results_read_int <= '0';
                              pow_int_results_read_int <= '0';
                            state_var(1) := state_0;
                            done_int <= '1';   -- processing finished, results are ready !!!
                            busy <= '0';
                           END IF;
                          ELSE
                            fp_mult_start_int <= '1';
                            state_var(1) := state_35;
                          END IF;
                         ELSIF fp_mult_busy = '1' AND fp_mult_start_int = '1'  THEN   -- when it
begins
                          fp_mult_start_int <= '0';
                          state_var(1) := state_35;
                         ELSIF fp_mult_done = '1'  THEN   -- when it is completed then read the
results
           temporary030 <= fp_mult_fp_mult;
                          fp_mult_start_int <= '0';
                          IF fp_mult_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                           fp_mult_results_read_int <= '1';
                           state_var(1) := state_35;
                          ELSIF fp_mult_results_read_int = '1'  AND
                                        exp_int_results_read_int = '1'  AND
                                        fp_div_1_results_read_int = '1'  AND
                                        fp_div_2_results_read_int = '1'  AND
                                        pow_int_results_read_int = '1'
                          THEN -- all calls are synchronized and completed
                            fp_mult_results_read_int <= '0';
                                        exp_int_results_read_int <= '0';
                                        fp_div_1_results_read_int <= '0';
                                        fp_div_2_results_read_int <= '0';
                                        pow_int_results_read_int <= '0';
                            state_var(1) := state_0;
                            done_int <= '1';   -- processing finished, results are ready !!!
                            busy <= '0';
                           END IF;
                         ELSE
                           state_var(1) := state_35;
                         END IF;
             ------ this is a call to module : exp_int -----
           exp_int_pow <= temporary030;
                         IF exp_int_busy = '0' AND exp_int_done = '0' THEN
```

```
                        IF exp_int_results_read_int = '1' THEN
                         IF fp_mult_results_read_int = '1'  AND
                               exp_int_results_read_int = '1'  AND
                               fp_div_1_results_read_int = '1'  AND
                               fp_div_2_results_read_int = '1'  AND
                               pow_int_results_read_int = '1'
                         THEN
                          fp_mult_start_int <= '0';
                               exp_int_start_int <= '0';
                               fp_div_1_start_int <= '0';
                               fp_div_2_start_int <= '0';
                               pow_int_start_int <= '0';
                          fp_mult_results_read_int <= '0';
                               exp_int_results_read_int <= '0';
                               fp_div_1_results_read_int <= '0';
                               fp_div_2_results_read_int <= '0';
                               pow_int_results_read_int <= '0';
                          state_var(2) := state_var(1);
                          IF state_var(1) = state_0 THEN
                           done_int <= '1';   -- processing finished, results are ready !!!
                           busy <= '0';
                          END IF;
                         END IF;
                        ELSE
                          exp_int_start_int <= '1';
                          state_var(2) := state_35;
                        END IF;
                       ELSIF exp_int_busy = '1' AND exp_int_start_int = '1'  THEN   -- when it
begins
                        exp_int_start_int <= '0';
                        state_var(2) := state_35;
                       ELSIF exp_int_done = '1'  THEN   -- when it is completed then read the
results
          v012_e_171 <= exp_int_exp_int;
                        exp_int_start_int <= '0';
                        IF exp_int_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                         exp_int_results_read_int <= '1';
                         state_var(2) := state_35;
                        ELSIF fp_mult_results_read_int = '1'  AND
                                     exp_int_results_read_int = '1'  AND
                                     fp_div_1_results_read_int = '1'  AND
                                     fp_div_2_results_read_int = '1'  AND
                                     pow_int_results_read_int = '1'
                        THEN -- all calls are synchronized and completed
```

```
                    fp_mult_results_read_int <= '0';
                                exp_int_results_read_int <= '0';
                                fp_div_1_results_read_int <= '0';
                                fp_div_2_results_read_int <= '0';
                                pow_int_results_read_int <= '0';
                    state_var(2) := state_var(1);
                    IF state_var(1) = state_0 THEN
                     done_int <= '1';   -- processing finished, results are ready !!!
                     busy <= '0';
                    END IF;
                   END IF;
                  ELSE
                   state_var(2) := state_35;
                  END IF;
    ------ this is a call to module : fp_div -----
fp_div_1_numa <= const9;
fp_div_1_numb <= v012_e_171;

                  IF fp_div_1_busy = '0' AND fp_div_1_done = '0' THEN
                   IF fp_div_1_results_read_int = '1' THEN
                    IF fp_mult_results_read_int = '1'  AND
                          exp_int_results_read_int = '1'  AND
                          fp_div_1_results_read_int = '1'  AND
                          fp_div_2_results_read_int = '1'  AND
                          pow_int_results_read_int = '1'
                    THEN
                     fp_mult_start_int <= '0';
                          exp_int_start_int <= '0';
                          fp_div_1_start_int <= '0';
                          fp_div_2_start_int <= '0';
                          pow_int_start_int <= '0';
                     fp_mult_results_read_int <= '0';
                          exp_int_results_read_int <= '0';
                          fp_div_1_results_read_int <= '0';
                          fp_div_2_results_read_int <= '0';
                          pow_int_results_read_int <= '0';
                     state_var(3) := state_var(2);
                     IF state_var(2) = state_0 THEN
                      done_int <= '1';   -- processing finished, results are ready !!!
                      busy <= '0';
                     END IF;
                    END IF;
                   ELSE
                     fp_div_1_start_int <= '1';
                     state_var(3) := state_35;
```

```
                        END IF;
                       ELSIF fp_div_1_busy = '1' AND fp_div_1_start_int = '1'  THEN   -- when it
begins
                        fp_div_1_start_int <= '0';
                        state_var(3) := state_35;
                       ELSIF fp_div_1_done = '1'   THEN   -- when it is completed then read the
results
          temporary031 <= fp_div_1_fp_div;
                        fp_div_1_start_int <= '0';
                        IF fp_div_1_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                         fp_div_1_results_read_int <= '1';
                         state_var(3) := state_35;
                        ELSIF fp_mult_results_read_int = '1'  AND
                                      exp_int_results_read_int = '1'  AND
                                      fp_div_1_results_read_int = '1'  AND
                                      fp_div_2_results_read_int = '1'  AND
                                      pow_int_results_read_int = '1'
                        THEN -- all calls are synchronized and completed
                          fp_mult_results_read_int <= '0';
                                      exp_int_results_read_int <= '0';
                                      fp_div_1_results_read_int <= '0';
                                      fp_div_2_results_read_int <= '0';
                                      pow_int_results_read_int <= '0';
                          state_var(3) := state_var(2);
                          IF state_var(2) = state_0 THEN
                           done_int <= '1';   -- processing finished, results are ready !!!
                           busy <= '0';
                          END IF;
                        END IF;
                       ELSE
                        state_var(3) := state_35;
                       END IF;
            ------ this is a call to module : fp_div -----
          fp_div_2_numa <= const10;
          fp_div_2_numb <= v012_e_171;

