ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Αρχιτεκτονική Σχεδίαση, Υλοποίηση και Εφαρμογή Ευφυών Αλγορίθμων Ελέγχου σε Ψηφιακά Συστήματα VLSI

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Κυριάκος Μ. Δεληπαράσχος

Αθήνα, Απρίλιος 2010

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Αρχιτεκτονική Σχεδίαση, Υλοποίηση και Εφαρμογή Ευφυών Αλγορίθμων Ελέγχου σε Ψηφιακά Συστήματα VLSI

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Κυριάκος Μ. Δεληπαράσχος

Συμβουλευτική Επιτροπή : Σπυρίδων Γ. Τζαφέστας, Ομότ. Καθηγητής ΕΜΠ
Πέτρος Μαραγκός, Καθηγητής ΕΜΠ
Τρύφων Κουσιουρής, Καθηγητής ΕΜΠ

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 28η Απριλίου 2010

.....	.....	.....
Σ. Τζαφέστας	Π. Μαραγκός	Τ. Κουσιουρής
Ομότ. Καθ. ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ
.....	.....	.....
Γ. Παπαβασιλόπουλος	Γ. Στασινόπουλος	Κ. Τζαφέστας
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Επικ. Καθηγητής ΕΜΠ

.....
Δ. Ευαγγελάτος
Επικ. Καθηγητής ΕΚΠΑ

Αθήνα, Απρίλιος 2010

.....
Κυριάκος Μ. Δεληπαράσχος

Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Στη μνήμη της γιαγιάς μου,
Αναστασίας Καλλιοντζή

Η σελίδα αυτή αφέθηκε σκόπιμα κενή

This page was left blank intentionally

Περίληψη

Ένα από τα πιο σοβαρά μειονεκτήματα των ευφυών αλγορίθμων ελέγχου που έχουν αναπτυχθεί σε λογισμικό, είναι κυρίως ο χρόνος εκτέλεσής τους και η αυξημένη ανάγκη υπολογιστικών πόρων. Για παράδειγμα, στην περίπτωση των γενετικών αλγορίθμων η σύγκλισή τους προς το βέλτιστο μπορεί να είναι υπερβολικά αργή για δύσκολα και περίπλοκα προβλήματα βελτιστοποίησης, με αποτέλεσμα να είναι αδύνατη η χρήση τους σε εφαρμογές πραγματικού χρόνου. Έτσι γίνεται άμεσα αντιληπτό ότι η εφαρμογή των αλγορίθμων αυτών σε ρομποτικές εφαρμογές πραγματικού χρόνου (real-time) καθιστά τις υλοποιήσεις σε λογισμικό ανεπαρκείς. Βάση του τελευταίου, τα τελευταία χρόνια λόγω της ταχείας ανάπτυξης στην τεχνολογία των ψηφιακών κυκλωμάτων, έχει αναφερθεί ένας σημαντικά μεγάλος αριθμός ερευνητικών εργασιών που ασχολούνται με υλοποιήσεις ευφυών αλγορίθμων σε υλικό.

Η υλοποίηση τέτοιων αλγορίθμων σε υλικό προσφέρει σημαντική αύξηση στην ταχύτητα επεξεργασίας των δεδομένων λόγω της ενδογενούς παραλληλίας που προσφέρει η ψηφιακή σχεδίαση επιτρέποντάς τους έτσι να χρησιμοποιηθούν ικανοποιητικά σε εφαρμογές πραγματικού χρόνου και αυξημένης υπολογιστικής πολυπλοκότητας. Η δημιουργία ξεχωριστών πυρήνων (cores) διαφορετικών ευφυών αλγορίθμων επιτρέπει την εύκολη ενσωμάτωσή τους με άλλες δομικές μονάδες (π.χ., πυρήνες μικροεπεξεργαστών) για την υλοποίηση συστημάτων σε ψηφίδα (System on a Chip – SoC) που τελικά μπορούν να ολοκληρώσουν μια αυτόνομη υπολογιστική πλατφόρμα. Επιπρόσθετα, οι πυρήνες αυτοί μπορούν να χρησιμοποιηθούν σε μελλοντικές εφαρμογές αυξάνοντας έτσι τη δυνατότητα επαναχρησιμότητας της σχεδίασης (design reusability). Τέλος, η υλοποίησή τους σε ολοκληρωμένα κυκλώματα προγραμματιζόμενης λογικής (Field Programmable Gate Arrays – FPGAs) συντελεί στη σημαντική μείωση της απαιτούμενης ισχύος λειτουργίας, τη σημαντική μείωση του μεγέθους, τη δυνατότητα λειτουργίας σε δύσκολα περιβάλλοντα, τη μείωση κόστους και την εύκολη μεταφορά τους σε δομημένα ολοκληρωμένα κυκλώματα ASIC (structured Application Specific Integrated Circuits) εάν αυτό απαιτείται.

Στην παρούσα εργασία παρουσιάζονται νέες αρχιτεκτονικές για τη σχεδίαση ασαφών ελεγκτών και γενετικών αλγορίθμων σε υλικό με τη χρήση γλωσσών περιγραφής υλικού (Hardware Description Languages – HDLs) και εργαλεία αυτοματοποίησης της σχεδίασης (Electronic Design Automation – EDA tools). Πιο συγκεκριμένα παρουσιάζεται η αρχιτεκτονική σχεδίαση και υλοποίηση ενός παραμετρικού πυρήνα ασαφούς ελεγκτή τύπου Takagi-Sugeno μηδενικού-βαθμού, που επεξεργάζεται μόνο τους ενεργούς κανόνες και επιτυγχάνει υψηλή συχνότητα λειτουργίας. Στη συνέχεια δίνεται μια τροποποιημένη εκδοχή του πυρήνα αυτού χρησιμοποιώντας μια μέθοδο που αυξάνει την παραλληλία της σχεδίασης και επιτυγχάνει διπλάσιο ρυθμό επεξεργασίας δεδομένων μέσω της ταυτόχρονης επεξεργασίας στην είσοδο του ελεγκτή περισσότερων από ένα ενεργών

κανόνων σε κάθε κύκλο ρολογιού. Στη συνέχεια, ο πυρήνας ασαφούς ελεγκτή συνδέθηκε με έναν πυρήνα μικροεπεξεργαστή και άλλες δευτερεύουσες δομικές μονάδες για να αποτελέσουν ένα SoC που ολοκληρώνει μία ρομποτική πλατφόρμα παρακολούθησης πορείας με τη χρήση ασαφούς λογικής για αυτόνομα κινητά ρομπότ. Η συγκεκριμένη πλατφόρμα προσφέρει αυξημένη δυνατότητα επεξεργασίας και ευέλικτο υλικό για διαφορετικές διεργασίες. Επιπρόσθετα, το αναφερόμενο SoC προσαρμόστηκε πάνω σε ένα κινητό ρομπότ Pioneer P3-DX8 και στη συνέχεια εκτελέστηκαν διάφορα πειράματα σε εσωτερικό και εξωτερικό χώρο, ούτως ώστε να γίνει αποτίμηση της γενικής απόδοσης του συστήματος. Τέλος, στην παρούσα εργασία αναλύεται η αρχιτεκτονική σχεδίαση και υλοποίηση ενός πυρήνα Γενετικού Αλγορίθμου που επιτυγχάνει μεγάλη συχνότητα λειτουργίας και εκμεταλλεύεται την παραλληλία που προσφέρει η σχεδίαση σε υλικό δίνοντάς του τη δυνατότητα να χρησιμοποιηθεί σε εφαρμογές πραγματικού χρόνου. Ο πυρήνας αξιολογείται με τη χρήση συναρτήσεων σύγκρισης και με την εφαρμογή του πυρήνα στην επίλυση του προβλήματος του Πλανόδιου Πωλητή για διαφορετικό αριθμό πόλεων.

Λέξεις Κλειδιά: Ασαφής Λογική, Γενετικοί Αλγόριθμοι, Κινητά Ρομπότ, Γλώσσες Περιγραφής Υλικού, Σύνθεση λογικών κυκλωμάτων, Εργαλεία EDA, FPGA, SoC, Core, Πρόβλημα Πλανόδιου Πωλητή.

Abstract

Software implementations of intelligent control algorithms suffer slow execution time and increased resources demand. Referring to Genetic Algorithms, convergence to the optimal point can prove extremely time-consuming for hard or complex optimization problems, hence not allowing their use in real-time applications. Therefore, it is obvious that applying these algorithms, implemented in software, in real-time robotic applications is practically unfeasible. Based on the last fact, and the fast growth in digital circuit technology, a large number of research work dealing with intelligent control algorithms in programmable logic chips such as Field Programmable Gate Arrays or FPGAs, have been conducted and published over the last few years.

The implementation of such algorithms in hardware, offers a substantial increase of data processing speed due to the inherent parallelism of the logic resources into the FPGA that allows for considerable computational throughput, thus rendering them capable of being used in real-time and increased computational complexity applications. Various intelligent algorithm cores can be easily combined with other core modules (e.g., microprocessor core) in order to form a System on a Chip (SoC), which it can be part of an autonomous robotic platform. Moreover, these cores could be used in future applications, thus increasing design re-usability. Finally, intelligent control algorithms implementation on FPGA devices helps to secure reduced power consumption, size, and cost, operation in harsh environments, and easy transfer to structured Application Specific Integrated Circuits (ASICs) if necessary.

In this work, several new architectures for the design of intelligent control algorithms are proposed, specifically fuzzy controllers and genetic algorithm cores with the use of Hardware Description Languages (HDLs) and Electronic Design Automation (EDA) tools. In particular, the architectural design and implementation of a parameterized zero-order Takagi-Sugeno Digital Fuzzy Controller (DFLC) core that processes only the active rules is presented which achieves a high clock frequency. Thereinafter, a modified version of the DFLC is presented using a method that increases the parallelism of the architecture and achieves twice the data processing rate of the first core, by processing more than one active rule at the input of the controller per clock cycle. The fuzzy controller core was successfully bound with a microprocessor core and other secondary modules into a SoC to be eventually integrated in a robotic platform for path tracking problems. This SoC offers increased data processing and flexible hardware for different tasks. The SoC was embodied onto a Pioneer P3-DX8 mobile robot and several experiments were performed to evaluate the system's overall performance. Finally in this work the design, implementation and performance evaluation of a Genetic Algorithm (GA) core is being analyzed. The GA core presented here possesses a high frequency of operation and logic design parallelism,

allowing it to be effectively used in real-time applications. The core was evaluated using several benchmarking functions and solving the Travelling Salesman Problem (TSP) for a different number of cities.

Keywords: Fuzzy Logic, Genetic Algorithms, Mobile Robots, Hardware Description Languages, Logic Synthesis, EDA tools, FPGA, SoC, Core, TSP.

Περιεχόμενα

Η σελίδα αυτή αφέθηκε σκόπιμα κενή

This page was left blank intentionally

Λίστα Σχημάτων

Λίστα Πινάκων

Αντί Προλόγου

Η παρούσα διδακτορική διατριβή εκπονήθηκε στον τομέα Σημάτων, Ελέγχου και Ρομποτικής, της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, του Εθνικού Μετσόβιου Πολυτεχνείου. Περιλαμβάνει την έρευνα και τα αποτελέσματα που προέκυψαν κατά τη διάρκεια των μεταπτυχιακών μου σπουδών στο εργαστήριο Ρομποτικής και Αυτοματισμού της σχολής αυτής.

Στις ακόλουθες γραμμές θα ήθελα να εκφράσω τις ευχαριστίες μου σε ένα πλήθος ανθρώπων που συνέβαλαν είτε άμεσα με την καθοδήγηση που μου προσέφεραν, είτε έμμεσα με την ηθική στήριξή τους και βοήθησαν ουσιαστικά στην ολοκλήρωση της παρούσας εργασίας. Πρώτον από όλους θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Σπυρίδωνα Τζαφέστα για τη σημαντικότατη επιστημονική καθοδήγηση καθ'όλη τη διάρκεια της έρευνάς μου αλλά και για την εμπιστοσύνη την οποία μου έδειξε και την αυτοπεποίθηση που μου μετέδιδε. Είναι ο άνθρωπος που μου έδωσε την ευκαιρία να διευρύνω τους επιστημονικούς μου ορίζοντες, να συμμετάσχω σε συνέδρια του εξωτερικού και να έρθω σε επαφή με άλλους επιστήμονες στο πεδίο της επιστήμης μου. Ειλικρινά τον ευχαριστώ για το χρόνο που διέθεσε για να παρακολουθήσει την έρευνά μου. Στη συνέχεια θα ήθελα να εκφράσω τις θερμές ευχαριστίες μου στο δεύτερο και τρίτο μέλος της συμβουλευτικής επιτροπής μου, τους καθηγητές Πέτρο Μαραγκό και Τρύφωνα Κουσιουρή. Η παρουσία τους στην τριμελή επιτροπή μου είναι εγγύηση επιστημονικής αρτιότητας.

Ιδιαίτερη αναφορά θα ήθελα επίσης να κάνω και στον υποψήφιο Διδάκτορα και φίλο Γιώργο Μούστρη, που η συνεργασία μαζί του αποδείχθηκε ιδιαίτερα δημιουργική. Η συνεισφορά του στην παρούσα εργασία ήταν πολύτιμη. Φυσικά θα ήταν παράλειψη να μην αναφερθώ ξεχωριστά στους Μάνθο Αλιφραγκή, Αντρέα Μαντέλο, Δημήτρη Αρίστο και στα υπόλοιπα νέα ή παλιότερα μέλη του εργαστηρίου Ρομποτικής και Αυτοματισμού τους οποίους και ευχαριστώ ιδιαιτέρως για το τέλειο κλίμα συνεργασίας που έχουν υιοθετήσει στο εργαστήριο. Εύχομαι σε όλους τα αποτελέσματα των ερευνών τους να ανταμείψουν τους κόπους και τις προσδοκίες τους. Ακόμη, ευχαριστώ θερμά τους προπτυχιακούς φοιτητές Φαίδωνα Νενεδάκη και Γιώργο Δογιάμη, όπου είχα την ευκαιρία να τους καθοδηγήσω κατά τη διάρκεια της διπλωματικής τους εργασίας και αποδείχθηκαν άριστοι συνεργάτες.

Τέλος, ευχαριστώ θερμά τον πατέρα μου Μιχάλη Δεληπαράσχο και τη μητέρα μου Κατερίνα Καλλιοντζή Δεληπαράσχου για την αμέριστη αγάπη και στήριξή τους καθώς και όλους εκείνους που κατέχουν ιδιαίτερη θέση στην καρδιά μου.

Η εργασία αυτή αφιερώνεται στη μνήμη της γιαγιάς μου, Αναστασίας Καλλιοντζή. Υπήρξε για μένα ένας ιδιαίτερα σημαντικός άνθρωπος στη ζωή μου, που από τα πρώτα μου κιόλας χρόνια στο σχολείο, στήριξε και ενθάρρυνε με κάθε μέσο το ενδιαφέρον και την αγάπη που έδειχνα στην ηλεκτρονική ή όπως εκείνη συνήθιζε να λέει «τα μηχανικά».

Η σελίδα αυτή αφέθηκε σκόπιμα κενή

This page was left blank intentionally

Κεφάλαιο 1

Εισαγωγή

1.1 Αντικείμενο της Διατριβής

Η ασαφής λογική και οι γενετικοί αλγόριθμοι χρησιμοποιούνται ευρέως σε προβλήματα υπολογιστικής νοημοσύνης. Η ανάγκη ενός συστήματος υψηλής υπολογιστικής ικανότητας προκύπτει από το γεγονός ότι ο ασαφής έλεγχος είναι ιδιαίτερα απαιτητικός όσον αφορά τον αριθμό των αναγκαίων υπολογισμών. Οι υλοποιήσεις σε λογισμικό ασαφών ελεγκτών αδυνατούν να πετύχουν μεγάλη ταχύτητα επεξεργασίας κατά πρώτον λόγω της ακολουθιακής εκτέλεσης των εντολών και κατά δεύτερον λόγω ότι οι τυπικοί επεξεργαστές δεν υποστηρίζουν άμεσα ασαφείς τελεστές. Κάποιες τροποποιημένες αρχιτεκτονικές βασισμένες σε τυπικούς επεξεργαστές ούτως ώστε να υποστηρίζουν ασαφή επεξεργασία, επιτυγχάνουν μεγαλύτερη ταχύτητα επεξεργασίας σε σχέση με τους τυπικούς επεξεργαστές που όμως σε ορισμένες περιπτώσεις εφαρμογών πραγματικού χρόνου δεν είναι αρκετή. Σε τέτοιες περιπτώσεις η αποκλειστική υλοποίηση σε υλικό (dedicated hardware implementation) είναι αναγκαία.

Η παρούσα διατριβή έρχεται να επεκτείνει τη βιβλιογραφία, προτείνοντας νέες αρχιτεκτονικές στη σχεδίαση ασαφών ελεγκτών και γενετικών αλγορίθμων σε υλικό και πιο συγκεκριμένα σε ολοκληρωμένα κυκλώματα προγραμματιζόμενης λογικής (Field Programmable Gate Arrays – FPGAs chips) με τη χρήση γλωσσών περιγραφής υλικού (Hardware Description Languages – HDLs) και εργαλεία αυτοματοποίησης της σχεδίασης (Electronic Design Automation –EDA tools).

Αρχικά, παρουσιάζεται η αρχιτεκτονική σχεδίαση και υλοποίηση ενός παραμετρικού πυρήνα ασαφούς ελεγκτή, που επεξεργάζεται μόνο τους ενεργούς κανόνες και επιτυγχάνει υψηλή συχνότητα λειτουργίας και στη συνέχεια δίνεται μια τροποποιημένη εκδοχή αυτού χρησιμοποιώντας μια μέθοδο που αυξάνει την παραλληλία της σχεδίασης και επιτυγχάνει

διπλάσιο ρυθμό επεξεργασίας δεδομένων μέσω της ταυτόχρονης επεξεργασίας στην είσοδο του ελεγκτή περισσότερων από έναν ενεργών κανόνων σε κάθε κύκλο ρολογιού.

Στη συνέχεια, μία ρομποτική πλατφόρμα παρακολούθησης πορείας με τη χρήση ασαφούς λογικής για αυτόνομα κινητά ρομπότ με αυξημένη δυνατότητα επεξεργασίας και ευέλικτο υλικό για διαφορετικές διεργασίες, αναπτύσσεται με την ενσωμάτωση του πυρήνα ασαφούς ελεγκτή και ενός μαλακού πυρήνα μικροεπεξεργαστή σε σύστημα σε ψηφίδα (System on a Chip – SoC). Το SoC προσαρμόστηκε πάνω σε ένα ρομπότ Pioneer P3-DX8 και στη συνέχεια εκτελέστηκαν διάφορα πειράματα, ούτως ώστε να γίνει αποτίμηση της γενικής απόδοσης του συστήματος.

Το πιο σοβαρό μειονέκτημα των γενετικών αλγορίθμων που έχουν αναπτυχθεί σε λογισμικό είναι οι απαιτήσεις τους σε μεγάλα ποσά μνήμης και σε χρόνο εκτέλεσης. Η σύγκλισή τους προς το βέλτιστο μπορεί να είναι υπερβολικά αργή για δύσκολα και περίπλοκα προβλήματα βελτιστοποίησης, με αποτέλεσμα να γίνεται δύσκολη η χρήση τους σε εφαρμογές πραγματικού χρόνου. Δεδομένων αυτών των περιορισμών, έχει υπάρξει μια προσπάθεια υλοποίησης μιας πληθώρας γενετικών αλγορίθμων σε υλικό. Αυτό έχει επιτευχθεί κατά κύριο λόγω της ταχείας ανάπτυξης που παρατηρείται στην τεχνολογία των ολοκληρωμένων κυκλωμάτων προγραμματιζόμενης λογικής τα τελευταία χρόνια. Ο πυρήνας του γενετικού αλγορίθμου που αναπτύχθηκε στην παρούσα εργασία επιτυγχάνει μεγάλη συχνότητα λειτουργίας και εκμεταλλεύεται την παραλληλία που προσφέρει η σχεδίαση σε υλικό δίνοντας του τη δυνατότητα να χρησιμοποιηθεί σε εφαρμογές πραγματικού χρόνου.

1.2 Οργάνωση της Διατριβής

Στο παρόν κεφάλαιο, συνοψίζονται το αντικείμενο της διατριβής, η συμβολή της διατριβής και οι δημοσιεύσεις που προέκυψαν από την έρευνα που προηγήθηκε της εργασίας αυτής.

Το Κεφάλαιο 2, αναλύει θεμελιώδεις βασικές γνώσεις της θεωρίας της ασαφούς λογικής (*fuzzy logic*) και εισάγει τους σχετικούς συμβολισμούς και ορισμούς που είναι απαραίτητοι για την παρουσίαση των προτεινόμενων μεθόδων που ακολουθούν στα επόμενα κεφάλαια. Πιο αναλυτικά, γίνεται μια εκτενής αναφορά στη θεωρία των ασαφών συνόλων και επεξηγούνται οι μέθοδοι εξαγωγής συμπερασμάτων κατά Mamdani και κατά Takagi-Sugeno. Επιπλέον, αναλύεται η γενική αρχιτεκτονική των ασαφών συστημάτων και παραθέτονται παραδείγματα εφαρμογών της ασαφούς λογικής.

Το Κεφάλαιο 3, παρουσιάζει τα βασικότερα σημεία της θεωρίας των γενετικών αλγορίθμων (*genetic algorithms*). Αρχικά περιγράφονται οι βιολογικές έννοιες που είναι απαραίτητες για την κατανόηση των γενετικών αλγορίθμων και εν συνεχεία, αναφέρεται ο βασικός γενετικός αλγόριθμος και οι διάφορες μέθοδοι γενετικών πράξεων που χρησιμοποιούνται. Τέλος δίνονται ορισμένα παραδείγματα εφαρμογών των αλγορίθμων αυτών.

Το Κεφάλαιο 4, αποτελεί μία εισαγωγή στην αυτοματοποίηση της ηλεκτρονικής σχεδίασης (*Electronic Design Automation* – EDA). Έννοιες και τεχνικές που αναφέρονται εκτενώς στα κεφάλαια που ακολουθούν, όπως η διαδικασία *προσομοίωσης* της περιγραφής του κυκλώματος σε *επίπεδο καταχωρητών (Register Transfer Level* – RTL*)*, η *λογική σύνθεση*

(logic synthesis) και η *θέση και δρομολόγηση (place and route)* του ψηφιακού κυκλώματος στην ψηφίδα, επεξηγούνται αναλυτικά.

Το Κεφάλαιο 5, κάνει μια αναφορά στη συνεχώς αναπτυσσόμενη τεχνολογία των ολοκληρωμένων κυκλωμάτων προγραμματιζόμενης λογικής (*Field Programmable Gate Arrays* – FPGA) και παρουσιάζονται τα βασικότερα χαρακτηριστικά τους και διάφορα πεδία εφαρμογών τους. Ιδιαίτερη αναφορά γίνεται στην οικογένεια ολοκληρωμένων κυκλωμάτων προγραμματιζόμενης λογικής Spartan–3 της εταιρίας Xilinx που χρησιμοποιήθηκαν στην παρούσα διατριβή.

Το Κεφάλαιο 6, περιγράφει τη σχεδίαση και υλοποίηση ενός παραμετρικού πυρήνα ψηφιακού ασαφούς ελεγκτή. Ο πυρήνας είναι διαμορφώσιμος ως προς τον αριθμό των εισόδων, εξόδων, συναρτήσεων συμμετοχής και τη μέθοδο συνάθροισης και συνεπαγωγής επιτρέποντας έτσι την προσαρμογή του σε εφαρμογές με διαφορετικές απαιτήσεις χωρίς την ανάγκη επανασχεδίασης του. Η αρχιτεκτονική του ασαφούς ελεγκτή έχει τη δυνατότητα να επεξεργάζεται μόνο τους ενεργούς κανόνες, δηλαδή του κανόνες εκείνους που έχουν μη μηδενική συμβολή στο τελικό αποτέλεσμα, αυξάνοντας σημαντικά με αυτό τον τρόπο το ρυθμό επεξεργασίας δεδομένων του πυρήνα. Η υλοποίηση του διαμορφωμένου για 4 εισόδους των 12-bit και 2401 κανόνες πυρήνα στο ολοκληρωμένο κύκλωμα προγραμματιζόμενης λογικής, επιτυγχάνει υψηλή συχνότητα λειτουργίας που ανέρχεται στα 100 MHz.

Το Κεφάλαιο 7, συμπληρώνει το προηγούμενο κεφάλαιο και παρουσιάζει μια τεχνική αύξησης του ρυθμού επεξεργασίας των δεδομένων στην αρχιτεκτονική του πυρήνα του ασαφούς ελεγκτή, που αναφέρεται ως τεχνική ODD-EVEN. Με την προαναφερθείσα τεχνική επιτυγχάνεται ρυθμός επεξεργασίας δεδομένων $2^n / 2$ για δύο εισόδους (σε σχέση με 2^n, της αρχιτεκτονικής του Κεφαλαίου 6), με τον έλεγχο δύο ενεργών κανόνων σε κάθε κύκλο ρολογιού. Η μέθοδος ονομάστηκε ODD-EVEN διότι στηρίζεται στον ταυτόχρονο έλεγχο των κανόνων που αναφέρονται στο περιττό ενεργό ασαφές σύνολο της πρώτης εισόδου με αυτούς που αναφέρονται στο άρτιο ενεργό σύνολο της πρώτης εισόδου. Η εξέταση δύο κανόνων ανά κύκλο ρολογιού επιτυγχάνεται διαχωρίζοντας τη μνήμη των κανόνων και τη μνήμη των παραμέτρων των ασαφών συνόλων της πρώτης εισόδου σε ODD και EVEN μέρη. Ο διαχωρισμός αυτός αυξάνει την παραλληλία τα αρχιτεκτονικής σχεδίασης με μικρή αύξηση της λογικής.

Το Κεφάλαιο 8, αποτελεί την εφαρμογή του πυρήνα του ασαφούς ελεγκτή στο έλεγχο ενός κινητού ρομπότ τύπου Pioneer P3-DX8. Ο πυρήνας του ασαφούς ελεγκτή που αναλύθηκε στο Κεφάλαιο 6 ολοκληρώθηκε με έναν πυρήνα μαλακού επεξεργαστή τύπου Microblaze για να αποτελέσει έναν αυτόνομο σύστημα σε ψηφίδα (System on a Chip – SoC) για το πρόβλημα της παρακολούθησης πορείας (path tracking) σε κινητά ρομπότ. Η παραμετρικότητα της σχεδίασης του ασαφούς ελεγκτή επέτρεψε την εύκολη προσαρμογή του πυρήνα (αλλαγή μόνο των παραμέτρων) στα χαρακτηριστικά του ασαφούς ελεγκτή για το πρόβλημα της παρακολούθησης πορείας, χωρίς την ανάγκη επανασχεδίασης του πυρήνα. Η ακρίβεια της ακολουθούμενης πορείας στα πειράματα που διεξήχθησαν επιβεβαιώθηκαν με την παράλληλη χρήση διαφορικού GPS για τη μέτρηση της πραγματικής πορείας του ρομπότ.

Το Κεφάλαιο 9, περιγράφει την αρχιτεκτονική σχεδίαση και υλοποίηση ενός παραμετρικού πυρήνα γενετικού αλγορίθμου σε κύκλωμα προγραμματιζόμενης λογικής FPGA. Η

παραμετρικότητα του πυρήνα αφορά στις διάφορες παραμέτρους του γενετικού αλγορίθμου, όπως για παράδειγμα, το μέγεθος του πληθυσμού, το μέγιστο όριο γενεών εκτέλεσης, το μέγιστο όριο τιμής προσαρμογής, κ.τ.λ. Η συγκεκριμένη υλοποίηση γενετικού αλγορίθμου επιτυγχάνει αρκετά υψηλή συχνότητα λειτουργίας και παρουσιάζει αξιοσημείωτη επιτάχυνση όταν συγκρίνεται με την αντίστοιχη υλοποίησή του σε λογισμικό. Η παρούσα σχεδίασή αξιολογήθηκε μέσω της χρήσης συναρτήσεων σύγκρισης και με την εφαρμογή του πυρήνα στην επίλυση του προβλήματος του Πλανόδιου Πωλητή για διαφορετικό αριθμό πόλεων.

Το Κεφάλαιο 10, αποτελεί τη σύνοψη και ανακεφαλαίωση της παρούσας διατριβής. Επιπλέον, προτείνονται μελλοντικές επεκτάσεις αυτής και πιθανές εφαρμογές της.

1.3 Συμβολή και Πρωτοτυπία της Διατριβής

Οι καινοτομίες της παρούσας διατριβής εντοπίζονται στα παρακάτω,

- Η αρχιτεκτονική σχεδίαση και υλοποίηση ενός παραμετρικού (διαμορφώσιμου) πυρήνα ψηφιακού ελεγκτή ασαφούς λογικής (DFLC) Takagi-Sugeno μηδενικής τάξης, που επεξεργάζεται μόνο τους ενεργούς κανόνες για ένα σύνολο δεδομένων στις εισόδους. Ο πυρήνας έχει τη δυνατότητα διαμόρφωσης των παραμέτρων του ασαφούς ελεγκτή χωρίς την ανάγκη επανασχεδίασης του και επιπλέον επιτυγχάνει σημαντική αύξηση του ρυθμού επεξεργασίας των δεδομένων εισόδου του συστήματος. Ο πυρήνας του ελεγκτή είναι παραμετροποιήσιμος όσον αφορά τον αριθμό των εισόδων και εξόδων (και μέγεθος διαύλου), του συστήματος, των αριθμό συναρτήσεων συμμετοχής (τριγωνικές ή τραπεζοειδείς) εισόδου και εξόδου (μονότιμα σύνολα), τη μέθοδο συνάθροισης και συνεπαγωγής, τον τύπο μοντέλου του διαιρέτη, και τον αριθμό καταχωρητών αγωγού των ανεξάρτητων μπλοκ στην ιεραρχία της σχεδίασης.
- Βάσει της προηγούμενης σχεδίασης παρουσιάζεται μια πιο αποτελεσματική μέθοδος η οποία επιτρέπει μεγαλύτερο ρυθμό επεξεργασίας δεδομένων, με τον έλεγχο δύο ενεργών κανόνων, αντί για ένα, σε κάθε κύκλο ρολογιού. Η μέθοδος στηρίζεται στον ταυτόχρονο έλεγχο των κανόνων που αναφέρονται στο περιττό ενεργό ασαφές σύνολο της πρώτης εισόδου, με αυτούς που αναφέρονται στο άρτιο ενεργό ασαφές σύνολο της πρώτης εισόδου. Η εξέταση δύο κανόνων ανά κύκλο επιτυγχάνεται διαχωρίζοντας τη μνήμη των κανόνων και τη μνήμη των παραμέτρων των ασαφών συνόλων της πρώτης εισόδου σε περιττά και άρτια μέρη. Η συγκεκριμένη κωδικοποίηση των μνημών αυξάνει την παραλληλία του πυρήνα αυξάνοντας έτσι το ρυθμό επεξεργασίας δεδομένων του ελεγκτή. Ως προς το χειρισμό των συναρτήσεων συμμετοχής, παρουσιάστηκαν δύο εναλλακτικές υλοποιήσεις. Μία με αριθμητικό υπολογισμό των συναρτήσεων συμμετοχής και μία άλλη στην οποία οι τιμές αυτών ήταν προϋπολογισμένες και αποθηκεύτηκαν σε μία ROM.
- Η υλοποίηση ενός συστήματος σε ψηφίδα (System on Chip – SoC) για το πρόβλημα της παρακολούθησης πορείας σε αυτόνομα μη-ολονομικά κινητά ρομπότ. Το SoC περιλαμβάνει έναν ψηφιακό ασαφή ελεγκτή, και ένα πρόγραμμα ελέγχου της ροής των διαδικασιών, που εκτελείται μέσα στον μαλακό πυρήνα μικροεπεξεργαστή Microblaze της εταιρίας Xilinx. Ο ασαφής ελεγκτής υλοποιεί έναν ασαφή αλγόριθμο παρακολούθησης πορείας (*path tracking*). Ολόκληρο το

σύστημα προσαρμόστηκε πάνω σε ένα ρομπότ Pioneer P3-DX8 και στη συνέχεια εκτελέστηκαν διάφορα πειράματα, ούτως ώστε να γίνει αποτίμηση της γενικής απόδοσης του συστήματος.

- Η σχεδίαση και υλοποίηση ενός πλήρως παραμετρικού πυρήνα γενετικού αλγορίθμου σε ολοκληρωμένο κύκλωμα προγραμματίζομενης λογικής FPGA. Η παραμετρικότητα του πυρήνα αφορά στις διάφορες παραμέτρους του γενετικού αλγορίθμου. Επιπλέον, η παραμετρική σχεδίαση επιτρέπει την προσαρμογή του ΓΑ σε διαφορετικές προδιαγραφές χωρίς την ανάγκη αλλαγών στην ψηφιακή σχεδίαση του πυρήνα. Ο πυρήνας του γενετικού αλγορίθμου λειτουργεί σε σημαντικά υψηλή συχνότητα ρολογιού και επιτυγχάνει μία αξιοσημείωτη επιτάχυνση όταν συγκρίνεται με την αντίστοιχη υλοποίησή του σε λογισμικό. Επιπρόσθετα, η υλοποίηση καταλαμβάνει λίγους πόρους στο FPGA. Συγκρινόμενο με άλλες υλοποιήσεις γενετικών αλγορίθμων σε υλικό που αναφέρονται στη βιβλιογραφία, η συγκεκριμένη υλοποίηση λειτουργεί σε υψηλότερη συχνότητα ρολογιού και υλοποιεί περισσότερες της μίας μεθόδου διασταύρωσης και μετάλλαξης, οι οποίες μπορούν να αλλάξουν σε πραγματικό χρόνο κατά την εκτέλεση του αλγορίθμου. Τέλος, η παρούσα σχεδίασή κάνει χρήση περισσότερων παραμέτρων και αξιολογείται όχι μόνο μέσω της χρήσης συναρτήσεων σύγκρισης αλλά και με τη χρήση του προβλήματος του Πλανόδιου Πωλητή.

1.4 Δημοσιεύσεις

Από την έρευνα που διεξήχθη στα πλαίσια της παρούσας διατριβής προέκυψαν οι παρακάτω δημοσιεύσεις σε επιστημονικά περιοδικά, διεθνή και πανελλήνια συνέδρια, και κεφάλαια βιβλίων.

1.4.1 Εργασίες σε Επιστημονικά Περιοδικά.

- S.G. Tzafestas, **K.M. Deliparaschos**, and G.P. Moustris, “Fuzzy logic path tracking control for autonomous non-holonomic mobile robots: Design of system on a chip,” *Journal of Robotics and Autonomous Systems.* DOI: 10.1016/j.robot.2010.03.014
- **K.M. Deliparaschos**, G.C. Doyamis, and S.G. Tzafestas, “A parameterised genetic algorithm IP - core: FPGA - design, implementation and performance evaluation,” *Int. Journal of Electronics*, vol. 95, 2008, p. 1149.
- **K.M. Deliparaschos** and S.G. Tzafestas, “A parameterized T-S digital fuzzy logic processor: Soft core VLSI design and FPGA implementation,” *Int. Journal of Factory Automation, Robotics and Soft Computing*, vol. 3, Jul. 2006, pp. 7-15.
- **K.M. Deliparaschos**, F.I. Nenedakis, and S.G. Tzafestas, “Design and implementation of a fast digital fuzzy logic controller using FPGA technology,” *Journal of Intelligent and Robotic Systems*, vol. 45, Jan. 2006, pp. 77-96.

1.4.2 Εργασίες σε Διεθνή και Πανελλήνια Συνέδρια.

- G. Moustris and **K.M. Deliparaschos**, “Tracking control using the strip-wise affine transformation: An experimental SoC design,” *European Control Conf. 2009 (ECC'09)*, Budapest, Hungary: 2009.

- **K.M. Deliparaschos** and S.G. Tzafestas, "Design paradigms of intelligent control systems on a chip," *Panhellenic Conf. on Electronics & Telecommunications (PACET'09)*, Patras, Greece: 2009.
- **K.M. Deliparaschos**, G.P. Moustris, and S.G. Tzafestas, "Autonomous SoC for fuzzy robot path tracking," *European Control Conf. 2007 (ECC'07)*, Kos, Greece: 2007
- **K.M. Deliparaschos**, G.C. Doyamis, and S.G. Tzafestas, "A parameterized genetic algorithm IP core design and implementation," *in Proc. of the 4th Int. Conf. on Informatics in Control, Automation and Robotics (ICINCO '07)*, Angers, France: 2007, pp. 417-423.
- **K.M. Deliparaschos**, F.I. Nenedakis, and S.G. Tzafestas, "A fast digital fuzzy logic controller: FPGA design and implementation," *in Proc. of 10th IEEE Conf. on Emerging Technologies and Factory Automation., 2005 (IEEE ETFA '05)*, Catania, Sicily: 2005, pp. 259-262.

1.4.3 Εργασία σε Κεφάλαιο Βιβλίου.

- **K. M. Deliparaschos, S. G. Tzafestas, "**A Parameterized T-S digital fuzzy logic processor: Soft core VLSI design and FPGA implementation", *Recent advances in Control Systems, Robotics and Automation*, ISBN: 88-901928-0-1.

1.4.4 Γνωστές αναφορές από τρίτους

- A. Razib, "Design and FPGA implementation of a log-domain high-speed fuzzy control system," MSc Thesis, University of Alberta, 2010.
- A. Razib, S. Dick, and V. Gaudet, "Design of a high-speed fuzzy logic controller based on log-domain arithmetic," *IEEE Int. Symp. on Multiple-Valued Logic*, Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 139-144.
- A. Téllez, L. Villa, H. Molina, O. Camacho, and R. Urbieta, "A practical approach for combinatorial fuzzy logic control design," *ICINCO 2009 - 6th International Conference on Informatics in Control, Automation and Robotics*, Proceedings, 2009, pp. 343-346.
- Antonio Z. Hernández, "High-performance architecture for fuzzy processors," PhD Thesis, National Polytechnic Institute, Computing Research Center, 2009.
- J.T. Alander, *Indexed bibliography of genetic algorithms in operations research*, Vaasa, Finland: University of Vaasa, Department of Electrical Engineering and Automation, 2009.
- N. Kanagaraj, P. Sivashanmugam, and S. Paramasivam, "A fuzzy logic based supervisory hierarchical control scheme for real time pressure control," *Int. Journal of Automation and Computing*, vol. 6, 2009, pp. 88-96.
- Q. Cao, Liqin Song, X. Shi, and J.H. Li, "Multi-tasking of fuzzy inference processor through real-time context switching," *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Singapore: 2009, pp. 1429-1434.
- S. Levente Lörinc, "Systematic retrofit method for chemical batch processes using indicators, heuristics and process models," PhD Thesis, ETH Zurich, 2007.

Κεφάλαιο 2

Ασαφής Λογική

2.1 Εισαγωγή

Στο κεφάλαιο αυτό δίνεται μία σύντομη εισαγωγή στην περιοχή της ασαφούς λογικής. Η ασαφής λογική είναι ένα ευρύ επιστημονικό πεδίο που δημιουργήθηκε από την ανάγκη για παράκαμψη της αυστηρής παραδοσιακής δυαδικής λογικής, στην οποία υπάρχουν μόνο οι καταστάσεις του αληθούς ή του ψευδούς. Μέσω της χρήσης ασαφούς λογικής είναι δυνατός ο ορισμός βαθμών αληθείας, οι οποίοι μετρούν το κατά πόσο κάποιο στοιχείο συμμετέχει σε ένα ασαφές σύνολο. Οι συναρτήσεις συμμετοχής ορίζουν την κατανομή των τιμών του βαθμού αληθείας (κατά πόσο συμμετέχει ένα στοιχείο σε ένα ασαφές σύνολο), που μπορεί να πάρει κάποιο στοιχείο του συνόλου, όπου το πεδίο τιμών τους είναι το [0,1]. Στην ασαφή λογική συνήθως τα ασαφή σύνολα χρησιμοποιούν λεκτικές μεταβλητές που υπάρχουν και στην ανθρώπινη γλώσσα. Κατά αυτόν τον τρόπο επιτυγχάνεται η απευθείας κωδικοποίηση έμπειρης γνώσης σε διάφορες εφαρμογές.

Ο Zadeh (1965) επέκτεινε την κλασική λογική, από το διακριτό σύνολο {0,1} στο συνεχές διάστημα [0,1], εισάγοντας μία ομαλή μετάβαση από το «λάθος» στο «σωστό». Ένα καθορισμένο ή αυστηρό σύνολο (*crisp set*) περιγράφεται πλήρως από τα στοιχεία του με βάση τη διακριτή σχέση, «ανήκει», 1 ή δεν «ανήκει», 0. Επιπλέον ο Zadeh παρατήρησε ότι πολλά σύνολα δεν είναι σαφώς καθορισμένα και πρότεινε η συνάρτηση συμμέτοχης, μ να βρίσκεται στο διάστημα [0,1], ομαλοποιώντας την απόφαση για το εάν ένα στοιχείο, x ανήκει ή όχι σε ένα σύνολο A.

Σκοπός του κεφαλαίου είναι να παρουσιάσει πρώτα τις βασικές έννοιες τις θεωρίας ασαφών συνόλων (βλ. §2.2) που είναι απαραίτητες για την κατανόηση της ασαφούς λογικής. Στη συνέχεια αναφέρονται οι δύο κυριότεροι μέθοδοι ασαφούς συλλογιστικής, δηλαδή η μέθοδος του Mamdani και η μέθοδος των Takagi-Sugeno (βλ. §2.3 και §2.4).

Ακολουθεί μια σύντομη παρουσίαση της αρχιτεκτονικής των ασαφών συστημάτων και επιπλέον γίνεται μια αναφορά στις σημαντικότερες μεθόδους από-ασαφοποιήσης (βλ. §2.5). Τέλος στην παράγραφο §2.6 αναφέρονται τα βασικότερα πεδία εφαρμογής της ασαφούς λογικής μέσα από πραγματικές εφαρμογές.

2.2 Θεωρία Ασαφών Συνόλων

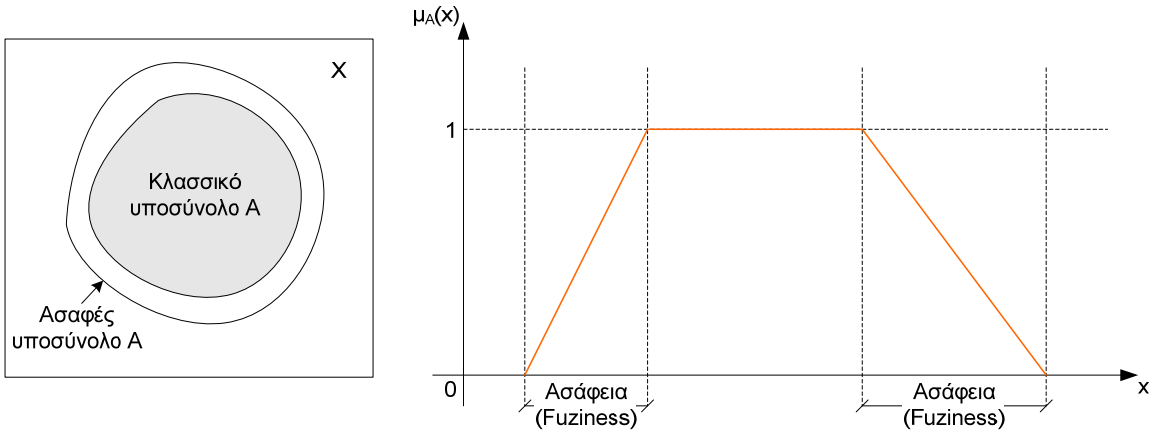
Έστω X υπερσύνολο αναφοράς (*Universe of Discourse*). Τότε ασαφές σύνολο A του X καλείται το διατεταγμένο ζεύγος,

$$A = \{(x, \mu_A(x)) \mid x \in X, \mu_A(x): X \rightarrow [0,1]\} \tag{2.1}$$

Λέμε τότε ότι τα στοιχεία, x του ασαφούς συνόλου ανήκουν στο υπερσύνολο αναφοράς X, το οποίο είναι το διάστημα τιμών μιας εισόδου του ασαφούς συστήματος.

Η συνάρτηση $\mu_A(x)$ καλείται συνάρτηση συμμετοχής και προσδιορίζει το βαθμό, στον οποίο ένα στοιχείο x του υπερσυνόλου αναφοράς X ανήκει στο ασαφές σύνολο A, παίρνοντας τιμές στο διάστημα [0,1].

Η γραφική απεικόνιση ενός ασαφούς συνόλου A με συνάρτηση συμμετοχής $\mu_A(x)$, φαίνεται στο Σχήμα 2.1 (α) και (β).



Σχήμα 2.1: (α) Ασαφές σύνολο A και η αντίστοιχη (β) συνάρτηση συμμετοχής του

Αν το A είναι διακριτό, τότε συμβολίζεται με,

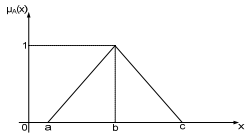
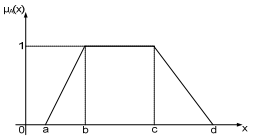
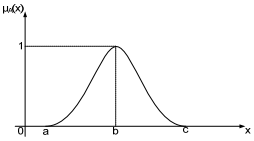
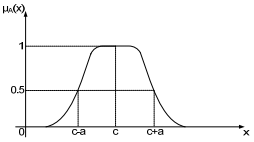
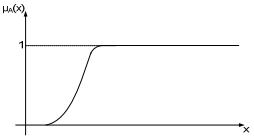
$$A = \mu_A(x_1)/x_1 + \ldots + \mu_A(x_n)/x_n = \sum_{i=1}^{n} \mu_A(x_i)/x_i \tag{2.2}$$

Η διαφορετικά αν είναι συνεχές συμβολίζεται με,

$$A = \int_x \mu_A(x)/x \tag{2.3}$$

όπου τα σύμβολα +, $\sum$, $\int$ εκφράζουν την 'ένωση' όλων των στοιχείων του συνόλου, ενώ το σύμβολο / δεν δηλώνει διαίρεση.

Μερικές από τις πιο γνωστές συναρτήσεις συμμετοχής δίνονται στον Πίνακα 2.2.

Συνάρτηση	**Τύπος**	**Γραφική απεικόνιση**
Τριγωνική	$\mu_A(x) = \max\left[\min\left\{(x-a)/(b-a);(c-x)/(c-b)\right\};0\right]$	
Τραπεζοειδής	$\mu_A(x) = \max\left[\min\left\{(x-a)/(b-a);1;(d-x)/(d-c)\right\};0\right]$	
Γκαουσιανή	$\mu_A(x) = e^{-((x-c/\sigma))^2}$	
Καμπανοειδής (*Bell*)	$\mu_A(x) = \left(1+\left\lvert(x-c)/a\right\rvert^{2b}\right)^{-1}$	
Σιγμοειδής	$\mu_A(x) = \left(1+\exp\left(-a\left(x-c\right)\right)\right)^{-1}$	

Πίνακας 2.1: Παραμετρικοί τύποι συνεχών συναρτήσεων συμμετοχής

Ένα ζευγάρι, x, $\mu_A(x)$ καλείται ασαφές σύνολο συγκεκριμένης τιμής ή μονότιμο σύνολο (*fuzzy singleton*) και όλο το ασαφές σύνολο αποτελείται από την ένωση των συνιστάμενων ασαφών συνόλων συγκεκριμένης τιμής, οπότε μπορούμε να θεωρήσουμε το ασαφές σύνολο A σαν ένα διάνυσμα,

$$A = (\mu_A(x_1),\ \mu_A(x_2),\ldots,\mu_A(x_n)) \tag{2.4}$$

Εάν μια είσοδος x έχει συνάρτηση συμμετοχής κοντά στο 1 ή $\mu_A(x)\rightarrow 1$, τότε η είσοδος x ανήκει στο A σε μεγάλο βαθμό και αντίστροφα.

Ένα ασαφές σύνολο A καλείται κανονικό (*normal*) αν η μέγιστη τιμή της συνάρτησης συμμετοχής του είναι ίση με τη μονάδα,

$$\max_{x\in X} \mu_A(x) = 1 \tag{2.5}$$

Αν το X είναι πεπερασμένο σύνολο, τότε ο αριθμός στοιχείων (*cardinality*) του ασαφούς συνόλου A στο X ορίζεται ως εξής:

$$|A| = \sum_{x \in X} \mu_A(x) \tag{2.6}$$

και ο σχετικός αριθμός στοιχείων (relative cardinality), δίνεται από:

$$\|A\| = \frac{|A|}{|X|} \tag{2.7}$$

2.2.1 Ιδιότητες Ασαφών Συνόλων

Προκειμένου να καλυφθούν οι ανάγκες σύγκρισης και χαρακτηρισμού των ασαφών συνόλων, χρησιμοποιούνται κάποιες χαρακτηριστικές έννοιες.

Το σύνολο όλων των στοιχείων του υπερσυνόλου αναφοράς X τα οποία έχουν θετική συνάρτηση συμμετοχής ορίζεται ως στήριγμα (*support* ή *supp*) του ασαφούς συνόλου A ($A \subset X$ και $supp(A) \subset X$),

$$supp(A) \ = \ \{ x \in X : \ \mu_A(x) > 0\} \tag{2.8}$$

Ένα ασαφές σύνολο συγκεκριμένης τιμής είναι ένα ασαφές σύνολο με στήριγμα ένα στοιχείο του X με συνάρτηση συμμετοχής μονάδα, $\mu_A(x)$=1.

Μια άλλη χαρακτηριστική έννοια είναι η μέγιστη τιμή (*supremum* ή *sup*) από τις μέγιστες τιμές των συναρτήσεων συμμετοχής, που αλλιώς ονομάζεται και ύψος (*height*),

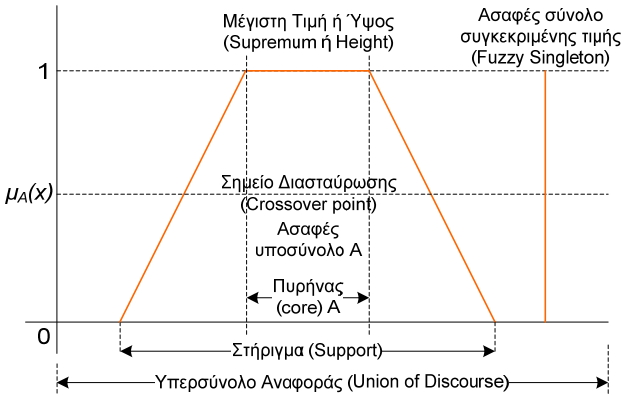
$$htg(A) = \sup_{x \in X} \mu_A(x) \tag{2.9}$$

Η περιοχή στην οποία οι τιμές του $x \in X$ έχουν τη μέγιστη τιμή συμμετοχής στο ασαφές σύνολο A, ονομάζεται πυρήνας (*core*) του A,

$$core(A) = \{ x \in X, \mu_A(x) = 1 \} \tag{2.10}$$

Η χαρακτηριστική αυτή έννοια μπορεί να είναι είτε μια διακριτή τιμή, είτε ένα εύρος τιμών ανάλογα με το σχήμα του ασαφούς συνόλου. Ο λόγος του εύρους του πυρήνα προς το εύρος των τιμών στήριξης είναι ένας τρόπος μέτρησης που χαρακτηρίζει τη μοναδικότητα του ασαφούς συνόλου. Όσο ο λόγος αυτός πλησιάζει τη μονάδα, τόσο το σύνολο γίνεται λιγότερο ασαφές ή τείνει να γίνει αυστηρό, ενώ αντίθετα όσο ο λόγος πλησιάζει το μηδέν το σύνολο τείνει να γίνει μοναδικό όσον αφορά το στήριγμα του.

Οι προαναφερθείσες χαρακτηριστικές έννοιες επεξηγούνται γραφικά στο Σχήμα 2.2.



Σχήμα 2.2: Ορολογία ασαφούς συνόλου

Για παράδειγμα ένα ασαφές σύνολο με τριγωνική συνάρτηση συμμετοχής, πυρήνα 10 και στήριγμα από 9 έως 11 μπορεί να αποδοθεί ως ο ασαφής αριθμός 10 με εύρος ±1. Τα ασαφή σύνολα με μέγιστη τιμή ή ύψος ίσο με τη μονάδα (*htg(A)*=1) περιγράφονται ως κανονικά ή κανονικοποιημένα (*normal* ή *normalized*), ενώ αντίθετα αυτά των οποίων η μέγιστη τιμή δεν φτάνει τη μονάδα, ονομάζονται υποκανονικά (*subnormal*).

Για να κανονικοποιήσουμε ένα σύνολο *A*, κανονικοποιούμε τη συνάρτηση συμμετοχής $\mu_A(x)$ διαιρώντας με το ύψος, ως εξής:

$$\mu_{\mu\acute{\varepsilon}\tau\rho o(A)}(x) = \frac{\mu_A(x)}{\max \mu_A(x)} \tag{2.11}$$

Ο τρόπος αυτός (βλ. Εξ. 2.11) συμπεριλαμβάνεται στους γλωσσικούς διαμορφωτές ή τροποποιητές [1]. Γλωσσικός διαμορφωτής είναι ένας τελεστής, που όταν εφαρμοστεί σε ένα ασαφές σύνολο *A*, μιας γλωσσικής μεταβλητής μεταβάλλει το νόημα της. Στον Πίνακα 2.2 παρουσιάζονται τέτοιοι γλωσσικοί διαμορφωτές.

Κανονικοποιητής	$\mu_{\mu\acute{\varepsilon}\tau\rho o(A)}(x) = \frac{\mu_A(x)}{\max \mu_A(x)}$
Συγκεντρωτής	$\mu_{\pi o\lambda\acute{\upsilon}(A)}(x) = [\mu_A(x)]^2$
Εξαπλωτής	$\mu_{\sigma\chi\varepsilon\delta\acute{o}\nu(A)}(x) = [\mu_A(x)]^{1/2}$
Συν	$\mu_{\sigma\chi\varepsilon\delta\acute{o}\nu(A)}(x) = [\mu_A(x)]^{5/4}$
Μείον	$\mu_{\sigma\chi\varepsilon\delta\acute{o}\nu(A)}(x) = [\mu_A(x)]^{5/4}$
Άρνηση	$\mu_{\acute{o}\chi\iota(A)}(x) = 1 - \mu_A(x)$

Πίνακας 2.2: Τυπικοί γλωσσικοί διαμορφωτές

Επιπλέον, ορίζονται ως *α-τομές* (*α-levels* ή *α-cuts*) ενός ασαφούς συνόλου *A* ως το αυστηρό σύνολο $L_\alpha A$ που περιλαμβάνει όλα τα στοιχεία του υπερσυνόλου αναφοράς *X* και συμμετέχει στο *A* με βαθμό μεγαλύτερο ή ίσο του *α*, όπου το *α* ανήκει στο [0,1],

$$L_\alpha A = \{x \in X \mid \mu_A(x) \geq \alpha\},\ \alpha \in [0,1] \tag{2.12}$$

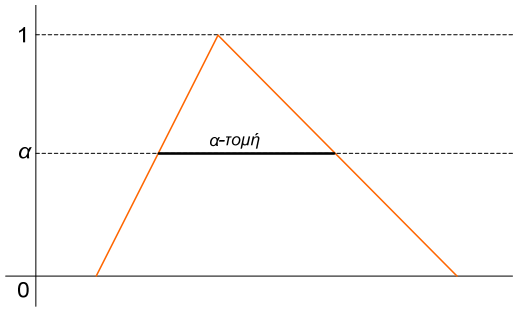
αντίστοιχα ορίζονται ως δυνατές α-τομές (*strong α-levels* ή *strong α-cuts*) το σύνολο,

$$L_\alpha A = \{x \in X \mid \mu_A(x) > \alpha\},\ \alpha \in [0,1] \tag{2.13}$$

Προφανώς, οι α-τομές με α=1 ή A_1 ονομάζονται πυρήνας (*core*) και συνεπώς πρόκειται για το σύνολο,

$$L_\alpha A = \{x \in X \mid \mu_A(x) = 1\} \tag{2.14}$$

Στο Σχήμα 2.3 αποδίδεται γραφικά η α-τομή μιας τριγωνικού-τύπου συνάρτησης συμμετοχής.



Σχήμα 2.3: α-τομή Τριγωνικής συνάρτησης συμμετοχής

2.2.2 Πράξεις Ασαφών Συνόλων

Το ασαφές σύνολο A ονομάζεται κενό αν η συνάρτηση συμμετοχής του είναι μηδενική παντού, η αλλιώς,

$$A = \varnothing \Leftrightarrow \mu_A(x) = 0,\ \forall x \in A \tag{2.15}$$

Για τα ασαφή σύνολα A,B που ανήκουν σε ένα υπερσύνολο αναφοράς X θα ισχύει:

$$\begin{aligned} A &= \{(x, \mu_A(x))\},\ \mu_A(x) \in [0,1] \\ B &= \{(x, \mu_B(x))\},\ \mu_B(x) \in [0,1] \end{aligned} \tag{2.16}$$

Οι πράξεις μεταξύ των A,B ορίζονται ως πράξεις μεταξύ των συναρτήσεων συμμετοχής των, $\mu_A(x),\mu_B(x)$. Εάν $A, B \subseteq X$, τότε θα ισχύει:

- Ισότητα, $A = B$ αν και μόνο αν $\mu_A(x) = \mu_B(x),\ \forall x \in X$
- Υποσύνολο, $A \subseteq B$ αν και μόνο αν $\mu_A(x) \leq \mu_B(x),\ \forall x \in X$

Τα ασαφή σύνολα όπως και τα κλασσικά σύνολα διατηρούν τις τρεις θεμελιώδεις πράξεις της τομής (*intersection*), της ένωσης (*union*) και του συμπληρώματος (*complement*). Οι τρεις θεμελιώδεις πράξεις των ασαφών συνόλων προτάθηκαν από τον Zadeh.

Έστω δύο ασαφή σύνολα A και B, στον Πίνακα 2.3 που ακολουθεί ορίζονται οι προαναφερθέντες πράξεις.

Η τομή τους	$A \cap B = \{x \in X : (x \in A) \wedge (x \in B)\}$ $\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x)),\ x \in X$
Η ένωση τους	$A \cup B = \{x \in X : (x \in A) \vee (x \in B)\}$ $\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x)),\ x \in X$
Το συμπληρωματικό του Α	$\overline{A} = \{x \in U : \neg(x \in A)\}$ $\mu_{\overline{A}}(x) = 1 - \mu_A(x)$

Πίνακας 2.3: Θεμελιώδεις πράξεις ασαφών συνόλων

2.2.2.1 Μέτρα t και s (t-norms και s-norms)

Μια δυαδική πράξη $\boldsymbol{t} : [0,1]\times[0,1] \to [0,1]$ ονομάζεται *triangular norm* (*μέτρο t* ή *t-norm* εν συντομία) εάν για όλα τα $\alpha, \beta, \gamma \in [0,1]$ τηρούνται οι παρακάτω ιδιότητες,

1. Μεταθετικότητα (*Commutativity*): $\alpha \boldsymbol{t} \beta = \beta \boldsymbol{t} \alpha$
2. Προσεταιριστικότητα (*Associativity*): $\alpha \boldsymbol{t} \beta(\beta \boldsymbol{t} \gamma) = \alpha \boldsymbol{t} \beta(\boldsymbol{t} \gamma)$
3. Μονοτονία (*Monotony*): $\beta \le \gamma \Rightarrow \alpha \boldsymbol{t} \beta \le \alpha \boldsymbol{t} \gamma$
4. Boundary Condition: $\alpha \boldsymbol{t} 1 = \alpha$

Αντίστοιχα, αν μία πράξη $\boldsymbol{s} : [0,1]\times[0,1] \to [0,1]$ ικανοποιεί τα παραπάνω (1-4) και ισχύει $\alpha \boldsymbol{s} 0 = \alpha$ τότε καλείται *triangular conorm* (*σύμμετρο t ή t-conorm* εν συντομία ή *s-norm*).

Γενικότερα, οι *t-norm* και *s-norm* τελεστές μπορούν να θεωρηθούν ως αντικαταστάτες των λογικών τελεστών «AND» και «OR» που ορίζονται στη λογική Bool (*Boolean Logic*) [2]. Οι *min* και *max* τελεστές δεν είναι οι μόνοι που χρησιμοποιούνται για ασαφή ένωση και τομή. Ο Zadeh στην πρώτη του δημοσίευση [3] όρισε δύο τελεστές, έναν για ασαφή τομή - *αλγεβρικό άθροισμα* (*algebraic sum*[*]),

$$[\ \mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)\ ] \tag{2.17}$$

και έναν για ασαφή ένωση - *αλγεβρικό γινόμενο* (*algebraic product*),

$$[\ \mu_{A\cap B}(x) = \mu_A(x)\,\mu_B(x)\ ] \tag{2.18}$$

[*] Είναι γνωστό και ως *probabilistic OR ή probor*

Αργότερα προτάθηκαν και άλλοι τελεστές *t* εμπειρικά ή αξιωματικά για ασαφή ένωση και τομή (βλ. Πίνακα 2.4) αλλά οι περισσότεροι δεν έχουν πρακτική εφαρμογή σε εφαρμογές της επιστήμης των μηχανικών. Τις περισσότερες φορές χρησιμοποιείται ο τελεστής t-norm ελαχίστου, $\top min(a,b)=min\{a,b\}$ ή αλγεβρικού γινομένου, $\top prod(a,b)=a\cdot b$ για τομή, ο τελεστής s-norm μεγίστου, $\perp max(a,b)=max\{a,b\}$ ή αλγεβρικού αθροίσματος, $\perp asum(a,b)=a+b-a\cdot b$ για την ένωση και ο $1 - \mu_A(x)$ για το συμπλήρωμα [4] (σελ. 42-43).

	t-norm (*)		**s-norm ($\oplus$)**
Φραγμένο γινόμενο	$x * y = \max\{0, x+y-1]\}$	Φραγμένο άθροισμα	$x \oplus y = \max\{1, x+y]\}$
Δραστικό γινόμενο	$x * y = \begin{cases} x, & \text{αν } y=1 \\ y, & \text{αν } x=1 \\ 0, & \text{αν } x,y<1 \end{cases}$	Δραστικό άθροισμα	$x \oplus y = \begin{cases} x, & \text{αν } y=0 \\ y, & \text{αν } x=0 \\ 0, & \text{αν } x,y>0 \end{cases}$

Πίνακας 2.4: Επιπρόσθετοι τελεστές ασαφών συνόλων

Θεωρώντας τα ασαφή σύνολα *A,B,C* οι πράξεις των ασαφών συνόλων έχουν τις παρακάτω ιδιότητες και συνοψίζονται στον πίνακα που ακολουθεί (βλ. Πίνακα 2.5).

Νόμοι De Morgan	$\overline{A \cup B} = \overline{A} \cap \overline{B}$, $\overline{A \cap B} = \overline{A} \cup \overline{B}$
Αντιμεταθετικότητα (Commutativity)	$A \cup B = B \cup A$, $A \cap B = B \cap A$
Προσεταιριστικότητα (Associativity)	$(A \cup B) \cup C = A \cup (B \cup C)$ $(A \cap B) \cap C = A \cap (B \cap C)$
Επιμεριστικότητα (Distributivity)	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
Απορόφηση (Absorption)	$(A \cap B) \cup A = A$, $(A \cup B) \cap A = A$
Ανακλαστικότητα (Idempotency)	$A \cup A = A$, $A \cap A = A$
Ταυτότητα (Law of Identity)	$A \cap (X \times [0,1]) = A$ $A \cup \varnothing = A$, $A \cap \varnothing = \varnothing$ $A \cup (X \times [0,1]) = U \times [0,1]$
Επαναφορά (Double negation)	$\overline{\overline{A}} = A$
Μεταβατικότητα	$A \subset B$ *και* $B \subset C$ *τότε* $A \subset C$
Οι παρακάτω ιδιότητες ισχύουν στα κλασσικά σύνολα αλλά δεν ισχύουν στα ασαφή σύνολα	
Δεν τηρείται η Αρχή της Αντίφασης (Law of Contradiction not satisfied)	$A \cap \overline{A} \neq 0$
Δεν τηρείται η Αρχή του Αποκλειόμενου Μέσου (Excluded Middle not satisfied)	$A \cup \overline{A} \neq X$

Πίνακας 2.5: Ιδιότητες πράξεων ασαφών συνόλων

2.2.3 Κανόνες του Θέτειν και του Αναιρείν

Η κλασική λογική στηρίζει τη διαδικασία απόδειξης εξ' ολοκλήρου σε «λογικές ταυτολογίες». Η ταυτολογία είναι μια πρόταση αποτελούμενη από συνδυασμό άλλων προτάσεων. Οι κυριότερες λογικές ταυτολογίες, στις οποίες ανάγονται και όλες οι υπόλοιπες, είναι οι ακόλουθες:

- Modus Ponens (κανόνας του Θέτειν): $\{A \wedge (A \to B)\} \to B$
- Modus Tolens (κανόνας του Αναιρείν): $\{(A \to B) \wedge \sim B\} \to \sim A$

Η γενική μορφή της λογικής ταυτολογίας του Modus Ponens (*MP*) μπορεί να διατυπωθεί ως εξής:

Προϋπόθεση (*Premise*): A is true

Συνεπαγωγή (*Implication*): if A then B

Συμπέρασμα (*Consequent*): B is true

Στην ειδική περίπτωση της ασαφούς λογικής ο MP επεκτείνεται στον γενικευμένο κανόνα του Θέτειν ή Generalized Modus Ponens (*GMP*) ή Fuzzy Modus Ponens αντικαθιστώντας τα καθορισμένα A και B σύνολα με ασαφή A και B σύνολα ή προτάσεις. Γενικεύοντας το MP μας επιτρέπει διαφορετικές προϋποθέσεις (*premises*) να καταλήγουν σε διαφορετικά συμπεράσματα (*consequents*) χρησιμοποιώντας την ίδια συνεπαγωγή (*implication*):

Προϋπόθεση (*Premise*): A' is true

Συνεπαγωγή (*Implication*): if $x=A$ then $y=B$

Συμπέρασμα (*Consequent*): B' is true

Στην περίπτωση του GMP το ασαφές σύνολο A' δεν είναι απαραίτητα το ίδιο όπως το ασαφές σύνολο A της εισόδου του κανόνα (*rule antecedent*) και αντίστοιχα το ασαφές σύνολο B δεν είναι απαραίτητα το ίδιο όπως το ασαφές σύνολο B' της εξόδου του κανόνα (*rule consequent*).

2.2.4 Ασαφείς Σχέσεις

Ένας ασαφής κανόνας εκφράζει μια συνεπαγωγή μεταξύ ασαφών προτάσεων, που συχνά συναντάμε στη φυσική γλώσσα, όπως για παράδειγμα: «Αν το αυτοκίνητο κάνει θόρυβο, υπάρχει πιθανότητα να έχει μηχανικό πρόβλημα». Ο ασαφής κανόνας περιγράφεται μαθηματικά από μια ασαφή σχέση μεταξύ των συνόλων της υπόθεσης και τού συμπεράσματος του κανόνα.

Έστω $X=\{x_1,\ldots,x_n\}$ και $Y=\{y_1,\ldots,y_n\}$ δυο υπερσύνολα αναφοράς και A,B τα αντίστοιχα ασαφή υποσύνολα τους. Η $R_i(x,y)$ ονομάζεται ασαφής σχέση (*fuzzy relation*) και είναι ένα ασαφές σύνολο που ορίζετε στο Καρτεσιανό Γινόμενο (*Cartesian Product*), $X \times Y = \{(x,y)/\, x \in X, y \in Y\}$ και περιγράφεται από τη συνάρτηση συμμετοχής (Εξ. 2.19), που αντιπροσωπεύει για κάθε ζεύγος (x,y) το βαθμό σύνδεσης ανάμεσα στα x και y,

$$\mu_R(x_n, y_n) \mid x_n \in X, y_n \in Y \tag{2.19}$$

Στην ειδική περίπτωση όπου X=Y έχουμε μια *δυαδική* ασαφή σχέση (binary fuzzy relation) επί του συνόλου X.

Έστω ο ασαφής κανόνας "***R*$_i$:** IF x is A_i AND y is B_i THEN z is C_i", τότε η ασαφής σχέση διατυπώνεται ως εξής: $R_i = (A_i \times B_i) \rightarrow C_i$, όπου $\rightarrow$ το σύμβολο της ασαφούς συνεπαγωγής (*fuzzy implication*). Είναι εμφανές ότι η διαφορά μιας καθορισμένης σχέσης (*crisp relation*) από μια ασαφή σχέση είναι ότι για την πρώτη ισχύει $\mu_R(x, y) \in \{0,1\}$ ενώ για τη δεύτερη $\mu_R(x, y) \in [0,1]$.

2.2.4.1 Πράξεις Ασαφών Σχέσεων

Οι πράξεις μεταξύ των ασαφών σχέσεων ορίζονται κατ' αναλογία με τις πράξεις μεταξύ των ασαφών συνόλων. Συνεπώς, για τις ασαφής σχέσεις R_1 και R_2 στον χώρο $X \times Y$ είναι εύκολο να δείξουμε ότι (βλ. Πίνακα 2.6),

Τομή (Intersection)	$R_1 \cap R_2 : \mu_{R_1 \cap R_2}(x, y) = \min\{\mu_{R_1}(x, y), \mu_{R_2}(x, y)\}$
Ένωση (Union)	$R_1 \cup R_2 : \mu_{R_1 \cup R_2}(x, y) = \max\{\mu_{R_1}(x, y), \mu_{R_2}(x, y)\}$
Συμπλήρωμα (Complement)	$\overline{R}_1 : \mu_{\overline{R}_1}(x, y) = 1 - \mu_{R_1}(x, y)$
Έγκληση (Inclusion)	$R_1 \subseteq R_2 : \mu_{R_1}(x, y) \leq \mu_{R_2}(x, y)$

Πίνακας 2.6: Πράξεις μεταξύ ασαφών σχέσεων

Σύνθεση Ασαφών Σχέσεων

Έστω ο κανόνας "R: IF x is A THEN y is B" μεταξύ των ασαφών συνόλων A και B,

$$\begin{aligned} A &= \{(x, \mu_A(x)) \mid x \in X\} \\ B &= \{(y, \mu_B(y)) \mid y \in Y\} \end{aligned} \tag{2.20}$$

και μιας ασαφούς σχέσης R επί του $X \times Y$,

$$R = \{((x, y), \mu_R(x, y)) \mid (x, y) \in X \times Y\} \tag{2.21}$$

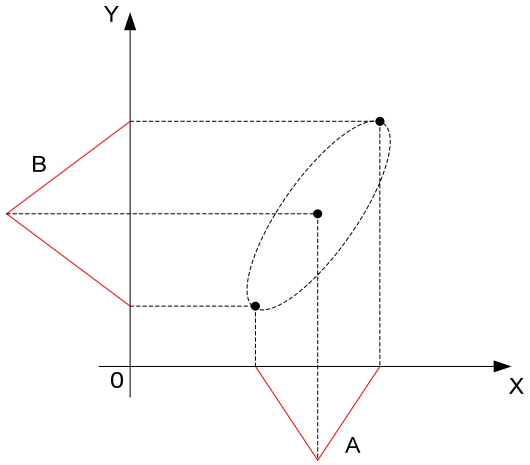
το B δίνεται από τη σχέση (όπου το $\circ$ συμβολίζει τη σύνθεση),

$$B = A \circ R \tag{2.22}$$

με συνάρτηση συμμετοχής,

$$\mu_B(x) = \max_{x \in X}\{\min[\mu_A(x), \mu_R(x, y)]\} \tag{2.23}$$

Αν αντί για το *min* τελεστή χρησιμοποιηθεί ο τελεστής του γινομένου (*Larsen's product*) (Larsen, 1980) τότε η μέθοδος σύνθεσης ονομάζεται *max-prod*. Η εικόνα της σύνθεσης μια ασαφούς σχέσης φαίνεται στο Σχήμα 2.4 παρακάτω. Είναι προφανές ότι για R_i κανόνες, ο κάθε κανόνας διαιρεί το χώρο $X \times Y$ σε μια ασαφή περιοχή που οριοθετείτε από το διαμερισμό αυτού του χώρου ανάλογα με τον αριθμό των κανόνων.



Σχήμα 2.4: Εικόνα της σύνθεσης μιας ασαφούς σχέσης στον χώρο $X \times Y$

2.2.4.2 Συνθετικός Κανόνας Συμπερασμού

Η διαδικασία εξαγωγής συμπερασμάτων σε αβεβαιότητα περιλαμβάνει τα παρακάτω βήματα:

- Τη μετατροπή των IF-THEN κανόνων σε ασαφείς σχέσεις.
- Την εξαγωγή συμπεράσματος χρησιμοποιώντας τις προηγούμενες ασαφείς σχέσεις και το συνθετικό κανόνα συμπερασμού (*compositional rule of inference*).

Έστω τώρα μία συνεπαγωγή R: IF A THEN B (ή R=$A \rightarrow B$) με A,B ασαφή σύνολα,

$$A = \left\{ \left(x_i, \mu_A \left(x_i \right) \right) | \, x_i \in X \right\} \text{ και } B = \left\{ \left(y_i, \mu_B \left(y_i \right) \right) | \, y_i \in Y \right\} \tag{2.24}$$

μία ασαφής σχέση ορίζεται ως εξής:

$$R(X,Y) = \left\{ \left(x_n, y_n \right), \mu_R \left(x_n, y_n \right) | \left(x_n, y_n \right) \in X \times Y \right\} \tag{2.25}$$

Συνεπαγωγή του Zadeh

Η συνεπαγωγή του Zadeh που παρουσίασε στην πρώτη δημοσίευση [3] του ορίζεται ως:

$$R = A \rightarrow B = (X \times B) \otimes (\bar{A} \times Y) \tag{2.26}$$

Το σύμβολο $\otimes$ δηλώνει το φραγμένο άθροισμα (*bounded sum*) που ορίζεται ως:

$$\mu_{A \otimes B}(x) = \min \left(1, \mu_A(x) + \mu_B(x) \right) \tag{2.27}$$

και η συνάρτηση συμμετοχής [5], [6] δίνεται από,

$$\mu_R(x_i, y_j) = \min\left\{1, 1 - \mu_A(x_i) + \mu_B(y_j)\right\} \tag{2.28}$$

Η συνεπαγωγή αυτή είναι δύσχρηστη και δεν αποδέχεται εύκολη υπολογιστική λύση. Περίπου δέκα χρόνια αργότερα ο Mamdani [7] παρουσίασε μια απλούστευση της*.

Συνεπαγωγή του Mamdani

$$R = A \to B = A \times B \tag{2.29}$$

με συνάρτηση συμμετοχής:

$$\mu_R(x_i, y_j) = \min\left\{\mu_A(x_i), \mu_B(y_j)\right\} \tag{2.30}$$

Για n εξαρτημένους κανόνες, η ασαφής σχέση R_i δημιουργείται από τη συνεπαγωγή $A_i \to B_i$ (όπου i=1,…, n). Η συνολική ασαφής σχέση R προκύπτει ως εξής:

κατά Zadeh,

$$R = \bigcap_{i=1}^{n} R_i \tag{2.31}$$

και κατά Mamdani,

$$R = \bigcup_{i=1}^{n} R_i \tag{2.32}$$

Ολοκληρώνοντας, η συνάρτηση συμμετοχής της $\mu_R(x_i, y_j)$, μπορεί να υπολογισθεί με έναν από τους τελεστές συνεπαγωγής (*implication operators*) $x \to y$, που δίνονται στον Πίνακα 2.7 [8], [9], [10], [11].

Για τη Mamdani συνεπαγωγή (1) μπορεί επίσης να χρησιμοποιηθεί το αλγεβρικό γινόμενο (2), το φραγμένο γινόμενο (3) και το δραστικό γινόμενο (4). Αυτές οι συνεπαγωγές ικανοποιούν τον ίδιο όρο που είναι γνωστός και ως μέτρο *t* ή *t-norm* (βλ. §2.2.2.1). Οι συνεπαγωγές 1-4 (βλ. Πίνακα 2.7) χρησιμοποιούν το *max*-τελεστή για τη συνάθροιση των ανεξάρτητων συμπερασμάτων του κάθε κανόνα όπως στη μέθοδο Mamdani (βλ. Εξ. 2.32). Αντίστοιχα οι συνεπαγωγές 5-8 (βλ. Πίνακα 2.7) χρησιμοποιούν το *min*-τελεστή όπως στη μέθοδο Zadeh (βλ. Εξ. 2.31).

* Το 1975 ο καθηγητής Ebrahim Mamdani του London University κατασκεύασε έναν από τους πρώτους ασαφείς ελεγκτές για τον έλεγχο ενός συστήματος ατμομηχανής με λέβητα εφαρμόζοντας ασαφείς κανόνες διαμορφωμένους από την γνώση έμπειρων χειριστών.

Στη βιβλιογραφία αναφέρονται αρκετές εργασίες που συγκρίνουν διάφορες μεθόδους ασαφών συνεπαγωγών [12], [13], [11], [14], [15].

1	Mamdani (Κανόνας Ελαχίστου) [16]	$\mu_R(x_i, y_j) = \min\left\{\mu_A(x_i), \mu_B(y_j)\right\}$
2	Larsen (Αλγεβρικό Γινόμενο) [17], [18]	$\mu_R(x_i, y_j) = \mu_A(x_i) \cdot \mu_B(y_j)$
3	Φραγμένο Γινόμενο (Bounded Product)	$\mu_R(x_i, y_j) = \max\left\{0, \mu_A(x_i) + \mu_B(y_j) - 1\right\}$
4	Δραστικο Γινόμενο (Drastic Product)	$\mu_R(x_i, y_j) = \begin{cases} \mu_A(x_i), & \text{καθώς } \mu_A(y_j) = 1 \\ \mu_B(y_j), & \text{καθώς } \mu_A(x_i) = 1 \\ 0, & \text{αλλιώτικα} \end{cases}$
5	Lukasiewicz [19], [20]	$\mu_R(x_i, y_j) = \min\left\{1, 1 - \mu_A(x_i) + \mu_B(y_j)\right\}$
6	Bool	$\mu_R(x_i, y_j) = \max\left\{1 - \mu_A(x_i), \mu_B(y_j)\right\}$
7	Gödel [20]	$\mu_R(x_i, y_j) = \begin{cases} 1, & \mu_A(x_i) \le \mu_B(y_j) \\ \mu_B(y_j), & \mu_A(x_i) > \mu_B(y_j) \end{cases}$
8	Goguen I [21]	$\mu_R(x_i, y_j) = \begin{cases} 1, & \mu_A(x_i) \le \mu_B(y_j) \\ \mu_B(y_j) / \mu_A(x_i), & \mu_A(x_i) > \mu_B(y_j) \end{cases}$
9	Zadeh (Κανόνας Μεγίστου) [6]	$\mu_R(x_i, y_j) = \max\left\{1 - \mu_A(x_i), \min\left[\mu_A(x_i), \mu_B(y_j)\right]\right\}$
10	Κανόνας Τυπικής Ακολουθίας [22]	$\mu_R(x_i, y_j) = \begin{cases} 1, & \mu_A(x_i) \le \mu_B(y_j) \\ 0, & \mu_A(x_i) > \mu_B(y_j) \end{cases}$
11	Yager [23]	$\mu_R(x_i, y_j) = \mu_B(y_j)^{\mu_A(x_i)}$
12	Gaines	$\mu_R(x_i, y_j) = \max\left\{\left(\mu_A(x_i) \le \mu_B(y_j)\right), \mu_B(y_j) / \mu_A(x_i)\right\}$
13	Dienes [24]	$\mu_R(x_i, y_j) = \max\left\{1 - \mu_A(x_i), \mu_B(y_j)\right\}$

Πίνακας 2.7: Διάφοροι κανόνες συνεπαγωγής

2.3 Μέθοδος Mamdani

Η ασαφής μέθοδος για την εξαγωγή συμπεράσματος του Mamdani είναι η συνηθέστερα ασαφής μεθοδολογία. Η μέθοδος του Mamdani ήταν μεταξύ των πρώτων συστημάτων ελέγχου που χτίστηκαν που χρησιμοποιούν τη θεωρία ασαφών συνόλων. Προτάθηκε το

1975 από τον Ebrahim Mamdani [25] ως προσπάθεια να ελεγχθεί ένας συνδυασμός μηχανών και λεβήτων ατμού με τη σύνθεση ενός συνόλου γλωσσικών κανόνων ελέγχου που λήφθηκαν από τους πεπειραμένους ανθρώπινους χειριστές. Η προσπάθεια του Mamdani βασίστηκε σε δημοσίευση του Lotfi Zadeh το 1973 για τους ασαφείς αλγορίθμους για σύνθετα συστήματα και τις διαδικασίες απόφασης [5]. Αν και η διαδικασία συμπεράσματος που περιγράφεται παρακάτω διαφέρει κάπως από τις μεθόδους που περιγράφονται στην αρχική δημοσίευση, η βασική ιδέα είναι σχεδόν η ίδια.

Η ασαφής μέθοδος εξαγωγή συμπεράσματος τύπου Mamdani περιλαμβάνει τα παρακάτω βήματα:

- Ασαφοποίηση (*Fuzzification*) των μεταβλητών εισόδου
- Αποτίμηση (*Evaluation*) των ασαφών κανόνων
- Συνάθροιση (*Aggregation*) των κανόνων εξόδου
- Από-ασαφοποίηση (*Defuzzification*)

Παρακάτω θα εξετάσουμε ένα απλό πρόβλημα δυο μεταβλητών εισόδου x_1, x_2 (μέλος της υπόθεσης-antecedents) και μιας μεταβλητής εξόδου y (μέλος του συμπεράσματος-consequents) που περιλαμβάνει δύο ασαφείς κανόνες με A_1^1, A_2^1, A_1^2, A_2^2 τα ασαφή σύνολα που αντιστοιχούν στις εισόδους και A_1^3, A_2^3 τα ασαφή σύνολα που αντιστοιχούν στις εξόδους,

Κανόνας *R1*: IF x_1 IS A_1^1 AND x_2 IS A_1^2 THEN y IS B_1

Κανόνας *R2*: IF x_1 IS A_2^1 OR x_2 IS A_2^2 THEN y IS B_2

Θεωρούμε ένα ζεύγος καθορισμένων τιμών $\overline{x}_1$, $\overline{x}_2$ των μεταβλητών εισόδου x_1, x_2 και στη συνεχεία υπολογίζουμε το βαθμό ενεργοποίησης* κάθε κανόνα για το ζεύγος εισόδου $\overline{x}_1$, $\overline{x}_2$,

Βαθμός ενεργοποίησης *R1*: $\mu_1 = \min\left\{\mu_{A_1^1}(\overline{x}_1), \mu_{A_1^2}(\overline{x}_2)\right\}$

Βαθμός ενεργοποίησης *R2*: $\mu_2 = \max\left\{\mu_{A_2^1}(\overline{x}_1), \mu_{A_2^2}(\overline{x}_2)\right\}$

όπου $\mu_{A_i^1}(\overline{x}_1)$, $\mu_{A_i^2}(\overline{x}_2)$, οι συναρτήσεις συμμετοχής των $\overline{x}_1$, $\overline{x}_2$ στα ασαφή σύνολα A_i^1, A_i^2 αντίστοιχα.

Στη συνέχεια εφαρμόζουμε το βαθμό ενεργοποίησης που προέκυψε, στα ασαφή σύνολα της εξόδου για να πάρουμε τελικά το συμπέρασμα κάθε κανόνα ως εξής:

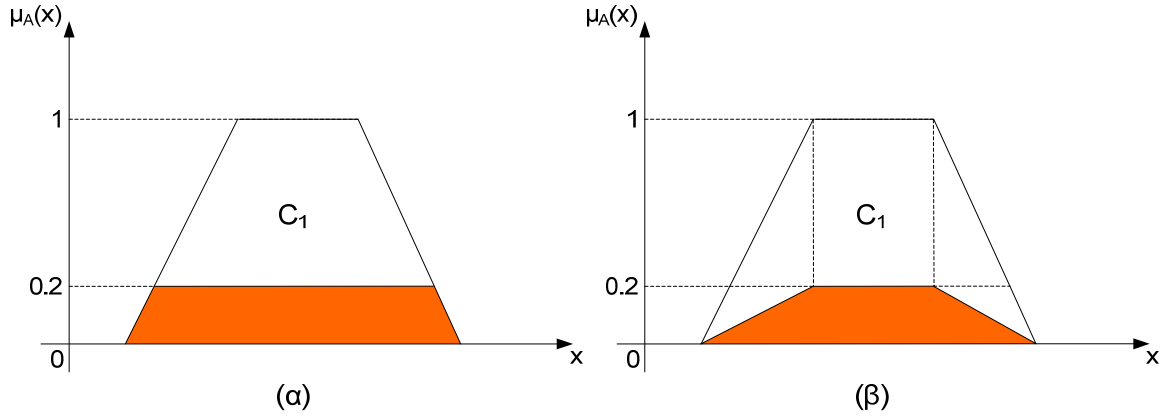
Συμπέρασμα *R1*: $\mu_{B_1'}(z) = \min\left\{\mu_1, \mu_{B_1}(y)\right\}$

Συμπέρασμα *R2*: $\mu_{B_2'}(z) = \min\left\{\mu_2, \mu_{B_2}(y)\right\}$

Στην περίπτωση που χρησιμοποιηθεί ο τελεστής *min*, η παραγωγή της εξόδου του κανόνα γίνεται με ευθεία αποκοπή (*clipping*) του τμήματος της καμπύλης της συνάρτησης

* Αντί τις min μπορεί να χρησιμοποιηθεί η prod, και αντί της max ή asum

συμμετοχής από την τιμή συμμετοχής και πάνω (βλ. Σχήμα 2.5(α)). Μια άλλη, καλύτερη αλλά πιο πολύπλοκη μέθοδος (λόγω του ότι διατηρείται το σχήμα του ασαφούς συνόλου) είναι η διαβαθμισμένη αποκοπή της καμπύλης (*scaling*) με τη χρήση του τελεστή του γινομένου (*Larsen's product,* βλ. §2.2.4.2, Πίνακα 2.7 και Σχήμα 2.5(β)).



Σχήμα 2.5: (α) Παραγωγή της εξόδου ενός κανόνα με ευθεία αποκοπή (*clipping*), (β) και με διαβαθμισμένη αποκοπή (*scaling*) του τμήματος της συνάρτησης συμμετοχής

Η ολική έξοδος $\mu_B(z)$ μπορεί να υπολογιστεί χρησιμοποιώντας διάφορες μεθόδους αξιολόγησης αντιδράσεων ή πιο απλά συναθροίζοντας (*aggregation*) τα ανεξάρτητα αποτελέσματα $\mu_{B_i'}(z)$ για κάθε κανόνα στην ασαφή βάση γνώσης χρησιμοποιώντας τον τελεστή ένωσης.

$$\textbf{Ολικό αποτέλεσμα}: \ \mu_B(z) = \max\left\{\mu_{B_1'}(z), \mu_{B_2'}(z)\right\} \tag{2.33}$$

Γενικά για μια ασαφή βάση *n*-κανόνων συναθροίζουμε (*aggregate*) τα *n*-ανεξάρτητα αποτελέσματα ως εξής:

$$\mu_B(z) = \bigcup_{i=1}^{n}\{\mu_{B_i'}(z)\} \tag{2.34}$$

Έχουν αναφερθεί διάφοροι μέθοδοι από-ασαφοποίησης στη βιβλιογραφία [26] (βλ. §2.5.4.1), αλλά η πιο δημοφιλής τεχνική είναι το κέντρο βάρους (*centroid*), που βρίσκει την τεταγμένη του κέντρου βάρους (*center of gravity – COG*) της επιφάνειας μεταξύ της καμπύλης (από τη συνάθροιση) και των αξόνων και υπολογίζεται από τον κάτωθι τύπο:

$$COG = \frac{\int_a^b \mu_A(x)\,x\,dx}{\int_a^b \mu_A(x)\,dx} \tag{2.35}$$

Πρακτικά και για λόγους αποφυγής αύξησης της πολυπλοκότητας χρησιμοποιείτε ένα δείγμα σημείων για τον υπολογισμό του κέντρου βάρους και όχι συνεχείς τιμές της συνάρτησης, επομένως η Εξίσωση 2.35 στη διακριτή της μορφή γίνεται,

$$COG = \frac{\sum_{x=a}^{b} \mu_A(x)\, x}{\sum_{x=a}^{b} \mu_A(x)} \tag{2.36}$$

Εναλλακτικά αν οι είσοδοι δεν είναι καθορισμένες τιμές $\bar{x}_1$, $\bar{x}_2$ αλλά δίνονται από ασαφή σύνολα A_1', A_2' τότε ο βαθμός ενεργοποίησης υπολογίζεται όπως φαίνεται παρακάτω,

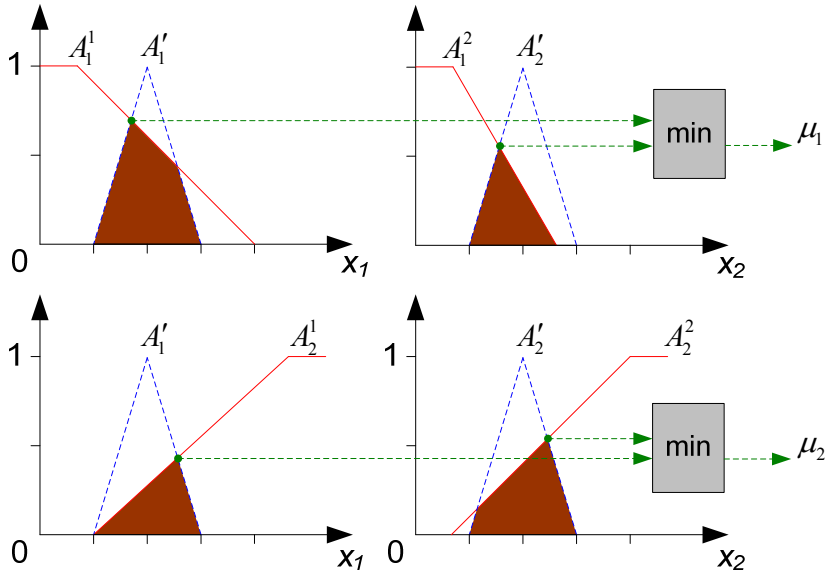
Βαθμός ενεργοποίησης R_1:

$$\mu_1 = \min\left\{\max_{x_1}\left[\mu_{A_1^1}(x_1), \mu_{A_1'}(x_1)\right], \max_{x_2}\left[\mu_{A_1^2}(x_2), \mu_{A_2'}(x_2)\right]\right\}$$

Βαθμός ενεργοποίησης R_2:

$$\mu_2 = \min\left\{\max_{x_1}\left[\mu_{A_2^1}(x_1), \mu_{A_1'}(x_1)\right], \max_{x_2}\left[\mu_{A_2^2}(x_2), \mu_{A_2'}(x_2)\right]\right\}$$

Η περίπτωση αυτή επεξηγείται με γραφικό τρόπο στο Σχήμα 2.6.

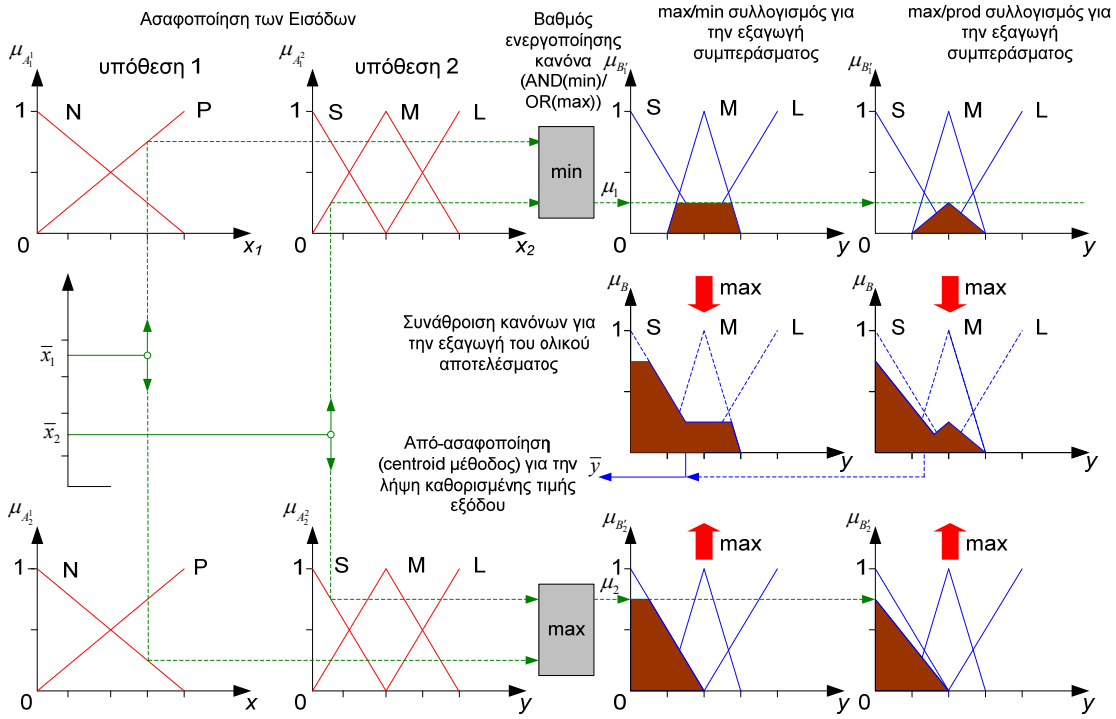


Σχήμα 2.6: Υπολογισμός προσαρμοστικότητας για ασαφείς εισόδους A_1', A_2'

Η ασαφής εξαγωγή συμπεράσματος τύπου Mamdani (βλ. Σχήμα 2.7) προϋποθέτει ότι οι συναρτήσεις συμμετοχής εξόδου είναι ασαφή σύνολα. Μετά από τη διαδικασία συνάθροισης (*aggregation*), υπάρχει ένα ασαφές σύνολο για κάθε μεταβλητή παραγωγής που χρειάζεται από-ασαφοποίηση. Είναι δυνατό, και σε πολλές περιπτώσεις αποδοτικότερος τρόπος, να χρησιμοποιηθεί ασαφές σύνολο με ένα στοιχείο ως συνάρτηση συμμετοχής εξόδου, παρά ένα διανεμημένο ασαφές σύνολο. Αυτό είναι μερικές φορές γνωστό ως μονότιμος (*singleton*) τύπος συνάρτηση συμμετοχής, και μπορεί να θεωρηθεί ως ένα προ-αποσαφοποιημένο ασαφές σύνολο. Ενισχύει την αποδοτικότητα της διαδικασίας της από-σαφοποιησης (*defuzzification*) επειδή απλοποιεί πολύ τον υπολογισμό

που διαφορετικά απαιτείται με τη γενικότερη μέθοδο Mamdani, η οποία βρίσκει το κέντρο βάρους (*centroid*) μιας δυσδιάστατης συνάρτησης. Επομένως αντί για την ανάγκη ολοκλήρωσης της δυσδιάστατης συνάρτησης για να βρεθεί το κέντρο βάρους, χρησιμοποιούμε το σταθμισμένο μέσο όρο (*weighted average*) μερικών σημείων στοιχείων. Τα συστήματα Takagi-Sugeno-τύπων υποστηρίζουν αυτόν τον τύπο προτύπου.

Γενικά, τα συστήματα τύπου Takagi-Sugeno μπορούν να χρησιμοποιηθούν για να υλοποιήσουν οποιοδήποτε σύστημα εξαγωγής συμπεράσματος στο οποίο οι συναρτήσεις συμμετοχής εξόδου είναι είτε εξισώσεις ενός οι περισσότερων όρων, είτε σταθερές τιμές (π.χ. τύπου Takagi-Sugeno μηδενικής τάξης).



Σχήμα 2.7: Διεργασία συλλογισμού κατά Mamdani

2.4 Μέθοδος Takagi – Sugeno – Kang

Στη συλλογιστική Takagi-Sugeno [27] που ονομάζεται και «συναρτησιακή συλλογιστική» (*functional reasoning*) το συμπέρασμα των κανόνων δίνεται με τη μορφή γραμμικών συναρτήσεων. Αυτή είναι και η ουσιαστική διαφορά με τη μέθοδο του Mamdani που αναφέρθηκε στην παράγραφο §2.3. Το μοντέλο Takagi-Sugeno έχει αποδειχθεί ότι είναι ικανό να προσεγγίσει οποιαδήποτε συνάρτηση με κάθε βαθμό ακρίβειας [28].

Οι κανόνες έχουν τη μορφή,

Κανόνας R_i:	IF x_1 είναι A_i^1 AND $\cdots$ AND x_n είναι A_i^n
	THEN $y_i = c_i^0 + c_i^1 x_1 + \ldots + c_i^n x_n$ όπου, $i = 1, 2, \ldots, m$

όπου m είναι ο ολικός αριθμός των κανόνων, x_k , k = 1, 2,…, n είναι η k-οστή είσοδος, y_i είναι η έξοδος του i-οστού κανόνα, A_i^k είναι τα ασαφή σύνολα και c_i^k είναι οι παράμετροι (σταθερές) της εξόδου (του συμπεράσματος).

Πρώτα από όλα υπολογίζονται οι βαθμοί αληθείας των ασαφών συνεπαγωγών, που ουσιαστικά αποτελούν τους βαθμούς ενεργοποίησης (*firing strengths*) των αριστερών μελών των κανόνων i. Ήτοι:

$$\mu_i = \left| x_n \ is \ A_i^1 \ \text{AND...AND} \ x_n \ is \ A_i^n \right| = \min[\mu_{A_i^1}(x_1),...,\mu_{A_i^n}(x_n)] \tag{2.37}$$

Πέρα από τον τελεστή *min* για το AND συχνά χρησιμοποιείται και το αλγεβρικό γινόμενο (μέθοδος Larsen) ως εξής:

$$\mu_i = \left| x_n \ is \ A_i^1 \ \text{AND...AND} \ x_n \ is \ A_i^n \right| = \mu_{A_i^1}(x_1) \cdot \mu_{A_i^2}(x_2),...,\mu_{A_i^n}(x_n) \tag{2.38}$$

ή αλλιώς:

$$\mu_i = \prod_{k=1}^{n} \mu_{A_i^k}(x_k) \tag{2.39}$$

Η έξοδος κάθε κανόνα υπολογίζεται από την Εξίσωση 2.40:

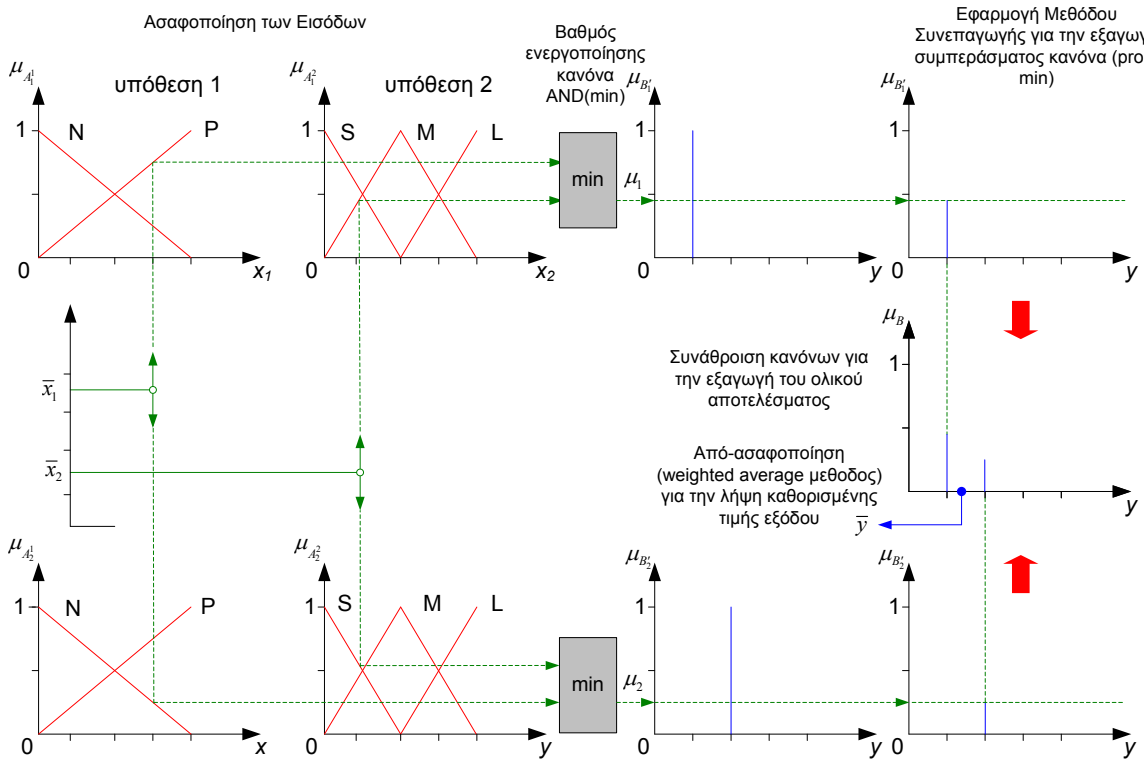
$$y_i = c_i^0 + c_i^1 x_1 + \ldots + c_i^n x_n \tag{2.40}$$

Το ολικό συμπέρασμα του ασαφούς συστήματος δίνεται υπολογίζοντας τη μέση τιμή των εξόδων y_i με βάρη μ_i, ως εξής:

$$y = \frac{\sum_{i=1}^{m} \mu_i y_i}{\sum_{i=1}^{m} \mu_i} \tag{2.41}$$

Εάν στην έξοδο (συμπέρασμα) των κανόνων χρησιμοποιούμε μόνο το σταθερό όρο c_i^0 (αντί της γραμμικής συνάρτησης) τότε έχουμε τη λεγόμενη "απλοποιημένη συναρτησιακή συλλογιστική Takagi-Sugeno" ή συλλογιστική τύπου Takagi-Sugeno μηδενικής τάξης (*T-S type zero-order*).

Ένα τέτοιο παράδειγμα συλλογιστικής τύπου Takagi-Sugeno μηδενικής τάξης για δύο κανόνες, φαίνεται στο Σχήμα 2.8.



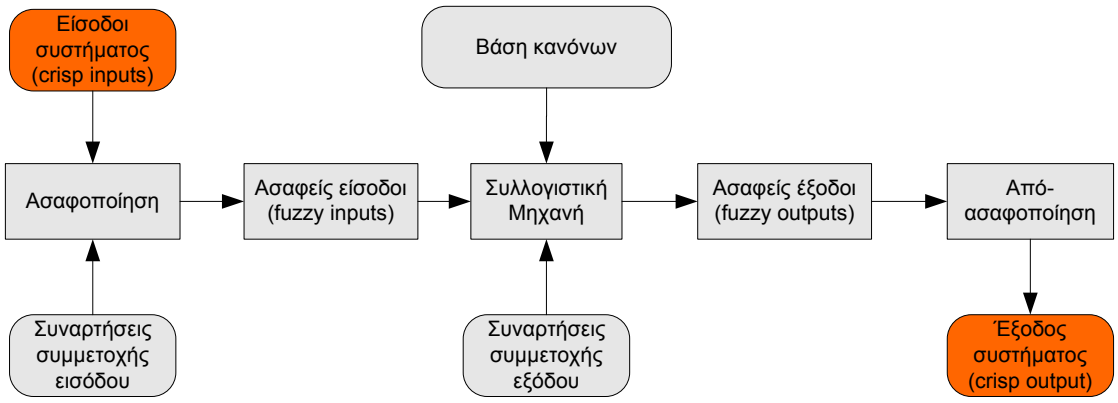
Σχήμα 2.8: Διεργασία συλλογισμού κατά Takagi-Sugeno

Συνοψίζοντας για τις μεθόδους Mamdani και Takagi-Sugeno,

- Η μέθοδος Mamdani είναι ευρέος αποδεκτή για τη σύλληψη έμπειρης γνώσης και επιτυγχάνει την περιγραφή της με έναν πιο «διαισθητικό» ή πιο ανθρώπινο τρόπο (human-like).Ωστόσο είναι αρκετά υπολογιστικά πολύπλοκη μέθοδος.
- Η συλλογιστική Takagi-Sugeno είναι πολύ απλή και οδηγεί σε ταχείς υπολογισμούς, είναι εύκολα εφαρμόσιμη, και αποδίδει ικανοποιητικά σε τεχνικές βελτιστοποίησης και προσαρμογής γεγονός που την καθιστούν κατάλληλη για δυναμικά μη γραμμικά προβλήματα ελέγχου.

2.5 Αρχιτεκτονική Ασαφών Συστημάτων

Η γενική αρχιτεκτονική των ασαφών συστημάτων περιλαμβάνει τέσσερις μονάδες οι οποίες αναφέρονται συνοπτικά στις παραγράφους που ακολουθούν. Ένα γενικό σχηματικό διάγραμμα της αρχιτεκτονικής παρατίθεται στο ακόλουθο σχήμα (βλ. Σχήμα 2.9).



Σχήμα 2.9: Σχηματικό διάγραμμα αρχιτεκτονικής ασαφών συστημάτων

2.5.1 Βάση Ασαφών Κανόνων

Η βάση ασαφών κανόνων (*fuzzy rule base*) ή ασαφής βάση γνώσης είναι μια βάση της μορφής *IF – THEN*. Στον Πίνακα 2.8 που ακολουθεί δίνεται ένα τέτοιο παράδειγμα.

$\boldsymbol{R_1}$**:**	IF x_1 is A_1^1 AND x_2 is A_1^2 AND … AND x_m is A_1^m THEN y_1 is B_1^1 AND y_2 is B_1^2 AND … AND y_p is B_1^p
$\boldsymbol{R_2}$**:**	IF x_1 is A_2^1 AND x_2 is A_2^2 AND … AND x_m is A_2^m THEN y_1 is B_2^1 AND y_2 is B_2^2 AND … AND y_p is B_2^p
	⋮
$\boldsymbol{R_i}$**:**	IF x_1 is A_i^1 AND x_2 is A_i^2 AND … AND x_m is A_i^m THEN y_1 is B_i^1 AND y_2 is B_i^2 AND … AND y_p is B_i^p

Πίνακας 2.8: Βάση ασαφών κανόνων

όπου $x_1, x_2, \ldots, x_m$ είναι οι είσοδοι, $y_1, y_2, \ldots, y_p$ οι έξοδοι, A_1^1, $A_2^1, \ldots, A_i^1$ τα ασαφή σύνολα που αντιστοιχούν στις εισόδους και B_1^1, B_2^1, … , B_i^1 τα ασαφή σύνολα που αντιστοιχούν στις εξόδους.

2.5.2 Ασαφής Συλλογιστική Μηχανή

Η ασαφής συλλογιστική μηχανή (*inference machine*) ή μηχανισμός εξαγωγής ασαφών συμπερασμάτων ή αποφάσεων. Αποτελεί τον πυρήνα του ασαφούς συστήματος και περιέχει τη λογική λήψης αποφάσεων. Περιέχει, συνήθως, εκτός από τους ασαφείς (γλωσσικούς) κανόνες και ένα τμήμα βάσης αριθμητικών δεδομένων τα οποία απαιτούνται για τη διαδικασία εξαγωγής των αποτελεσμάτων.

2.5.3 Μονάδα Ασαφοποίησης

Μια μονάδα ασαφοποίησης (*ασαφοποιητική μονάδα* ή *fuzzification unit*) αντιστοιχεί τα καθορισμένα δεδομένα εισόδου στα ασαφή σύνολα που έχουν οριστεί στο σύστημα.

Η μονάδα ασαφοποίησης (ασαφοποιητής, *fuzzifier*) εκτελεί τις παρακάτω εργασίες:

- Μετράει (παραλαμβάνει) τις (μη ασαφείς) τιμές των εισόδων του συστήματος.
- Απεικονίζει τις περιοχές μεταβολής των τιμών εισόδου σε κατάλληλα υπερσύνολα αναφοράς.
- Ασαφοποιεί τις εισερχόμενες τιμές των εισόδων δηλαδή τις μετατρέπει σε ασαφή ή γλωσσική μορφή

2.5.4 Μονάδα Από-ασαφοποίησης

Μια μονάδα από-ασαφοποίησης (από-ασαφοποιητική μονάδα ή *defuzzification unit*) η οποία αντιστοιχεί τα ασαφή συμπεράσματα ή αποφάσεις που έχουμε ως ασαφή σύνολα σε μια σαφώς καθορισμένη («ντετερμινιστική») μορφή μεταβλητών εξόδου.

Η μονάδα από-ασαφοποίησης (από-ασαφοποιητής, *defuzzifier*) εκτελεί τις παρακάτω εργασίες:

- Απεικονίζει τις περιοχές μεταβολής των τιμών εισόδου σε κατάλληλα υπερσύνολα αναφοράς.
- Από-ασαφοποιεί τα αποτελέσματα που δίνει η ασαφής συλλογιστική μηχανή, δηλαδή τα μετατρέπει σε ντετερμινιστική (μη ασαφή) μορφή για παραπέρα χρήση από επόμενα συστήματα ή διεργασίες απόφασης.

Από-ασαφοποίηση είναι η διαδικασία μετατροπής ενός ασαφούς συνόλου B σε μια καθορισμένη τιμή $\overline{y}$, η οποία είναι και η έξοδος του ασαφούς συστήματος.

Οι κυριότερες μέθοδοι από-ασαφοποίησης συνοψίζονται στην επόμενη παράγραφο.

2.5.4.1 Κυριότερες Μέθοδοι Από-ασαφοποίησης

Ακολουθούν οι πέντε σημαντικότεροι από-ασαφοποιητές [26].

1. **Μεγίστου** (*maximum*):

Ο από-ασαφοποιητής αυτός εξετάζει το ασαφές σύνολο B και στη συνεχεία διαλέγει την τιμή εξόδου του ασαφούς συστήματος y, για την οποία η συνάρτηση συμμετοχής $\mu_B(y)$ έχει μέγιστη τιμή,

$$y_{\max} = \max\left\{\mu_B(y)\right\} \tag{2.42}$$

2. **Μέσου Όρου των Μεγίστων** (*Mean of Maxima – MoM*):

Ο από-ασαφοποιητής αυτός εξετάζει το ασαφές σύνολο *B* και αρχικά προσδιορίζει τις τιμές y για τις οποίες η συνάρτηση $\mu_B(y)$ έχει μέγιστη τιμή, ενώ στη συνέχεια παράγει την τιμή εξόδου υπολογίζοντας το μέσο όρο των τιμών αυτών,

$$ymom = \frac{1}{m}\sum_{i=1}^{m} \max\left\{\mu_B(y_i)\right\} \tag{2.43}$$

3. **Κέντρο Βάρους** (*Centroid*):

Ο από-ασαφοποιητής αυτός προσδιορίζει το κέντρο βάρους (*Center of Gravity – CoG*) $\overline{y}$, της κατανομής του ασαφούς συνόλου *B* και χρησιμοποιεί αυτήν την τιμή ως την έξοδο του ασαφούς συστήματος. Η από-ασαφοποίηση Κεντρικής τιμής δίνεται από,

$$\overline{y} = \frac{\int_S y\mu_B(y)dy}{\int_S \mu_B(y)dy} \tag{2.44}$$

όπου το *S* είναι το στήριγμα της συνάρτησης συμμετοχής $\mu_B(y)$. Διακριτοποιώντας το *S*, η προσέγγιση του $\overline{y}$ μπορεί να γίνει χρησιμοποιώντας την επόμενη Εξίσωση 2.45, που κάνει χρήση αθροίσματος αντί ολοκλήρωσης,

$$\overline{y} = \frac{\sum_{i=1}^{I} y_i\mu_B(y_i)}{\sum_{i=1}^{I} \mu_B(y_i)} \tag{2.45}$$

4. **Ύψους** (*Height*):

Έστω y^{-l} το κέντρο βάρους του ασαφούς συνόλου B^l που σχετίζεται με το βαθμό ενεργοποίησης του κανόνα $R^{(l)}$. Ο συγκεκριμένος από-ασαφοποιητής αξιολογεί πρώτα τη συνάρτηση συμμετοχής $\mu_{B^l}(y)$ στο y^{-l} και στη συνεχεία υπολογίζει την έξοδο του ασαφούς συστήματος, ως εξής:

$$y_h = \frac{\sum_{l=1}^{M} y^{-l}\mu_{B^l}(y^{-l})}{\sum_{l=1}^{M} \mu_{B^l}(y^{-l})} \tag{2.46}$$

Παρόλο που ο παραπάνω τρόπος είναι εύκολος να χρησιμοποιηθεί αδυνατεί στο γεγονός ότι ενώ η y^h εκμεταλλεύεται όλο το σχήμα της συνάρτησης συμμετοχής της υπόθεσης (η πληροφορία βρίσκεται μέσα στο $\mu_{B^l}(y^{-l})$), χρησιμοποιεί από τη συνάρτηση συμμετοχής του συμπεράσματος μόνο το κέντρο του στηρίγματος y^{-l} . Αυτό έχει ως αποτέλεσμα ότι ανεξαρτήτως του πλάτους της συμμετοχής εξόδου, η μέθοδος αυτή δίνει πάντοτε το ίδιο αποτέλεσμα.

5. **Τροποποιημένη Μέθοδος Ύψους** (*Modified Height*):

Δουλεύει όπως και η μέθοδος του ύψους, με τη βασική διαφορά ότι εισάγεται το δ^l που είναι ένα μέγεθος μέτρησης της έκτασης του συμπεράσματος του κανόνα $R^{(l)}$. Στην περίπτωση τριγωνικής ή τραπεζοειδούς συναρτήσεως συμμετοχής το δ^l μπορεί να είναι κάλλιστα το στήριγμα του τριγώνου η του τραπεζίου αντίστοιχα. Η έξοδος δίνεται από:

$$y_{mh} = \frac{\sum_{l=1}^{M} y^{-l}\, \mu_{B^l}(y^{-l}) \big/ \delta^l}{\sum_{l=1}^{M} \mu_{B^l}(y^{-l}) \big/ \delta^l} \tag{2.47}$$

2.6 Εφαρμογές της Ασαφούς Λογικής

Η χρήση της ασαφούς λογικής είναι ενδεδειγμένη στις παρακάτω περιπτώσεις:

- Για πολύ πολύπλοκες διαδικασίες, όταν δεν υπάρχει απλό μαθηματικό μοντέλο.
- Σε υψηλής μη-γραμμικότητας προβλήματα.
- Εάν πρόκειται να γίνει επεξεργασία έμπειρης γνώσης (γλωσσικά διατυπωμένης).

Ενώ δεν θα πρέπει να χρησιμοποιείται σε περιπτώσεις όπου:

- Η κλασσική θεωρία ελέγχου μπορεί να δώσει ικανοποιητικά αποτελέσματα.
- Υπάρχει επαρκές μαθηματικό μοντέλο που μπορεί εύκολα να λυθεί.
- Παρακάτω ακολουθούν κάποια παραδείγματα εφαρμογής της ασαφούς λογικής στην πράξη από διάφορες εταιρίες [29].
- Αυτόματος έλεγχος πυλών υδροφράκτη σε υδροηλεκτρικό εργοστάσιο παραγωγής ισχύος *(Tokio Electric Pow.)*.
- Απλοποιημένος έλεγχος ρομπότ (Hirota, Fuji Electric, Toshiba, Omron).
- Αυτοματοποιημένος στόχος (*camera-aiming*) κάμερας για την αναμετάδοση αθλητικών εκδηλώσεων *(Omron)*.
- Αντικατάσταση εμπειρογνώμονα για την εκτίμηση της δραστηριότητας μετοχών χρηματιστηρίου *(Yamaichi, Hitachi)*.
- Πρόβλεψη ανεπιθύμητων θερμοκρασιακών διακυμάνσεων σε κλιματιστικά συστήματα *(Mitsubishi, Sharp)*.
- Αποδοτικός και ευσταθής έλεγχος μηχανών αυτοκινήτων *(Nissan)*.
- Αυτόματη πλοήγηση (Cruise-control) σε αυτοκίνητα *(Nissan, Subaru)*.
- Βελτίωση της αποδοτικότητας και βελτιστοποίηση της λειτουργίας βιομηχανικών εφαρμογών ελέγχου *(Aptronix, Omron, Meiden, Sha, Micom, Mitsubishi, Nisshin-Denki, Oku-Electronics)*.
- Τοποθέτηση των wafer-steppers (*wafer-steppers positioning*) κατά τη διαδικασία κατασκευής ολοκληρωμένων κυκλωμάτων *(Canon)*.

- Βελτιστοποίηση σχεδιασμού χρονοδιαγραμμάτων λεωφορείων *(Toshiba, Nippon-System, Keihan-Express)*.
- Σύστημα αρχειοθέτησης εγγράφων *(Mitsubishi Elec.)*.
- Σύστημα πρόβλεψης πρόωρης αναγνώρισης σεισμών (Inst. of Seismology Bureau of Metrology, Japan).
- Ιατρική τεχνολογία: Διάγνωση καρκίνου *(Kawasaki Medical School)*.
- Συνδυασμός ασαφούς λογικής με νευρωνικά δίκτυα *(Matsushita)*.
- Αναγνώριση χειρόγραφων συμβόλων με υπολογιστές τσέπης (*pocket computers*) *(Sony)*.
- Αναγνώριση πρόθεσης σε εικόνες από βίντεο κάμερα *(Canon, Minolta)*.
- Αυτόματος έλεγχος ηλεκτροκινητήρα ηλεκτρικής σκούπας με δυνατότητα αναγνώρισης της κατάστασης της επιφάνειας και το βαθμό ακαθαρσίας της *(Matsushita)*.
- Έλεγχος οπίσθιου φωτισμού σε βίντεο κάμερες *(Sanyo)*.
- Εξουδετέρωση δονήσεων σε βίντεο κάμερες *(Matsushita)*.
- Έλεγχος με ένα μόνο κουμπί για πλυντήρια *(Matsushita, Hitatchi)*.
- Αναγνώριση γραφής, αντικειμένων και φωνής *(CSK, Hitachi, Hosai Univ., Ricoh)*.
- Βοήθεια πτήσης ελικοπτέρων *(Sugeno)*.
- Προσομοίωση νομίμων διαδικασιών (Simulation for legal proceedings) (Meihi Gakuin Univ, Nagoy Univ.).
- Σχεδιασμός λογισμικού βιομηχανικών εφαρμογών (Aptronix, Harima, Ishikawajima-OC Engineering).
- Έλεγχος της ταχύτητας των μηχανημάτων και της θερμοκρασίας σε ατσαλό-κατασκευές (*steel-works*) *(Kawasaki Steel, New-Nippon Steel, NKK)*.
- Έλεγχος υπόγειου σιδηροδρόμου με σκοπό τη βελτίωση της άνεσης κατά τη μεταφορά, την ακρίβεια χειρισμού, και την οικονομία κατανάλωσης *(Hitachi)*.
- Βελτίωση κατανάλωσης καυσίμου σε αυτοκίνητα *(NOK, Nippon Denki Tools)*.
- Βελτίωση ευαισθησίας και απόδοσης στον έλεγχο ανελκυστήρων *(Fujitec, Hitachi, Toshiba)*.
- Βελτίωση ασφάλειας σε πυρηνικούς αντιδραστήρες *(Hitachi, Bernard, Nuclear Fuel div.)*.

Κεφάλαιο 3

Γενετικοί Αλγόριθμοι

3.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζεται μία σύντομη εισαγωγή στους Γενετικούς Αλγορίθμους (*Genetic Algorithms – GA*). Οι γενετικοί αλγόριθμοι αναπτύχθηκαν από τον John Holland [30] αρχικά μεν για τη μελέτη του φαινομένου της «φυσικής προσαρμογής», μεταγενέστερα δε για την επίλυση πρακτικών προβλημάτων βελτιστοποίησης χωρίς τη χρήση των παραγώγων της αντικειμενικής συνάρτησης (κόστους ή συμπεριφοράς). Οι γενετικοί αλγόριθμοι (*ΓΑ*) βασίζονται στην έννοια του πληθυσμού των ατόμων (χρωμοσωμάτων) στα οποία εφαρμόζονται γενετικοί τελεστές (πράξεις) της διασταύρωσης (*crossover*), της μετάλλαξης (*mutation*), της επιλογής (*selection*) και του ελιτισμού (*elitism*).

Οι ΓΑ λειτουργούν υπακούοντας στην αρχή της φυσικής επιλογής (*natural selection*) του Δαρβίνου για την οποία ο Άγγλος φιλόσοφος Herbert Spencer εισήγαγε τον όρο: «Επιβίωση του ισχυρότερου προσαρμοστικά ατόμου» ("*Survival of the fittest*"). Ο ΓΑ δεν επιλύει το πρόβλημα με μαθηματικό τρόπο αλλά με βιολογικό. Συνεπώς, έχει μεγάλη ενδογενή ευελιξία και την ελευθερία να επιλέγει μια επιθυμητή βέλτιστη λύση σύμφωνα με τις προδιαγραφές σχεδίασης του προβλήματος/συστήματος. Οι γενετικοί αλγόριθμοι έχουν την ίδια δυνατότητα προσδιορισμού της βέλτιστης (ή σχεδόν βέλτιστης) λύσης ανεξάρτητα εάν οι προδιαγραφές είναι μη γραμμικές, διακριτού χρόνου, πολλών ακροτάτων, υποκείμενες σε ισοτικούς ή ανισοτικούς περιορισμούς ή ακόμα και μη πολυωνυμικά πλήρεις (NP–complete) [31], [32].

Σκοπός του κεφαλαίου είναι να παρουσιάσει τις βασικές έννοιες των γενετικών αλγορίθμων. Αφού δοθούν οι βασικές βιολογικές έννοιες που είναι απαραίτητες για την κατανόηση των ΓΑ (βλ. §3.2), πραγματοποιείται μία σύντομη επεξήγηση των όρων «χώρος αναζήτησης» και «τοπίο προσαρμογής» (βλ. §3.3). Στη συνέχεια (βλ. §3.4)

περιγράφεται ο βασικός ΓΑ και ακολούθως, παρουσιάζονται οι κυριότερες μέθοδοι επιλογής και γενετικών πράξεων που χρησιμοποιούνται στους γενετικούς αλγόριθμους (βλ. §3.5). Τέλος, εκθέτονται τα βασικότερα πεδία εφαρμογής των γενετικών αλγορίθμων (βλ. §3.6).

3.2 Βιολογικές Έννοιες

Στην παράγραφο αυτή παρουσιάζονται οι βασικές βιολογικές έννοιες, οι οποίες είναι απαραίτητες για την κατανόηση των γενετικών αλγορίθμων που εφαρμόζονται σε επιστημονικά και τεχνολογικά προβλήματα. Οι έννοιες αυτές χρησιμοποιούνται ακολουθώντας την αναλογία με την πραγματική βιολογία, αλλά βεβαίως οι τεχνολογικές οντότητες στις οποίες αναφέρονται είναι πολύ απλούστερες από τις πραγματικές βιολογικές οντότητες.

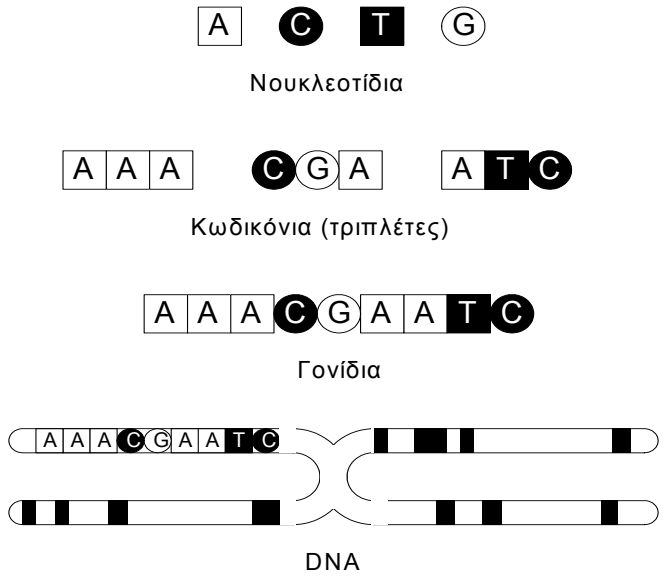
Όλοι οι ζώντες οργανισμοί αποτελούνται από κύτταρα και κάθε κύτταρο περιέχει το ίδιο σύνολο από ένα ή περισσότερα χρωμοσώματα (αλυσίδες από DNA*: *Δεο(σ)ξυριβο(ζο)νουκλεϊ(νι)κό οξύ*) που χρησιμεύουν ως «φωτογραφικό σχεδιάγραμμα» για τον οργανισμό. Ένα χρωμόσωμα μπορεί εννοιολογικά να χωρισθεί σε γονίδια, δηλαδή λειτουργικές ομάδες από DNA κάθε μια από τις οποίες κωδικοποιεί μια συγκεκριμένη πρωτεΐνη. Κάθε γονίδιο είναι τοποθετημένο σε μία συγκεκριμένη θέση *(loci)* του χρωμοσώματος. Οι διάφορες δυνατές εκδοχές για ένα χαρακτηριστικό (λ.χ. χρώμα ματιών μπλε, καστανό, πράσινο) ονομάζονται αλληλόμορφα (*alleles*) γωνίδια.

Πολλοί οργανισμοί έχουν πολλαπλά χρωμοσώματα σε κάθε κύτταρο. Η πλήρης συλλογή του γενετικού υλικού (δηλαδή όλων των χρωμοσωμάτων μαζί) ονομάζεται «γονιδίωμα» (*genome*) του οργανισμού. Ο όρος «γονότυπος» (*genotype*) αναφέρεται σε ένα συγκεκριμένο σύνολο γονιδίων το οποίο περιέχεται στο γονιδίωμα. Κατά την εμβρυακή και μεταγενέστερη ανάπτυξη, ο γονότυπος προκαλεί την έγερση του «φαινοτύπου» *(phenotype)* του οργανισμού, δηλαδή των φυσικών και πνευματικών χαρακτηριστικών του (χρώμα ματιών, χρώμα μαλλιών, ύψος, μέγεθος εγκεφάλου, εξυπνάδα, κλπ.).

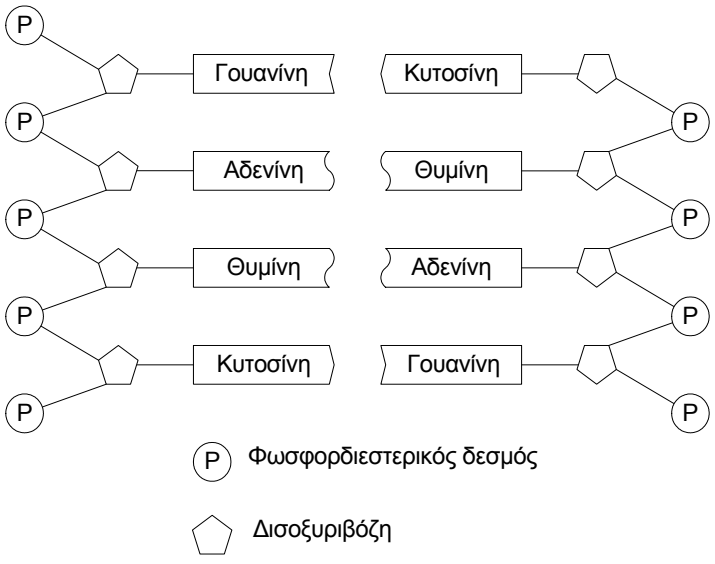
Οργανισμοί των οποίων τα χρωμοσώματα διατάσσονται σε ζευγάρια ονομάζονται διπλοειδικοί (*diploid*) ενώ οι οργανισμοί με χρωμοσώματα όχι σε ζευγάρια ονομάζονται απλοειδικοί (*haploid*). Στη φύση, τα περισσότερα είδη που αναπαράγονται με συνεύρεση των δύο φύλων είναι διπλοειδικά συμπεριλαμβανομένου και του ανθρώπου, ο οποίος έχει 23 ζευγάρια χρωμοσωμάτων σε κάθε σωματικό (μη αναπαραγωγικό) κύτταρο του οργανισμού. Κατά τη διάρκεια της αναπαραγωγής λαβαίνει χώρα ή διαδικασία της διασταύρωσης. Σε κάθε γονέα, ανταλλάσσονται γονίδια ανάμεσα σε κάθε ζευγάρι χρωμοσωμάτων για να σχηματίσουν ένα απλό χρωμόσωμα (*γαμέτη*) και ακολούθως τα απλά αυτά χρωμοσώματα από τους δύο γονείς ζευγαρώνονται για να δημιουργήσουν ένα πλήρες σύνολο διπλοειδών χρωμοσωμάτων. Στους απλοειδείς οργανισμούς, ανταλλάσσονται γονίδια μεταξύ των δύο γονέων (δηλαδή μεταξύ των μη διπλών χρωμοσωμάτων). Οι απόγονοι υπόκεινται σε μετάλλαξη (*mutation*) στην οποία απλά νουκλεοτίδια (στοιχειώδη ψηφία του DNA) αλλάζουν από το γονέα στον απόγονο. Συχνά,

* **D**eoxyribo **N**ucleic **A**cid

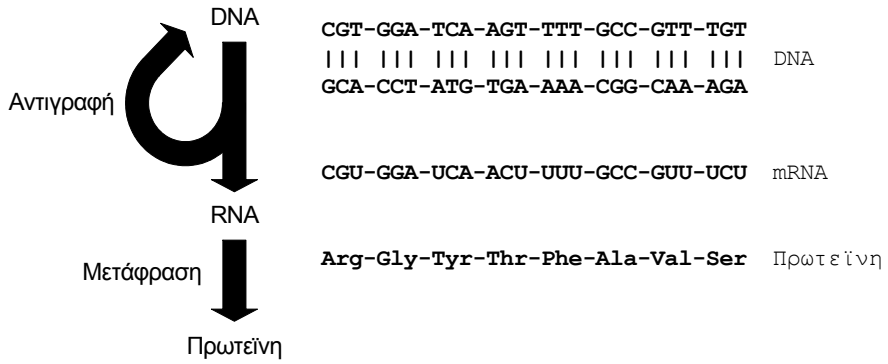
οι αλλαγές αυτές είναι το αποτέλεσμα σφαλμάτων αντιγραφής. Η προσαρμογή (*fitness*) ενός οργανισμού ορίζεται τυπικά ως η πιθανότητα που έχει ο οργανισμός να αναπαραχθεί (*βιωσιμότητα*) ή ως συνάρτηση του αριθμού των απογόνων που έχει ο οργανισμός (*γονιμότητα*). Η ιεραρχική οργάνωση του DNA, η δομή ενός διπλοειδούς DNA και η μετάβαση από DNA σε πρωτεΐνη παρουσιάζονται συμβολικά στα παρακάτω σχήματα (βλ. Σχ. 3.1–3.4) [33], [31].



Σχήμα 3.1: Οργανωτική ιεραρχία του DNA



Σχήμα 3.2: Συμπληρωματική δομή διπλοειδούς DNA



Σχήμα 3.3: Μετάβαση από DNA σε πρωτεΐνη (RNA[†] = Ριβο(ζο)νουκλεϊκό οξύ)

Στους εξελικτικούς – γενετικούς αλγορίθμους, ο όρος χρωμόσωμα αναφέρεται τυπικά σε μια υποψήφια λύση του προβλήματος, η οποία συχνά κωδικοποιείται με μια ακολουθία (*sequence*) από δυαδικά ψηφία (*bits*). Τα «γονίδια» είναι είτε απλά δυαδικά ψηφία ή μικρές αλυσίδες γειτονικών ψηφίων που κωδικοποιούν ένα συγκεκριμένο στοιχείο της υποψήφιας λύσης. Για παράδειγμα, σε ένα πρόβλημα πολυπαραμετρικής βελτιστοποίησης συναρτήσεων, μπορεί να θεωρηθεί ότι τα δυαδικά ψηφία που κωδικοποιούν κάποια συγκεκριμένη παράμετρο συνιστούν το γονίδιο. Ένα αλληλόμορφο σε μια αλυσίδα από δυαδικά ψηφία είναι «1» ή «0». Για μεγαλύτερα αλφάβητα, σε κάθε θέση του γονιδίου μπορεί να υπάρχουν περισσότερα από ένα αλληλόμορφα. Η διασταύρωση συνίσταται τυπικά στην ανταλλαγή γενετικού υλικού μεταξύ των δύο απλοειδών γονέων. Η μετάλλαξη συνίσταται στο "πέταγμα" (αλλαγή) του δυαδικού ψηφίου μιας τυχαία επιλεγείσας θέσης με ένα επίσης τυχαία επιλεγμένο σύμβολο.

Οι περισσότερες εφαρμογές των γενετικών αλγορίθμων χρησιμοποιούν απλοειδή άτομα. Ο γονότυπος κάθε ατόμου σε ένα γενετικό αλγόριθμο που χρησιμοποιεί αλυσίδες δυαδικών ψηφίων είναι απλά η διάταξη των ψηφίων στο χρωμόσωμα του ατόμου. Συχνά, στους γενετικούς αλγορίθμους δεν υπάρχει η έννοια του φαινοτύπου, αν και τελευταία πολλοί ερευνητές πειραματίσθηκαν με γενετικούς αλγορίθμους όπου υπάρχει τόσο γονοτυπικό επίπεδο όσο και φαινοτυπικό επίπεδο.

3.3 Χώροι Αναζήτησης και Τοπία Προσαρμογής

Ο όρος «χώρος αναζήτησης» (*search space*) αναφέρεται σε μια συλλογή υποψηφίων λύσεων ενός προβλήματος και σε κάποια έννοια απόστασης μεταξύ των υποψηφίων λύσεων. Για παράδειγμα, ας θεωρήσουμε το σημαντικό πρόβλημα της υπολογιστικής σχεδίασης πρωτεϊνών. Έστω ότι θέλουμε να χρησιμοποιήσουμε τον υπολογιστή για την αναζήτηση μιας πρωτεΐνης (δηλαδή μιας ακολουθίας αμινοξέων) η οποία διπλώνεται σε μια συγκεκριμένη τρισδιάστατη μορφή προκειμένου να εξουδετερώσει κάποιον ιό. Ο χώρος αναζήτησης είναι το σύνολο όλων των δυνατών πρωτεϊνικών ακολουθιών, ο οποίος είναι θεωρητικά άπειρος σε αριθμό. Για να περιορίσουμε το χώρο αναζήτησης, έστω ότι αναζητούμε όλες τις δυνατές ακολουθίες μήκους το πολύ μέχρι 100, όπου και πάλι έχουμε

[†] **R**ibo **N**ucleic **A**cid

ένα χώρο έρευνας τεραστίων διαστάσεων, αφού υπάρχουν 20 αμινοξέα σε κάθε θέση της ακολουθίας.

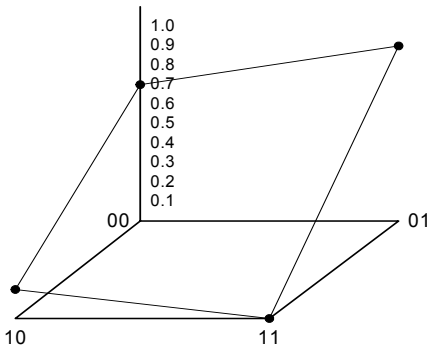
Εάν παραστήσουμε τα 20 αμινοξέα με γράμματα, οι υποψήφιες λύσεις θα έχουν τη μορφή:

AGG MC GBL κοκ.

Η απόσταση μεταξύ δύο ακολουθιών μπορεί να ορισθεί ως ο αριθμός των θέσεων στις οποίες τα αντίστοιχα γράμματα στις δύο ακολουθίες διαφέρουν (απόσταση *Hamming*). Έτσι, η απόσταση μεταξύ των **AGGMCGBL** και **MGGMCGBL** είναι 1 και η απόσταση μεταξύ των **AGGMCGBL** και **LBMPAFGA** είναι 8.

Ένας αλγόριθμος αναζήτησης στο χώρο αυτό είναι μια μέθοδος επιλογής των υποψήφιων λύσεων που πρέπει να δοκιμασθεί σε κάθε βήμα της αναζήτησης. Στις περισσότερες περιπτώσεις η επόμενη υποψήφια λύση που θα δοκιμασθεί εξαρτάται από τα αποτελέσματα της αξιολόγησης προηγουμένων λύσεων. Οι πιο χρήσιμοι αλγόριθμοι είναι αυτοί οι οποίοι υποθέτουν ότι υπάρχει μια συσχέτιση ανάμεσα στην ποιότητα "γειτονικών" υποψήφιων λύσεων. Οι γενετικοί αλγόριθμοι υποθέτουν ότι μπορούν να συνδυασθούν με διασταύρωση, υψηλής ποιότητας γενικές υποψήφιες λύσεις (γονείς) από διάφορες περιοχές του χώρου αναζήτησης, για να παραγάγουν απόγονες υποψήφιες λύσεις υψηλής ή/και υψηλότερης ποιότητας.

Η έννοια του τοπίου προσαρμογής (*fitness landscape*) ορίσθηκε στην περιοχή της πληθυσμιακής γενετικής, για πρώτη φορά, από το βιολόγο Sewell Wright (1931) ως μια παράσταση του χώρου όλων των δυνατών γονοτύπων μαζί με τις προσαρμογές τους [34]. Έστω ότι κάθε γονότυπο είναι μια ακολουθία l δυαδικών ψηφίων και ότι η απόσταση μεταξύ δύο γονοτύπων είναι η απόσταση Hamming. Επίσης, έστω ότι σε κάθε γονότυπο αντιστοιχεί μια πραγματική τιμή προσαρμογής. Ένα τοπίο προσαρμογής μπορεί να απεικονισθεί ως ένα διάγραμμα διαστάσεως $l+1$ στο οποίο κάθε γονότυπο είναι ένα σημείο στο χώρο των l διαστάσεων και η προσαρμογή (συνάρτηση συμπεριφοράς) σχεδιάζεται κατά μήκος του $l+1$ άξονα. Ένα απλό τοπίο προσαρμογής για $l=2$ δίνεται στο σχήμα 3.4.



Σχήμα 3.4: Ένα απλό τοπίο προσαρμογής με $l=2$

Τα διαγράμματα αυτά ονομάσθηκαν τοπία προσαρμογής γιατί μπορεί να περιέχουν κορυφές, οροπέδια, κοιλάδες, χαράδρες και άλλα χαρακτηριστικά των φυσικών τοπίων. Σύμφωνα με τη διατύπωση του Wright, η εξέλιξη οδηγεί τους πληθυσμούς σε κίνηση μέσα στα τοπία, με ειδικούς τρόπους και προσαρμοστικότητα, έτσι ώστε να μετακινούνται και να καταλήγουν σε τοπικές κορυφές.

Αντίστοιχα, στους γενετικούς αλγορίθμους οι τελεστές της διασταύρωσης και μετάλλαξης μπορούν να θεωρηθούν ως τρόποι μετακίνησης ενός πληθυσμού μέσα σε ένα τοπίο οριζόμενο με τη συνάρτηση προσαρμογής (*fitness function*).

Ένας γενετικός αλγόριθμος είναι μια μέθοδος αναζήτησης στο χώρο αυτό (τοπίο προσαρμογής) για τον προσδιορισμό ακολουθίας ή ακολουθιών μεγάλης προσαρμογής. Όπως ήδη φαίνεται από τα παραπάνω, το τοπίο προσαρμογής είναι αντίστοιχο του μηχανισμού της φυσικής επιλογής όπου τα ισχυρότερα άτομα έχουν μεγαλύτερη πιθανότητα να είναι οι νικητές σε ένα ανταγωνιστικό περιβάλλον. Μέσω της γενετικής εξέλιξης, μπορεί να βρεθεί μια βέλτιστη λύση η οποία παριστάνεται από τον τελικό νικητή στο γενετικό παιγνίδι [31].

3.4 Βασικός Γενετικός Αλγόριθμος

Οι ΓΑ υποθέτουν ότι κάθε υποψήφια λύση κάποιου προβλήματος είναι ένα άτομο το οποίο μπορεί να παρασταθεί με ένα σύνολο παραμέτρων. Οι παράμετροι αυτές θεωρούνται ότι είναι τα γονίδια ενός χρωμοσώματος και μπορούν να δομηθούν ως ακολουθίες δυαδικών ψηφίων. Η τιμή προσαρμογής αντανακλά το πόσο «καλό» είναι το χρωμόσωμα για το πρόβλημα. Μέσω της γενετικής εξέλιξης, το πιο προσαρμοσμένο χρωμόσωμα (fitter chromosome) τείνει να παράγει υψηλής ποιότητας απόγονο που σημαίνει καλύτερη λύση στο πρόβλημα. Σε μια πρακτική εφαρμογή ΓΑ, πρέπει να εγκαταστήσουμε μια δεξαμενή χρωμοσωμάτων (pool), των οποίων η αρχική επιλογή είναι τυχαία. Το μέγεθος του πληθυσμού αυτού κυμαίνεται από πρόβλημα σε πρόβλημα. Σε κάθε κύκλο της γενετικής λειτουργίας (που ονομάζεται «εξελικτική διαδικασία»), παράγεται μια επόμενη γενιά από τα χρωμοσώματα της παρούσας γενιάς. Τούτο μπορεί να επιτευχθεί μόνο εάν μια ομάδα από αυτά τα χρωμοσώματα επιλέγεται με μια διαδικασία επιλογής. Τα γονίδια των γονέων ανακατεύονται και διασταυρώνονται για την παραγωγή απογόνων στην επόμενη γενιά. Αναμένεται ότι από αυτήν την εξελικτική διαδικασία το «καλύτερο» χρωμόσωμα θα παραγάγει μεγαλύτερο αριθμό απογόνων και συνεπώς έχει μεγαλύτερη πιθανότητα επιβίωσης στις ακόλουθες γενιές, απομιμούμενο το φυσικό μηχανισμό της επιβίωσης του ισχυρότερου προσαρμοστικά γονιδίου.

Η απλούστερη μορφή γενετικού αλγορίθμου περιλαμβάνει τρεις τελεστές που αναλύονται παρακάτω: Επιλογή (*selection*), διασταύρωση (*crossover*) και μετάλλαξη (*mutation*).

- **Επιλογή**

Ο τελεστής αυτός επιλέγει χρωμοσώματα στον πληθυσμό προς αναπαραγωγή. Όσο περισσότερο προσαρμόζεται το χρωμόσωμα (*fitter*), τόσο περισσότερο πιθανό είναι να επιλεγεί προς αναπαραγωγή πιο πολλές φορές.

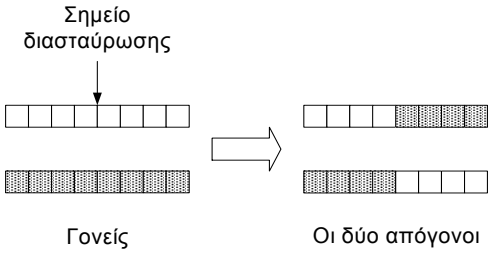
- **Διασταύρωση**

Ο τελεστής αυτός επιλέγει τυχαία μία θέση και ανταλλάσσει τις υπό–ακολουθίες πριν και μετά από αυτήν τη θέση μεταξύ δύο χρωμοσωμάτων για να παραγάγει δύο απογόνους.

Για παράδειγμα, οι ακολουθίες 1 0 0 0 0 1 0 0 και 1 1 1 1 1 1 1 1 μπορούν να διασταυρωθούν μετά την τρίτη θέση κάθε μιας για να δώσουν τους δύο απογόνους : 1 0 0 1 1 1 1 1 και 1 1 1 0 0 1 0 0. Ο τελεστής διασταύρωσης μιμείται χονδρικά το βιολογικό

ανασυνδυασμό μεταξύ δύο απλοειδών οργανισμών. Η διασταύρωση μπορεί να λάβει χώρα σε κάθε θέση του χρωμοσώματος με πιθανότητα που τυπικά έχει τιμή μεταξύ 0.6 και 1.0.

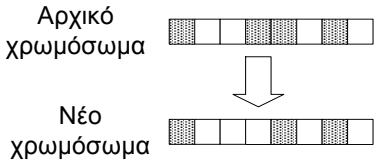
Ο μηχανισμός που περιγράφηκε απεικονίζεται στο Σχήμα 3.5.



Σχήμα 3.5: Διασταύρωση ενός σημείου (θέση 4)

- **Μετάλλαξη**

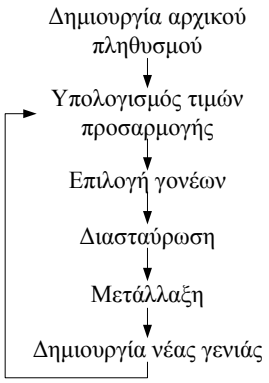
Ο τελεστής αυτός, που εφαρμόζεται μετά τη διασταύρωση, αλλάζει τυχαία μερικά από τα ψηφία ενός χρωμοσώματος. Για παράδειγμα, η ακολουθία 0 0 0 0 1 0 0 θα μπορούσε να μεταλλαχτεί στη δεύτερη θέση οπότε το αποτέλεσμα θα ήταν 0 1 0 0 0 1 0 0. Η μετάλλαξη μπορεί να λάβει χώρα σε κάθε θέση του χρωμοσώματος με κάποια πιθανότητα, που είναι συνήθως πολύ μικρή. Ο μηχανισμός που περιγράφηκε απεικονίζεται και στο παρακάτω σχήμα (βλ. Σχήμα 3.6).



Σχήμα 3.6: Μετάλλαξη ενός σημείου (θέση 5)

Ο εξελικτικός κύκλος επαναλαμβάνεται μέχρις ότου να ικανοποιηθεί κάποιο επιθυμητό κριτήριο τερματισμού. Το όριο αυτό μπορεί να είναι ο αριθμός των εξελικτικών κύκλων ή το πλήθος των μεταβολών των ατόμων ανάμεσα στις διάφορες γενιές, ή τέλος μια προκαθορισμένη τιμή της συνάρτησης προσαρμογής.

Ο βασικός ΓΑ περιλαμβάνει τα ακόλουθα βήματα που συνοψίζονται στο Σχήμα 3.7.



Σχήμα 3.7: Βασικός γενετικός αλγόριθμος

Ο παραπάνω ΓΑ είναι η βάση για τις περισσότερες εφαρμογές γενετικών αλγορίθμων. Βεβαίως, υπάρχουν πολλές λεπτομέρειες που πρέπει να συμπληρωθούν, όπως το μέγεθος του πληθυσμού, οι ρυθμοί διασταύρωσης και μετάλλαξης και το κριτήριο τερματισμού. Οι παραπάνω λεπτομέρειες καθορίζουν σε μεγάλο βαθμό την επιτυχία του ΓΑ.

3.5 Μέθοδοι επιλογής και γενετικών τελεστών

Στις παραγράφους που ακολουθούν αναλύονται οι μέθοδοι που χρησιμοποιούνται για την επιλογή των γονέων καθώς και οι μέθοδοι των γενετικών πράξεων.

3.5.1 Μέθοδοι επιλογής

Για την παραγωγή καλών απογόνων, είναι απαραίτητος ένας καλός μηχανισμός επιλογής των γονέων. Αυτό είναι μια διαδικασία που χρησιμοποιείται για να καθορισθεί ο αριθμός των δοκιμών για κάθε συγκεκριμένο άτομο που χρησιμοποιείται στην αναπαραγωγή. Η πιθανότητα επιλογής ενός χρωμοσώματος ως γονέα πρέπει να είναι ανάλογη του αριθμού των παραγόμενων απογόνων.

Τρία κριτήρια μέτρησης της συμπεριφοράς των αλγορίθμων επιλογής γονέων που αναλύονται παρακάτω είναι η *πόλωση*, η *εξάπλωση* και η *απόδοση*.

- **Πόλωση**

Η πόλωση καθορίζει την απόλυτη διαφορά, ανάμεσα στα άτομα, της πραγματικής και αναμενόμενης πιθανότητας επιλογής. Η βέλτιστη (μηδενική) πόλωση επιτυγχάνεται όταν η πιθανότητα ενός ατόμου είναι ίση με τον αναμενόμενο αριθμό δοκιμών.

- **Εξάπλωση**

Η εξάπλωση είναι μια περιοχή στον πιθανό αριθμό δοκιμών που μπορεί να επιτύχει ένα άτομο. Εάν $g(i)$ είναι ο πραγματικός αριθμός δοκιμών που οφείλονται στο άτομο i, τότε η «ελάχιστη εξάπλωση» είναι η μικρότερη εξάπλωση που θεωρητικά επιτρέπει μηδενική πόλωση, δηλαδή:

$$g(i) \in \left\{ \underline{\alpha\delta}(i), \overline{\alpha\delta}(i) \right\} \tag{3.1}$$

όπου $\alpha\delta$ είναι ο αναμενόμενος αριθμός δοκιμών του ατόμου i, $\underline{\alpha\delta}(i)$ είναι το κάτω όριο και $\overline{\alpha\delta}(i)$ είναι το πάνω όριο του $\alpha\delta$. Συνεπώς η εξάπλωση μιας επιλογής είναι ένα μέτρο της συνέπειάς της.

- **Απόδοση**

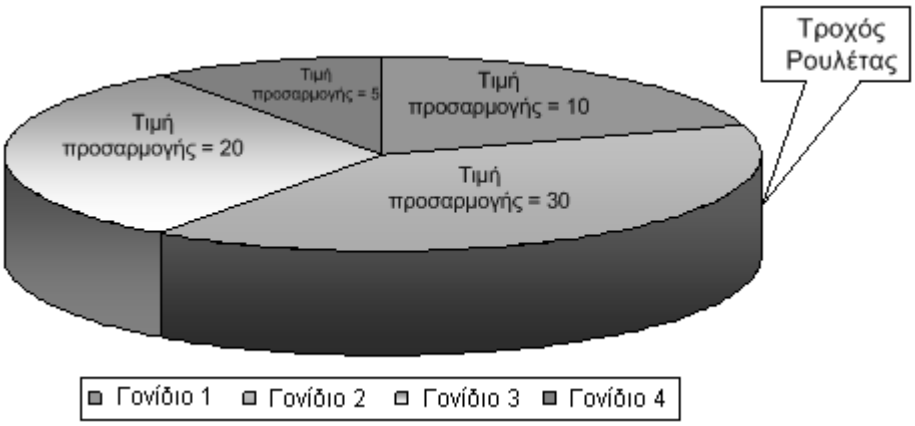
Η απόδοση συνδέεται με τον ολικό απαιτούμενο χρόνο (πολυπλοκότητα) του αλγορίθμου επιλογής.

Γενικά, θέλουμε ο αλγόριθμος επιλογής να πετυχαίνει μηδενική πόλωση και ταυτόχρονα να διατηρεί ελάχιστη εξάπλωση και να μην οδηγεί σε αυξημένη πολυπλοκότητα του ΓΑ.

Οι τρεις βασικότεροι μηχανισμοί επιλογής είναι:

- Επιλογή με τον τροχό ρουλέτα (τροχός της τύχης), ο οποίος αναλύεται παρακάτω, αφού αποτελεί και τον πιο συχνά χρησιμοποιούμενο μηχανισμό στους γενετικούς αλγορίθμους.
- Στοχαστική δειγματοληψία με μερική αντικατάσταση, κατά την οποία κάθε φορά που επιλέγεται ένα χρωμόσωμα, το μέγεθος του τμήματός του μειώνεται κατά έναν παράγοντα.
- Καθολική στοχαστική δειγματοληψία

Όπως προαναφέρθηκε, ένα από τα βασικότερα σχήματα επιλογής γονέων είναι ο μηχανισμός τροχού ρουλέτας. Εδώ οι τιμές προσαρμογής των χρωμοσωμάτων αντιπροσωπεύουν και καθορίζουν το πλάτος της περιφέρειας που καταλαμβάνει το συγκεκριμένο χρωμόσωμα στη ρουλέτα. Όσο πιο μεγάλη η τιμή προσαρμογής, τόσο πιο μεγάλη η περιφέρεια που καταλαμβάνει το χρωμόσωμα στη ρουλέτα και άρα τόσο πιο μεγάλη η πιθανότητα που έχει να επιλεγεί ως γονέας, όπως φαίνεται και στο Σχήμα 3.8 [1], [31].



Σχήμα 3.8: Επιλογή γονέων με το μηχανισμό ρουλέτας

Ο αλγόριθμος επιλογής σύμφωνα με αυτό το μηχανισμό περιγράφεται στο Σχήμα 3.9.

```
Άθροισε την προσαρμογή όλων των μελών του πληθυσμού για να βρεις την
ολική προσαρμογή
while δεν έχει επιλεγεί ο απαιτούμενος αριθμός γονέων
    {
    Βρες ένα τυχαίο αριθμό r ϵ [0,1].
    Πολλαπλασίασε το άθροισμα της ολικής προσαρμογής με τον τυχαίο
    αριθμό και αποθήκευσε τη ποσότητα αυτή ως Α.
    Επέστρεψε το πρώτο μέλος του πληθυσμού του οποίου η προσαρμογή,
    προστιθέμενη στις προσαρμογές των προηγούμενων μελών του
    πληθυσμού, ξεπερνά την ποσότητα Α.
    }
```

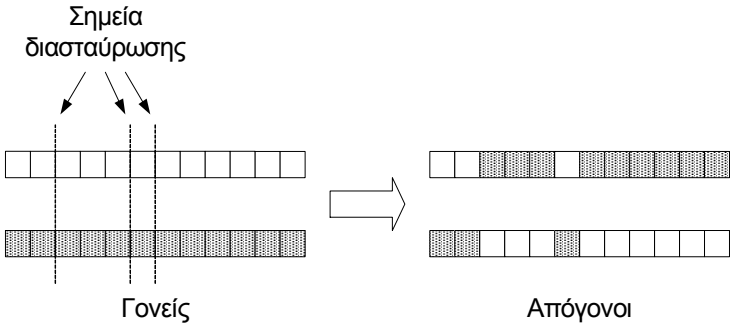
Σχήμα 3.9: Αλγόριθμος μηχανισμού ρουλέτας

3.5.2 Μέθοδοι Γενετικών Πράξεων

Στις παρακάτω παραγράφους αναλύονται οι μέθοδοι διασταύρωσης και μετάλλαξης των γενετικών πράξεων.

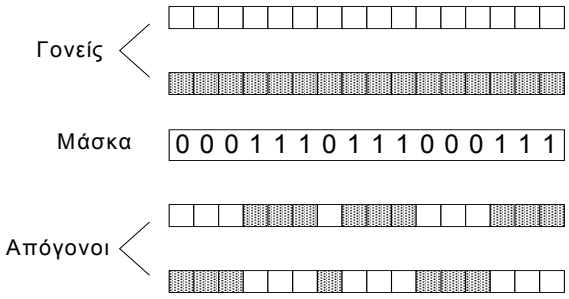
3.5.2.1 Μέθοδοι Διασταύρωσης

Αν και η διασταύρωση ενός σημείου εμπνεύσθηκε από τις βιολογικές διαδικασίες, έχει το μειονέκτημα ότι κάποιοι συνδυασμοί σχημάτων σε ορισμένες περιπτώσεις δεν μπορούν να συνδυασθούν. Γι' αυτό χρησιμοποιείται διασταύρωση πολλαπλού σημείου, όπως φαίνεται στο Σχήμα 3.10.



Σχήμα 3.10: Διασταύρωση πολλαπλού σημείου

Ένας άλλος τρόπος είναι η ομοιόμορφη διασταύρωση. Η μέθοδος αυτή παράγει απογόνους από τους γονείς, με βάση μια τυχαία παραγόμενη μάσκα διασταύρωσης, όπως παρουσιάζεται στο Σχήμα 3.11.



Σχήμα 3.11: Ομοιόμορφη διασταύρωση

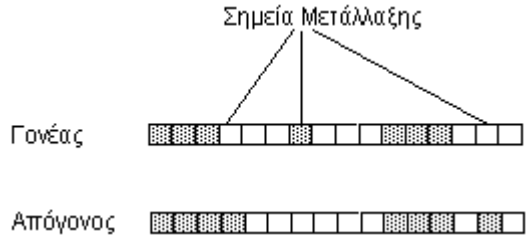
Οι παραγόμενοι απόγονοι περιέχουν ένα μίγμα γονιδίων από κάθε γονέα. Το πλήθος των αποδοτικών σημείων διασταύρωσης δεν είναι σταθερό, αλλά έχει μέση τιμή $L/2$ όπου L είναι το μήκος του χρωμοσώματος.

Το ζήτημα του ποια μέθοδος διασταύρωσης είναι προτιμητέα είναι ακόμη υπό συζήτηση. Γενικά έχει επαληθευθεί ότι η διασταύρωση δύο θέσεων είναι η καλύτερη από όλες τις άλλες διασταυρώσεις πολλαπλής θέσης. Επειδή η ομοιόμορφη διασταύρωση ανταλλάσσει δυαδικά ψηφία παρά τμήματα, μπορεί να συνδυάζει ιδιότητες ανεξάρτητα από τη σχετική τους θέση. Τέλος, σημειώνεται ότι πράξεις διασταύρωσης μπορούν να υιοθετηθούν και σε

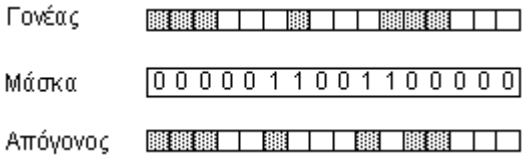
ένα χρωμόσωμα με παράσταση πραγματικών αριθμών. Η μόνη διαφορά είναι ότι η ακολουθία συγκροτείται από πραγματικούς και όχι δυαδικούς αριθμούς [31], [33].

3.5.2.2 Μέθοδοι Μετάλλαξης

Πέρα από τη μετάλλαξη ενός σημείου, συναντάμε ακόμη τη μετάλλαξη πολλαπλού σημείου αλλά και την ομοιόμορφη μετάλλαξη. Στα παρακάτω σχήματα (βλ. Σχήμα 3.12 και Σχήμα 3.13) απεικονίζονται οι δύο αυτοί μέθοδοι μετάλλαξης.



Σχήμα 3.12: Μετάλλαξη πολλαπλού σημείου



Σχήμα 3.13: Ομοιόμορφη μετάλλαξη

Αρχικά η μετάλλαξη σχεδιάσθηκε μόνο για χρωμοσώματα με δυαδική παράσταση. Για να εισαχθούν όμως μεταβολές στο χρωμόσωμα που παριστάνεται με πραγματικούς αριθμούς, έχει επίσης σχεδιασθεί μια τυχαία μετάλλαξη, δηλαδη:

$$g = g + \Gamma\left(m, \sigma\right) \tag{3.2}$$

όπου g είναι το γονίδιο πραγματικής τιμής και $\Gamma(m,\sigma)$ είναι μια Γκαουσιανή (*Gaussian*) κατανομή με μέση τιμή m και τυπική απόκλιση σ.

3.6 Εφαρμογές Γενετικών Αλγορίθμων

Οι γενετικοί αλγόριθμοι με διάφορες παραλλαγές και τροποποιήσεις έχουν εφαρμοστεί σε πλήθος πρακτικών εφαρμογών. Μερικές από αυτές είναι οι ακόλουθες [31]:

- **Βελτιστοποίηση**

Προβλήματα βελτιστοποίησης που περιλαμβάνουν αριθμητική βελτιστοποίηση και συνδυαστική βελτιστοποίηση σε εφαρμογές όπως σχεδίαση κυκλωμάτων, χρονοπρογραμματισμός εργασιών, οργάνωσης και λειτουργίας αποθηκών, κλπ.

- **Αυτόματος προγραμματισμός**

Προβλήματα εξέλιξης υπολογιστικών προγραμμάτων για ειδικές εργασίες και σχεδίαση άλλων υπολογιστικών δομών όπως κυτταρικά αυτόματα και δίκτυα κατάταξης.

- **Μάθηση μηχανής**

Προβλήματα μηχανικής μάθησης (*machine learning*) για εργασίες όπως ταξινόμηση, σύντηξη σημάτων, εκτίμηση και πρόβλεψη, λ.χ. πρόβλεψη καιρού, σεισμού, ζήτησης αγαθών, καθορισμού τιμών προϊόντων, πρωτεϊνικής δομής, κλπ. Επίσης προβλήματα βέλτιστης επιλογής παραμέτρων βιομηχανικών ελεγκτών, ρομποτικών συστημάτων, νευρωνικών δικτύων, ασαφών συστημάτων, κοκ.

- **Οικονομικά Συστήματα**

Προβλήματα μοντελοποίησης οικονομικών διεργασιών, διεργασιών καινοτομίας, τιμολογιακών διεργασιών, διεργασιών αγοράς, κοκ.

- **Βιολογικά συστήματα**

Σπουδή προβλημάτων γενετικής των πληθυσμών, εξέλιξης των ειδών και μάθησης των ατόμων.

- **Οικολογικά συστήματα**

Προβλήματα μοντελοποίησης οικολογικών φαινομένων, όπως ροή φυσικών πηγών, συμβίωση, συνανάπτυξη παρασίτων, κοκ.

- **Κοινωνικά συστήματα**

Προβλήματα κοινωνικής συμπεριφοράς, συνεργασίας και επικοινωνίας σε πολύ-πρακτορικά συστήματα (*multi-agent systems*), κοκ.

- **Τεχνολογικά συστήματα**

Συστήματα οργάνωσης, επίβλεψης και ελέγχου βιομηχανικών συστημάτων συνεχούς και διακριτής παραγωγής, ρομποτικά συστήματα κατασκευής και υπηρεσιών (σχεδιασμός τροχιών και καθηκόντων, παρακολούθηση δρόμου), συστήματα παραγωγής και διανομής ενέργειας, αεροναυπηγικά συστήματα, συστήματα μεταφορών και επικοινωνιών, κλπ.

Κεφάλαιο 4

Αυτοματοποίηση Ηλεκτρονικής Σχεδίασης

4.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζεται μια εισαγωγή στην αυτοματοποίηση ηλεκτρονικής σχεδίασης (*Electronic Design Automation* – EDA) που εφαρμόζεται στον σχεδιασμό ολοκληρωμένων κυκλωμάτων προγραμματιζόμενης λογικής (*Field Programmable Gate Array* – FPGA) και μη (*Full ή Semi Application Specific Integrated Circuits* – ASIC) με τη χρήση αυτής.

4.2 Ιστορική Αναδρομή

Αυτοματοποίηση Ηλεκτρονικής Σχεδίασης (*Electronic Design Automation* - EDA), είναι η κατηγορία εργαλείων για τα ηλεκτρονικά συστήματα από τις τυπωμένες πλακέτες κυκλωμάτων (*PCBs*) ως τα ολοκληρωμένα κυκλώματα. Μερικές φορές αναφέρετε και ως ECAD (*Electronic Computer-aided Design*) ή απλά CAD (*Computer-aided Design*). Ο όρος EDA χρησιμοποιείται επίσης ως ένας όρος που εμπεριέχει τούς όρους computer-aided engineering, computer-aided design και computer-aided manufacturing για τα ηλεκτρονικά στον τομέα των Ηλεκτρολόγων Μηχανικών. Η χρήση του όρου EDA πιθανότατα προέρχεται από το τεχνικό επιμελητήριο IEEE Design Automation.

Τα εργαλεία EDA τα τελευταία χρόνια έχουν αναπτυχθεί πολύ λόγω της συνεχούς προόδου στην περιοχή της τεχνολογίας των ημιαγωγών. Πριν από την EDA τα ολοκληρωμένα κυκλώματα σχεδιάζονταν και γίνονταν lay out με το χέρι . Μέχρι τα μέσα της δεκαετίας του '70, οι υπεύθυνοι για την ανάπτυξη άρχιζαν να αυτοματοποιούν το σχέδιο. Τα πρώτα εργαλεία θέσης και δρομολόγησης (*Place and Route*) αναπτύχθηκαν. Τα πρακτικά συνεδρίων του Design Automation Conference (DAC) καλύπτει ένα μεγάλο μέρος αυτής

της χαρακτηριστικής περιόδου. Η επόμενη περίοδος/εποχή ουσιαστικά ξεκίνησε με το βιβλίο "Introduction to VLSI Systems" των Carver Mead και Lynn Conway το 1980 [35]. Η συγκεκριμένη δημοσίευση εισάγει τη σχεδίαση ολοκληρωμένων κυκλωμάτων με γλώσσες προγραμματισμού που με τη βοήθεια ενός μεταγλωττιστή μετατρέπονται σε κύκλωμα πάνω σε πυρίτιο. Αυτό είχε ως άμεσο αποτέλεσμα, μία εκατονταπλή αύξηση στην πολυπλοκότητα, του ως προς σχεδίαση ολοκληρωμένου κυκλώματος και συνεπώς ευκολότερη πρόσβαση σε εργαλεία σχεδιαστικής επαλήθευσης (*design verification tools*) που χρησιμοποιούν προσομοίωση δυαδικής λογικής (*logic simulation*). Επιπλέον τα ολοκληρωμένα κυκλώματα εκτός ότι είναι πιο εύκολο να μετατραπούν σε *σχέδιο*[*] (*layout*), είναι και πιο σωστά εν γένει, διότι ο κυκλωματικός τους σχεδιασμός μπορεί να προσομοιωθεί πολύ πιο εκτενώς και να διορθωθούν πιθανά λάθη πριν την κατασκευή τους. Το σχέδιο δημιουργείται με τη μετατροπή των λογικών στοιχείων (*logic component*), όπως transistor, πύλες, στοιχεία μνήμης κλπ. σε γεωμετρικά σχήματα σε πολλαπλά επίπεδα (*layers*) τα οποία εκτελούν τη λογική συνάρτηση που προορίζονταν για το αντίστοιχο λογικό στοιχείο. Συνδέσεις μεταξύ διαφορετικών στοιχείων εκφράζονται στο γεωμετρικό πρότυπο από γραμμές σε διάφορα επίπεδα. Οι ακριβείς λεπτομέρειες του σχεδίου καθορίζονται και από περιορισμούς στη διαδικασία κατασκευής και ηλεκτρικές ιδιότητες των υλικών κατασκευής.

Τα πρώτα EDA εργαλεία αναπτύχθηκαν μέσα από την ακαδημία και δεν απαιτούσαν πνευματικά δικαιώματα. Ένα από τα πιο γνωστά ήταν το "Berkley VLSI Tools Tarball" [36] που αποτελείτε από εφαρμογές σε UNIX περιβάλλον για τη σχεδίαση συστημάτων Πολύ Υψηλής Κλίμακας Ολοκλήρωσης (*Very Large Scale Integration - VLSI*). Μέχρι και σήμερα χρησιμοποιείται ευρέως το "ESPRESSO heuristic logic minimizer" [37], το οποίο βοηθάει στη μείωση της πολυπλοκότητας ψηφιακών ηλεκτρονικών κυκλωμάτων με πύλες.

Το 1981 σηματοδοτεί την αρχή των EDA εργαλείων σαν βιομηχανία. Για αρκετά χρόνια οι μεγαλύτερες εταιρίες ηλεκτρονικών όπως η Hewlett Packard, Tektronix και η Intel κρατάγανε τα εργαλεία EDA για δικιά τους εσωτερική χρήση. Το 1981 κάποιοι διευθυντές και προγραμματιστές των εταιριών αυτών αποχωρούν για να ασχοληθούν αποκλειστικά με την ανάπτυξη εργαλείων EDA και παράλληλα δημιουργούνται εταιρίες όπως η Daisy Systems, Mentor Graphics, Valid Logic Systems που είναι συχνά γνωστές και ως DMV. Μέσα στα επόμενα χρόνια ακολουθούν και άλλες παρόμοιες εταιρίες που ειδικεύονται στην ανάπτυξη EDA εργαλείων, με πολύ μικρή διαφορά στην έμφαση η καθεμιά.

Το 1981 το Υπουργείο Εθνικής Άμυνας των Ηνωμένων Πολιτειών χρηματοδότησε τη δημιουργία της γλώσσας VHDL (*Very-High-Speed Integrated Circuits - VHSIC Hardware Description Language - HDL*) για να αντιμετωπίσει την κρίση στον κύκλο ζωής του υλικού (*hardware*) και πιο συγκεκριμένα των ASIC ολοκληρωμένων που χρησιμοποιούσαν οι εταιρίες στα προϊόντα τους. Το κόστος επαναεπεξεργασίας του υλικού καθώς διάφορες τεχνολογίες σταδιακά κρίνονταν απαρχαιωμένες ήταν υψηλό, διότι η λειτουργία των τμημάτων δεν ήταν κατάλληλα τεκμηριωμένη και οι διάφορες συνιστώσες (*components*) που αποτελούσαν το σύστημα ελέγχονταν ξεχωριστά χρησιμοποιώντας ένα μεγάλο εύρος από διαφορετικές, μη συμβατές γλώσσες προσομοίωσης και εργαλεία. Η απαίτηση ήταν

[*] μια *γεωμετρική* (geometric) *αναπαράσταση* του κυκλώματος ή οποία καλείται *σχέδιο* (layout).

για μια γλώσσα με ένα μεγάλο εύρος περιγραφικής δυνατότητας που θα μπορούσε να δουλέψει με τον ίδιο τρόπο σε οποιονδήποτε προσομοιωτή και θα ήταν ανεξάρτητη από την τεχνολογία ή τη μεθοδολογία σχεδίασης. Η VHDL δανείζεται πολλά προγραμματιστικά στοιχεία (έννοιες και σύνταξη) από τη γλώσσα προγραμματισμού ADA.

Η διαδικασία τυποποίησης της VHDL επιτεύχθηκε αρκετά γρήγορα λόγω της συμμετοχής της βιομηχανίας. Η βασική έκδοση της γλώσσας (7.2) δημοσιεύτηκε 2 χρόνια πριν το επίσημο πρότυπο ώστε η να αρχίσει η ανάπτυξη εργαλείων. Όλα τα δικαιώματα για τον ορισμό της γλώσσας δόθηκαν στην IEEE προκειμένου να ενθαρρυνθεί η αποδοχή της γλώσσας από τη βιομηχανία. Ως πρότυπο της IEEE, η VHDL πρέπει να υποβάλλεται σε διαδικασία επιθεώρησης κάθε 5 έτη (ή πιο σύντομα) για να διασφαλιστεί η τρέχουσα σχετικότητα της με τη βιομηχανία. Η πρώτη τέτοια αναθεώρηση ολοκληρώθηκε το Σεπτέμβριο του 1993. Το πρότυπο ονομάστηκε 1076-1993 [38], [39].

Μεταξύ 1983 και 1984 οι Phil Moorby και Prabhu Goel της εταιρίας Automated Integrated Design Systems (αργότερα το 1985 μετονομάστηκε σε Gateway Design Automation και το 1990 εξαγοράστηκε από την Cadence Design Systems) παρουσίασαν μια γλώσσα περιγραφής υλικού (*Hardware Description Language - HDL*) που την ονόμασαν Verilog [40].

Η σύσταση αυτών των γλωσσών επέτρεψε τη γρήγορη ανάπτυξη εργαλείων προσομοίωσης που επιτρέπουν την απευθείας προσομοίωση σχεδίων ολοκληρωμένων κυκλωμάτων. Στα επόμενα χρόνια αναπτύχθηκαν επιπρόσθετα εργαλεία που επιτρέπουν τη σύνθεση λογικών κυκλωμάτων (*logic synthesis*) από μια γλώσσα περιγραφής υλικού.

Η σημερινή ροή του σχεδιασμού ψηφιακών κυκλωμάτων είναι υπερβολικά αρθρωτή υπό την έννοια ότι τα εργαλεία EDA παράγουν τυποποιημένες σχεδιαστικές περιγραφές που μεταγλωττίζονται σε ένα δίκτυο κυττάρων (*cells*) χωρίς ιδιαίτερη σημασία να δίνεται στην τεχνολογία του κελιού. Τα κελιά υλοποιούν τη λογική χρησιμοποιώντας μια συγκεκριμένη τεχνολογία ολοκληρωμένων κυκλωμάτων. Οι κατασκευάστριες εταιρίες ολοκληρωμένων κυκλωμάτων παρέχουν βιβλιοθήκες με στοιχεία για τη δικιά τους διαδικασία παραγωγής, και μοντέλα προσομοίωσης τα οποία μπορούν φορτωθούν από τα εργαλεία προσομοίωσης.

4.3 Περιοχές Εφαρμογών των Εργαλείων EDA

Η Αυτοματοποίηση Ηλεκτρονικής Σχεδίασης διαιρείται σε πολλές υποκατηγορίες όπου σε κάποιες περιπτώσεις ενδέχεται η μια να υπερκαλύπτει την άλλη. Η σειρά για την κατασκευή του ολοκληρωμένου κυκλώματος συνήθως ξεκινάει από τη σχεδίαση μέχρι τη δημιουργία της μάσκας.