                       IF fp_div_2_busy = '0' AND fp_div_2_done = '0' THEN
                        IF fp_div_2_results_read_int = '1' THEN
                         IF fp_mult_results_read_int = '1'  AND
                               exp_int_results_read_int = '1'  AND
                               fp_div_1_results_read_int = '1'  AND
                               fp_div_2_results_read_int = '1'  AND
                               pow_int_results_read_int = '1'
                         THEN
```

```
                        fp_mult_start_int <= '0';
                             exp_int_start_int <= '0';
                             fp_div_1_start_int <= '0';
                             fp_div_2_start_int <= '0';
                             pow_int_start_int <= '0';
                        fp_mult_results_read_int <= '0';
                             exp_int_results_read_int <= '0';
                             fp_div_1_results_read_int <= '0';
                             fp_div_2_results_read_int <= '0';
                             pow_int_results_read_int <= '0';
                        state_var(4) := state_var(3);
                        IF state_var(3) = state_0 THEN
                         done_int <= '1';   -- processing finished, results are ready !!!
                         busy <= '0';
                        END IF;
                       END IF;
                      ELSE
                        fp_div_2_start_int <= '1';
                        state_var(4) := state_35;
                      END IF;
                     ELSIF fp_div_2_busy = '1' AND fp_div_2_start_int = '1'  THEN   -- when it
begins
                      fp_div_2_start_int <= '0';
                      state_var(4) := state_35;
                     ELSIF fp_div_2_done = '1'   THEN   -- when it is completed then read the
results
           temporary032 <= fp_div_2_fp_div;
                      fp_div_2_start_int <= '0';
                      IF fp_div_2_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                       fp_div_2_results_read_int <= '1';
                       state_var(4) := state_35;
                      ELSIF fp_mult_results_read_int = '1'  AND
                                   exp_int_results_read_int = '1'  AND
                                   fp_div_1_results_read_int = '1'  AND
                                   fp_div_2_results_read_int = '1'  AND
                                   pow_int_results_read_int = '1'
                      THEN -- all calls are synchronized and completed
                        fp_mult_results_read_int <= '0';
                                   exp_int_results_read_int <= '0';
                                   fp_div_1_results_read_int <= '0';
                                   fp_div_2_results_read_int <= '0';
                                   pow_int_results_read_int <= '0';
                        state_var(4) := state_var(3);
                        IF state_var(3) = state_0 THEN
```

```
                        done_int <= '1';   -- processing finished, results are ready !!!
                        busy <= '0';
                       END IF;
                      END IF;
                     ELSE
                      state_var(4) := state_35;
                     END IF;
            ------ this is a call to module : pow_int -----
          pow_int_num <= temporary032;
          pow_int_pow <= const3;

                     IF pow_int_busy = '0' AND pow_int_done = '0' THEN
                      IF pow_int_results_read_int = '1' THEN
                       IF fp_mult_results_read_int = '1'  AND
                            exp_int_results_read_int = '1'  AND
                            fp_div_1_results_read_int = '1'  AND
                            fp_div_2_results_read_int = '1'  AND
                            pow_int_results_read_int = '1'
                       THEN
                        fp_mult_start_int <= '0';
                            exp_int_start_int <= '0';
                            fp_div_1_start_int <= '0';
                            fp_div_2_start_int <= '0';
                            pow_int_start_int <= '0';
                        fp_mult_results_read_int <= '0';
                            exp_int_results_read_int <= '0';
                            fp_div_1_results_read_int <= '0';
                            fp_div_2_results_read_int <= '0';
                            pow_int_results_read_int <= '0';
                        state_var(5) := state_var(4);
                        IF state_var(4) = state_0 THEN
                         done_int <= '1';   -- processing finished, results are ready !!!
                         busy <= '0';
                        END IF;
                       END IF;
                      ELSE
                        pow_int_start_int <= '1';
                        state_var(5) := state_35;
                      END IF;
                     ELSIF pow_int_busy = '1' AND pow_int_start_int = '1'  THEN   -- when it
begins
                      pow_int_start_int <= '0';
                      state_var(5) := state_35;
                     ELSIF pow_int_done = '1'  THEN   -- when it is completed then read the
results
```

```
          temporary033 <= pow_int_pow_int;
                       pow_int_start_int <= '0';
                       IF pow_int_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                        pow_int_results_read_int <= '1';
                        state_var(5) := state_35;
                       ELSIF fp_mult_results_read_int = '1'  AND
                                    exp_int_results_read_int = '1'  AND
                                    fp_div_1_results_read_int = '1'  AND
                                    fp_div_2_results_read_int = '1'  AND
                                    pow_int_results_read_int = '1'
                       THEN -- all calls are synchronized and completed
                         fp_mult_results_read_int <= '0';
                                    exp_int_results_read_int <= '0';
                                    fp_div_1_results_read_int <= '0';
                                    fp_div_2_results_read_int <= '0';
                                    pow_int_results_read_int <= '0';
                         state_var(5) := state_var(4);
                         IF state_var(4) = state_0 THEN
                          done_int <= '1';   -- processing finished, results are ready !!!
                          busy <= '0';
                         END IF;
                        END IF;
                       ELSE
                        state_var(5) := state_35;
                       END IF;
               state <= state_var(5);
           ELSE
           END IF;
         WHEN state_36 =>
           v007_fcval <= temporary031 - temporary033;

           state <= state_44;
         WHEN state_37 =>
           IF temporary036 /= const5 THEN temporary037 <= '1'; ELSE temporary037 <= '0'; END IF;
           temporary035 <= temporary_mtype + v003_i;
           state <= state_38;
         WHEN state_38 =>
           temporary034 <= temporary037 ;
           state <= state_39;
         WHEN state_39 =>
           IF temporary034 = '1' THEN
           state <= state_40;
           ELSE
           state <= state_41;
```