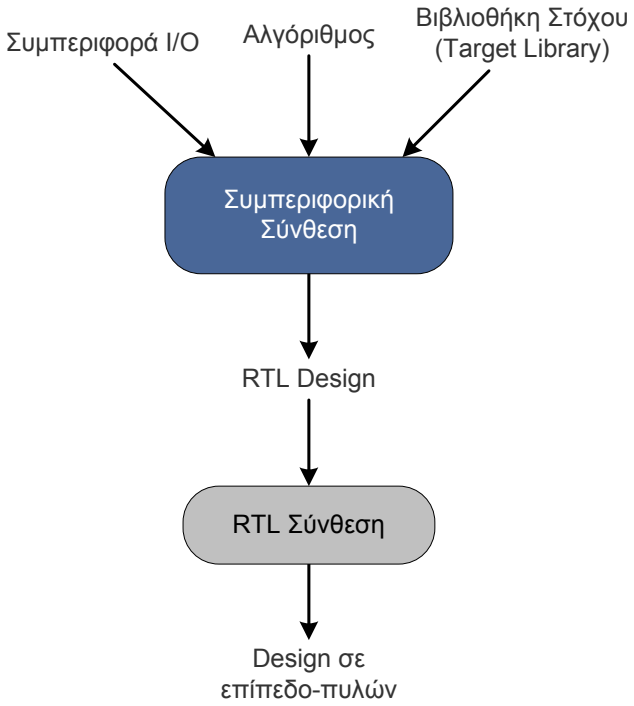
Οι παρακάτω υποκατηγορίες ισχύουν για τα ολοκληρωμένα κυκλώματα ASIC (*Application Specific Integrated Circuits*) και τα ψηφιακά κυκλώματα προγραμματιζόμενης λογικής (*Field Programmable Gate Array – FPGA*), αλλά έχουν πολλά κοινά χαρακτηριστικά και με τη σχεδίαση τυπωμένων πλακετών κυκλωμάτων (*Printed Circuit Board – PCB*).

- **Σχεδίαση και Αρχιτεκτονική**

Σχεδίαση και αρχιτεκτονική με τη χρήση VHDL, Verilog ή άλλης γλώσσας περιγραφής υλικού (*Hardware Description Language – HDL*).

- **Συμπεριφορική Σύνθεση (*Behavioral Synthesis*)**

Η Σύνθεση Υψηλού Επιπέδου (*High Level Synthesis*) που συχνά αναφέρεται και ως Συμπεριφορική Σύνθεση (*Behavioral Synthesis*) ή Αλγοριθμική Σύνθεση (*Algorithmic Synthesis*) αυξάνει την αφαιρετικότητα της περιγραφής σχεδιασμού και επιτρέπει την αυτοματοποίηση της διαδικασίας διερεύνησης της αρχιτεκτονικής. Περιλαμβάνει τη διαδικασία μετάφρασης από μία αφαιρετική συμπεριφορική περιγραφή ενός design, σε μια συνθέσιμη περιγραφή επιπέδου μεταφοράς καταχωρητών (*synthesizeable Register Transfer Level – RTL description*). Η συμπεριφορικη περιγραφή μπορεί να δοθεί σε συμπεριφορική VHDL, αλγοριθμική SystemC, C++ και το αποτέλεσμα της σύνθεσης να παράγει μια RTL περιγραφή σε VHDL ή Verilog (Σχήμα 4.1).



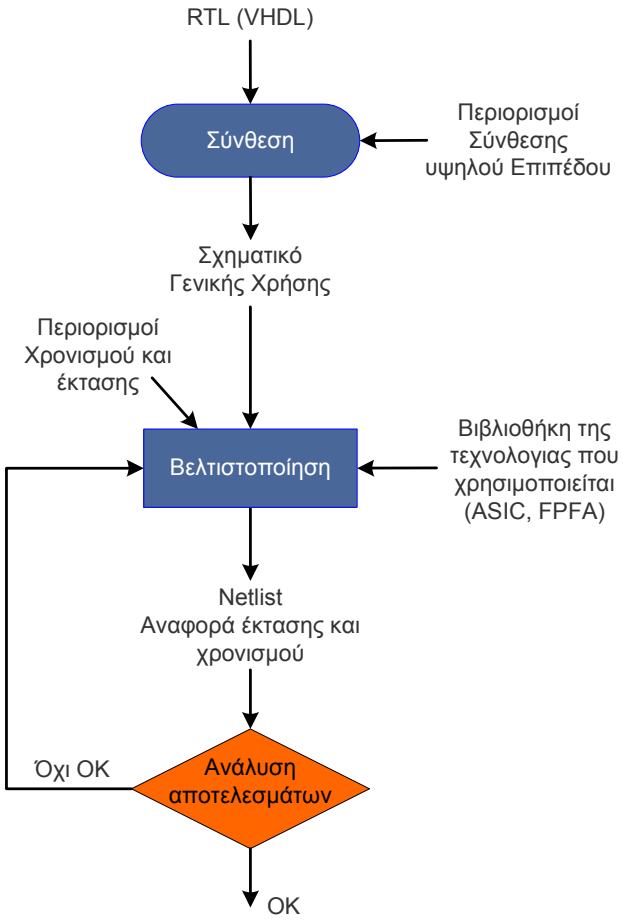
Σχήμα 4.1: Διάγραμμα συμπεριφορικής λογικής

- **Λογική Σύνθεση (*Logic Synthesis*)**

Η μετάφραση της αφαιρετικής λογικής RTL (*Register Transfer Level* ή *Επίπεδο Μεταφοράς Καταχωρητών*) – περιγραφής ενός chip σε ένα ξεχωριστό δικτύωμα (*netlist*[*]) αποτελούμενο από στοιχεία λογικών πυλών (η αλλιώς *Boolean* λογικής). Συνήθως η RTL περιγραφή δημιουργείται με μια γλώσσα HDL όπως VHDL ή Verilog.

[*] Μια Boolean-αλγεβρική αναπαράσταση (πύλες, στάνταρ κύτταρα – standard cells) που περιλαμβάνει πληροφορίες διασυνδέσεων, προκείμενα (instances), δίκτυα και χαρακτηριστικά. Τα netlists διακρίνονται σε *physical* ή *logical*, σε *instance-βασισμένα* ή *net-βασισμένα* και σε *επίπεδα (flat)* ή *ιεραρχικά* που μπορεί να είναι *folded* ή *unfolded.*

Στο Σχήμα 4.2 φαίνεται το διάγραμμα ροής για τη λογική σύνθεση μιας RTL περιγραφής (π.χ. σε VHDL) και τη διαδικασία βελτιστοποίησης.



Σχήμα 4.2: Διάγραμμα σύνθεσης λογικής και βελτιστοποίησης

- **Χωροθέτηση (*Floorplanning*)**

Το βήμα για την προετοιμασία της δημιουργίας ενός βασικού χάρτη στην επιφάνεια του πυριτίου (die-map) που θα δείχνει τις αναμενόμενες θέσεις για τις λογικές πύλες, των power και ground πλάνων (planes), τη διανομή των ρολογιών, των pads εισόδων/εξόδων (I/O) και των αυστηρών μακροεντολών (hard macros).

- **IP cores**

Τα IP (Intellectual Property) cores είναι επαναχρησιμοποιήσιμες δομικές μονάδες λογικής που υλοποιούν διάφορες λειτουργίες σύμφωνα με την εφαρμογή για την οποία προορίζονται. Συνήθως διατίθενται από τις εταιρίες σε κάποια συνθέσιμη γλώσσα περιγραφής υλικού (VHDL, Verilog, Handel-C) ή σε χωροδικτυώματα (*netlists*) γενικής χρήσης ή σε τρανζίστορ layout format και ονομάζονται σκληροί πυρήνες (hard cores) αφού δεν δίνεται η δυνατότητα να τροποποιηθούν από τον τελικό χρήστη. Τα συνθέσιμα IP cores που προσφέρονται σε γλώσσα περιγραφής υλικού προσφέρουν το πλεονέκτημα ότι επιτρέπουν την τροποποίηση τους σε λειτουργικό επίπεδο. Τα netlists καθώς και τα συνθέσιμα IP cores ονομάζονται και μαλακοί πυρήνες (soft cores) καθώς και τα δύο ακολουθούν τη ροή σχεδίασης, σύνθεση, θέση και δρομολόγηση (synthesis, placement and

route – SPR). Τα IP cores παίζουν πολύ σημαντικό ρόλο στη σχεδίαση συστημάτων-σε-ψηφίδα (SoC – System on a Chip).

- **Προσομοίωση (*Simulation*):**

Προσομοίωση της λειτουργίας ενός κυκλώματος ούτως ώστε να ελεγχθεί η ορθότητα και η απόδοση του.

  - **Προσομοίωση σε επίπεδο Transistor (*Transistor Simulation*):** Χαμηλού επιπέδου (τρανζίστορ) προσομοίωση της συμπεριφοράς του σχηματικού/layout, που προσφέρει ακρίβεια σε επίπεδο ψηφίδας.
  - **Προσομοίωση Λογικής (*Logic Simulation*):** Ψηφιακή προσομοίωση συμπεριφοράς μιας RTL περιγραφής ή ενός netlist, που προσφέρει ακρίβεια σε δυαδικό (Boolean) επίπεδο.
  - **Προσομοίωση Συμπεριφοράς (*Behavioral Simulation*):** Υψηλού επιπέδου προσομοίωση της αρχιτεκτονικής λειτουργίας μιας σχεδίασης, που προσφέρει ακρίβεια σε επίπεδο κύκλων ρολογιού (*cycle-level*) ή μέσου αλληλεπίδρασης (*interface*).
  - **Εξομοίωση Υλικού (*Hardware Emulation*):** Γίνεται χρήση ειδικού σκοπού υλικό για την εξομοίωση της σχεδίασης. Υπάρχει περίπτωση να τοποθετηθεί σε ένα σύστημα του οποίου το chip δεν έχει ακόμα δημιουργηθεί. Η τελευταία περίπτωση καλείται εξομοίωση μέσα-στο-κύκλωμα (*in-circuit emulation*).

- **Επαλήθευση Μεθόδου (*Formal Verification*) και Έλεγχος Μοντέλου (*Model Checking*):**

Με τη βοήθεια μαθηματικών μεθόδων επιχειρείται η απόδειξη συγκεκριμένων επιθυμητών χαρακτηριστικών της σχεδίασης όπως επίσης ότι κάποια ανεπιθύμητα αίτια (*deadlock*[*]) μπορεί να συμβούν.

- **Επαλήθευση Διασταύρωσης Πεδίου Ορισμού Ρολογιού (*Clock Domain Crossing Verification – CDC check*):**

Έλεγχος ούτως ώστε να ανιχνευτούν σημαντικά θέματα σχεδιασμού που μπορεί να προκαλέσουν χάσιμο δεδομένων, όπως το πρόβλημα της μετά-σταθερότητας (*meta-stability*) λόγω της χρήσης πολλαπλών πεδίων ρολογιού (*clock domains*) στον σχεδιασμό.

- **Έλεγχος Ισοδυναμίας (*Equivalence Checking*):**

Αλγοριθμική σύγκριση μεταξύ της RTL περιγραφής μίας ψηφίδας και ενός συνθέσιμου netlist-πυλών έτσι ώστε να βεβαιωθεί η λειτουργική τους ισοδυναμία σε επίπεδο λογικής.

- **Ανάλυση Ισχύος και Βελτιστοποίηση (*Power Analysis and Optimization*):**

Βελτιστοποίηση του κυκλώματος για τη μείωση της καταναλισκόμενης ισχύος που απαιτείται για τη λειτουργία του χωρίς η μείωση αυτή να αλλάζει ή να αλλοιώνει τη λειτουργικότητα του.

---

[*] Πρόκειται για μια ξεχωριστή περίπτωση όπου δυο οι περισσότερες διαδικασίες (processes) περιμένουν η μια την άλλη να ελευθερώσει κάποιον πόρο του συστήματος ή περιμένουν για διαθέσιμους πόρους σε κυκλική αλυσίδα

- **Θέση και Δρομολόγηση (*Place and Route*):**

Αρχικά το εργαλείο θέσης εκτελεί την αυτόματη τοποθέτηση των λογικών στοιχείων από τη netlist περιγραφή, στις επαναπρογραμματίζόμενες δομικές μονάδες της τεχνολογίας FPGA που χρησιμοποιείται. Μετά επιχειρείται μια διαδοχική διασύνδεση των δομικών μονάδων από το εργαλείο δρομολόγησης που επιλέγει τις διαδρομές (*wires*) που θα ενώσει τα ακραία τμήματα (terminals) των μονάδων (τα τμήματα σημάτων και τροφοδοσίας).

- **Ανάλυση Στατικού Χρονισμού (*Static Timing Analysis*):**

Ανάλυση του χρονισμού του κυκλώματος με τρόπο ανεξάρτητης εισόδου, τέτοια ώστε να βρεθεί η χειρότερη περίπτωση (*worst case*) μέσα από όλες τις πιθανές εισόδους.

- **Πλάνο Transistor (*Transistor Layout*):** ¨

Έχει εφαρμογή σε αναλογικές ή ανάμεικτες (αναλογικές/ψηφιακές) ψηφίδες. Το έτοιμο σχηματικό μετατρέπεται σε χάρτη layout που υποδεικνύει όλα τα στρώματα της ψηφίδας. Αναφέρεται και ως *polygon pushing*.

- **Σχεδίαση που προορίζεται για κατασκευή (*Design for Manufacturability*):**

Εργαλεία που βοηθάνε στο να βελτιστοποιήσουν μια σχεδίαση και παράλληλα να διευκολύνουν και να μειώσουν το κόστος κατασκευής στο ελάχιστο δυνατό.

- **Κλειστότητα Σχεδίου (*Design Closure*):**

Η σχεδίαση ολοκληρωμένων κυκλωμάτων έγκειται σε πολλούς περιορισμούς και συνήθως η διόρθωση ενός προβλήματος οδηγεί κάποιο άλλο να γίνει χειρότερο από ότι ήταν. Το *design closure* είναι η διαδικασία κατά την οποία η αρχική VLSI σχεδίαση τροποποιείται από την αρχική της περιγραφή ούτως ώστε να ικανοποιεί ένα ευρύ φάσμα σχεδιαστικών περιορισμών και στόχων ταυτόχρονα.

- **Σχεδίαση και Ανάλυση Δικτύου Ισχύος (*Power Network Design and Analysis*):**

Στα ολοκληρωμένα κυκλώματα η ηλεκτρική ισχύς διανέμεται στα στοιχεία του chip μέσο ενός δικτύου αγωγών. Η σχεδίαση δικτύου ισχύος εμπλέκει την ανάλυση και τη σχεδίαση τέτοιων δικτύων.

- **Φυσική Επαλήθευση (*Physical Verification* – PV):**

Έλεγχος του κατά πόσο μία σχεδίαση είναι φυσικά κατασκευάσιμη και ότι τα chip που θα κατασκευαστούν δεν θα έχουν ελαττώματα που θα επηρεάζουν τη σωστή λειτουργία τους που είχε τεθεί εξαρχής.

  - **Έλεγχος Κανόνων Σχεδίου (*Design Rule Checking – DRC*):** Έλεγχος ενός αριθμού κανόνων που σχετίζονται με την τοποθέτηση και τη συνδεσιμότητα που απαιτούνται για τη διαδικασία της κατασκευής.
  - **Πλάνο εναντίων Σχηματικού Διαγράμματος (*Layout versus Schematic – LVS*):** Έλεγχος ισοδυναμίας του layout του chip με το σχηματικό κύκλωμα της αρχικής προδιαγραφής.
  - **Εξαγωγή Πλάνου (*Layout Extraction – RCX*):** Εξαγωγή netlist από το layout συμπεριλαμβανομένων στοιχείων, όπως παρασιτικών αντιστάσεων (*parasitic resistors – PRE*), πυκνωτών και πηνίων ενυπάρχοντα στο layout του chip.

- **Προετοιμασία Μάσκας (*Mask Data Preparation – MDP*):**

Παραγωγή τις πραγματικής φωτογραφικής μάσκας λιθογραφίας για τη φυσική κατασκευή τού chip.

  - Τεχνικές Εμπλουτισμού Ανάλυσης (Resolution Enhancements Techniques – RET): Μέθοδοι για την αύξηση της ποιότητας της τελικής φωτογραφικής μάσκας.
  - Διόρθωση Οπτικής Εγγύτητας (Optical Correction Proximity – OPC): Τεχνική εμπλουτισμού φωτολιθογραφίας* (ή οπτικής λιθογραφίας) που χρησιμοποιείται ευρέως για την αντιστάθμιση των λαθών που δημιουργούνται λόγω διάθλασης ή που είναι συνέπειες της επεξεργασίας.
  - Παραγωγή Μάσκας (Mask Generation): Παραγωγή επίπεδης μάσκας από το ιεραρχικό σχέδιο.

- **Κατασκευαστικός Έλεγχος (Manufacturing Test)**
  - **Αυτόματη Παραγωγή Δειγμάτων Ελέγχου (*Automatic Test Pattern Generation – ATPG*):** Δημιουργία δειγμάτων δεδομένων (*pattern data*) για τον έλεγχο των όσο το δυνατόν περισσότερων λογικών πυλών στο chip. Τα δείγματα δεδομένων χρησιμοποιούνται σαν είσοδος στο chip μετά την κατασκευή του ούτως ώστε να φανεί αν η συμπεριφορά του chip είναι η αναμενομένη, πράγμα που σημαίνει ότι το chip λειτουργεί σωστά ή σε αντίθετη περίπτωση ότι το chip είναι ελαττωματικό.
  - **Ενσωματωμένος Αυτοέλεγχος (*Built-in Self-Test – BIST*):** Μέσα στο chip ενσωματώνεται ένας ελεγκτής που μπορεί αυτόματα να ελέγξει τη λειτουργία ενός λογικού τμήματος ή μιας μνήμης.
  - **Σχέδιο Ελέγχου (*Design for Test – DFT*):** Στο netlist-πυλών προστίθενται λογικές δομές για να διευκολύνουν τον έλεγχο πιθανών ατελειών μετά την κατασκευή του δίσκου πυριτίου του chip.

- **TCAD (*Technology CAD*):**

Ουσιαστικά είναι ο κλάδος της αυτοματοποίησης της ηλεκτρονικής σχεδίασης που μοντελοποιεί την κατασκευή και τη λειτουργία ημιαγωγών. Η TCAD μπορεί ακόμα να περιλαμβάνει τη δημιουργία συμπαγών μοντέλων, όπως για παράδειγμα το πολύ γνωστά SPICE τρανζίστορ μοντέλα, που αποσκοπούν στην ηλεκτρική συμπεριφορά τέτοιων διατάξεων χωρίς αυτή να προκύπτει από τη θεμελιώδη φυσική. Ο προσομοιωτής SPICE μόνος του ανήκει στην κατηγορία ECAD.

- **Επίλυση Ηλεκτρομαγνητικών Πεδίων (*Electromagnetic Field Solvers*):**

Επίλυση των εξισώσεων Maxwell για τις περιπτώσεις που υπάρχει ενδιαφέρον ως προς την πλευρά μελέτης των ηλεκτρομαγνητικών πεδίων στο ολοκληρωμένο κύκλωμα.

---

* Διαδικασία που χρησιμοποιείται στην μικρό-κατασκευή (micro-fabrication) ώστε να αφαιρεθούν επιλεκτικά τμήματα μιας λεπτής μεμβράνης (film) ή η ακατέργαστη μάζα από ένα υπόστρωμα. Micro-fabrication είναι ο όρος που χρησιμοποιείται σε τεχνολογίες κατασκευής εξαρτημάτων κλίμακας μικρών του μέτρου (1 μm ή $1x10^{-6}$ m)

Κεφάλαιο 5

Τεχνολογία Ολοκληρωμένων Κυκλωμάτων FPGA

5.1 Εισαγωγή

Στο εισαγωγικό αυτό κεφάλαιο δίνουμε μία σύντομη εισαγωγή και περιγραφή της τεχνολογίας ψηφιακών κυκλωμάτων προγραμματιζόμενης λογικής (Field Programmable Gate Array – FPGA). Αφού ερμηνευθεί ο όρος FPGA (βλ. §5.2), πραγματοποιείται μία σύντομη ιστορική αναδρομή στα προγραμματιζόμενα ψηφιακά κυκλώματα (βλ. §5.3) την οποία ακολουθεί η περιγραφή της αρχιτεκτονικής των FPGAs (βλ. §5.4). Στη συνέχεια (βλ. §5.5) περιγράφεται η διαδικασία σχεδίασης ενός συστήματος σε FPGA και ο προγραμματισμός αυτού. Ακολούθως, παρουσιάζονται οι κυριότερες εφαρμογές των FPGA (βλ. §5.6) και οι γνωστότεροι κατασκευαστές ψηφιακών κυκλωμάτων προγραμματιζόμενης λογικής (βλ. §5.7). Τέλος, εκθέτονται τα βασικότερα χαρακτηριστικά της γενιάς Xilinx Spartan–3 FPGA, που χρησιμοποιήθηκε για τις ανάγκες της παρούσας διπλωματικής (βλ. §5.8).

5.2 Ψηφιακά Ολοκληρωμένα Κυκλώματα Προγραμματιζόμενης Λογικής

Ένα ψηφιακό ολοκληρωμένο κύκλωμα προγραμματιζόμενης λογικής (FPGA) είναι μία ψηφίδα ημιαγωγού που περιέχει προγραμματίσιμα λογικά μέρη και διασυνδέσεις. Τα προγραμματίσιμα λογικά μπλοκ προγραμματίζονται μέσω της επαναληπτικής διασύνδεσης βασικών λογικών πυλών, όπως οι ΚΑΙ (AND), Η’ (OR), Αποκλειστικό – Η’ (XOR), Αντιστροφή (NOT), αλλά και πιο σύνθετων συνδυαστικών συναρτήσεων, όπως αποκωδικοποιητές και απλές μαθηματικές συναρτήσεις. Στα περισσότερα FPGA, τα λογικά μπλοκ περιλαμβάνουν επίσης κάποια στοιχεία μνήμης, τα οποία μπορεί να είναι σύνθετα ή να αποτελούνται απλά από flip-flops.

Η ιεραρχία των προγραμματίσιμων διασυνδέσεων επιτρέπει στα λογικά μπλοκ να διασυνδέονται ανάλογα με την επιθυμία του σχεδιαστή του κάθε συστήματος, μοιάζει δηλαδή με breadboard σε μορφή chip. Αυτά τα λογικά μπλοκ μπορούν να προγραμματιστούν από τον πελάτη/σχεδιαστή μετά τη διαδικασία παραγωγής της ψηφίδας για τη δημιουργία οποιασδήποτε λογικής συνάρτησης στο FPGA (γι' αυτό άλλωστε χρησιμοποιείται ο όρος «προγραμματιζόμενη λογική»).

Τα πλεονεκτήματα των FPGA είναι ποικίλα. Μπορεί σε σχέση με άλλα ολοκληρωμένα κυκλώματα όπως τα ASIC (Application Specific Integrated Circuits) να είναι γενικά αργότερα, έχουν όμως το πολύ σημαντικό χαρακτηριστικό του επαναπρογραμματισμού. Ταυτόχρονα ο χρόνος που απαιτείται για τη διάθεσή τους στην αγορά είναι πολύ μικρός ενώ η ανάπτυξη συστημάτων σε FPGA απαιτεί σχετικά *μικρό κόστος έρευνας, σχεδίασης και ελέγχου του συστήματος*[*].

5.3 Ιστορική Αναδρομή

Η σύλληψη για τη δημιουργία των ψηφιακών κυκλωμάτων προγραμματιζόμενης λογικής (FPGA) ανήκει στον Ross Freeman, έναν από τους συνιδιοκτήτες της εταιρείας Xilinx, που είναι σήμερα ο πρωτοπόρος στη κατασκευή FPGA (βλ. §5.7).

Οι ιστορικές ρίζες των FPGA βρίσκονται στα κυκλώματα προγραμματιζόμενης λογικής CPLD (*Complex Programmable Logic Devices*) που αναπτύχθηκαν στις αρχές της δεκαετίας του 1980. Τα CPLD και τα FPGA περιλαμβάνουν έναν σχετικά μεγάλο αριθμό από προγραμματίσιμα λογικά στοιχεία. Η πυκνότητα λογικών πυλών στα CPLD κυμαίνεται από μερικές χιλιάδες έως δεκάδες χιλιάδες πύλες, ενώ στα FPGA συνήθως από μερικές δεκάδες χιλιάδες έως πολλά εκατομμύρια πύλες.

Η διαφορά μεταξύ CPLD και FPGA έγκειται κυρίως στην αρχιτεκτονική τους. Τα CPLD έχουν γενικά μια περιορισμένη δομή που αποτελείται από έναν ή περισσότερους λογικούς πίνακες «αθροίσματος γινομένων» (*sum–of–products*), οι οποίοι τροφοδοτούν έναν σχετικά μικρό αριθμό από καταχωρητές. Αποτέλεσμα αυτής της δομής είναι η μικρή ευελιξία. Ταυτόχρονα όμως έχουν το πλεονέκτημα της ευκολότερης πρόβλεψης των χρονικών καθυστερήσεων (*timing delays*) αλλά και του υψηλού λόγου λογικής προς διασυνδέσεις. Σε αντίθεση με τα CPLD, η δομή των FPGA κατακλύζεται από διασυνδέσεις, γεγονός το οποίο εγγυάται τη μεγαλύτερη ευελιξία, αλλά συνάμα αυξάνει τη δυσκολία σχεδίασης.

Μία ακόμη αξιοσημείωτη διαφορά μεταξύ των FPGA και των προκατόχων του CPLD είναι η ύπαρξη υψηλού επιπέδου λογικών συναρτήσεων στα FPGA, όπως οι ενσωματωμένοι πολλαπλασιαστές, πολυπλέκτες και μνήμες. Ακόμη τα σύγχρονα FPGA έχουν τη δυνατότητα μερικού προγραμματισμού, δηλαδή τον προγραμματισμό μέρους του

[*] Αγγλικός όρος: Low non-recurring engineering cost

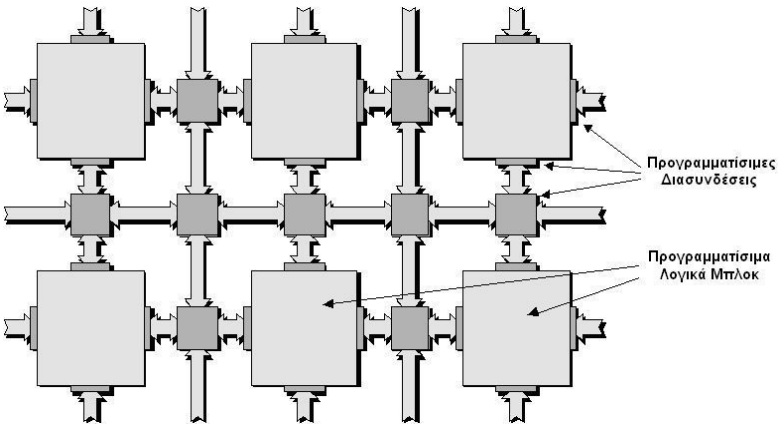
chip, ενώ κάποιες άλλες περιοχές του να συνεχίζουν να «εκτελούνται» παραμένοντας αναλλοίωτες.

Τα τελευταία κυρίως χρόνια παρατηρείται ο συνδυασμός των κλασσικών FPGA που αναφέρθηκαν προηγουμένως με ενσωματωμένους μικροεπεξεργαστές και περιφερειακές διασυνδέσεις. Μία εναλλακτική προσέγγιση αυτού είναι η υλοποίηση μαλακών πυρήνων επεξεργαστών (*soft processor cores*) σε FPGA (όπως οι Microblaze και Picoblaze της Xilinx και Nios II της Altera) [41][42][43].

Όπως προαναφέρθηκε τα σύγχρονα FPGA έχουν τη δυνατότητα προγραμματισμού ενώ κάποιο τμήμα τους βρίσκεται στη φάση εκτέλεσης (*run–time*). Το γεγονός αυτό οδηγεί στη ιδέα των επαναδιαμορφώσιμων συστημάτων (*reconfigurable systems*), τα οποία επαναδιαμορφώνονται μόνα τους με σκοπό να ανταποκριθούν όσο καλύτερα γίνεται σε διαφορετικές συνθήκες. Ακόμη τα FPGA τελευταίας γενιάς έχουν αυξημένη πυκνότητα (εκατομμύρια προγραμματιζόμενες λογικές μονάδες), είναι ιεραρχικά δομημένα (ύπαρξη ιεραρχίας τόσο στη διασύνδεση όσο και στη λογική), ενσωματώνουν μνήμες (RAM, ROM, CAM) και αριθμητικές μονάδες επεξεργασίας (αθροιστές, πολλαπλασιαστές, μετρητές), υποστηρίζουν διάφορα standards E/E και περιέχουν κυκλώματα διαχείρισης του ρολογιού (DLL, PLL).

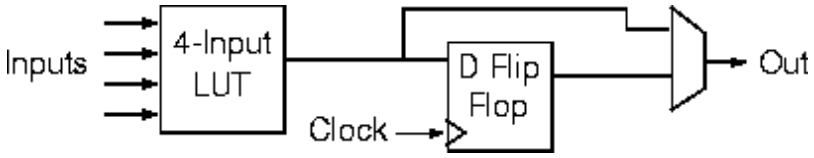
5.4 Περιγραφή Αρχιτεκτονικής FPGA

Η κλασσική βασική αρχιτεκτονική των FPGA αποτελείται από έναν πίνακα διαμορφώσιμων λογικών μπλοκ (*Configurable Logic Blocks* – CLBs) και ένα σύνολο καναλιών διασύνδεσης που διατρέχουν οριζόντια και κάθετα το chip υπό μορφή γραμμών και στηλών. Κάθε κανάλι διασύνδεσης αποτελείται από πολλά τμήματα καλωδίωσης. Πολλαπλά τερματικά εισόδου – εξόδου (I/O *pads*) είναι δυνατόν να χωρέσουν στο ύψος μίας γραμμής ή στο πλάτος μίας στήλης του πίνακα. Γενικά, όλα τα κανάλια διασύνδεσης έχουν το ίδιο πλάτος. Στο Σχήμα 5.1 που ακολουθεί αποτυπώνεται ένα απλοποιημένο σχέδιο της αρχιτεκτονικής που μόλις περιγράφηκε.



Σχήμα 5.1: Απλοποιημένη σχεδίαση της αρχιτεκτονικής ενός FPGA

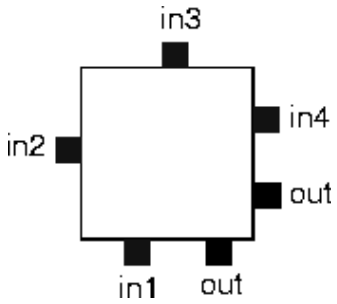
Το θεμελιώδες δομικό στοιχείο ενός FPGA αποτελείται από ένα LUT* τεσσάρων εισόδων και έναν μανδαλωτή (*flip-flop*), όπως απεικονίζεται στο Σχήμα 5.2.



Σχήμα 5.2: Θεμελιώδες δομικό στοιχείο (λογικό μπλοκ) ενός FPGA

Όπως φαίνεται και στο προηγούμενο σχήμα, υπάρχει μόνο μία έξοδος που μπορεί είτε να τροφοδοτείται σε κάποιον καταχωρητή (*registered output*) είτε όχι. Το λογικό μπλοκ έχει τέσσερις εισόδους και ένα ρολόι. Οι είσοδοι και το ρολόι διαχειρίζονται ξεχωριστά αφού η διαδρομή που ακολουθεί το σήμα του ρολογιού «δρομολογείται» από ξεχωριστά ειδικού-σκοπού δίκτυα δρομολόγησης (*special – purpose routing networks*).

Για τη συγκεκριμένη αρχιτεκτονική, η διάταξη των ακίδων (*pins*) του λογικού μπλοκ στο εσωτερικό του FPGA δίνονται στο Σχήμα 5.3.

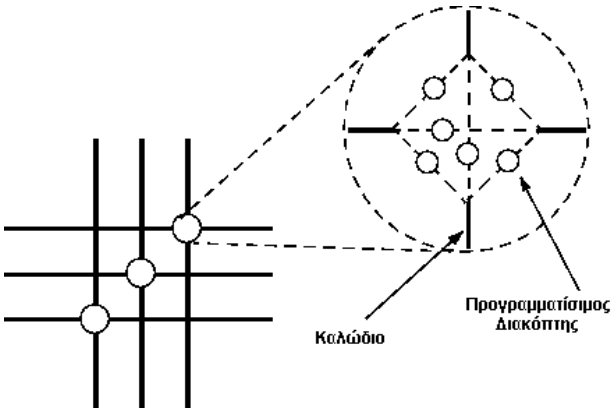


Σχήμα 5.3: Διάταξη των ακίδων (*pins*) ενός λογικού μπλοκ

Κάθε έξοδος του λογικού μπλοκ μπορεί να συνδεθεί με κάθε τμήμα καλωδίωσης των καναλιών διασύνδεσης που γειτονεύει. Όμοια μπορεί να συνδεθεί και κάθε pad εισόδου – εξόδου με τα γειτονικά του κανάλια διασύνδεσης.

Γενικά, κάθε τμήμα καλωδίωσης μέσα στα κανάλια διασύνδεσης έχει μήκος όσο και το μήκος του λογικού μπλοκ καταλήγοντας σε ένα σύνολο προγραμματίσιμων διακοπτών (*switch box*). Αν ένας από αυτούς τους διακόπτες ενεργοποιηθεί, τότε μεγαλύτερα μονοπάτια διασύνδεσης (*routing paths*) μπορούν να δημιουργηθούν. Οποτεδήποτε ένα οριζόντιο και ένα κάθετο κανάλι διασύνδεσης τέμνονται, εκεί υπάρχει ένα σύνολο προγραμματίσιμων διακοπτών. Στη συγκεκριμένη αρχιτεκτονική κάθε ένα καλώδιο του καναλιού συνδέεται με τρεις διακόπτες που του δίνουν τη δυνατότητα να συνδεθεί με τρία άλλα καλώδια του γειτονικού καναλιού διασύνδεσης. Στο Σχήμα 5.4 που ακολουθεί παρουσιάζεται η τοπολογία των διακοπτών που περιγράφηκαν.

* **L**ook – **U**p **T**able



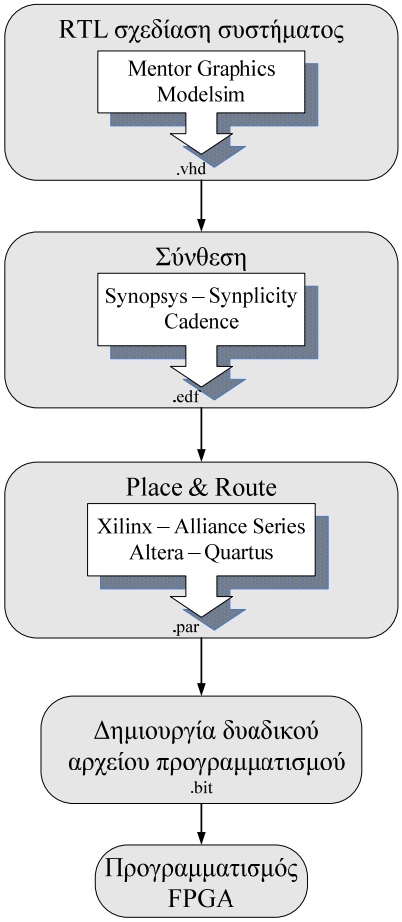
Σχήμα 5.4: Τοπολογία προγραμματίσιμων διακοπτών σε μία τομή δύο καναλιών διασύνδεσης

5.5 Διαδικασία Σχεδίασης Συστήματος σε FPGA και Προγραμματισμός FPGA

Για τη σχεδίαση ενός συστήματος σε υλικό και τον ορισμό της συμπεριφοράς ενός FPGA, ο σχεδιαστής οφείλει να χρησιμοποιήσει αρχικά μία γλώσσα περιγραφής υλικού ή ένα σχηματικό (*schematic*). Οι πιο γνωστές γλώσσες περιγραφής υλικού είναι η VHDL και η Verilog. Αφού το σύστημα σχεδιαστεί σε RTL επίπεδο, ελέγχεται η σωστή του λειτουργία μέσω προσομοιώσεων με τη βοήθεια ενός εργαλείου προσομοίωσης (*simulator*) και διανύσματα δοκιμής (test vectors) που παρέχονται στο σύστημα μέσω ενός περιβάλλοντος ελέγχου (*testbench*).

Στη συνέχεια χρησιμοποιώντας εργαλεία αυτοματοποίησης της σχεδίασης (*Electronic Design Automation* – EDA tools), η σχεδίαση μεταφράζεται σε ένα αρχείο περιγραφής δικτυωμάτων (*netlist*), η μορφή του οποίου εξαρτάται από την εκάστοτε τεχνολογία που χρησιμοποιείται (*technology mapped netlist*). Η διαδικασία αυτή αποκαλείται «Λογική σύνθεση» (*Logic synthesis*). Για να εισαχθεί το σχεδιασμένο κύκλωμα στο FPGA πρέπει το netlist αυτό να περάσει από μία διαδικασία που ονομάζεται «Θέση και Δρομολόγηση» (*Place and Route – PR*), η οποία γίνεται συνήθως μέσω μιας πλατφόρμας εργαλείων που διατίθεται από την εταιρεία διάθεσης του FPGA που χρησιμοποιείται. Ο σχεδιαστής πρέπει σε αυτό το στάδιο να ελέγξει και να επιβεβαιώσει τα σωστά αποτελέσματα που παράγουν τα τρία στάδια της διαδικασίας PR, συγκεκριμένα της διαδικασίας μετάφρασης (*translation*), της διαδικασίας αντιστοίχησης (*map*) και της διαδικασίας θέσης και δρομολόγησης.

Ο έλεγχος αυτός γίνεται μέσω προσομοιώσεων σε επίπεδο καταχωρητών (RTL *simulation*) και ανάλυσης χρονισμού (*timing simulation*) και μέσω άλλων διάφορων μεθοδολογιών ελέγχου. Από τη στιγμή που η διαδικασία σχεδίασης και ελέγχου τελειώσει, παράγεται ένα δυαδικό αρχείο (*bitstream*) προγραμματισμού του FPGA. Η διαδικασία που αναλύθηκε παραπάνω απεικονίζεται συνοπτικά στο Σχήμα 5.5.



Σχήμα 5.5: Διαδικασία και εργαλεία σχεδίασης συστήματος και προγραμματισμού FPGA

Οι μεγαλύτερες και πιο διάσημες εταιρείας διάθεσης των EDA λογισμικών που αναφέρθηκαν είναι:

- Cadence Design Systems (California, USA)
- Synopsys (California, USA)
- Mentor Graphics (Oregon, USA)
- Magma Design Automation (California, USA)
- Zuken Inc. (Yokohama, Japan)

Προκειμένου να μειωθεί ο βαθμός πολυπλοκότητας και ο χρόνος που απαιτείται για τη σχεδίαση, έχουν γίνει αρκετές προσπάθειες για την αύξηση του επιπέδου αφαίρεσης (*abstraction level*) στις γλώσσες περιγραφής υλικού. Ο συνδυασμός υψηλού επιπέδου γλωσσών προγραμματισμού που δανείζονται έννοιες από τις υπάρχουσες HDL γλώσσες, οδήγησε στην ανάπτυξη γλωσσών συμπεριφορικής περιγραφής και περιγραφής υλικού όπως η SystemC, και Handel–C αντίστοιχα.

Τέλος, για την απλοποίηση της σχεδίασης σύνθετων συστημάτων σε FPGA, έχουν δημιουργηθεί βιβλιοθήκες που υλοποιούν διάφορες δομικές μονάδες (πυρήνες) και αναφέρονται ως IP Cores. Τέτοιοι πυρήνες είναι συνήθως διαθέσιμοι από τις εταιρείες κατασκευής FPGA ή από δικτυακές κοινότητες σχεδιαστών (e.g., OpenCores, www.opencores.org, Design & Reuse, www.design-reuse.com).

5.6 Εφαρμογές FPGA

Οι εφαρμογές των FPGA είναι ποικίλες και περιλαμβάνουν κυρίως τους παρακάτω τομείς:

- Ψηφιακή επεξεργασία σήματος (DSP)
- Επικοινωνία βασισμένη σε λογισμικό
- Ανάπτυξη αλγορίθμων σε υλικό
- Συστήματα αεροδιαστημικής και άμυνας
- Ιατρική απεικόνιση
- Όραση υπολογιστών
- Αναγνώριση φωνής
- Κρυπτογραφία
- Βιοπληροφορική
- Δημιουργία συστημάτων σε ψηφίδα (*Systems on a Chip – SoC*)

5.7 Κατασκευαστές FPGA

Στο τέλος του 2006, η αγορά των FPGA αναδεικνύει δύο κύριους κατασκευαστές FPGA και ένα πλήθος άλλων εταιρειών παραγωγής FPGA που διαφοροποιούνται από τους δύο πρωτοπόρους προσφέροντας μοναδικές δυνατότητες στον σχεδιαστή.

Παρακάτω αναφέρονται οι σημαντικότεροι εξ' αυτών:

- Η εταιρεία Xilinx είναι μία εκ των δύο πρωτοπόρων στην κατασκευή FPGA
- Η εταιρεία Altera αποτελεί το δεύτερο ισάξιο ηγέτη
- Η εταιρεία Lattice Semiconductors είναι ο τρίτος προμηθευτής FPGA στον κόσμο και είναι ο πρώτος προμηθευτής μη πτητικών (*non-volatile*), βασισμένων σε μνήμες flash
- Η εταιρεία Actel προσφέρει μη τηκόμενα και επαναπρογραμματίσιμα FPGA βασισμένα σε μνήμες flash
- Η εταιρεία Quicklogic προσφέρει μη τηκόμενα (*fused*) μιας φοράς προγραμματίσιμα FPGA
- Η εταιρεία Achronix Semiconductor αναπτύσσει αυτό το διάστημα πολύ γρήγορα FPGA (ταχύτητες 2GHz)
- Η εταιρεία MathStar, Inc. προσφέρει ολοκληρωμένο κύκλωμα προγραμματιζόμενης λογικής που μοιάζει με FPGA και αποκαλείται FPOA (*Field Programmable Object Array*)

5.8 Οικογένεια FPGA Xilinx® Spartan™ – 3

Στις παραγράφους που ακολουθούν αναλύεται η οικογένεια FPGA Spartan–3 της Xilinx που αποτελεί και την κύρια πλατφόρμα ολοκληρωμένων κυκλωμάτων προγραμματιζόμενης λογικής που χρησιμοποιήθηκε στην παρούσα διατριβή για την υλοποίηση των ψηφιακών κυκλωμάτων.

5.8.1 Εισαγωγή

Η οικογένεια FPGA Spartan–3 της Xilinx [44] είναι σχεδιασμένη έτσι ώστε να ανταποκρίνεται στις ανάγκες σχεδίασης ηλεκτρονικών κυκλωμάτων υψηλής κλίμακας, όγκου και χαμηλού κόστους. Η οκταμελής οικογένεια προσφέρει πυκνότητες που εκτείνονται από 50.000 έως 5.000.000 πύλες. Σε σχέση με τους προγόνους της, περιλαμβάνει την αύξηση της ποσότητας των λογικών πόρων (logic resources), του μεγέθους των εσωτερικών RAM*, το συνολικό αριθμό εισόδων – εξόδων (I/Os) και γενικά του συνολικού επιπέδου απόδοσης, όπως επίσης και την εξέλιξη διαχείρισης των λειτουργιών του ρολογιού. Λόγω του εκπληκτικά χαμηλού κόστους, η οικογένεια Spartan–3 έχουν ιδανική προσαρμογή σε μία ευρεία κλίμακα ηλεκτρονικών εφαρμογών.

5.8.2 Περιγραφή Αρχιτεκτονικής

Η αρχιτεκτονική της οικογένειας Spartan–3 αποτελείται από πέντε θεμελιώδη προγραμματίσιμα λειτουργικά στοιχεία.

- **Διαμορφώσιμα Λογικά Μπλοκ (CLB†)**

Τα CLB περιέχουν LUTs βασισμένα σε RAM για την υλοποίηση λογικών και αποθηκευτικών στοιχείων, τα οποία μπορούν να χρησιμοποιηθούν ως flip–flops ή μανδαλωτές (*latches*). Τα CLB μπορούν να προγραμματιστούν για την υλοποίηση μιας ευρείας ποικιλίας λογικών συναρτήσεων όπως επίσης και για την αποθήκευση δεδομένων.

- **Μπλοκ εισόδων – εξόδων (I/O Blocks – IOBs).**

Τα IOBs ελέγχουν τη ροή δεδομένων μεταξύ των pins εισόδου– εξόδου και της εσωτερικής λογικής. Κάθε μπλοκ υποστηρίζει αμφίδρομη ροή δεδομένων μαζί με λειτουργία τριών καταστάσεων (*3–state*).

- **RAM μπλοκ**

Τα μπλοκ της RAM προσδίδουν αποθήκευση δεδομένων σε μορφή δίθυρων μπλοκ (dual port) των 18Kbit‡.

---

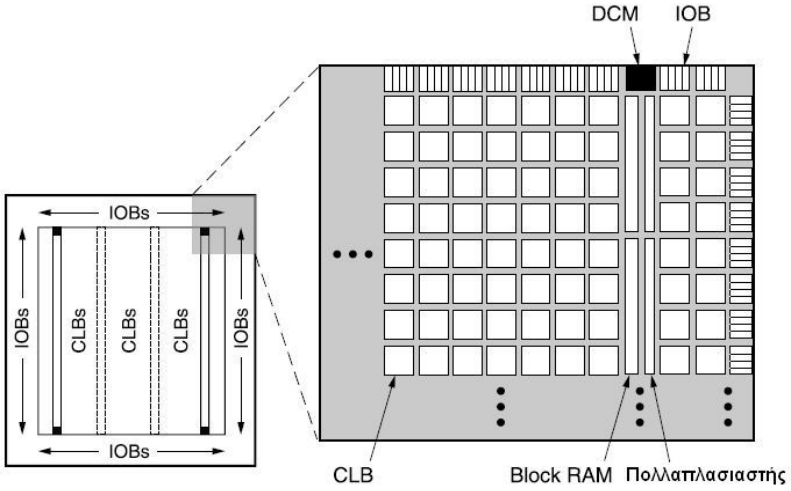
* **R**andom **A**ccess **M**emory
† **C**onfigurable **L**ogic **B**locks
‡ 1 Kbit = 1024 bits

- **Πολλαπλασιαστές**

Τα μπλοκ πολλαπλασιασμού δέχονται σαν είσοδο 18 bit δυαδικούς αριθμούς και υπολογίζουν το αποτέλεσμα.

- **Μπλοκ διαχείρισης ρολογιού (*Digital Clock Manager* – DCM).**

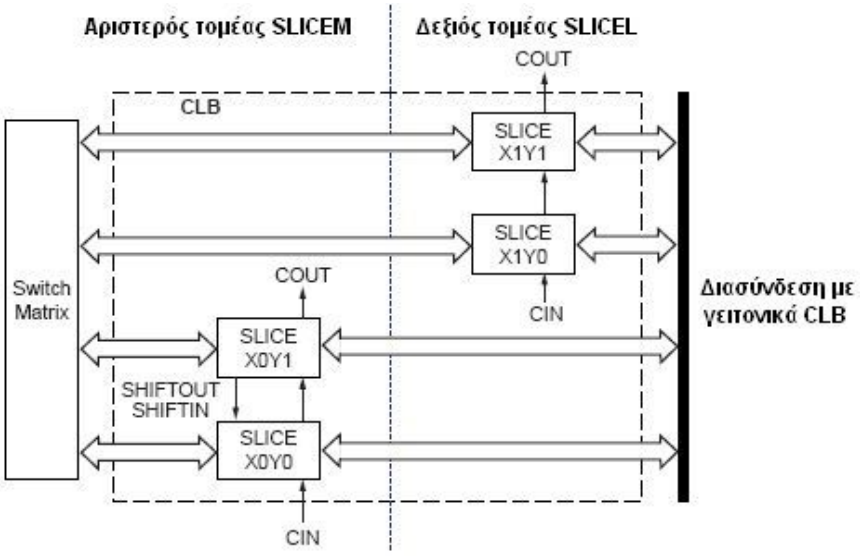
Τα μπλοκ αυτά προφέρουν αυτό–βαθμονομούμενες (*self–calibrating*), πλήρως ψηφιακές λύσεις για διανομή, καθυστέρηση, πολλαπλασιασμό, διαίρεση και ολίσθηση φάσης σημάτων ρολογιού. Τα DCM μπλοκ απεικονίζονται στο Σχήμα 5.6.



Σχήμα 5.6: DCM μπλοκ στην αρχιτεκτονική της οικογένειας Xilinx Spartan–3

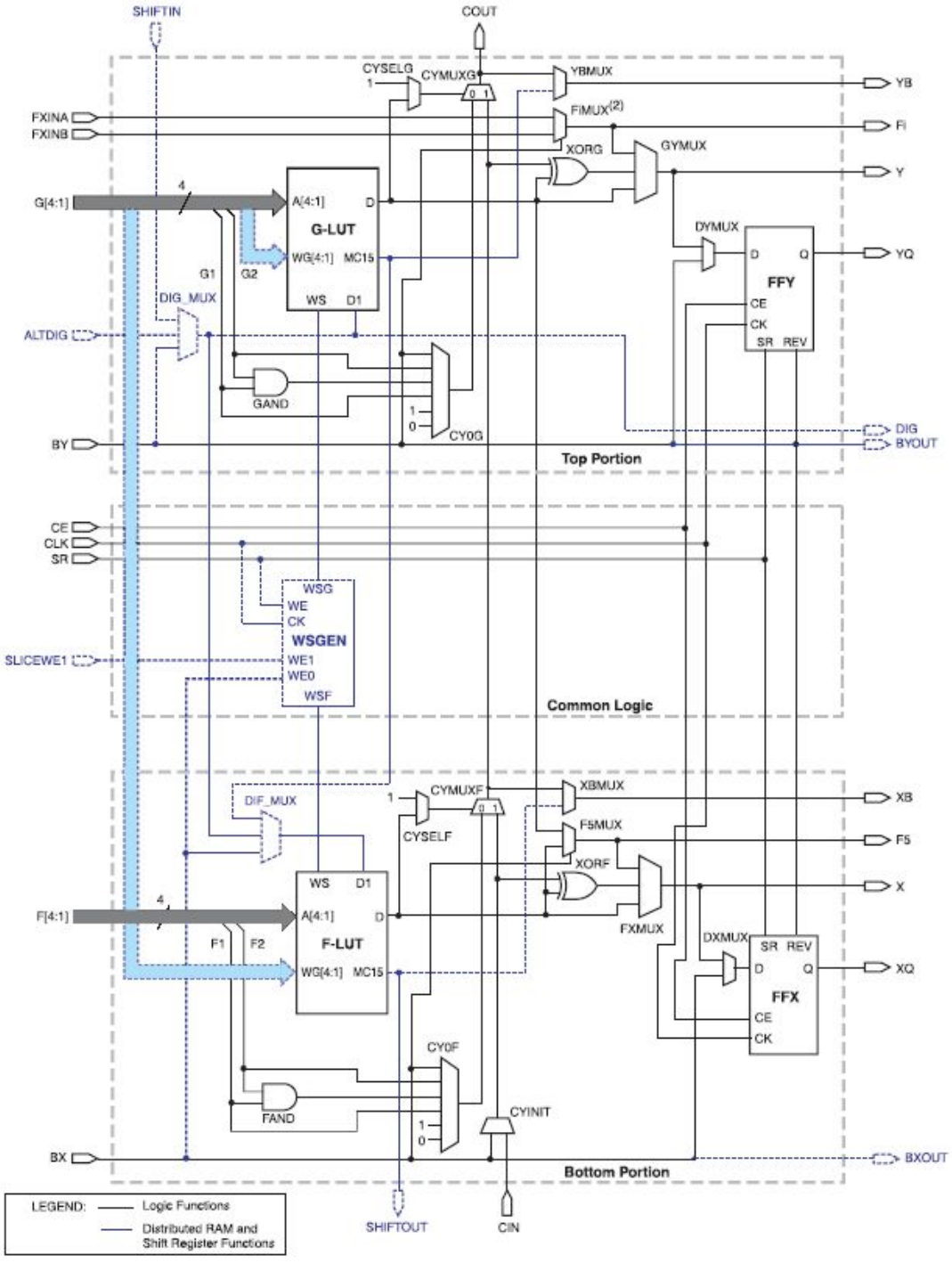
5.8.2.1 Περιγραφή των Διαμορφώσιμων Λογικών "Μπλοκ" (*CLBs*)

Τα CLB αποτελούν την κεντρική λογική μονάδα για υλοποίηση σύγχρονων αλλά και συνδυαστικών κυκλωμάτων. Κάθε CLB εμπεριέχει τέσσερεις διασυνδεδεμένους τομείς, όπως φαίνεται στο Σχήμα 5.7. Αυτοί οι τομείς είναι ομαδοποιημένοι σε ζεύγη. Κάθε ζεύγος είναι οργανωμένο ως μία στήλη με μία ανεξάρτητη αλυσίδα κρατουμένου.



Σχήμα 5.7: Διαρρύθμιση των τομέων στο εσωτερικό ενός CLB

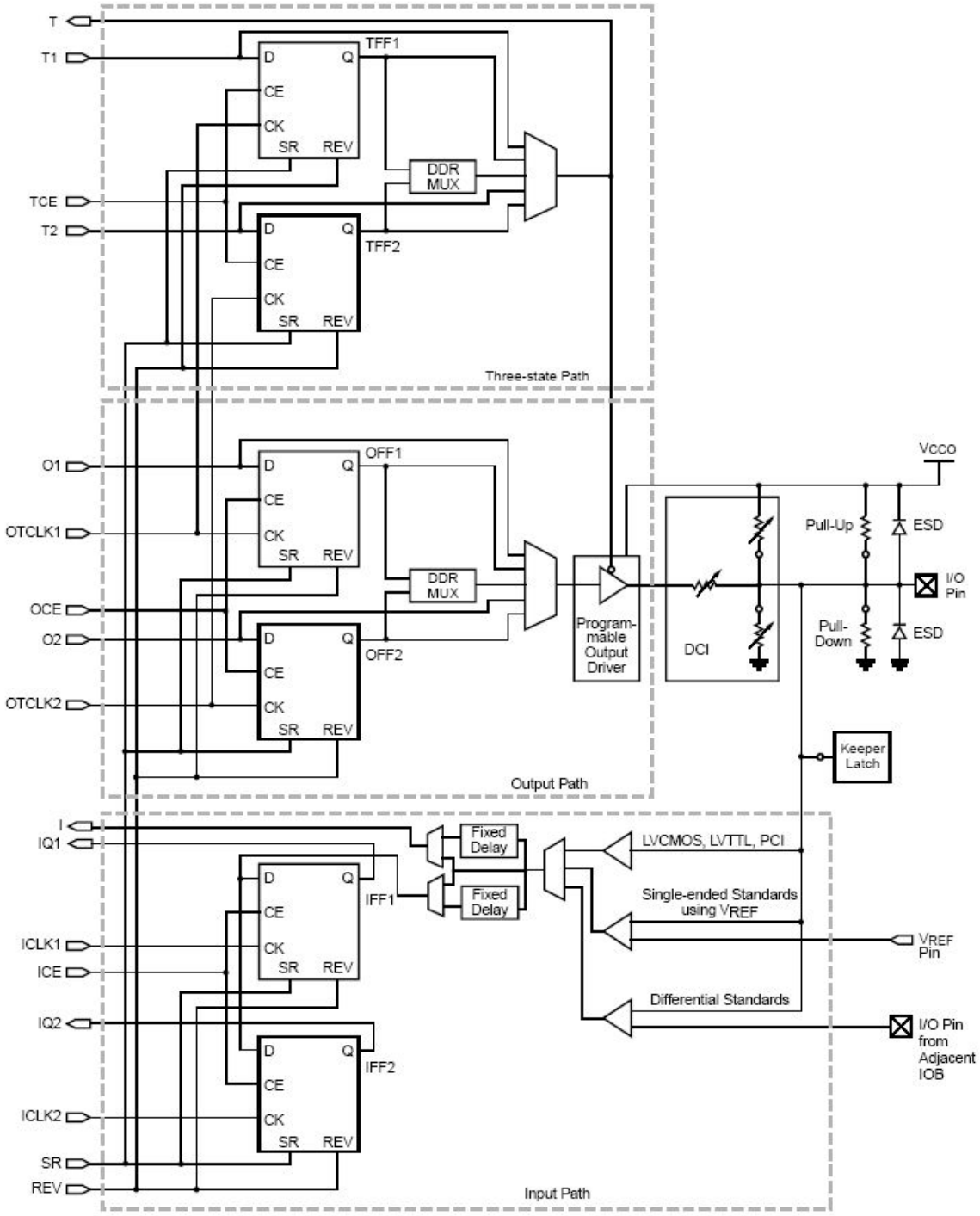
Και οι τέσσερεις τομείς έχουν τα ακόλουθα κοινά στοιχεία: Δύο γεννήτριες λογικών συναρτήσεων, δύο στοιχεία αποθήκευσης, δύο ευρείας λειτουργίας πολυπλέκτες, λογική κρατουμένων και αριθμητικές πύλες, όπως εμφανίζεται στο Σχήμα 5.8. Τόσο το αριστερό (SLICEM) όσο και το δεξιό (SLICEL) ζεύγος τομέων χρησιμοποιούν αυτά τα στοιχεία για να παράσχουν λογικές, αριθμητικές όπως και ROM συναρτήσεις. Εκτός από αυτές το αριστερό ζεύγος υποστηρίζει επιπρόσθετες λειτουργίες: Αποθήκευση δεδομένων σε κατανεμημένες RAM και ολίσθηση δεδομένων με καταχωρητές 16 bits.



Σχήμα 5.8: Απλουστευμένο διάγραμμα του αριστερού τομέα SLICEM

5.8.2.2 Περιγραφή των "Μπλοκ" Εισόδων – Εξόδων (*IOBs*)

Τα μπλοκ εισόδου–εξόδου προσδίδουν μία προγραμματίσιμη αμφίδρομη διεπιφάνεια μεταξύ ενός pin εισόδου εξόδου και της εσωτερικής λογικής του FPGA. Ένα απλουστευμένο διάγραμμα της εσωτερικής δομής ενός IOB εμφανίζεται στο Σχήμα 5.9.



Σχήμα 5.9: Απλουστευμένο διάγραμμα IOB

Στο IOB ενυπάρχουν τρεις κεντρικοί δίοδοι σημάτων (*signal path*): η δίοδος εξόδου, η δίοδος εισόδου και η τρικατάστατη (*tri-state*) δίοδος. Κάθε δίοδος έχει το δικό της ζεύγος

στοιχείων αποθήκευσης και μπορεί να λειτουργήσει είτε ως καταχωρητής είτε ως μανδαλωτής.

5.8.2.3 Περιγραφή των RAM "Μπλοκ"

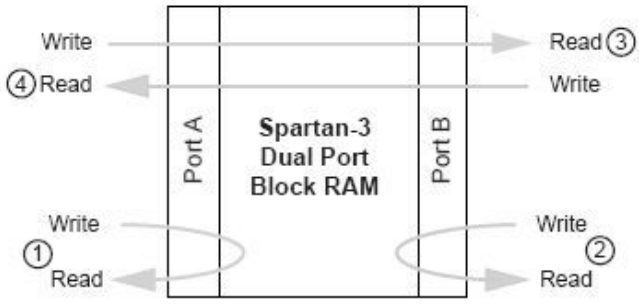
Όλα τα ολοκληρωμένα κυκλώματα προγραμματιζόμενης λογικής (*τσιπ – chip*) Spartan–3 υποστηρίζουν μπλοκ μνήμης RAM, τα οποία είναι οργανωμένα ως διαμορφώσιμα (*configurable*), σύγχρονα (*synchronous*) μπλοκ των 18 kbits. Ένα μπλοκ RAM αποθηκεύει σχετικά μεγάλες ποσότητες δεδομένων περισσότερο αποτελεσματικά, σε σχέση με την κατανεμημένη RAM. Ο λόγος πλάτος προς βάθος του καθενός μπλοκ είναι διαμορφώσιμος.

Επιπροσθέτως, πολλαπλά μπλοκ μπορούν να διαταχθούν ώστε να δημιουργηθεί ακόμα πλατύτερη ή/και βαθύτερη μνήμη. Στο Πίνακα 5.1 φαίνεται ο αριθμός των RAM μπλοκ, η χωρητικότητα αποθήκευσης δεδομένων.

Τσιπ	**Συνολικός Αριθμός μπλοκ RAM**	**Σύνολο διευθυνσιοδοτήσιμων περιοχών (bits)**
XC3S50	4	73.728
XC3S200	12	221.184
XC3S400	16	294.912
XC3S1000	24	442.368
XC3S1500	32	589.824
XC3S2000	40	737.280
XC3S4000	96	1.769.472
XC3S5000	104	1.916.928

Πίνακας 5.1: Αριθμός και μέγεθος των RAM μπλοκ

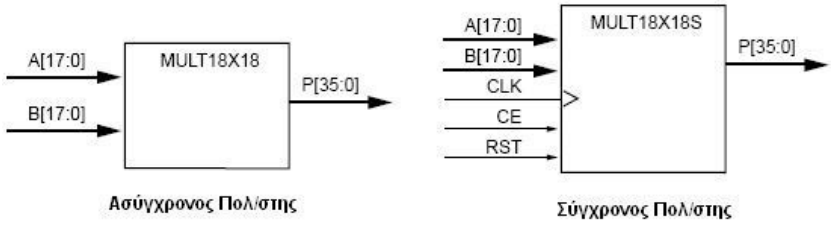
Το μπλοκ μνήμης RAM έχει δομή διπλής θύρας (*dual port*). Οι δύο ταυτόσημες θύρες δεδομένων, που ονομάζονται A και B, επιτρέπουν την ανεξάρτητη πρόσβαση στο ίδιο RAM μπλοκ, το οποίο έχει ως ανώτατη χωρητικότητα τα 18.432 bits. Στο Σχήμα 5.10 απεικονίζεται η δομή ενός μπλοκ RAM.



Σχήμα 5.10: Δομή μπλοκ μνήμης RAM

5.8.2.4 Περιγραφή Πολλαπλασιαστών

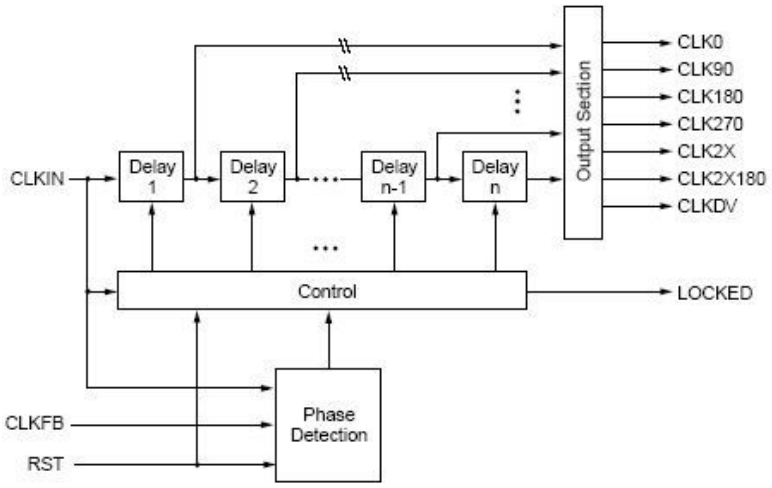
Όλα τα τσιπ Spartan–3 διαθέτουν ενσωματωμένους πολλαπλασιαστές που δέχονται ως είσοδο δύο 18-bit λέξεις παράγοντας ένα 36-bit γινόμενο. Οι είσοδοι εισάγονται σε μορφή συμπληρώματος του 2 και οι πολλαπλασιαστές αντιστοιχίζονται σε ξεχωριστά μπλοκ RAM ο καθένας. Υπάρχουν δύο είδη πολλαπλασιαστών, ο ασύγχρονος και ο σύγχρονος πολλαπλασιαστής που διαθέτει έναν καταχωρητή στην έξοδο (βλ. Σχήμα 5.11).



Σχήμα 5.11: Τύποι ενσωματωμένων πολλαπλασιαστών

5.8.2.5 Περιγραφή του "Μπλοκ" Διαχείρισης Ρολογιού (*DLL*)

Τα τσιπ Spartan–3 προσφέρουν ευέλικτο και πλήρη έλεγχο της συχνότητας του ρολογιού, της ολίσθησης φάσης και του skew* κάνοντας χρήση του διαχειριστή σημάτων DCM. Για την επίτευξη των προαναφερόμενων, το DCM μπλοκ χρησιμοποιεί ένα βρόχο καθυστέρησης (*Delay–Locked Loop* - DLL), ένα πλήρως ψηφιακό σύστημα ελέγχου που χρησιμοποιεί ανάδραση για να διατηρεί τα χαρακτηριστικά του σήματος του ρολογιού με υψηλό βαθμό ακρίβειας παρά τις συνήθεις μεταβολές στη θερμοκρασία λειτουργίας και τροφοδοσίας. Ένα απλουστευμένο λειτουργικό διάγραμμα του DLL απεικονίζεται στο Σχήμα 5.12.

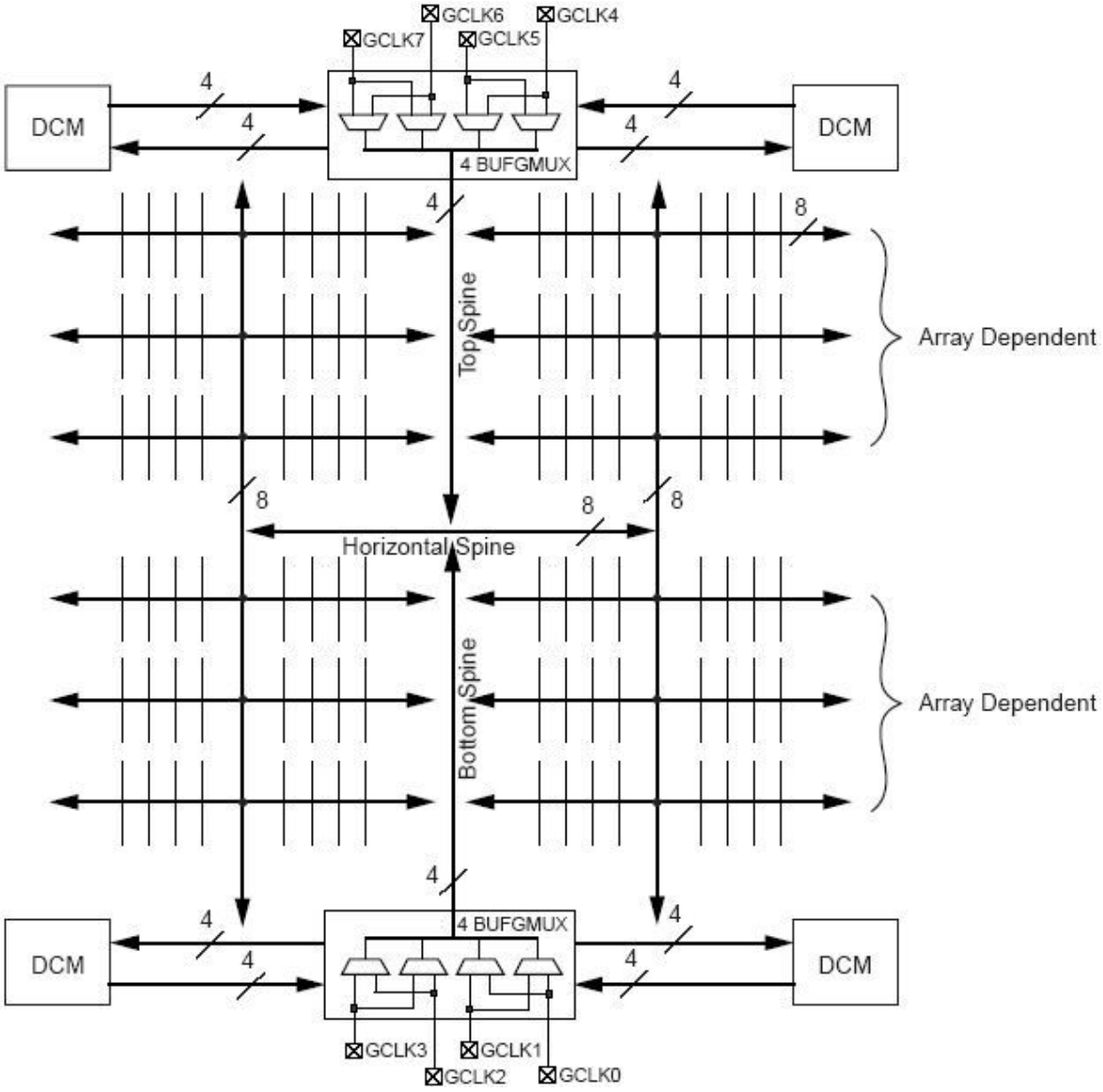


Σχήμα 5.12: Απλουστευμένο διάγραμμα του DLL

* Η κατάσταση στην οποία το σήμα του ρολογιού αποκλίνει από τη θέση μηδενικής φάσης (zero–phase alignment state).

5.8.2.6 Δίκτυο Ρολογιών (Global Clock Network)

Όλα τα τσιπ της οικογένειας Spartan–3 ενσωματώνουν 8 ρολόγια που ονομάζονται GCLK0 – GCLK7. Αυτά τα σήματα ρολογιού προσδίδουν πρόσβαση σε ένα δίκτυο χαμηλής χωρητικότητας και χαμηλού skew, το οποίο προσαρμόζεται καλά σε σήματα υψηλής συχνότητας. Το δίκτυο αυτό εμφανίζεται στο Σχήμα 5.13. Τα ρολόγια GCLK0 έως GCLK3 βρίσκονται στο κέντρο της κάτω γωνίας του FPGA ενώ τα GCLK4 έως GCLK7 στο κέντρο της πάνω γωνίας του FPGA.



Σχήμα 5.13: Δίκτυο ρολογιού Spartan–3

5.8.2.7 Δίκτυο Διασύνδεσης

Το δίκτυο διασύνδεσης μεταφέρει σήματα μεταξύ των διαφόρων λειτουργικών στοιχείων του FPGA. Υπάρχουν τέσσερα είδη διασύνδεσης:

- Μακριές γραμμές

Οι γραμμές αυτές συνδέονται σε ένα από μία ομάδα έξι CLBs. Λόγω της χαμηλής χωρητικότητας οι γραμμές αυτές ενδείκνυνται για τη μεταφορά υψίσυχνων σημάτων (βλ. Σχήμα 5.14(α)).

- Hex – γραμμές

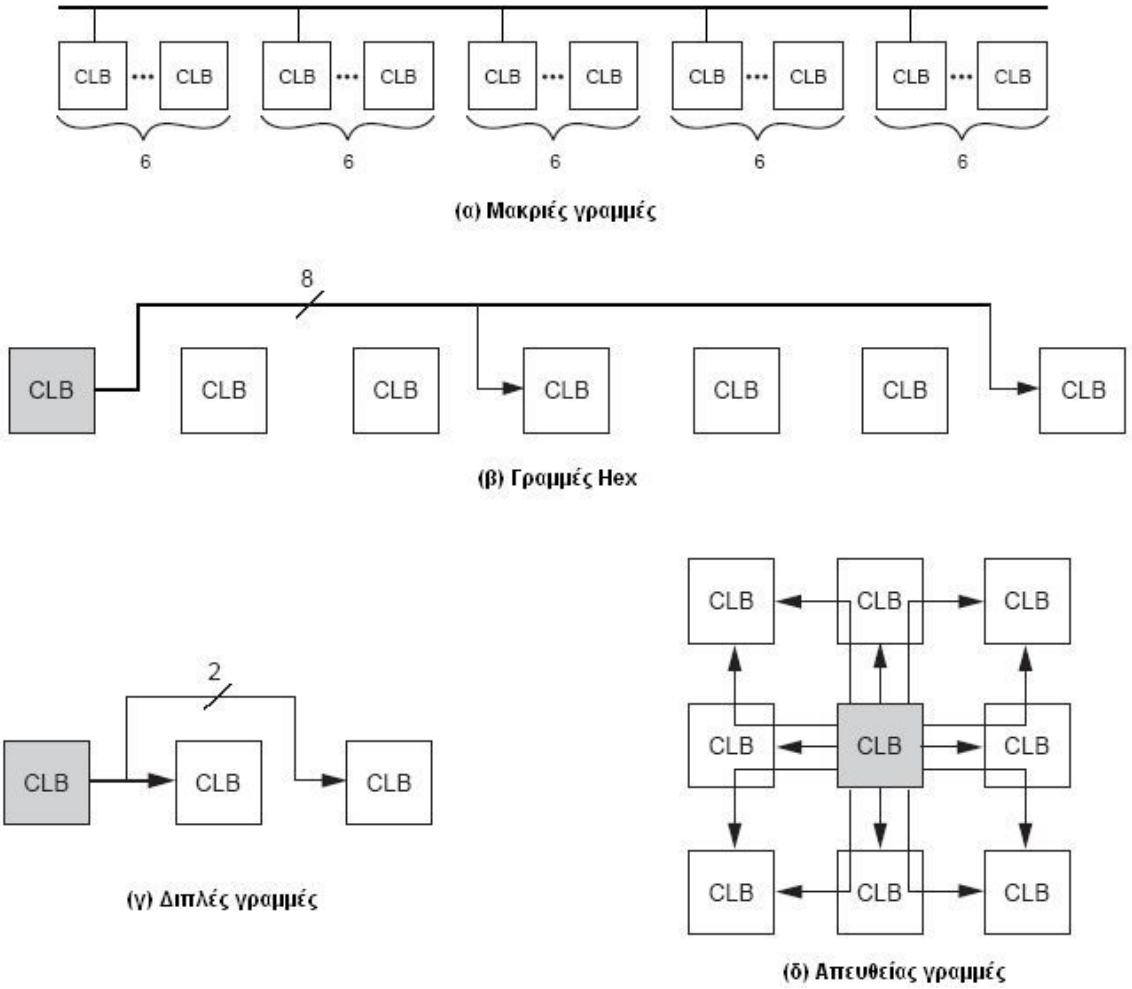
Οι γραμμές αυτές συνδέονται σε ένα από μία ομάδα τριών CLBs. Αυτές οι γραμμές, όσον αφορά τις δυνατότητές τους βρίσκονται μεταξύ των μακριών γραμμών και των διπλών γραμμών: Οι γραμμές Hex πλησιάζουν ως προς τα χαρακτηριστικά υψηλής συχνότητας τις μακριές γραμμές, ενώ την ίδια στιγμή προσφέρουν καλύτερη συνδεσιμότητα (βλ. Σχήμα 5.14(β)).

- Διπλές γραμμές

Οι γραμμές αυτές συνδέονται σε κάθε δεύτερο CLB. Συγκρινόμενες με τους τύπους που προαναφέραμε, οι διπλές γραμμές προσφέρουν έναν υψηλότερο βαθμό ελαστικότητας κατά τη διάρκεια των συνδέσεων (βλ. Σχήμα 5.14(γ)).

- Απευθείας γραμμές

Αυτές οι γραμμές προσφέρουν τη δυνατότητα απευθείας σύνδεσης οποιουδήποτε CLB με τα γειτονικά του CLB (βλ. Σχήμα 5.14(δ)).



Σχήμα 5.14: Διάφοροι τύποι διασύνδεσης

Κεφάλαιο 6

Σχεδίαση Ασαφούς Ελεγκτή

6.1 Εισαγωγή

Κάθε μορφή αυτοματισμού είναι στενότατα συνυφασμένη με τη δυνατότητα ελέγχου. Ο πιο σημαντικός ελεγκτής είναι ο κλασσικός PID ελεγκτής *(ελεγκτής τριών όρων)* [45]. Με την κατάλληλη ρύθμιση τιμών στα κέρδη του Αναλογικού *(Proportional)*, Ολοκληρωτικού *(Integral)* και Διαφορικού *(Derivative)* όρου καταφέρνουμε να ελέγξουμε ένα πλήθος συστημάτων για περιορισμένο εύρος εισόδων. Τα τελευταία χρόνια ωστόσο βρίσκουν όλο και περισσότερη εφαρμογή οι *ασαφείς ελεγκτές*, είτε σαν αυτοτελείς ελεγκτές είτε σαν μέρος υβριδικών συστημάτων ελέγχου [1][46][47][26][48][49][50]. Η υπεροχή αυτών, έναντι του κλασσικού PID ελεγκτή είναι έκδηλη, όσο η πολυπλοκότητα και το εύρος εισόδων του εξεταζόμενου συστήματος αυξάνει.

Η θεωρία των ασαφών συνόλων θεμελιώθηκε στην παρούσα της μορφή περίπου το 1965 από τον Καθηγητή αυτομάτου ελέγχου Lotfi Zadeh στο UC-Bercley [51]. Ο Zadeh διατύπωσε το πρόβλημα της αβεβαιότητας και της ανακρίβειας με την *αρχή του ασυμβιβάστου*:

«Καθώς η πολυπλοκότητα ενός συστήματος αυξάνει, η ικανότητά μας να προβαίνουμε σε ακριβείς και σημαντικές δηλώσεις για τη συμπεριφορά του μειώνεται μέχρι που να φτάσουμε σε ένα όριο (κατώφλι) πέρα από το οποίο ακρίβεια και σημαντικότητα (ή σχετικότητα) καθίστανται σχεδόν αμοιβαίως αποκλειόμενα χαρακτηριστικά»

Έτσι, επέκτεινε την κλασσική (Αριστοτελική-Boolean) λογική από το διακριτό σύνολο {0,1} στο συνεχές [0,1] εισάγοντας μια ομαλή μετάβαση από το "ανήκει" στο "δεν ανήκει". Στόχος του ήταν να παραστήσει τη συγκεχυμένη, αόριστη και μη ακριβής γνώση του ανθρώπου άμεσα (απευθείας) , χωρίς τη μεσολάβηση κάποιας τεχνικής παράστασης,

όπως λ.χ. ένας ακριβής μαθηματικός τύπος. Ο Zadeh ανέπτυξε τη θεωρία των ασαφών συνόλων σαν ένα τρόπο αντιμετώπισης προβλημάτων αλληλεπίδρασης μεταξύ ανθρώπων και μηχανών. Ωστόσο, στη συνέχεια δημιουργήθηκε ένας ολόκληρος κλάδος μαθηματικών γύρω από την ιδέα, ότι οποιαδήποτε μαθηματική δομή μπορεί να "ασαφοποιηθεί", δηλαδή να διατυπωθεί με τη βοήθεια ασαφών συνόλων. Έτσι, αναπτύχθηκαν μαθηματικοί κλάδοι όπως "*ασαφής τοπολογία"*, "*ασαφείς ομάδες"* κλπ.

Οι ιδέες του Zadeh είχαν άμεση απήχηση στην Ιαπωνική επιστημονική κοινότητα, κυρίως λόγω της ιδιοσυγκρασίας των Ιαπώνων. Σαν αποτέλεσμα, παρατηρήθηκε ραγδαία αύξηση της έρευνας και των εφαρμογών της ασαφούς λογικής στον τομέα του αυτομάτου ελέγχου. Η πρώτη πρακτική εφαρμογή έγινε από τον Mamdani το 1974 [7] στο Queen Mary College στον έλεγχο της λειτουργίας μιας ατμομηχανής. Ακολούθησε η πρώτη βιομηχανική εφαρμογή του ασαφούς ελεγκτή στον έλεγχο της ποιότητας παραγωγής το 1982 στην Κοπεγχάγη, από τον Δανό κατασκευαστή τσιμέντου F.L. Smidth. Το 1986 η Hitachi χρησιμοποίησε ασαφή έλεγχο στα αυτόματα τρένα της Sendai, Japan με αποτέλεσμα να μειωθεί η κατανάλωση ενέργειας κατά 10% και να αυξηθεί η ακρίβεια στο σταμάτημα κατά 10cm. Το 1989 ιδρύεται το *Laboratory for International Fuzzy Engineering (LIFE)*, μια συνεργασία 48 εταιριών για την προώθηση της έρευνας στον τομέα των ασαφών συστημάτων. Στην Αμερική ο ασαφής έλεγχος αρχικά περιορίστηκε στη μείωση της κατανάλωσης των μηχανών.

Η έρευνα και οι εφαρμογές του ασαφούς ελέγχου συνεχίζονται μέχρι και σήμερα. Η ασαφής λογική ταξινομείται πλέον ως ένα από τα τρία βασικά δομικά στοιχεία (πεδία) της *Υπολογιστικής Νοημοσύνης*. Τα άλλα δύο είναι τα *νευρωνικά δίκτυα* και οι *εξελικτικοί αλγόριθμοι* που αναπτύχθηκαν μετέπειτα. Έτσι, εμφανίστηκαν *υβριδικά συστήματα* που συνδυάζουν τα τρία πεδία (π.χ. «*νεύρο-ασαφής ελεγκτής*»), αξιοποιώντας τις δυνατότητες του καθενός.

Ένας ασαφής ελεγκτής σε *τσιπ (ασαφές τσιπ)* πρέπει να έχει τη δυνατότητα να αποθηκεύει και να επεξεργάζεται ασαφείς κανόνες. Τα ασαφή τσιπ μπορεί να είναι *αναλογικά* ή *ψηφιακά*. Το πρώτο ψηφιακό ασαφές τσιπ κατασκευάστηκε το 1984 στα AT & T Bell Laboratories από τους Masaki Togai και Hiroyuki Watanabe [52] επεξεργαζόταν *16* απλούς κανόνες σε *12.5us* και ήταν ο προπομπός για κάποιες άλλες ενδιαφέρουσες υλοποιήσεις που ακολούθησαν [53][54]. Επιπλέον ψηφιακές υλοποιήσεις ασαφών ελεγκτών αναφέρονται στις [55][56] σύμφωνα με τη βιβλιογραφία. Η κατασκευή αναλογικών ασαφών τσιπ ξεκίνησε με τον Takeshi Yamakawa το 1987 [57][58].

Στο παρόν κεφάλαιο, εξετάζεται η κατασκευή ενός ψηφιακού ασαφούς τσιπ. Επιλέχθηκε η ψηφιακή υλοποίηση λόγο των ισχυρών εργαλείων που προσφέρει η ψηφιακή τεχνολογία έναντι της αναλογικής, που εξασφαλίζουν αξιοπιστία, παραμετρικότητα και ταχύτητα. Το σημαντικότερο ωστόσο πλεονέκτημα των αναλογικών τσιπ είναι ότι δεν έχουν ανάγκη από A/D και D/A μετατροπείς που περιορίζουν την ταχύτητα ενός ψηφιακού τσιπ και ανεβάζουν το κόστος του. Χρησιμοποιώντας ψηφιακή σχεδίαση και FPGA* τσιπ δίνεται η δυνατότητα του εύκολου *prototyping* με μικρό κόστος, χωρίς την ανάγκη να παραχθούν

* **F**ield **P**rogrammable **G**ate **A**rray

μάσκες όπως στην περίπτωση των ASIC*. Η περιγραφή του κυκλώματος έγινε σε γλώσσα VHDL† [59][60], η οποία προσφέρει δυνατότητα συγγραφής παραμετρικού κώδικα και υποστηρίζεται από τα περισσότερα εργαλεία σύνθεσης.

Στην παρούσα εργασία παρουσιάζετε η σχεδίαση ενός παραμετρικού ψηφιακού ελεγκτή ασαφούς λογικής (DFLC‡). Με τον όρο παραμετρικό εννοούμε ότι το DFLC επιτρέπει την κλιμάκωση και μπορεί να διαμορφωθεί ως προς τον αριθμό εισόδων και εξόδων, αριθμός τριγωνικών ή τραπεζοειδών ασαφών συνόλων ανά είσοδο, αριθμό ασαφών συνόλων καθορισμένης τιμής (*singletons*) ανά έξοδο, μέθοδος υπολογισμού βαθμού ενεργοποίησης των προϋποθέσεων του κανόνα (t-norm, s-norm), τύπος διαιρέτη και αριθμό pipeline registers. Αυτή η παραμετροποίηση επιτρέπει τη δημιουργία ενός γενικού ασαφούς πυρήνα ελεγκτών και μπορεί να χρησιμοποιηθεί για να παραγάγει ασαφείς επεξεργαστές διαφορετικών προδιαγραφών χωρίς την ανάγκη εξολοκλήρου επανασχεδιασμού του πυρήνα από την αρχή. Η παρούσα αρχιτεκτονική επεξεργαστών ασαφούς λογικής προϋποθέτει μέγιστη επικάλυψη δύο ασαφών συνόλων μεταξύ των παρακείμενων ασαφών συνόλων και απαιτεί 2^n κύκλους ρολογιών (input data processing rate), όπου το n είναι ο αριθμός εισόδων, δεδομένου ότι επεξεργάζεται έναν ενεργό κανόνα ανά κύκλο ρολογιού. Η αρχιτεκτονική της σχεδίασης επιτρέπει την επίτευξη ταχύτητας συχνότητας πυρήνα 100 MHz, ενώ τα δεδομένα εισόδου μπορούν να υποστούν δειγματοληψία με ρυθμό ρολογιού ίσο με $1/2^n$ της ταχύτητας συχνότητας του πυρήνα (100 MHz), επεξεργάζοντας μόνο τους ενεργούς κανόνες. Για να επιτύχει αυτό το αποτέλεσμα της ταχύτητας το latency της αρχιτεκτονικής του τσιπ περιλαμβάνει 11 βαθμίδες αγωγού (pipelines), όπου η καθεμία απαιτεί 10 ns. Το προτεινόμενο DFLC είναι βασισμένο σε έναν απλό αλγόριθμο παρόμοιο με των Takagi-Sugeno μηδενικού τύπου συμπεράσματος και της σταθμισμένης μέσης μεθόδου από-ασαφοποίησης, και με βάση τις επιλεγμένες παραμέτρους χρησιμοποιεί τέσσερις εισόδους των 12-bit και μια έξοδο 12-bit, με μέχρι 7 τραπεζοειδείς ή τριγωνικές συναρτήσεις συμμετοχής ανά είσοδο με 8-bit ανάλυση για το βαθμό αλήθειας και βάση κανόνων μέχρι 2401 κανόνες [61].

6.2 Μέθοδος Takagi-Sugeno

Το DFLC που αναλύεται σε αυτό το κεφάλαιο είναι βασισμένο στο ασαφές μοντέλο εξαγωγής συμπερασμάτων των Takagi-Sugeno μηδενικού τύπου [62][27]. Είναι γνωστό ότι το T-S μοντέλο μπορεί να παρέχει μια αποτελεσματική αντιπροσώπευση των σύνθετων μη γραμμικών συστημάτων όσον αφορά τα ασαφή σύνολα και τον ασαφή συλλογισμό. Η μέθοδος T-S είναι πραγματικά αρκετά απλή καθώς οδηγεί σε γρήγορους υπολογισμούς και είναι σχετικά εύκολο να εφαρμοστεί. Επιπλέον, ένας ασαφής ελεγκτής βασισμένος στη μέθοδο T-S παρέχει μια καλή λύση μεταξύ της απλότητας υλικού και της αποδοτικότητας του ελέγχου. Στον κανόνα συμπεράσματος T-S, το συμπέρασμα εκφράζεται υπό μορφή γραμμικών συναρτήσεων.

---

* **A**pplication **S**pecific **I**ntegrated **C**ircuit
† **V**ery High Speed Integrated Circuit (VHSIC) **H**ardware **D**escription **L**anguage
‡ **D**igital **F**uzzy **L**ogic **C**ontroller

Οι T-S MIMO κανόνες έχουν την παρακάτω μορφή:

***R*$_i$:** IF x_1 IS A^i_1 AND x_2 IS A^i_2 AND… AND x_n IS A^i_n THEN y^i_1 AND y^i_2 AND… AND y^i_l

όπου $y^i_j = c^i_{0j} + c^i_{1j} x_1 + \ldots + c^i_{nj} x_n$

$i = 1, 2, \ldots, m$ όπου m είναι ο συνολικός αριθμός κανόνων, x_k, $k = 1, 2, \ldots, n$ εκφράζει την $k^{οστή}$ μεταβλητή εισόδου, A^i_k είναι τα ασαφή σύνολα εισόδου, y^i_j, $j = 1, 2, \ldots, l$ εκφράζει την $j^{οστή}$ μεταβλητή εξόδου του $i^{οστού}$ κανόνα, και c^i_{kj} είναι όλες οι σταθερές. Ο παραπάνω κανόνας είναι γλωσσικά (και λογικά) ισοδύναμος με έναν αριθμό MISO κανόνων, όπως:

***R*$_i$:** IF x_1 IS A^i_1 AND x_2 IS A^i_2 AND… AND x_n IS A^i_n THEN y^i_1

***R*$_i$:** IF x_1 IS A^i_1 AND x_2 IS A^i_2 AND… AND x_n IS A^i_n THEN y^i_j

Το λογικό «AND» στο συμπέρασμα του MIMO κανόνα υπάρχει επίσης και στην περίπτωση των MISO κανόνων, διότι και στους δύο ασαφής συλλογισμούς και το ένα σύνολο κανόνων είναι αληθές «AND (ΚΑΙ)» το άλλο είναι επίσης αληθές [47]. Το μοντέλο μηδενικού τύπου προκύπτει (απλοποιημένη T-S συναρτησιακή λογική) αν στα συμπεράσματα των κανόνων εξόδου χρησιμοποιηθούν σταθερές c^i_{0j} (μονότιμα σύνολα) και επομένως $y^i_j = c^i_{0j}$. Στο T-S μοντέλο, ο συλλογισμός με έναν αριθμό κανόνων διεξάγεται κανονικά, συνδέοντας τον κάθε κανόνα με ένα βαθμό ενεργοποίησης, αλλά η κάθε έξοδος είναι γραμμικά εξαρτώμενη με τις εισόδους. Η έξοδος του κάθε κανόνα είναι ένα κινούμενο ασαφές σύνολο μοναδιαίας τιμής και η από-ασαφοποιημένη έξοδος είναι το σταθμισμένο μέσο της συνεισφοράς του κάθε κανόνα, όπως φαίνεται από την παρακάτω σχέση,

$$y_j = \frac{\sum_{i=1}^{m} w^i y^i_j}{\sum_{i=1}^{m} w^i} \tag{6.1}$$

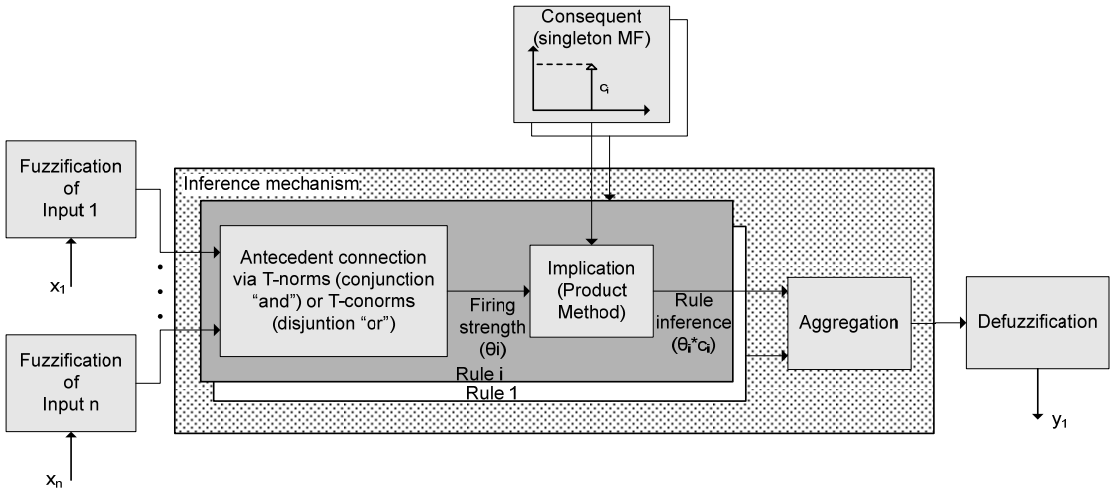
όπου w^i είναι η συνεισφορά (*βάρος - weight*) του αριστερού μέρους του $i^{οστού}$ κανόνα. Η μέθοδος σύνδεσης «AND» με χρήση αλγεβρικού γινομένου δίνεται από τη σχέση,

$$w^i = \prod_{k=1}^{n} \mu_{A^i_k}(x_k) \tag{6.2}$$

και με χρήση ελαχίστου από τη σχέση,

$$w^i = \min_{k=1}^{n} \mu_{A^i_k}(x_k) \tag{6.3}$$

Ο T-S μηχανισμός φαίνεται στο σχήμα που ακολουθεί (βλ. Σχήμα 6.1).



Σχήμα 6.1: Μηχανισμός Takagi-Sugeno

6.3 Χαρακτηριστικά DFLC

Τα χαρακτηριστικά του DFLC (βλ. Πίνακα 6.2) που παρουσιάζεται στο παρόν κεφάλαιο καθορίζονται με βάση τις επιλεγμένες τιμές (*generic parameters*) που ορίζονται σε ένα VHDL αρχείο παραμέτρων (parameters package). Οι παράμετροι του DFLC πυρήνα περιγράφονται στον Πίνακα 6.1.

Fuzzy Inference System (FIS) type	**Takagi-Sugeno zero-order type**
Inputs	4
Input resolution	12-bit
Outputs	1
Output resolution	12-bit
Antecedent Membership Functions (MF's)	7 Triangular or Trapezoidal shaped per fuzzy set
Antecedent MF degree of truth (α value) resolution	8-bit
Consequent MF's	2401 Singleton type
Consequent MF resolution	8-bit
Maximum number of fuzzy inference rules	2401 (number of fuzzy sets number. of inputs)
AND method	MIN (T-norm operator implemented by minimum)
Implication method	PROD (product operator)
MF overlapping degree	2
Defuzzification method	Weighted average

Πίνακας 6.1: Χαρακτηριστικά του DFLC πυρήνα

Παράμετροι (VHDL generics)	**Τιμή**	**Περιγραφή παραμέτρου**
ip_no	4	number of inputs
ip_sz	12	input bus width (bits)
op_no	1	number of outputs
op_sz	8	output bus width (bits)
FS_no	7	number of membership functions
dy	8	dy degree of truth width
sel_op	0	antecedent method connection: 0 : min, 1: prod, 2: max, 3: probor
div_type	0	divider model type 0: restoring array, 1 : LUT reciprocal approximation [63]
Path Synchronization Registers		*Signal Path Route*
psr1_no	1	ip_set→psr1_no→trap_gen_p
psr2_no	4	s_rom→psr2_no→mult
psr3_no	2	s_rom→psr3_no→rul_sel_p
psr3_no	0	cpr5→psr→int_uns
Component Pipeline Registers		*Component (Entity) Name*
cpr1_no	1	addr_gen_p
cpr2_no	1	cons_map_p
cpr3_no	3	trap_gen_p
cpr4_no	0	rule_sel_p
cpr5_no	2	minmax_p
cpr6_no	2	mult
cpr7_no	2	int_uns
cpr8_no	0	int_sig
cpr9_no	3	div_array
cpr2_no	1	cons_map_p

Πίνακας 6.2: Παράμετροι που επιλέχθηκαν για το DFLC

Οι αναγκαίες παράμετροι αντλούνται από το αρχείο των παραμέτρων στα generics της κάθε οντότητας (*entity*) των παραμετροποιημένων δομικων μονάδων που συνθέτουν την ιεραρχία του πυρήνα του ασαφούς ελεγκτή. Επιπλέον, στο ίδιο αρχείο παραμέτρων αρχείο ορίζονται και ο αριθμός των βαθμίδων αγωγού (*pipeline stages*) κάθε λειτουργικού μπλοκ του πυρήνα, καθώς και οι καταχωρητές συγχρονισμού μονοπατιού (*path synchronization registers*).

6.4 Υλοποίηση του DFLC σε Hardware

Σε αυτήν την ενότητα επιχειρείται μια αναλυτική παρουσίαση των διάφορων ιεραρχικών μονάδων του DFLC και επιπλέον εξηγείται ο τρόπος με τον οποίο η καθεμία από αυτές τις μονάδες έχει σχεδιαστεί παραμετρικά ούτως ώστε όλες μαζί να αποτελέσουν έναν εξ' ολοκλήρου παραμετρικό πυρήνα.

6.4.1 Αρχιτεκτονική του DFLC

Η αρχιτεκτονική του DFLC πυρήνα που μελετάται στην παρούσα εργασία φαίνεται στο Σχήμα 6.2. Τα ακόλουθα τρία κύρια ιεραρχικά μπλοκ συνθέτουν τη συγκεκριμένη αρχιτεκτονική:

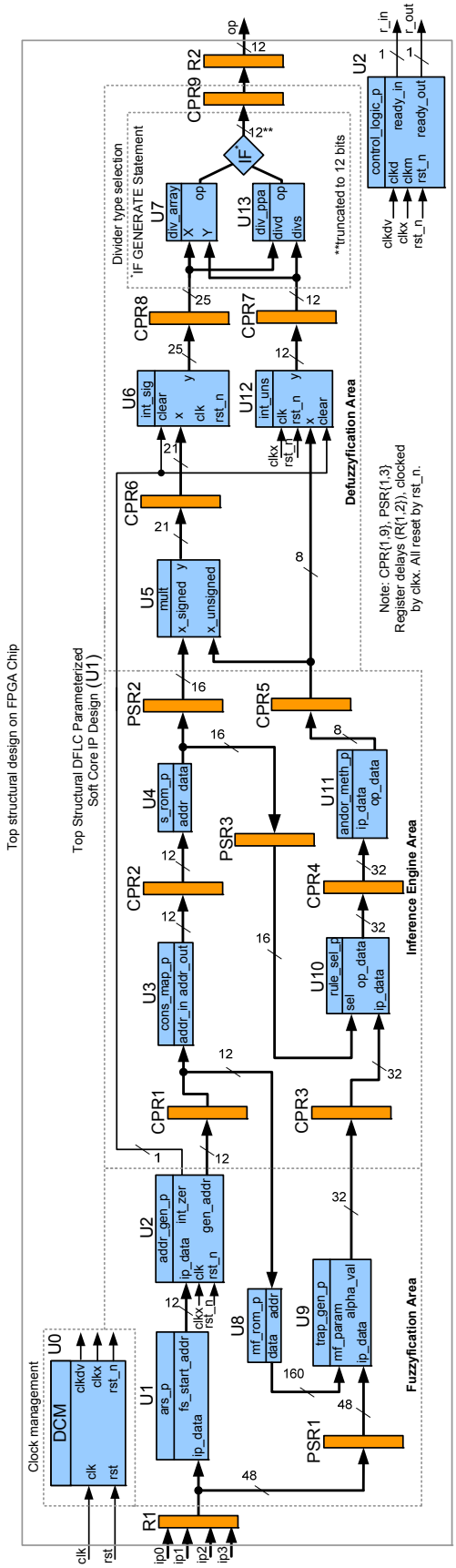
- Ασαφοποιητής – Fuzzifier,
- Συλλογιστική μηχανή – Inference
- Από-ασαφοποιητής – Defuzzifier

Τα «CPR» μπλοκ στο Σχήμα 6.2 αντιπροσωπεύουν τον αριθμό (καταχωρητών) των βαθμίδων αγωγού (*pipeline stages*) του κάθε component (*Component Pipeline Registers*), τα «PSR» μπλοκ σχετίζονται με τους καταχωρητές συγχρονισμού του μονοπατιού (*Path Synchronization Registers*), ενώ τα «U» μπλοκ αντιπροσωπεύουν τα διάφορα λειτουργικά τμήματα του DFLC.

Το μπλοκ διαχείρισης ρολογιού ή «DCM*» (*Digital Clock Manager*), είναι ένα από τα τέσσερα διαθέσιμα στην FPGA βιβλιοθήκη που επιλέχθηκε να χρησιμοποιηθεί (Spartan-3 1500-4FG676) [44]. Το «DCM» μπλοκ οδηγείται από ένα ρολόι (*clk*) των 75 MHz (από τον ταλαντωτή επάνω στην πλακέτα η οποία φιλοξενεί το FPGA) και είναι διαμορφωμένο έτσι ώστε να παράγει ένα διαιρεμένο σήμα ρολογιού (*clkdv*) στα 6.25 MHz ($75 \div 12$) και ένα πολλαπλασιασμένο σήμα ρολογιού (*clkfx*), το οποίο είναι 100 MHz (6.25×16) και προορίζεται για τη λειτουργία του πυρήνα του DFLC (DFLC Core, βλ. Παράρτημα Α.1).

Σε αυτό το σημείο αξίζει να σημειώσουμε ότι οι παραπάνω συχνότητες μπορούν να διαφοροποιηθούν, αλλάζοντας τις γενικές παραμέτρους (generic parameters) του «DCM». Στην ανώτατη δομική οντότητα (top structural entity) του design (που δια-συνδέει τα FC Core, DCM, R1, R2 και το control logic – βλ. Σχήμα 6.2), υλοποιείται μία μηχανή καταστάσεων (state machine) που λειτουργεί ως ελεγκτής διακοπών (*control_logic_p*). Σκοπός του είναι να παράγει δυο σήματα διακοπών εισόδου-εξόδου (I/O interrupts) που αναφέρονται ως «*ready_in*» και «*ready_out*». Τα σήματα αυτά χρησιμεύουν στο να ειδοποιούν εξωτερικές συσκευές ότι ο DFLC πυρήνας είναι σε κατάσταση να δεχτεί ή να στείλει δεδομένα ή το αντίθετο (handshaking).

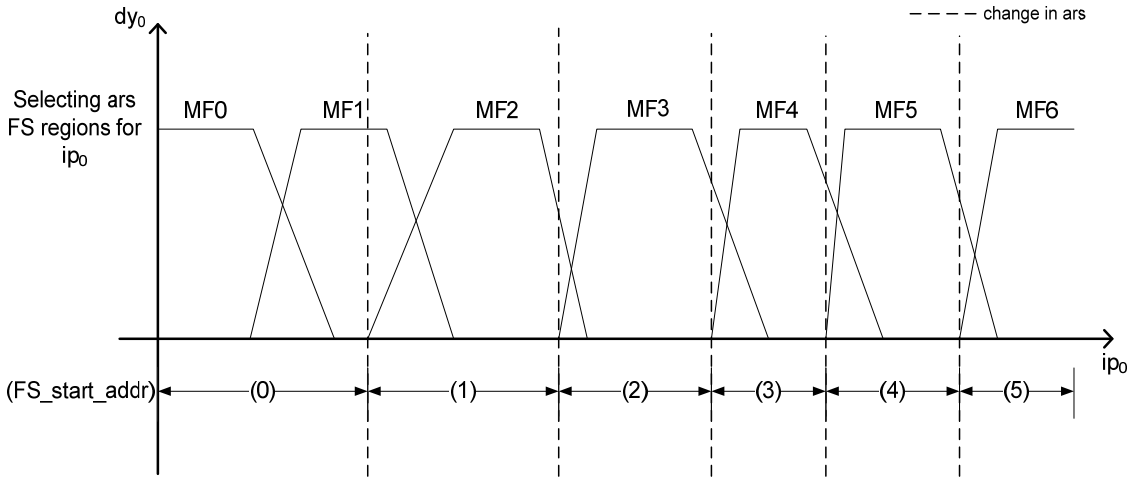
* Τα DCM μπλοκ προσφέρουν αυτό-βαθμονομούμενες (self-calibrating), πλήρως ψηφιακές λύσεις για διανομή, καθυστέρηση, πολλαπλασιασμό, διαίρεση, και ολίσθηση φάσης σημάτων ρολογιού.



Σχήμα 6.2: Αρχιτεκτονική DFLC

6.4.2 Επιλογή Ενεργών Κανόνων

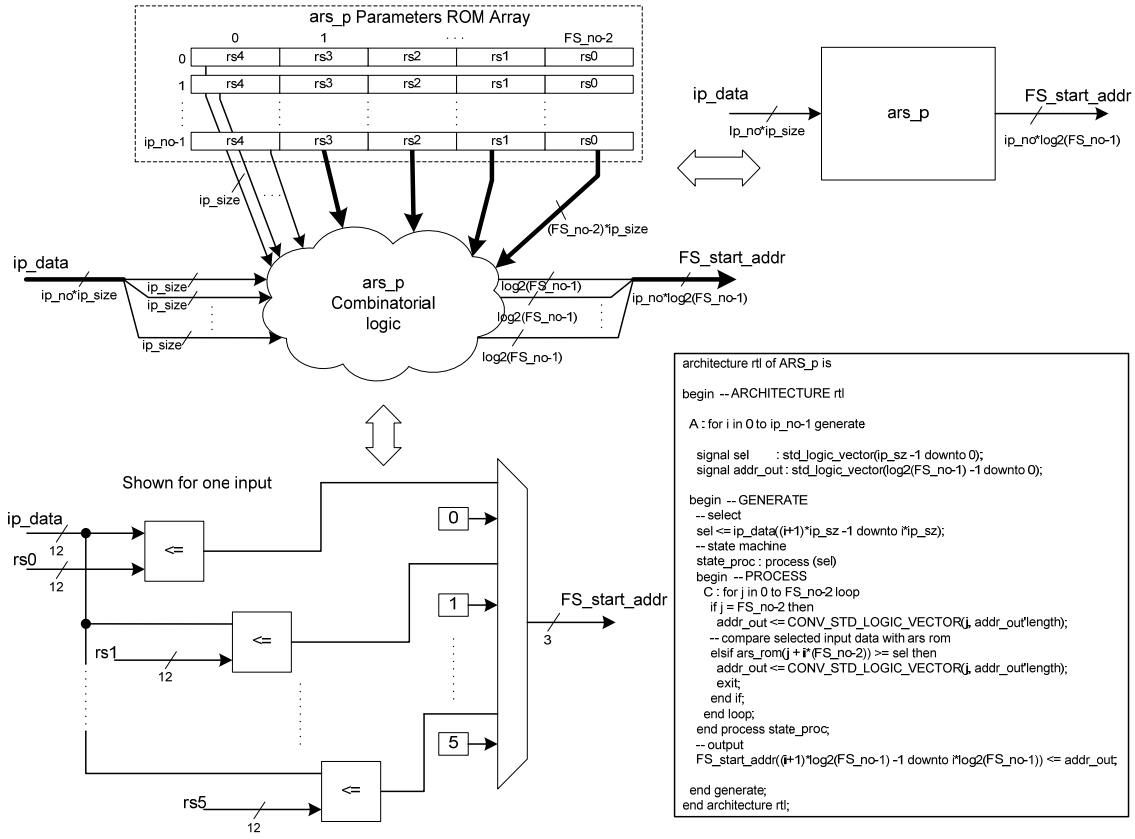
Αντί να επεξεργαζόμαστε όλους τους συνδυασμούς κανόνων για κάθε καινούργιο σύνολο δεδομένων εισόδου, έχουμε επιλέξει να επεξεργαζόμαστε μόνο τους ενεργούς κανόνες, δηλαδή τους κανόνες εκείνους που έχουν μη μηδενική συμβολή στο τελικό αποτέλεσμα. Για να αντιμετωπιστεί το παραπάνω πρόβλημα και δεδομένου ότι η επικάλυψη μεταξύ των γειτονικών ασαφών συνόλων είναι της τάξης του 2, υλοποιήσαμε ένα μπλοκ επιλογής ενεργών κανόνων (active rule selection – *ars_p*) που σκοπό έχει να υπολογίζει την περιοχή του ασαφούς συνόλου στην οποία τα δεδομένα εισόδου αντιστοιχούνται σε αυτό. Αυτό έχει ως αποτέλεσμα στη περίπτωση του DFLC με 4 εισόδους και 7 συναρτήσεις συμμετοχής σε κάθε είσοδο, αντί να χρειάζεται να επεξεργαστούν όλοι οι 2401 συνδυασμοί κανόνων που περιέχονται στη βάση κανόνων, να επεξεργάζονται μόνο οι 16 ενεργοί κανόνες, οι οποίοι όπως εξηγήσαμε και παραπάνω συμβάλουν στο τελικό αποτέλεσμα. Στο Σχήμα 6.3 που παρατίθεται παρακάτω, φαίνεται η κωδικοποίηση των ασαφών συνόλων που χρησιμοποιείται για τις εισόδους ($ip_{0,\ldots,4}$), όπως επίσης και η διάσπαση της περιοχής των ασαφών συνόλων που προκύπτει από την επαναλαμβανόμενη σύγκριση των εισόδων με τα σημεία ανόδου ($rise\_start_{0,\ldots,4}$) των συναρτήσεων συμμετοχής, που συνθέτουν τα ασαφή σύνολα. Είναι προφανές ότι εφόσον χρησιμοποιούμε επικάλυψη δύο (συναρτήσεων συμμετοχής) μεταξύ των γειτονικών συναρτήσεων συμμετοχής, η σύγκριση της $1^{ης}$ και $2^{ης}$ συνάρτησης συμμετοχής στο ασαφές σύνολο δεν είναι απαραίτητη. Αυτό συμβαίνει αφού το αρχικό ζευγάρι συναρτήσεων συμμετοχής είναι πάντα το $1^{ο}$ και το $2^{ο}$ και έτσι απλά χρειάζεται να ολισθήνουμε δεξιά στην επόμενη συνάρτηση συμμετοχής όταν υπερβούμε το σημείο ανόδου (*rise_start*) της $3^{ης}$ συνάρτησης συμμετοχής. Τα τμήματα μετάβασης στο ασαφές σύνολο απεικονίζονται στο Σχήμα 6.3 με διακεκομμένη γραμμή που αναγράφεται ως «*change in ars*». Γενικά το «$FS\_start\_addr_{0,\ldots,4}$» δείχνει το ενεργό ζεύγος συναρτήσεων συμμετοχής, όπου στο συγκεκριμένο παράδειγμα το εύρος εξόδου του είναι από το 0 έως το 5.



Σχήμα 6.3: Κωδικοποίηση συναρτήσεων συμμετοχής και περιοχές διευθύνσεων

Το Σχήμα 6.4 αποτελεί ένα αναλυτικό διάγραμμα για το παραμετρικό *ars_p* μπλοκ. Επιπλέον, φαίνεται η αρχιτεκτονική του συγκεκριμένου μπλοκ, και παρατίθεται ένα κομμάτι VHDL κώδικα που αντικατοπτρίζει των τρόπο γραφής ούτως ώστε να επιτευχθεί η παραμετροποίηση του μπλοκ.

Η παραμετροποίηση σε μεγάλο βαθμό επιτυγχάνεται κάνοντας χρήση της δυνατότητας που παρέχει η VHDL γλώσσα, generate εντολών στο RTL (*Register Transfer Level*) σώμα της αρχιτεκτονικής του μπλοκ, καθώς και τη χρήση generics στην οντότητα (*entity*) του μπλοκ, που με τη σειρά τους χρησιμοποιούνται για να ορίσουν τις παραμέτρους του.



Σχήμα 6.4: Αρχιτεκτονική του *ars_p* μπλοκ

6.4.3 Περιγραφή της Γεννήτριας Διευθύνσεων

Η γεννήτρια διευθύνσεων (*address generator - addr_gen_p*) παράγει στην έξοδο της, τις διευθύνσεις εκείνες που υποδεικνύονται από το *ars_p* μπλοκ και χρειάζονται για τους ενεργούς ασαφείς κανόνες (*gen_addr_p*). Εφ' όσον το *ars_p* μπλοκ έχει προηγουμένως αναγνωρίσει όλες τις περιλαμβανόμενες συναρτήσεις συμμετοχής κατευθείαν (πλήρως συνδυαστικό κύκλωμα), το *addr_gen_p* μπλοκ παράγει με τη σειρά του ανά περίοδο ρολογιού τις διευθύνσεις εκείνες που αντιστοιχούν στους ενεργούς κανόνες. Το *addr_gen_p* μπλοκ για 16 ενεργούς κανόνες όπως μελετάτε στη παρούσα εργασία, απαιτεί 16 περιόδους ρολογιού για να ολοκληρώσει τη διαδικασία παραγωγής όλων των ενεργών κανόνων [64]. Συνεπώς, η περίοδος δειγματοληψίας των δεδομένων εισόδου πρέπει να είναι 16 φορές η περίοδος του εσωτερικού ρολογιού λειτουργίας του DFLC. Στο Σχήμα 6.5 διακρίνεται το σήμα *int_zer* που χρησιμοποιείται σαν σημαία μηδενισμού για τα μπλοκ ολοκλήρωσης, ούτως ώστε μόλις ολοκληρωθεί η επεξεργασία και των 16 ενεργών κανόνων να αποφευχθεί η περίπτωση υπερχείλισης των ολοκληρωτών. Το προαναφερθέν σήμα εξαρτάται άμεσα από το μπλοκ generic *int_zer_delay* και δίνεται από την παρακάτω σχέση,

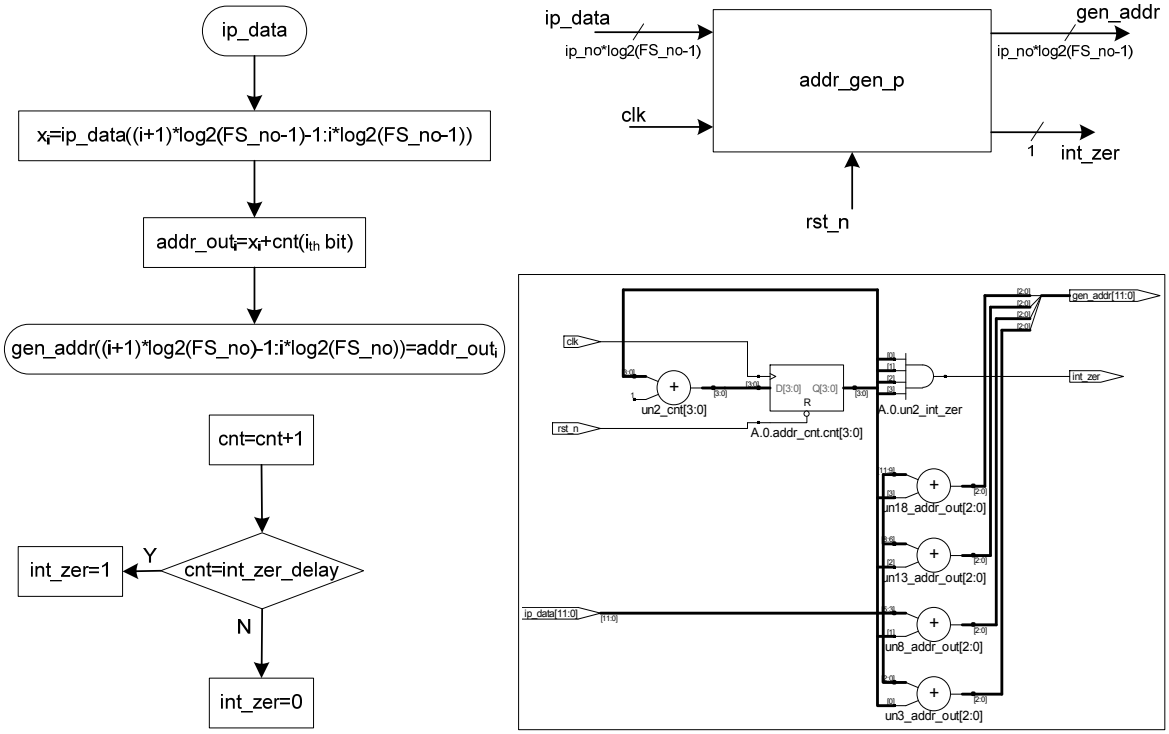
$$int\_zer\_delay = cpr1\_no + cpr2\_no + psr2\_no + cpr6\_no\text{-}1 \quad (6.4)$$

ή στο συγκεκριμένο design με βάση τις παραμέτρους του Πίνακα 6.2 έχει την τιμή 7.

Σε αυτό το σημείο θα πρέπει να διευκρινιστεί ότι το -1 που εμφανίζεται στη παραπάνω σχέση οφείλεται στον τρόπο υλοποίησης των ολοκληρωτών, οι οποίοι δουλεύουν στην αρνητική παρυφή του ρολογιού (σε αντίθετη περίπτωση, δηλαδή λειτουργίας στη θετική παρυφή του ρολογιού, το *int_zer_delay* = 8).

Το σήμα *cnt* δημιουργείται από έναν αυξητικό μετρητή (*incremental counter*) *ip_no* καταστάσεων με δείκτη $i \in [0, ip\_no - 1]$.

Τέλος στο Σχήμα 6.5 φαίνεται και το σχηματικό διάγραμμα ως αποτέλεσμα της σύνθεσης της RTL περιγραφής του μπλοκ.



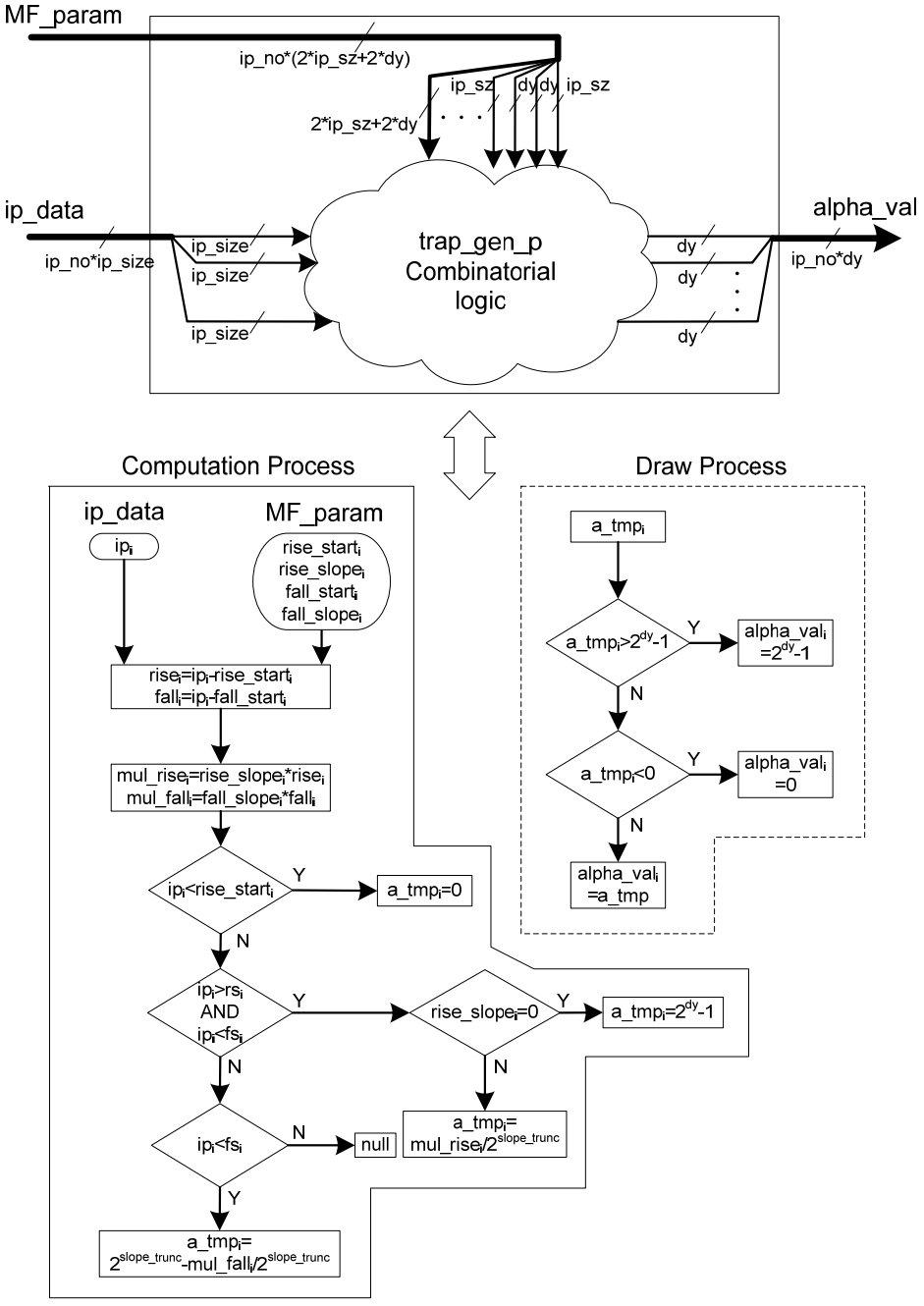
Σχήμα 6.5: Αλγόριθμος αρχιτεκτονικής του *trap_gen_p* μπλοκ και αποτέλεσμα RTL σύνθεσης

6.4.4 Γεννήτρια Τριγωνικών/Τραπεζοειδών συναρτήσεων συμμετοχής

Στην παρούσα DFLC αρχιτεκτονική που εξετάζουμε, απαιτούνται τέσσερα *trap_gen_p* μπλοκ που έχουν σαν εισόδους, τις εισόδους του ελεγκτή ($ip_{0,\ldots,3}$), και επιπλέον το σημείο ανόδου (*rise_start*), την κλίση του (*rise_slope*), το σημείο καθόδου (*fall_start*) και την κλίση του (*fall_slope*). Αυτά λαμβάνονται από την αντίστοιχη μνήμη ROM στην οποία φυλάσσονται οι παράμετροι που προορίζονται για τις εισόδους που περιγράφηκαν προηγουμένως. Οι παράμετροι αυτοί έχουν μορφή συνδεδεμένων σημάτων (*concatenated signals*), *MF_param* (βλ. Σχήμα 6.6), για το 1^{ο} έως και το 4^{ο} *trap_gen_p* μπλοκ αντίστοιχα.

Το σήμα εξόδου με ονομασία «*alpha_val*» αντιπροσωπεύει το βαθμό αληθείας της τρέχουσας συνάρτησης συμμετοχής μέσα στο ασαφές σύνολο. Το συγκεκριμένο μπλοκ όπως φαίνεται και στο διάγραμμα ροής του αλγορίθμου δεν κάνει χρήση διαίρεσης, με αποτέλεσμα να μην αυξάνει σε μεγάλο βαθμό την πολυπλοκότητα του DFLC.

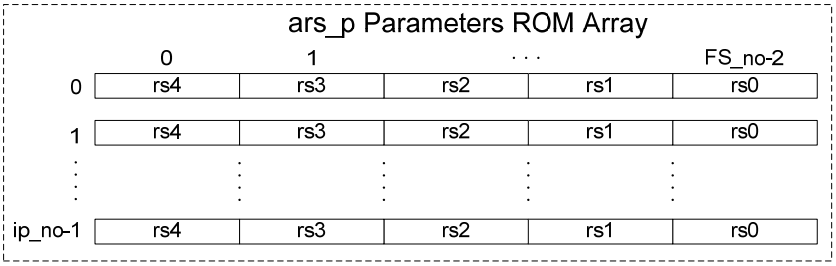
Το διάγραμμα ροής της γεννήτριας τραπεζοειδών συναρτήσεων συμμετοχής (*trap_gen_p*) φαίνεται στο Σχήμα 6.6.



Σχήμα 6.6: Αρχιτεκτονική γεννήτριας τραπεζοειδών συναρτήσεων συμμετοχής

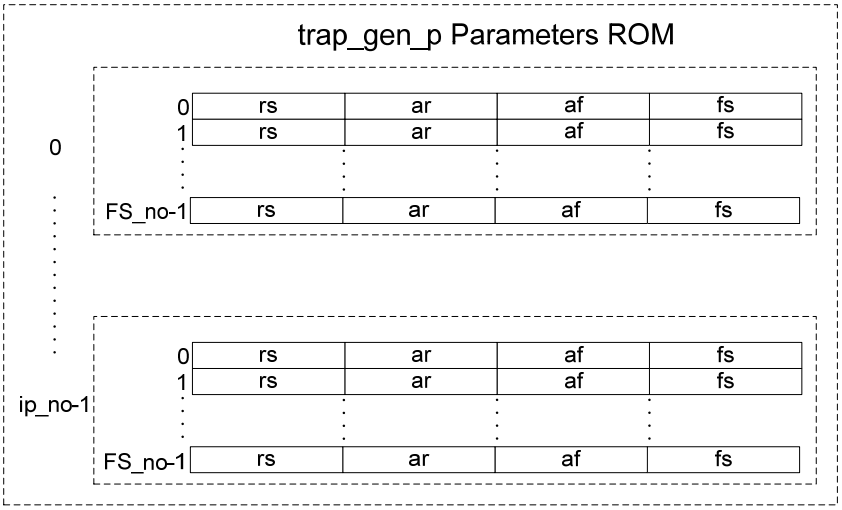
6.4.5 Οργάνωση της Μνήμης

Η δομή του πίνακα της μνήμης ROM που φυλάσσει τις παραμέτρους για το *ars_p* μπλοκ φαίνονται στο Σχήμα 6.7.



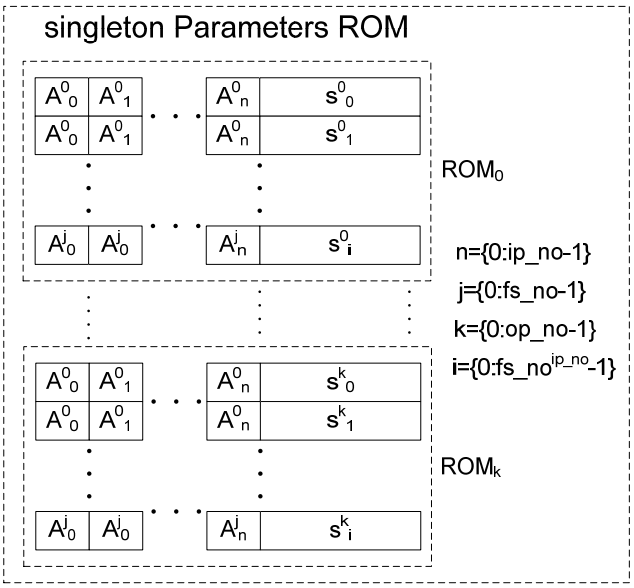
Σχήμα 6.7: Οργάνωση παραμέτρων του πίνακα μνήμης του *ars_p* μπλοκ

Ακολουθεί το Σχήμα 6.8 που δείχνει την οργάνωση των πινάκων μνήμης για το *trap_gen_p* μπλοκ.



Σχήμα 6.8: Οργάνωση παραμέτρων των πινάκων μνήμης του *trap_gen_p* μπλοκ

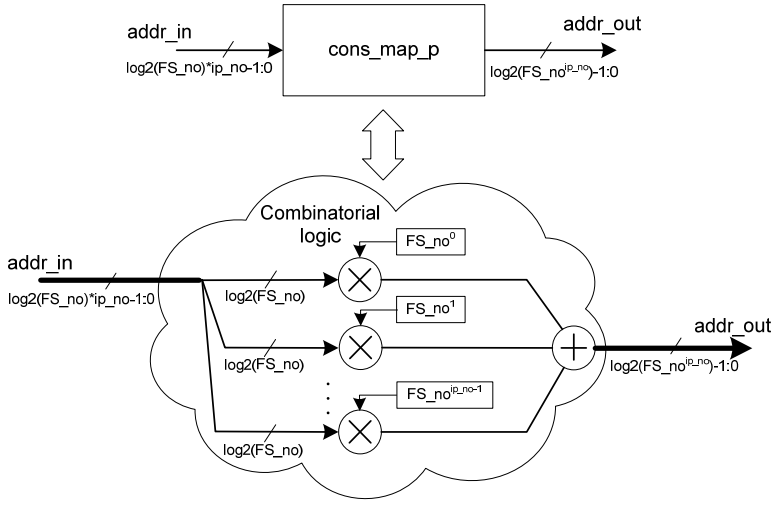
Η οργάνωση της μνήμης ROM (*s_rom_p*) που περιέχει τα ασαφή σύνολα μοναδιαίας τιμής (singletons) αποτυπώνεται στο Σχήμα 6.9.



Σχήμα 6.9: Οργάνωση παραμέτρων πίνακα μνήμης ασαφών συνόλων μοναδιαίας τιμής

6.4.6 Consequent Address Mapper

Στο Σχήμα 6.10 φαίνεται το consequent address mapper* μπλοκ (*cons_map_p*) το οποίο χρησιμεύει στο να παράγει τις διευθύνσεις (*addr_out*) για τη μνήμη ROM που κρατάει τα ασαφή σύνολα μοναδιαίας τιμής, συλλέγοντας τις παραγόμενες διευθύνσεις (*addr_in*) που προέρχονται από το *addr_gen_p* μπλοκ. Ο δίαυλος εισόδου *addr_in* πολλαπλασιάζετε με τον αριθμό των συναρτήσεων συμμετοχής (στην παρούσα υλοποίηση είναι 7) του ασαφούς συνόλου κάθε εισόδου του DFLC.



Σχήμα 6.10: Τμήμα διευθυνσιοδότησης συμπερασμάτων (cons_map_p)

* Τμήμα διευθυνσιοδότησης συμπερασμάτων

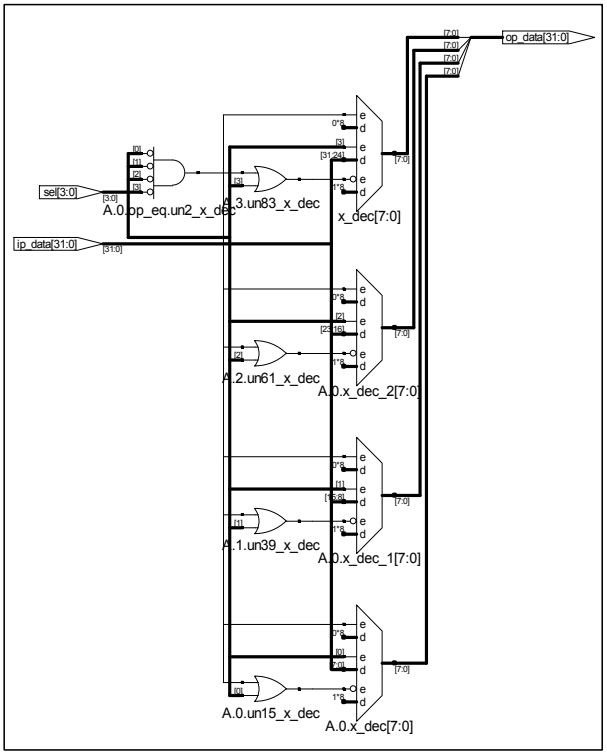
6.4.7 Επιλογή Προϋποθέσεων Κανόνων

Ο σκοπός αυτού του μπλοκ επιλογής είναι να επιτρέψει την επιλογή των επιθυμητών κανόνων μέσα από την ασαφή συλλογιστική βάση κανόνων. Ως επιθυμητούς κανόνες θεωρούμε τους κανόνες αυτούς που συμβάλλουν στο τελικό αποτέλεσμα για ένα σύνολο δεδομένων εισόδου. Οι συνδυασμοί των κανόνων είναι αποθηκευμένες στη μνήμη ROM ασαφών συνόλων μοναδιαίας τιμής (singletons ROM - *s_rom_p*) και από κάθε περιεχόμενο της μνήμης (data) καταλαμβάνουν τα 4 περισσότερο σημαντικά bit (msb) συνδεδεμένα (concatenated) με τα υπόλοιπα bit, που καταλαμβάνουν οι τιμές των ασαφών συνόλων μοναδιαίας τιμής (singletons).

Η επιλογή των κανόνων λειτουργεί ως εξής: Οι προϋποθέσεις (antecedents) του κάθε κανόνα αποθηκεύεται στη μνήμη ROM είτε ως 0 είτε ως 1 ούτως ώστε να επιτρέπει την ενεργοποίηση ή την απενεργοποίηση ενός μέρους του κανόνα αντίστοιχα. Το μπλοκ της επιλογής κανόνων (rule selector – *rule_sel_p*), δέχεται σαν εισόδους τους βαθμούς αληθείας (alpha values – *alpha_val*), από τις γεννήτριες τραπεζοειδών συναρτήσεων συμμετοχής. Επιπλέον, λαμβάνει ένα σήμα επιλογής (selection signal – *active_sel*), από τη μνήμη ROM ασαφών συνόλων μοναδιαίας τιμής.

Ο παραμετρικός VHDL κώδικας (*entity* και *architecture* του κώδικα) του μπλοκ της επιλογής κανόνων, όπως και το αποτέλεσμα της σύνθεσης για 4 εισόδους διακρίνεται στο Σχήμα 6.11.

```
entity rule_sel_p is
  generic (
    ip_no : integer := 4;  -- number of inputs
    dy    : integer := 8);   -- input bus width
  port (
    sel     : in  std_logic_vector(ip_no-1 downto 0);      -- select
    ip_data : in  std_logic_vector(ip_no*dy-1 downto 0);  -- crisp data
    op_data : out std_logic_vector(ip_no*dy-1 downto 0));  -- crisp data
end entity rule_sel_p;

architecture rtl of rule_sel_p is

begin  -- ARCHITECTURE rtl

  A : for i in 0 to ip_no-1 generate

    signal x     : std_logic_vector(dy-1 downto 0);
    signal x_dec : std_logic_vector(dy-1 downto 0);
    signal test : std_logic_vector(sel'left downto 0);
    signal low  : std_logic_vector(sel'left downto 0) := (others =>'0');
    signal high : std_logic_vector(sel'left downto 0) := (others =>'1');

  begin  -- GENERATE
    -- select input data
    x <= ip_data((i+1)*dy -1 downto i*dy);
    -- select active alpha's
    x_dec <= (others=>'0') when sel = low else x when sel(i) = '1'
                                          else (others=>'1');
    -- collect output data
    op_data((i+1)*dy -1 downto i*dy) <= x_dec;

  end generate;

end architecture rtl;
```



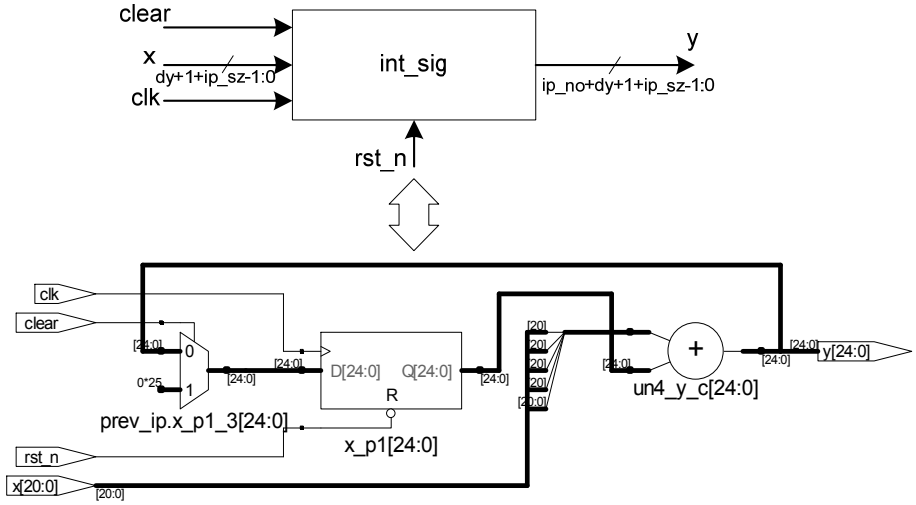
Σχήμα 6.11: Τμήμα επιλογής κανόνων και σχηματικό αποτέλεσμα της σύνθεσης

Πιο συγκεκριμένα, όταν κάποια από τις προϋποθέσεις ενός κανόνα δεν είναι ενεργή, τότε αυτή δεν συνεισφέρει στο συνολικό βάρος του κανόνα. Δεδομένου ότι, η συνεισφορά κάθε κανόνα (τιμή *theta*) υπολογίζεται κάνοντας χρήση του τελεστή ελαχίστου (*min* operator),

μπορούμε να αγνοήσουμε το βαθμό αληθείας (τιμή *alpha*) τις επιλεχθείσας προϋπόθεσης απλά θέτοντας τη στη μέγιστη τιμή, που στην περίπτωση που εξετάζουμε για *dy* = 8-bit ανάλυση βαθμού αληθείας (μη προσημασμένο αριθμοί αφού η alpha τιμή, $a \in \mathbb{R}^+$) είναι ίση με 255 (2^8-1). Τέλος αν όλες οι προϋποθέσεις του κανόνα είναι ανενεργές, τότε θα πρέπει έστω μία *alpha* τιμή να τεθεί σε κάποια ελάχιστη τιμή, ούτως ώστε η συνολική συνεισφορά (βάρος) του κανόνα να είναι μηδέν.

6.4.8 Ολοκληρωτής

Το μπλοκ αυτό (*int_sig*) υλοποιεί έναν προσημασμένο διακριτό ολοκληρωτή (βλ. Σχήμα 6.12). Η ίδια αρχιτεκτονική με ελάχιστες διαφορές χρησιμοποιείται και για την υλοποίηση του μη-προσημασμενου διακριτού ολοκληρωτή (*int_uns*). Για την αποφυγή υπερχείλισης έχει προβλεφτεί ένα σήμα εισόδου clear, που ελέγχεται από το *addr_gen_p* μπλοκ, το οποίο έχει περιγραφεί στην παράγραφο §6.4.3.



Σχήμα 6.12: Μπλοκ διακριτού ολοκληρωτή και σχηματικό αποτέλεσμα της σύνθεσης

6.4.9 Πολλαπλασιαστής

Η οικογένεια Spartan 3 των FPGA τσιπ της εταιρίας Xilinx [44] (που χρησιμοποιήθηκε για την υλοποίηση της αρχιτεκτονικής του DFLC στην παρούσα εργασία), παρέχει εμφυτευμένους (*embedded*) πολλαπλασιαστές, ο αριθμός των οποίων εξαρτάτε από τη σειρά του τσιπ της συγκεκριμένης οικογένειας. Στην παρούσα εργασία, χρησιμοποιήθηκε ένα Spartan-3 1500 FPGA το οποίο περιέχει 32 πολλαπλασιαστές αποκλειστικής χρήσης (dedicated multipliers). Οι δίαυλοι εισόδων των πολλαπλασιαστών δέχονται δεδομένα σε μορφή συμπληρώματος του δύο (2*'s complement*) (είτε 18-bit προσημασμένους αριθμούς ή 17-bit μη- προσημασμένους αριθμούς).

Κάθε πολλαπλασιαστής γειτονεύει με κάθε μπλοκ μνήμης RAM στη μήτρα πυριτίου (*die*) του FPGA. Η κοντινή φυσική τοπική γειτνίαση των δύο εξασφαλίζει αποδοτικό χειρισμό δεδομένων. Ο πολλαπλασιαστής μπορεί να χρησιμοποιηθεί σε ένα design είτε ως ασύγχρονος (ονομάζεται MULT18X18), είτε ως σύγχρονος (MULT18X18S) με ένα

καταχωρητή στην έξοδο του. Η αρχιτεκτονική του DFLC που μελετάμε χρησιμοποιεί το δεύτερο τύπο πολλαπλασιαστή.

6.4.10 Τελεστής Τομής και Ένωσης (AND/OR)

Το μπλοκ αυτό υλοποιεί διαφορετικές μεθόδους τομής και ένωσης (AND/OR) για τις προϋποθέσεις των κανόνων. Είναι πλήρως παραμετρικό όσον αναφορά στον αριθμό εισαγωγής συμπερασμάτων και την ανάλυση τους και στην επιλογή μεθόδου (*sel_op*). Οι τελεστές που υλοποιούνται είναι οι παρακάτω:
sel_op = 00 για min*, 01 για prod†, 10 για max‡, 11 για probor§ (probabilistic OR ή αλγεβρικό άθροισμα).

6.4.11 Διαιρέτης

Η διαίρεση από πλευράς υλικού, παραμένει μια από τις πιο πολύπλοκες, και χρονοβόρες πράξεις. Διάφορες μέθοδοι έχουν προταθεί στη βιβλιογραφία [65]. Οι μέθοδοι που στοχεύουν σε καλύτερα αποτελέσματα από πλευράς απαιτούμενου χρόνου για την ολοκλήρωση της διαδικασίας της διαίρεσης, συνήθως ξεκινάνε με μία χονδρική εκτίμηση του *Αντιστρόφου = 1/Διαιρέτης*. Στη συνέχεια, ακολουθούν μια επαναληπτική αριθμητική μέθοδο, όπου ο αριθμός των επαναλήψεων εξαρτάται από τη διαφορά ακρίβειας του αντιστρόφου με την επιθυμητή ακρίβεια του αποτελέσματος της διαίρεσης.

Μια παρόμοια μέθοδος ακολουθείται και εδώ, με τη διαφορά ότι, η ακρίβεια του αντιστρόφου υπολογίζεται κατά τέτοιο τρόπο που είναι ο ελάχιστος δυνατός για την επίτευξη του επιθυμητού αποτελέσματος στην έξοδο του διαιρέτη χωρίς την ανάγκη επαναλήψεων. Με αυτόν τον τρόπο, η διαδικασία της διαίρεσης απλοποιείτε ουσιαστικά σε έναν πολλαπλασιασμό μεταξύ της εκτίμησης του αντιστρόφου και του διαιρετέου. Εδώ θα πρέπει να διευκρινίσουμε ότι προσδιορίζουμε την ελάχιστη ακρίβεια της εκτίμησης του αντιστρόφου, με βάση όλες τις πιθανές τιμές που προκύπτουν από τη μέθοδο από-ασαφοποίησης του σταθμισμένου μέσου όρου, που δίνεται από την παρακάτω σχέση:

$$op = \frac{\sum_{i=1}^{m} w^i y_j^i}{\sum_{i=1}^{m} w^i} \tag{6.5}$$

Η ακρίβεια του αντιστρόφου υπολογίστηκε στα 22-bit με τη βοήθεια του μοντέλου *Simulink* που φαίνεται στο Σχήμα 6.13.

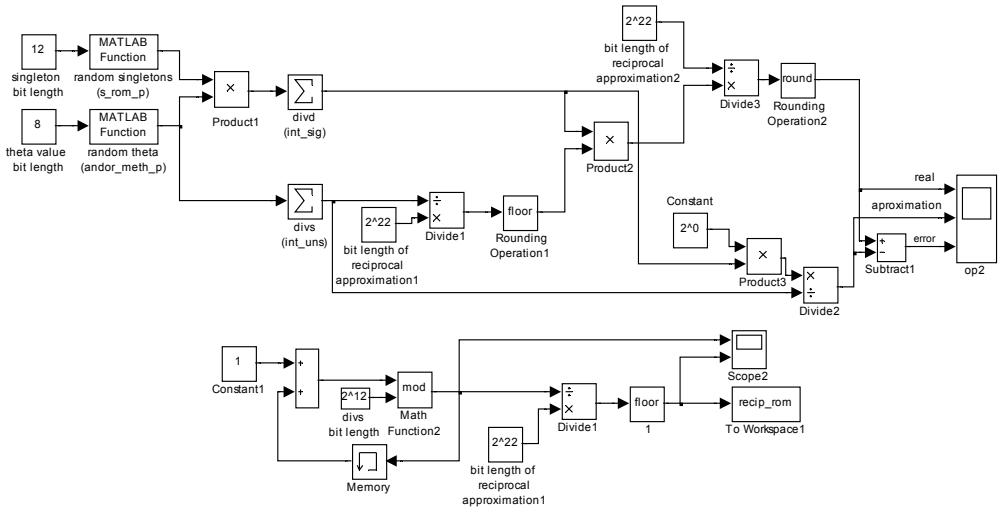
---

* t-norm: $\top\min(a,b)=\min\{a,b\}$

† t-norm: $\top\text{prod}(a,b)=a\cdot b$

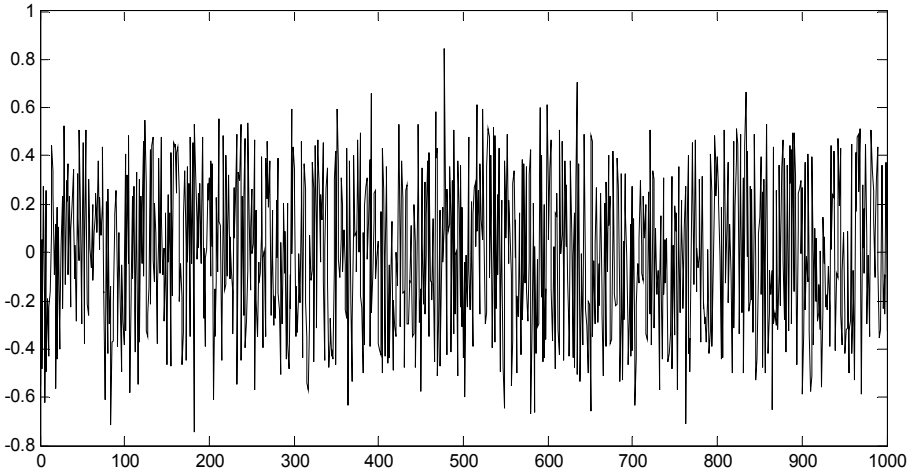
‡ s-norm: $\bot\max(a,b)=\max\{a,b\}$

§ s-norm: $\bot\text{sum}(a,b)=a+b-a\cdot b$



Σχήμα 6.13: Μοντέλο εκτίμησης της ακρίβειας του αντιστρόφου και δημιουργία της μνήμης ROM του αντιστρόφου

Το σφάλμα εξόδου που προκύπτει από το παραπάνω μοντέλο δίνεται στο παρακάτω σχήμα (βλ. Σχήμα 6.14).



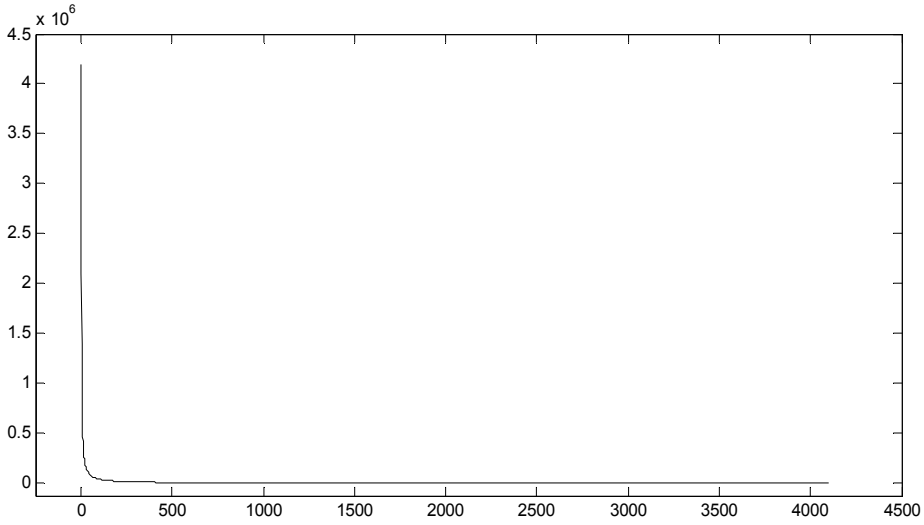
Σχήμα 6.14: Σφάλμα εξόδου

Στο Σχήμα 6.14 φαίνεται καθαρά ότι το σφάλμα που προκύπτει δεν είναι μεγαλύτερο από 0.8 και το αποτέλεσμα της διαίρεσης είναι έγκυρο για την επιθυμητή έξοδο ανάλυσης 12-bit. Πρακτικά το σφάλμα αυτό στρογγυλοποιείται, αφού η έξοδος περιορίζεται κλιμακωτά στα 12-bit. Η περίπτωση της διαίρεσης με το μηδέν μπορεί να συμβεί όταν όλες οι *theta* τιμές (συνεισφορά κανόνων) είναι μηδέν ή όταν όλες οι προϋποθέσεις των κανόνων έχουν απενεργοποιηθεί (από τον επιλογέα κανόνων – *rule_sel_p*) και οδηγούν σε συμπεράσματα που δεν συνεισφέρουν στο τελικό αποτέλεσμα (από το *s_rom_p*). Η τελευταία περίπτωση σημαίνει ότι για το δεδομένο σύνολο εισόδων, δεν ορίζεται κανένας κανόνας. Συνεπώς σε αυτές τις περιπτώσεις θέτουμε την έξοδο στην τιμή -1 (1/0 ≡ -1) με τη χρήση ενός επιπλέον κυκλώματος.

Λογαριάζοντας την περίπτωση που ο διαιρέτης είναι ίσος με τη μονάδα (*divs* = 1) και επομένως το αποτέλεσμα της διαίρεσης θα είναι ίσο με το διαιρετέο (*divd*/1 = *divd*), απλά

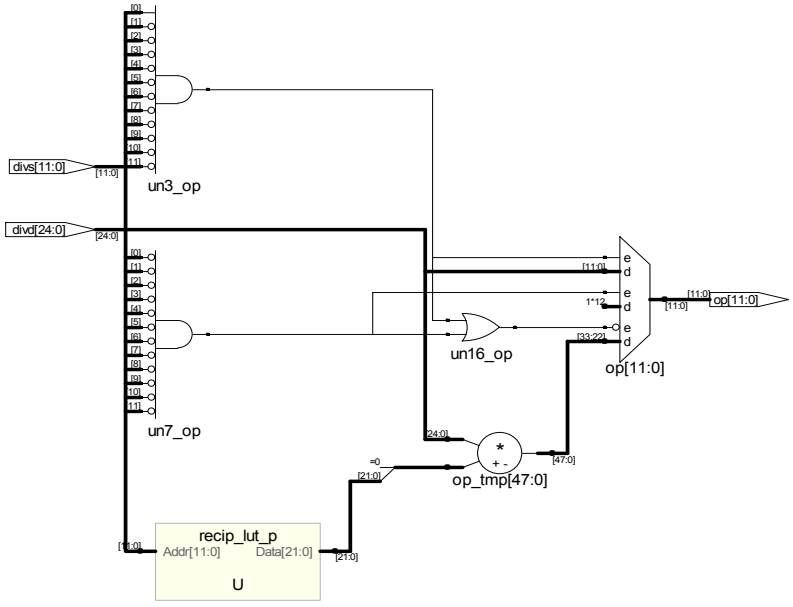
οδηγεί σε αύξηση του μεγέθους της μνήμης του αντιστρόφου (*reciprocal* LUT μπλοκ) από τα 22 στα 23-bit. Μιας και αυτό αυξάνει το μέγεθος της απαιτούμενης μνήμης και επιπλέον οδηγεί σε μεγαλύτερο πολλαπλασιασμό, επιλέχθηκε να χρησιμοποιηθεί ένα κατάλληλο κύκλωμα που ανιχνεύει την περίπτωση που περιγράφηκε και θέτει το αποτέλεσμα της διαίρεσης ίσο με το διαιρετέο.

Τα περιεχόμενα της μνήμης του αντιστρόφου αποτυπόνονται στη γραφική παράσταση που ακολουθεί (βλ. Σχήμα 6.15).



Σχήμα 6.15: Περιεχόμενα μνήμης αντιστρόφου

Το αποτέλεσμα της σύνθεσης του μπλοκ της διαίρεσης (*div_ppa*) δίνεται στο Σχήμα 6.16, όπου το στοιχείο του κυκλώματος που αναφέρεται ως *recip_lut_p* είναι η μνήμη ROM που κρατάει τα περιεχόμενα του πίνακα του αντιστρόφου (reciprocal LUT).



Σχήμα 6.16: Αποτέλεσμα της RTL σύνθεσης του διαιρέτη

6.5 Μεθοδολογία Σχεδίασης και Υλοποίησης του DFLC

Στην παράγραφο αυτή παρουσιάζετε η ροή της σχεδίασης (βλ. Σχήμα 6.17) κατά τρόπο «από επάνω προς τα κάτω» (*top–down*), που ακολουθείται στο DFLC. Στη σχεδίαση τέτοιου είδους, πρώτα εξετάζονται οι προδιαγραφές της τελικής υλοποίησης και έπειτα η λειτουργικότητα των επιμέρους τμημάτων.
Η αφετηρία της διαδικασίας σχεδίασης είναι η μοντελοποίηση του προτεινόμενου DFLC (μοντέλο Simulink) σε επίπεδο συστήματος (*system–level modeling*). Το μοντέλο Simulink μας επέτρεψε να αξιολογήσουμε το προτεινόμενο DFLC, και να εξάγουμε τις διανυσματικές τιμές δοκιμής (*test vectors*) που χρησιμοποιήσαμε αργότερα για την RTL και τη χρονική (timing ή Vital) προσομοίωση.

Η γλώσσα VHDL χρησιμοποιήθηκε για την περιγραφή του κυκλώματος σε RTL (*Register Transfer Level*). Ειδική προσοχή δόθηκε στη μεθοδολογία γραφής του κώδικα των διαφορετικών component του design, δεδομένου ότι στοχεύσαμε στο γράψιμο ενός πλήρως παραμετρικού κώδικα.

Το DFLC μπορεί να είναι παραμετρικό από την άποψη του αριθμού διαθέσιμων εισόδων και της ανάλυσής τους (σε bit), της ανάλυσης των τιμών alpha , του αριθμού εξόδων και της ανάλυσης τους, τη μέθοδο σύνδεσης των προϋποθέσεων, τον τύπο διαιρέτη, καθώς επίσης και του αριθμού βαθμίδων αγωγού (*pipeline stages*) του κάθε μπλοκ.

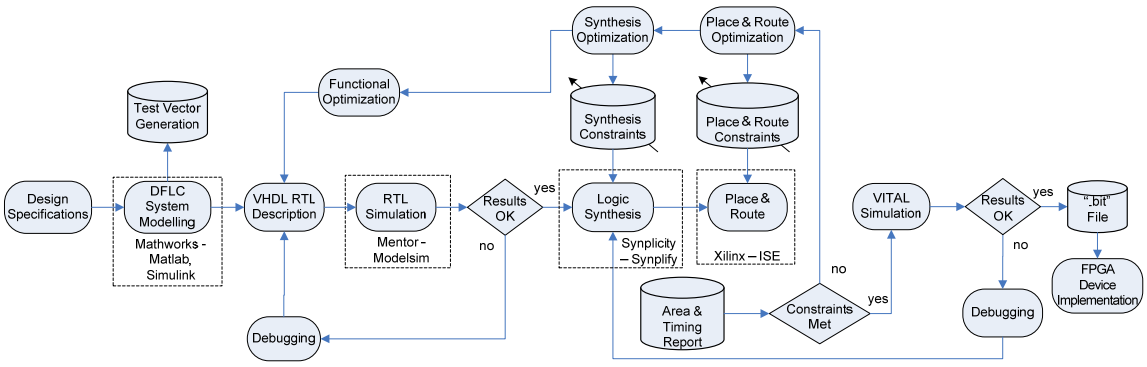
Το DFLC που παρουσιάζεται εδώ χρησιμοποιεί τις γενικές παραμέτρους που επιλέγονται στον Πίνακα 6.2. Χρησιμοποιείται ένα VHDL package που αποθηκεύει τις ανωτέρω γενικές παραμέτρους μαζί με τον αριθμό απαραίτητων βαθμίδων αγωγού για κάθε μπλοκ. Μια προσομοίωση RTL με τις διανυσματικές τιμές από το Simulink μοντέλο, εκτελέστηκε για να εξασφαλίσει τη σωστή λειτουργία του κυκλώματος. Στη συνέχεια, ακολούθησε η σύνθεση λογικής [60], όπου το εργαλείο σύνθεσης δημιούργησε αρχικά μια γενική (ανεξάρτητος τεχνολογίας) σχηματική αναπαράσταση βάση του κώδικα VHDL και έπειτα βελτιστοποίησε το κύκλωμα στη συγκεκριμένη βιβλιοθήκη FPGA που επιλέχθηκε (Spartan-3 1500-4FG676).

Σε αυτό το σημείο, οι περιορισμοί (*constraints*) για area και timing καθώς και οι συγκεκριμένες απαιτήσεις για το συγκεκριμένο design ορίστηκαν μιας και παίζουν σημαντικό ρόλο στο αποτέλεσμα της σύνθεσης.

Το αρχείο netlist (.edf), που είχε δημιουργηθεί προηγουμένως από το εργαλείο λογικής σύνθεσης εισάχθηκε στο εργαλείο θέσης και δρομολόγηση (*place and route*) του FPGA και ακολούθησαν τα επόμενα βήματα.

Κατ' αρχάς, το εργαλείο μετάφρασης (*translate*), μετέφρασε το netlist εισαγωγής μαζί με τους περιορισμούς (*constraints*) του design, σε μια βάση δεδομένων της Xilinx. Μετά από την επιτυχή μετάφραση του netlist, το πρόγραμμα αντιστοίχησης (*map*), δρομολόγησε τη λογική σχεδίαση (*logical design*) στο FPGA. Τέλος, το εργαλείο θέσης και δρομολόγησης (par) δέχθηκε το δρομολογημένο σχέδιο, και επέλεξε τις θέσεις των ιεραρχικών μπλοκ βάσει της σχεδίασης (*place*) και τις διαδρομές των διασυνδέσεων τους (*route*) μέσα στο FPGA με βάση τους προαπαιτούμενους περιορισμούς, και τέλος παρήγαγε ένα bitstream αρχείο (με το βοηθητικό εργαλείο BitGen). Το τελευταίο χρησιμοποιήθηκε για τον προγραμματισμό και τη διαμόρφωση του FPGA. Σε αυτό το σημείο αξίζει να σημειωθεί

ότι πριν τη διαδικασία προγραμματισμού και διαμόρφωσης του FPGA, εκτελέστηκε μια προσομοίωση χρονισμού (*timing* ή *vital* προσομοίωση) για να εξασφαλιστεί ότι, το κύκλωμα καλύπτει τις καθορισμένες απαιτήσεις χρονισμού και λειτουργεί σωστά (σε επίπεδο κυκλώματος πλέον και όχι μόνο RTL επίπεδο).



Σχήμα 6.17: Διάγραμμα ροής των βημάτων σχεδίασης και υλοποίησης

6.6 Αποτελέσματα

Ένα νέο σύνολο δεδομένων εισόδου εισάγεται στο σύστημα στη θετική παρυφή του ρολογιού *clkdv_out*, με το 1/16 της συχνότητας του ρολογιού *clkfx_out*. Συνεπώς παρέχει τον απαραίτητο χρόνο (16 αλγοριθμικοί κύκλοι ρολογιών) στη γεννήτρια διευθύνσεων (βαθμίδα αγωγού, *cpr1*) να ολοκληρώσει την παραγωγή των σημάτων που αντιστοιχούν σε όλους τους ενεργούς κανόνες ($2^{0\varsigma}$ – $17^{0\varsigma}$ κύκλοι ρολογιών).

Εδώ, υπενθυμίζουμε ότι με τη χρησιμοποίηση του μπλοκ *ars_p* προσδιορίζουμε αποτελεσματικά και επεξεργαζόμαστε μόνο τους 16 ενεργούς κανόνες ανά κύκλο ρολογιού, αντί του συνόλου των 2401 κανόνων [57].

Η γεννήτρια τραπεζοειδών συναρτήσεων συμμετοχής, (*trap_gen_p*) απαιτεί τρεις βαθμίδες αγωγού (*cpr3*) για να υπολογίσει τις τιμές των βαθμών αληθείας των συναρτήσεων συμμετοχής. Ο υπολογισμός προκύπτει ενώ το σύστημα εξετάζει και διαβάζει τη μνήμη ROM που περιέχει τα μονότιμα ασαφή σύνολα (*singletons* ROM). Δύο ρολόγια αργότερα (*cpr5*), αφού έχει πραγματοποιηθεί η επιλογή κανόνα (από το τμήμα επιλογής κανόνων – *rule_sel_p*) και ο τελεστής ελαχίστου (*andor_meth_p*) για το ζευγάρι των ενεργών κανόνων, οι *theta* τιμές τους (προκύπτουν κάνοντας *min* στις τιμές *alpha*) μαζί με τα συμπεράσματα τους και το σήμα για το μηδενισμό των ολοκληρωτών, διατίθενται στη συλλογιστική μηχανή (*Inference Engine*) και τον από-ασαφοποιητή (*Defuzziffizifier*).

Το αποτέλεσμα της συνεπαγωγής (implication) των κανόνων (*mult*), εμφανίζεται δύο ρολόγια αργότερα μαζί με την προσθήκη των τιμών *theta*. Το άθροισμα των συνεπαγωγών υπολογίζεται στο επόμενο ρολόι. Κατά τη διάρκεια αυτού του χρόνου οι μη-προσημασμένοι ολοκληρωτές (*div_uns*) εμφανίζουν την τιμή του διαιρέτη (*cpr7*). Αντίθετα, η τιμή του διαιρετέου υπολογίζεται από τον προσημασμένο ολοκληρωτή (*div_sig*), τρία ρολόγια αργότερα (*cpr9*) διατίθεται στην έξοδο και το αποτέλεσμα της διαίρεσης (*div_ppa*).

Ο συνολικός χρόνος επεξεργασίας των δεδομένων, ξεκινάει από τη στιγμή που ένα νέο σύνολο δεδομένων φτάνει στις εισόδους μέχρις ότου να παραχθεί στη έξοδο του συστήματος ένα έγκυρο αποτέλεσμα. Η προηγούμενη διαδικασία έχει συνολική καθυστέρηση (*latency*) 27 σταδίων αγωγών ή αντίστοιχα 270 ns, που αναλύεται σε 16 αλγοριθμικούς κύκλους ρολογιού (βλ. γεννήτρια διευθύνσεων στην παράγραφο §6.4.3)και 11 κύκλους ρολογιού εξαιτίας των σταδίων αγωγού (βλ. Πίνακα 6.2).

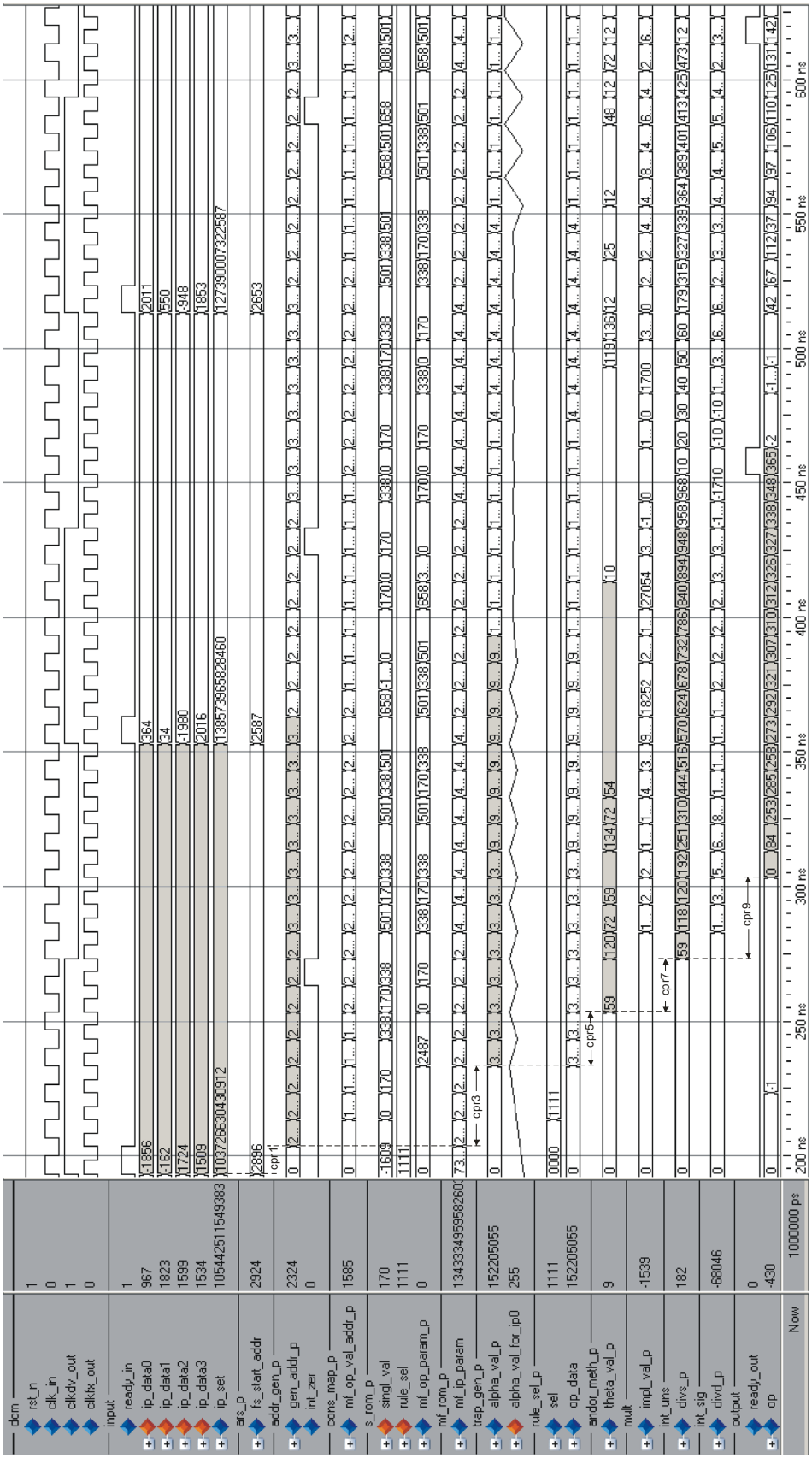
Αυτό χαρακτηρίζει αποτελεσματικά το ρυθμό επεξεργασίας δεδομένων του συστήματος (*throughput data processing rate*), δηλαδή ένα νέο σύνολο δεδομένων εισόδου μπορεί να δειγματοληπτιθεί κάθε 16 ρολόγια ή 160 ns), ενώ η εσωτερική λειτουργία συχνότητας του ρολογιού του πυρήνα DFLC είναι 100 MHz ή 10 ns.

Οι πόροι που χρησιμοποιούνται στο FPGA (XC3S1500) για το DFLC (με βάση τις επιλεγμένες παραμέτρους του Πίνακα 6.2) συνοψίζονται στον Πίνακα 6.3 που ακολουθεί.

Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops:	725	26,624	2%
Number of 4 input LUTs:	2,195	26,624	8%
Logic Distribution:			
Number of occupied Slices:	2,054	13,312	15%
Number of Slices containing only related logic:	2,054	2,054	100%
Number of Slices containing unrelated logic:	0	2,054	0%
Total Number 4 input LUTs:	3,635	26,624	13%
Number used as logic:	2,195		
Number used as a route-thru:	32		
Number used as 16x1 ROMs:	1,408		
Number of bonded IOBs:	64	487	13%
Number of Block RAMs:	3	32	9%
Number of MULT18X18s:	4	32	12%
Number of GCLKs:	3	8	37%
Number of DCMs:	1	4	25%

Πίνακας 6.3: Χρησιμοποιούμενοι πόροι στο FPGA

Στο Σχήμα 6.18 αποτυπώνεται η ροή δεδομένων για όλα τα σήματα του ελεγκτή που φαίνονται στο Σχήμα 6.2.



Σχήμα 6.18: Αποτελέσματα προσομοίωσης και δίοδος δεδομένων (dataflow) στον αγωγό (pipeline)

6.7 Συμπεράσματα

Το παρόν κεφάλαιο παρουσίασε ένα παραμετροποιήσιμο (διαμορφώσιμο) ψηφιακό ελεγκτή ασαφούς λογικής (DFLC) που επεξεργάζεται μόνο τους ενεργούς κανόνες για ένα σύνολο δεδομένων στις εισόδους. Ως συνέπεια των προηγούμενων, δίνεται η δυνατότητα διαμόρφωσης των παραμέτρων του πυρήνα χωρίς την ανάγκη της επανασχεδίασης του DFLC και επιπλέον επιτυγχάνεται σημαντική αύξηση του ρυθμού επεξεργασίας των δεδομένων εισόδου του συστήματος.

Ο πυρήνας του ελεγκτή είναι παραμετροποιήσιμος όσον αφορά τον αριθμό των εισόδων και εξόδων (και μέγεθος *διαύλου – bus*), του συστήματος, των αριθμό συναρτήσεων συμμετοχής (τριγωνικές ή τραπεζοειδείς) εισόδου και εξόδου (*singletons*), τη μέθοδο συνάθροισης και συνεπαγωγής, τον τύπο μοντέλου του διαιρέτη, και τον αριθμό καταχωρητών αγωγού των ανεξάρτητων μπλοκ στην ιεραρχία του design.

Στο κεφάλαιο που ακολουθεί παρουσιάζεται μία τεχνική αύξησης του ρυθμού επεξεργασίας δεδομένων του παρόντος DFLC με παρόμοιες παραμέτρους.

Κεφάλαιο 7

Βελτιστοποίηση Απόδοσης Ασαφούς Ελεγκτή

7.1 Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζεται μια βελτιωμένη μέθοδος για τη σχεδίαση ψηφιακών ελεγκτών ασαφούς λογικής (Digital Fuzzy Logic Controller – DFLC), που αναφέρεται στο κείμενο ως ODD-EVEN μέθοδος. Η συγκεκριμένη μέθοδος συμβάλει στη μείωση των κύκλων ρολογιών που απαιτούνται για την παραγωγή των διευθύνσεων των ενεργών κανόνων που υποβάλλονται σε επεξεργασία. Οι χαρακτηριστικές αρχιτεκτονικές ελεγκτών ασαφούς λογικής που αναφέρονται στη βιβλιογραφία [64][66][67] υποθέτουν επικάλυψη δύο μεταξύ των παρακείμενων ασαφών συνόλων, και απαιτούν 2^n κύκλους ρολογιών (ρυθμός επεξεργασίας δεδομένων εισόδου) όπου n είναι ο αριθμός εισόδων, δεδομένου ότι επεξεργάζονται έναν ενεργό κανόνα ανά κύκλο ρολογιών.

Χρησιμοποιώντας την προτεινόμενη ODD-EVEN μέθοδο, κατορθώνουμε να επεξεργαστούμε για το ίδιο πρότυπο μοντέλο σεναρίου, δύο ενεργούς κανόνες ανά κύκλο ρολογιών, αυξάνοντας κατά συνέπεια σημαντικά τη ρυθμό-απόδοση (throughput) του συστήματος. Η αρχιτεκτονική του σχεδίου που παρουσιάζουμε εδώ επιτυγχάνει ταχύτητα χρονισμού του πυρήνα στα 200 MHz (5 ns). Τα δεδομένα εισόδου μπορούν να δειγματοληπτηθούν κάθε δύο κύκλους του ρολογιού χρονισμού του πυρήνα (δηλαδή 2×5 ns = 10 ns) ή με το μισό της ταχύτητας συχνότητας του πυρήνα (100 MHz), με επεξεργασία μόνο των ενεργών κανόνων. Το συγκεκριμένο DFLC είναι βασισμένο σε έναν απλό αλγόριθμο παρόμοιο με τη μέθοδο εξαγωγής συμπεράσματος και από-ασαφοποίησης των Takagi-Sugeno [62][27] μηδενικού όρου και χρησιμοποιεί δύο εισόδους των 8-bit και

μια έξοδο των 12-bit, με μέχρι 7 τραπεζοειδείς ή τριγωνικές μορφής συναρτήσεις συμμετοχής ανά είσοδο και ασαφή βάση 49 κανόνων. Η συνολική καθυστέρηση (latency) της αρχιτεκτονικής του DFLC περιλαμβάνει 13 στάδια αγωγών*, όπου το καθένα απαιτεί χρόνο 5 ns (περίοδος εσωτερικού ρολογιού). Λόγω της τεχνικής συνεχούς διοχέτευσης (pipelining) δεδομένων, που επικρατεί στο κύκλωμα η ρυθμό-απόδοση (throughtput) του ελεγκτή είναι ίση με έναν ολοκληρωμένο υπολογισμό ανά δύο κύκλους ρολογιού [63][68].

7.2 Χαρακτηριστικά του DFLC

Οι παράμετροι του DFLC πυρήνα συνοψίζονται στον κάτωθι πίνακα (βλ. Πίνακα 7.1).