```
  END IF;
  IF temporary034 = '1' THEN
  temporary038 <= plus32_1_out;
  temporary040 <= notequal32_1_out;
  ELSE
  END IF;
WHEN state_40 =>
  temporary034 <= temporary040 ;
  state <= state_41;
WHEN state_41 =>
  IF temporary034 = '1' THEN
  state <= state_42;
  ELSE
  state <= state_43;
  END IF;
  IF temporary034 = '1' THEN
  v011_test(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;




   ------ this is a call to module : fp_mult -----
 fp_mult_numa <= const11;
 fp_mult_numb <= v009_r;

              IF fp_mult_busy = '0' AND fp_mult_done = '0' THEN
               IF fp_mult_results_read_int = '1' THEN
                IF fp_mult_results_read_int = '1'  AND
                      exp_int_results_read_int = '1'  AND
                      fp_div_1_results_read_int = '1'  AND
                      pow_int_results_read_int = '1'  AND
                      fp_div_2_results_read_int = '1'
                THEN
                 fp_mult_start_int <= '0';
                      exp_int_start_int <= '0';
                      fp_div_1_start_int <= '0';
                      pow_int_start_int <= '0';
                      fp_div_2_start_int <= '0';
                 fp_mult_results_read_int <= '0';
                      exp_int_results_read_int <= '0';
                      fp_div_1_results_read_int <= '0';
                      pow_int_results_read_int <= '0';
                      fp_div_2_results_read_int <= '0';
                 state_var(1) := state_0;
```

```
                        done_int <= '1';   -- processing finished, results are ready !!!
                        busy <= '0';
                       END IF;
                      ELSE
                        fp_mult_start_int <= '1';
                        state_var(1) := state_41;
                      END IF;
                     ELSIF fp_mult_busy = '1' AND fp_mult_start_int = '1'  THEN   -- when it
begins
                      fp_mult_start_int <= '0';
                      state_var(1) := state_41;
                     ELSIF fp_mult_done = '1'  THEN   -- when it is completed then read the
results
          temporary041 <= fp_mult_fp_mult;
                      fp_mult_start_int <= '0';
                      IF fp_mult_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                       fp_mult_results_read_int <= '1';
                       state_var(1) := state_41;
                      ELSIF fp_mult_results_read_int = '1'  AND
                                   exp_int_results_read_int = '1'  AND
                                   fp_div_1_results_read_int = '1'  AND
                                   pow_int_results_read_int = '1'  AND
                                   fp_div_2_results_read_int = '1'
                      THEN -- all calls are synchronized and completed
                        fp_mult_results_read_int <= '0';
                                   exp_int_results_read_int <= '0';
                                   fp_div_1_results_read_int <= '0';
                                   pow_int_results_read_int <= '0';
                                   fp_div_2_results_read_int <= '0';
                        state_var(1) := state_0;
                        done_int <= '1';   -- processing finished, results are ready !!!
                        busy <= '0';
                       END IF;
                      ELSE
                       state_var(1) := state_41;
                      END IF;
            ------ this is a call to module : exp_int -----
          exp_int_pow <= temporary041;

                      IF exp_int_busy = '0' AND exp_int_done = '0' THEN
                       IF exp_int_results_read_int = '1' THEN
                        IF fp_mult_results_read_int = '1'  AND
                              exp_int_results_read_int = '1'  AND
                              fp_div_1_results_read_int = '1'  AND
```

```
                              pow_int_results_read_int = '1'  AND
                              fp_div_2_results_read_int = '1'
                            THEN
                             fp_mult_start_int <= '0';
                              exp_int_start_int <= '0';
                              fp_div_1_start_int <= '0';
                              pow_int_start_int <= '0';
                              fp_div_2_start_int <= '0';
                             fp_mult_results_read_int <= '0';
                              exp_int_results_read_int <= '0';
                              fp_div_1_results_read_int <= '0';
                              pow_int_results_read_int <= '0';
                              fp_div_2_results_read_int <= '0';
                             state_var(2) := state_var(1);
                             IF state_var(1) = state_0 THEN
                              done_int <= '1';   -- processing finished, results are ready !!!
                              busy <= '0';
                             END IF;
                            END IF;
                           ELSE
                             exp_int_start_int <= '1';
                             state_var(2) := state_41;
                           END IF;
                          ELSIF exp_int_busy = '1' AND exp_int_start_int = '1'  THEN   -- when it
begins
                           exp_int_start_int <= '0';
                           state_var(2) := state_41;
                          ELSIF exp_int_done = '1'  THEN   -- when it is completed then read the
results
          v013_e_135 <= exp_int_exp_int;
                           exp_int_start_int <= '0';
                           IF exp_int_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                            exp_int_results_read_int <= '1';
                            state_var(2) := state_41;
                           ELSIF fp_mult_results_read_int = '1'  AND
                                        exp_int_results_read_int = '1'  AND
                                        fp_div_1_results_read_int = '1'  AND
                                        pow_int_results_read_int = '1'  AND
                                        fp_div_2_results_read_int = '1'
                           THEN -- all calls are synchronized and completed
                             fp_mult_results_read_int <= '0';
                                        exp_int_results_read_int <= '0';
                                        fp_div_1_results_read_int <= '0';
                                        pow_int_results_read_int <= '0';
```

```
                                    fp_div_2_results_read_int <= '0';
                        state_var(2) := state_var(1);
                        IF state_var(1) = state_0 THEN
                         done_int <= '1';   -- processing finished, results are ready !!!
                         busy <= '0';
                        END IF;
                       END IF;
                      ELSE
                       state_var(2) := state_41;
                      END IF;
            ------ this is a call to module : fp_div -----
          fp_div_1_numa <= const12;
          fp_div_1_numb <= v013_e_135;

                      IF fp_div_1_busy = '0' AND fp_div_1_done = '0' THEN
                       IF fp_div_1_results_read_int = '1' THEN
                        IF fp_mult_results_read_int = '1'  AND
                              exp_int_results_read_int = '1'  AND
                              fp_div_1_results_read_int = '1'  AND
                              pow_int_results_read_int = '1'  AND
                              fp_div_2_results_read_int = '1'
                        THEN
                         fp_mult_start_int <= '0';
                              exp_int_start_int <= '0';
                              fp_div_1_start_int <= '0';
                              pow_int_start_int <= '0';
                              fp_div_2_start_int <= '0';
                         fp_mult_results_read_int <= '0';
                              exp_int_results_read_int <= '0';
                              fp_div_1_results_read_int <= '0';
                              pow_int_results_read_int <= '0';
                              fp_div_2_results_read_int <= '0';
                         state_var(3) := state_var(2);
                         IF state_var(2) = state_0 THEN
                          done_int <= '1';   -- processing finished, results are ready !!!
                          busy <= '0';
                         END IF;
                        END IF;
                       ELSE
                         fp_div_1_start_int <= '1';
                         state_var(3) := state_41;
                       END IF;
                      ELSIF fp_div_1_busy = '1' AND fp_div_1_start_int = '1'  THEN   -- when it
begins

                       fp_div_1_start_int <= '0';
```

```
                        state_var(3) := state_41;
                       ELSIF fp_div_1_done = '1'   THEN   -- when it is completed then read the
results
           temporary042 <= fp_div_1_fp_div;
                        fp_div_1_start_int <= '0';
                        IF fp_div_1_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                         fp_div_1_results_read_int <= '1';
                         state_var(3) := state_41;
                        ELSIF fp_mult_results_read_int = '1'  AND
                                     exp_int_results_read_int = '1'  AND
                                     fp_div_1_results_read_int = '1'  AND
                                     pow_int_results_read_int = '1'  AND
                                     fp_div_2_results_read_int = '1'
                        THEN -- all calls are synchronized and completed
                          fp_mult_results_read_int <= '0';
                                     exp_int_results_read_int <= '0';
                                     fp_div_1_results_read_int <= '0';
                                     pow_int_results_read_int <= '0';
                                     fp_div_2_results_read_int <= '0';
                          state_var(3) := state_var(2);
                          IF state_var(2) = state_0 THEN
                           done_int <= '1';   -- processing finished, results are ready !!!
                           busy <= '0';
                          END IF;
                        END IF;
                       ELSE
                        state_var(3) := state_41;
                       END IF;
             ------ this is a call to module : pow_int -----
           pow_int_num <= temporary042;
           pow_int_pow <= const3;

                       IF pow_int_busy = '0' AND pow_int_done = '0' THEN
                        IF pow_int_results_read_int = '1' THEN
                         IF fp_mult_results_read_int = '1'  AND
                               exp_int_results_read_int = '1'  AND
                               fp_div_1_results_read_int = '1'  AND
                               pow_int_results_read_int = '1'  AND
                               fp_div_2_results_read_int = '1'
                         THEN
                          fp_mult_start_int <= '0';
                               exp_int_start_int <= '0';
                               fp_div_1_start_int <= '0';
                               pow_int_start_int <= '0';
```

```
                              fp_div_2_start_int <= '0';
                            fp_mult_results_read_int <= '0';
                              exp_int_results_read_int <= '0';
                              fp_div_1_results_read_int <= '0';
                              pow_int_results_read_int <= '0';
                              fp_div_2_results_read_int <= '0';
                            state_var(4) := state_var(3);
                            IF state_var(3) = state_0 THEN
                             done_int <= '1';   -- processing finished, results are ready !!!
                             busy <= '0';
                            END IF;
                           END IF;
                          ELSE
                            pow_int_start_int <= '1';
                            state_var(4) := state_41;
                          END IF;
                         ELSIF pow_int_busy = '1' AND pow_int_start_int = '1'  THEN   -- when it
begins
                          pow_int_start_int <= '0';
                          state_var(4) := state_41;
                         ELSIF pow_int_done = '1'  THEN   -- when it is completed then read the
results
           temporary043 <= pow_int_pow_int;
                          pow_int_start_int <= '0';
                          IF pow_int_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                           pow_int_results_read_int <= '1';
                           state_var(4) := state_41;
                          ELSIF fp_mult_results_read_int = '1'  AND
                                       exp_int_results_read_int = '1'  AND
                                       fp_div_1_results_read_int = '1'  AND
                                       pow_int_results_read_int = '1'  AND
                                       fp_div_2_results_read_int = '1'
                          THEN -- all calls are synchronized and completed
                            fp_mult_results_read_int <= '0';
                                       exp_int_results_read_int <= '0';
                                       fp_div_1_results_read_int <= '0';
                                       pow_int_results_read_int <= '0';
                                       fp_div_2_results_read_int <= '0';
                            state_var(4) := state_var(3);
                            IF state_var(3) = state_0 THEN
                             done_int <= '1';   -- processing finished, results are ready !!!
                             busy <= '0';
                            END IF;
                          END IF;
```

```
                        ELSE
                         state_var(4) := state_41;
                        END IF;
            ------ this is a call to module : fp_div -----
           fp_div_2_numa <= const13;
           fp_div_2_numb <= v013_e_135;

                        IF fp_div_2_busy = '0' AND fp_div_2_done = '0' THEN
                         IF fp_div_2_results_read_int = '1' THEN
                          IF fp_mult_results_read_int = '1'  AND
                                exp_int_results_read_int = '1'  AND
                                fp_div_1_results_read_int = '1'  AND
                                pow_int_results_read_int = '1'  AND
                                fp_div_2_results_read_int = '1'
                           THEN
                            fp_mult_start_int <= '0';
                                exp_int_start_int <= '0';
                                fp_div_1_start_int <= '0';
                                pow_int_start_int <= '0';
                                fp_div_2_start_int <= '0';
                            fp_mult_results_read_int <= '0';
                                exp_int_results_read_int <= '0';
                                fp_div_1_results_read_int <= '0';
                                pow_int_results_read_int <= '0';
                                fp_div_2_results_read_int <= '0';
                            state_var(5) := state_var(4);
                            IF state_var(4) = state_0 THEN
                             done_int <= '1';   -- processing finished, results are ready !!!
                             busy <= '0';
                            END IF;
                          END IF;
                         ELSE
                           fp_div_2_start_int <= '1';
                           state_var(5) := state_41;
                         END IF;
                        ELSIF fp_div_2_busy = '1' AND fp_div_2_start_int = '1'  THEN   -- when it
begins
                         fp_div_2_start_int <= '0';
                         state_var(5) := state_41;
                        ELSIF fp_div_2_done = '1'   THEN   -- when it is completed then read the
results
           temporary044 <= fp_div_2_fp_div;
                         fp_div_2_start_int <= '0';
                         IF fp_div_2_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
```