Τύπος Ασαφούς Συλλογιστικής Μηχανής (*Fuzzy Inference System - FIS*)	Takagi-Sugeno μηδενικού-βαθμού
Αριθμός Εισόδων	2
Μέγεθος Διαύλου Εισόδου (*Input Bus Width*)	8-bit
Αριθμός Εξόδων	1
Μέγεθος Διαύλου Εξόδου (*Output Bus Width*)	12-bit
Συναρτήσεις Συμμετοχής Υποθέσεων (*Antecedent Membership Functions*)	7 Τραπεζοειδούς σχήματος
Ανάλυση βαθμού αληθείας (α τιμή) (*MF degree of truth resolution*)	4-bit
Συναρτήσεις Συμμετοχής Συμπερασμάτων (*Consequent MF's*)	49 Μονότιμα σύνολα†
Ανάλυση Συμπερασμάτων (*Consequent MF resolution*)	8-bit
Μέγιστος Αριθμός Κανόνων (*Maximum Number of Fuzzy Inference Rules*)	49‡
Μέθοδος Εξαγωγής Βαθμού Ενεργοποίησης Κανόνα (*Rule Firing Strength*)	Min (Mamdani§ τελεστής ελαχίστου)
Μέθοδος Συνεπαγωγής Εξαγωγής Συμπεράσματος Κανόνα (*Implication Method*)	Prod (Larsen§ αλγεβρικό γινόμενο)
Μέγιστος Βαθμός Επικάλυψης Ασαφών Συνόλων (*Overlapping Degree*)	2
Μέθοδος Από-ασαφοποίησης (*Defuzzification Method*)	Μέθοδος Σταθμισμένου Μέσου**

* Κάθε στάδιο αγωγού ισοδυναμεί με έναν κύκλο ρολογιού
† Singletons
‡ Προκυπτει ως: $\textit{αριθμός ασαφών συνόλων}^{\textit{αριθμός εισόδων}}$
§ βλ. §2.2.4.3, Πίνακας 2.6
** Weighted Average Method

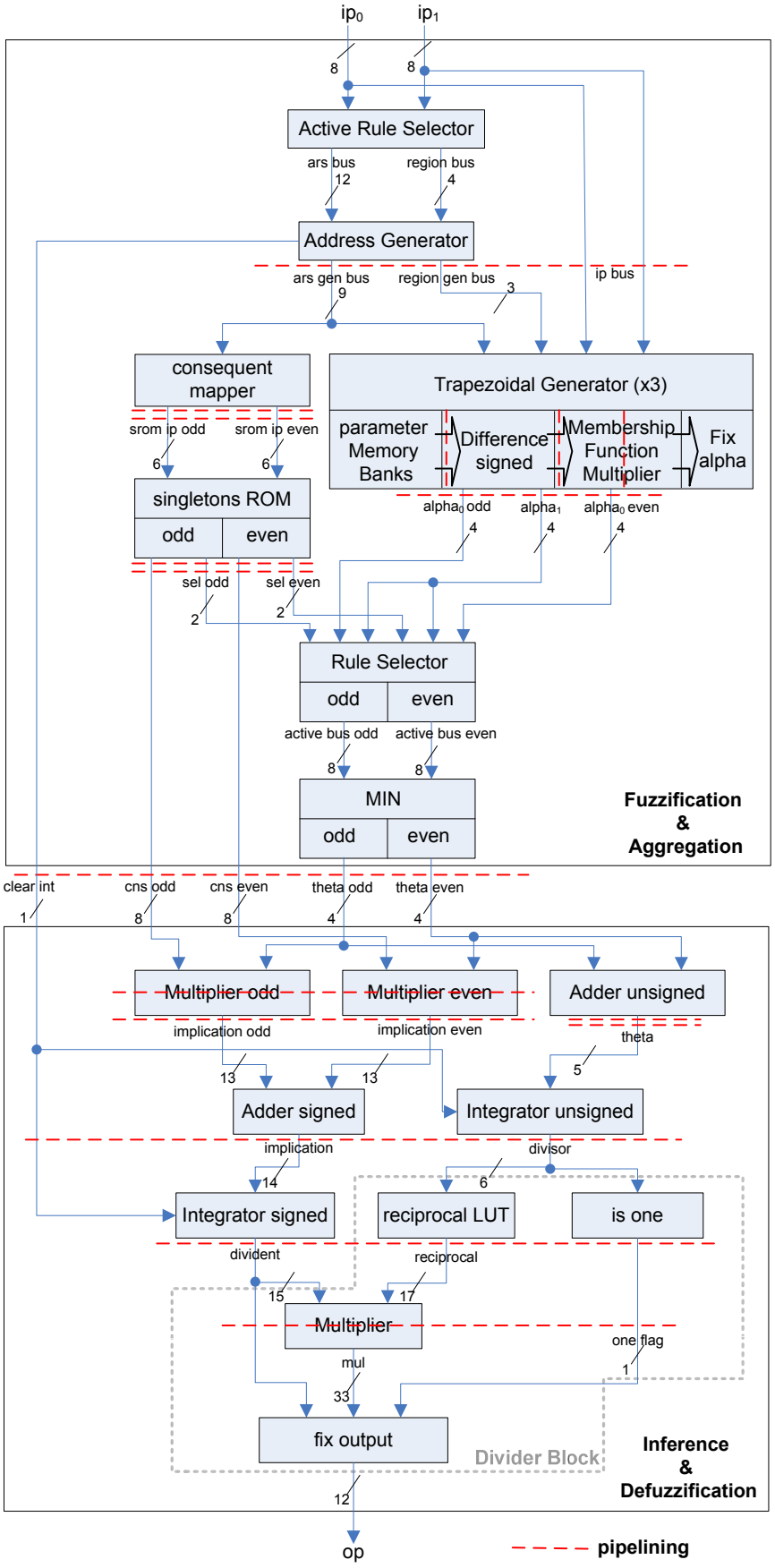
Πίνακας 7.1: Χαρακτηριστικά του DFLC

7.3 Υλοποίηση Υλικού του DFLC

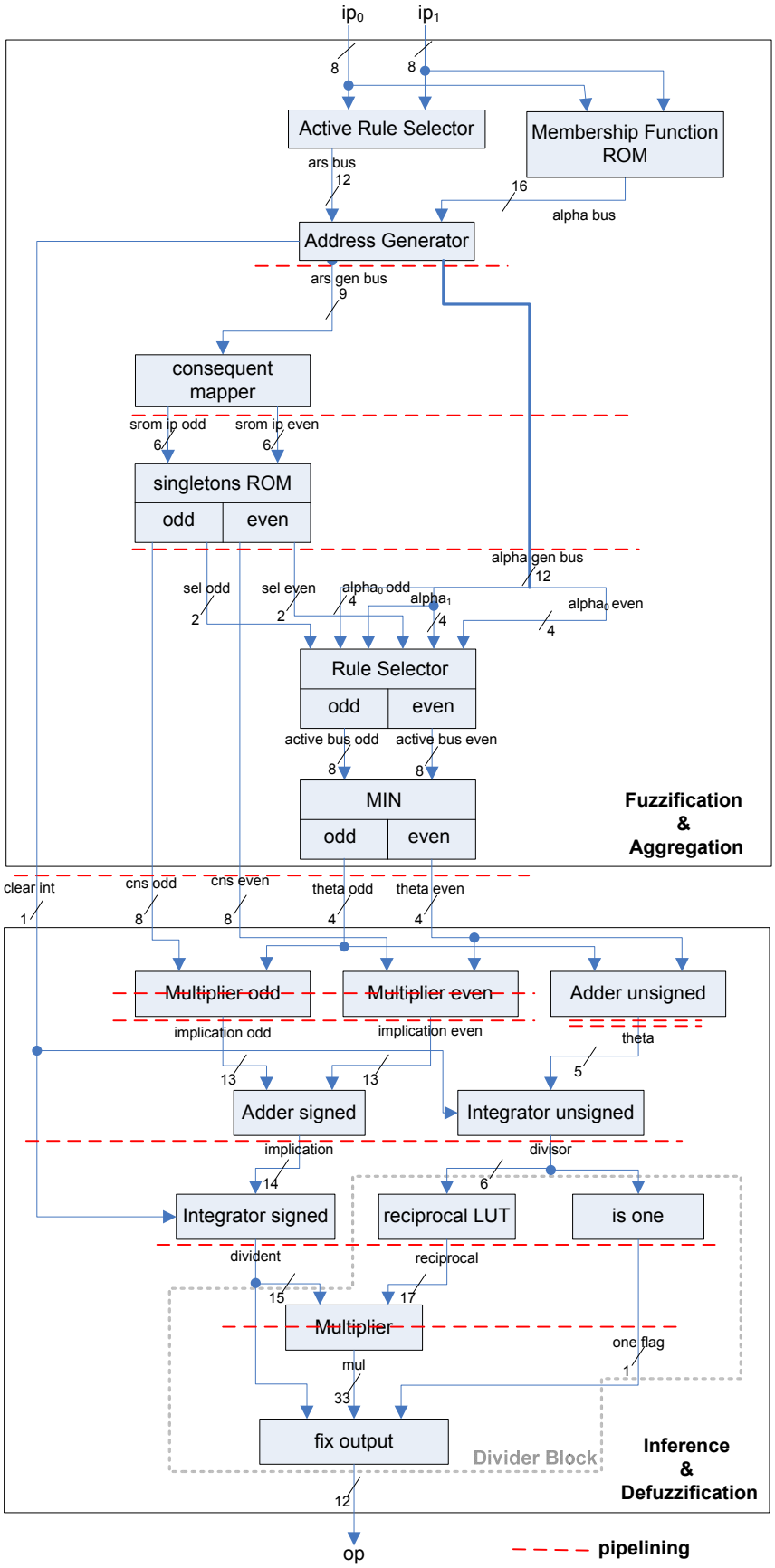
Στην ενότητα που ακολουθεί επιχειρείται μια αναλυτική περιγραφή των διάφορων ιεραρχικών μπλοκ (*block*) της αρχιτεκτονικής που συνθέτει το DFLC, και επιπλέον εξηγείται η ODD-EVEN μέθοδος που ακολουθείται για την αύξηση της ρυθμό-απόδοσης (*throughput*) του ελεγκτή.

7.3.1 Αρχιτεκτονική DFLC

Η αρχιτεκτονική του πυρήνα που μελετάται εδώ, παρουσιάζεται στα Σχήματα 7.1 και 7.2 στις επόμενες σελίδες που ακολουθούν. Το πρώτο σχήμα ενσωματώνει αριθμητική συνάρτηση συμμετοχής (βλ. Παράρτημα Α.2), ενώ το δεύτερο συνάρτηση συμμετοχής βασισμένη σε μνήμη ROM ή αλλιώς πίνακα αναζήτησης (Look Up Table – LUT). Τα δυο κύρια μπλοκ που συνθέτουν την αρχιτεκτονική DFLC είναι το «Ασαφοποίησης και Συνεπαγωγής» και «Συνάθροισης και Από-ασαφοποίησης». Οι διακεκομμένες γραμμές στο Σχήμα 7.1 υποδηλώνουν τον αριθμό των σταδίων αγωγού του κάθε μπλοκ. Επιπλέον στο σχήμα σημειώνονται και τα μεγέθη διαύλων των σημάτων.



Σχήμα 7.1: Αρχιτεκτονική DFLC με αριθμητική συνάρτηση συμμετοχής



Σχήμα 7.2: Αρχιτεκτονική DFLC με συνάρτηση συμμετοχής βασισμένη σε LUT

7.3.2 Επιλογέας Ενεργών Κανόνων

Προκειμένου να αποφευχθεί η επεξεργασία όλων των συνδυασμών των κανόνων για κάθε νέο σύνολο στοιχείων εισόδου, έχουμε επιλέξει να επεξεργαστούμε μόνο τους ενεργούς κανόνες. Εξετάζοντας το ανωτέρω πρόβλημα και λαμβάνοντας υπόψη το γεγονός ότι η επικάλυψη μεταξύ των παρακείμενων ασαφών συνόλων είναι 2, έχουμε εφαρμόσει ένα μπλοκ επιλογής ενεργών κανόνων (Active Rule Selector - ARS) για να υπολογίσουμε την περιοχή των ασαφών συνόλων στην οποία αντιστοιχεί το δεδομένο εισόδου. Κατά συνέπεια, για το DFLC 2 εισόδων με 7 συναρτήσεις συμμετοχής ανά είσοδο, αντί της επεξεργασίας και των 49 συνδυασμών των κανόνων που υπάρχουν στη βάση κανόνων, μόνο 4 ενεργοί κανόνες χρειάζεται να ληφθούν υπόψη (ανά κάθε νέο σύνολο στοιχείων εισόδου).

Στο Σχήμα 7.3, επεξηγείται η κωδικοποίηση των ασαφών συνόλων (Fuzzy Sets - FS) που αντιστοιχούν στις εισόδους του ελεγκτή, καθώς επίσης και ο διαχωρισμός των περιοχών FS που προκύπτει με την επαναλαμβανόμενη σύγκριση των δύο τιμών δεδομένων εισόδου ($ip_{0,1}$) με τα σημεία ανόδου ($rise\_start_{0,1}$) των συναρτήσεων συμμετοχής στα ασαφή σύνολα.

Αξίζει να επισημανθεί ότι η σύγκριση του $1^{ου}$ και $2^{ου}$ FS δεν είναι απαραίτητη, δεδομένου ότι το αρχικό ζευγάρι FS είναι πάντα το $1^{ο}$ και το $2^{ο}$ και η μετατόπιση δεξιά στο επόμενο ζευγάρι FS γίνεται μόνο όταν ξεπερνιέται το σημείο ανόδου (rise start) του $3^{ου}$ FS. Δεδομένου ότι απαιτούμε τη μετάβαση στο παρακείμενο ζευγάρι FS όταν $ip_{0,1} = rise\_start_{0,1}$ και όχι όταν $ip_{0,1} > rise\_start_{0,1}$, το πρόσημο (most significant bit - msb) του $ip0,1$ - $rise\_start0,1$ υπολογίζεται με αυτή τη σειρά.

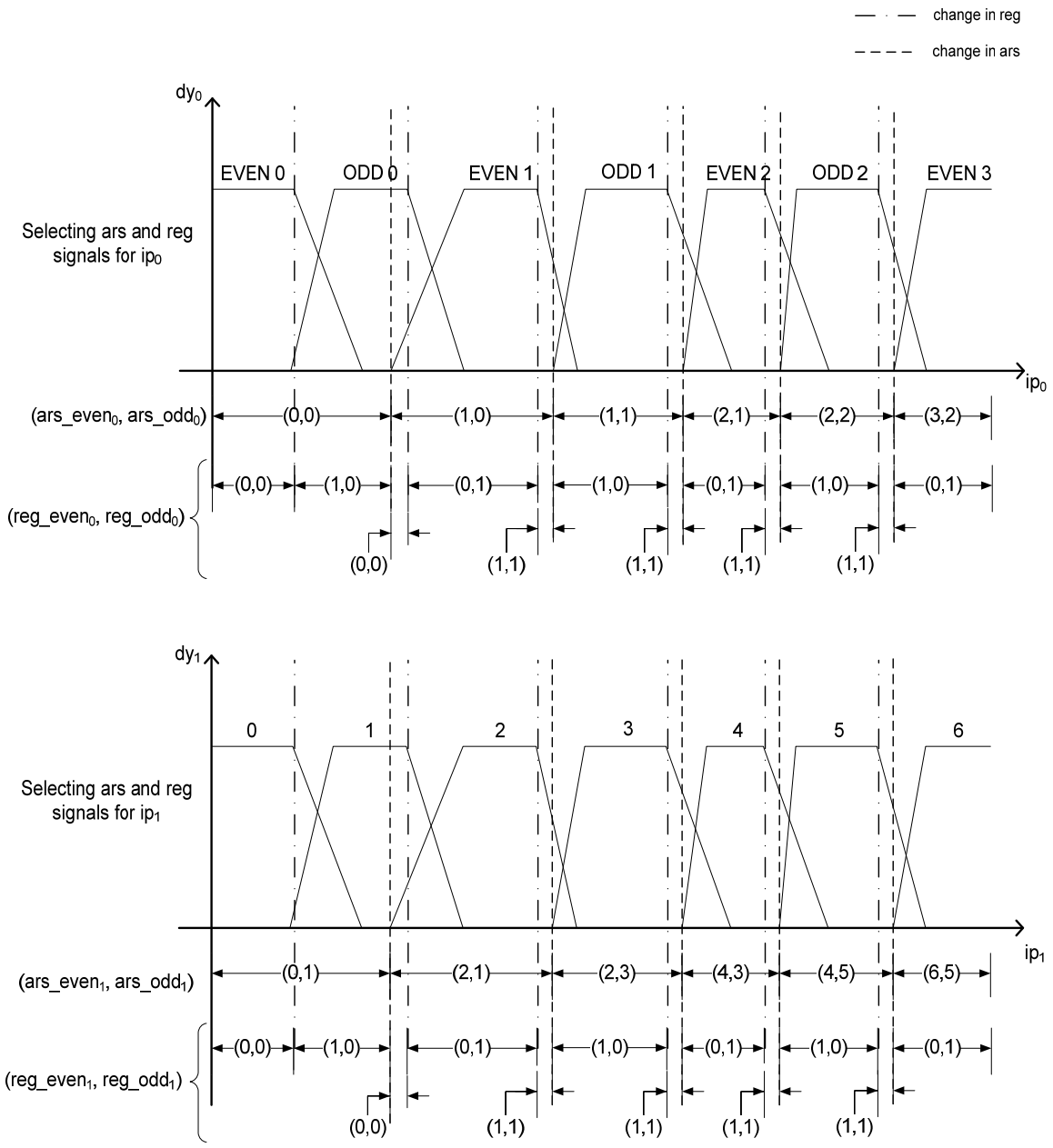
Η μετάβαση τμημάτων των FS απεικονίζεται στο Σχήμα 7.3 με γραμμές που φέρουν την ονομασία «*change in ARS*». Επιπλέον φαίνεται για το επιλεγμένο ζευγάρι (ODD-EVEN), κατά πόσο το $ip_{0,1}$ εκτείνεται στο σημείο ανόδου ($min_{\text{βαθμός αληθείας}}$ ή 0) ή στο σημείο καθόδου ($max_{\text{βαθμός αληθείας}}$ ή 1) του κάθε FS. Τα τμήματα στα οποία προκύπτει αλλαγή περιοχών εμφανίζονται ξεχωριστά με την ονομασία «*change in REG*». Πρέπει να σημειωθεί εδώ ότι ο υπολογισμός των περιοχών είναι ίδιος για όλες τις εισόδους και επιπλέον ότι το τελευταίο FS δεν λαμβάνεται υπόψη δεδομένου ότι είναι μια συνάρτηση συμμετοχής S-τύπου που δεν διαθέτει σημείο καθόδου ($fall\_start_{0,1}$). Γενικά ισχύει,

i. $ars_{0,1}\_odd, ars_{0,1}\_even \rightarrow$ ενεργό ζεύγος FS

για το ενεργό ζεύγος FS θα ισχύουν τα παρακάτω:

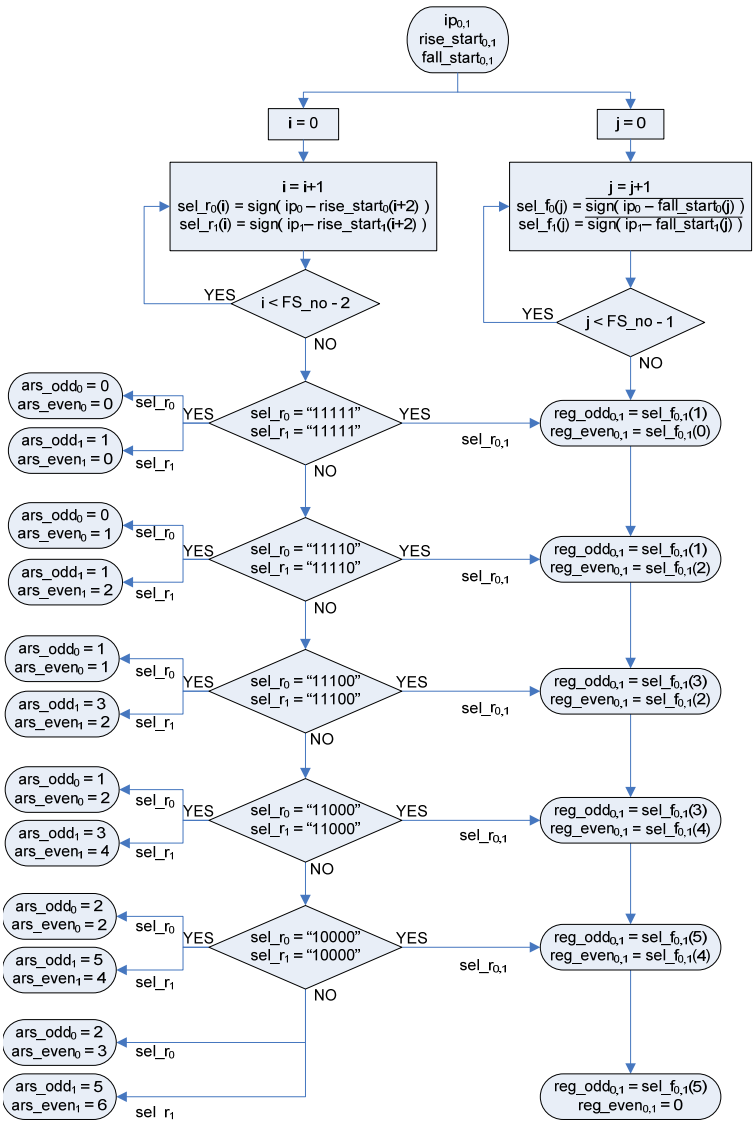
ii. $$reg\_odd_{0,1} \begin{cases} 0, if\ ip_{0,1} < fall\_start_{ars\_odd_{0,1}} \\ 1, if\ ip_{0,1} \geq fall\_start_{ars\_odd_{0,1}} \end{cases}$$

iii. $$reg\_even_{0,1} \begin{cases} 0, if\ ip_{0,1} < fall\_start_{ars\_even_{0,1}} \\ 1, if\ ip_{0,1} \geq fall\_start_{ars\_even_{0,1}} \end{cases}$$



Σχήμα 7.3: Κωδικοποίηση συναρτήσεων συμμετοχής και περιοχές ODD-EVEN

Το διάγραμμα ροής του αλγορίθμου που εφαρμόζεται στην υπομονάδα ARS δίνεται στο Σχήμα 7.4 στη σελίδα που ακολουθεί.



Σχήμα 7.4: Αλγόριθμος Επιλογής Ενεργών Κανόνων (ARS)

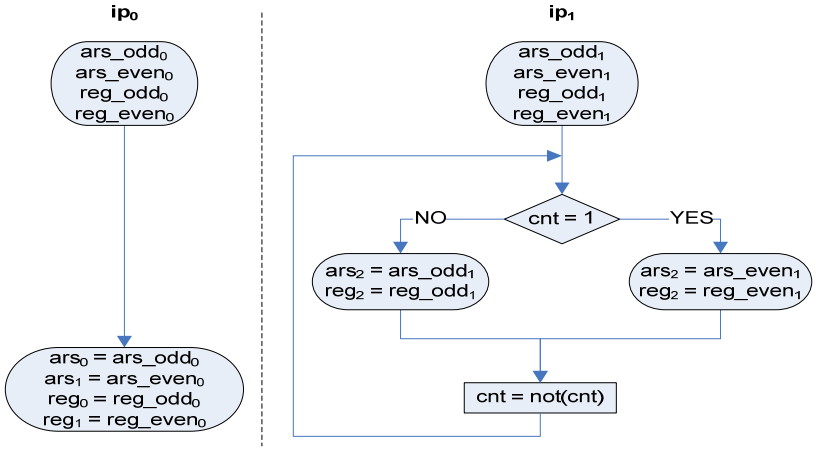
7.3.3 Γεννήτρια Διευθύνσεων

Το μπλοκ αυτό επιλέγει ανάμεσα στις εξόδους του ARS, παράγοντας σε κάθε κύκλο ρολογιού από ένα διαφορετικό συνδυασμό. Σκοπός μας είναι να εξετάσουμε όλους τους δυνατούς συνδυασμούς που προκύπτουν από την επιλογή ενός από τα δύο ενεργά ασαφή σύνολα κάθε εισόδου, για όλες τις εισόδους. Ο αριθμός των δυνατών συνδυασμών είναι 2^n ή 4 (για n = 2 αριθμό εισόδων) και αντιστοιχεί στον αριθμό των ενεργών κανόνων (συνεπώς και στον αριθμό κύκλων ρολογιού που απαιτούνται για την επεξεργασία όλων των ενεργών κανόνων). Μπορούμε ωστόσο να μειώσουμε τους κύκλους ρολογιού στο μισό, όπως θα φανεί παρακάτω, εξετάζοντας παράλληλα τους συνδυασμούς που προκύπτουν αν επιλέξουμε το ODD με αυτούς που προκύπτουν αν επιλέξουμε το EVEN ασαφές σύνολο στην πρώτη είσοδο του ασαφούς ελεγκτή. Υπό αυτό το σκεπτικό, περνάμε κατευθείαν στην έξοδο τις εισόδους που αφορούν την ip_0 ενώ αντίθετα κάνουμε επιλογή μεταξύ των ODD και EVEN εισόδων που αφορούν την ip_1. Σαν σήμα επιλογής

χρησιμοποιούμε ένα μετρητή *(cnt)* 1-bit, τον οποίο χρησιμοποιούμε και σαν σήμα μηδενισμού των ολοκληρωτών, το περιεχόμενο του οποίου αντιστρέφουμε σε κάθε θετική παρυφή του ρολογιού. Το διάγραμμα ροής φαίνεται στο Σχήμα 7.5.

Η γεννήτρια διευθύνσεων (Address Generator - ADG) παράγει τις διευθύνσεις που υποδεικνύονται από το ARS και απαιτούνται για τους ενεργούς ασαφείς κανόνες ($ars\_even_{0,1}$, $ars\_odd_{0,1}$, $reg\_even_{0,1}$, $reg\_odd_{0,1}$). Δεδομένου ότι το ARS έχει προηγουμένως ήδη προσδιορίσει όλα τα εμπλεκόμενα ασαφή σύνολα κατευθείαν (πλήρως συνδυαστικό κύκλωμα – combinational circuit*), το ADG παράγει ανά περίοδο ρολογιού τις διευθύνσεις που αντιστοιχούν στους ενεργούς ασαφείς κανόνες (βλ. Παράρτημα Α.2).

Η συμβατική αρχιτεκτονική της γεννήτριας διευθύνσεων (βλ. §6.4.3) απαιτεί 4 κύκλους ρολογιών για να παράγει τις 4 διευθύνσεις των ενεργών κανόνων, αυξάνοντας συνεπώς την περίοδο του ρυθμού εισαγωγής νέων δεδομένων στον ελεγκτή κατά 4 φορές επί το εσωτερικό ρολόι λειτουργίας του. Στην παρούσα διαμορφωμένη σχεδίαση με τη χρήση της ODD-EVEN μεθόδου, επιτυγχάνεται μείωση του αριθμού των κύκλων ρολογιού που απαιτείται από το ADG για την παραγωγή των διευθύνσεων στους μισούς.



Σχήμα 7.5: Αλγόριθμος γεννήτριας διευθύνσεων

Από τα παραπάνω γίνεται κατανοητό ότι για να προκύψουν όλοι οι δυνατοί συνδυασμοί των ενεργών ασαφών συνόλων των δύο εισόδων χρειάζονται *4/2=2* κύκλοι ρολογιού. Αυτό είναι εφικτό επειδή σε κάθε κύκλο εξετάζονται δύο κανόνες ταυτόχρονα αυτός που αναφέρεται στο περιττό (ODD) ενεργό ασαφές σύνολο της πρώτης εισόδου και αυτός που αναφέρεται στο άρτιο (EVEN). Συνεπώς, είναι δυνατή η εισαγωγή και επεξεργασία ενός καινούριου ζεύγους εισόδων κάθε 2 κύκλους ρολογιού. Φυσικά, η μέθοδος σχεδίασης αυτή, την οποία ονομάσαμε «ODD-EVEN» μέθοδο, μπορεί να βρει εφαρμογή και σε έναν ασαφή ελεγκτή περισσότερων εισόδων. Στη γενική περίπτωση, λοιπόν, η συχνότητα δειγματοληψίας της εισόδου ($f_{εξωτερική}$) είναι $(1/2^{n-1}) \cdot f_{DFLC}$, όπου n ο αριθμός των εισόδων του ασαφούς ελεγκτή ή ισοδύναμα η ρυθμό-απόδοση (*throughput*) του συστήματος θα δίνεται από,

* Ψηφιακό λογικό κύκλωμα στο οποίο η έξοδος προκύπτει αποκλειστικά συναρτήσει της παρούσας εισόδου.

$$throughput = \text{ samples} / 2^{n-1} \cdot T_{DFLC}(\text{sec}) \quad (7.1)$$

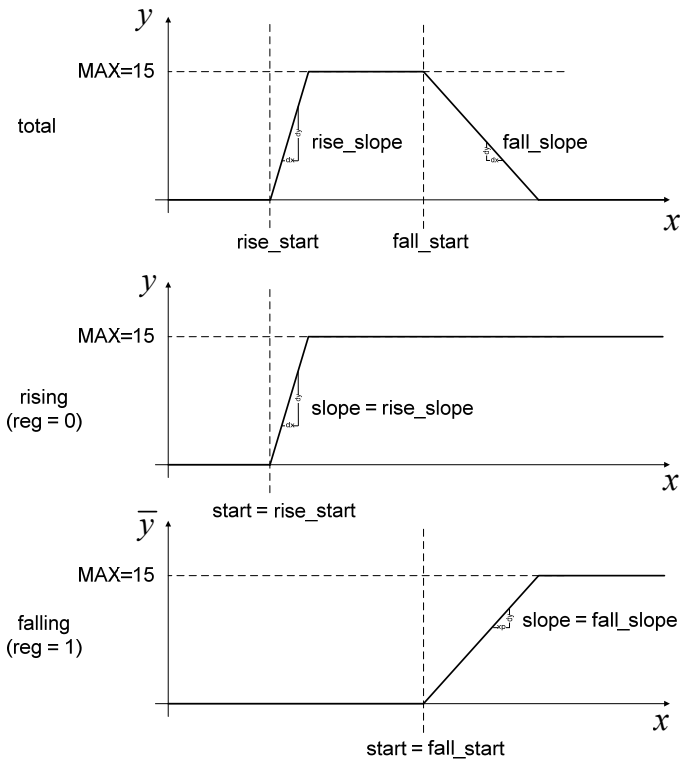
7.3.4 Γεννήτρια Τραπεζοειδών Συναρτήσεων Συμμετοχής

Το μπλοκ αυτό αποτελεί την υλοποίηση της Γεννήτριας Τραπεζοειδών Συναρτήσεων Συμμετοχής (Trapezoidal Membership Function Generator – TMFG). Η κωδικοποίηση που ακολουθήθηκε για το τραπέζιο φαίνεται στο Σχήμα 7.6.

Για κάθε ασαφές σύνολο A_{ij} αρκεί να γνωρίζουμε τα εξής:

- σημείο ανόδου (rise_start) – μέγεθος (8-bit)
- κλήση ανόδου (rise_slope) – μέγεθος (4-bit)
- σημείο καθόδου (fall_start) – μέγεθος (8-bit)
- κλήση καθόδου (fall_slope) – μέγεθος (4-bit)

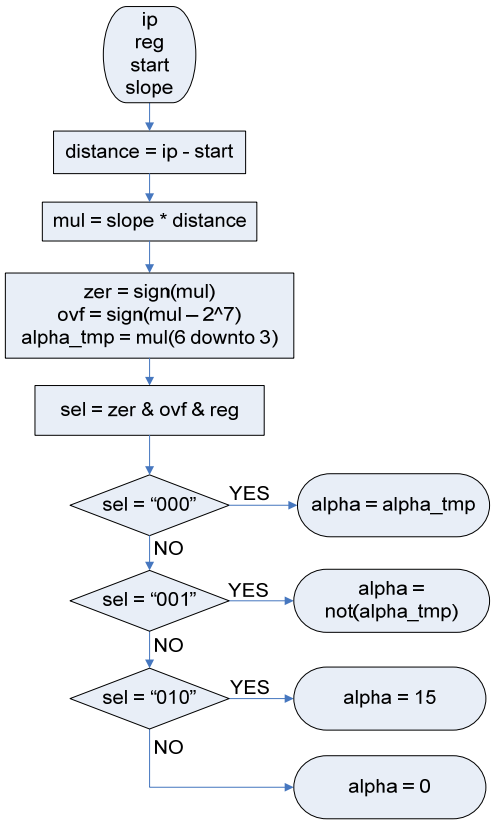
Τα δεδομένα φυλάσσονται στις Μνήμες αποθήκευσης των παραμέτρων (Membership Function parameters ROM) με τον τρόπο που φαίνεται στον Πίνακα 7.2, σύμφωνα με την ονοματολογία των ασαφών συνόλων που δίνονται στο Σχήμα 7.3.



Σχήμα 7.6: Κωδικοποίηση Τραπεζίου και χωρισμός αυτού σε περιοχές

Από το Σχήμα 7.6 παρατηρούμε ότι η επιλεχθείσα κωδικοποίηση μας επιτρέπει μια ενιαία αντιμετώπιση των τμημάτων ανόδου και καθόδου της συνάρτησης συμμετοχής τραπεζοειδούς μορφής με τον ίδιο τρόπο, με μόνη διαφορά τη λογική αντιστροφή *(not)* του βαθμού αληθείας *(y ή α τιμή)* στην περίπτωση του τμήματος πτώσης. Η επίδραση του

τελευταίου είναι ότι το τμήμα καθόδου προκύπτει ως συμμετρικό του τμήματος ανόδου αναφορικά ως προς τον άξονα του υπερσυνόλου αναφοράς (*X*). Το γεγονός αυτό είναι πολύ ικανοποιητικό αφού μειώνει αρκετά την πολυπλοκότητα του κυκλώματος σε σχέση με τη σχεδίαση της ίδιας δομικής μονάδας που αναλύθηκε στην παράγραφο §6.4.4.
Με δεδομένη την κωδικοποίηση μπορούμε πλέον να εξετάσουμε τον αλγόριθμο υπολογισμού του οποίου το διάγραμμα ροής εικονίζεται στο Σχήμα 7.7.



Σχήμα 7.7: Αλγόριθμος Γεννήτριας Τραπεζοειδών Συναρτήσεων Συμμετοχής (TMFG)

Τα σήματα εισόδου προκύπτουν ως εξής:

- **ip:** η αντίστοιχη είσοδος (ip_0, ip_0, ip_1)
- **reg:** τροφοδοτείτε από την έξοδο του ADG (reg_0, reg_1, reg_2)
- **start:** τροφοδοτείτε από την έξοδο της αντίστοιχης μνήμης παραμέτρων που λαμβάνει ως είσοδο τα αντίστοιχα σήματα *ars&reg* (ars_0&reg_0, ars_1&reg_1, ars_2&reg_2)
- **slope:** τροφοδοτείτε από την έξοδο της αντίστοιχης μνήμης παραμέτρων με όμοιο τρόπο

Πρώτα υπολογίζουμε την απόσταση (*distance*) και έπειτα πολλαπλασιάζουμε με την κλήση (*mul*). Έπειτα κάνουμε περικοπή (*truncation*) κατά 3-bit (*alpha_tmp*). Το πόσα bit θα περικόψουμε έχει να κάνει με το τι ακρίβεια θέλουμε να έχει η κλήση (*slope*). Εδώ επιλέχθηκαν τα 3-bit που επιτρέπουν ομοιόμορφή ακρίβεια τόσο στις μεγάλες όσο και στις μικρές κλίσεις. Τέλος, κάνουμε διόρθωση αποτελέσματος (τιμή *alpha, a*) με τη βοήθεια $3^{\omega v}$ σημαιών (*flags*):

- **reg:** υποδηλώνει σε ποια περιοχή (*ανόδου*, *καθόδου*) του τραπεζίου βρισκόμαστε
- **ovf:** ενεργοποιείται σε περίπτωση που έχουμε υπερχείλιση *(overfloating)*
- **zer:** ενεργοποιείται σε περίπτωση που η απόσταση *(distance)* είναι αρνητική

Η έξοδος που ονομάζεται άλφα τιμή (*alpha value*) αντιπροσωπεύει το βαθμό αλήθειας του τρέχοντος ασαφούς συνόλου. Η σημαία *zer* απεικονίζει την ειδική περίπτωση όπου $ip_{0,1} < rise\_start_{0,1}$, και η σημαία *ovf* εάν η άλφα τιμή έχει υποστεί υπερχείλιση (*overfloat*).

Τρία πανομοιότυπα μπλοκ του TMFG απαιτούνται στη συγκεκριμένη αρχιτεκτονική. Η $1^{η}$, $2^{η}$ και $3^{η}$ αντίστοιχα περίπτωση του TMFG δέχονται ως εισόδους τις εισόδους του ελεγκτή (ip_0, ip_0, ip_1), την περιοχή (reg_0, reg_1, reg_2), το σημείο ανόδου (*rise_start*) και την κλίση (*slope*) από την αντίστοιχη μνήμη ROM που φυλάσσει τις παραμέτρους (Parameter Memory Bank ROM) ως συνδεδεμένα σήματα (*concatenated signals*) ars_0&reg_0, ars_1&reg_1, ars_2&reg_2.

7.3.5 Μνήμες Αποθήκευσης Παραμέτρων Συναρτήσεων Συμμετοχής

Η οργάνωση των παραμέτρων για τα TMFG μπλοκ που είναι αποθηκευμένοι στις μνήμες ROM φαίνεται στον Πίνακα 7.2.

ip_0ODD		**ip_0EVEN**		**ip_1**	
$rise\_start_{ODD0}$	$rise\_slope_{ODD0}$	$rise\_start_{EVEN0}$	$rise\_slope_{EVEN0}$	$rise\_start_0$	$rise\_slope_0$
$fall\_start_{ODD0}$	$fall\_slope_{ODD0}$	$fall\_start_{EVEN0}$	$fall\_slope_{EVEN0}$	$fall\_start_0$	$fall\_slope_0$
$rise\_start_{ODD1}$	$rise\_slope_{ODD1}$	$rise\_start_{EVEN1}$	$rise\_slope_{EVEN1}$	$rise\_start_1$	$rise\_slope_1$
$fall\_start_{ODD1}$	$fall\_slope_{ODD1}$	$fall\_start_{EVEN1}$	$fall\_slope_{EVEN1}$	$fall\_start_1$	$fall\_slope_1$
$rise\_start_{ODD2}$	$rise\_slope_{ODD2}$	$rise\_start_{EVEN2}$	$rise\_slope_{EVEN2}$	$rise\_start_2$	$rise\_slope_2$
$fall\_start_{ODD2}$	$fall\_slope_{ODD2}$	$fall\_start_{EVEN2}$	$fall\_slope_{EVEN2}$	$fall\_start_2$	$fall\_slope_2$
		$rise\_start_{EVEN3}$	$rise\_slope_{EVEN3}$	$rise\_start_3$	$rise\_slope_3$
				$fall\_start_3$	$fall\_slope_3$
				$rise\_start_4$	$rise\_slope_4$
				$fall\_start_4$	$fall\_slope_4$
				$rise\_start_5$	$rise\_slope_5$
				$fall\_start_5$	$fall\_slope_5$
				$rise\_start_6$	$rise\_slope_6$

Πίνακας 7.2: Οργάνωση μνήμης παραμέτρων των TMFG μπλοκ

Η οργάνωση των μνημών ROM που φυλάσσει τα μονότιμα σύνολα (*singletons*) υποδεικνύεται στον Πίνακα 7.3 παρακάτω.

Στη μνήμη αυτή, η οποία είναι χωρισμένη σε δύο κομμάτια (ODD και EVEN) αποθηκεύουμε τα εξής:

- το συμπέρασμα (*consequent*) κάθε IF-THEN κανόνα (cns_{ij})
- την πληροφορία για το ποιες από τις δύο υποθέσεις (*antecedents*) κάθε IF-THEN κανόνα είναι ενεργές ($active\_sel_{ij}$)

Ο δείκτης *i* αναφέρεται στο ασαφές σύνολο της εισόδου ip_0 που εμπλέκεται στον κανόνα, ενώ ο δείκτης *j* σε αυτό της εισόδου ip_1. Και οι δύο δείκτες έχουν ως τιμές τα ονόματα των ασαφών συνόλων που προτάθηκαν στο Σχήμα 7.3 για τις δύο εισόδους.

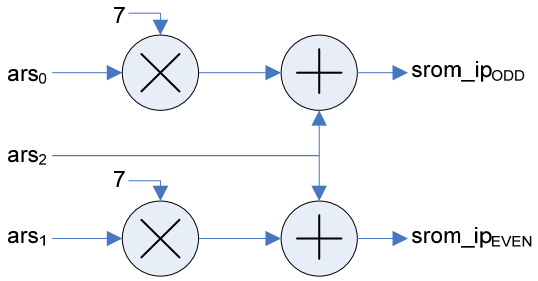
ODD		**EVEN**	
$cns_{ODD0\&0}$	$active\_sel_{ODD0\&0}$	$cns_{EVEN0\&0}$	$active\_sel_{EVEN0\&0}$
$cns_{ODD0\&1}$	$active\_sel_{ODD0\&1}$	$cns_{EVEN0\&1}$	$active\_sel_{EVEN0\&1}$
$cns_{ODD0\&2}$	$active\_sel_{ODD0\&2}$	$cns_{EVEN0\&2}$	$active\_sel_{EVEN0\&2}$
$cns_{ODD0\&3}$	$active\_sel_{ODD0\&3}$	$cns_{EVEN0\&3}$	$active\_sel_{EVEN0\&3}$
$cns_{ODD0\&4}$	$active\_sel_{ODD0\&4}$	$cns_{EVEN0\&4}$	$active\_sel_{EVEN0\&4}$
$cns_{ODD0\&5}$	$active\_sel_{ODD0\&5}$	$cns_{EVEN0\&5}$	$active\_sel_{EVEN0\&5}$
$cns_{ODD0\&6}$	$active\_sel_{ODD0\&6}$	$cns_{EVEN0\&6}$	$active\_sel_{EVEN0\&6}$
$cns_{ODD1\&0}$	$active\_sel_{ODD1\&0}$	$cns_{EVEN1\&0}$	$active\_sel_{EVEN1\&0}$
$cns_{ODD1\&1}$	$active\_sel_{ODD1\&1}$	$cns_{EVEN1\&1}$	$active\_sel_{EVEN1\&1}$
$cns_{ODD1\&2}$	$active\_sel_{ODD1\&2}$	$cns_{EVEN1\&2}$	$active\_sel_{EVEN1\&2}$
$cns_{ODD1\&3}$	$active\_sel_{ODD1\&3}$	$cns_{EVEN1\&3}$	$active\_sel_{EVEN1\&3}$
$cns_{ODD1\&4}$	$active\_sel_{ODD1\&4}$	$cns_{EVEN1\&4}$	$active\_sel_{EVEN1\&4}$
$cns_{ODD1\&5}$	$active\_sel_{ODD1\&5}$	$cns_{EVEN1\&5}$	$active\_sel_{EVEN1\&5}$
$cns_{ODD1\&6}$	$active\_sel_{ODD1\&6}$	$cns_{EVEN1\&6}$	$active\_sel_{EVEN1\&6}$
$cns_{ODD2\&0}$	$active\_sel_{ODD2\&0}$	$cns_{EVEN2\&0}$	$active\_sel_{EVEN2\&0}$
$cns_{ODD2\&1}$	$active\_sel_{ODD2\&1}$	$cns_{EVEN2\&1}$	$active\_sel_{EVEN2\&1}$
$cns_{ODD2\&2}$	$active\_sel_{ODD2\&2}$	$cns_{EVEN2\&2}$	$active\_sel_{EVEN2\&2}$
$cns_{ODD2\&3}$	$active\_sel_{ODD2\&3}$	$cns_{EVEN2\&3}$	$active\_sel_{EVEN2\&3}$
$cns_{ODD2\&4}$	$active\_sel_{ODD2\&4}$	$cns_{EVEN2\&4}$	$active\_sel_{EVEN2\&4}$
$cns_{ODD2\&5}$	$active\_sel_{ODD2\&5}$	$cns_{EVEN2\&5}$	$active\_sel_{EVEN2\&5}$
$cns_{ODD2\&6}$	$active\_sel_{ODD2\&6}$	$cns_{EVEN2\&6}$	$active\_sel_{EVEN2\&6}$
		$cns_{EVEN3\&0}$	$active\_sel_{EVEN3\&0}$
		$cns_{EVEN3\&1}$	$active\_sel_{EVEN3\&1}$
		$cns_{EVEN3\&2}$	$active\_sel_{EVEN3\&2}$
		$cns_{EVEN3\&3}$	$active\_sel_{EVEN3\&3}$
		$cns_{EVEN3\&4}$	$active\_sel_{EVEN3\&4}$
		$cns_{EVEN3\&5}$	$active\_sel_{EVEN3\&5}$
		$cns_{EVEN3\&6}$	$active\_sel_{EVEN3\&6}$

Πίνακας 7.3: Οργάνωση μνήμης ασαφών συνόλων μοναδιαίας τιμής

7.3.6 Αντιστοίχηση Διευθύνσεων Συμπερασμάτων

Το μπλοκ της αντιστοίχησης διευθύνσεων συμπερασμάτων (Consequent Address Mapper - CAM) χρειάζεται απλά ώστε από τα *ars* σήματα του ADG να προκύψουν οι διευθύνσεις

srom_ip$_{ODD}$ και *srom_ip*$_{EVEN}$ για την ODD και EVEN μνήμη ROM μονότιμων συνόλων (Singletons ROM - SROM) αντίστοιχα. Το μπλοκ διάγραμμα φαίνεται στο Σχήμα 7.8. Τα σήματα *ars*$_0$ και *ars*$_1$ πολλαπλασιάζονται επί τον αριθμό των ασαφών συνόλων της εισόδου *ip*$_1$, που στην περίπτωσή μας είναι 7. Στα αποτελέσματα προστίθεται το σήμα *ars*$_2$.



Σχήμα 7.8: Αντιστοίχηση Διευθύνσεων Συμπερασμάτων (CAM)

7.3.7 Επιλογέας Κανόνων

Ο σκοπός αυτού του μπλοκ (βλ. Σχήμα 7.9) είναι να επιτρέψει την επιλογή των επιθυμητών κανόνων που αποθηκεύονται στη ασαφή βάση κανόνων ή διαφορετικά είναι να αξιοποιεί την πληροφορία για το ποιες υποθέσεις του εξεταζόμενου κανόνα είναι ενεργές και να διαμορφώνει κατάλληλα τις αριθμητικές τιμές των γλωσσικών μεταβλητών. Οι επιλεχθέντες κανόνες συμβάλουν στο τελικό αποτέλεσμα για ένα σύνολο δεδομένων εισόδου. Οι συνδυασμοί των κανόνων αποθηκεύονται στη μνήμη που κρατάει τα μονότιμα σύνολα. Μιας και οι εξεταζόμενοι κανόνες σε κάθε κύκλο ρολογιού είναι δύο, λόγω της δομής ODD-EVEN που εφαρμόσαμε παραπάνω, χρειαζόμαστε και δύο ARS (ODD-EVEN), η λειτουργία των οποίων είναι πανομοιότυπη. Η επιλογή εκτελείται με τον ακόλουθο τρόπο που αναλύεται παρακάτω.

Οι προϋποθέσεις του κάθε κανόνα είτε αποθηκεύονται στη μνήμη ROM ως λογικό 0, είτε λογικό 1, κατά συνέπεια ενεργοποιώντας ή απενεργοποιώντας ένα μέρος του κανόνα είτε ολόκληρο τον κανόνα. Λόγω της ODD-EVEN μεθόδου που ακολουθείται σε αυτήν την αρχιτεκτονική, είναι προφανές ότι απαιτούνται δύο παρόμοια μπλοκ επιλογών κανόνων, δεδομένου ότι σε κάθε κύκλο ρολογιού εξετάζονται δύο κανόνες.

Το μπλοκ επιλογής κανόνων για τον ODD κανόνα δέχεται σαν εισόδους τα *alpha*$_0$*odd* και *alpha*$_1$ των αντίστοιχων γεννητριών τραπεζοειδών συναρτήσεων συμμετοχής και το σήμα επιλογής *active_sel* της μνήμης ROM που φυλάσσει τα ODD συμπεράσματα (*sel_odd*). Γενικά ισχύει:

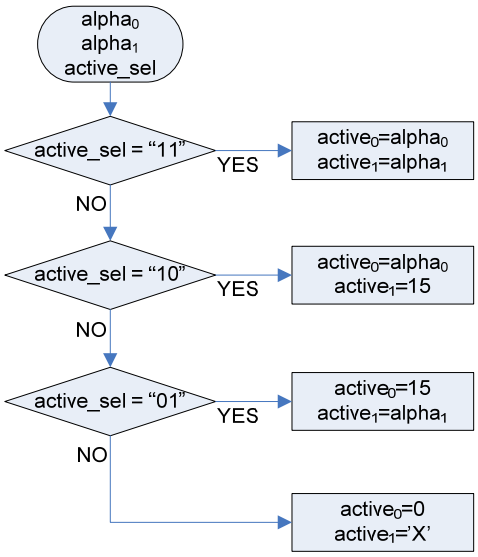
- alpha$_0$=alpha$_0$odd
- alpha$_1$=alpha$_1$
- active_sel=sel_odd

Παρόμοια, το μπλοκ επιλογής κανόνων για τον EVEN κανόνα δέχεται σαν εισόδους τα *alpha*$_0$*even* και *alpha*$_1$ των αντίστοιχων γεννητριών τραπεζοειδών συναρτήσεων συμμετοχής και το σήμα επιλογής *active_sel* της μνήμης ROM που φυλάσσει τα EVEN συμπεράσματα (*sel_even*). Αντίστοιχα θα ισχύει:

- $alpha_0$=$alpha_0$even
- $alpha_1$=$alpha_1$
- active_sel=sel_even

Πιο συγκεκριμένα, αναφορικά με το Σχήμα 7.9, όταν η υπόθεση ενός κανόνα δεν είναι ενεργή, τότε αυτή δεν θα συμβάλει στη διαμόρφωση της δύναμης (βάρος) του κανόνα *(rule firing strength)*. Υπό τον όρο ότι η συμβολή βάρους του κανόνα (*theta τιμή*) εξάγεται εφαρμόζοντας τον τελεστή ελαχίστου σε όλες τις *alpha* τιμές, μπορούμε να αγνοήσουμε την *alpha* τιμή της επιλεγμένης υπόθεσης θέτοντας τη στη μέγιστη αξία της (ή μέγιστη δυνατή τιμή). Η πρακτική αυτή λειτουργεί όταν έχουμε μια τουλάχιστον ενεργή υπόθεση. Στην περίπτωσή μας όπου η ανάλυση του βαθμού αληθείας είναι 4-bit (μη προσημασμένος αριθμός αφού η τιμή $alpha \in R^+$) η μέγιστη αξία είναι 2^4-1=15.

Εάν όλες οι υποθέσεις του κανόνα είναι ανενεργές, τουλάχιστον μια *alpha* τιμή πρέπει να τεθεί στην ελάχιστη τιμή, έτσι ώστε η δύναμη του κανόνα να προκύψει μηδέν.

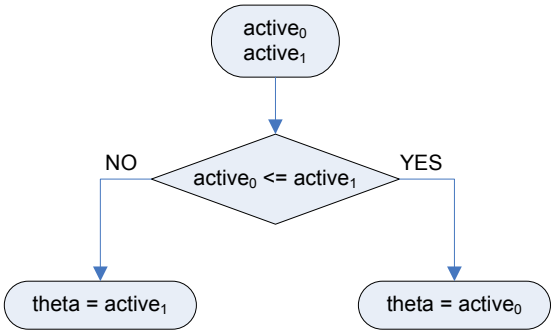


Σχήμα 7.9: Διάγραμμα ροής του επιλογέα κανόνων

7.3.8 Τελεστής Ελαχίστου (MIN)

Το μπλοκ αυτό εξάγει τον ελάχιστο *(minimum)* μεταξύ δύο μη προσημασμένων (*unsigned*) αριθμών, ίδιας ακρίβειας, τους οποίους δέχεται σαν είσοδο. Ουσιαστικά, αποτελεί τη *hardware* υλοποίηση της *Mamdani t-norm*[*]. Η έξοδος ισούται με τη δύναμη του κανόνα *(theta τιμή)*. Φυσικά, μιας και εξετάζουμε δύο κανόνες (ODD-EVEN) σε κάθε κύκλο ρολογιού, θα έχουμε και εδώ δύο MIN (ODD-EVEN), κάθε ένα από τα οποία εξάγει την ελάχιστη από τις εξόδους του αντίστοιχου ARS. Το διάγραμμα ροής του MIN μπλοκ φαίνεται στο Σχήμα 7.10.

[*] Mamdani t-norm (t(a, b) = min(a, b))



Σχήμα 7.10: Διάγραμμα ροής του MIN μπλοκ

7.3.9 Πολλαπλασιαστές

Οι πολλαπλασιαστές που χρησιμοποιούνται στην παρούσα αρχιτεκτονική έχουν αναφερθεί στην παράγραφο §6.4.9.

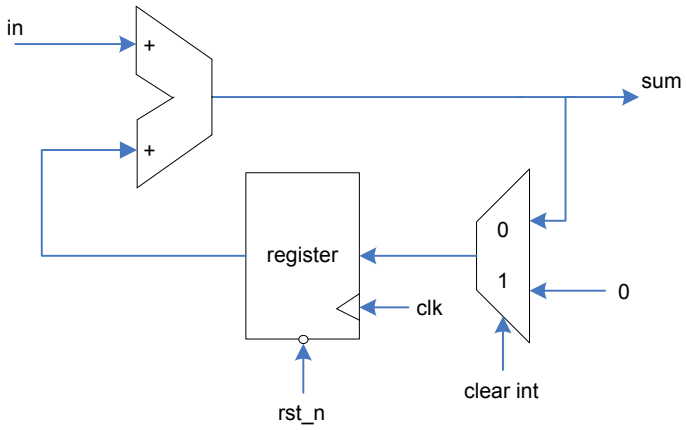
7.3.10 Προσημασμένος και Μη προσημασμένος Αθροιστής

Είναι ένας προσημασμένος και ένας μη προσημασμένος αντίστοιχα αθροιστής (*signed and unsigned Adder*) ο οποίος αποτελεί μέρος της διαδικασίας απο-ασαφοποίησης.

7.3.11 Προσημασμένος και Μη προσημασμένος Ολοκληρωτής

Είναι ένας προσημασμένος και ένας μη προσημασμένος ολοκληρωτής ο οποίος αποτελεί μέρος της διαδικασίας απο-ασαφοποίησης. Όπως έχει αναφερθεί παραπάνω, χρειάζονται $2^2 / 2 = 2$ κύκλοι ρολογιών για να εξεταστούν οι τέσσερις ενεργοί κανόνες. Τα αποτελέσματα του πρώτου κύκλου αποθηκεύονται στη μνήμη του ολοκληρωτή και προστίθενται στα αποτελέσματα του δεύτερου κύκλου. Η διαδικασία αυτή επαναλαμβάνεται συνέχεια.

Για τον περιοδικό μηδενισμό της μνήμης μετά το τέλος του δεύτερου κύκλου χρησιμοποιείται ο μετρητής (*cnt*) του ADG (βλ. Σχήμα 7.4), τον οποίο δέχονται σαν είσοδο (*clear int*) οι δύο ολοκληρωτές, με την προσθήκη της κατάλληλης καθυστέρησης ώστε ο δεύτερος ολοκληρωτής να μηδενίζεται ένα κύκλο ρολογιού μετά το μηδενισμό του πρώτου, αφού βρίσκεται στην επόμενη βαθμίδα αγωγού όπως φαίνεται στα Σχήματα 7.17.1 και 7.2.



Σχήμα 7.11: Κυκλωματικό διάγραμμα Integrator

7.3.12 Πολλαπλασιαστές Odd-Even

Υπολογίζει το συνεπαγόμενο ασαφές σύνολο του ενεργού κανόνα εκείνου που εμπλέκει το ODD-EVEN αντίστοιχα ασαφές σύνολο της εισόδου ip_0. Αφού χρησιμοποιούμε μονότιμο σύνολο για το συμπέρασμα του κανόνα, το αποτέλεσμα της συνεπαγωγής θα είναι επίσης ένα μονότιμο σύνολο. Για τον υπολογισμό χρησιμοποιούμε τον τελεστή γινομένου ή *Product t-norm*[*]. Η κυκλωματική υλοποίησή του είναι ένας πολλαπλασιαστής.

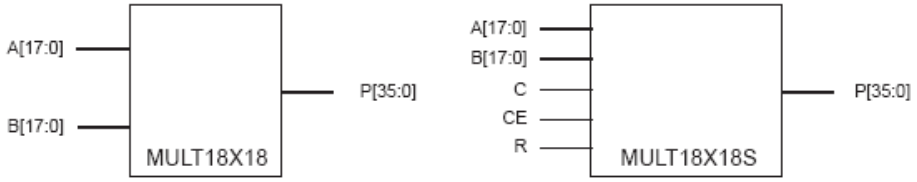
Η οικογένεια των FPGA devices, Spartan-3 [44] της εταιρίας *Xilinx*, η οποία χρησιμοποιήθηκε για την υλοποίηση του ασαφούς ελεγκτή παρέχει ενσωματωμένους πολλαπλασιαστές *(embedded multipliers)*, των οποίων ο αριθμός ποικίλει ανάλογα με το μοντέλο της οικογένειας. Συγκεκριμένα, το μοντέλο Spartan-3 XC3S1500-4 FG676, το οποίο επιλέχθηκε, περιέχει 32 ενσωματωμένους πολλαπλασιαστές. Αυτοί πολλαπλασιάζουν δύο αριθμούς, καθένας εκ των οποίων είναι 18-bit προσημασμένος ή 17-bit μη προσημασμένος. Ο πολλαπλασιασμός γίνεται σε αριθμητική συμπληρώματος του 2 *(two's complement)*, με εφαρμογή του τροποποιημένου αλγορίθμου του Booth [69], χρησιμοποιώντας πολυπλέκτες *(multiplexers)* για τη δημιουργία των μερικών γινομένων *(partial products)*. Η θέση των πολλαπλασιαστών στο FPGA φαίνεται στην παράγραφο §5.8.2 (Σχήμα 5.6). Όπως φαίνεται στο σχήμα, υπάρχουν Διαμορφώσιμα Λογικά Μπλοκ (*Configurable Logic Blocks* - CLBs) και μνήμες RAMs σε κοντινή απόσταση από τους πολλαπλασιαστές ώστε να μειωθούν οι διαδρομές με τη μέγιστη καθυστέρηση *(critical path delay)* των συνδέσεων των εισόδων και των εξόδων αυτών με τη λογική που παρεμβάλλεται.

Υπάρχουν δύο τύποι πολλαπλασιαστών, ο ασύγχρονος (MULT18X18) και ο σύγχρονος (MULT18X18S) [70], όπως δείχνει το Σχήμα 7.12, οι οποίοι ουσιαστικά αναφέρονται στον ίδιο πολλαπλασιαστή. Η μόνη διαφορά μεταξύ των δύο είναι ότι στον σύγχρονο χρησιμοποιείται ο καταχωρητής εξόδου του πολλαπλασιαστή. Αυτός μας δίνει τη

[*] Product t-norm (t(a,b) = a·b)

δυνατότητα προσθήκης μίας βαθμίδας αγωγού, με αποτέλεσμα να μειωθεί η διαδρομή με τη μέγιστη καθυστέρηση και να αυξηθεί η συχνότητα του ρολογιού λειτουργίας του κυκλώματος. Συνεπώς, προτιμήθηκε η χρησιμοποίηση σύγχρονων πολλαπλασιαστών ώστε να πετύχουμε μεγαλύτερη συχνότητα λειτουργίας με κόστος την αύξηση της καθυστέρησης (*latency*) κατά μία περίοδο ρολογιού. Αυτό υποδηλώνεται στα Σχήματα 7.1 και 7.2 των αρχιτεκτονικών, με την τοποθέτηση ενός σταδίου αγωγού στο μέσο του κάθε πολλαπλασιαστή.

Στην περίπτωση που εξετάζουμε ο κάθε ένας από τους δύο πολλαπλασιαστές (ODD-EVEN) πολλαπλασιάζει ένα προσημασμένο (*signed*) αριθμό 8-bit (το συμπέρασμα του κανόνα, *cns*) με ένα μη προσημασμένο (*unsigned*) αριθμό 4-bit (τη δύναμη του κανόνα, *theta*). Το αποτέλεσμα είναι ένας προσημασμένος αριθμός 8+(4+1)=13-bit (η τιμή της συνεπαγωγής του κανόνα, *implication*).



Σχήμα 7.12: Ασύγχρονος & Σύγχρονος Ενσωματωμένος Πολλαπλασιαστής στα Spartan-3 FPGA ολοκληρωμένα κυκλώματα της Xilinx

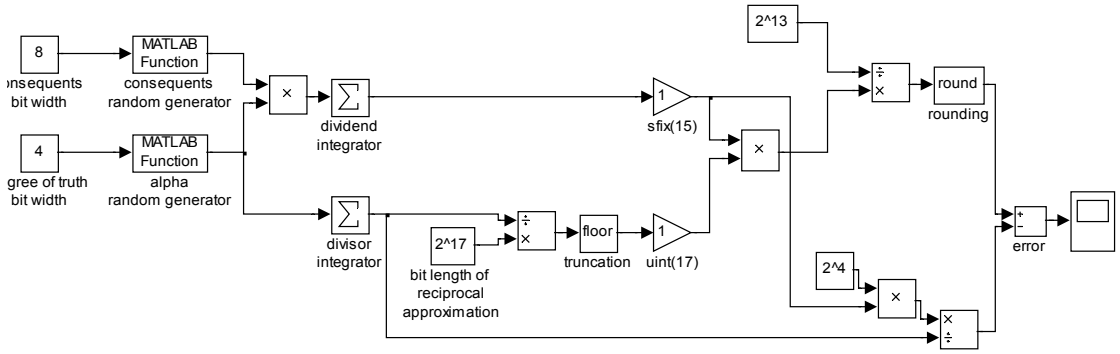
7.3.13 Διαιρέτης

Το μοντέλο για το μπλοκ της διαίρεσης που χρησιμοποιείται στην αρχιτεκτονική του ελεγκτή είναι ανάλογο με το μοντέλο που περιγράφηκε στην παράγραφο §6.4.11. Η ουσιαστική διαφορά στο μοντέλο που χρησιμοποιείτε εδώ επικεντρώνεται στην ακρίβεια του αντιστρόφου.

7.3.13.1 Πίνακας Αναζήτησης Αντιστρόφου (Reciprocal LUT)

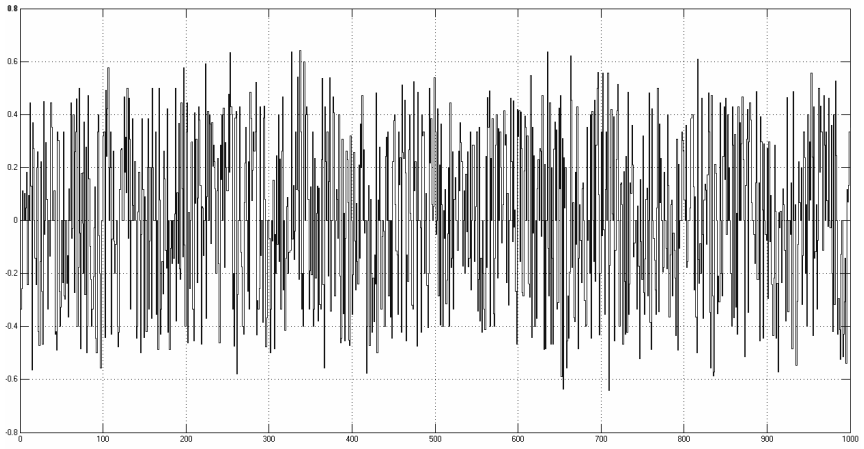
Η ακρίβεια του αντιστρόφου (συνάρτηση *1/x*) υπολογίστηκε πειραματικά στα 17-bit με τη βοήθεια του MATLAB - Simulink ώστε να μην έχουμε σφάλμα στον υπολογισμό της 12-bit εξόδου του κυκλώματος.

Το μοντέλο Simulink που χρησιμοποιήθηκε για την εκτίμηση του αντιστρόφου δίνεται στο Σχήμα 7.13.



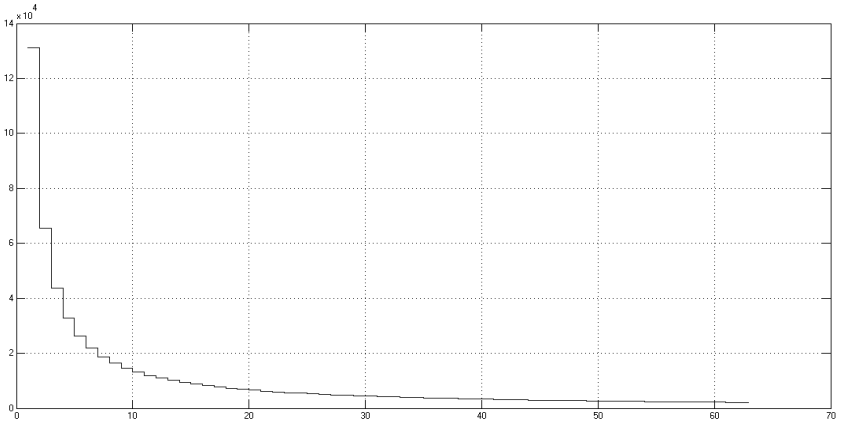
Σχήμα 7.13: Μοντέλο εκτίμησης της ακρίβειας του αντιστρόφου

Το σφάλμα εξόδου που προκύπτει από το παραπάνω μοντέλο φαίνεται στο Σχήμα 7.14 που ακολουθεί παρακάτω.



Σχήμα 7.14: Σφάλμα εξόδου

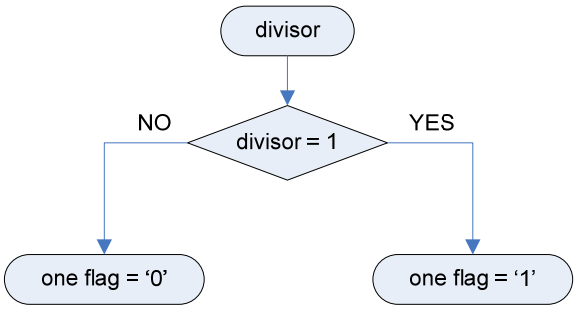
Τα περιεχόμενα του πίνακα αναζήτησης αντιστρόφου (Reciprocal LUT - RLUT) αποτυπώνονται στη γραφική παράσταση στο σχήμα που ακολουθεί (βλ. Σχήμα 7.15).



Σχήμα 7.15: Περιεχόμενα Μνήμης Αντιστρόφου

7.3.13.2 Έλεγχος Μοναδιαίου Παρανομαστή (IsOne)

Το μπλοκ αυτό (Is One) ελέγχει αν ο διαιρέτης στον τύπο απο-ασαφοποίησης είναι ίσος με τη μονάδα. Η ειδική αυτή περίπτωση διαίρεσης εξετάζεται διότι το *1/1* θα απαιτούσε 18-bit ακρίβειας στο RLUT αυξάνοντας, κατά πολύ το μέγεθος της απαιτούμενης μνήμης ενώ παράλληλα θα απαιτούσε και μεγαλύτερο πολλαπλασιαστή για τον υπολογισμό της εξόδου. Το διάγραμμα ροής του μπλοκ φαίνεται στο Σχήμα 7.16.



Σχήμα 7.16: Διάγραμμα ροής του Ελεγκτή Μοναδιαίου Παρανομαστή

7.3.13.3 Πολλαπλασιαστής

Είναι ένας πολλαπλασιαστής που εκτελεί τον πολλαπλασιασμό μεταξύ του διαιρετέου του τύπου απο-ασαφοποίησης (6-bit μη προσημασμένος αριθμός) και της προσέγγισης του αντιστρόφου του διαιρέτη που προκύπτει από την έξοδο του RLUT (17-bit μη προσημασμένος αριθμός). Όπως και στην περίπτωση των ODD-EVEN πολλαπλασιαστών (βλ. §7.3.12), έτσι και για την υλοποίηση αυτού του πολλαπλασιαστή χρησιμοποιείται ένας ενσωματωμένος σύγχρονος πολλαπλασιαστής MULT18X18S [70].

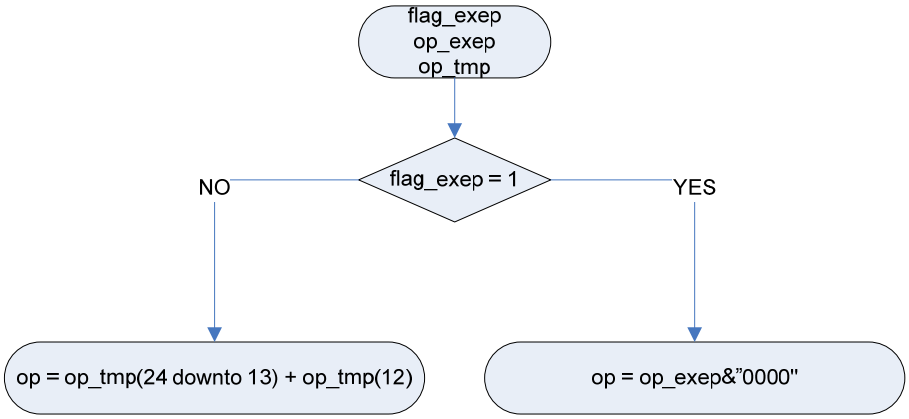
7.3.13.4 Παραγωγή Τελικής Εξόδου (Fix Output)

Στο μπλοκ αυτό γίνεται διόρθωση του αποτελέσματος του πολλαπλασιασμού και παράγεται η τελική έξοδος. Το διάγραμμα ροής φαίνεται στο Σχήμα 7.17.

Το μπλοκ δέχεται για είσοδο τα εξής σήματα:

- **flag_exep** = *one flag*, είναι η σημαία που παράγει στην έξοδο το *Is one μπλοκ* και ελέγχει αν έχουμε διαίρεση με τη μονάδα *(1)* ή όχι *(0)*
- **op_exep** = *divident*, είναι η τιμή του διαιρετέου (μόνο τα *8-bit*, μιας και αφού ο διαιρέτης είναι μονάδα, δηλαδή όλα τα *alpha* είναι *0* εκτός από *1*, δεν χρειάζονται περισσότερα bits για την αναπαράσταση του διαιρετέου)
- **op_tmp** = *mul*, είναι η έξοδος του πολλαπλασιαστή

Στην περίπτωση που ο διαιρέτης είναι ίσος με τη μονάδα, τότε περνάμε κατευθείαν στην έξοδο την τιμή του διαιρετέου (8-bit) και προσθέτουμε και τα 4-bit του κλασματικού μέρους. Σε αντίθετη περίπτωση, παίρνουμε από το $24^{ο}$ έως το $13^{ο}$ bit του αποτελέσματος του πολλαπλασιασμού και στρογγυλοποιούμε (*rounding*) προσθέτοντας το $12^{ο}$ bit.



Σχήμα 7.17: Διάγραμμα ροής της Παραγωγής Τελικής Εξόδου

7.3.14 Εναλλακτική Υλοποίηση με Συνάρτηση Συμμετοχής βασισμένη σε ROM

Μετά από την ολοκλήρωση της περιγραφής των μπλοκ της $1^{ης}$ αρχιτεκτονικής του Σχήματος 7.1 εξετάζουμε τα σημεία στα οποία διαφοροποιείται η $2^{η}$ αρχιτεκτονική του Σχήματος 7.2. Αυτά είναι:

- Αντί του TMFG υπάρχει η μνήμη των συναρτήσεων συμμετοχής (Membership Function ROM - MFROM). Αυτή παίρνει ως διευθύνσεις τις εισόδους ip_0 και ip_1 και βγάζει στην έξοδο τις τιμές των συναρτήσεων συμμετοχής των ενεργών ασαφών συνόλων κάθε εισόδου στο σημείο που υποδεικνύει η αντίστοιχη είσοδος. Τα περιεχόμενα της φαίνονται στον Πίνακα 7.4.
- Δεν χρειάζεται η έξοδος *region bus* του ARS.
- Αντικαθιστούμε την είσοδο *region bus* του ADG με την *alpha bus*. Στην έξοδο, συνεπώς, θα έχουμε το σήμα *alpha gen bus* αντί του *region gen bus*. Δεν αλλάζει η λειτουργία του μπλοκ.

ip_0		**ip_1**	
$alpha_0(-128)_{ODD}$	$alpha_0(-128)_{EVEN}$	$alpha_1(-128)_{ODD}$	$alpha_1(-128)_{EVEN}$
$alpha_0(-127)_{ODD}$	$alpha_0(-127)_{EVEN}$	$alpha_1(-127)_{ODD}$	$alpha_1(-127)_{EVEN}$
...	...	...	...
$alpha_0(126)_{ODD}$	$alpha_0(126)_{EVEN}$	$alpha_1(126)_{ODD}$	$alpha_1(126)_{EVEN}$
$alpha_0(127)_{ODD}$	$alpha_0(127)_{EVEN}$	$alpha_1(127)_{ODD}$	$alpha_1(127)_{EVEN}$

Πίνακας 7.4: Τα περιεχόμενα της MFROM

7.4 Διαδικασία Σχεδίασης

Η διαδικασία σχεδίασης που αναφέρθηκε στο Κεφάλαιο 6 (βλ. §6.5, Σχήμα 6.17) ακολουθήθηκε και εδώ. Στις παραγράφους που ακολουθούν αναλύονται τα διάφορα βήματα της διαδικασίας σχεδίασης.

7.4.1 Προδιαγραφές Σχεδίασης

Ο ορισμός των προδιαγραφών που θα πρέπει να έχει το προς υλοποίηση κύκλωμα αποτελεί το πρώτο βήμα στη διαδικασία σχεδίασης. Αυτές συνήθως υπαγορεύονται από περιορισμούς στο υλικό αλλά και από την απαιτούμενη χρήση. Στη συγκεκριμένη περίπτωση, αν και ο έμμεσος στόχος ήταν η δημιουργία ενός όσο το δυνατόν παραμετρικού κώδικα, τέθηκε ως άμεσος στόχος η δημιουργία ενός ασαφούς ελεγκτή με τα χαρακτηριστικά του Πίνακα 7.1.

7.4.2 Μοντελοποίηση Συστήματος Ψηφιακού Ασαφούς Ελεγκτή

Προτού ξεκινήσει η εγγραφή κώδικα VHDL και αφού έχουν τεθεί οι περιορισμοί είναι χρήσιμο να γίνει πρώτα η μοντελοποίηση του συστήματος. Για το σκοπό αυτό σχεδιάστηκε αρχικά η λειτουργία του κυκλώματος στην πλατφόρμα MATLAB–Simulink (βλ. Σχήμα 7.18–7.21).

Με τη δημιουργία του μοντέλου του συστήματος στο Simulink όχι μόνο διευκολύνθηκε ο χωρισμός του σε επιμέρους μπλοκ αλλά δόθηκε και η δυνατότητα παραγωγής διανυσμάτων δοκιμής (test vectors) για τον έλεγχο των επόμενων σταδίων της σχεδίασης.

7.4.3 Περιγραφή σε γλώσσα VHDL σε RTL επίπεδο

Μετά τη μοντελοποίηση του συστήματος έπεται η περιγραφή αυτού σε VHDL, η οποία αποτελεί μια γλώσσα περιγραφής υλικού (Hardware Description Language) με ευρύτατη χρήση τα τελευταία χρόνια τόσο στο βιομηχανικό όσο και στον επιστημονικό τομέα. Η VHDL επιτρέπει την περιγραφή μιας λειτουργίας με διάφορους τρόπους (Συμπεριφορικός τρόπος - Behavioral, Επιπέδου Μεταφοράς Καταχωρητή – Register Transfer Level (RTL), Επίπεδου Λογικών Πυλών - Gate level). Η παρούσα σχεδίαση των δομικών μονάδων του ασαφούς ελεγκτή έγινε σε RTL περιγραφή, ενώ το περιβάλλον ελέγχου σωστής λειτουργίας ή testbench (βλ. Παράρτημα Α.2) του ελεγκτή με διανύσματα δοκιμης (test vectors), σε behavioral περιγραφή (π.χ. οι εντολές WAIT δεν γίνονται σύνθεση σε λογικό επίπεδο).

Όπως φαίνεται και από τους ενδεικτικούς κώδικες που παρουσιάζονται στο Παράρτημα Α.2, πραγματοποιήθηκε ο διαχωρισμός του συστήματος σε διαφορετικές δομικές μονάδες (μπλοκ), σύμφωνα με τις αρχιτεκτονικές που παρουσιάστηκαν στο Σχήμα 7.1 και Σχήμα 7.2. Σε κάθε μπλοκ δόθηκε έμφαση στη διατήρηση της παραμετρικότητας της σχεδίασης. Για το σκοπό αυτό, έγινε ευρεία χρήση των εντολών *Generate* και *Loop* της VHDL όπου τα όρια επαναλήψεων καθορίζονταν από εντολές ανάθεσης παραμέτρων (Generic). Επίσης, με τη χρήση της τελευταίας εντολής καθορίστηκαν και τα μεγέθη των διαφόρων σημάτων.

7.4.4 Προσομοίωση RTL

Μετά την ολοκλήρωση της περιγραφής του συστήματος σε VHDL, ακολούθησε η αποσφαλμάτωση (debugging) αυτού. Για τη διαπίστωση της ορθότητας των

αποτελεσμάτων χρησιμοποιήθηκαν τα διανύσματα δοκιμής που παρήγαγε το αντίστοιχο μοντέλο του συστήματος στο Simulink. Η προσομοίωση και η σύγκριση των αποτελεσμάτων του Simulink με αυτά της VHDL έγινε με τη βοήθεια του εργαλείου Modelsim της εταιρίας Mentor Graphics. Στα Σχήματα 7.22 και 7.23 παρουσιάζονται οι κυματομορφές που προέκυψαν από την RTL προσομοίωση για τις δύο αρχιτεκτονικές των σχημάτων Σχήματα 7.1 και 7.2 αντίστοιχα. Τα ονόματα και οι χρονισμοί των σημάτων στο διάγραμμα ροής δεδομένων (data flow) βρίσκονται σε πλήρη συμφωνία με αυτά των Σχημάτων 7.1 και 7.2.

Όπως γίνεται φανερό από το Σχήμα 7.22, χρησιμοποιούμε δύο ρολόγια για το χρονισμό του κυκλώματος. Το γρήγορο-εσωτερικό (clkx) δουλεύει στα 200MHz (ή 5ns) και αποτελεί το ρολόι εσωτερικής λειτουργίας του ασαφούς ελεγκτή. Το αργό-εξωτερικό ρολόι (clk) δουλεύει στα 100MHz (ή 10ns) και αποτελεί το ρολόι των καταχωρητών εισόδου και εξόδου του ασαφούς ελεγκτή. Όπως αποδείχτηκε στην παράγραφο §7.3.3, αρκούν δύο κύκλοι ρολογιού του ADG ώστε να παραχθούν όλοι οι δυνατοί συνδυασμοί των ενεργών κανόνων που υποδεικνύει ασύγχρονα ο ARS. Συνεπώς, το αργό ρολόι έχει τη μισή συχνότητα του γρήγορου ώστε να κρατάει σταθερές τις εισόδους του ασαφούς ελεγκτή στους καταχωρητές εισόδου όσο χρόνο χρειάζεται η παραγωγή όλων των συνδυασμών των ενεργών κανόνων που αναφέρονται στις συγκεκριμένες εισόδους. Και τα δύο ρολόγια προκύπτουν από το DCM (Digital Control Manager) και βρίσκονται σε φάση μεταξύ τους.

Ένα νέο ζεύγος εισόδων αποθηκεύεται στον καταχωρητή εισόδου, στην αρνητική παρυφή (falling edge) του εξωτερικού ρολογιού. Ο ADG παράγει τους συνδυασμούς των ενεργών κανόνων στη 1^{η}-2^{η} βαθμίδα αγωγού (pipeline stage). Κάθε ένας από τους τρεις TMFG χρειάζονται 4 στάδια αγωγού για να υπολογίσει την alpha τιμή της τραπεζοειδούς συνάρτησης συμμετοχής. Ο υπολογισμός γίνεται παράλληλα με τη διευθυνσιοδότηση (3^{η}–4^{η} βαθμίδα αγωγού) και την ανάγνωση (5^{η}–6^{η} βαθμίδα αγωγού) των SROMs. Ένα κύκλο εσωτερικού ρολογιού αργότερα (6^{η}–7^{η} βαθμίδα αγωγού), αφότου έχει γίνει η επεξεργασία του ARS και του MIN για το εξεταζόμενο ζεύγος των ενεργών κανόνων, οι theta τιμές αυτών (αποτέλεσμα του Min τελεστή στις alpha τιμές) μαζί με τα συμπεράσματα (συμπεράσματα) και το σήμα για το μηδενισμό των ολοκληρωτών γίνονται διαθέσιμα στο Inference & Defuzzification μέρος του κυκλώματος. Η συνεπαγωγή (implication) κάθε κανόνα προκύπτει δύο κύκλους εσωτερικού ρολογιού αργότερα (8^{η}–9^{η} βαθμίδα αγωγού), μαζί με το άθροισμα των theta values. Στον επόμενο κύκλο (9^{η}–10^{η} βαθμίδα αγωγού), υπολογίζεται το άθροισμα των συνεπαγωγών. Στη διάρκεια αυτή ο μη προσημασμένος ολοκληρωτής (Unsigned Integrator) εξάγει την τιμή του διαιρέτη (divisor) (10^{η} βαθμίδα αγωγού). Η τιμή του διαιρετέου (dividend) προκύπτει από τον προσημασμένο ολοκληρωτή (Signed Integrator) στον επόμενο κύκλο (11^{η} βαθμίδα αγωγού) μαζί με την εκτίμηση του αντιστρόφου από το Reciprocal LUT. Ο πολλαπλασιασμός του διαιρετέου με τον αντίστροφο και ο έλεγχος Is One γίνονται στη 12^{η} βαθμίδα αγωγού, όπου πραγματοποιείται και η διόρθωση της εξόδου στο FIX Output μπλοκ. Τέλος, η διορθωμένη έξοδος αποθηκεύεται στον καταχωρητή εξόδου (output register) στη θετική ακμή (rising edge) του εξωτερικού ρολογιού (13^{η} βαθμίδα αγωγού).

Ο συνολικός χρόνος που απαιτείται για την παραγωγή της εξόδου στον καταχωρητή εξόδου, ξεκινώντας από τη στιγμή που αποθηκεύονται οι καινούριες είσοδοι στον καταχωρητή εισόδου είναι 13 στάδια αγωγών ή 65ns. Συνεπώς, έχουμε δειγματοληψία εισόδου κάθε 10ns και 65ns καθυστέρηση (latency). Τα παραπάνω ισχύουν για το Μοντέλο Ασαφούς Ελεγκτή με Αριθμητική Συνάρτηση Συμμετοχής (βλ. Σχήμα 7.1, 7.22).

Για το Μοντέλο Ασαφούς Ελεγκτή με Συνάρτηση Συμμετοχής βασισμένη σε ROM (βλ. Σχήμα 7.2, 7.23) αντίστοιχα, για την παραγωγή της εξόδου απαιτούνται 11 στάδια αγωγών ή 55ns. Συνεπώς, έχουμε δειγματοληψία εισόδου κάθε 10ns και 55ns καθυστέρηση (latency).

Αξίζει να σημειωθεί ότι χωρίς τη χρήση της ODD-EVEN αρχιτεκτονικής που παρουσιάζεται και επιτρέπει την εξέταση δύο κανόνων σε κάθε κύκλο εσωτερικού ρολογιού, θα είχαμε 12 και 10 στάδια αγωγού για την $1^{η}$ και τη $2^{η}$ αρχιτεκτονική αντίστοιχα. Αυτό θα συνέβαινε επειδή δεν θα υπήρχε πλέον ανάγκη για τους Signed/Unsigned Adders. Επίσης, θα μειωνόταν το μέγεθος του κυκλώματος λόγω μείωσης της παραλληλίας, αφού θα χρησιμοποιούσαμε δυο TMFG μπλοκ αντί για τρία, ένα ARS μπλοκ αντί για δυο, ένα MIN μπλοκ αντί για δυο, και ένα Multiplier μπλοκ για την παραγωγή της συνεπαγωγής αντί για δυο. Ωστόσο, η ρυθμό-απόδοση του συστήματος θα αυξανόταν στους 4 κύκλους ρολογιών για έναν πλήρη υπολογισμό ή διαφορετικά ή εισαγωγή νέων δεδομένων θα έπρεπε να γίνεται κάθε 20ns. Αποφασίστηκε λοιπόν ότι ήταν προτιμότερο να θυσιαστεί λίγο μέγεθος ώστε να αυξηθεί η ταχύτητα του ασαφούς ελεγκτή. Αυτό σημαίνει ότι θα έχει τη δυνατότητα να "κλειδώνει" (ελέγχει) πιο γρήγορα συστήματα. Πιο συγκεκριμένα, συστήματα τα οποία έχουν σταθερά χρόνου μεγαλύτερη από πέντε φορές τη σταθερά χρόνου του ελεγκτή ($5 \times 65 = 325$ns για την $1^{η}$ αρχιτεκτονική και $5 \times 55 = 275$ns για τη $2^{η}$ αρχιτεκτονική).

7.4.5 Σύνθεση σε Επίπεδο Λογικών Πυλών (Logic Synthesis)

Αφού διαπιστώθηκε η ορθότητα της VHDL περιγραφής, ακολούθησε η σύνθεση του RTL κώδικα σε επίπεδο λογικών πυλών. Για το σκοπό αυτό χρησιμοποιήθηκε το πρόγραμμα σύνθεσης Synplify Pro της εταιρίας Synplicity. Εκεί, τέθηκαν επιπλέον περιορισμοί (*constrains*) για τη βελτίωση χαρακτηριστικών του κυκλώματος, όπως αύξηση της συχνότητας λειτουργίας. Τα αποτελέσματα φαίνονται στα Σχήματα 7.24-7.30.

7.4.6 Θέση και Δρομολόγηση (Place and Route) του FPGA

Το αρχείο περιγραφής δικτυωμάτων (netlist) του DFLC που παράχθηκε από το εργαλείο σύνθεσης έπρεπε να μετατραπεί σε δυαδικό κώδικα (bitstream) πριν μεταφορτωθεί στο FPGA. Μιας και το ολοκληρωμένο κύκλωμα FPGA στο οποίο προοριζόταν να υλοποιηθεί ο DFLC ήταν το Spartan-3 XC3S1500-4 FG676 της εταιρίας Xilinx, για τη θέση (place) και δρομολόγηση (route) (καθώς και την παράγωγη του bitstream αρχείου) χρησιμοποιήθηκε το πακέτο εφαρμογών ISE Foundation της ίδιας εταιρίας. Τα αποτελέσματα για τις δύο αρχιτεκτονικές φαίνονται στους Πίνακες 7.5 και 7.6.