```
                    fp_div_2_results_read_int <= '1';
                    state_var(5) := state_41;
                  ELSIF fp_mult_results_read_int = '1'  AND
                                exp_int_results_read_int = '1'  AND
                                fp_div_1_results_read_int = '1'  AND
                                pow_int_results_read_int = '1'  AND
                                fp_div_2_results_read_int = '1'
                  THEN -- all calls are synchronized and completed
                    fp_mult_results_read_int <= '0';
                                exp_int_results_read_int <= '0';
                                fp_div_1_results_read_int <= '0';
                                pow_int_results_read_int <= '0';
                                fp_div_2_results_read_int <= '0';
                    state_var(5) := state_var(4);
                    IF state_var(4) = state_0 THEN
                     done_int <= '1';   -- processing finished, results are ready !!!
                     busy <= '0';
                    END IF;
                  END IF;
                 ELSE
                  state_var(5) := state_41;
                 END IF;
        state <= state_var(5);
    ELSE
    END IF;
  WHEN state_42 =>
    v007_fcval <= temporary043 - temporary044;

    state <= state_44;
  WHEN state_43 =>
    v007_fcval(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    state <= state_44;
  WHEN state_44 =>
    v005_k(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    tempint002(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;
    index002(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
    state <= state_45;
  WHEN state_45 =>
    IF index002 <= tempint002 THEN var13 <= '1'; ELSE var13 <= '0'; END IF;
    state <= state_46;
  WHEN state_46 =>
    IF var13 = '1' THEN
    state <= state_47;
    ELSE
    state <= state_55;
```

```
            END IF;
            IF var13 = '1' THEN
            var14 <= mult32_1_out;
            ELSE
            END IF;
          WHEN state_47 =>
            temporary045 <= v003_i + var14;
            temporary046 <= v008_dr(CONV_INTEGER(v005_k));
            ------ this is a call to module : fp_div -----
           fp_div_numa <= temporary046;
           fp_div_numb <= v009_r;

                       IF fp_div_busy = '0' AND fp_div_done = '0' THEN
                        IF fp_div_results_read_int = '1' THEN
                         IF fp_div_results_read_int = '1'  AND
                                fp_mult_results_read_int = '1'
                         THEN
                          fp_div_start_int <= '0';
                                fp_mult_start_int <= '0';
                          fp_div_results_read_int <= '0';
                                fp_mult_results_read_int <= '0';
                          state_var(1) := state_0;
                          done_int <= '1';   -- processing finished, results are ready !!!
                          busy <= '0';
                         END IF;
                        ELSE
                          fp_div_start_int <= '1';
                          state_var(1) := state_47;
                        END IF;
                       ELSIF fp_div_busy = '1' AND fp_div_start_int = '1' THEN   -- when it begins
                        fp_div_start_int <= '0';
                        state_var(1) := state_47;
                       ELSIF fp_div_done = '1'  THEN   -- when it is completed then read the
results
            temporary047 <= fp_div_fp_div;
                        fp_div_start_int <= '0';
                        IF fp_div_results_read_int = '0' THEN  -- if it is done then indicate that
the results are read
                         fp_div_results_read_int <= '1';
                         state_var(1) := state_47;
                        ELSIF fp_div_results_read_int = '1'  AND
                                       fp_mult_results_read_int = '1'
                        THEN -- all calls are synchronized and completed
                          fp_div_results_read_int <= '0';
                                       fp_mult_results_read_int <= '0';
```

```
                        state_var(1) := state_0;
                        done_int <= '1';   -- processing finished, results are ready !!!
                        busy <= '0';
                       END IF;
                      ELSE
                       state_var(1) := state_47;
                      END IF;
           ------ this is a call to module : fp_mult -----
          fp_mult_numa <= v007_fcval;
          fp_mult_numb <= temporary047;

                      IF fp_mult_busy = '0' AND fp_mult_done = '0' THEN
                       IF fp_mult_results_read_int = '1' THEN
                        IF fp_div_results_read_int = '1'  AND
                               fp_mult_results_read_int = '1'
                        THEN
                         fp_div_start_int <= '0';
                               fp_mult_start_int <= '0';
                         fp_div_results_read_int <= '0';
                               fp_mult_results_read_int <= '0';
                         state_var(2) := state_var(1);
                         IF state_var(1) = state_0 THEN
                          done_int <= '1';   -- processing finished, results are ready !!!
                          busy <= '0';
                         END IF;
                        END IF;
                       ELSE
                         fp_mult_start_int <= '1';
                         state_var(2) := state_47;
                       END IF;
                      ELSIF fp_mult_busy = '1' AND fp_mult_start_int = '1' THEN   -- when it
begins
                       fp_mult_start_int <= '0';
                       state_var(2) := state_47;
                      ELSIF fp_mult_done = '1'   THEN   -- when it is completed then read the
results
          temporary048 <= fp_mult_fp_mult;
                       fp_mult_start_int <= '0';
                       IF fp_mult_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                        fp_mult_results_read_int <= '1';
                        state_var(2) := state_47;
                       ELSIF fp_div_results_read_int = '1'  AND
                                      fp_mult_results_read_int = '1'
                       THEN -- all calls are synchronized and completed
```

```
                    fp_div_results_read_int <= '0';
                                fp_mult_results_read_int <= '0';
                    state_var(2) := state_var(1);
                    IF state_var(1) = state_0 THEN
                     done_int <= '1';   -- processing finished, results are ready !!!
                     busy <= '0';
                    END IF;
                   END IF;
                  ELSE
                   state_var(2) := state_47;
                  END IF;
        state <= state_var(2);
  WHEN state_48 =>
    temporary049 <= v010_ra(CONV_INTEGER(temporary045));
    state <= state_49;
  WHEN state_49 =>
    temporary049 <= temporary049 + temporary048;
    state <= state_50;
  WHEN state_50 =>
    v010_ra(CONV_INTEGER(temporary045)) <= temporary049;
    var15 <= conv_std_logic_vector(CONV_INTEGER(v005_k) * CONV_INTEGER(const4), 32);
    state <= state_51;
  WHEN state_51 =>
    temporary050 <= v004_j + var15;
    temporary051 <= v008_dr(CONV_INTEGER(v005_k));
    ------ this is a call to module : fp_div -----
   fp_div_numa <= temporary051;
   fp_div_numb <= v009_r;

                  IF fp_div_busy = '0' AND fp_div_done = '0' THEN
                   IF fp_div_results_read_int = '1' THEN
                    IF fp_div_results_read_int = '1'  AND
                           fp_mult_results_read_int = '1'
                    THEN
                     fp_div_start_int <= '0';
                           fp_mult_start_int <= '0';
                     fp_div_results_read_int <= '0';
                           fp_mult_results_read_int <= '0';
                     state_var(1) := state_0;
                     done_int <= '1';   -- processing finished, results are ready !!!
                     busy <= '0';
                    END IF;
                   ELSE
                     fp_div_start_int <= '1';
                     state_var(1) := state_51;
```

```
                        END IF;
                       ELSIF fp_div_busy = '1' AND fp_div_start_int = '1' THEN   -- when it begins
                        fp_div_start_int <= '0';
                        state_var(1) := state_51;
                       ELSIF fp_div_done = '1'  THEN   -- when it is completed then read the
results
           temporary052 <= fp_div_fp_div;
                        fp_div_start_int <= '0';
                        IF fp_div_results_read_int = '0' THEN  -- if it is done then indicate that
the results are read
                         fp_div_results_read_int <= '1';
                         state_var(1) := state_51;
                        ELSIF fp_div_results_read_int = '1'  AND
                                       fp_mult_results_read_int = '1'
                        THEN -- all calls are synchronized and completed
                          fp_div_results_read_int <= '0';
                                       fp_mult_results_read_int <= '0';
                          state_var(1) := state_0;
                          done_int <= '1';   -- processing finished, results are ready !!!
                          busy <= '0';
                        END IF;
                       ELSE
                        state_var(1) := state_51;
                       END IF;
            ------ this is a call to module : fp_mult -----
           fp_mult_numa <= v007_fcval;
           fp_mult_numb <= temporary052;

                       IF fp_mult_busy = '0' AND fp_mult_done = '0' THEN
                        IF fp_mult_results_read_int = '1' THEN
                         IF fp_div_results_read_int = '1'  AND
                                fp_mult_results_read_int = '1'
                         THEN
                          fp_div_start_int <= '0';
                                fp_mult_start_int <= '0';
                          fp_div_results_read_int <= '0';
                                fp_mult_results_read_int <= '0';
                          state_var(2) := state_var(1);
                          IF state_var(1) = state_0 THEN
                           done_int <= '1';   -- processing finished, results are ready !!!
                           busy <= '0';
                          END IF;
                         END IF;
                        ELSE
                          fp_mult_start_int <= '1';
```

```
                        state_var(2) := state_51;
                      END IF;
                     ELSIF fp_mult_busy = '1' AND fp_mult_start_int = '1' THEN   -- when it
begins
                      fp_mult_start_int <= '0';
                      state_var(2) := state_51;
                     ELSIF fp_mult_done = '1'   THEN   -- when it is completed then read the
results
          temporary053 <= fp_mult_fp_mult;
                      fp_mult_start_int <= '0';
                      IF fp_mult_results_read_int = '0' THEN  -- if it is done then indicate
that the results are read
                       fp_mult_results_read_int <= '1';
                       state_var(2) := state_51;
                      ELSIF fp_div_results_read_int = '1'  AND
                                    fp_mult_results_read_int = '1'
                      THEN -- all calls are synchronized and completed
                        fp_div_results_read_int <= '0';
                                    fp_mult_results_read_int <= '0';
                        state_var(2) := state_var(1);
                        IF state_var(1) = state_0 THEN
                         done_int <= '1';   -- processing finished, results are ready !!!
                         busy <= '0';
                        END IF;
                      END IF;
                     ELSE
                      state_var(2) := state_51;
                     END IF;
               state <= state_var(2);
          WHEN state_52 =>
            temporary054 <= v010_ra(CONV_INTEGER(temporary050));
            state <= state_53;
          WHEN state_53 =>
            temporary054 <= temporary054 - temporary053;
            state <= state_54;
          WHEN state_54 =>
            v010_ra(CONV_INTEGER(temporary050)) <= temporary054;
            v005_k <= v005_k + const5;
            index002 <= index002 + 1;

            state <= state_45;
          WHEN state_55 =>
            v004_j <= v004_j + const5;
            index001 <= index001 + 1;
```

```
        state <= state_13;
      WHEN state_56 =>
        v003_i <= v003_i + const5;
        index000 <= index000 + 1;

        state <= state_10;
      WHEN state_57 =>
        index000(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
        tempint000(31 DOWNTO 0) <= const3(31 DOWNTO 0) ;
        v006_m(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
        state <= state_58;
      WHEN state_58 =>
        IF index000 <= tempint000 THEN var16 <= '1'; ELSE var16 <= '0'; END IF;
        state <= state_59;
      WHEN state_59 =>
        IF var16 = '1' THEN
        state <= state_60;
        ELSE
        state <= state_71;
        END IF;
        IF var16 = '1' THEN
        v004_j(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
        tempint001(31 DOWNTO 0) <= const2(31 DOWNTO 0) ;
        index001(31 DOWNTO 0) <= const1(31 DOWNTO 0) ;
        ELSE
        END IF;
      WHEN state_60 =>
        IF index001 <= tempint001 THEN var17 <= '1'; ELSE var17 <= '0'; END IF;
        state <= state_61;
      WHEN state_61 =>
        IF var17 = '1' THEN
        state <= state_62;
        ELSE
        state <= state_70;
        END IF;
        IF var17 = '1' THEN
        temporary056 <= plus32_1_out;
        temporary058 <= equal32_1_out;
        ELSE
        END IF;
      WHEN state_62 =>
        temporary055 <= not temporary058 ;
        state <= state_63;
      WHEN state_63 =>
        IF temporary055 = '1' THEN
```

```
          state <= state_64;
          ELSE
          state <= state_65;
          END IF;
          IF temporary055 = '1' THEN
          temporary059 <= plus32_1_out;
          temporary061 <= equal32_1_out;
          ELSE
          END IF;
        WHEN state_64 =>
          temporary055 <= temporary061 ;

          state <= state_66;
        WHEN state_65 =>
          temporary055 <= not temporary055 ;
          state <= state_66;
        WHEN state_66 =>
          IF temporary055 = '1' THEN
          state <= state_67;
          ELSE
          state <= state_69;
          END IF;
          IF temporary055 = '1' THEN
          var18 <= mult32_1_out;
          ELSE
          END IF;
        WHEN state_67 =>
          temporary062 <= v004_j + var18;
          state <= state_68;
        WHEN state_68 =>
          v010_ra(CONV_INTEGER(temporary062)) <= const1;
          state <= state_69;
        WHEN state_69 =>
          v004_j <= v004_j + const5;
          index001 <= index001 + 1;

          state <= state_60;
        WHEN state_70 =>
          v006_m <= v006_m + const5;
          index000 <= index000 + 1;

          state <= state_58;
        WHEN state_71 =>
          computeforces(31 DOWNTO 0) <= v010_ra(31 DOWNTO 0) ;
          state <= state_0;
```