7.4.7 Προσομοίωση Χρονισμού

Τέλος, ελέγχθηκε η ορθότητα του κυκλώματος που προέκυψε από το εργαλείο θέσης και δρομολόγησης, χρησιμοποιώντας πάλι σαν αναφορά τα διανύσματα δοκιμής που παρήχθησαν από το αντίστοιχο Simulink μοντέλο. Η ανάγκη για επανέλεγχο του τελικού

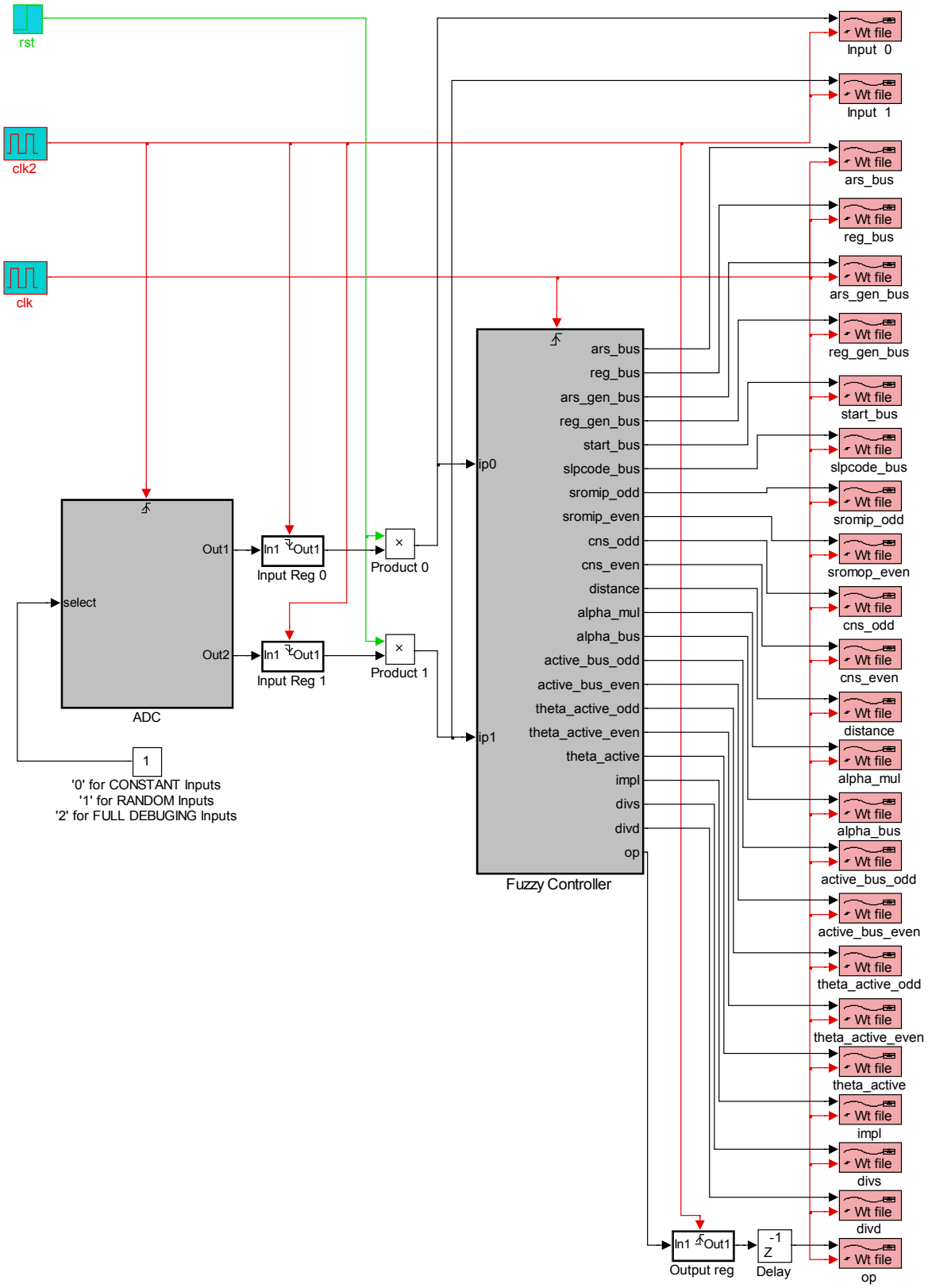
κυκλώματος προκύπτει από το γεγονός ότι σε RTL επίπεδο δεν συνυπολογίζονται στην προσομοίωση οι καθυστερήσεις των διαφόρων σημάτων, εξαιτίας της μη ιδανικότητας των πυλών, των συνδέσεων κλπ, αφού δεν υπάρχει πληροφορία για αυτές. Αντίθετα, το κύκλωμα που προκύπτει μετά την Τοποθέτηση και Δρομολόγηση, ακριβώς επειδή έχει υπολογίσει τις θέσεις των διαφόρων πυλών στο επιλεχθέν FPGA ολοκληρωμένο κύκλωμα, είναι σε θέση να γνωρίζει τις καθυστερήσεις των σημάτων, και άρα αυτές λαμβάνονται υπ' όψιν στη χρονική προσομοίωση. Έτσι, σε RTL προσομοίωση χρησιμοποιούνται delta unit times για τις καθυστερήσεις, ενώ στην προσομοίωση χρονισμού (timing-simulation*) εμπεριέχονται οι πραγματικές χρονικές καθυστερήσεις (delay times) των πυλών.

7.4.8 FPGA

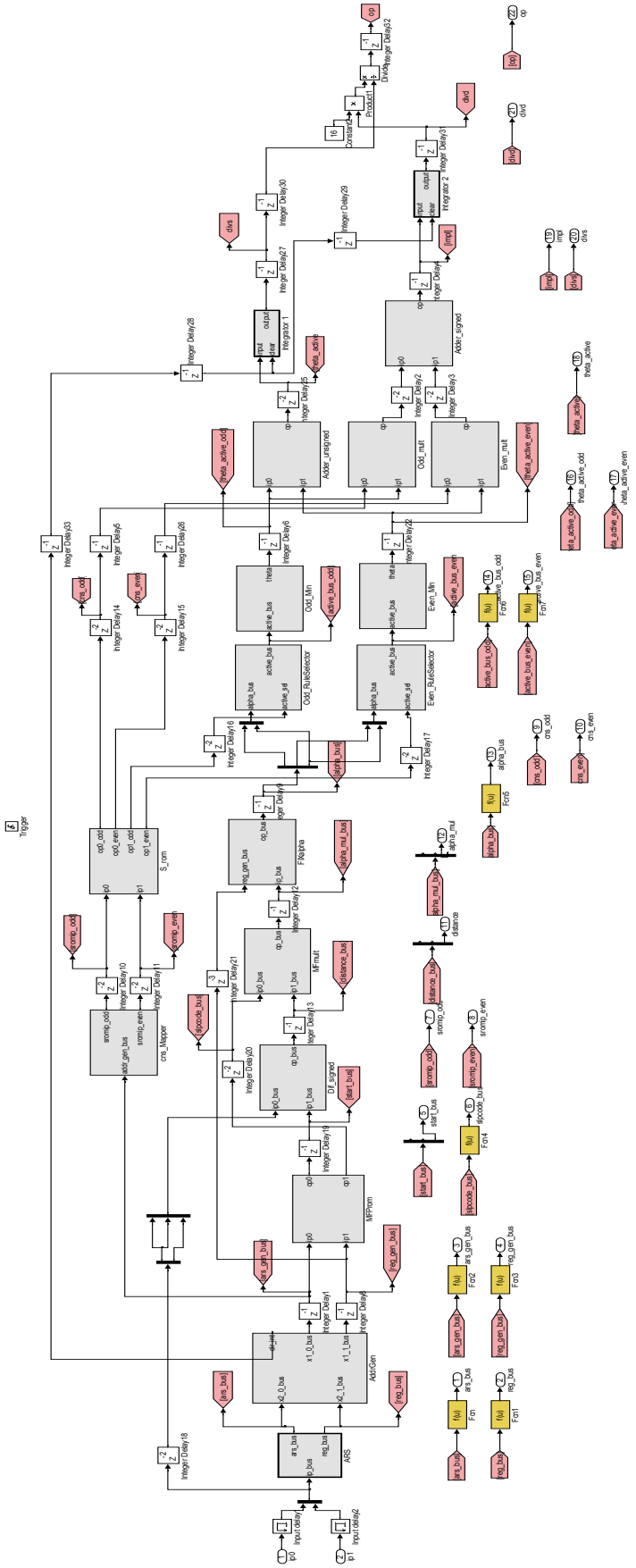
Τελευταίο βήμα είναι η παραγωγή του bitstream αρχείου (χρησιμοποιήθηκε η εφαρμογή iMPACT του περιβάλλοντος ISE Foundation της εταιρίας Xilinx) το οποίο θα φορτωθεί στο ολοκληρωμένο κύκλωμα FPGA. Φυσικά, μετά τη φόρτωση του αρχείου στο FPGA και τη δημιουργία του πραγματικού κυκλώματος μέσα σε αυτό, είναι πλέον δυνατός ο έλεγχος αυτού στον πάγκο για τυχόν λάθη (bench testing) αλλά και κάτω από τις πραγματικές συνθήκες εργασίας.

Επιπλέον εργαλεία όπως το ChipScope της *Xilinx* επιτρέπουν την παρακολούθηση σε πραγματικό χρόνο (real-time) της εσωτερικής λειτουργίας του κυκλώματος που έχει υλοποιηθεί μέσα στο FPGA, συνήθως μέσω της θύρας JTAG. Συνεπώς διευκολύνοντας σημαντικά την εύρεση και αντιμετώπιση πιθανών σφαλμάτων, που δεν εντοπίστηκαν στο στάδιο της RTL προσομοίωσης ή προσομοίωσης χρονισμού.

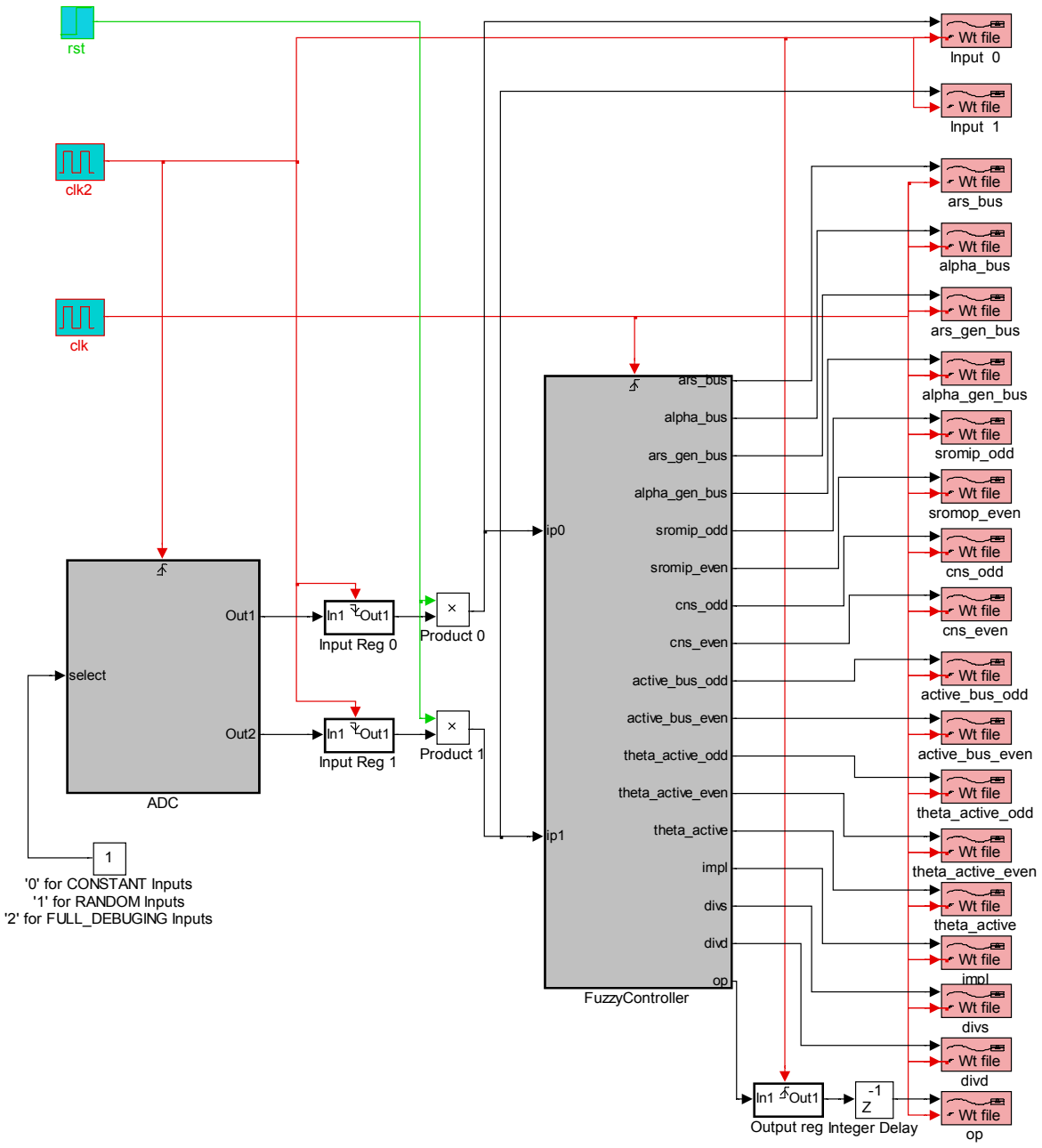
* Συναντάτε και ως *Back annotated simulation, Back annotation*



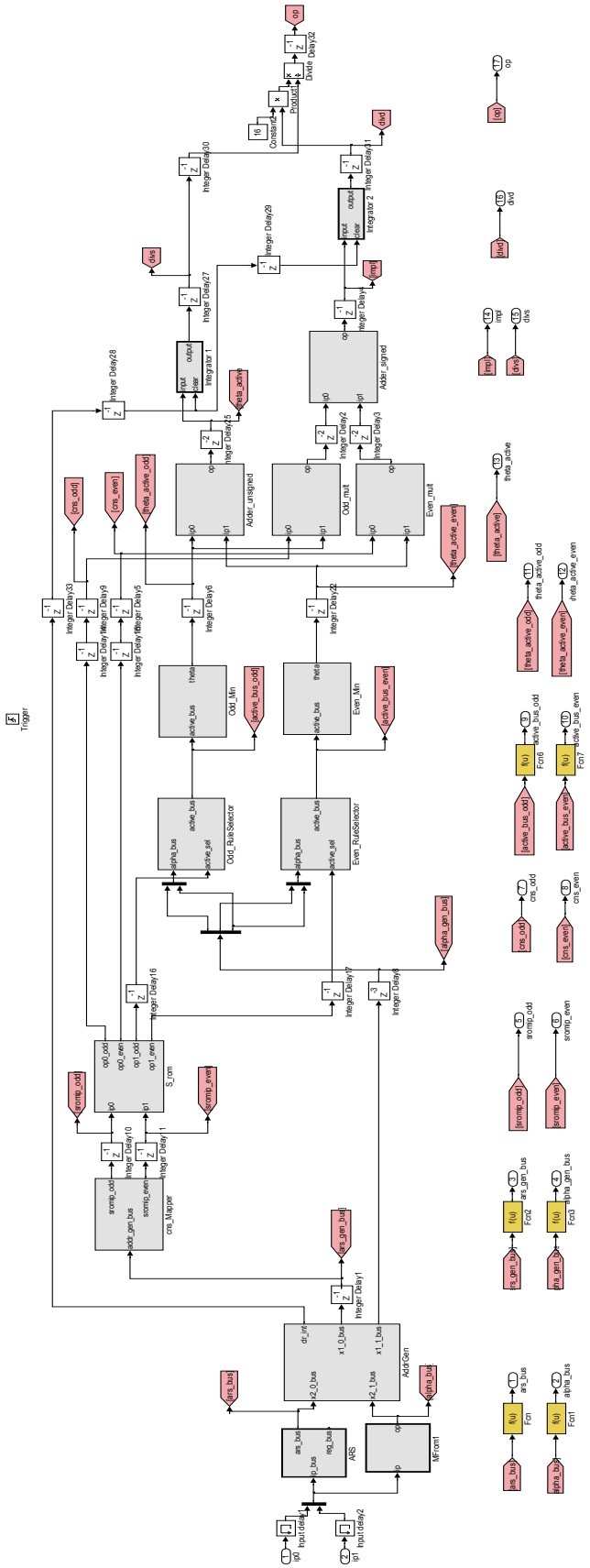
Σχήμα 7.18: Μοντέλο (Simulink) Ασαφούς Ελεγκτή με Αριθμητική Συνάρτηση Συμμετοχής για την παραγωγή Διανυσμάτων Δοκιμής (*testvectors*)



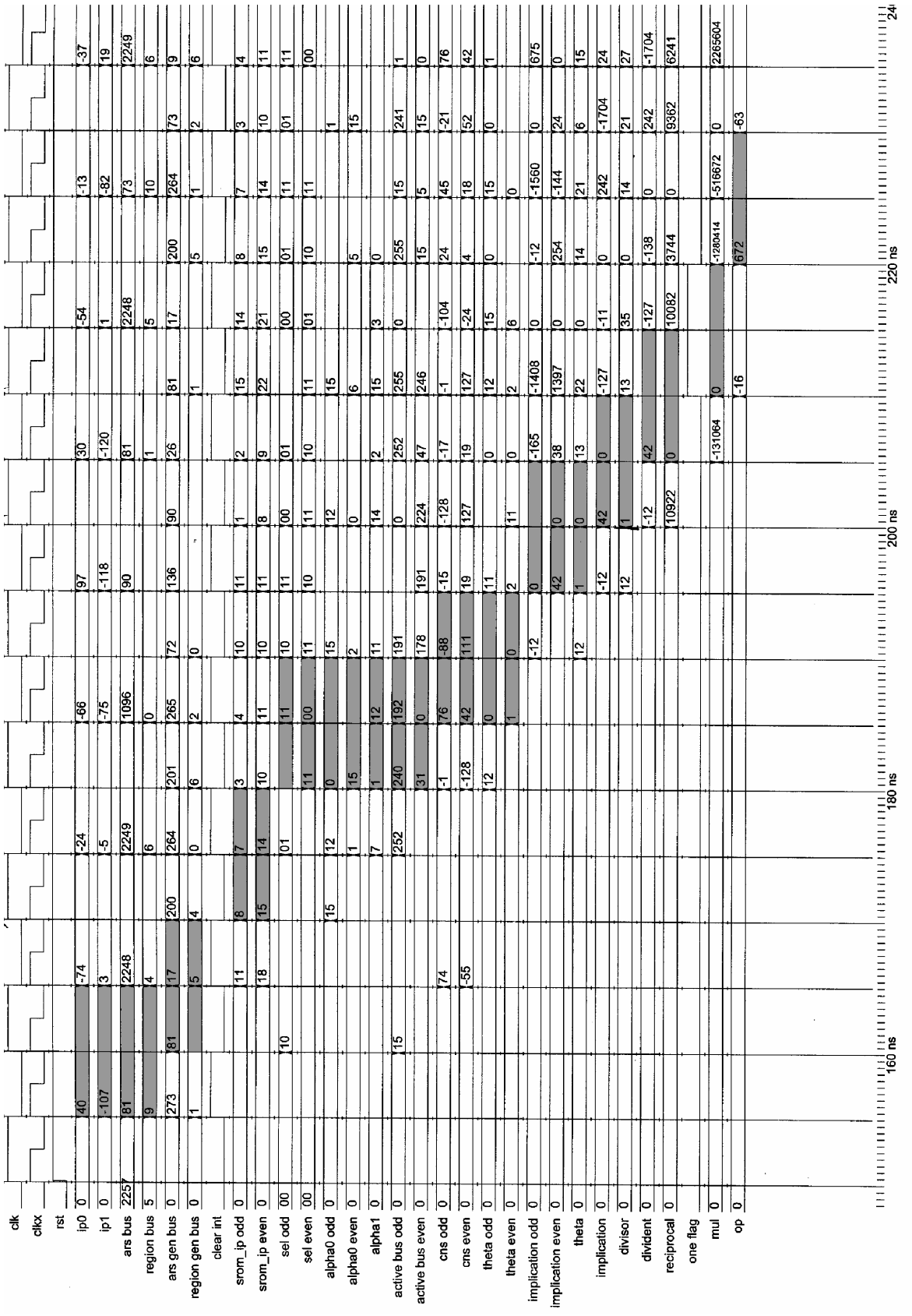
Σχήμα 7.19: Μοντέλο (Simulink) Ασαφούς Ελεγκτή με Αριθμητική Συνάρτηση Συμμετοχής



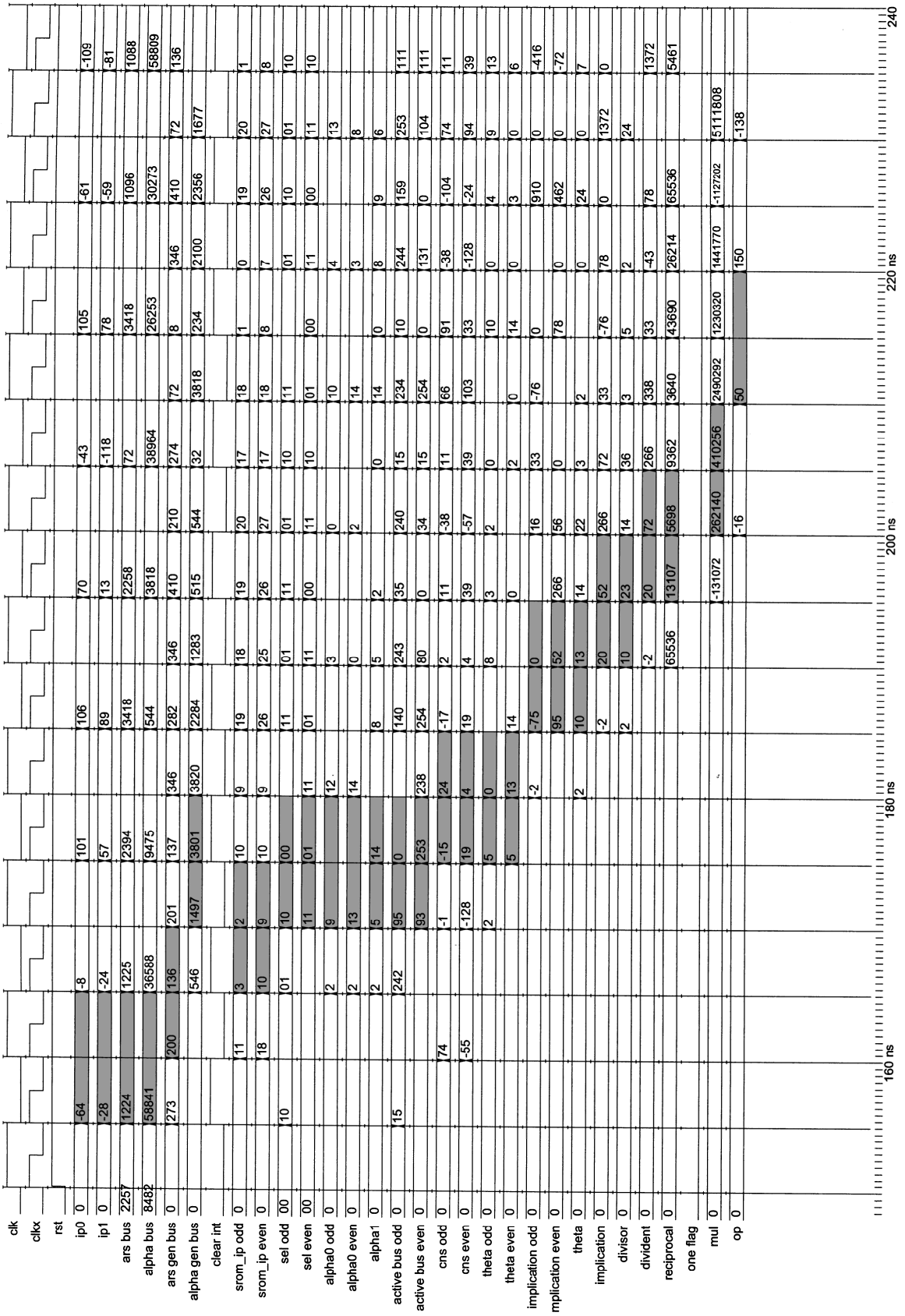
Σχήμα 7.20: Μοντέλο (Simulink) Ασαφούς Ελεγκτή με Συνάρτηση Συμμετοχής βασισμένη σε ROM για την παραγωγή Διανυσμάτων Δοκιμής (*testvectors*)



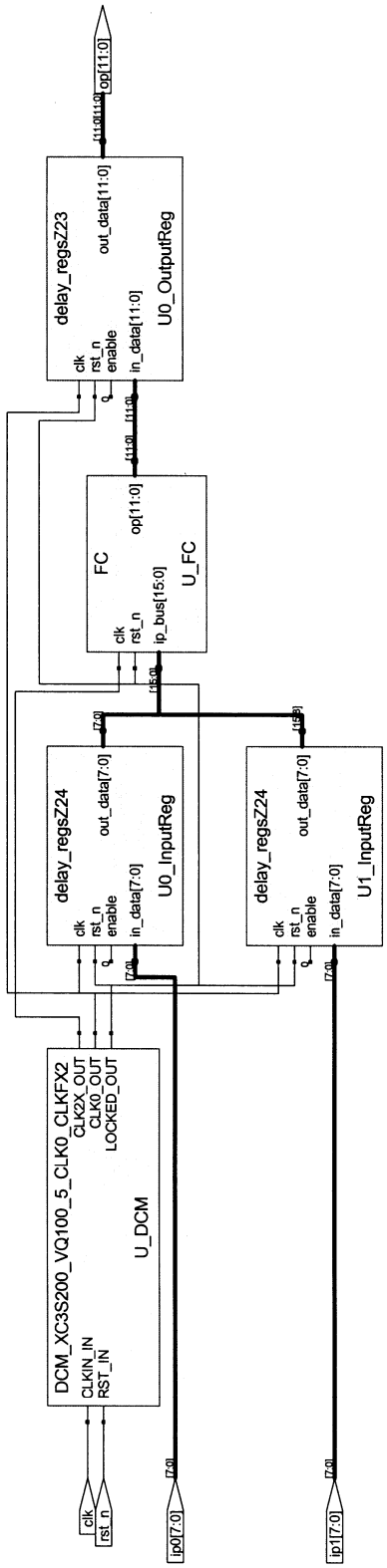
Σχήμα 7.21: Μοντέλο (Simulink) Ασαφούς Ελεγκτή με Συνάρτηση Συμμετοχής βασισμένη σε Πίνακα Αναζήτησης (LUT)



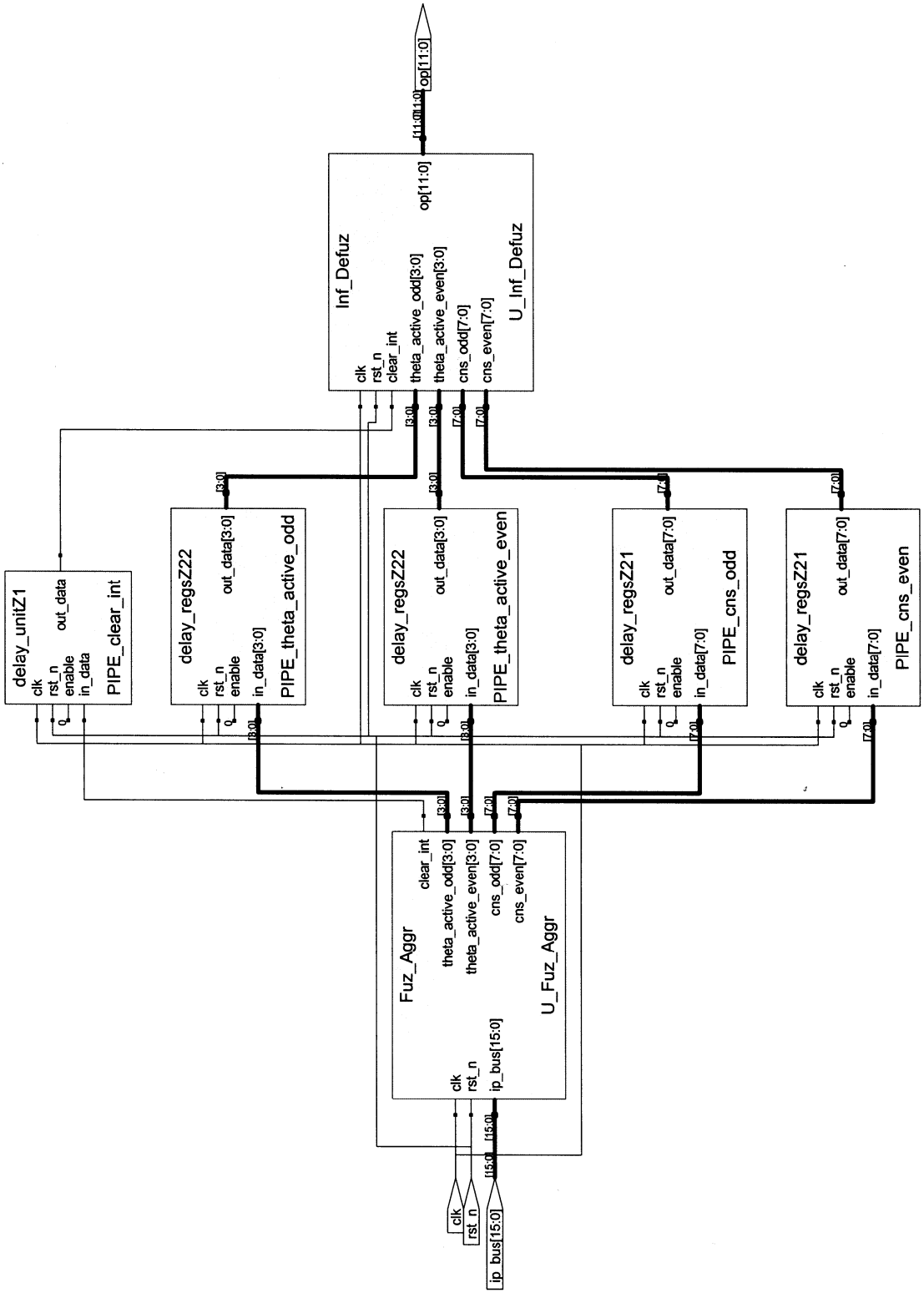
Σχήμα 7.22: RTL προσομοίωση του μοντέλου με Αριθμητική Συνάρτηση Συμμετοχής



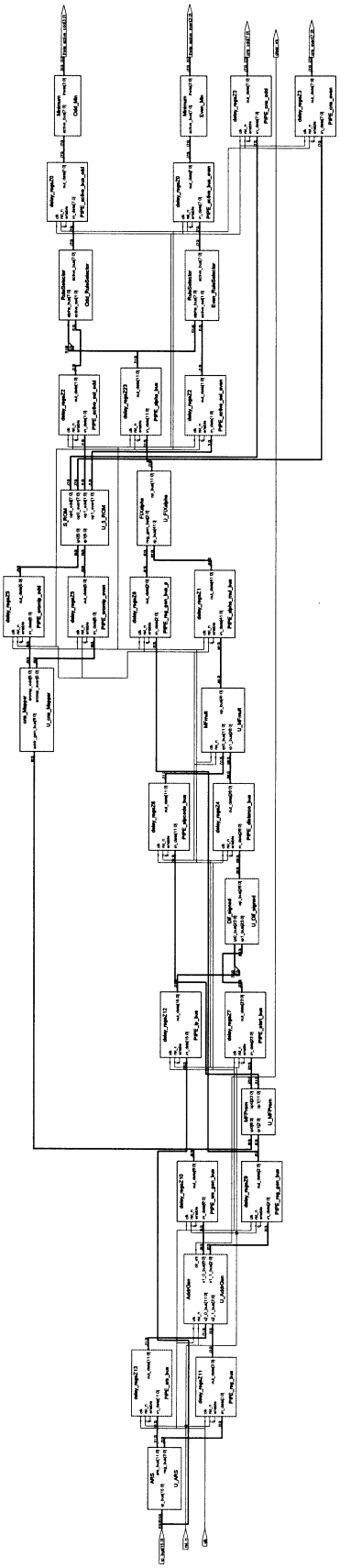
Σχήμα 7.23: RTL προσομοίωση του μοντέλου με Συνάρτηση Συμμετοχής βασισμένη σε Πίνακα Αναζήτησης (LUT)



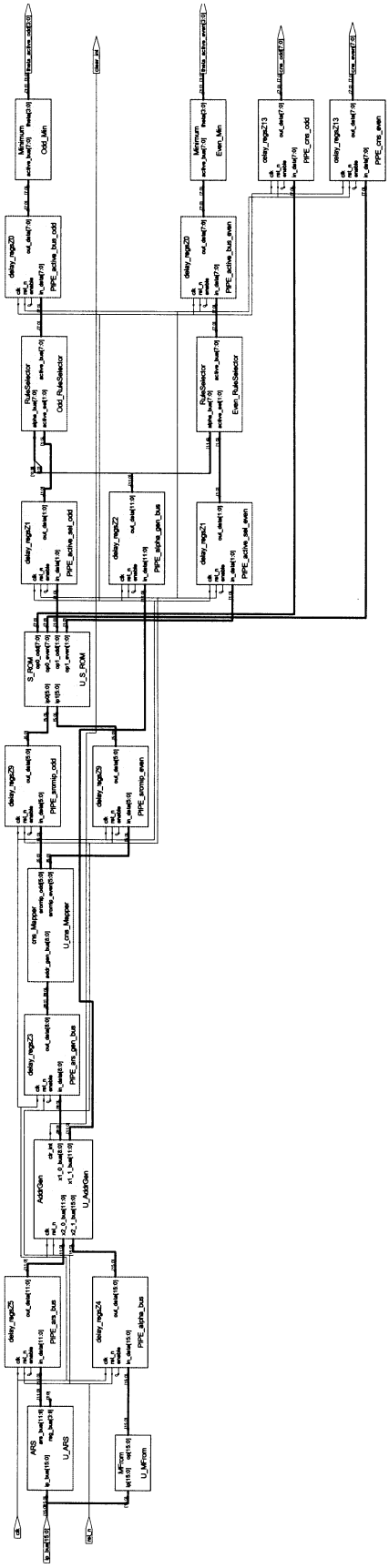
Σχήμα 7.24: RTL όψη του DFLC



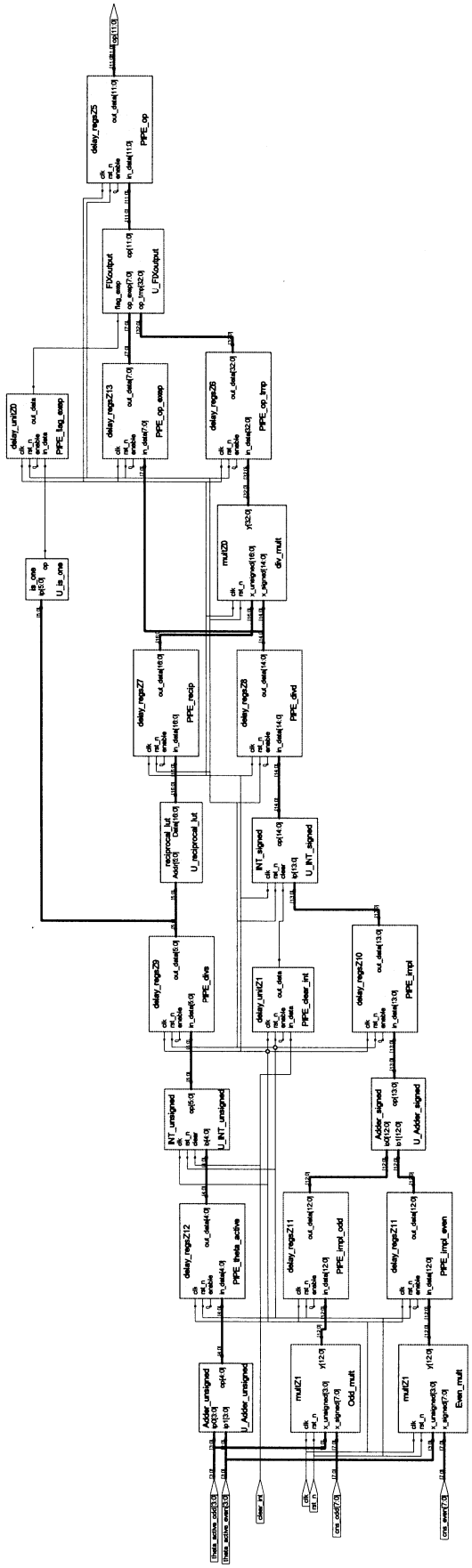
Σχήμα 7.25: RTL όψη του πυρήνα (*core*) του DFLC



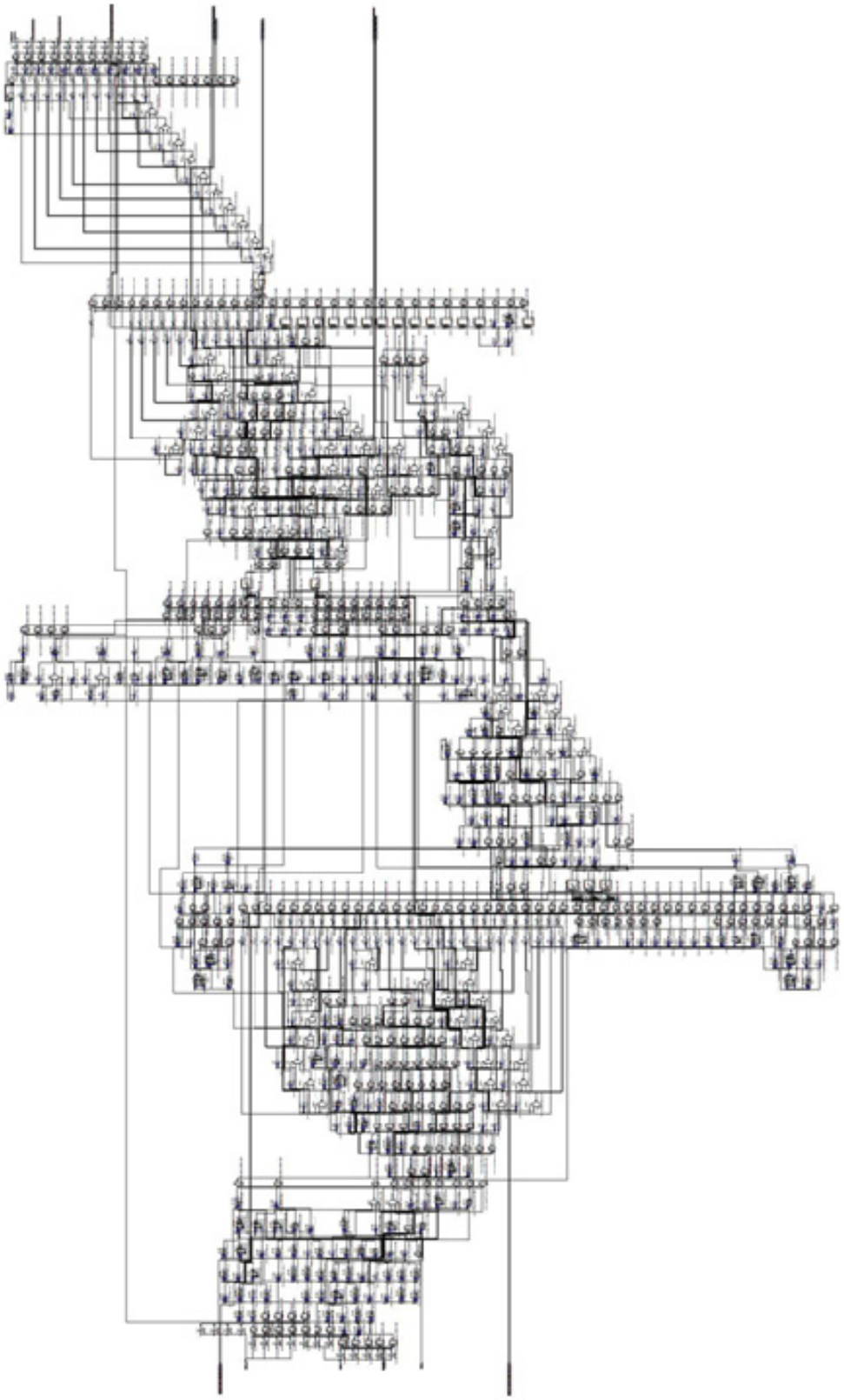
Σχήμα 7.26: RTL όψη του Fuzzification & Implication μέρους με Αριθμητική Συνάρτηση Συμμετοχής



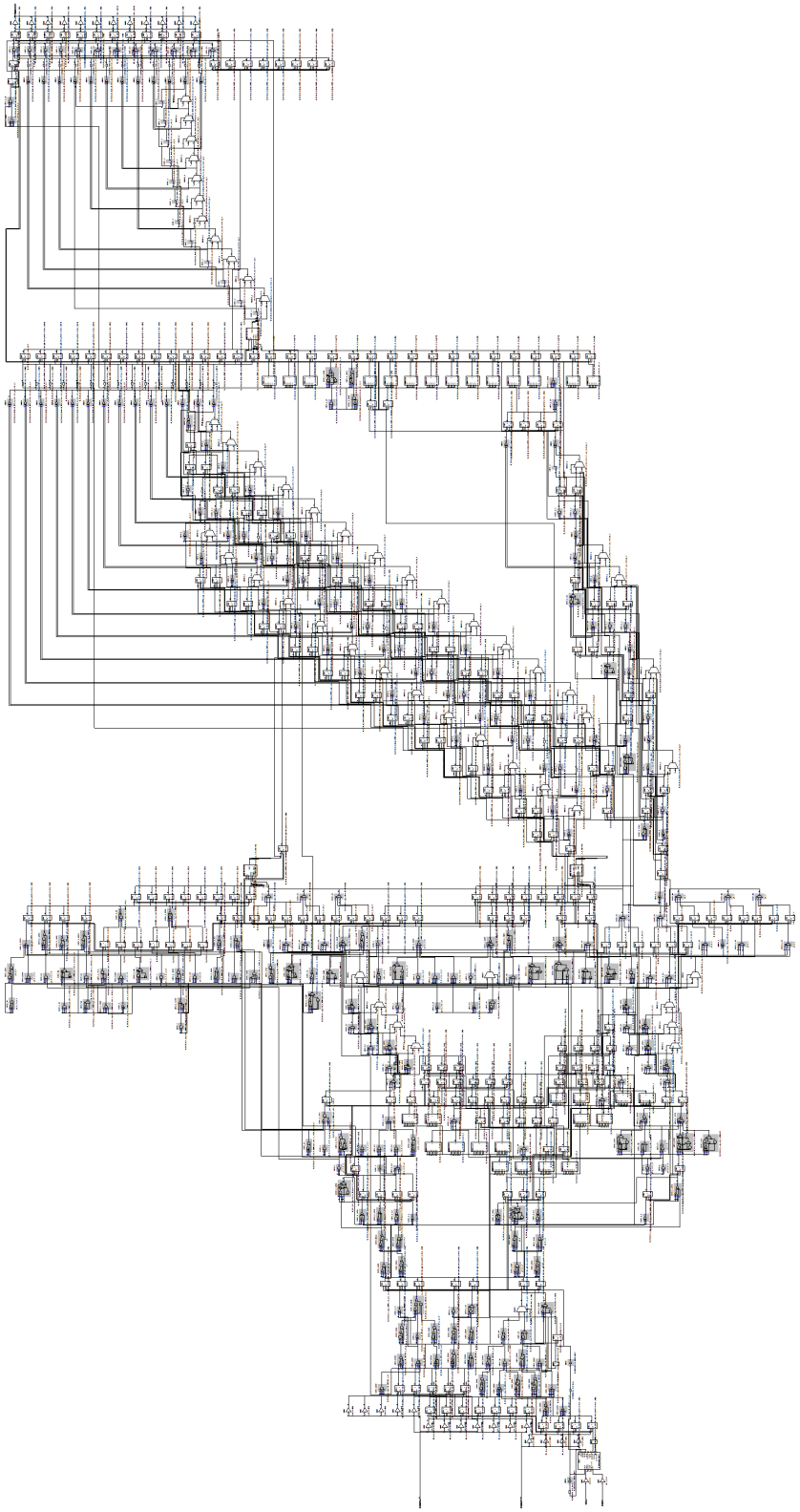
Σχήμα 7.27: RTL όψη του Fuzzification & Implication μέρους με Συνάρτηση Συμμετοχής βασισμένη σε Πίνακα Αναζήτησης (LUT)



Σχήμα 7.28: RTL όψη του Aggregation & Defuzzification μέρους



Σχήμα 7.29: Όψη πυλών του μοντέλου με Αριθμητική Συνάρτηση Συμμετοχής



Σχήμα 7.30: Όψη πυλών του μοντέλου με Συνάρτηση Συμμετοχής βασισμένη σε Πίνακα Αναζήτησης (LUT)

Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops:	344	26,624	1%
Number of 4 input LUTs:	406	26,624	1%
Logic Distribution:			
Number of occupied Slices:	294	13,312	2%
Number of Slices containing only related logic:	294	294	100%
Number of Slices containing unrelated logic:	0	294	0%
Total Number 4 input LUTs:	419	26,624	1%
Number used as logic:	406		
Number used as a route-thru:	13		
Number of bonded IOBs:	30	487	6%
Number of MULT18X18s:	6	32	18%
Number of GCLKs:	2	8	25%
Number of DCMs:	1	4	25%
Constraint(s)	Requested	Actual	Logic Levels
* TS_U_DCM_clk2x_buf = PERIOD TIMEGRP "U_DCM clk2x_buf" 5 ns HIGH 50%	5.000ns	5.078ns	6

Πίνακας 7.5: Χρησιμοποιούμενοι πόροι (device utilization) και περιορισμοί (constraints) για το μοντέλο με Αριθμητική Συνάρτηση Συμμετοχής

Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops:	238	26,624	1%
Number of 4 input LUTs:	419	26,624	1%
Logic Distribution:			
Number of occupied Slices:	368	13,312	2%
Number of Slices containing only related logic:	368	368	100%
Number of Slices containing unrelated logic:	0	368	0%
Total Number 4 input LUTs:	560	26,624	2%
Number used as logic:	419		
Number used as a route-thru:	13		
Number used as 16X1 ROMs:	128		
Number of bonded IOBs:	30	487	6%
Number of MULT 18X18s:	3	32	9%
Number of GCLKs:	2	8	25%
Number of DCMs:	1	4	25%
Constraint(s)	**Requested**	**Actual**	**Logic Levels**
* TS_U_DCM_clk2x_buf = PERIOD TIMEGRP "U_DCM clk2x_buf" 5 ns HIGH 50%	5.000ns	5.103ns	7

Πίνακας 7.6: Αντίστοιχα με τον Πίνακα 6.5 για το μοντέλο με Συνάρτηση Συμμετοχής βασισμένη σε ROM

7.5 Συμπεράσματα

Στο παρόν κεφάλαιο παρουσιάστηκε μία πιο αποδοτική σχεδίαση (σε σχέση με τη σχεδίαση του κεφαλαίου 6) ενός ασαφούς ελεγκτή τύπου Takagi-Sugeno μηδενικής τάξης σε VHDL και η υλοποίηση αυτού σε FPGA ολοκληρωμένο κύκλωμα. Ιδιαίτερο βάρος δόθηκε στο σχεδιασμό του ελεγκτή ώστε να είναι παραμετρικός και να επεξεργάζεται δεδομένα με μεγάλη συχνότητα.

Η παραμετρικότητα επιτεύχθηκε με το χωρισμό του ελεγκτή σε ανεξάρτητα μπλοκ με διακριτές λειτουργίες και την περιγραφή καθενός από αυτά σε γλώσσα υλικού VHDL, η οποία παρέχει τη δυνατότητα παραμετροποίησης του κώδικα μέσω των εντολών *Generic* και *Generate*.

Για τη μείωση των κύκλων υπολογισμού υποθέσαμε επικάλυψη το πολύ δύο μεταξύ των γειτονικών ασαφών συνόλων σε όλο το πεδίο ορισμού κάθε εισόδου. Αυτό είχε σαν συνέπεια να αντιστοιχούν δύο το πολύ *ενεργά ασαφή σύνολα* (με μη μηδενική συνάρτηση συμμετοχής) σε κάθε είσοδο. Συνεπώς, αντί για τον έλεγχο όλων των κανόνων αρκούσε κάθε φορά να ελεγχθούν μόνο οι ενεργοί, αυτοί δηλαδή που ενεργοποιούνταν από τους συνδυασμούς των ενεργών ασαφών συνόλων των εισόδων. Η ιδέα της εξέτασης μόνο των ενεργών κανόνων δεν είναι καινούρια και έχει ήδη εφαρμοστεί στην κατασκευή ασαφών ελεγκτών [64]. Αυτού του είδους οι ελεγκτές, υποθέτοντας επικάλυψη δύο μεταξύ των γειτονικών ασαφών συνόλων των εισόδων, απαιτούν την εξέταση 2^n ενεργών κανόνων, όπου n ο αριθμός των εισόδων του ελεγκτή. Επειδή ο έλεγχος κάθε ενεργού κανόνα πραγματοποιείται σε ένα κύκλο ρολογιού, η συχνότητα δειγματοληψίας προκύπτει $1/2^n$.

Στο παρόν κεφάλαιο, περιγράφεται μια πιο αποτελεσματική μέθοδος η οποία επιτρέπει συχνότητα δειγματοληψίας $2/2^n$, με τον έλεγχο δύο ενεργών κανόνων, αντί για ένα, σε κάθε κύκλο ρολογιού. Η μέθοδος αυτή ονομάστηκε «ODD-EVEN» επειδή στηρίζεται στον ταυτόχρονο έλεγχο των κανόνων που αναφέρονται στο περιττό (ODD) ενεργό ασαφές σύνολο της πρώτης εισόδου με αυτούς που αναφέρονται στο άρτιο (EVEN) ενεργό ασαφές σύνολο της πρώτης εισόδου.

Η εξέταση δύο κανόνων ανά κύκλο επιτεύχθηκε διαχωρίζοντας τη μνήμη των κανόνων (SROM) και τη μνήμη των παραμέτρων των τραπεζοειδών ασαφών συνόλων της πρώτης εισόδου *(parameter Memory Bank)* σε ODD και EVEN μέρη. Ο διαχωρισμός των μνημών είναι θεμιτός γιατί εκτός από το ότι αυξάνει την παραλληλία, μειώνει και το χρόνο αναζήτησης της κάθε μνήμης. Το πρόβλημα της αναζήτησης παρουσιάζεται κυρίως στη μνήμη των κανόνων (SROM) η οποία συνήθως είναι μεγάλη, ιδιαίτερα σε ελεγκτές με πολλές εισόδους. Η αύξηση λοιπόν της παραλληλίας του ελεγκτή έγινε με το ελάχιστο κόστος σε λογική που συνεπάγεται η τοποθέτηση ενός επιπλέον TMFG, ενός ARS, ενός MIN, ενός Πολλαπλασιαστή και δύο Αθροιστών.

Αξίζει να σημειωθεί ότι η αύξηση της λογικής λόγω του ODD-EVEN μοντέλου οφείλεται αποκλειστικά και μόνο στην τοποθέτηση των προαναφερθέντων μπλοκ ανεξάρτητα από τον αριθμό των εισόδων του ασαφούς ελεγκτή. Επιπλέον, η επεξεργασία κάθε κανόνα γίνεται ανεξάρτητα, και μόνο στους δύο (προσημασμένο και μη προσημασμένο) Αθροιστές *(Signed and Unsigned Adders)* γίνεται συνδυασμός των αποτελεσμάτων τους. Άρα, η εισαγωγή του ODD-EVEN μοντέλου προκαλεί μια αύξηση του συνολικού *delay* μόνο κατά μία βαθμίδα αγωγού εξαιτίας της παρεμβολής των Αθροιστών πριν από τους

Ολοκληρωτές. Το συμπέρασμα, συνεπώς, που προκύπτει από την εξέταση του ODD-EVEN μοντέλου είναι ότι έχει να επιδείξει περισσότερα προτερήματα από μειονεκτήματα και για το λόγο αυτό προτιμήθηκε από την απλή υλοποίηση.

Ως προς το χειρισμό των συναρτήσεων συμμετοχής, παρουσιάστηκαν δύο εναλλακτικές υλοποιήσεις. Μία με αριθμητικό υπολογισμό των συναρτήσεων συμμετοχής (με χρήση του TMFG) και μία άλλη στην οποία οι τιμές αυτών ήταν προϋπολογισμένες και αποθηκεύτηκαν σε μία ROM. Η πρώτη αρχιτεκτονική, όπως είναι φυσικό, καταλαμβάνει λιγότερο χώρο στο ολοκληρωμένο κύκλωμα απ' ότι η δεύτερη αλλά χρησιμοποιεί περισσότερους πολλαπλασιαστές. Το βασικό μειονέκτημα της πρώτης είναι ότι επιτρέπει μόνο τραπεζοειδείς συναρτήσεις συμμετοχής. Αντίθετα, στη δεύτερη δεν υπάρχει κανένας περιορισμός στο είδος των συναρτήσεων συμμετοχής. Αξίζει να σημειωθεί ότι τα σημεία διαφοροποίησης των δύο αρχιτεκτονικών είναι ελάχιστα και στο μεγαλύτερο μέρος χρησιμοποιούν κοινά μπλοκ. Αυτό αποτέλεσε και μια επιδίωξη της σχεδίασης, ώστε να είναι δυνατόν να γίνει η μετάβαση από τη μία αρχιτεκτονική στην άλλη με εύκολο τρόπο.

Πρωτοτυπία υπήρξε και στην υλοποίηση της διαίρεσης που επιβάλει ο τύπος της από-ασαφοποίησης. Η διαίρεση είναι μία από τις πιο δύσκολες και απαιτητικές αριθμητικές πράξεις όσον αφορά την κυκλωματική της υλοποίηση [71]. Πολλές μέθοδοι έχουν προταθεί. Εκείνες οι οποίες αποσκοπούν στη βελτιστοποίηση του χρόνου, ξεκινούν από μια αρχική εκτίμηση του $reciprocal = 1/divisor$ και έπειτα ακολουθούν μια επαναληπτική αριθμητική μέθοδο, όπου ο αριθμός των επαναλήψεων εξαρτάται από τη διαφορά της ακρίβειας του *reciprocal* από την επιθυμητή ακρίβεια της διαίρεσης [65]. Υπάρχει και μια άλλη μέθοδος η οποία χρησιμοποιεί το εσωτερικό ενός πολλαπλασιαστή για να πραγματοποιήσει την πράξη της διαίρεσης [72].

Η τεχνική που ακολουθήθηκε εδώ είναι η πρώτη, με τη διαφορά ότι η ακρίβεια του *reciprocal* υπολογίστηκε να είναι τόση ώστε να επιτευχθεί η επιθυμητή ακρίβεια της διαίρεσης χωρίς να χρειαστεί επανάληψη. Για την εύρεση της ελάχιστης ακρίβειας του reciprocal παρατηρήθηκε ότι ο τύπος της απο-ασαφοποίησης *Weighted Average* παρουσιάζει μια *"κανονικότητα"*. Με άλλα λόγια ποτέ δεν θα χρειαστεί να διαιρέσουμε την ελάχιστη τιμή του διαιρέτη με τη μέγιστη τιμή που μπορεί να πάρει ο διαιρετέος, πράγμα που θα απαιτούσε επιπλέον ακρίβεια στον υπολογισμό του *reciprocal*. Έτσι, η ακρίβεια υπολογίστηκε για όλες τις δυνατές τιμές των παραμέτρων του τύπου απο-ασαφοποίησης, με τη βοήθεια του MATLAB - Simulink. Τελικά, καταφέραμε να απλοποιήσουμε την πράξη της διαίρεσης σε ένα πολλαπλασιασμό μεταξύ του αντιστρόφου του διαιρέτη *(reciprocal)* και του διαιρετέου *(dividend)*.

Εξαντλητική εξέταση των δύο αρχιτεκτονικών του ασαφούς ελεγκτή έγινε για την περίπτωση δύο εισόδων, με την υλοποίηση ενός ασαφούς ελεγκτή με τα χαρακτηριστικά του Πίνακα 7.1. Η σύνθεση του DFLC έγινε με τη βοήθεια του εργαλείου Synplify Pro, και η τοποθέτηση και δρομολόγηση με το πακέτο εργαλείων του περιβάλλοντος ISE Foundation σε ένα FPGA ολοκληρωμένο κύκλωμα (Spartan-3 XC3S1500-4 FG676) της *Xilinx*.

Η πρώτη αρχιτεκτονική είχε σαν αποτέλεσμα έναν ασαφή ελεγκτή με εσωτερική συχνότητα λειτουργίας στα 200MHz, εξωτερική συχνότητα δειγματοληψίας 100MHz, και delay 65ns, ο οποίος κατέλαβε συνολικά 294 slices (2% του FPGA ολοκληρωμένου κυκλώματος). Η δεύτερη αρχιτεκτονική αντίστοιχα, είχε σαν αποτέλεσμα έναν ασαφή

ελεγκτή με εσωτερική συχνότητα λειτουργίας στα 200MHz, εξωτερική συχνότητα δειγματοληψίας 100MHz, και delay 55ns, ο οποίος κατέλαβε συνολικά 368 slices (2% του FPGA). Οι δύο αρχιτεκτονικές ελέγχθηκε ότι λειτουργούν για την περίπτωση $3^{ων}$ και $4^{ων}$ εισόδων καθώς και για ανάλυση συνάρτησης συμμετοχής 8-bit.

Κεφάλαιο 8

Σύστημα σε Ψηφίδα (SoC) για το Πρόβλημα της Παρακολούθησης Πορείας σε Κινητά Ρομπότ

8.1 Εισαγωγή

Στο κεφάλαιο αυτό θα μελετήσουμε ένα σύστημα σε ψηφίδα (*System on Chip* – SoC) για το πρόβλημα της παρακολούθησης πορείας σε αυτόνομα μη-ολονομικά κινητά ρομπότ. Το SoC που θα παρουσιάσουμε περιλαμβάνει έναν ψηφιακό ασαφή ελεγκτή [61] (παρουσιάστηκε στο Κεφάλαιο 6), και ένα πρόγραμμα ελέγχου της ροής των διαδικασιών που εκτελείται μέσα στον μαλακό πυρήνα μικροεπεξεργαστή Microblaze της εταιρίας Xilinx. Ο ασαφής ελεγκτής υλοποιεί έναν ασαφή αλγόριθμο παρακολούθησης πορείας (*path tracking*) [73]. Ολόκληρο το σύστημα προσαρμόστηκε πάνω σε ένα ρομπότ Pioneer P3-DX8 και στη συνέχεια εκτελέστηκαν διάφορα πειράματα, ούτως ώστε να γίνει αποτίμηση της γενικής απόδοσης του συστήματος. Τέλος, αναλύονται διάφορα προβλήματα κβάντισης και περιορισμοί που προκύπτουν λόγω της δεδομένης σύνθεσης του συστήματος.

Ανάλογες ρομποτικές εφαρμογές με τη χρήση FPGA ολοκληρωμένων κυκλωμάτων σημειώνονται στη βιβλιογραφία από διάφορους ερευνητές [74][75][76][77], δεδομένου ότι η χρήση τους προσφέρει σημαντικά πλεονεκτήματα σε σχέση με συστήματα βασισμένα σε μικροεπεξεργαστή από τη μία πλευρά ή υλοποιήσεις σε ASIC ολοκληρωμένα κυκλώματα από την άλλη. Μερικά πλεονεκτήματα των FPGA σε σχέση με τα ASIC, είναι ότι προσφέρουν ταχύτερο κύκλο σχεδίασης, επαναπρογραμματισμό και μικρότερο απαιτούμενο χρόνο από τη σύλληψη έως τη διάθεση του προϊόντος στην αγορά (*Time-to-Market* – TTM) διότι δεν δαπανάται κατασκευαστικός χρόνος (παραγωγή σχεδίου, μάσκες και άλλα κατασκευαστικά βήματα). Επίσης τα FPGA σε σχέση με τους μικροεπεξεργαστές, επιτυγχάνουν μεγαλύτερο βαθμό παραλληλίας. Τα σημερινά FPGA

περιέχουν εκατοντάδες ισχυρά τμήματα ειδικά για επεξεργασία σήματος (*Digital Signal Processing* – DSP) και επιτυγχάνουν ταχύτητες μέχρι και 500 MHz ξεπερνώντας σε επεξεργαστική ισχύ αρκετούς DSP και RISC επεξεργαστές. Η δυνατότητα αυξημένης παραλληλίας ισοδυναμεί με μεγαλύτερη επεξεργαστική ισχύ σε σχέση με τους DSP επεξεργαστές καταρρίπτοντας το παράδειγμα της ακολουθιακής εκτέλεσης και επιτυγχάνοντας περισσότερη επεξεργασία ανά κύκλο ρολογιού, Στη μελέτη των Leong και Tsoi [75] γίνεται μια εκτενής αναφορά στην εφαρμογή FPGA σε ρομποτικά συστήματα. Μία αξιοσημείωτη μελέτη που αναφέρεται στη βιβλιογραφία [76] είναι η χρήση των FPGA στα διαστημόπλοια προσεδάφισης, Mars Pathfinder, Mars Surveyor ’98, and Mars Surveyor ’01.

Η ανάγκη ενός συστήματος υψηλής υπολογιστικής ικανότητας προκύπτει από το γεγονός ότι ο έλεγχος παρακολούθησης πορείας είναι ιδιαίτερα απαιτητικός όσον αφορά τον αριθμό των αναγκαίων υπολογισμών. Με τη χρήση SoC σε FPGA, εκμεταλλευόμαστε τη δυνατότητα επαναπροσδιορισμού (*re-configurability*) του υλικού/λογισμικού που μας παρέχει το FPGA για να καλύψουμε τις ανάγκες της πλατφόρμας παρακολούθησης πορείας με τη χρήση ασαφούς λογικής για αυτόνομα κινητά ρομπότ με αυξημένη δυνατότητα επεξεργασίας και ευέλικτο υλικό για διαφορετικές διεργασίες.

Οι software υλοποιήσεις ασαφών ελεγκτών αδυνατούν να πετύχουν μεγάλη ταχύτητα επεξεργασίας κατά πρώτον λόγω της ακολουθιακής εκτέλεσης των εντολών και κατά δεύτερον λόγω ότι οι τυπικοί επεξεργαστές δεν υποστηρίζουν άμεσα ασαφείς τελεστές (π.χ., minimum ή maximum). Στη βιβλιογραφία αναφέρονται κάποιες τροποποιημένες αρχιτεκτονικές βασισμένες σε τυπικούς επεξεργαστές ούτως ώστε να υποστηρίζουν ασαφή επεξεργασία [78][79][80]. Με τη χρήση τέτοιων τροποποιημένων επεξεργαστών επιτυγχάνεται μεγαλύτερη ταχύτητα επεξεργασίας σε σχέση με τους τυπικούς επεξεργαστές που όμως σε ορισμένες περιπτώσεις εφαρμογών πραγματικού χρόνου δεν είναι αρκετή. Σε τέτοιες περιπτώσεις η αποκλειστική υλοποίηση σε υλικό (*dedicated hardware implementation*) είναι αναγκαία [81].

Όπως δείξαμε στο Κεφάλαιο 6, ο DFLC έχει τη δυνατότητα παραμετροποίησης για διαφορετικό αριθμό εισόδων και εξόδων, αριθμό τριγωνικών ή τραπεζοειδών ασαφών συνόλων ανα είσοδο, μέθοδο για τις προϋποθέσεις των κανόνων (t-norm , s-norm), τύπο διαιρέτη και επιπλέον αριθμό σταδίων αγωγού των διαφόρων λειτουργικών μονάδων (*modules* ή *functional blocks*) του κυκλώματος. Η αρχιτεκτονική του ελεγκτή προϋποθέτει επικάλυψη μέχρι 2 ασαφών συνόλων μεταξύ των παρακείμενων συνόλων και απαιτεί 2^n (όπου n αριθμός των εισόδων του ελεγκτή) κύκλους ρολογιού στη συχνότητα που δουλεύει ο πυρήνας για κάθε νέο σύνολο δεδομένων εισόδου (ο ρυθμός επεξεργασίας δεδομένων είναι 56.34ns), δεδομένου ότι επεξεργάζεται έναν ενεργό κανόνα ανά κύκλο ρολογιού. Στην παρούσα εφαρμογή το SoC (που εμπεριέχει και το DFLC) επιτυγχάνει μία συχνότητα λειτουργίας της τάξης των 71 MHz.

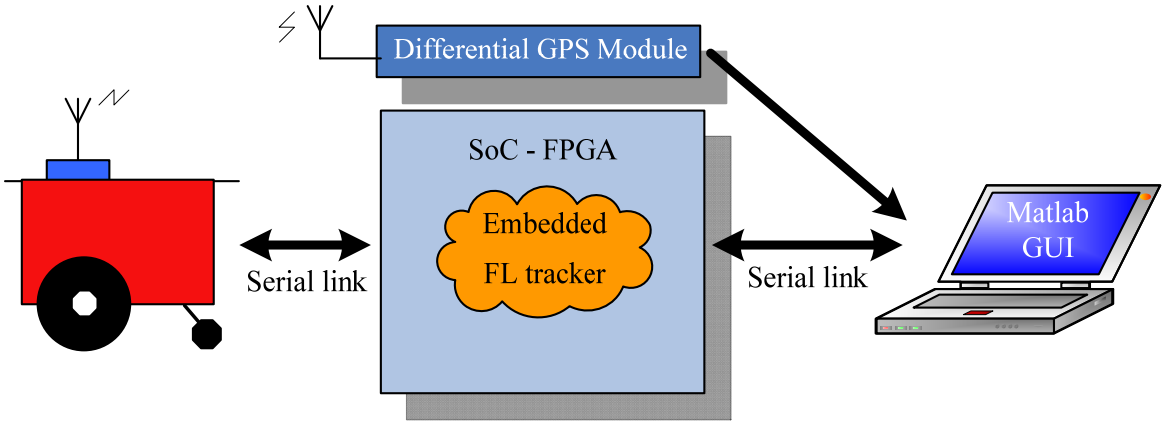
Για την επίτευξη αυτού του χρονισμού η καθυστέρηση (latency) του DFLC αποτελείται από 9 βαθμίδες αγωγού (*pipeline stages*) σταθερής καθυστέρησης όπου η καθεμία απαιτεί 14.085 ns. Ο DFLC όπως παρουσιάστηκε στο Κεφάλαιο 6, είναι βασισμένος σε έναν αλγόριθμο παρόμοιο με τον Takagi-Sugeno μηδενικού τύπου συμπεράσματος και της μεθόδου από-ασαφοποίησης σταθμισμένης μέσης τιμής (ή μέση τιμή με βάρη) και με βάση τις παραμέτρους που επιλέξαμε για την εφαρμογή της παρακολούθησης πορείας (βλ. Πίνακα 8.3) χρησιμοποιεί δύο εισόδους των 12-bit και μια έξοδο 12-bit, με 9 τραπεζοειδείς

ή τριγωνικές συναρτήσεις συμμετοχής ανά είσοδο με 8-bit ανάλυση για το βαθμό αλήθειας και βάση 81 κανόνων.

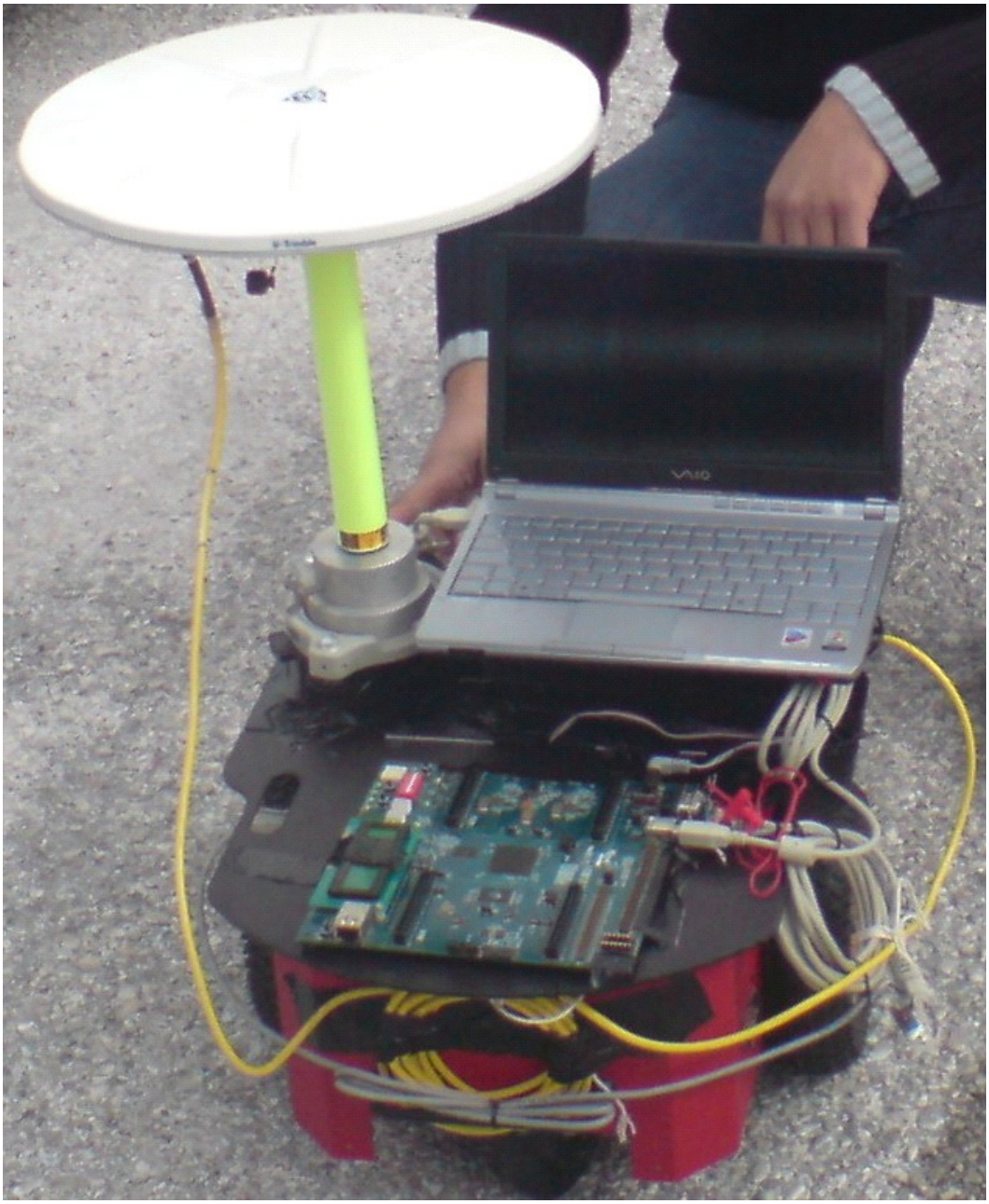
Ο ασαφής αλγόριθμος παρακολούθησης πορείας που χρησιμοποιείται [73] στην εφαρμογή αυτή έχει υποστεί κάποιες τροποποιήσεις ώστε να είναι δυνατή η προσαρμογή του στο DFLC. Ενώ ο πρωτότυπος αλγόριθμος του ελεγκτή που αναφέρεται στη βιβλιογραφία είναι Mamdani τύπου με Γκαουσιανές συναρτήσεις συμμετοχής, ο ελεγκτής που χρησιμοποιήθηκε εδώ είναι Takagi-Sugeno μηδενικού τύπου με Τριγωνικές συναρτήσεις συμμετοχής. Επιπλέον ο αλγόριθμος της παρακολούθησης πορείας κάνει χρήση της τεχνικής που αναφέρεται ως «spatial window» στη βιβλιογραφία [73] και αναλύεται στις επόμενες παραγράφους.

8.2 Συνοπτική Περίληψη του Συστήματος

Το σύστημα που μελετάμε εδώ αποτελείται από τέσσερεις λειτουργικές μονάδες κατάλληλα διασυνδεμένες μεταξύ τους. Η γενική μορφή της πλατφόρμας της εφαρμογής φαίνεται στο Σχήμα 8.1 ενώ το Σχήμα 8.2 είναι μια φωτογραφία του πραγματικού συστήματος κατά τη διάρκεια πειραμάτων. Στη φωτογραφεία διακρίνονται το Pioneer P3-DX8 ρομπότ, η πλακέτα που περιλαμβάνει το FPGA, ο φορητός υπολογιστής που τρέχει το περιβάλλον της ρομποτικής εφαρμογής καθώς και μέρος του διαφορικού GPS.



Σχήμα 8.1: Γενική μορφή του ρομποτικού συστήματος



Σχήμα 8.2: Το ρομποτικό σύστημα κατά τη διάρκεια πειραμάτων

Σκοπός του FPGA SoC είναι η υλοποίηση της αυτόνομης λογικής ελέγχου του ρομπότ. Λαμβάνει τις πληροφορίες της οδομετρίας από το ρομπότ και στέλνει τις κατευθυντήριες εντολές (*steering commands*) που εξάγονται από τον ασαφή ελεγκτή παρακολούθησης πορείας (*fuzzy logic tracker*). Πέραν της πλοήγησης, το SoC αναλαμβάνει να αποδικοποιήσει τα πακέτα δεδομένων που στέλνει το ρομπότ που περιλαμβάνουν τη θέση του, την κατάσταση των κινητήρων, τις μετρήσεις από τους αισθητήρες υπερήχων, κλπ. και επίσης κωδικοποιεί τις κατευθυντήριες εντολές σε ένα πακέτο δεδομένων που λαμβάνεται από το ρομπότ. Άρα με βάση τα προηγούμενα το SoC υλοποιεί έναν κωδικοποιητή/αποδικοποιητή (*codec*) που χρησιμεύει στην επικοινωνία εισόδου/εξόδου με το ρομπότ. Επίσης στέλνει κρίσιμες πληροφορίες σε μία εφαρμογή παρακολούθησης (*monitor program*) που έχει υλοποιηθεί σε περιβάλλον MATLAB. Το άνω επιπέδου πρόγραμμα (*top-level program*) που

αναλαμβάνει όλες αυτές τις δραστηριότητες έχει υλοποιηθεί σε γλώσσα C και εκτελείται στον μαλακό πυρήνα επεξεργαστή Microblaze. Επιπλέον το ίδιο πρόγραμμα είναι υπεύθυνο για το χειρισμό των απαιτούμενων συγχρονισμών στο σύστημα

Η εφαρμογή MATLAB εμφανίζει τις πληροφορίες όσον αναφορά τη θέση και την ταχύτητα του ρομπότ, όπως αυτή προκύπτει από την οδομετρία του, καθώς και επιπρόσθετα δεδομένα που χρησιμοποιούνται στον έλεγχο παρακολούθησης πορείας (path tracking control). Επιπροσθέτως, υπολογίζει τη θέση του ρομπότ σχετικά με τον κόσμο του (*world*) και τα συστήματα των τοπικών συντεταγμένων (*local coordinate systems*). Μια σημαντική λειτουργία της εφαρμογής MATLAB είναι ότι παρέχει στο ρομπότ μια πορεία που πρέπει να παρακολουθήσει. Αυτό επιτυγχάνεται ζωγραφίζοντας στην εφαρμογή MATLAB την επιθυμητή πορεία και στη συνέχεια στέλνοντας τη στο SoC. Αμέσως μετά την επιτυχή φόρτωση της πορείας, το SoC ξεκινάει τη διαδικασία toy ελέγχου παρακολούθησης (*tracking control*).

Η ρομποτική πλατφόρμα που χρησιμοποιήθηκε για τον έλεγχο του SoC είναι το ρομπότ P3-DX8 της εταιρίας ActivMedia [82]. Το εν λόγω ρομπότ έχει ανάλυση 1 mm για την εκτίμηση της θέσης του (*position estimation*) και ανάλυση 1° γωνία για την πορεία του (*heading*). Το κινηματικό μοντέλο του ρομπότ εξομοιώνεται ως κατευθυνόμενο όχημα περιορισμένης καμπυλότητας (bounded curvature steering vehicle) και όχι ως διαφορικής κίνησης (*differential drive*) που έχει ως αποτέλεσμα έναν περιορισμό στη μέγιστη καμπυλότητα που μπορεί να στρίψει. Ο λόγος του περιορισμού αυτού οφείλεται στο γεγονός ότι αρχικά ο ασαφής αλγόριθμος παρακολούθησης πορείας προοριζόταν για το Dubin's Car μοντέλο [83], το οποίο έχει περιορισμό στην ελάχιστη ακτίνα στρέψης του ρομπότ και μόνο προς την εμπρόσθια κίνηση. Όπως θα εξηγήσουμε αναλυτικά στις επόμενες παραγράφους ο περιορισμός στην καμπυλότητα σε συνάρτηση με την ανάλυση 1° γωνία του P3 ρομπότ εισάγει ένα πρόβλημα κβαντοποίησης στην καμπυλότητα..

Το ρομπότ επικοινωνεί με το FPGA μέσω σειριακού προτοκόλου (RS232) για την αποστολή και λήψη των μορφοποιημένων δεδομένων (*data frames*). Η εταιρία ActivMedia κάνει χρήση του δικού της προτοκόλου εντολών για την επικοινωνία με το ρομπότ που είναι αποθηκευμένο στον μικροελεγκτή του ρομπότ. Τα πακέτα δεδομένων που αποστέλλονται από το ρομπότ αναφέρονται ως SIP (*Server Information Packets*), ενώ τα πακέτα που λαμβάνονται από το ρομπότ ως CP (*Command Packets*) [82][*].

Τα πειράματα που έγιναν έδειξαν ότι ενώ ο FL Tracker δουλεύει ικανοποιητικά, η απόδοση του μειώνεται σημαντικά λόγω των λαθών της οδομετρίας του ρομπότ που αθροίζονται κατά τη διάρκεια του χρόνου που εκτελείται το πείραμα. Διάφορες προσπάθειες βαθμονόμησης που έγιναν με σκοπό τη βελτίωση του εντοπισμού θέσης μέσω της οδομετρίας του ρομπότ αποδείχτηκαν μη αποτελεσματικές.

[*] Αναφέρονται αναλυτικά στο εγχειρίδιο του ρομπότ στις σελ. 33-36.

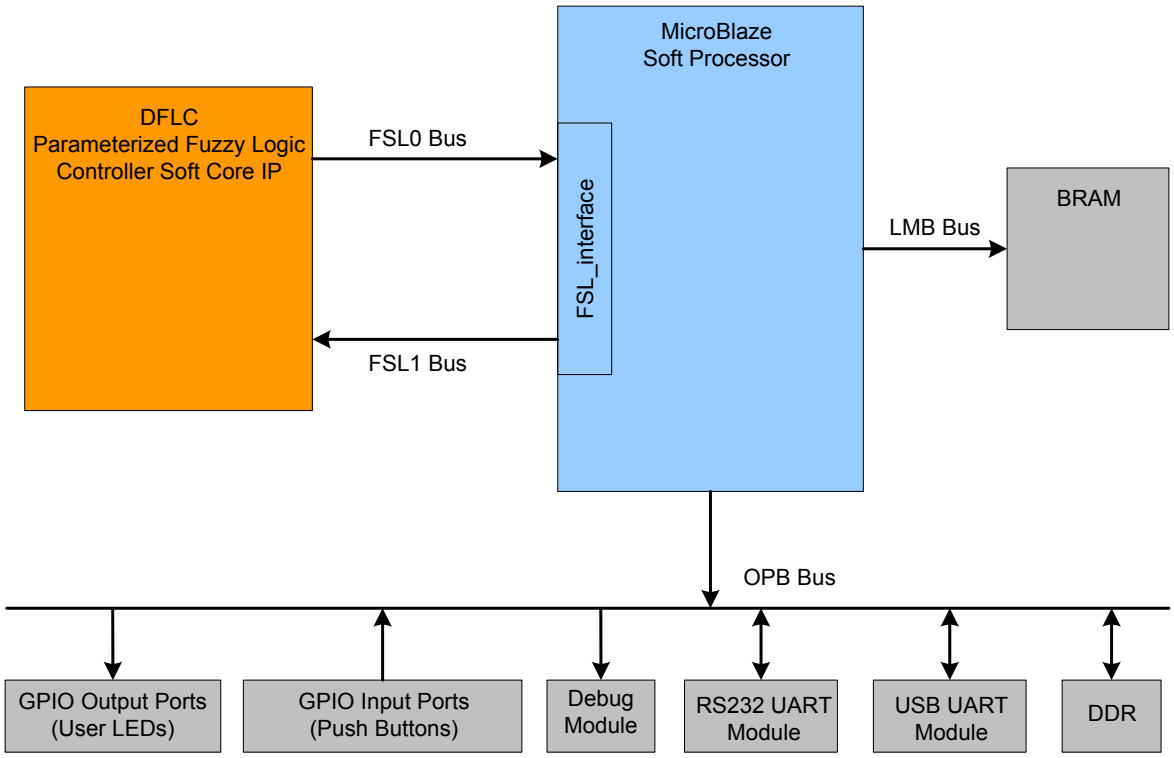
8.3 Περιγραφή Υλικού σε Υψηλό Επίπεδο

Στο Σχήμα 8.3 φαίνεται μια περιγραφή σε υψηλό επίπεδο της αρχιτεκτονικής του SoC. Ο μαλακός πυρήνας του επεξεργαστή Microblaze [41][84] είναι ένας από τους διαθέσιμους πυρήνες του Xilinx Embedded Development Kit [85] της εταιρίας Xilinx. Ο όρος μαλακός πυρήνας (*soft core*), σημαίνει ότι η υλοποίηση της συγκεκριμένης λειτουργικής μονάδας του επεξεργαστή κάνει χρήση της διαθέσιμης προγραμματιζόμενης λογικής του FPGA και όχι ένα δεσμευμένο κομμάτι πυριτίου (*σκληρός πυρήνας – hard core*) πάνω στην ψηφίδα από κατασκευής του FPGA.

Ο επεξεργαστής Microblaze βασίζεται στην αρχιτεκτονική μηχανών περιορισμένου συνόλου εντολών (*Reduced Instruction Set Computer* – RISC) και έχει αρκετές ομοιότητες με την αρχιτεκτονική DLX που περιγράφεται στην αναφορά [86]. Χαρακτηριστικό του Microblaze είναι ο αγωγός $3^{ων}$ βαθμίδων, ενώ οι περισσότερες εντολές ολοκληρώνονται σε ένα μόνο κύκλο ρολογιού, επίσης ο δίαυλος εντολών (*instruction set*) και ο δίαυλος δεδομένων του (data words) είναι 32-bit. Ο πυρήνας του επεξεργαστή έχει τη δυνατότητα να επιτύχει μία συχνότητα λειτουργίας της τάξης των 100 MHz στην οικογένεια των FPGA Spartan 3 που χρησιμοποιούμε. Επιπλέον ο επεξεργαστής Microblaze δίνει τη δυνατότητα διασύνδεσης με τον On-chip περιφερειακό δίαυλο επικοινωνίας (*On-chip Peripheral Bus* – OPB) [87] που επιτρέπει τη γρήγορη πρόσβαση σε ένα ευρύ αριθμό περιφερειακών λειτουργικών μονάδων, καθώς επίσης και με τον τοπικό δίαυλο μνήμης (*Local Memory Bus* – LMB) [88] για τη γρήγορη πρόσβαση σε τοπικές μνήμες (συνήθως ενσωματωμένη μνήμη (*Block* RAM – BRAM) μέσα στο FPGA). Επιπροσθέτως ο επεξεργαστής έχει τη δυνατότητα διασύνδεσης με έναν γρήγορο δίαυλο επικοινωνίας (*Fast Simplex Bus* – FSL) [89], που καθιστά εύκολη τη διασύνδεση του με μαλακούς πυρήνες τρίτων, έτσι ώστε οι τελευταίοι να λειτουργήσουν ως συνεπεξεργάστες για την ταχύτερη εκτέλεση κρίσιμων από πλευράς χρόνου εκτέλεσης ή πολύπλοκων αλγορίθμων. Συνοψίζοντας μπορούμε να πούμε ότι δίαυλος FSL είναι ένας δίαυλος μονής κατεύθυνσης (*unidirectional*) από σημείο σε σημείο (*point-to-point*) που συμπεριφέρεται ως μέσο αλληλεπίδρασης (*interface*) για τη ροή των δεδομένων από και προς τον επεξεργαστή. Κάθε κανάλι FSL παρέχει ένα μικρής καθυστέρησης (*latency*) μέσο αλληλεπίδρασης στον αγωγό (pipeline) του επεξεργαστή που επιτρέπει τη διασύνδεση του με άλλους πυρήνες για την επιτάχυνση του χρόνου εκτέλεσης των εντολών του. Η παρούσα εφαρμογή που αναλύεται στο κεφάλαιο αυτό χρησιμοποιεί το DFLC ως συνεπεξεργάστη του Microblaze μέσω του διαύλου FSL για τους λόγους που αναφέρθηκαν προηγουμένως [90].

Όπως έχουμε είδη πει η αρχιτεκτονική του SoC αποτελείται από το DFLC που επικοινωνεί με το Microblaze επεξεργαστή μέσω του διαύλου FSL, επίσης οι μνήμες BRAM επικοινωνούν με τον επεξεργαστή μέσω του διαύλου LMB, και τέλος όλες οι υπόλοιπες περιφερειακές μονάδες (όπως για παράδειγμα οι γενικής χρήσης είσοδοι/έξοδοι (*General Purpose* IO *ports* – GPIO), το UART (*Universal Asynchronous Receive Transmit Module*) που βρίσκονται στο FPGA μέσω του διαύλου OPB (βλ. Παράρτημα Α.3).

Συνοπτικά λοιπόν μπορούμε να πούμε ότι το DFLC έχει αναλάβει τον αλγόριθμο της παρακολούθησης πορείας, ενώ ο Microblaze επεξεργαστής κυρίως εκτελεί το C κώδικα που έχει τη γενική εποπτεία της ροής ελέγχου του συστήματος.



Σχήμα 8.3: Σχηματικό διάγραμμα υψηλού επιπέδου της αρχιτεκτονικής υλικού του SoC

8.4 Ασαφής Αλγόριθμος Παρακολούθησης Πορείας

Ο ασαφής αλγόριθμος παρακολούθησης πορείας (*fuzzy logic path tracking algorithm*) βασίζεται στην αναφορά [73], ενώ έχουν γίνει κάποιες μετατροπές έτσι ώστε να είναι δυνατή η προσαρμογή του στην πλατφόρμα υλικού του FPGA. Όπως αναφέρθηκε και παραπάνω, ο πρωτότυπος αλγόριθμος αποτελείται από έναν 9x9 τύπου Mamdani ασαφή ελεγκτή παρακολούθησης πορείας (FL *tracker*) με Γκαουσιανές συναρτήσεις συμμετοχής, ενώ στην εφαρμογή αυτή ο ίδιος ελεγκτής παρακολούθησης πορείας μετατράπηκε μέσω της συνάρτησης *mam2sug* του MATLAB σε Takagi–Sugeno τύπου ελεγκτή με Τριγωνικές συναρτήσεις συμμετοχής. Αξίζει να σημειωθεί ότι από την προσομοίωση που έγινε και με τους δύο τύπους ελεγκτών (Mamdani–Takagi), ο δεύτερος έδωσε πολύ καλύτερα αποτελέσματα από τον πρώτο.