```
            done_int <= '1';    -- processing finished, results are ready !!!
            busy <= '0';

         WHEN OTHERS => state <= state_0;
                        done_int <= '0';
                        busy <= '0';

        END CASE ;

      END IF ;
     END PROCESS fsm_core ;



--  Datapath operators (Functional Units) --
  lessequal32_1_out <= '1' WHEN lessequal32_1_in1 <= lessequal32_1_in2 ELSE '0';
  mult32_1_out <= conv_std_logic_vector(CONV_INTEGER(mult32_1_in1) * CONV_INTEGER(mult32_1_in2),
32);
  plus32_1_out <= plus32_1_in1 + plus32_1_in2;
  plusone32_1_out <= plusone32_1_in1 + plusone32_1_in2;
  minus32_1_out <= minus32_1_in1 - minus32_1_in2;
  plus32_2_out <= plus32_2_in1 + plus32_2_in2;
  less32_1_out <= '1' WHEN less32_1_in1 < less32_1_in2 ELSE '0';
  equal32_1_out <= '1' WHEN equal32_1_in1 = equal32_1_in2 ELSE '0';
  notequal32_1_out <= '1' WHEN notequal32_1_in1 /= notequal32_1_in2 ELSE '0';
  not1_1_out <= not not1_1_in2;


-- now the data routing --

-- now the less-equality comparisons --
  routing_lessequal32_1_in1:
   WITH state SELECT
   lessequal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_2,
                        conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_4,
                        conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_10,
                        conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_13,
                        conv_std_logic_vector(CONV_INTEGER(index002), 32) WHEN state_15,
                        conv_std_logic_vector(CONV_INTEGER(index002), 32) WHEN state_45,
                        conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_58,
                        conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_60,
                         (OTHERS => '0') WHEN OTHERS;
  routing_lessequal32_1_in2:
   WITH state SELECT
   lessequal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(tempint000), 32) WHEN state_2,
```

```
                    conv_std_logic_vector(CONV_INTEGER(tempint001), 32) WHEN state_4,
                    conv_std_logic_vector(CONV_INTEGER(tempint000), 32) WHEN state_10,
                    conv_std_logic_vector(CONV_INTEGER(tempint001), 32) WHEN state_13,
                    conv_std_logic_vector(CONV_INTEGER(tempint002), 32) WHEN state_15,
                    conv_std_logic_vector(CONV_INTEGER(tempint002), 32) WHEN state_45,
                    conv_std_logic_vector(CONV_INTEGER(tempint000), 32) WHEN state_58,
                    conv_std_logic_vector(CONV_INTEGER(tempint001), 32) WHEN state_60,
                     (OTHERS => '0') WHEN OTHERS;


-- now the multiplications --
  routing_mult32_1_in1:
   WITH state SELECT
   mult32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_5,
                    conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_16,
                    conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_17,
                    conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_46,
                    conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_50,
                    conv_std_logic_vector(CONV_INTEGER(v006_m), 32) WHEN state_66,
                     (OTHERS => '0') WHEN OTHERS;
  routing_mult32_1_in2:
   WITH state SELECT
   mult32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_5,
                    conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_16,
                    conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_17,
                    conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_46,
                    conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_50,
                    conv_std_logic_vector(CONV_INTEGER(const4), 32) WHEN state_66,
                     (OTHERS => '0') WHEN OTHERS;


-- now the additions --
  routing_plus32_1_in1:
   WITH state SELECT
   plus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_6,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_7,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_8,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_11,
                    conv_std_logic_vector(CONV_INTEGER(temporary_mr), 32) WHEN state_16,
                    conv_std_logic_vector(CONV_INTEGER(var7), 32) WHEN state_17,
                    conv_std_logic_vector(CONV_INTEGER(var9), 32) WHEN state_18,
                    conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_19,
                    conv_std_logic_vector(CONV_INTEGER(temporary007), 32) WHEN state_21,
                    conv_std_logic_vector(CONV_INTEGER(var11), 32) WHEN state_22,
                    conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_24,
```

```
                    conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_26,
                    conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_29,
                    conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_31,
                    conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_37,
                    conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_39,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_47,
                    conv_std_logic_vector(CONV_INTEGER(temporary049), 32) WHEN state_49,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_51,
                    conv_std_logic_vector(CONV_INTEGER(v005_k), 32) WHEN state_54,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_55,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_56,
                    conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_61,
                    conv_std_logic_vector(CONV_INTEGER(temporary_mtype), 32) WHEN state_63,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_67,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_69,
                    conv_std_logic_vector(CONV_INTEGER(v006_m), 32) WHEN state_70,
                     (OTHERS => '0') WHEN OTHERS;
routing_plus32_1_in2:
 WITH state SELECT
 plus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(var3), 32) WHEN state_6,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_7,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_8,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_11,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_16,
                    conv_std_logic_vector(CONV_INTEGER(var8), 32) WHEN state_17,
                    conv_std_logic_vector(CONV_INTEGER(var10), 32) WHEN state_18,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_19,
                    conv_std_logic_vector(CONV_INTEGER(temporary009), 32) WHEN state_21,
                    conv_std_logic_vector(CONV_INTEGER(temporary011), 32) WHEN state_22,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_24,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_26,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_29,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_31,
                    conv_std_logic_vector(CONV_INTEGER(v003_i), 32) WHEN state_37,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_39,
                    conv_std_logic_vector(CONV_INTEGER(var14), 32) WHEN state_47,
                    conv_std_logic_vector(CONV_INTEGER(temporary048), 32) WHEN state_49,
                    conv_std_logic_vector(CONV_INTEGER(var15), 32) WHEN state_51,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_54,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_55,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_56,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_61,
                    conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_63,
                    conv_std_logic_vector(CONV_INTEGER(var18), 32) WHEN state_67,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_69,
```

```
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_70,
                     (OTHERS => '0') WHEN OTHERS;