Το πρόβλημα της παρακολούθησης πορείας μπορεί να τυποποιηθεί μαθηματικά ως εξής. Έστω Σ ένα σύστημα που περιγράφεται από την παρακάτω εξίσωση,

$$\begin{aligned} \dot{x} &= f(t,x,u) \\ y &= h(t,x,u) \end{aligned} \tag{8.1}$$

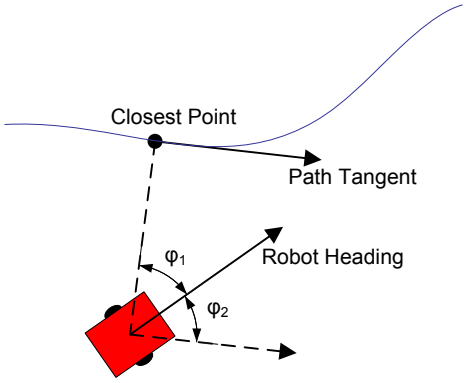
όπου $x \in \mathbb{R}^n$ είναι το διάνυσμα κατάστασης, $u \in \mathbb{R}^m$ το διάνυσμα εισόδου και $y \in \mathbb{R}^k$ είναι το διάνυσμα εξόδου. Έστω $x_r(t)$ μια εφικτή τροχιά αναφοράς στον χώρο κατάστασης που ικανοποιεί την Εξίσωση 8.1. Η λύση αυτή αντιστοιχεί σε μια είσοδο αναφοράς u_r, π.χ., $\dot{x}_r = f(t,x_r,u_r)$. Ο σκοπός είναι να βρούμε ένα νόμο ανάδρασης $u = u(t,x,x_r,u_r)$ τέτοιο

ώστε $\lim_{x\to\infty}\left(x(t)-x_r(t)\right)=0$. Το πρόβλημα παρακολούθησης πορείας μπορεί να διατυπωθεί με παρόμοιο τρόπο με τη διαφορά ότι πλέον ο σκοπός μας είναι να παρακολουθήσουμε την *εικόνα* της τροχιάς αναφοράς $x_r(t)$. Κατά αυτόν τον τρόπο, είναι δυνατή η επαναπαραμετροποίηση της εικόνας μέσω μια παραμέτρου $\sigma(t)$. Η απεικόνιση, $\sigma: I_1 \to I_2$, $\sigma(t)' \neq 0$, $t \in I_1$, είναι μια αμφίρριψη (bijection) στα δυο χωρία I_1, I_2 που δίνει μια σχέση ισοδυναμίας μεταξύ των δυο καμπυλών $x_r(\sigma(t)), x_r(t)$. Ουσιαστικά αυτή η επαναπαραμετροποίηση επιβάλει ένα διαφορετικό *χρονικό κανόνα* στην καμπύλη, π.χ., αλλάζει την ταχύτητα διάσχισης της καμπύλης. Στην εφαρμογή της παρακολούθησης πορείας που μελετάμε η ταχύτητα δεν σχετίζεται αυστηρά με το πρόβλημα και επομένως μπορούμε να τη θεωρήσουμε σταθερή τροποποιώντας κατάλληλα την είσοδο. Στην παρούσα εφαρμογή χρησιμοποιήθηκε το κινηματικό μοντέλο Dubin's Car [91]. Το συγκεκριμένο μοντέλο περιγράφει ένα σύστημα 3ου βαθμού που δίνεται από την Εξίσωση 8.2,

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{u}\end{bmatrix}=\begin{bmatrix}u\cos\theta\\ u\sin\theta\\ 0\end{bmatrix}+\begin{bmatrix}0\\ 0\\ \kappa\end{bmatrix}\kappa \tag{8.2}$$

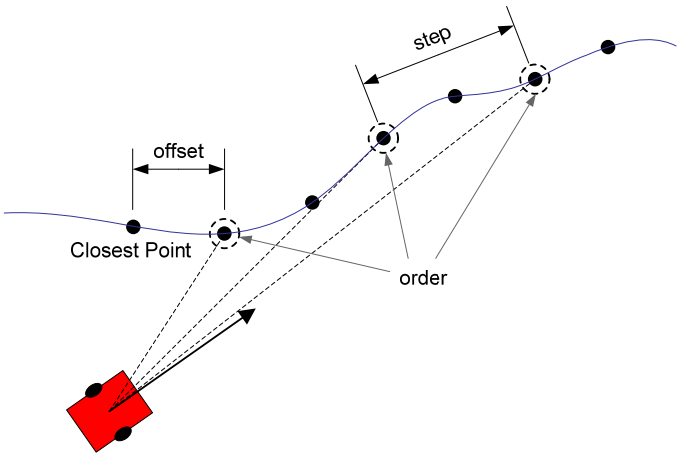
τα *x, y* είναι οι συντεταγμένες του κεντρικού σημείου του άξονα μετάδοσης του ρομπότ σε σχέση με το σύστημα αναφοράς, θ είναι η διεύθυνση του αναφορικά με το x-άξονα, *u* είναι η γραμμική του ταχύτητα και *κ* είναι η καμπυλότητα του ρομπότ. Λόγω του ότι το ρομπότ σταματάει με μηδενική ταχύτητα και κατά συνέπεια το σύστημα γίνετε μη ελέγξιμο, θέσαμε την ταχύτητα σε μια σταθερή τιμή. Από την Εξίσωση 8.2 είναι φανερό ότι το σύστημα έχει μετατραπεί σε αφινικό (*control-affine system*) ως προς τον έλεγχο με μη μηδενικό όρο ολίσθησης. Αυτό είναι φυσικά λογικό αφού από τη στιγμή που θέτουμε την ταχύτητα σταθερή, το σύστημα απλά μετατρέπεται στο μοντέλο του απλού αυτοκινήτου (*Simple Car*) [92] που δεν σταματάει ποτέ. Παρόλα αυτά στα πραγματικά πειράματα που γίνανε, αν και η ταχύτητα στην τελική κατάσταση γίνεται μηδέν, αυτό δεν επηρεάζει τον έλεγχο. Στην Εξίσωση 8.2 η είσοδος ελέγχου είναι η καμπυλότητα *κ* και όχι η γωνιακή ταχύτητα *ω*. Κατά αυτόν τον τρόπο επιτυγχάνουμε την από-σύζευξη του ελέγχου της ταχύτητας από το πρόβλημα παρακολούθησης πορείας, αφού η καμπύλη που ακολουθείται στο επίπεδο από το ρομπότ μπορεί να περιγραφεί αποκλειστικά και μόνο από την καμπυλότητα. Στην πραγματικότητα, εφόσον μια επίπεδη καμπύλη μπορεί να παραμετροποιηθεί από την καμπυλότητα της και το μήκος τόξου της $s(t)$, τότε αν μετατρέψουμε τη $s(t)$ σε μια αυστηρά αύξουσα συνάρτηση (θέτοντας την ταχύτητα σταθερή), καταφέρνουμε τελικά ελέγχοντας την καμπυλότητα *κ* να προσδιορίσουμε την πορεία του ρομπότ στο επίπεδο.

Ο ασαφής παρακολουθητής πορείας δέχεται ως εισόδους δύο γωνίες και παρέχει στην έξοδο του την καμπυλότητα *κ* με την οποία πρέπει να στραφεί το ρομπότ. Παίρνουμε ως υπόθεση ότι η πορεία δίνεται στον ελεγκτή ως μία σειρά σημείων στο $\mathbb{R}^2$ που ακολουθούν ένα σταθερό βήμα δειγματοληψίας Δs (*fixed sampling spacing*). Σε κάθε βρόχο ελέγχου λαμβάνεται το κοντινότερο σημείο της πορείας και στη συνεχεία υπολογίζονται οι δύο γωνίες εισόδου φ_1, φ_2. Η φ_1 είναι η γωνία του κοντινότερου σημείου ως προς την πορεία (*heading*) του ρομπότ, και η γωνία φ_2 αντίστοιχα αντιπροσωπεύει την εφαπτομένη του τρέχοντος σημείου της πορείας. Οι γωνίες φ_1, φ_2 επεξηγούνται διαγραμματικά στο Σχήμα 8.4.



Σχήμα 8.4: Επεξήγηση των εισόδων του ελεγκτή

Στη μελέτη [73] εισάγεται μία τεχνική που αναφέρεται ως «spatial window» που βελτιστοποιεί τον έλεγχο παρακολούθησης. Η τεχνική αυτή βασίζεται στην ιδέα της δημιουργίας ενός «χωρικού-παραθύρου» (*spatial-window*) της πορείας αντί ενός μοναδικού σημείου. Κατά αυτό τον τρόπο το εκάστοτε παράθυρο της πορείας χρησιμοποιείται για τον υπολογισμό της κατευθυντήριας εντολής (*steering command*) που ως συνέπεια εισάγει την έννοια της ύπαρξης «προοπτικής» στον ελεγκτή. Το «spatial window» ορίζεται από τρεις παραμέτρους: το βαθμό του παραθύρου (*window order*), που δηλώνει τον αριθμό των σημείων που έχει το παράθυρο, το βήμα του παραθύρου (window step) ή αλλιώς τον αριθμό των σημείων που παραλείπονται σε κάθε ενεργό σημείο, και τη μετατόπιση του παραθύρου (*window offset*) ή διαφορετικά τον αριθμό των σημείων αρχίζοντας τη μέτρηση από το κοντινότερο σημείο που κινεί το παράθυρο εμπρόσθια στη πορεία. Οι προαναφερθέντες παράμετροι επεξηγούνται με τη βοήθεια διαγράμματος στο Σχήμα 8.5.



Σχήμα 8.5: Επεξήγηση της τεχνικής «spatial window»

Εν συνεχεία κάθε σημείο του παραθύρου εισάγεται στον ελεγκτή και υπολογίζεται η καμπυλότητα του. Συνεπώς, σε κάθε βρόχο ελέγχου υπολογίζονται *n* καμπυλότητες, όπου *n* ο βαθμός του παραθύρου. Τελικά από το σύνολο των καμπυλοτήτων υπολογίζεται η τελική καμπυλότητα που δέχεται το ρομπότ ως εντολή. Για τον υπολογισμό αυτό και για

να διατηρηθεί η απλότητα επιλέχθηκε το αριθμητικό μέσο των καμπυλοτήτων που δίνεται από την Εξίσωση 8.3 ως εξής:

$$\kappa = \frac{\kappa_1 + \kappa_2 + \ldots + \kappa_n}{n} \tag{8.3}$$

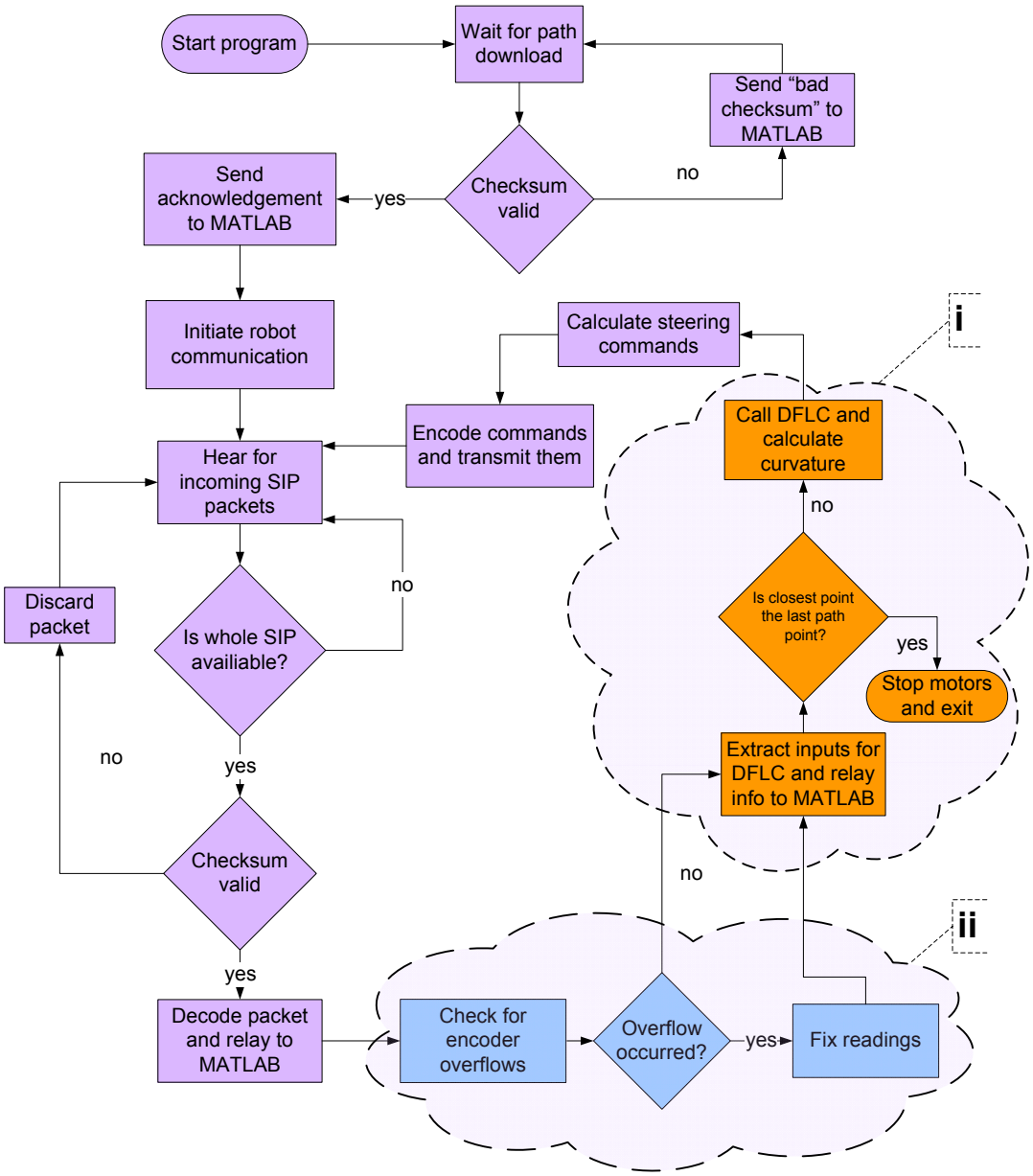
Η τεχνική «spatial window» επιτυγχάνει ομαλότερο έλεγχο ως προς την παρακολούθηση πορείας και σε γενικές γραμμές καλύτερη απόδοση. Επιπλέον συμβάλει στην υλοποίηση πιο σθεναρού ελέγχου και στη μείωση του θορύβου της πορείας.

8.5 Πρόγραμμα Επίβλεψης Ελέγχου

Το πρόγραμμα που έχει την καθολική εποπτεία του συντονισμού όλων των διαδικασιών υλοποιήθηκε σε C γλώσσα (βλ. Παράρτημα Α.3) με τη βοήθεια της εφαρμογής Platform Studio Software Development Kit (SDK), που είναι μέρος του πακέτου EDK Platform Studio, και εκτελείται στον επεξεργαστή Microblaze που βρίσκεται μέσα στο FPGA. Ο επεξεργαστής είναι χρονισμένος στα 71 MHz και κάνει χρήση RISC συνόλου εντολών. Για την εξυπηρέτηση της επικοινωνίας με τον έξω κόσμο χρησιμοποιούμε δύο κανάλια εισόδων/εξόδων, ένα σειριακό (RS232) και ένα USB (*Universal Serial Bus*) με 16 byte προσωρινής μνήμης (*buffer*) στις εισόδους και εξόδους. Το διάγραμμα ροής του κυρίως προγράμματος φαίνεται στο Σχήμα 8.6.

Κατά την έναρξη της εκτέλεσής του, το πρόγραμμα περιμένει δεδομένα εισόδου από το χρήστη είτε για να δεχτεί μία νέα πορεία, είτε όχι. Αυτή η διαδικασία ελέγχεται από δύο πιεστικούς διακόπτες (*push buttons*) και σηματοδοτείτε από δύο LED που βρίσκονται στην πλακέτα που φιλοξενεί το FPGA. Εάν πατηθεί ο πιεστικός διακόπτης Α, τότε το πρόγραμμα είναι σε κατάσταση να δεχτεί (μέσω της USB σύνδεσης) τα εισερχόμενα πακέτα δεδομένων από το τη MATLAB εφαρμογή. Αμέσως μόλις η πορεία έχει φορτωθεί στο FPGA, υπολογίζεται το άθροισμα ελέγχου λάθους (*checksum*) και αποστέλλεται μια επιβεβαίωση στη MATLAB εφαρμογή και κατόπιν ο έλεγχος του προγράμματος μεταφέρεται πίσω στον εξωτερικό βρόχο όπου περιμένει ξανά για καινούργια εντολή από το χρήστη. Εάν πατηθεί ο πιεστικός διακόπτης Β, τότε το πρόγραμμα συνεχίζει την εκτέλεση. Πιο συγκεκριμένα ανοίγει μια σύνδεση με το ρομπότ και αρχίζει τη διαδικασία συγχρονισμού στέλνοντας ειδικά διαμορφωμένα πακέτα δεδομένων. Σε αυτό το σημείο της ροής του προγράμματος το ρομπότ αρχίζει να στέλνει τα SIP πακέτα στο FPGA.

Στη συνέχεια το πρόγραμμα μεταφέρεται στον κύριο βρόχο ελέγχου (*main control loop*) και περιμένει για τα εισερχόμενα SIP πακέτα, μετακινώντας την προσωρινή μνήμη σε μία μεταβλητή πίνακα. Από τη στιγμή που θα υπάρξει ένα ολόκληρο έγκυρο πακέτο SIP, τότε αυτό αποκωδικοποιείται από όπου και εξάγονται τα δεδομένα της οδομετρίας (x, y, θ) και παράλληλα μεταβιβάζονται και στην εφαρμογή MATLAB. Λόγω ότι τα SIP πακέτα χρησιμοποιούν μικρό μήκος λέξης για την κωδικοποίηση των δεδομένων, μαζί με το γεγονός ότι η ανάλυση της οδομετρίας είναι σχετικά μεγάλη (της τάξης του χιλιοστού του μέτρου), οι μετρήσεις που προέρχονται από τις οπτικούς κωδικοποιητές (*optical encoders*) δεν είναι εύκολο να χρησιμοποιηθούν για τον εντοπισμό θέσης (*localization*).

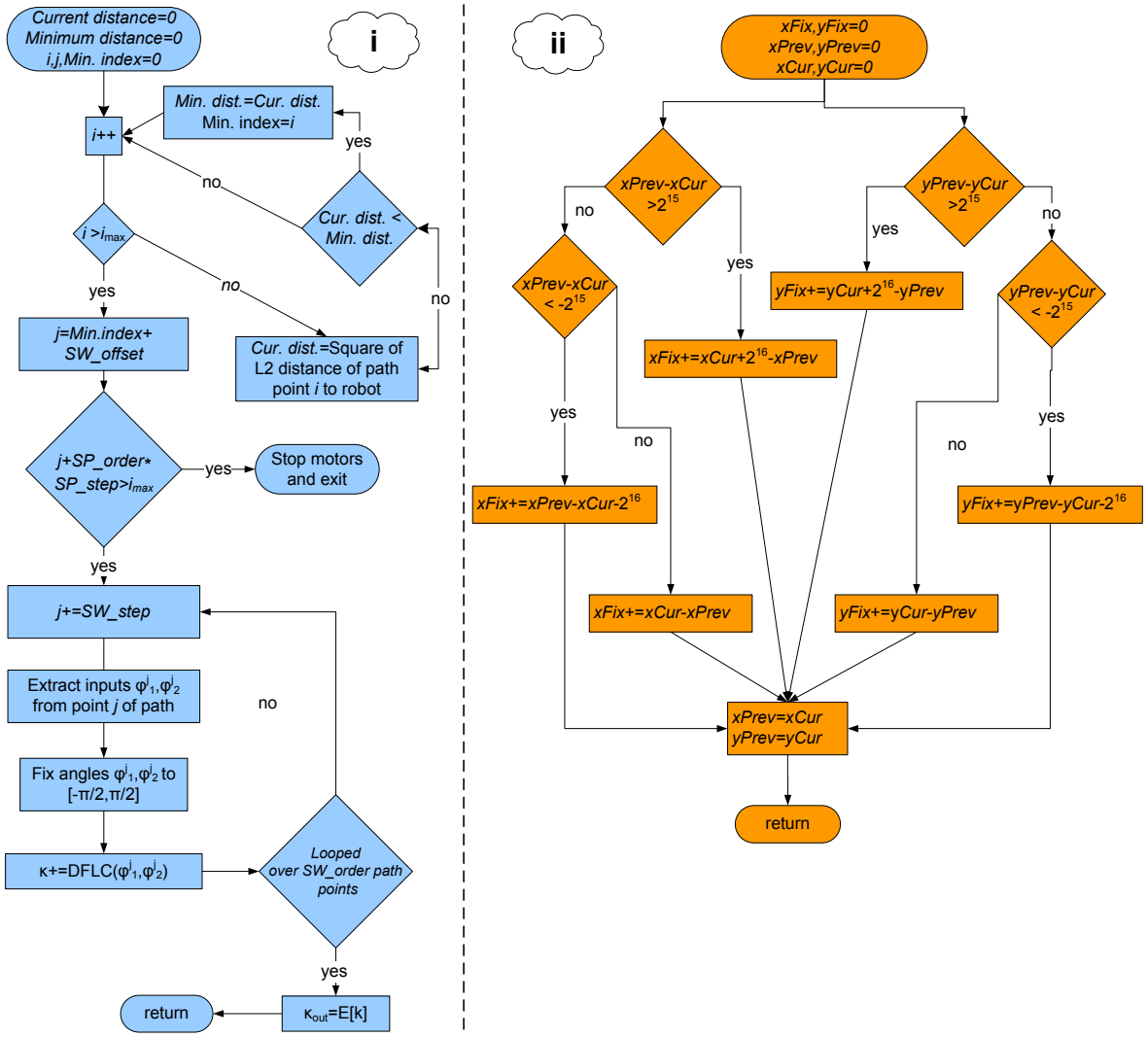


Σχήμα 8.6: Διάγραμμα ροής του κυρίως προγράμματος

Τα δεδομένα στα SIP πακέτα κωδικοποιούνται ως προσημασμένοι ακέραιοι των 16-bit (*signed integers*). Τα πακέτα δεδομένων της οδομετρίας πολλαπλασιάζοντας με το συντελεστή 0.485, που ανεβάζει το ωφέλιμο εύρος σε 15892 έως 15892 mm, ή -16 έως 16 m, που και πάλι είναι αρκετά περιορισμένο. Προκειμένου να μεγαλώσει το εύρος, περισσότερο χρησιμοποιούνται δύο εσωτερικές μεταβλητές συντεταγμένες *xFix, yFix* τύπου *double*. Κατά αυτό τον τρόπο, θεωρούμε τα δεδομένα των SIP πακέτων *x, y* ως *Δx, Δy* και μηδενίζουμε τις συντεταγμένες του κέντρου του ρομπότ (registration point) όποτε έχουμε υπερχείλιση. Για παράδειγμα όταν η μη-βαθμονομημένη συντεταγμένη *x* κινείται από το 32766 στο 32770 έχουμε υπερχείλιση στα δεδομένα της μεταβλητής *x* του πακέτου που ως αποτέλεσμα έχει να θεωρηθεί από το πρόγραμμα ως αρνητικός αριθμός, που είναι φυσικά λάθος. Για να αποφύγουμε μια τέτοια κατάσταση οι τιμές δύο συνεχόμενων πακέτων φυλλάσονται στη μνήμη ως *xPrev, xCur* για τη *x* συντεταγμένη, και η διαφορά τους προστίθεται στις απόλυτες συντεταγμένες *xFix, yFix*. Έτσι για παράδειγμα αν

εντοπιστεί ότι ένα Δx είναι μεγαλύτερο από 2^{15} αυτό σημαίνει ότι έγινε υπερχείλιση, οπότε η διαφορά διορθώνεται ανάλογα. Ο τρόπος επίλυσης του προβλήματος αυτού επεξηγείται στο διάγραμμα ροής που δίνεται στο Σχήμα 8.7(ii).

Αμέσως μετά το τέλος του υπολογισμού των δεδομένων οδομετρίας, η ροή του προγράμματος μεταφέρεται στον αλγόριθμο που είναι υπεύθυνος για τον έλεγχο της παρακολούθησης πορείας (Σχήμα 8.7(i)). Ο συγκεκριμένος αλγόριθμος υλοποιεί την τεχνική «spatial window» του παρακολουθητή πορείας.



Σχήμα 8.7: Διαγράμματα ροής (i) ρουτίνα για το «spatial window», (ii) ρουτίνα διόρθωσης της υπερχείλισης των οπτικών κωδικοποιητών

Λαμβάνει το κοντινότερο σημείο της πορείας στο ρομπότ και υπολογίζει τις δυο εισόδους του ελεγκτή ϕ_1^j, ϕ_2^j του σημείου j του παραθύρου και στη συνεχεία καλεί το DFLC με ορίσματα τις εισόδους αυτές. Όταν ολοκληρωθεί η διαδικασία επιστρέφει τη μέση τιμή όλων των υπολογισμένων καμπυλοτήτων κ_j και επιστρέφει στον κύριο βρόχο (*main loop*) του προγράμματος. Ο αλγόριθμος σταματάει το ρομπότ εάν ανιχνεύσει ότι βρίσκεται πολύ κοντά στη πορεία και επομένως το πλήρες «spatial window» δεν μπορεί να χρησιμοποιηθεί στη πορεία.

Ούτως ώστε να βρούμε το κοντινότερο σημείο της πορείας, χρησιμοποιούμε το τετράγωνο της Ευκλείδειας L^2 - νόρμας, δεδομένου ότι ο υπολογισμός της τετραγωνικής ρίζας της κανονικής Ευκλείδειας απόστασης είναι υπολογιστικά δαπανηρή. Οι μεταβλητές που χρησιμοποιούνται για τον υπολογισμό της νόρμας είναι τύπου *long long int (64bit integer)* και όχι τύπου *double*, που αυξάνει σημαντικά την ταχύτητα εκτέλεσης του κώδικα στον επεξεργαστή Microblaze χωρίς τη χρήση συν-επεξεργαστή κινητής υποδιαστολής (*Floating Point Unit* – FPU). Εδώ αξίζει να σημειώσουμε ότι ο προηγούμενος χειρισμός έχει μεγάλη σημασία στην επίδοση του αλγορίθμου μιας και ο μεγαλύτερος υπολογιστικός φόρτος λαμβάνει χώρα κατά τη διάρκεια αυτού του υπολογισμού. Λαμβάνοντας υπόψη ότι η πορεία αποτελείται από εκατοντάδες δείγματα σημείων, ο αλγόριθμος καλείται να τα ελέγξει όλα για να βρει το κοντινότερο σημείο. Είναι αναγκαίο λοιπόν η προηγούμενη διαδικασία να γίνει όσο το δυνατό γρηγορότερα, ώστε να αποφευχθεί η περίπτωση τα πακέτα που λαμβάνονται από το ρομπότ να μην αναθέτονται έγκαιρα στις μεταβλητές τους, πράγμα που θα οδηγήσει σε τεμαχισμένα SIP πακέτα, και απώλεια συγχρονισμού του FPGA με το ρομπότ αφού τα πακέτα δεν μπορούν πλέον να ανακτηθούν σωστά από το FPGA.

Μετά το υπολογισμό της καμπυλότητας, σειρά έχει η δημιουργία των κατευθυντήριων εντολών προς το ρομπότ. Το Pioneer ρομπότ που χρησιμοποιούμε δεν διαθέτει κάποια ειδική εντολή σχετικά με την καμπυλότητα, οπότε η τελευταία πρέπει να εκφραστεί συναρτήσει της γραμμικής και της γωνιακής ταχύτητας. Κατά αυτόν τον τρόπο η καμπυλότητα κ εκφράζεται από την Εξίσωση 8.4 ως:

$$\kappa = \frac{d\theta}{ds} = \frac{d\theta / dt}{ds / dt} = \frac{\omega}{\upsilon} \tag{8.4}$$

όπου θ είναι η κατευθυντήρια γωνία του ρομπότ, s η απόσταση που έχει διανύσει το ρομπότ, ω η γωνιακή ταχύτητα (rad/sec) και υ η γραμμική ταχύτητα (m/sec). Τα πακέτα εντολών που δέχεται το ρομπότ πρέπει να περιέχουν τη γραμμική ταχύτητα σε mm/sec και τη γωνιακή ταχύτητα σε deg/sec.

Αν υποθέσουμε ότι η πραγματική καμπυλότητα που πρέπει να ακολουθήσει το ρομπότ είναι κ σε rad/m, και έστω ότι η γραμμική ταχύτητα υ' (mm/sec) δίνεται, τότε η γωνιακή ταχύτητα ω' (deg/sec) που θα σταλθεί στο ρομπότ με τη μορφή εντολής υπολογίζεται ως εξής (βλ. Εξ. 8.5):

$$\kappa = \frac{\omega}{\upsilon}\left(\frac{rad}{m}\right) = \frac{\omega'(\text{deg}/\text{sec})}{\upsilon'(mm/\text{sec})}\frac{\pi/180}{/1000} \Rightarrow \omega' = \kappa \cdot \upsilon' \cdot \frac{180}{1000 \cdot \pi} \tag{8.5}$$

Η πραγματική καμπυλότητα κ που παίρνουμε από το DFLC είναι ένας προσημασμένος ακέραιος αριθμός με εύρος 12-bit ανάλυση, δηλαδή $-2^{11} \le k \le 2^{11} - 1$ ή -2048 έως 2047. Εδώ κανονικά το θετικό εύρος των τιμών θα έπρεπε να διαιρεθεί με 2047, αλλά για να αποφύγουμε τον έλεγχο πρόσημου, διαιρούμε με 2048. Αυτό εισάγει 1 λιγότερο σημαντικό bit (*least significant bit* - lsb) λάθος στο θετικό εύρος τιμών, που όμως θεωρείται ασήμαντο συγκρινόμενο με τα λάθη κβάντισης που εισάγει το ρομπότ (βλέπε παράγραφο §8.6). Η καμπυλότητα κανονικοποιείτε στο [-1,1] και πολλαπλασιάζετε με μια σταθερά μέγιστη καμπυλότητα κ_{max} λόγω ότι η διαφορική κίνηση (*differential drive*) του ρομπότ δεν θέτει

κάποιο όριο στην ακτίνα περιστροφής του. Με αυτό τον τρόπο ελέγχεται η ελάχιστη ακτίνα στρέψης του ρομπότ. Συνεπώς η Εξίσωση 8.5 γίνεται (βλ. Εξ. 8.6):

$$\omega' = \frac{\kappa}{2048}\kappa_{\max} \cdot \upsilon' \cdot \frac{180}{1000 \cdot \pi} \quad (\pm 1 \text{ lsb}) \tag{8.6}$$

Μόλις η γραμμική ταχύτητα[†] *υ'* του ρομπότ εξαχθεί από το SIP πακέτο και η γωνιακή ταχύτητα *ω'* υπολογιστεί με τη βοήθεια της Εξίσωσης 8.6, τότε οι εντολές κίνησης στέλνονται πίσω στο ρομπότ (ισοδυναμούν με μία αύξουσα καμπυλοειδή κίνηση του ρομπότ). Αξίζει να σημειώσουμε ότι η ταχύτητα ουσιαστικά απομπλέκετε από τον έλεγχο παρακολούθησης αφού η είσοδος ελέγχου είναι η καμπυλότητα. Η ταχύτητα μπορεί κάλλιστα να ελεγχθεί από έναν ανεξάρτητο ελεγκτή ταχύτητας. Θα χαρακτηρίζαμε καλύτερα το συγκεκριμένο ελεγκτή ως γεωμετρικό ελεγκτή παρακολούθησης πορείας από τι στιγμή που η καμπυλότητα είναι το μοναδικό μέγεθος που ορίζει εξολοκλήρου τη πορεία του ρομπότ. Σε άλλες εφαρμογές όπως κινηματικό-δυναμικό έλεγχο παρακολούθησης πορείας όπου οι η δυναμική του ρομπότ λαμβάνεται υπόψη θα έπρεπε να εισαχθεί και η ταχύτητα στον βρόχο ελέγχου. Στη συγκεκριμένη εφαρμογή ο έλεγχος ταχύτητας δεν είναι απαραίτητος στο πρόβλημα της παρακολούθησης αφού θεωρούμε ότι η ταχύτητα του ρομπότ είναι πολύ μικρή για να επηρεάσει την κινηματική του συμπεριφορά.

Σε αντίθεση με ένα πραγματικό τύπου-όχημα ρομπότ (*car-like robot*) που η καμπυλότητα και η πραγματική κίνηση περιορίζονται λόγω της μηχανικής διάταξης του συστήματος διεύθυνσης (γωνία στρέψης των τροχών), στην παρούσα εφαρμογή (το ρομπότ μπορεί να γυρίσει επιτόπου γύρω από τον άξονα του) μία τέτοια συμπεριφορά car-like robot εξομοιώνεται με τον τρόπο που εξηγήσαμε παραπάνω. Ο τρόπος αυτός (τρόπος εξομοίωσης της καμπυλότητας) εισάγει ένα νέο πρόβλημα που αναλύεται στην επόμενη παράγραφο.

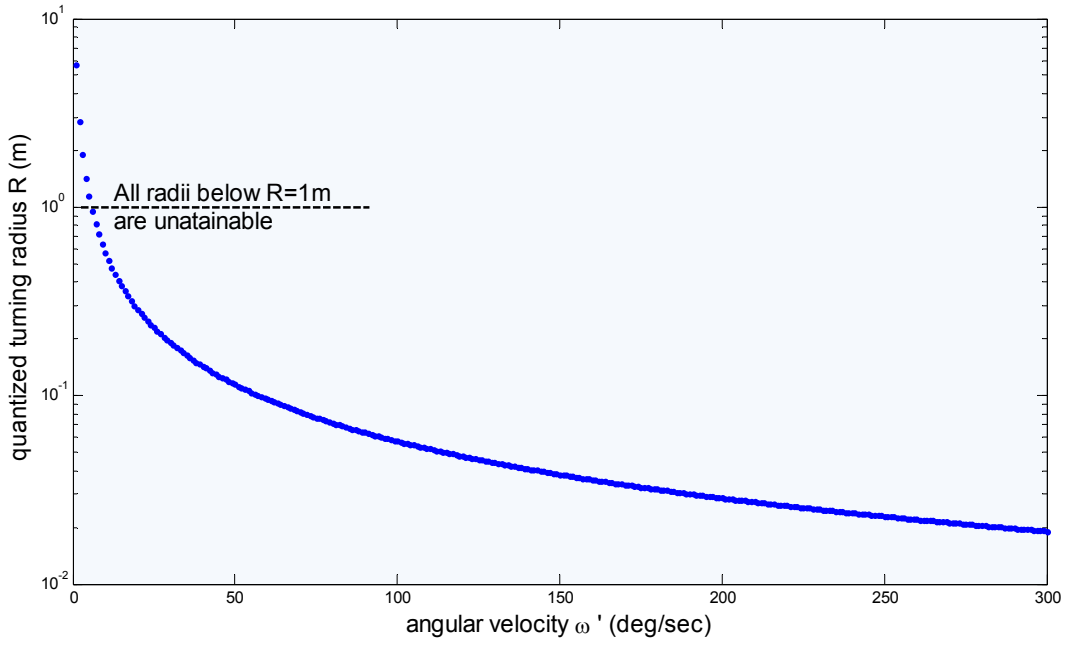
8.6 Προβλήματα Κβαντοποίησης

Το Pioneer ρομπότ χρησιμοποιεί 16-bit ακέραιους για την κωδικοποίηση των SIP πακέτων άρα και για την κωδικοποίηση των ορισμάτων στα πακέτα εντολών. Το όρισμα της εντολής για τη γωνιακή ταχύτητα έχει εύρος [0, 300] deg/sec με 1 deg/sec/bit. Δεδομένου ότι η *ω'* δέχεται ακέραιες τιμές με ανάλυση deg/sec, πρέπει να γίνει επίσης κβάντιση της καμπυλότητας. Λύνοντας την Εξίσωση 8.5 ως προς *κ* (ή $R=1/\kappa$, ακτίνα στρέψης σε μέτρα, *m*) παίρνουμε (βλ. Εξ. 8.7),

$$R = \frac{\upsilon'}{\omega'} \cdot \frac{180}{1000 \cdot \pi} \tag{8.7}$$

[†] Το ρομπότ της εφαρμογής (Pioneer P3-DX8) έχει ενσωματωμένο έναν ελεγκτή ταχύτητας. Οι εντολές ταχύτητας που λαμβάνει προσαρμόζονται ανάλογα μέσω του αλγορίθμου ταχύτητας που χρησιμοποιεί. Επομένως η ταχύτητα που χρησιμοποιούμε για τον υπολογισμό της Εξίσωσης 8.6 δεν είναι η αρχική ταχύτητα που δόθηκε ως εντολή προς το ρομπότ, αλλά η πραγματική τρέχουσα ταχύτητα που ανάφερε το ρομπότ στο τελευταίο SIP πακέτο που έστειλε.

Από τη γραφική παράσταση της ακτίνας R με την κβαντισμένη γωνιακή ταχύτητα ω' για υ'=100 *mm/sec* (βλέπε Σχήμα 8.8) φαίνεται ότι η ανάλυση για R=5.73 (ω'=1) έως R=2.86 (ω'=2) είναι ανεπαρκής, ενώ για μεγάλες καμπυλότητες (μικρές ακτίνες) είναι υψηλή. Αυτό ως συνέπεια επηρεάζει τον τρόπο που συμπεριφέρεται ο ελεγκτής στο πρόβλημα παρακολούθησης πορείας. Όταν για παράδειγμα το ρομπότ βρίσκεται μέσα στη πορεία, δηλαδή $\varphi_1, \varphi_2 \approx 0$, ο DFLC στέλνει εντολές μικρής καμπυλότητας (μεγάλη ακτίνα) για την αποφυγή «νευρικών» κινήσεων (π.χ. ταλαντώσεις) στην πλοήγηση. Εάν η ανάλυση στο εύρος μικρών καμπυλοτήτων είναι ανεπαρκής, ο έλεγχος υποβαθμίζεται διότι οι εντολές καμπυλότητας ψαλιδίζονται (*clipped*) στο διαθέσιμο επίπεδο ανάλυσης. Επιπλέον, εφόσον υπάρχει περιορισμός στην ελάχιστη ακτίνα στρέψης, όλες οι τιμές στο Σχήμα 8.8 κάτω από το χαμηλότερο όριο R_{min} (ορίστηκε στο 1 *m* στην εφαρμογή που μελετάμε) δεν πραγματοποιούνται από το ρομπότ.



Σχήμα 8.8: Κβάντιση της πραγματικής ακτίνας στρέψης για υ'=100 mm/sec

Επειδή η ακτίνα R είναι μονοτονικά φθίνουσα ($\forall x, y$ όταν $x \le y$, τότε $f(x) \ge f(y)$) ως προς το ω', και το ω' αρχίζει από 1 deg/sec, μπορούμε να βρούμε την τιμή των διαθέσιμων επιπέδων κβάντισης, αντικαθιστώντας με R=1 στην Εξίσωση 8.7 και λύνοντας για ω' ως εξής (βλ. Εξ. 8.8):

$$L_{num} = \left\lfloor \upsilon' \cdot \frac{180}{1000 \cdot \pi} \right\rfloor \tag{8.8}$$

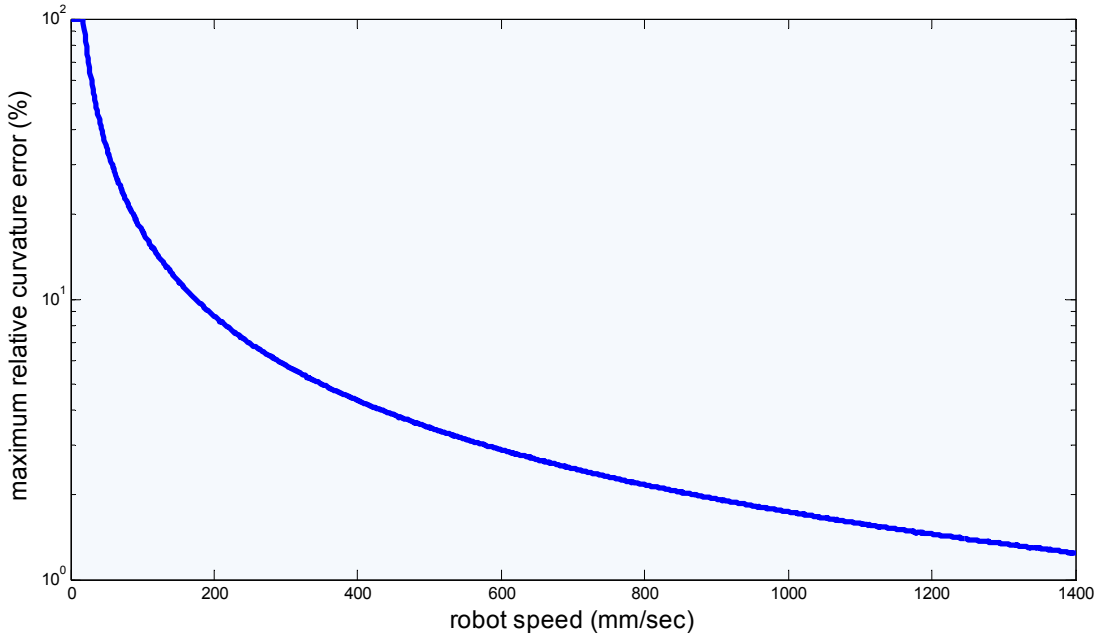
, όπου $\lfloor\ \rfloor$ είναι η συνάρτηση κάτω περικοπής (trancation).

Παρατηρώντας τις εξισώσεις 8.5 και 8.6 βλέπουμε ότι αυξάνοντας την ταχύτητα υ' , η μέγιστη ακτίνα στρέψης αυξάνει γραμμικά μαζί με την ανάλυση σε αυτό το εύρος, δηλαδή την τιμή L_{num}. Η προφανής λύση για να αυξήσουμε την ανάλυση της καμπυλότητας είναι να αυξήσουμε την ταχύτητα. Η λύση αυτή παρουσιάζει ως πρόβλημα την εύρεση μίας κατάλληλης ταχύτητας για το ρομπότ, μιας και οι χαμηλές ταχύτητες μειώνουν την

ανάλυση της καμπυλότητα, ενώ οι υψηλές ταχύτητες μπορούν να οδηγήσουν σε ένα μη αποκρινόμενο σύστημα. Προκειμένου να υπολογίσουμε ένα αποδεκτό επίπεδο λάθους μεταξύ της καμπυλότητας από το DFLC και της πραγματικής καμπυλότητας που ακολουθεί το ρομπότ, βρίσκουμε το μέγιστο σχετικό λάθος σε αντιπαράθεση με όλες τις διαθέσιμες ταχύτητες προς όλες τις διαθέσιμες εισόδους, ως εξής (βλ. Εξ. 8.9):

$$100 \cdot \max(\kappa_{DFLC} - \kappa_{ACTUAL}) / \kappa_{ACTUAL}, \forall \varphi_1, \varphi_2 \tag{8.9}$$

Κατά τον τρόπο αυτό γίνεται εύκολα αντιληπτό το μέγιστο δυνατό σχετικό λάθος για κάθε ταχύτητα μεταξύ της πραγματικής και της επιθυμητής καμπυλότητας. Η γραφική αναπαράσταση του λάθους δίνετε στο Σχήμα 8.9. Η ελάχιστη ακτίνα στρέψης ορίστηκε στο ένα μέτρο (κ_{max}=10^{-3}).



Σχήμα 8.9: Μέγιστο σχετικό λάθος μεταξύ της πραγματικής και της επιθυμητής καμπυλότητας στρέψης σε αντιπαράθεση με την ταχύτητα, για όλες τις εισόδους του ελεγκτή

Όπως φαίνεται στο Σχήμα 8.9, καθώς η ταχύτητα αυξάνει το λάθος μειώνεται. Ένας συνδυασμός λογικού κόστους και ποιότητας (*trade-off*) για την ταχύτητα είναι 1000 *mm/sec* (ή 1 *m/sec*) όπου το λάθος πέφτει κάτω από 1.745%. Τα διαθέσιμα επίπεδα κβάντισης σε αυτήν την ταχύτητα, όπως προκύπτουν από την Εξίσωση 8.8 είναι L_{num}=57. Ως εκ τούτου η ταχύτητα ορίστηκε στα 1000 *mm/sec* σε όλα τα πειράματα που έγιναν.

8.7 Αρχιτεκτονική Υλικού του SoC

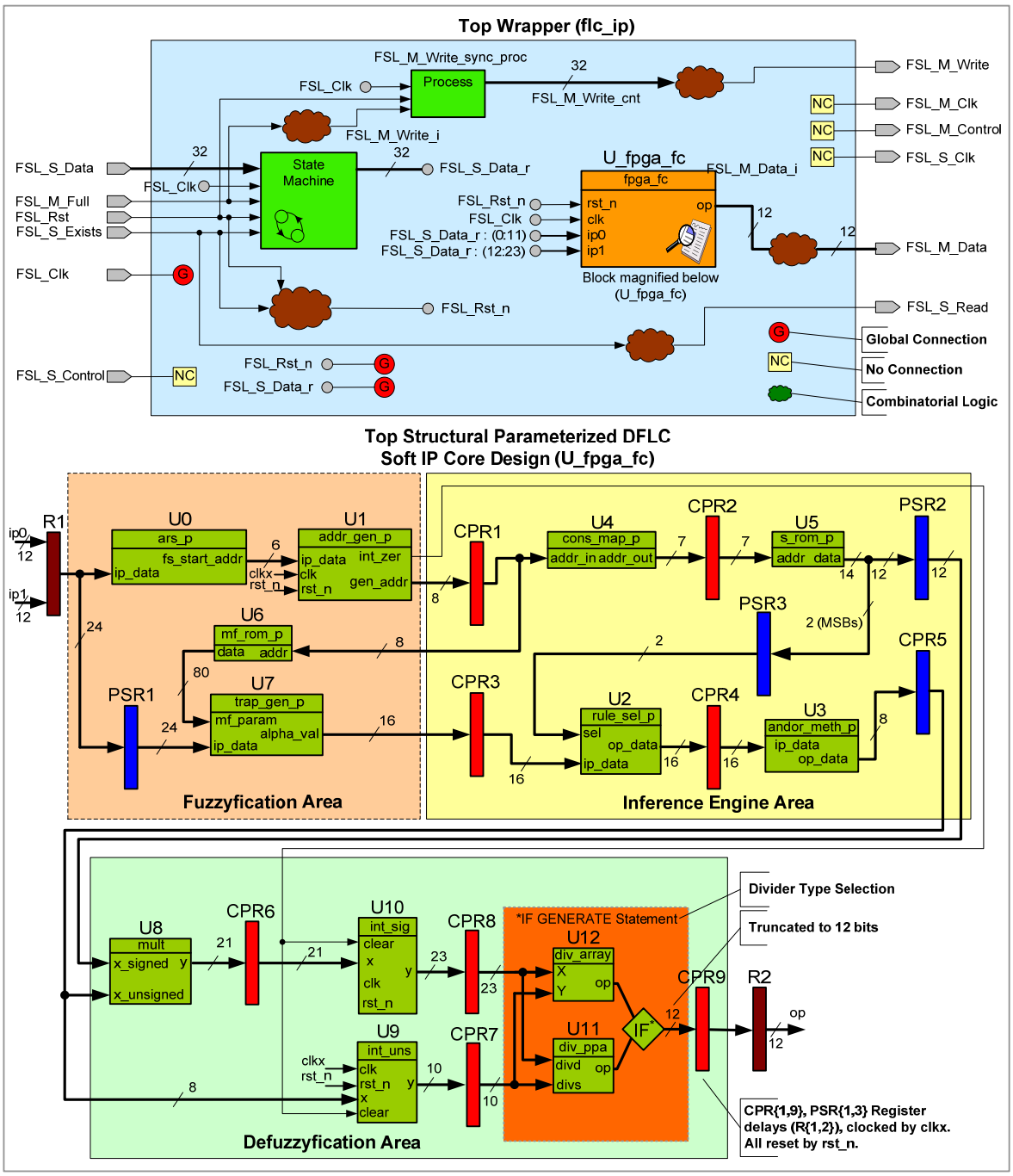
Η σχεδίαση του SoC που παρουσιάζουμε σε αυτό το κεφάλαιο υλοποιήθηκε στην πλατφόρμα (kit) ανάπτυξης Spartan-3 MB (με κωδικό DS–KIT–3SMB1500) της εταιρίας Memec Design [93]. Η πλατφόρμα Spartan-3 MB χρησιμοποιεί το FPGA Spartan-3 (XC3S1500–4FG676) σε συσκευασία 676–ακίδων (676–pin fine-grid array package) και

περιέχει περίπου 1.5 εκατομμύρια πύλες. Τα πιο σημαντικά χαρακτηριστικά της FPGA πλατφόρμας συνοψίζονται στον Πίνακα 5.1.

Xilinx XC3S1500-4FG676C Spartan-3 FPGA
16 M x 16 DDR μνήμη, 2 M x 16 flash μνήμη
Platform Flash ISP PROMs
10/100 Ethernet PHY, USB 2.0 και RS232
2 7-τμημάτων LED οθόνες
4 Γενικής χρήσης LEDs, 2 Πιεστικούς διακόπτες, 8 Θέσεων Dip Switches
On-board clock oscillator
JTAG configuration port
75 MHz Clock Oscillator
2 x 16 Χαρακτήρων LCD οθόνη
Δύο P160 slot επέκτασης
System ACE/User I/O Header
LVDS TX/RX interface

Πίνακας 8.1: Τα σημαντικότερα χαρακτηριστικά της πλατφόρμας Spartan-3 MB

Τα «CPR» μπλοκ στο Σχήμα 8.10 αντιπροσωπεύουν τον αριθμό (καταχωρητών) των βαθμίδων αγωγού (pipeline stages) της κάθε λειτουργικής μονάδας (*Component Pipeline Registers* – CPR), τα «PSR» μπλοκ σχετίζονται με τους καταχωρητές συγχρονισμού της διόδου δεδομένων (*datapath*) του κυκλώματος (*Path Synchronization Registers* – PSR), ενώ τα «U» μπλοκ αντιπροσωπεύουν τις υπόλοιπες λειτουργικές μονάδες του DFLC [61]. Ο πυρήνας U_fpga_fc ενσωματώνεται στην υψηλή δομικής περιγραφής οντότητα (top structural entity) flc_ip που παρέχει όλα τις απαραίτητες περιφερειακές μονάδες στον DFLC πυρήνα, ούτως ώστε ο τελευταίος να μπορεί να στείλει/λάβει δεδομένα στον δίαυλο FSL και συνεπώς να ακολουθεί τις προδιαγραφές που θέτει το προτόκολο του διαύλου FSL. Το σχηματικό διάγραμμα του flc_ip δίνετε στο Σχήμα 8.10. Οι τιμές των παραμέτρων (*generics*) που επιλέχθηκαν για τη διαμόρφωση του DFLC πυρήνα της εφαρμογής (ορίζονται σε ένα VHDL αρχείο ορισμών (*package*)) καθώς και τα χαρακτηριστικά του πυρήνα, επεξηγούνται στους Πίνακες 8.2 και 8.3 αντίστοιχα.
Ο πυρήνας U_fpga_fc έγινε σύνθεση με το εργαλείο Synplify Pro της εταιρίας Synplicity, ενώ οι υπόλοιπες λειτουργικές μονάδες με το εργαλείο Xilinx Synthesis Tool (XST) μέσα από το EDK Platform Studio της εταιρίας Xilinx. Το αρχείο edif (*Electronic Design Interchange Format)* του πυρήνα U_fpga_fc, που παρήγαγε το πρώτο εργαλείο συνδέθηκε (ενσωματώθηκε) με το flc_ip σαν «μαύρο-κουτί», που πρακτικά σημαίνει ότι κατά την εκτέλεση του XST εργαλείου σύνθεσης το U_fpga_fc παραμένει ανέπαφο. Η θέση και δρομολόγηση (*place and route*) του SoC σχεδίου στο FPGA ολοκληρώθηκε με τη βοήθεια της σειράς εργαλείων Xilinx ISE που μπορούν να φορτωθούν και μέσα από το EDK Platform Studio.



Σχήμα 8.10: Διάγραμμα της αρχιτεκτονικής του μαλακού πυρήνα DFLC

Ακολουθούν οι Πίνακες 8.2 και 8.3 που περιγράφουν αναλυτικά τις παραμέτρους και τα χαρακτηριστικά του DFLC όπως διαμορφώθηκε στην τελική μορφή που χρησιμοποιήθηκε στην πλατφόρμα των πειραμάτων.

Parameters (VHDL generics)	Value	Generic Description
ip_no	2	Number of inputs
ip_sz	12	Input bus width (bits)
op_no	1	Number of outputs
op_sz	12	Output bus width (bits)
FS_no	9	Number of membership functions (same for all inputs)
dy	8	Degree of Truth width
sel_op	0	Antecedent method connection (rule activation method): 0 : min, 1: prod, 2: max, 3: probor
div_type (Divider Model)	1	0 : restoring array 1 : LUT reciprocal approximation
PSR		**Signal Path Route**
psr1_no	1	ip_set→psr1_no→trap_gen_p
psr2_no	4	s_rom→psr2_no→mult
psr3_no	1	s_rom→psr3_no→rul_sel_p
psr4_no	1	cpr5→psr→int_uns
CPR		**Component (Entity) Name**
cpr1_no	1	addr_gen_p
cpr2_no	1	cons_map_p
cpr3_no	3	trap_gen_p
cpr4_no	0	rule_sel_p
cpr5_no	2	minmax_p
cpr6_no	1	mult
cpr7_no	0	int_uns
cpr8_no	0	int_sig
cpr9_no	2	div_array

Πίνακας 8.2: Παράμετροι που επιλέχθηκαν για το μαλακό πυρήνα DFLC της εφαρμογής

Ο χάρτης μνήμης που δείχνει τις διευθύνσεις που βρίσκονται τα περιφερειακά του συστήματος δίνετε στον Πίνακα 8.4. Οι πόροι που καταλαμβάνει στο FPGA το SoC σχέδιο μετά την τοποθέτηση και δρομολόγηση αναλύονται στον Πίνακα 8.5.

Συνοπτικά ο πυρήνας DFLC καταλαμβάνει 1600 (6%) LUTs, 4 Block Multipliers (MULT18X18s), 12 64x1 ROMs (ROM64X1), και 56 256x1 ROMs (ROM256X1). Η υλοποίηση χρησιμοποιεί δύο Digital Clock Manager (DCM) μονάδες (DCM_0 and DCM_1) για την παραγωγή των ρολογιών στο FPGA. Το SoC της εφαρμογής επιτυγχάνει συχνότητα χρονισμού (ρολόι συστήματος, μονάδα DCM_0) 14.085 ns ή ~71 MHz, ενώ η

μονάδα DCM_1 χρησιμοποιείτε για το χρονισμό της εξωτερικής από το FPGA μονάδας μνήμης (DDR RAM τύπος μνήμης).

Fuzzy Inference System (FIS) type	**Takagi-Sugeno zero-order type**
Inputs	2
Input resolution	12-bit
Outputs	1
Output resolution	12-bit
Antecedent Membership Functions (MF's)	9 Triangular or Trapezoidal shaped per fuzzy set
Antecedent MF Degree of Truth (α value) resolution width	8-bit
Consequent MF's	5 Singleton type
Consequent MF resolution	8-bit
Maximum number of fuzzy inference rules	81 (number of fuzzy sets $^{\text{number of inputs}}$)
AND method	MIN (T-norm operator implemented by minimum)
Implication method	PROD (product operator)
MF overlapping degree	2
Defuzzification method	Weighted average

Πίνακας 8.3: Χαρακτηριστικά του μαλακού πυρήνα DFLC της εφαρμογής

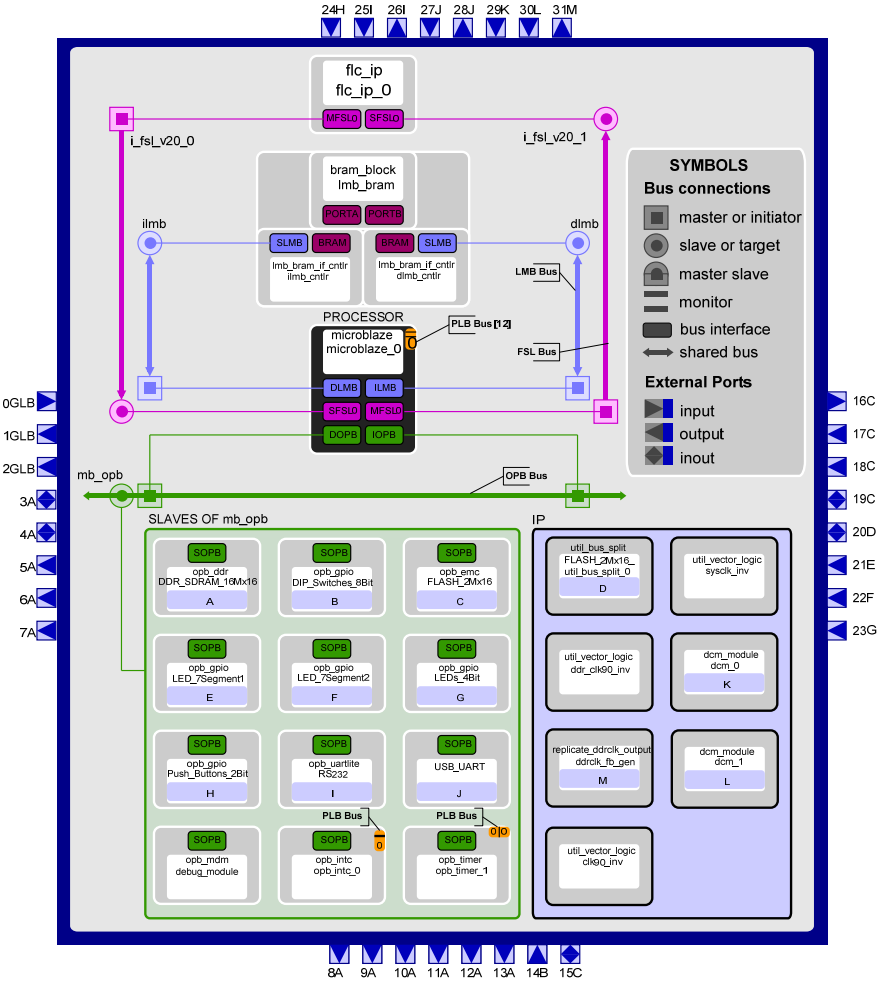
BASE	**HIGH**	**Μονάδα**
0x00000000	0x00007FFF	ilmb_cntlr
0x00000000	0x00007FFF	dlmb_cntlr
0x24000000	0x25FFFFFF	DDR_SDRAM_16Mx16
0x26000000	0x263FFFFF	FLASH_2Mx16
0x40000000	0x4000FFFF	Push_Buttons_2Bit
0x40020000	0x4002FFFF	LEDs_4Bit
0x40040000	0x4004FFFF	LED_7Segment2
0x40060000	0x4006FFFF	LED_7Segment1
0x40080000	0x4008FFFF	DIP_Switches_8Bit
0x40600000	0x4060FFFF	USB_UART
0x40620000	0x4062FFFF	RS232
0x41200000	0x4120FFFF	opb_intc_0

Πίνακας 8.4: Χάρτης μνήμης

Φυσικοί Πόροι	Χρήση	Διαθέσιμα	Χρήση %
BSCANs	1	1	100%
BUFGMUXs	6	8	75%
DCMs	2	4	50%
External IOBs	121	487	24%
LOCed IOBs	120	121	99%
MULT18X18s	11	32	34%
RAMB16s	16	32	50%
Slices	4021	13312	30%
SLICEMs	668	6656	10%
Total LUTs:	5,956	26,624	22%

Πίνακας 8.5: Φυσικοί πόροι που καταλαμβάνονται στο FPGA

Το διάγραμμα δομικών μονάδων του υλοποιημένου SoC καθώς και ένας πίνακας επεξήγησης των διαφορετικών συμβόλων που χρησιμοποιούνται δίνεται στο Σχήμα 8.11 που ακολουθεί παρακάτω.

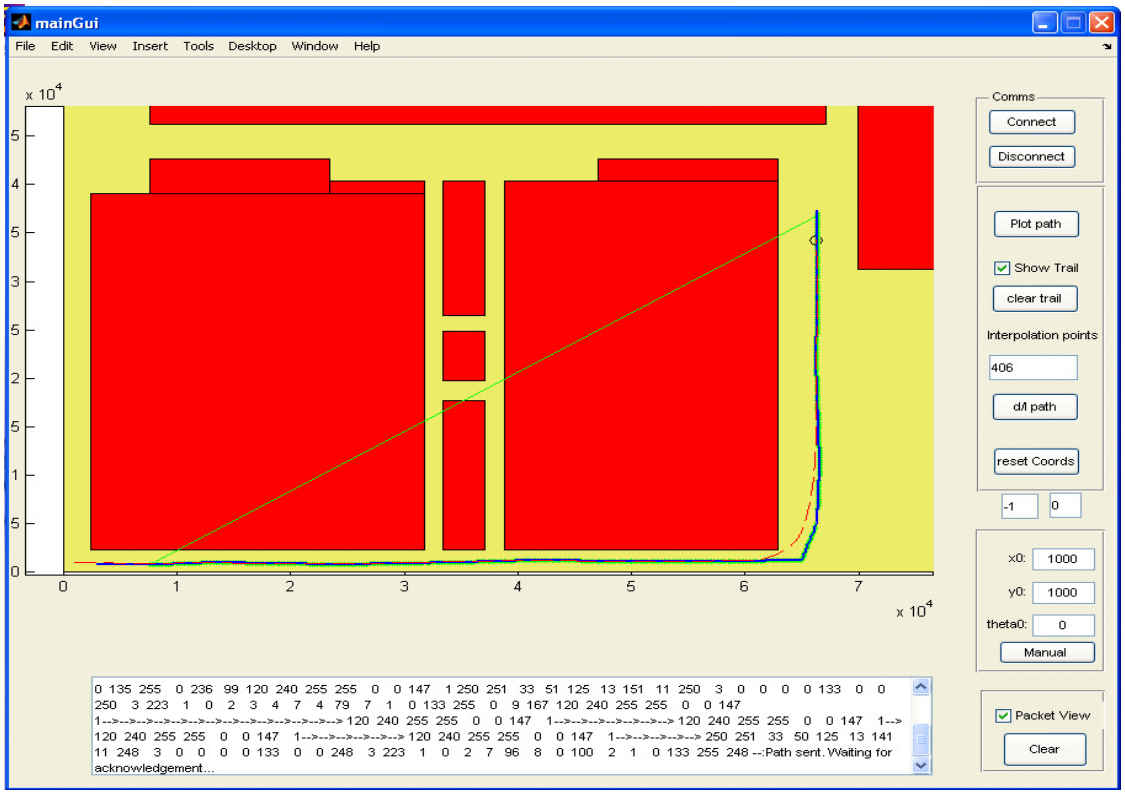


Σχήμα 8.11: Διάγραμμα δομικών μονάδων του SoC της εφαρμογής

8.8 Εφαρμογή MATLAB

Στο πλαίσιο της εφαρμογής που παρουσιάζουμε σε αυτό το κεφάλαιο, υλοποιήσαμε ένα πρόγραμμα σε MATLAB (βλ. Παράρτημα Α.2) με γραφική διεπαφή χρήστη (*Graphical User Interface* – GUI), για το σκοπό της επίβλεψης και της έναρξης (π.χ. διάφορες αρχικοποιήσεις) της λειτουργίας του συστήματος. Το πρόγραμμα συνδέεται με το FPGA μέσω USB σύνδεσης. Στη συνέχεια λαμβάνει και αναλύει τα δεδομένα από το SoC (κυρίως τα πακέτα που δεδομένων που έχει στείλει το ρομπότ στο SoC), και εξάγει τις πληροφορίες που αφορούν την οδομετρία. Η ίδια ρουτίνα που βρίσκεται στο SoC για τον εντοπισμό και τη διόρθωση υπερχείλησεων (από τους οπτικούς αποκωδικοποιητές του ρομπότ) χρησιμοποιείται και εδώ, όπως επίσης και η ρουτίνα για την τεχνική «spatial-window». Οι μετατροπές μεταβλητών από τοπικές σε παγκόσμιες επίσης υλοποιούνται από το πρόγραμμα. Το γραφικό περιβάλλον του προγράμματος απεικονίζει το χάρτη του κόσμου του ρομπότ σε παγκόσμιες συντεταγμένες, όπως φαίνεται στο Σχήμα 8.12.

Στην εφαρμογή που μελετάμε δεν υπάρχει αλγόριθμος σχεδιασμού πορείας (*path planning algorithm*), όποτε η πορεία που πρέπει να ακολουθήσει το ρομπότ σχεδιάζεται από το χρήστη στο GUI της εφαρμογής ως μια σειρά σημείων. Εν συνεχεία η εφαρμογή με τη χρήση ενός αλγορίθμου γραμμικής παρεμβολής (*linear interpolation*) παράγει όλα τα δείγματα σημείων της πορείας με σταθερό βήμα δειγματοληψίας Δs (η απόσταση μεταξύ δύο δειγμάτων της πορείας είναι σταθερή). Ο χρήστης έχει τη δυνατότητα να ορίσει τον αριθμό των σημείων παρεμβολής. Η ρουτίνα παρεμβολής επιλέχθηκε μεταξύ άλλων (πολυωνυμική, τρίτου βαθμού, και γραμμική) με γνώμονα τα πειράματα που έγιναν και η γραμμική παρεμβολή έδωσε τα καλύτερα αποτελέσματα στη συγκεκριμένη εφαρμογή.



Σχήμα 8.12: Η εφαρμογή MATLAB

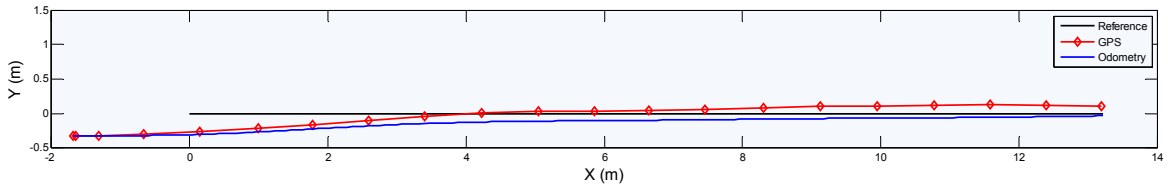
Το GUI της εφαρμογής απεικονίζει τη θέση του ρομπότ σε πραγματικό χρόνο μαζί με άλλες πληροφορίες που λαμβάνει από το SoC. Συγκεκριμένα, όταν το «spatial-window» είναι πρώτου βαθμού, δηλαδή λαμβάνεται υπόψη μόνο το κοντινότερο σημείο, το SoC στέλνει τις δύο υπολογισμένες εισόδους του ελεγκτή.

Η εικόνα του GUI της εφαρμογής που δίνεται στο Σχήμα 8.12 έχει παρθεί κατά τη διάρκεια πειράματος στον δεύτερο όροφο του κτηρίου Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Η στικτή γραμμή αντικατοπτρίζει την επιθυμητή πορεία, ενώ η διακεκομμένη γραμμή την πραγματική πορεία. Όλες οι μονάδες είναι σε χιλιοστά του μέτρου, mm.

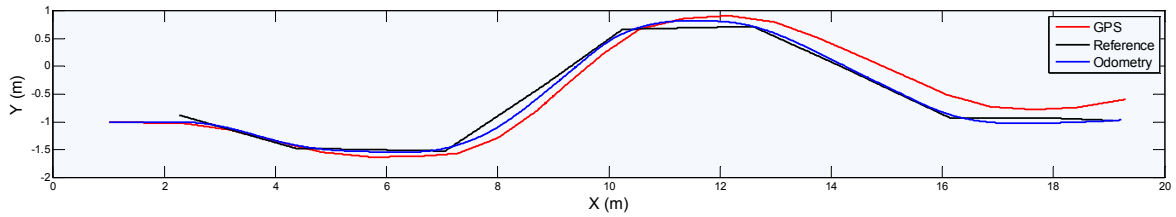
8.9 Πειράματα

Στην παράγραφο αυτή παρουσιάζουμε δύο πειράματα με το σύστημα του ρομπότ, που διεξήχθησαν στον χώρο του Εθνικού Μετσόβιου Πολυτεχνείου. Σκοπός των πειραμάτων ήταν να εκτιμήσουμε την απόδοση του συστήματος και πιο συγκεκριμένα του ασαφούς ελεγκτή παρακολούθησης πορείας. Τα πειράματα βασίζονται στην παρακολούθηση δυο προκαθορισμένων διαδρομών. Η πρώτη πορεία είναι ευθεία γραμμή (*straight line*) και η δεύτερη είναι τύπου «Σίγμα» (S–*shaped path*). Προκειμένου να λάβουμε την πραγματική πορεία του ρομπότ κατά τη διάρκεια των πειραμάτων, τοποθετήσαμε έναν διαφορικό δέκτη και την κεραία του (DGPS antenna) πάνω στο ρομπότ. Το διαφορικό GPS σύστημα είναι το μοντέλο Trimble 4700 GPS και τέθηκε σε λειτουργία κινηματικής παρακολούθησης (*Kinematic Survey mode*), στην οποία η πορεία που έχει διανύσει το ρομπότ υπολογίζεται μετά το τέλος του πειράματος. Η ανάλυση του DGPS σε αυτό τον τρόπο λειτουργίας έχει ακρίβεια ±1 cm+1 ppm για απόσταση έως 10 Km, ενώ τα δείγματα θέσης λαμβάνονται κάθε 1 δευτερόλεπτο.

Τα αποτελέσματα των πειραμάτων για την ευθεία πορεία και τη πορεία «Σίγμα» αντίστοιχα φαίνονται στα αποτυπώνονται στα διαγράμματα που ακολουθούν (βλ. Σχήμα 8.13 και Σχήμα 8.14).



Σχήμα 8.13: Πείραμα ευθείας πορείας. Διακρίνεται η πορεία αναφοράς (στικτή γραμμή), εκτίμηση της θέσης από την οδομετρία (διακεκομμένη γραμμή με παύλες) και η εκτίμηση του DGPS (διακεκομμένη γραμμή με τελείες)



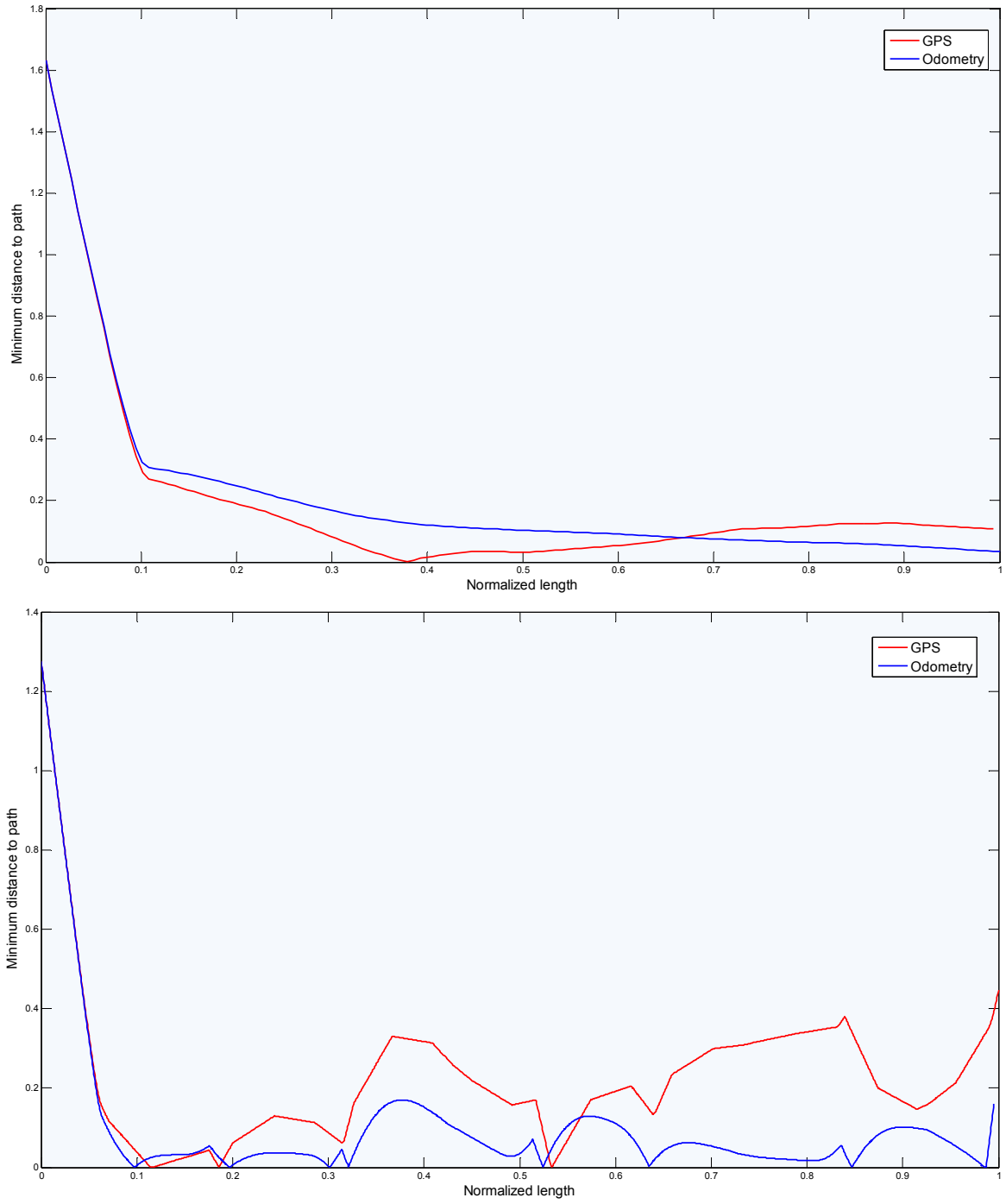
Σχήμα 8.14: Πείραμα «Σίγμα» πορείας. Διακρίνεται η πορεία αναφοράς (στικτή γραμμή), εκτίμηση της θέσης από την οδομετρία (διακεκομμένη γραμμή με παύλες) και η εκτίμηση του DGPS (διακεκομμένη γραμμή με τελείες)

Στο πείραμα της ευθείας πορείας το ρομπότ ακολούθησε μια ευθεία 25 μέτρων. Η αρχική θέση του ρομπότ δεν βρισκόταν πάνω στη πορεία. Το Σχήμα 8.13 δεν δείχνει όλη τη διάρκεια του πειράματος αλλά το κομμάτι εκείνο στο οποίο το DGPS έχει υψηλή ποιότητα εύρεσης θέσης (βαθμός ποιότητας Q=1) δεδομένου ότι για να αξιολογήσουμε τον ελεγκτή της εφαρμογής χρειαζόμαστε αξιόπιστες τιμές της θέσης του ρομπότ από το DGPS. Εδώ πρέπει να διευκρινίσουμε ότι η θέση από το DGPS δεν συγχέεται με τη θέση που χρησιμοποιεί ο εκλεκτής παρακολούθησης και προέρχεται από τα δεδομένα της οδομετρίας. Επομένως δεδομένα θέσης από το DGPS με Q>1 (1 είναι η καλύτερη και 6 η χειρότερη ποιότητα) απορρίφθηκαν.

Στο δεύτερο πείραμα της «Σίγμα» πορείας εφαρμόζονται οι ίδιες προδιαγραφές για το DGPS όπως και στο πρώτο πείραμα. Η «Σίγμα» πορεία έχει περίπου 25 μέτρα μήκος και όπως και στο πείραμα ευθείας πορείας όλα τα δεδομένα του DGPS με Q>1 απορρίφθηκαν. Αξίζει να σημειώσουμε ότι η «Σίγμα» πορεία δεν αποτελεί εφικτή πορεία αναφοράς μιας και η παράγωγος της καμπυλότητας είναι ασυνεχής στις κορυφές των πολυγώνων. Ωστόσο, αν η ασυνέχεια είναι μικρή το ρομπότ καταφέρνει να ακολουθήσει την πορεία με ακρίβεια. Προκειμένου να καταλήξουμε σε ένα τεκμηριωμένο συμπέρασμα για τα αποτελέσματα των πειραμάτων, υπολογίσαμε τις ελάχιστες αποστάσεις της πορείας από το GPS και την οδομετρία του ρομπότ, από την πορεία αναφοράς. Ωστόσο, επειδή οι ακολουθούμενες πορείες είναι ουσιαστικά ένα σύνολο από σημεία στο επίπεδο που προκύπτουν από την επίλυση των δεδομένων του GPS (η ανάλυση του DGPS είναι 1 στίγμα/s και η ταχύτητα του ρομπότ 1 m/s, οπότε προκύπτει 1 στίγμα/m, που είναι αρκετά σποραδικό), προβήκαμε σε αύξηση της δειγματοληψίας των πορειών χρησιμοποιώντας τη μέθοδο της γραμμικής παρεμβολής. Ακόμη, η απευθείας σύγκριση των επιλύσεων που προκύπτουν από το GPS και από την οδομετρία δεν είναι δυνατή διότι οι δειγματοληπτικοί χρόνοι των δύο δεν βρίσκονται σε άμεσο συγχρονισμό μεταξύ τους, επομένως δεν μπορούμε να συγκρίνουμε τις ελάχιστες αποστάσεις των δύο πορειών σε συγκεκριμένα χρονικά διαστήματα. Ως αποτέλεσμα του περιορισμού αυτού η γραφική παράσταση που φαίνεται στο Σχήμα 8.15 δίνετε ως η απόσταση σε αντιδιαστολή με το κανονικοποιημένο μήκος της κάθε πορείας αναφορικά με το μέγιστο μήκος και των δύο.

Μπορούμε να παρατηρούμε, ότι όταν το ρομπότ βρίσκεται κοντά στο ίχνος της πορείας, η απόσταση είναι κάτω από 40 cm (ή 20 cm για την παρακολούθηση ευθείας). Επιπλέον, η εκτίμηση θέσης από την οδομετρία του ρομπότ είναι πολύ κοντά στην πορεία. Αυτό σημαίνει ότι αν χρησιμοποιηθούν δεδομένα DGPS μεγαλύτερης ακρίβειας, σε πραγματικό χρόνο (*Real-Time Kinematic* DGPS *data feed*) ως είσοδος θέσης (αντί της οδομετρίας του ρομπότ ή και σε συνδυασμό) στον ελεγκτή, τότε αυτός θα μπορούσε να αποδώσει πολύ καλύτερα.

Η τελευταία παρατήρηση θα μπορούσε να προταθεί ως μελλοντική επέκταση της εφαρμογής. Μελλοντικά επίσης, θα ήταν εφικτό να τροφοδοτήσουμε το FPGA με περισσότερα δεδομένα από διαφορετικούς αισθητήρες ούτως ώστε να έχουμε περισσότερες εξόδους ελέγχου.



Σχήμα 8.15: Η ελάχιστη απόσταση (μέτρα, m) της πορείας αναφοράς που προκύπτει από την επίλυση των δεδομένων του GPS και της οδομετρίας σε αντιδιαστολή με το κανονικοποιημένο μήκος, για τις δύο πορείες (ευθεία, «Σίγμα») αντίστοιχα

8.10 Συμπεράσματα

Στο κεφάλαιο αυτό παρουσιάσαμε μια ρομποτική πλατφόρμα βασισμένη σε SoC, για το πρόβλημα της παρακολούθησης πορείας σε μη-ολονομικά (*non–holonomic*) κινητά ρομπότ. Η συνολική καθυστέρηση του ελέγχου είναι πολύ μικρή παρότι περιορίζεται από την απόκριση του ελεγχόμενου συστήματος (ρομπότ). Οι εξομοιώσεις και τα πειράματα που έγιναν έδειξαν ότι η συνολική επίδοση του ελεγκτή είναι ικανοποιητική παρ'όλους τους περιορισμούς που εμφανίζει το πραγματικό σύστημα, όπως για παράδειγμα, η κβάντιση των διαθέσιμων εντολών διεύθυνσης και η «νεκρή περιοχή» της διεύθυνσης. Η επίδοση του ελεγκτή οφείλεται στη μεγάλη σθεναρότητα του αλγορίθμου παρακολούθησης σε συνδυασμό με την «ήρεμη» συμπεριφορά κίνησης που προσδίδει η τεχνική «spatial-window» όταν εισάγεται στον βρόχο ελέγχου.

Κεφάλαιο 9

Σχεδίαση και Υλοποίηση Γενετικού Αλγορίθμου

9.1 Εισαγωγή

Οι γενετικοί αλγόριθμοι (ΓΑ) είναι μια τεχνική τυχαίας αναζήτησης που επεξεργάζεται έναν πληθυσμό από λύσεις και βασίζεται στο νόμο της φυσικής επιλογής του Δαρβίνου. Ωστόσο, η σύγκλισή τους προς το βέλτιστο μπορεί να είναι υπερβολικά αργή για δύσκολα και περίπλοκα προβλήματα βελτιστοποίησης, με αποτέλεσμα να γίνεται δύσκολη η χρήση τους σε εφαρμογές πραγματικού χρόνου (*real–time applications*). Στην παρούσα εργασία ένας πλήρως παραμετρικός γενετικός αλγόριθμος έχει σχεδιαστεί και υλοποιηθεί σε υλικό (*hardware*), πετυχαίνοντας με αυτό τον τρόπο εντυπωσιακές επιταχύνσεις της ταχύτητας σύγκλισης συγκρινόμενος με τον ίδιο γενετικό αλγόριθμο ανεπτυγμένο σε λογισμικό. Ο όρος «πλήρως παραμετρικός» αναφέρεται στην παραμετροποίηση των βασικών χαρακτηριστικών του γενετικού αλγορίθμου, όπως το μέγεθος του πληθυσμού, η ανάλυση σε bit του κάθε γονιδίου, η πιθανότητα μετάλλαξης και η ανάλυσή της σε bit, οι μέθοδοι διασταύρωσης και μετάλλαξης που υιοθετούνται, ο μέγιστος αριθμός γενεών, ο αριθμός των ελίτ απογόνων κάθε γενιάς, κ.α.. Η αρχιτεκτονική που παρουσιάζεται στην παρούσα εργασία υλοποιήθηκε σε ένα ψηφιακό κύκλωμα προγραμματιζόμενης λογικής (FPGA) με τη χρήση της γλώσσας περιγραφής υλικού VHDL και προχωρημένης τεχνολογίας εργαλεία σύνθεσης, τοποθέτησης και διασύνδεσης του συστήματός μας στο υλικό. Ο υλοποιημένος σε υλικό γενετικός αλγόριθμος που παρουσιάζεται εδώ επιτυγχάνει συχνότητα λειτουργίας ίση με 92 MHz (10.8 ns) και η απόδοση του αξιολογείται τόσο με τη χρήση του γνωστού προβλήματος του Πλανόδιου Πωλητή (*Traveling Salesman Problem* – TSP) όσο και με ειδικές συναρτήσεις σύγκρισης (*benchmarking functions*) [94][95].

Οι ΓΑ βασίζονται στην έννοια του πληθυσμού των ατόμων (χρωμοσωμάτων) στα οποία εφαρμόζονται γενετικοί τελεστές (πράξεις) της διασταύρωσης (crossover), της μετάλλαξης (*mutation*), της επιλογής (*selection*) και του ελιτισμού (*elitism*). Οι ΓΑ λειτουργούν

υπακούοντας στην αρχή της φυσικής επιλογής (natural selection) του Δαρβίνου. Έχουν επιτυχώς εφαρμοστεί σε διάφορα δύσκολα προβλήματα βελτιστοποίησης, λόγω της μεγάλης ενδογενούς ευελιξίας τους και της ελευθερίας που εμφανίζουν στην επιλογή μιας επιθυμητής βέλτιστης λύση σύμφωνα με τις προδιαγραφές σχεδίασης του προβλήματος [33][96].

Ωστόσο, τα πιο σοβαρά μειονεκτήματα των ΓΑ που έχουν αναπτυχθεί σε λογισμικό είναι οι απαιτήσεις τους σε τεράστια ποσά μνήμης και χρόνου εκτέλεσης. Λόγω των περιορισμών αυτών μια πλειάδα γενετικών αλγορίθμων υλοποιημένων σε υλικό έχουν αναπτυχθεί, κυρίως την τελευταία δεκαετία [97][98][99][100][101][102], εκμεταλλευόμενοι την ταχεία ανάπτυξη που παρατηρήθηκε στην τεχνολογία των FPGA τα τελευταία χρόνια. Αποτέλεσμα αυτού ήταν η επίτευξη εντυπωσιακών επιταχύνσεων (speed–up) του χρόνου εκτέλεσης των γενετικών αλγορίθμων.

Το κεφάλαιο αυτό αναλύει το σχεδιασμό και την υλοποίηση σε υλικό ενός πλήρως παραμετρικού γενετικού αλγορίθμου σε FPGA chip. Οι γενετικοί τελεστές που εφαρμόστηκαν στα γονίδια του πληθυσμού είναι η διασταύρωση, η μετάλλαξη και ο ελιτισμός, των οποίων η μέθοδος που κάθε φορά χρησιμοποιείται επιλέγεται παραμετρικά. Ο υλοποιημένος αλγόριθμος για την εφαρμογή της φυσικής επιλογής είναι ο «Αλγόριθμος Επιλογής Ρουλέτας» (*Roulette Wheel Selection Algorithm*). Το FPGA chip που χρησιμοποιήθηκε είναι το Xilinx XC3S1500–4FG676C Spartan-3 FPGA [44]. Επιπρόσθετα ο σχεδιασμένος ΓΑ υλοποιήθηκε και σε λογισμικό στην πλατφόρμα MATLAB με στόχο την παραγωγή των απαραίτητων διανυσμάτων ελέγχου (test vectors) για τον έλεγχο των αποτελεσμάτων του ΓΑ που υλοποιήθηκε στο FPGA, όπως επίσης και για τη δυνατότητα σύγκρισης μεταξύ της υλοποίησής του στο υλικό και της υλοποίησής του στο λογισμικό. Ο ΓΑ αξιολογήθηκε με τη χρήση benchmarking functions που χρησιμοποιούνται εκτενώς στη βιβλιογραφία για αυτό τον σκοπό. Επιπλέον ο ΓΑ έλυσε επιτυχώς το πρόβλημα του Πλανόδιου Πωλητή αφού πρώτα τροποποιήθηκε κατάλληλα και υλοποιήθηκε ξανά στο FPGA.