-- now the increments by one --
  routing_plusone32_1_in1:
   WITH state SELECT
   plusone32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_7,
                    conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_8,
                    conv_std_logic_vector(CONV_INTEGER(index002), 32) WHEN state_19,
                    conv_std_logic_vector(CONV_INTEGER(index002), 32) WHEN state_54,
                    conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_55,
                    conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_56,
                    conv_std_logic_vector(CONV_INTEGER(index001), 32) WHEN state_69,
                    conv_std_logic_vector(CONV_INTEGER(index000), 32) WHEN state_70,
                     (OTHERS => '0') WHEN OTHERS;
  routing_plusone32_1_in2:
   WITH state SELECT
   plusone32_1_in2 <= conv_std_logic_vector(1, 32) WHEN state_7,
                    conv_std_logic_vector(1, 32) WHEN state_8,
                    conv_std_logic_vector(1, 32) WHEN state_19,
                    conv_std_logic_vector(1, 32) WHEN state_54,
                    conv_std_logic_vector(1, 32) WHEN state_55,
                    conv_std_logic_vector(1, 32) WHEN state_56,
                    conv_std_logic_vector(1, 32) WHEN state_69,
                    conv_std_logic_vector(1, 32) WHEN state_70,
                     (OTHERS => '0') WHEN OTHERS;


-- now the subtractions --
  routing_minus32_1_in1:
   WITH state SELECT
   minus32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(const6), 32) WHEN state_12,
                    conv_std_logic_vector(CONV_INTEGER(temporary002), 32) WHEN state_18,
                    conv_std_logic_vector(CONV_INTEGER(temporary031), 32) WHEN state_36,
                    conv_std_logic_vector(CONV_INTEGER(temporary043), 32) WHEN state_42,
                    conv_std_logic_vector(CONV_INTEGER(temporary054), 32) WHEN state_53,
                     (OTHERS => '0') WHEN OTHERS;
  routing_minus32_1_in2:
   WITH state SELECT
   minus32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_12,
                    conv_std_logic_vector(CONV_INTEGER(temporary004), 32) WHEN state_18,
                    conv_std_logic_vector(CONV_INTEGER(temporary033), 32) WHEN state_36,
                    conv_std_logic_vector(CONV_INTEGER(temporary044), 32) WHEN state_42,
                    conv_std_logic_vector(CONV_INTEGER(temporary053), 32) WHEN state_53,
```

```
                    (OTHERS => '0') WHEN OTHERS;


-- now the additions --
  routing_plus32_2_in1:
   WITH state SELECT
   plus32_2_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary_mr), 32) WHEN state_17,
                    (OTHERS => '0') WHEN OTHERS;
  routing_plus32_2_in2:
   WITH state SELECT
   plus32_2_in2 <= conv_std_logic_vector(CONV_INTEGER(v004_j), 32) WHEN state_17,
                    (OTHERS => '0') WHEN OTHERS;


-- now the less-than comparisons --
  routing_less32_1_in1:
   WITH state SELECT
   less32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary013), 32) WHEN state_23,
                    (OTHERS => '0') WHEN OTHERS;
  routing_less32_1_in2:
   WITH state SELECT
   less32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(temporary014), 32) WHEN state_23,
                    (OTHERS => '0') WHEN OTHERS;


-- now the equality comparisons --
  routing_equal32_1_in1:
   WITH state SELECT
   equal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary018), 32) WHEN state_24,
                    conv_std_logic_vector(CONV_INTEGER(temporary028), 32) WHEN state_31,
                    conv_std_logic_vector(CONV_INTEGER(temporary057), 32) WHEN state_61,
                    conv_std_logic_vector(CONV_INTEGER(temporary060), 32) WHEN state_63,
                     (OTHERS => '0') WHEN OTHERS;
  routing_equal32_1_in2:
   WITH state SELECT
   equal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_24,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_31,
                    conv_std_logic_vector(CONV_INTEGER(const14), 32) WHEN state_61,
                    conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_63,
                     (OTHERS => '0') WHEN OTHERS;


-- now the non-equality comparisons --
  routing_notequal32_1_in1:
   WITH state SELECT
```

```
  notequal32_1_in1 <= conv_std_logic_vector(CONV_INTEGER(temporary021), 32) WHEN state_26,
                      conv_std_logic_vector(CONV_INTEGER(temporary025), 32) WHEN state_29,
                      conv_std_logic_vector(CONV_INTEGER(temporary036), 32) WHEN state_37,
                      conv_std_logic_vector(CONV_INTEGER(temporary039), 32) WHEN state_39,
                       (OTHERS => '0') WHEN OTHERS;
 routing_notequal32_1_in2:
  WITH state SELECT
  notequal32_1_in2 <= conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_26,
                      conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_29,
                      conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_37,
                      conv_std_logic_vector(CONV_INTEGER(const5), 32) WHEN state_39,
                       (OTHERS => '0') WHEN OTHERS;



 END  rtl ;
```

Report του Εργαλείου CCC

```
-------------------------------------------------------------------
 home directory = C:\xampp\htdocs\ccc\users\dac4a94e92e2964e238dbb329f83e893\uploads\run\new_func\
 executable name = backend.exe
-------------------------------------------------------------------


-------------------------------------------------------------------
  ITF file itf_fact.dba was processed on: 18/2/2023
   backend compilation started successfully at: 18:54:10

-------------------------------------------------------------------

  Global constraint from command-line is
         1000000  parallel operations at maximum
-------------------------------------------------------------------


  ----- Design module: fp_mult
                  ( initial schedule: 6 FSM states in total)
                   ( reduced schedule: to 5 optimized states )


  ----- Design module: fp_div
                  ( initial schedule: 6 FSM states in total)
                   ( reduced schedule: to 4 optimized states )
```

```
  ----- Design module: sqrt2
                  ( initial schedule: 15 FSM states in total)
                   ( reduced schedule: to 8 optimized states )


  ----- Design module: pow_int
                  ( initial schedule: 37 FSM states in total)
                   ( reduced schedule: to 22 optimized states )


  ----- Design module: exp_int
                  ( initial schedule: 32 FSM states in total)
                   ( reduced schedule: to 15 optimized states )


  ----- Design module: computeforces
                  ( initial schedule: 167 FSM states in total)
                   ( reduced schedule: to 71 optimized states )


-----------------------------------------------------------------
   backend compilation completed on : 18/2/2023
   backend compilation completed at: 18:54:12

-----------------------------------------------------------------
```