9.2 Επισκόπηση του Συστήματος

Η σύνθεση της αρχιτεκτονικής δομής του ΓΑ αποτελείται από τις παρακάτω έξι υπομονάδες (modules),

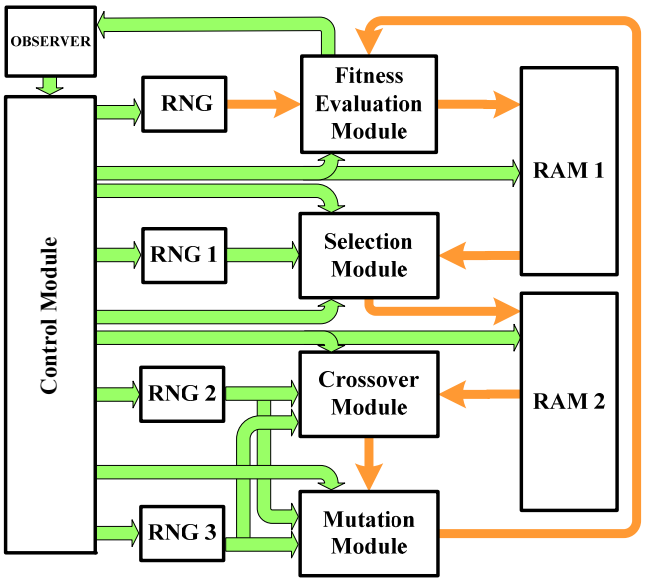
- τμήμα ελέγχου (*control module*)
- τμήμα επιλογής (*selection module*)
- τμήμα διασταύρωσης (*crossover module*)
- τμήμα μετάλλαξης (*mutation module*)
- τμήμα εκτίμησης της τιμής προσαρμογής (*fitness evaluation module*)
- τμήμα επιτήρησης (*observer module*)

Πιο αναλυτικά, το τμήμα ελέγχου υλοποιεί μια μηχανή καταστάσεων (*state machine*) τύπου Mealy [103][59][104], που η βασική λειτουργία της είναι η τροφοδότηση των υπόλοιπων τμημάτων με τα απαραίτητα σήματα ελέγχου και επιπλέον φροντίζει για τη συγχρονισμένη εκτέλεσή τους. Το τμήμα επιλογής που χρησιμοποιεί τη μέθοδο επιλογής ρουλέτας (*roulette selection method*) επιλέγει γονίδια από τον πληθυσμό (γονείς), τα οποία επεξεργάζονται

γενετικά ούτως ώστε να παράγουν απογόνους που με τη σειρά τους στη συνέχεια θα δημιουργήσουν νέα γενιά γονιδίων. Στη συνέχεια, τα τμήματα διασταύρωσης και μετάλλαξης εφαρμόζουν τις αντίστοιχες γενετικές πράξεις στους επιλεγμένους γονείς. Το τμήμα εκτίμησης της τιμής προσαρμογής υπολογίζει την τιμή προσαρμογής του απογόνου που έχει παραχθεί από τα προηγούμενα τμήματα ενώ ταυτόχρονα εφαρμόζει και την πράξη του ελιτισμού, έτσι ώστε να επιλεχθούν τα ελίτ γονίδια της επόμενης γενιάς. Τελικά, το τμήμα επιτήρησης ελέγχει αν ικανοποιούνται τα κριτήρια τερματισμού του ΓΑ, όπως το μέγιστο όριο της τιμής προσαρμογής και το μέγιστο όριο των γενεών εκτέλεσης και ανάλογα συνεχίζει ή τερματίζει την εκτέλεση του ΓΑ.

Επιπρόσθετα, τέσσερις γεννήτριες τυχαίων αριθμών (*Random Number Generators* – RNG) αναλαμβάνουν να παράγουν τις απαραίτητες για τον αλγόριθμο τυχαίες ακολουθίες αριθμών και επιπλέον δύο μνήμες RAM αποθηκεύουν, η πρώτη την τρέχουσα γενιά και η δεύτερη τα επιλεγμένα γονίδια (γονείς) από το τμήμα επιλογής.

Η αρχιτεκτονική δομή υψηλού επιπέδου (*high level architectural structure*) του συστήματος φαίνεται στο Σχήμα 9.1.



Σχήμα 9.1: Αρχιτεκτονική δομή υψηλού επιπέδου του ΓΑ

9.3 Υλοποίηση ΓΑ σε Υλικό

Η ενότητα αυτή περιγράφει και εξηγεί αναλυτικά τα επιμέρους ιεραρχικά μπλοκ της μελετώμενης αρχιτεκτονικής του ΓΑ. Στις παρακάτω υπό-ενότητες περιγράφονται και αναλύονται τα διάφορα θεμελιώδη υποσυστήματα που αποτελούν το ΓΑ που παρουσιάζεται σε αυτό το κεφάλαιο.

9.3.1 Χαρακτηριστικά ΓΑ

Η καινοτομία του ΓΑ αλγορίθμου που αναλύεται στο παρόν κεφάλαιο είναι ότι το σύστημα που υλοποιήθηκε είναι πλήρως παραμετρικό, που σημαίνει ότι οι τιμές των διαφόρων σταθερών που χρησιμοποιεί ο ΓΑ (π.χ., μέγιστο όριο της τιμής προσαρμογής, μέγιστο όριο γενεών εκτέλεσης, μέγεθος του πληθυσμού, κ.ά.) δίνονται στο σύστημα κάνοντας χρήση generics στον VHDL κώδικα. Η παραμετροποίηση του πυρήνα (*core*) του ΓΑ έχει ως πλεονέκτημα να μην χρειάζεται η τροποποίηση του VHDL κώδικα όταν χρειάζεται να αλλάξει κάποια σταθερά (βλ. Παράρτημα Α.4). Επίσης, ο πυρήνας διαθέτει τρεις διαφορετικές μεθόδους για τις γενετικές πράξεις (μετάλλαξης και διασταύρωσης) και μέσω δύο εισόδων είναι δυνατή η επιλογή της μεθόδου που θα χρησιμοποιηθεί (σε πραγματικό χρόνο) σε κάθε εκτέλεση του ΓΑ.

Οι παράμετροι του ΓΑ δίνονται στον Πίνακα 9.1 παρακάτω.

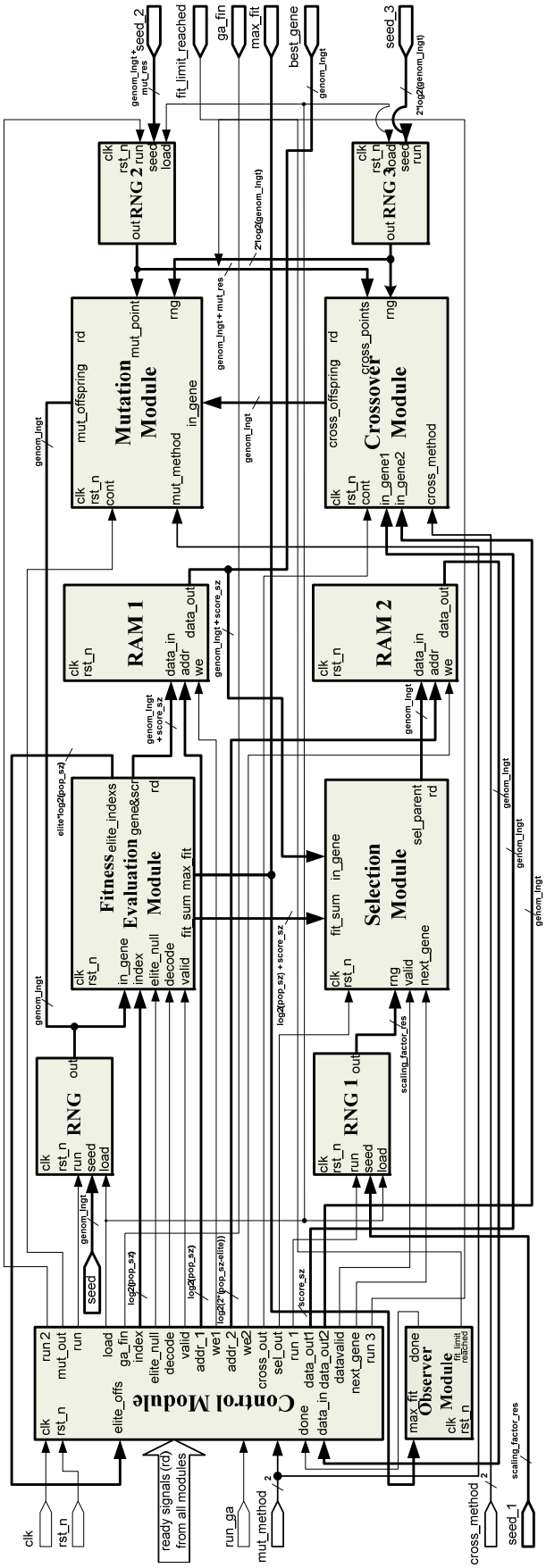
Όνομα παραμέτρου	**Περιγραφή**	**Τιμή**
genom_lngt	Μήκος γονιδίου σε bits	16
score_sz	Ανάλυση (σε bit) της τιμής προσαρμογής	16
pop_sz	Μέγεθος πληθυσμού	32
scaling_factor_res	Ανάλυση (σε bit) του τυχαίου αριθμού που χρησιμοποιείται στον Αλγόριθμο Ρουλέτα	
elite	Αριθμός ελίτ απογόνων	2
mr	Ρυθμός μετάλλαξης	80
mut_res	Ανάλυση (σε bit) του τυχαίου αριθμού που χρησιμοποιείται κατά τη διαδικασία μετάλλαξης	8
fit_limit	Όριο τιμής προσαρμογής	Τιμή εξαρτώμενη από το πρόβλημα
max_gen	Μέγιστος αριθμός γενεών	1000
inv_type	Τύπος αντιστροφής της τιμής προσαρμογής (χρήση στο TSP) 1: διαίρεση 2: αφαίρεση	2

Πίνακας 9.1: Χαρακτηριστικά ΓΑ

9.3.2 Αρχιτεκτονική ΓΑ

Η αρχιτεκτονική σχεδίαση του ΓΑ είναι δομημένη σε ξεχωριστά ανεξάρτητα μπλοκ, κάθε ένα από τα οποία εκτελεί μία συγκεκριμένη εργασία, ενώ όλα συντονίζονται μέσω του μπλοκ ελέγχου. Κάθε μπλοκ με τη σειρά του ειδοποιεί την υπομονάδα ελέγχου για την κατάστασή του, στέλνοντας του σήματα επικοινωνίας (*handshake signals*), πχ. σήματα ready_out (rd), κ.ά.

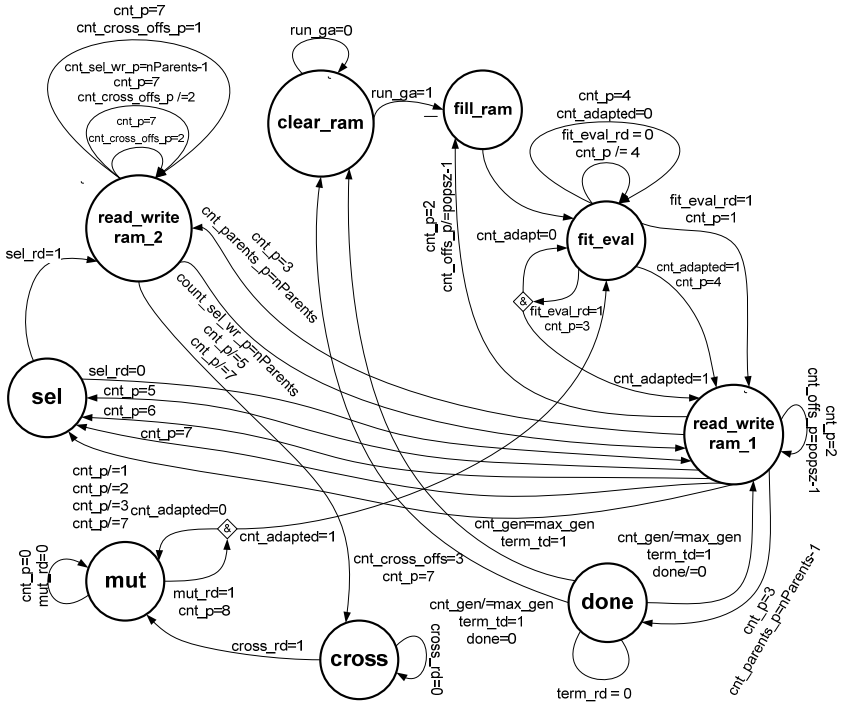
Τα διάφορα σήματα και οι δίαυλοι δεδομένων διακρίνονται στο σχηματικό διάγραμμα (βλ. Σχήμα 9.2) με λεπτές και χοντρές γραμμές αντίστοιχα.



Σχήμα 9.2: Αρχιτεκτονική σχεδίαση του πυρήνα ΓΑ

9.3.2.1 Υπομονάδα Ελέγχου

Για να διασφαλιστεί, ελεγχθεί και συγχρονιστεί η σειρά εκτέλεσης των διαφόρων υπομονάδων που έχουν δημιουργηθεί, υλοποιήθηκε για αυτό το σκοπό ένα υποσύστημα ελέγχου. Το μπλοκ αυτό, παράγει και τροφοδοτεί όλες τις άλλες υπομονάδες με τα απαραίτητα σήματα ελέγχου χρησιμοποιώντας μια Mealy μηχανή εννέα καταστάσεων (βλ. Σχήμα 9.3) [104].



Σχήμα 9.3: Μηχανή καταστάσεων ελέγχου

Η εργασία που εκτελείται από καθεμία από τις εννέα καταστάσεις (πχ. clear_ram) περιγράφεται στον πίνακα που ακολουθεί (βλ. Πίνακα 9.2).

Καταστάσεις	**Λειτουργία**
clear_ram	Σβήσιμο της RAM 1 για την τρέχουσα γενιάς
fill_ram	Γέμισμα της RAM 1 με ένα τυχαίο γονίδιο, ώστε να δημιουργηθεί ο αρχικός πληθυσμός
fit_eval	Εκτίμηση της τιμής προσαρμογής ενός γονιδίου και παραγωγή των ελίτ απογόνων
sel	Επιλογή ενός γονέα από τα γονίδια της τρέχουσας γενιάς
cross	Εφαρμογή της γενετικής πράξης της διασταύρωσης
mut	Εφαρμογή της γενετικής πράξης της μετάλλαξης
done	Έλεγχος κριτηρίων τερματισμού
read_write_ram_1	Ανάγνωση/Εγγραφή στη μνήμη της τρέχουσας γενιάς (RAM 1)
read_write_ram_2	Ανάγνωση/Εγγραφή στη μνήμη των επιλεγμένων γονέων (RAM 2)

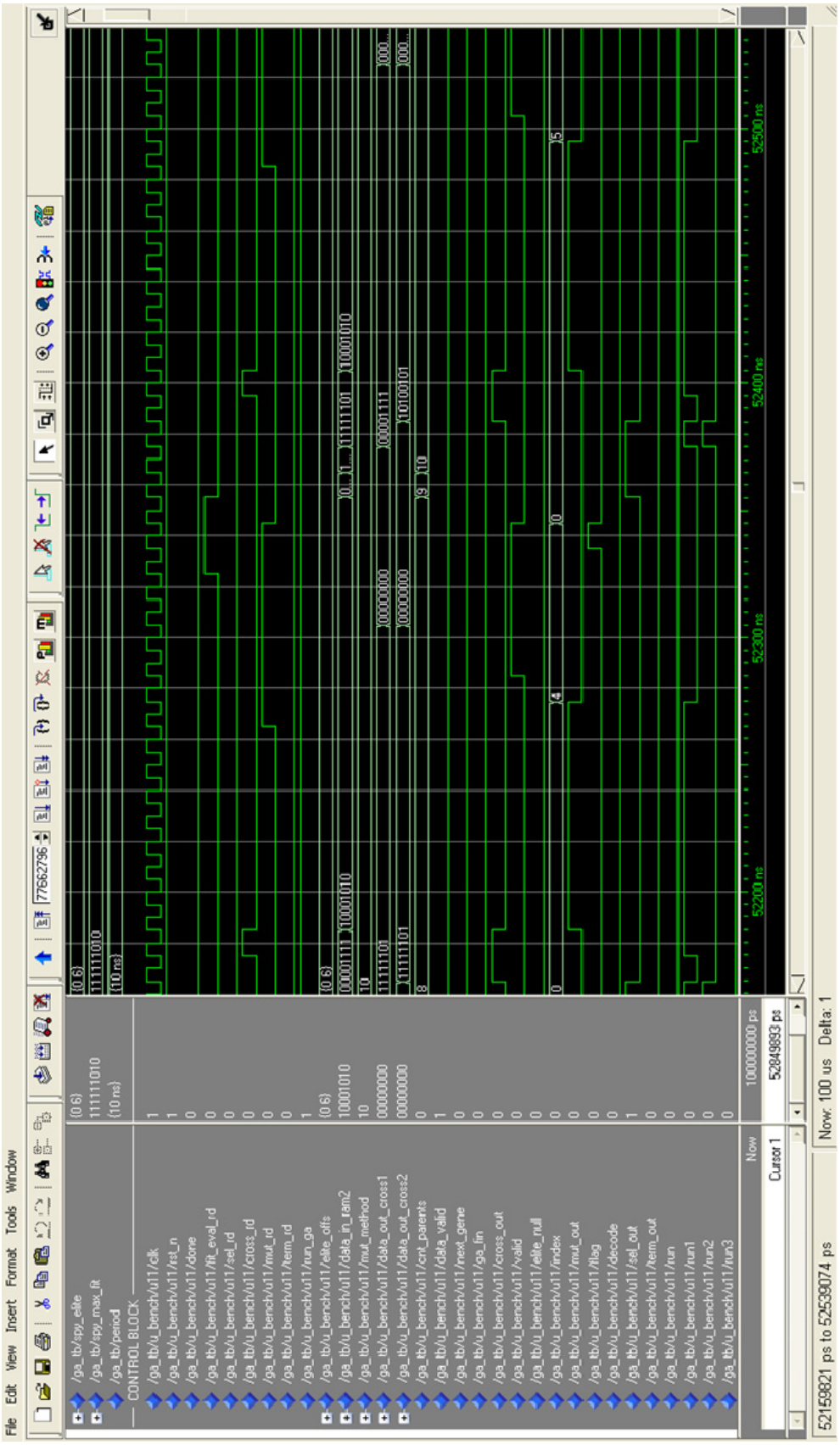
Πίνακας 9.2: Καταστάσεις της υλοποιημένης Mealy μηχανής καταστάσεων

Στον Πίνακα 9.3 επεξηγούνται τα βασικότερα σήματα της υπομονάδας ελέγχου. Τα σήματα διακοπών σημειώνονται ως IRQ (*Interrupt Request*).

Σήμα	**Τύπος**	**Ανάλυση (σε bit)**	**Περιγραφή λειτουργίας**
term_rd	είσοδος	1	ready out IRQ σήμα του συστήματος επιτήρησης
fit_eval_rd	είσοδος	1	ready out IRQ σήμα του συστήματος εκτίμησης της τιμής προσαρμογής
sel_rd	είσοδος	1	ready out IRQ σήμα του συστήματος επιλογής
cross_rd	είσοδος	1	ready out IRQ σήμα του συστήματος διασταύρωσης
mut_rd	είσοδος	1	ready out IRQ σήμα του συστήματος μετάλλαξης
done	είσοδος	1	IRQ σήμα που αν είναι 1 τότε ο αλγόριθμος τερματίζεται γιατί το κριτήριο τερματισμού ικανοποιήθηκε
run_ga	είσοδος	1	IRQ σήμα που αν είναι 1 τότε ο αλγόριθμος θα εκτελεστεί, διαφορετικά το κύκλωμα παραμένει αδρανές
mut_method	είσοδος	2	Ανάλογα με τη τιμή (00,01,10) επιλέγεται μία από τις τρεις μεθόδους μετάλλαξης που έχει υλοποιηθεί
cross_out	έξοδος	1	IRQ σήμα ενεργοποίησης του υποσυστήματος διασταύρωσης
addr_1_c	έξοδος	32	Διεύθυνση ανάγνωσης/εγγραφής από/στη RAM 1
addr_2_c	έξοδος	32	Διεύθυνση ανάγνωσης/εγγραφής από/στη RAM 2
we1_c	έξοδος	1	Σήμα write enable (we) της RAM 1
we2_c	έξοδος	1	Σήμα write enable (we) της RAM 1
valid	έξοδος	1	IRQ σήμα που αν είναι 1 ειδοποιεί το υποσύστημα εκτίμησης της τιμής προσαρμογής ότι τα δεδομένα στην είσοδό του είναι έγκυρα
mut_out	έξοδος	1	IRQ σήμα ενεργοποίησης του υποσυστήματος μετάλλαξης
sel_out	έξοδος	1	IRQ σήμα ενεργοποίησης του υποσυστήματος επιλογής
term_out	έξοδος	1	IRQ σήμα ενεργοποίησης του υποσυστήματος επιτήρησης
ga_fin	έξοδος	1	IRQ σήμα που όταν είναι 1 ειδοποιεί ότι ο αλγόριθμος τελείωσε
run, run1, run2, run3	έξοδοι	1	IRQ σήματα ενεργοποίησης των 4 γεννητριών τυχαίων αριθμών
load	έξοδος	1	IRQ σήμα φόρτωσης νέου σπόρου (seed) στις γεννήτριες τυχαίων αριθμών

Πίνακας 9.3: Επεξήγηση των σημάτων της υπομονάδας ελέγχου

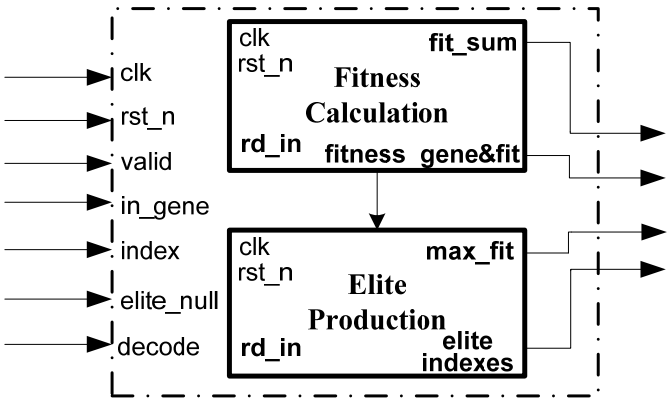
Στο Σχήμα 9.4 που ακολουθεί φαίνεται ένα τυπικό παράθυρο προσομοίωσης του εργαλείου Modelsim της εταιρίας Mentor Graphics και διακρίνονται ορισμένα σήματα της υπομονάδας ελέγχου.



Σχήμα 9.4: Προσομοίωση σε RTL επίπεδο της υπομονάδας ελέγχου

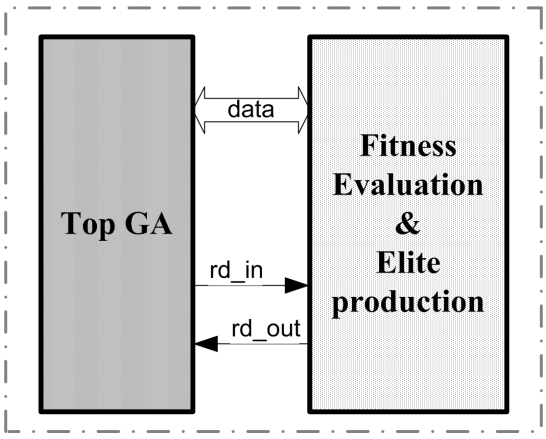
9.3.2.2 Υπομονάδα Εκτίμησης της τιμής προσαρμογής

Στην παράγραφο αυτή περιγράφεται το υποσύστημα εκτίμησης της τιμής προσαρμογής (*fitness evaluation*), το οποίο εκτελείται κάθε φορά που δημιουργείται ένας νέος πληθυσμός ατόμων/γονιδίων. Η συγκεκριμένη υπομονάδα έχει διπλή λειτουργία, πρώτον, υπολογίζει την τιμή προσαρμογής (*fitness calculation*) του κάθε γονιδίου κάνοντας χρήση της δοσμένης συνάρτησης προσαρμογής και δεύτερον, εφαρμόζει ελιτισμό (*elite production*) στα γονίδια της τρέχουσας γενιάς, παράγοντας με αυτό τον τρόπο τα ελίτ γονίδια της επόμενης γενιάς. Όπως είναι φανερό, το υποσύστημα αυτό αποτελείται δομικά από δύο άλλα μικρότερα μπλοκ, όπου το ένα υλοποιεί τον υπολογισμό της τιμής προσαρμογής ενώ το άλλο υλοποιεί την πράξη του ελιτισμού (βλ. Σχήμα 9.5), ενώ ταυτόχρονα είναι εξωτερικά συνδεδεμένο με την υπόλοιπη αρχιτεκτονική (βλ. Σχήμα 9.6).



Σχήμα 9.5: Αρχιτεκτονική δομή της υπομονάδας εκτίμησης της τιμής προσαρμογής

Το πλεονέκτημα της συνδεσμολογίας αυτής είναι ότι έτσι μας δίνετε η δυνατότητα να εισάγουμε οποιαδήποτε συνάρτηση προσαρμογής χωρίς αυτό να επιφέρει αλλαγές στην υπόλοιπη σχεδίαση του πυρήνα.



Σχήμα 9.6: Συνολική δομή του ΓΑ: Συνδεσμολογία υπομονάδας εκτίμησης τιμής προσαρμογής με τον υπόλοιπο πυρήνα του ΓΑ

Επιπρόσθετα, η υπομονάδα αυτή παράγει το άθροισμα των τιμών προσαρμογής όλων των γονιδίων, τη μέγιστη τιμή προσαρμογής της τρέχουσας γενιάς αλλά και τους δείκτες στη μνήμη RAM των ελίτ γονιδίων. Ο αριθμός των ελίτ γονιδίων θέτεται παραμετρικά. Ένα

γονίδιο για να γίνει ελίτ απόγονος για την επόμενη γενιά (δηλαδή να επιζήσει στην επόμενη γενιά) πρέπει να έχει την υψηλότερη τιμή προσαρμογής από τα υπόλοιπα γονίδια. Ακολουθεί ο Πίνακας 9.4 επεξήγησης των βασικότερων σημάτων της υπομονάδας εκτίμησης της τιμής προσαρμογής.

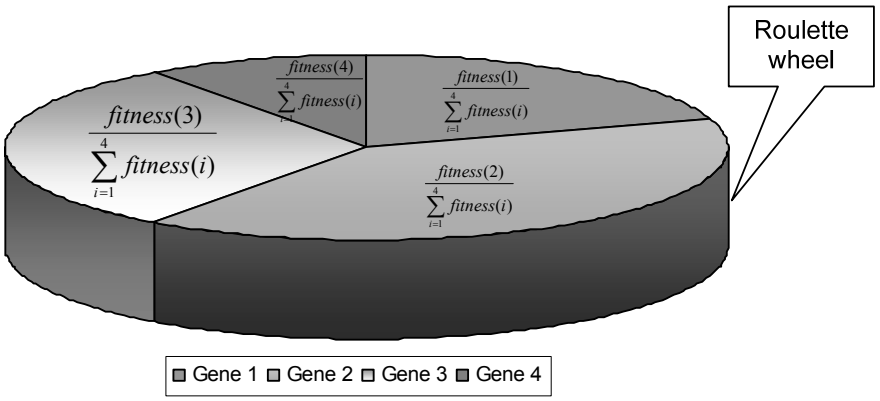
Σήμα	**Τύπος**	**Ανάλυση (σε bit)**	**Περιγραφή λειτουργίας**
valid	είσοδος	1	IRQ σήμα που ειδοποιεί ότι τα δεδομένα στην είσοδο της υπομονάδας είναι έγκυρα
in_genes	είσοδος	$2 \times genom\_lngt$	Γονίδια εισόδου
index	είσοδος	32	Δείκτης γονιδίου στη μνήμη RAM 1
gene_score	έξοδος	$genom\_lngt + score\_sz$	Δυαδικό διάνυσμα Γονιδίου και τιμής προσαρμογής του για εγγραφή στη μνήμη RAM 1
elite_offs	έξοδος	$elite \times 32$	Δείκτες στη μνήμη RAM 1 των ελίτ απογόνων
fit_sum	έξοδος	$score\_sz + \log_2(pop\_sz)$	Άθροισμα τιμών προσαρμογής όλων των γονιδίων της τρέχουσας γενιάς που τροφοδοτείται στο υποσύστημα επιλογής
rd	έξοδος	1	ready out IRQ σήμα τερματισμού λειτουργίας της υπομονάδας

Πίνακας 9.4: Επεξήγηση των σημάτων της υπομονάδας εκτίμησης της τιμής προσαρμογής

9.3.2.3 Υπομονάδα Επιλογής

Η υπομονάδα επιλογής λειτουργεί μετά το τέλος της εκτίμησης των τιμών προσαρμογής των γονιδίων της τρέχουσας γενιάς και ελέγχεται από το μπλοκ ελέγχου. Έχει πρόσβαση και στις δύο RAM του συστήματος και υλοποιεί το μηχανισμό επιλογής «Ρουλέτα» ή "Roulette Wheel Selection" - RWS [31][105]. Η μέθοδος επιλογής καθώς και ο αλγόριθμος RWS σε μορφή ψευδό-κώδικα δίνονται στο Σχήμα 9.7(α) και (β) αντίστοιχα.

Αρχικά η υπομονάδα επιλογής λαμβάνει ένα τυχαίο αριθμό *r* που έχει παραχθεί από τη γεννήτρια παραγωγής τυχαίων αριθμών (RNG). Το μήκος του αριθμού αυτού τίθεται μέσω της παραμέτρου *scaling_factor_res* και παριστάνει ένα δεκαδικό αριθμό μικρότερο της μονάδας. Ο αριθμός αυτός πολλαπλασιάζετε με το άθροισμα των τιμών προσαρμογής και εφαρμόζεται ο αλγόριθμος επιλογής RWS που τελικά επιλέγει ένα γονίδιο (γονέας) και εγγράφεται στη μνήμη RAM 2. Η διαδικασία επιλογής γονέων συνεχίζεται με τον ίδιο τρόπο μέχρις ότου το πλήθος των επιλεγμένων γονέων να επαρκεί ώστε να δημιουργηθεί μια νέα γενιά με τον ίδιο αριθμό γονέων με την τρέχουσα γενιά. Διευκρινίζεται ότι ο συγκεκριμένος αλγόριθμος επιλογής στηρίζεται στην τιμή προσαρμογής κάθε γονιδίου και επομένως τα γονίδια με μεγαλύτερη τιμή προσαρμογής είναι πιθανότερο να επιλεγούν ως γονείς περισσότερες φορές.



(α)

```
Read sum of fitnesses (fit_sum)
while not selected the desired number of parents
    {
    Set accumulative sum of fitnesses equal to zero (cum_sum = 0);
    Scale down fit_sum by a random number r ϵ [0,1] (scaled_fit_sum = r·fit_sum);
    Read both 1st gene and its fitness of the current population (gene1, fit1);
    Set cum_sum equal to fit1 (cum_sum = fit1);

    while cum_sum < scaled_fit_sum
        {
        Read next gene and its fitness from the population (genei fiti, i ϵ [2,3,...popsz]);
        Accumulate the new fitness into cum_sum (cum_sum = cum_sum+ fiti);
        i=i+1;
        }
    Set the selected parent equal to the last read gene (parent=genei)
    }
```

(β)

Σχήμα 9.7: Μέθοδος επιλογής «Ρουλέτα» (α) Αλγόριθμος (β) Ψευδό-κώδικας

Στον Πίνακα 9.5 που ακολουθεί αναφέρονται μερικά από τα σήματα της υπομονάδας επιλογής.

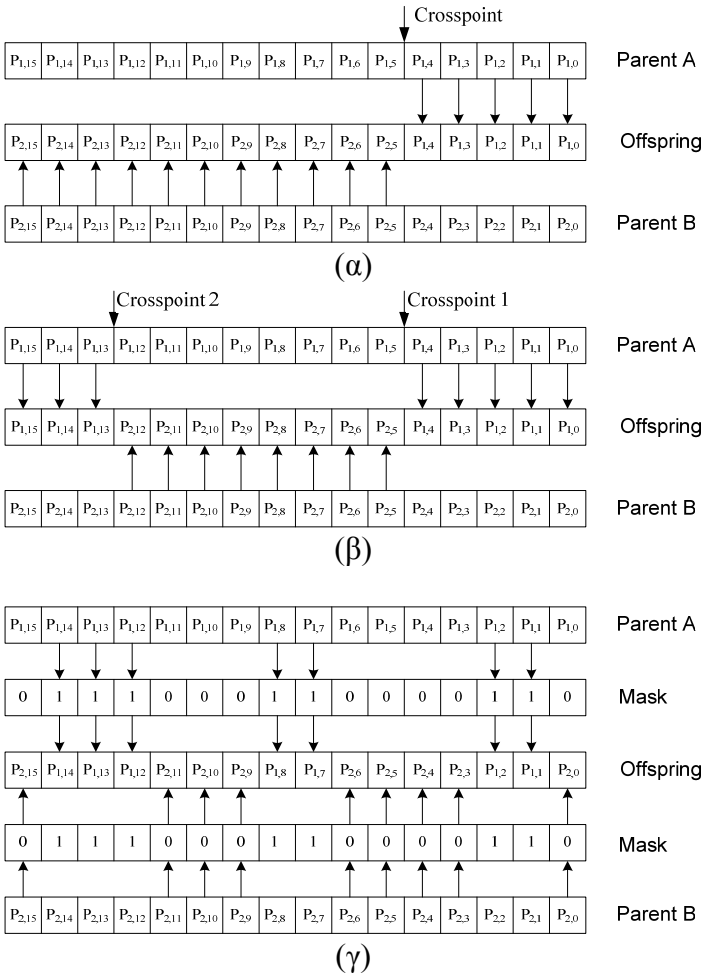
Σήμα	**Τύπος**	**Ανάλυση (σε bit)**	**Περιγραφή λειτουργίας**
rng	είσοδος	scaling_factor_res	Τυχαίος αριθμός
data_valid	είσοδος	1	IRQ έγκυρων δεδομένων εισόδου
fit_sum	είσοδος	score_sz + $\log_2$(pop_sz)	Άθροισμα τιμών προσαρμογής όλων των γονιδίων της τρέχουσας γενιάς
sel_parent	έξοδος	genom_lngt	Επιλεγμένος γονέας
rd	έξοδος	1	ready out IRQ σήμα τερματισμού λειτουργίας της υπομονάδας

Πίνακας 9.5: Επεξήγηση των σημάτων της υπομονάδας επιλογής

9.3.2.4 Υπομονάδα Διασταύρωσης

Η υπομονάδα διασταύρωσης (*crossover*) εκτελείται ύστερα από το τερματισμό της υπομονάδας επιλογής και εφαρμόζει τη γενετική πράξη της διασταύρωσης στους επιλεγμένους γονείς. Η μέθοδος διασταύρωσης που θα εφαρμοσθεί επιλέγεται παραμετρικά. Στη βιβλιογραφία αναφέρονται διάφορες μέθοδοι διασταύρωσης [31][105][106]. Η παρούσα υλοποίηση πραγματοποιεί τρεις διαφορετικές μεθόδους για διασταύρωση και πιο συγκεκριμένα τη διασταύρωση ενός σημείου, δύο σημείων *(single, two point crossover)* και την ομοιόμορφη διασταύρωση *(uniform crossover)*.

Προκειμένου να λειτουργήσει το μπλοκ διασταύρωσης και να εφαρμόσει την επιθυμητή μέθοδο διασταύρωσης, που έχει τεθεί παραμετρικά, απαιτούνται κάποιοι τυχαίοι αριθμοί (*random crossover points, random mask*) που παράγονται από δύο γεννήτριες τυχαίων αριθμών. Η πρώτη γεννήτρια παράγει τα τυχαία σημεία διασταύρωσης (διασταύρωση ενός ή δύο σημείων) μέσα στο γονίδιο, ενώ η δεύτερη παράγει την τυχαία δυαδική μάσκα (ίση με το μήκος του γονιδίου) που είναι αναγκαία για την ομοιόμορφη διασταύρωση. Οι μέθοδοι διασταύρωσης φαίνονται στο Σχήμα 9.8(α)–(γ).



Σχήμα 9.8: Μέθοδοι Διασταύρωσης (α) Διασταύρωση ενός σημείου (β) Διασταύρωση δύο σημείων και (γ) Ομοιόμορφη Διασταύρωση

Η επεξήγηση των βασικών σημάτων της υπομονάδας διασταύρωσης αναφέρονται στον Πίνακα 9.6 που ακολουθεί.

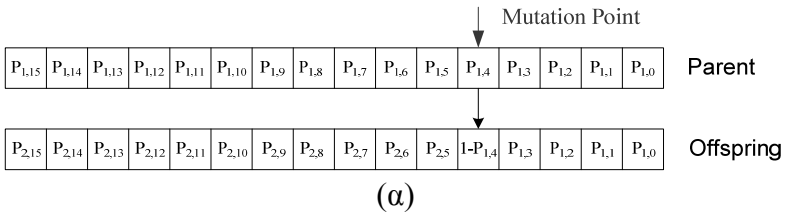
Σήμα	**Τύπος**	**Ανάλυση (σε bit)**	**Περιγραφή λειτουργίας**
rng	είσοδος	genom_lngt	Τυχαία δυαδική μάσκα
cross_points	είσοδος	$2\times\log_2$(genom_lngt)	Τυχαία σημεία διασταύρωσης
cross_method	είσοδος	2	Μέθοδος Διασταύρωσης (00: Ενός σημείου, 01: Διπλού σημείου, 10: Ομοιόμορφη)
inGene1, inGene2	είσοδος	genom_lngt	Γονείς εισόδου, που θα εφαρμοστεί η διασταύρωση
rd	έξοδος	1	ready out IRQ σήμα τερματισμού λειτουργίας της υπομονάδας
cross_offs	έξοδος	genom_lngt	Απόγονος διασταύρωσης

Πίνακας 9.6: Επεξήγηση των σημάτων της υπομονάδας διασταύρωσης

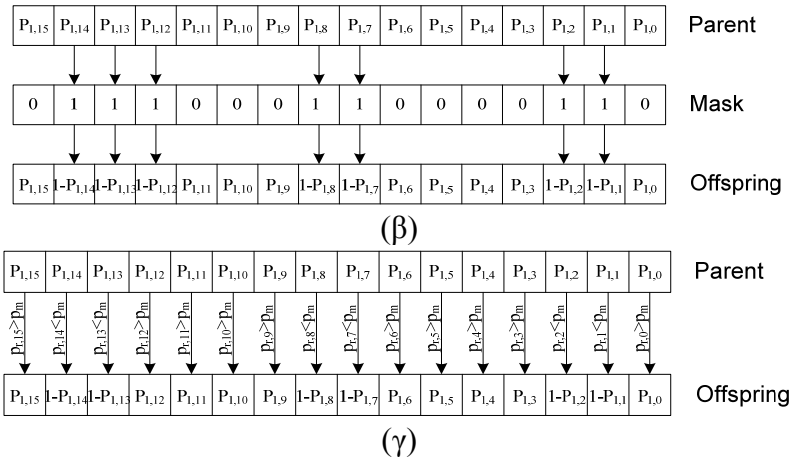
9.3.2.5 Υπομονάδα Μετάλλαξης

Η υπομονάδα μετάλλαξης (*mutation*) ακολουθεί μετά από τη λειτουργία του υποσυστήματος διασταύρωσης και εφαρμόζει τη γενετική πράξη της μετάλλαξης (η οποία επιλέγεται παραμετρικά) στον απόγονο που έχει προέλθει από τα υποσύστημα διασταύρωσης. Στη βιβλιογραφία αναφέρονται διάφορες μέθοδοι [33][31][105][106]. Στην παρούσα υπομονάδα υλοποιούνται τρεις διαφορετικές μέθοδοι μετάλλαξης και συγκεκριμένα η μετάλλαξη ενός σημείου, η μετάλλαξη με μάσκα *(*masked mutation*)* και η ομοιόμορφη μετάλλαξη *(uniform mutation)*.

Για να εφαρμοστεί η επιθυμητή μέθοδος μετάλλαξης απαιτείται μια σειρά τυχαίων αριθμών από δύο γεννήτριες τυχαίων αριθμών (*random mutation points, random mask, random numbers* $p_{r,i}$),. Η πρώτη γεννήτρια παράγει τα τυχαία σημεία μετάλλαξης μέσα στο γονίδιο για τη μετάλλαξη ενός σημείου. Η δεύτερη παράγει την απαραίτητη για τη μετάλλαξη τυχαία δυαδική μάσκα (ίση με το μήκος του γονιδίου) και επιπλέον τον απαραίτητο τυχαίο αριθμό, για να αποφασιστεί αν η πράξη της μετάλλαξης θα εφαρμοστεί ή όχι. Πιο συγκεκριμένα η πράξη της μετάλλαξης θα εφαρμοστεί μόνο αν η τιμή του τυχαίου αυτού αριθμού είναι μικρότερη ή ίση της παραμετρικά ορισμένης πιθανότητας μετάλλαξης p_m. Επιπρόσθετα, η γεννήτρια αυτή παράγει και τους τυχαίους αριθμούς $p_{r,i}$ αν η μέθοδος που έχει επιλεγεί παραμετρικά είναι η ομοιόμορφη μετάλλαξη. Οι τρεις μέθοδοι που υλοποιούνται στην υπομονάδα μετάλλαξης αποτυπώνονται στο Σχήμα 9.9(α)–(γ).

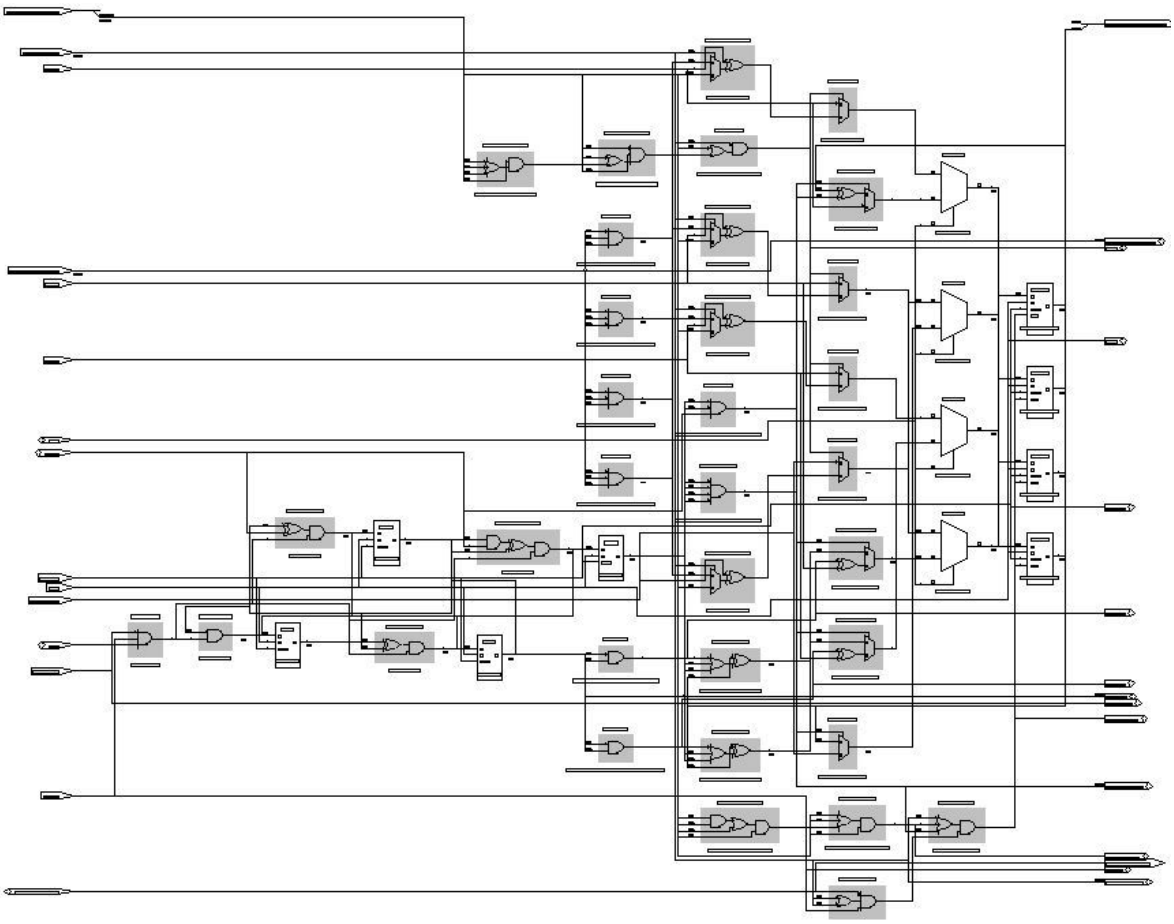


(α)



Σχήμα 9.9: Μέθοδοι Μετάλλαξης (α) Μετάλλαξη ενός σημείου (β) Μετάλλαξη με μάσκα και (γ) Ομοιόμορφη Μετάλλαξη

Στο Σχήμα 9.10 που ακολουθεί φαίνεται ένα μέρος της κυκλωματικής υλοποίησης (netlist σε επίπεδο RTL) της υπομονάδας μετάλλαξης (βλ. Παράρτημα Α.4) όπως απεικονίζεται στο εργαλείο λογικής σύνθεσης Synplify Pro που χρησιμοποιήθηκε.



Σχήμα 9.10: Μέρος της κυκλωματική υλοποίησης της υπομονάδας μετάλλαξης σε επίπεδο RTL

Η επεξήγηση των σημάτων της υπομονάδας μετάλλαξης δίνονται στον Πίνακα 9.7 παρακάτω.

Σήμα	**Τύπος**	**Ανάλυση σε bit**	**Περιγραφή λειτουργίας**
rng	είσοδος	genom_lngt + mut_res	Τυχαία δυαδική μάσκα και τυχαίος δυαδικός αριθμός
mutPoint	είσοδος	log2(genom_lngt)	Τυχαίο σημείο μετάλλαξης
mutMethod	είσοδος	2	Μέθοδος μετάλλαξης (00: Ενός σημείου, 01: Με μάσκα, 10: Ομοιόμορφη)
inGene	είσοδος	genom_lngt	Γονίδιο που θα εφαρμοστεί η μετάλλαξη
rd	έξοδος	1	ready out IRQ σήμα τερματισμού λειτουργίας της υπομονάδας
mutOffs	έξοδος	genom_lngt	Απόγονος μετάλλαξης

Πίνακας 9.7: Επεξήγηση των σημάτων της υπομονάδας μετάλλαξης

9.3.2.6 Υπομονάδα Επιτήρησης

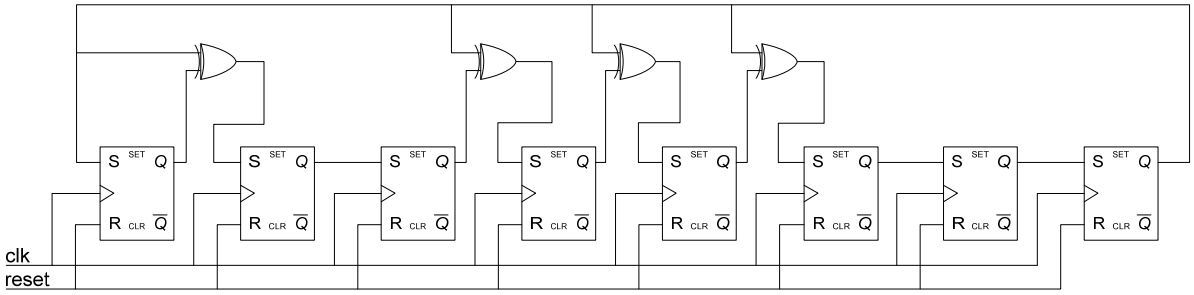
Η υπομονάδα επιτήρησης ενεργοποιείται κάθε φορά που μια νέα γενιά δημιουργείται και ελέγχει αν τα κριτήρια τερματισμού (μέγιστος αριθμός γενεών, όριο τιμής προσαρμογής) έχουν ικανοποιηθεί, έτσι ώστε να αποφασιστεί αν η εκτέλεση του αλγορίθμου πρέπει να συνεχιστεί ή όχι. Η βασική λειτουργία των σημάτων της υπομονάδας αυτής δίνεται στον Πίνακα 9.8.

Σήμα	**Τύπος**	**Ανάλυση σε bit**	**Περιγραφή λειτουργίας**
max_fit	είσοδος	score_sz	Μέγιστη επιθυμητή τιμή της τιμής προσαρμογής
fitlimit_rd	έξοδος	1	IRQ σήμα που ειδοποιεί ότι η τιμή προσαρμογής κάποιου γονιδίου έχει φτάσει τη μέγιστη επιθυμητή τιμή
rd	έξοδος	1	ready out IRQ σήμα τερματισμού λειτουργίας της υπομονάδας

Πίνακας 9.8: Επεξήγηση των σημάτων της υπομονάδας επιτήρησης

9.3.2.7 Γεννήτριες Τυχαίων Αριθμών

Στην ενότητα αυτή περιγράφονται οι γεννήτριες τυχαίων αριθμών, που τροφοδοτούν τις περισσότερες υπομονάδες με τυχαίους αριθμούς. Εσωτερικά οι γεννήτριες υλοποιούν έναν καταχωρητή ολίσθησης με γραμμική ανάδραση (Linear Feedback Shift Register - LFSR), του οποίου το μήκος ορίζεται παραμετρικά. Η κυκλωματική υλοποίηση του LFSR των 8-bit δίνεται στο Σχήμα 9.11.



Σχήμα 9.11: Κυκλωματική υλοποίηση του LFSR των 8-bit

Συνολικά τέσσερις γεννήτριες τυχαίων αριθμών (*Random Number Generator* – RNG) χρησιμοποιούνται ώστε να παραχθεί μια αρχική τυχαία γενιά καθώς και οι απαραίτητες τυχαίες τιμές. Το μέγιστο μήκος μιας τυχαίας ακολουθίας που μπορεί να παραχθεί είναι 128-bit.

Τα χαρακτηριστικά των τεσσάρων γεννητριών που χρησιμοποιούνται στον πυρήνα του ΓΑ παρατίθενται στον Πίνακα 9.9.

RNG	**Μήκος LFSR**
RNG 0	genom_lngt
RNG 1	scaling_factor_res
RNG 2	genom_lngt + mut_res
RNG 3	$2 \times \log_2$(genom_lngt)

Πίνακας 9.9: Χαρακτηριστικά των γεννητριών τυχαίων αριθμών

9.3.2.8 Μνήμες RAM

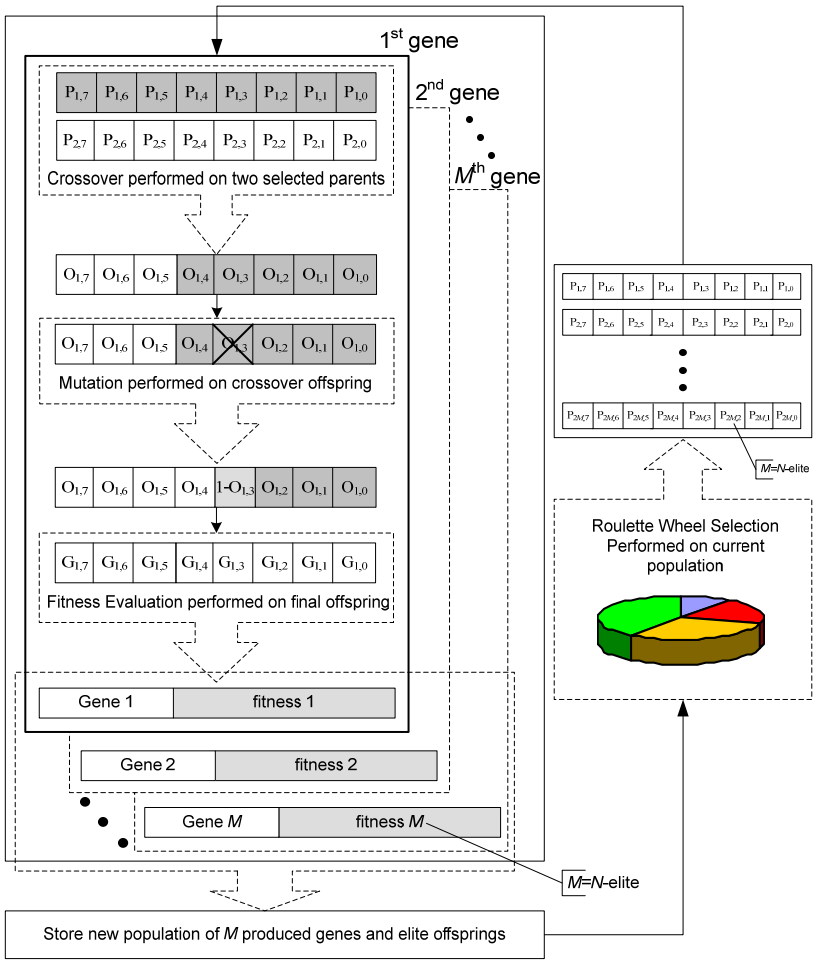
Οι μνήμες τυχαίας προσπέλασης RAM 1 και RAM 2, αποθηκεύουν την τρέχουσα γενιά και τους επιλεγμένους γονείς αντίστοιχα. Το μέγεθος της μνήμης (μήκος διεύθυνσης και δεδομένων) ορίζεται παραμετρικά όπως φαίνεται και στον Πίνακα 9.10.

Μνήμες RAM	**Μήκος Διεύθυνσης (Address width)**	**Μήκος Δεδομένων (Data width)**
RAM 1 - τρέχουσας γενιάς	genom_lngt	genom_lngt + score_sz
RAM 2 - γονέων	2×(pop_sz-elite)	genom_lngt

Πίνακας 9.10: Παράμετροι των μνημών RAM

9.3.2.9 Μηχανισμός Λειτουργίας του ΓΑ

Η γενική μορφή της λειτουργίας του πυρήνα του ΓΑ που υλοποιήθηκε αποτυπώνεται στο Σχήμα 9.12.

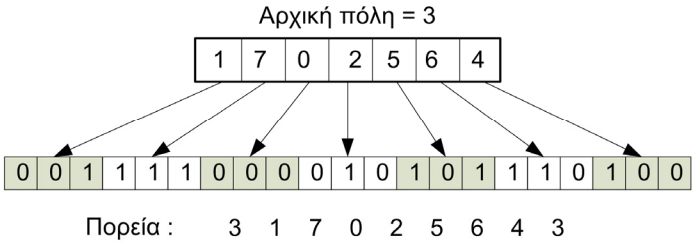


Σχήμα 9.12: Μηχανισμός λειτουργίας του ΓΑ

9.4 Προσαρμογή του Πυρήνα του ΓΑ στο Πρόβλημα του Πλανόδιου Πωλητή

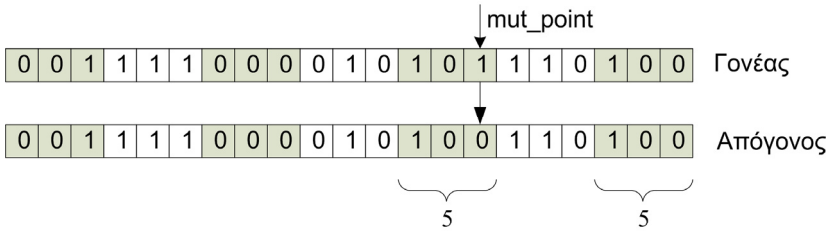
Το πρόβλημα του Πλανόδιου Πωλητή (*Travelling Salesman Problem* – TSP) είναι ένα κλασσικό συνδυαστικό πρόβλημα βελτιστοποίησης (*combinatorial optimization problem*). Δεδομένης μιας λίστας πόλεων και των αποστάσεων μεταξύ τους, ο σκοπός είναι να βρεθεί το μικρότερο σε μήκος μονοπάτι μέσω του οποίου ο πωλητής επισκέπτεται την κάθε πόλη μόνο μια φορά [107] και επιστρέφει στην πόλη από την οποία ξεκίνησε. Στον συμμετρικό TSP αλγόριθμο που εφαρμόζεται εδώ, η απόσταση μεταξύ δύο πόλεων είναι ίση και προς τις δυο κατευθύνσεις (μη κατευθυνόμενος γράφος), ενώ αντίθετα στη μη συμμετρική εκδοχή του αλγορίθμου μπορεί να απουσιάζει το ένα μονοπάτι μεταξύ των δυο κατευθύνσεων ή η απόσταση των δυο κατευθύνσεων να είναι διαφορετική (κατευθυνόμενος γράφος).

Προκειμένου να προσαρμόσουμε το ΓΑ στο TSP το κάθε γονίδιο θα παριστάνει και ένα διαφορετικό μονοπάτι που επιτρέπεται να ακολουθήσει ο πωλητής, ενώ θα περιέχει την ακολουθία των πόλεων κατά σειρά επίσκεψης. Η κάθε πόλη παριστάνεται δυαδικά με κάποιο αριθμό bit που τίθεται παραμετρικά, επομένως για n αριθμό πόλεων απαιτούνται $\log_2(n)$ bit για την παράσταση τους. Στο Σχήμα 9.13 απεικονίζεται ένα πιθανό γονίδιο για το TSP με 8 πόλεις.



Σχήμα 9.13: Αναπαράσταση ενός πιθανού γονιδίου

Βάση του ορισμού του TSP, ο πωλητής πρέπει να επισκεφτεί μόνο μια φορά την κάθε πόλη ή αντίστοιχα το κάθε γονίδιο στον πληθυσμό πρέπει να περιέχει μόνο μια φορά το δείκτη της κάθε πόλης. Ως αποτέλεσμα του προηγούμενου είναι ότι τα γονίδια δεν έχουν εντελώς τυχαία μορφή και επομένως δεν είναι δυνατό να χρησιμοποιηθούν οι προαναφερόμενες μέθοδοι διασταύρωσης και μετάλλαξης, αφού αυτό θα οδηγούσε πιθανότατα σε λανθασμένη μορφή γονιδίου (π.χ., γονίδιο με διπλές εγγραφές της ίδιας πόλης). Επομένως για να είναι δυνατή η προσαρμογή του πυρήνα του ΓΑ στο TSP, είναι αναγκαίο να υλοποιηθούν διαφορετικοί τελεστές διασταύρωσης και μετάλλαξης [108]. Στο Σχήμα 9.14 αναπαρίσταται σχηματικά ένα γονίδιο απόγονος που προέκυψε με μετάλλαξη ενός σημείου στο προηγούμενο γονίδιο (βλ. Σχήμα 9.13) και περιέχει δύο φορές τη πόλη νούμερο 5.

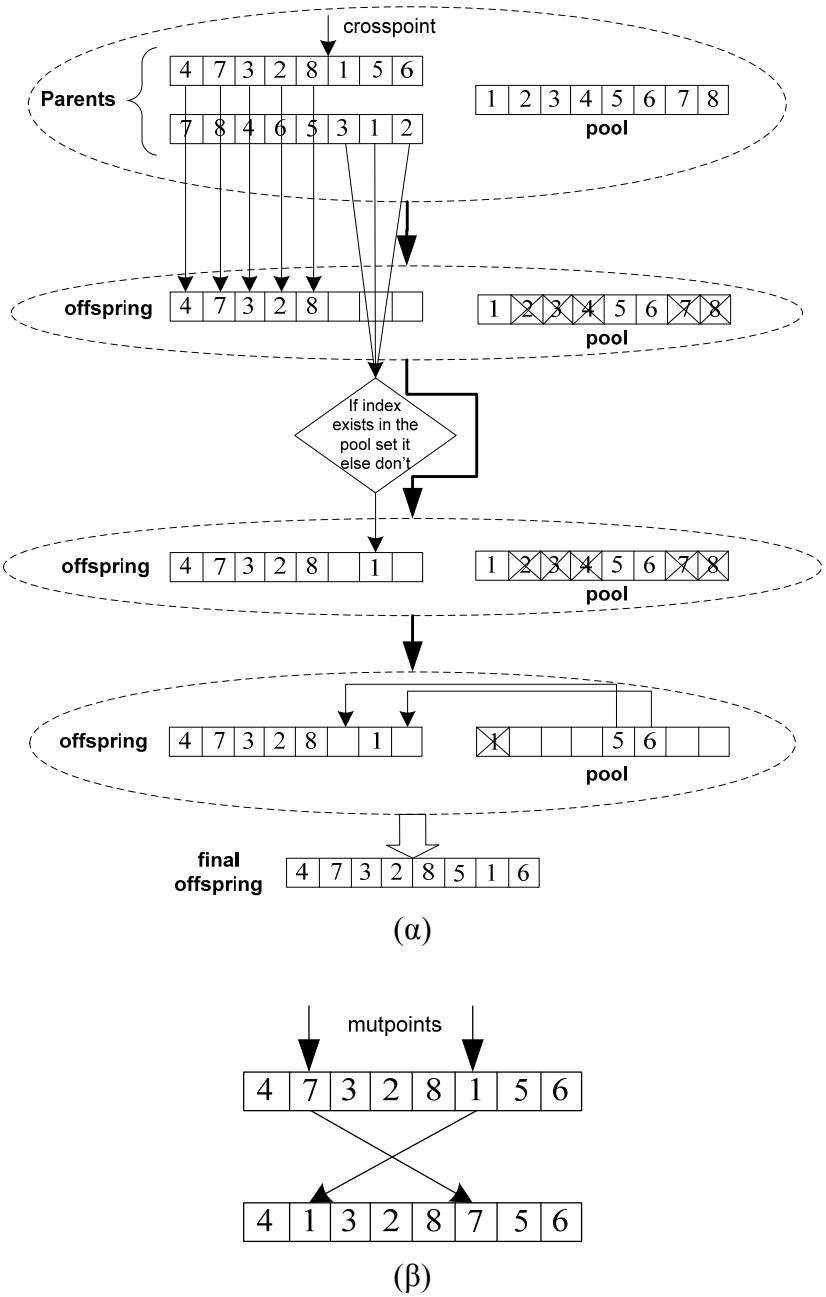


Σχήμα 9.14: Λανθασμένη περίπτωση γονιδίου με διπλή εγγραφή πόλης

Ο νέος τελεστής διασταύρωσης που υλοποιήθηκε χρησιμοποιεί μια δεξαμενή (pool) που περιέχει όλους τους δείκτες των πόλεων. Προκειμένου το κάθε γονίδιο να περιέχει μόνο μια καταχώρηση της κάθε πόλης, οι πόλεις που περιέχονται στο γονίδιο απομακρύνονται από τη δεξαμενή. Η διαδικασία που ακολουθείται για την παραγωγή ενός απογόνου από δύο γονείς που υφίστανται τη γενετική πράξη της διασταύρωσης περιγράφεται αναλυτικά στην παράγραφο που ακολουθεί.

Αρχικά, παράγεται ένας τυχαίος αριθμός, που θα αποτελεί το σημείο διασταύρωσης. Στη συνέχεια οι δείκτες των πόλεων που περιέχονται στον πρώτο γονέα αντιγράφονται στον απόγονο ξεκινώντας από το σημείο διασταύρωσης και καταλήγοντας στην αρχή του γονιδίου. Οι δείκτες των πόλεων που αντιγράφηκαν προηγουμένως απομακρύνονται από τη δεξαμενή. Στη συνέχεια, γεμίζεται το δεξιό μέρος του απογόνου (δηλαδή από το σημείο διασταύρωσης ως το τέλος του γονιδίου) ελέγχοντας αν οι δείκτες του δεύτερου γονέα από το σημείο διασταύρωσης ως το τέλος του γονιδίου περιέχονται στη δεξαμενή. Αν ένας δείκτης είναι ελεύθερος, δηλαδή περιέχεται και στη δεξαμενή και στον δεύτερο γονέα, τότε αντιγράφεται στον απόγονο και διαγράφεται από τη δεξαμενή. Αν ένας δείκτης δεν περιέχεται στη δεξαμενή, τότε προσπερνιέται και η αντίστοιχη θέση του απογόνου μένει κενή. Τέλος, οι κενές θέσεις συμπληρώνονται από τους εναπομείναντες δείκτες στη δεξαμενή.

Ο νέος τελεστής μετάλλαξης που υλοποιήθηκε χρησιμοποιεί δύο τυχαίους αριθμούς, που παριστάνουν δύο σημεία μετάλλαξης, και απλώς ανταλλάζει τους δείκτες των πόλεων μεταξύ αυτών των δύο σημείων. Οι νέες μέθοδοι διασταύρωσης και μετάλλαξης για τις 8 πόλεις απεικονίζονται στο Σχήμα 9.15.



Σχήμα 9.15: Μέθοδοι διασταύρωσης και μετάλλαξης για το πρόβλημα του Πλανόδιου Πωλητή

Η συνάρτηση προσαρμογής (υλοποιείται στην υπομονάδα εκτίμησης της τιμής προσαρμογής - *fitness evaluation module*) από την οποία προκύπτει η τιμή προσαρμογής είναι το άθροισμα πάνω στις N πόλεις των τετραγώνων των Ευκλείδειων αποστάσεων μεταξύ δύο διαδοχικών πόλεων, σύμφωνα με το υπολογισμένο μονοπάτι που θα ακολουθήσει ο πωλητής. (βλ. Σχέση 9.1). Αξίζει να διευκρινιστεί ότι επιλέχθηκε ο

υπολογισμός του τετραγώνου των Ευκλείδειων αποστάσεων ούτως ώστε να αποφευχθεί η υλοποίηση σε υλικό της τετραγωνικής ρίζας, κάτι το οποίο δεν είναι αναγκαίο αφού απλά μας αρκεί η σύγκριση των τιμών προσαρμογής μεταξύ των γονιδίων.

$$\begin{aligned} fit_j &= \sum_{i=1}^{N-1}\left(\sqrt{\left(x_i - x_{i+1}\right)^2 + \left(y_i - y_{i+1}\right)^2}\right)^2 + \left(\sqrt{\left(x_N - x_1\right)^2 + \left(y_N - y_1\right)^2}\right)^2 \\ &= \sum_{i=1}^{N-1}\left(\left(x_i - x_{i+1}\right)^2 + \left(y_i - y_{i+1}\right)^2\right) + \left(\left(x_N - x_1\right)^2 + \left(y_N - y_1\right)^2\right) \quad j \in [1, pop\_sz] \end{aligned} \tag{9.1}$$

Ο γενετικός αλγόριθμος είναι σχεδιασμένος έτσι ώστε να καταλήγει στο γονίδιο με τη μεγαλύτερη τιμή προσαρμογής (μακρύτερο μονοπάτι). Επειδή ο στόχος είναι η εύρεση της βέλτιστης ελάχιστης πορείας δεδομένων των πόλεων, πρέπει είτε να αντιστραφεί η τιμή προσαρμογής ή εναλλακτικά να αφαιρεθεί από τη μέγιστη δυναμική περιοχή που ισούται με $2^{score\_size} - 1$. Με αυτόν τον τρόπο μεγάλες τιμές προσαρμογής αντιστοιχούν σε μικρά μονοπάτια. Το VHDL generic *inv_type,* (βλ. Πίνακα 9.1) επιτρέπει γι' αυτό το σκοπό την παραμετρική επιλογή της μεθόδου αλλαγής της τιμής προσαρμογής που θα εφαρμοστεί τελικά.

9.5 Μεθοδολογία Σχεδίασης και Υλοποίησης του Πυρήνα ΓΑ

Η μεθοδολογία σχεδίασης και υλοποίησης του συστήματος που ακολουθήθηκε στο παρόν κεφάλαιο κατά τη σχεδίαση και υλοποίηση του πυρήνα του ΓΑ σε FPGA, είναι ίδια με αυτή που είδη αναφέρθηκε αναλυτικά στο κεφάλαιο 6. Επιγραμματικά οι διαδικασία περιλαμβάνει τα παρακάτω στάδια,

- Προδιαγραφές σχεδίασης (*Design specifications*)
- Μοντελοποίηση πυρήνα ΓΑ με συγγραφή κώδικα σε γλώσσα MATLAB της εταιρίας Mathworks)
- Περιγραφή του συστήματος σε γλώσσα περιγραφής υλικού VHDL σε επίπεδο RTL
- Προσομοίωση του πυρήνα ΓΑ σε επίπεδο RTL (RTL *simulation*) με χρήση του εργαλείου Modelsim της εταιρίας Mentor Graphics
- Σύνθεση σε λογικό επίπεδο (*Logic synthesis*) – χρήση του εργαλείου Synplify Pro της εταιρίας Synplicity
- Θέση και Δρομολόγηση στο FPGA (*Place and Route*) με χρήση της πλατφόρμας ISE της εταιρίας Xilinx
- Προσομοίωση χρονισμού (*Timing simulation*) με χρήση του εργαλείου Modelsim

9.6 Αποτελέσματα Υλοποίησης

Η ενότητα αυτή περιγράφει τα αποτελέσματα της υλοποίησης του ΓΑ που σχεδιάστηκε. Ύστερα από την περιγραφή του κυκλώματός σε επίπεδο RTL με τη χρήση VHDL κώδικα και τη σύνθεσή του σε λογικό επίπεδο (Synplify Pro) ακολουθεί η θέση και η δρομολόγηση του στο FPGA μέσω του εργαλείου Xilinx ISE. Το αποτέλεσμα της

υλοποίησης του πυρήνα του ΓΑ στο υλικό παρουσιάζεται στον Πίνακα 9.11 και 9.12 αντίστοιχα για τον προσαρμοσμένο πυρήνα στο TSP.

Ο πυρήνας του ΓΑ στο FPGA επιτυγχάνει συχνότητα λειτουργίας ίση με 92 MHz (10.8 ns) ενώ η προσαρμοσμένη έκδοση του πυρήνα στο TSP επιτυγχάνει αντίστοιχα συχνότητα λειτουργίας 80 MHz (12.5 ns), ενώ απαιτούνται 2.450 ns (2.4 μs) και 14.391 ns (14.3 μs) για τη δημιουργία μιας νέας γενιάς 8 γονιδίων και στις δύο εκδόσεις του ΓΑ αντίστοιχα. Τέλος, ο πυρήνας για τα μοντέλα των ΓΑ που παρουσιάζονται εδώ είναι πλήρως παραμετρικός, επιτρέποντας τον έλεγχο των μοντέλων με διάφορες προδιαγραφές.

Logic Utilization	**Πυρήνας ΓΑ**	**Διαθέσιμα**	**Ποσοστό χρήσης**
Slice Flip Flops	681	26.624	2%
4 input LUT's	1.086	26.624	4%
Logic distribution			
Occupied Slices	892	13.312	6%
4 input LUT's	1.116	26.624	6%
Used as logic	1.086		
Used as route–thru	6		
Used as 16x1 RAMs	24		
Bonded IOBs	59	487	12%
MULT 18x18s	1	32	3%
GCLKs	1	8	12%

Πίνακας 9.11: Χρησιμοποιούμενοι πόροι υλοποίησης του πυρήνα του ΓΑ

Logic Utilization	**Πυρήνας ΓΑ προσαρμοσμένος στο TSP**	**Διαθέσιμα**	**Ποσοστό χρήσης**
Slice Flip Flops	1045	26.624	3%
4 input LUT's	1630	26.624	6%
Logic distribution			
Occupied Slices	1305	13.312	9%
4 input LUT's	1686	26.624	6%
Used as logic	1630		
Used as route–thru	4		
Used as 16x1 RAMs	52		
Bonded IOBs	53	487	10%
MULT 18x18s	3	32	9%
GCLKs	1	8	12%

Πίνακας 9.12: Χρησιμοποιούμενοι πόροι υλοποίησης του προσαρμοσμένου πυρήνα ΓΑ στο TSP

9.7 Μοντελοποίηση του ΓΑ σε λογισμικό

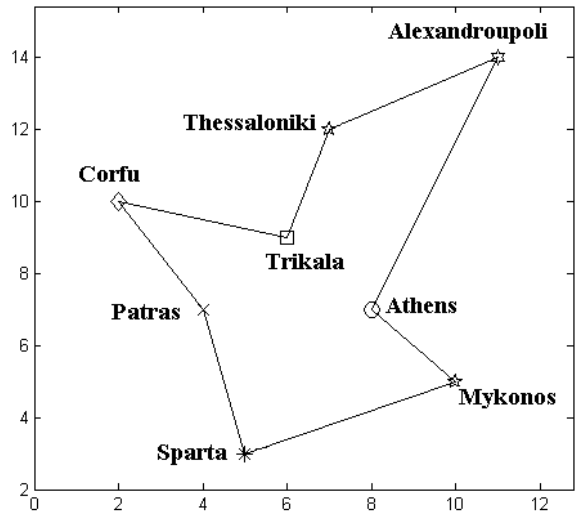
Ο ΓΑ αλγόριθμος που περιγράφεται σε αυτό το κεφάλαιο αναπτύχθηκε σε αρχικά σε λογισμικό και συγκεκριμένα στην πλατφόρμα MATLAB (βλ. Παράρτημα Α.4). Η ανάπτυξη του συστήματος σε λογισμικό έγινε ούτως ώστε να υπάρξει ένα μοντέλο του ΓΑ πάνω στο οποίο μπορούμε να ελέγξουμε τη σωστή λειτουργία του και τη δυνατότητα του να επιλύει δύσκολα προβλήματα βελτιστοποίησης πριν τη σχεδίαση και υλοποίησή του σε υλικό. Επιπλέον, μέσω του μοντέλου του ΓΑ είναι δυνατή η παραγωγή διανυσμάτων ελέγχου (test vectors) για την πιστοποίηση της ορθότητας του σχεδιασμού στο FPGA. Τα διανύσματα ελέγχου περιλαμβάνουν τιμές εισόδων, εξόδων και ενδιάμεσων καταστάσεων του συστήματος και κατά την επαλήθευση της υλοποίησης συγκρίνονται άμεσα με τα αντίστοιχα σήματα του κυκλώματος στο FPGA.
Η δομή που ακολουθήθηκε κατά την ανάπτυξη του λογισμικού είναι ίδια με τη δομή της υλοποίησης του πυρήνα του ΓΑ στο υλικό και συγκεκριμένα για κάθε υπομονάδα του ΓΑ δημιουργήθηκε και η αντίστοιχη συνάρτηση στο MATLAB που παράγει ακριβώς τα ίδια αποτελέσματα. Αξίζει να σημειωθεί, ότι και η υλοποίηση στο λογισμικό είναι παραμετρική σε αντιστοιχία με τις παραμέτρους του Πίνακα 9.1.

9.8 Αξιολόγηση του Πυρήνα ΓΑ

Ο πυρήνας ΓΑ αξιολογήθηκε με δύο τρόπους. Αρχικά, ο χρόνος επίλυσης του TSP που απαιτεί ο προσαρμοσμένος πυρήνας ΓΑ, συγκρίθηκε με τον απαιτούμενου χρόνο επίλυσης του TSP που χρειάζεται η λογισμική υλοποίηση του ίδιου ΓΑ σε περιβάλλον MATLAB. Ουσιαστικά δηλαδή έγινε μια άμεση σύγκριση της υλοποίησης του ΓΑ σε υλικό έναντι της ίδιας υλοποίησης σε λογισμικό (*hardware vs. software implementation*). Εν συνεχεία, χρησιμοποιήθηκαν κάποιες συναρτήσεις αξιολόγησης (*benchmark functions*) προκειμένου να αξιολογηθεί ο πυρήνας περαιτέρω [109][110].

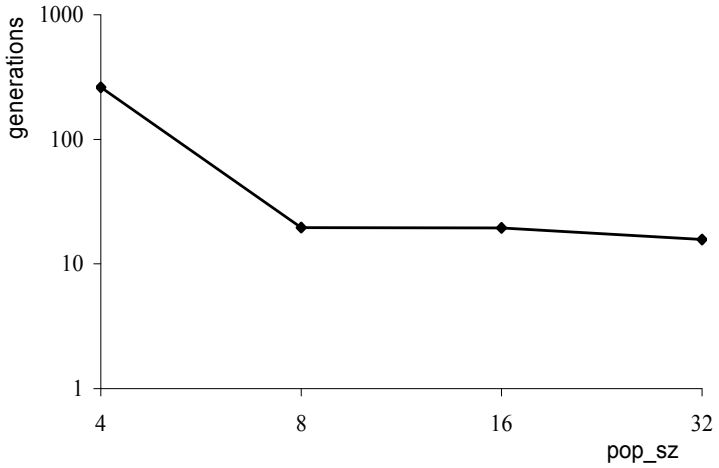
9.8.1 Αξιολόγηση Απόδοσης του Προσαρμοσμένου Πυρήνα ΓΑ στο TSP

Η αξιολόγηση της απόδοσης του ΓΑ μέσω του προβλήματος του Πλανόδιου Πωλητή έγινε συγκρίνοντας το χρόνο που χρειάστηκε η έκδοση του ΓΑ σε λογισμικό για να συγκλίνει σε σχέση με τον αντίστοιχο χρόνο του υλοποιημένου σε υλικό ΓΑ. Στο Σχήμα 9.16 απεικονίζεται ο χάρτης των 8 πόλεων που χρησιμοποιήθηκε, οι οποίες είναι υπαρκτές πόλεις της ελληνικής περιφέρειας και οι αποστάσεις μεταξύ τους είναι ακριβώς ανάλογες με τις πραγματικές.



Σχήμα 9.16: Χάρτης πόλεων που χρησιμοποιήθηκαν

Στο Σχήμα 9.17 φαίνεται η συμβολή του αριθμού γονιδίων (μέγεθος πληθυσμού – pop_sz) στις απαιτούμενες γενεές που απαιτούνται από τον αλγόριθμο προκειμένου να συγκλίνει προς τη βέλτιστη λύση.



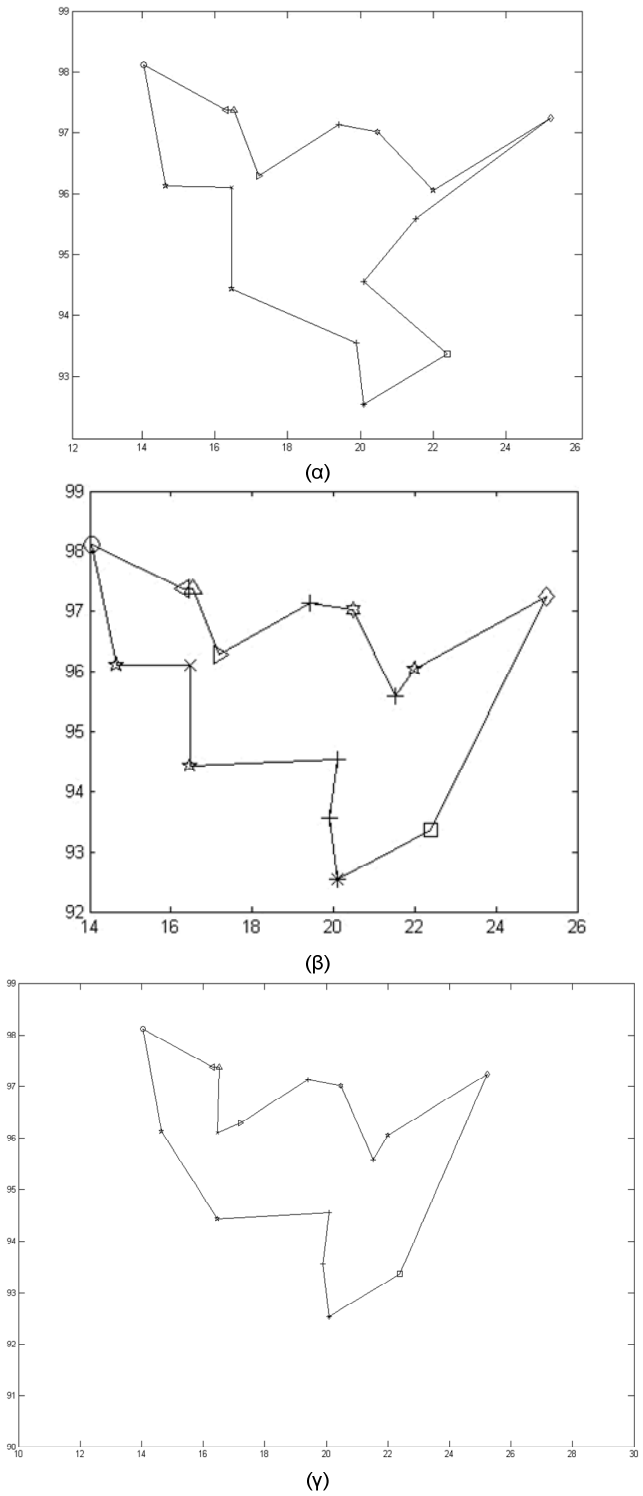
Σχήμα 9.17: Συσχετισμός του αριθμού γονιδίων στις απαιτούμενες γενεές για σύγκλιση

Τα αποτελέσματα για οκτώ πόλεις, 60 γενιές και 32 γονίδια για κάθε γενιά απεικονίζονται στον Πίνακα 9.13, όπου και παρατηρείται μια εντυπωσιακή επιτάχυνση της ταχύτητας σύγκλισης του αλγορίθμου με λόγο ταχύτητας λογισμικού προς ταχύτητα υλικού ίσο με ~11.036.

Υλοποίηση ΓΑ	**Χρόνος (msec)**
Hardware (συχνότητα πυρήνα 80 MHz)	1.702
Software (Pentium-4 3,2 GHz 1Gb RAM)	18783

Πίνακας 9.13: Σύγκριση του χρόνου σύγκλησης της υλοποίησης του ΓΑ σε λογισμικό σε αντιδιαστολή με την υλοποίησή του σε υλικό

Επιπλέον, ο ΓΑ αξιολογήθηκε χρησιμοποιώντας ένα σύνολο πόλεων που ονομάζεται Burma14 και έχει ληφθεί από τη διεθνή βιβλιοθήκη για την αξιολόγηση του TSP (TSPLIB) [111]. Οι λύσεις του TSP με το σύνολο πόλεων Burma14 για διαφορετικά ζεύγη μεγέθους πληθυσμού, πλήθους γενεών (pop_sz, max_gen), αποτυπώνονται στο Σχήμα 9.18(α)-(γ).



Σχήμα 9.18: Λύση του TSP με Burma14 για (α) *pop_sz*=16, *max_gen*=371, (β) *pop_sz*=32, *max_gen*=1600, (γ) *pop_sz*=64, *max_gen*=360

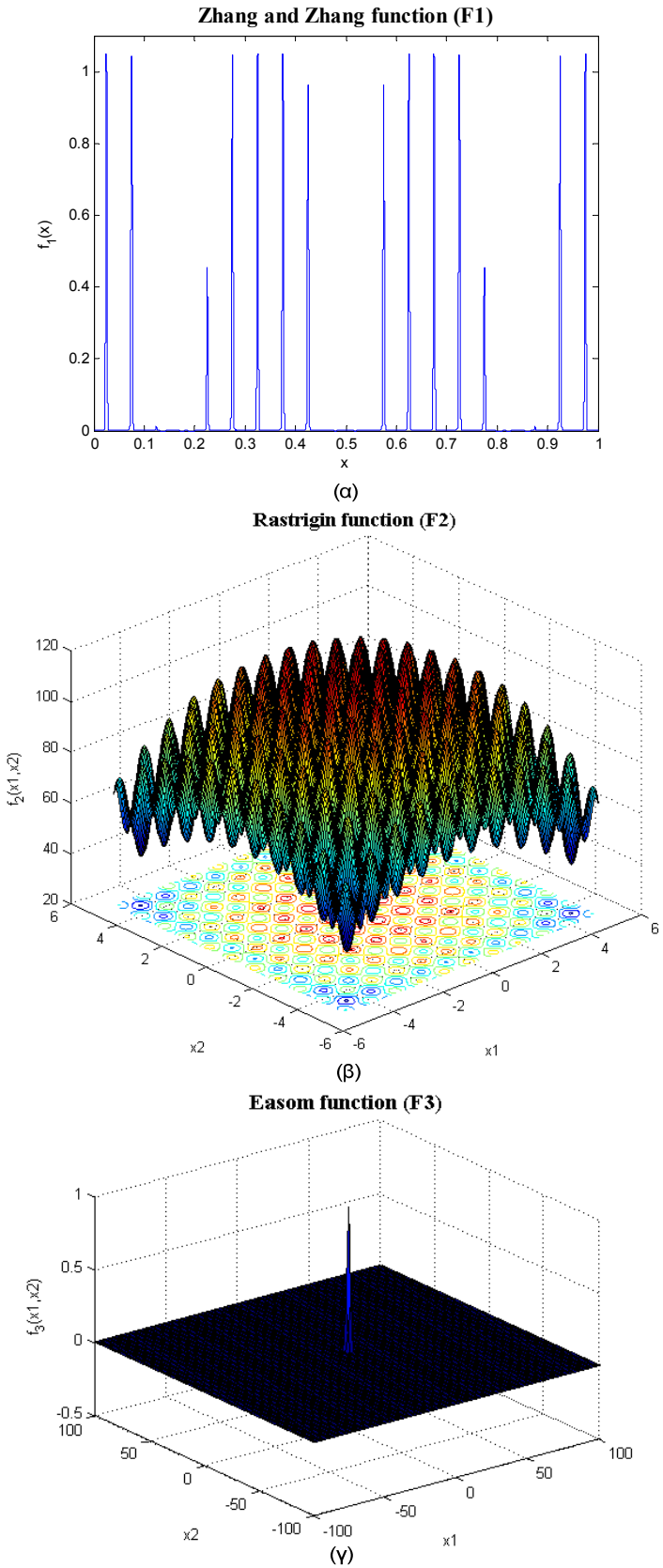
9.8.1.1 Αξιολόγηση Απόδοσης Πυρήνα ΓΑ με Χρήση Benchmark Functions

Στη βιβλιογραφία έχουν προταθεί αρκετές αντικειμενικές συναρτήσεις (objective functions for GA benchmarking) για την αξιολόγηση της απόδοσης των ΓΑ [110][109][112][113], δηλαδή της ικανότητάς τους να συγκλίνουν προς το βέλτιστο σημείο (optimum point) μίας αντικειμενικής συναρτήσης.

Οι αντικειμενικές συναρτήσεις που χρησιμοποιήθηκαν παρουσιάζονται συγκεντρωτικά στον Πίνακα 9.14 παρακάτω και απεικονίζονται στο Σχήμα 9.19 που ακολουθεί στην επόμενη σελίδα. Όλες οι συναρτήσεις είναι συνεχείς (continuous) στο πεδίο ορισμού τους, παρουσιάζουν πολλά ακρότατα (multimodal) και είναι κυρτές (convex).

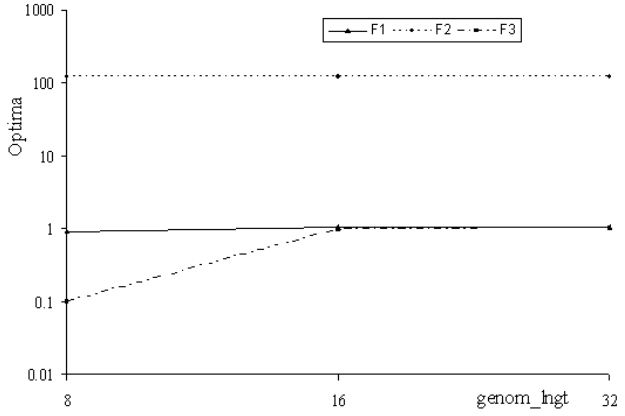
Ονομασία συνάρτησης, *f*	**Μαθηματικός τύπος**	**Βέλτιστο σημείο (Optimum point)**
F1: Zhang and Zhang [109]	$f_1(x) = \left(1 - 2\sin^{20}\left(3\pi x\right) + \sin^{20}\left(20\pi x\right)\right)^{20}, x \in \left[0,1\right]$	Ολικό μέγιστο $\left(0.675, 1.0485\right)$
F2: Rastrigin [112]	$f_2(\bar{x}) = 100 + \sum_{i=1}^{2}\left(x_i^2 - 10\cos\left(2\pi x_i\right)\right), x_i \in \left[\text{-}5.12, 5.12\right]$	Ολικό μέγιστο $\left(\left(0,0\right), 120\right)$
F3: Easom [114]	$f_3(\bar{x}) = \left(\prod_{i=1}^{2}\cos\left(x_i\right)\right) \times \left(e^{-\sum_{i=1}^{2}(x_i - \pi)^2}\right), x_i \in [-100, 100]$	Ολικό μέγιστο $\left(\left(\pi, \pi\right), 1\right)$

Πίνακας 9.14: Συναρτήσεις αξιολόγησης

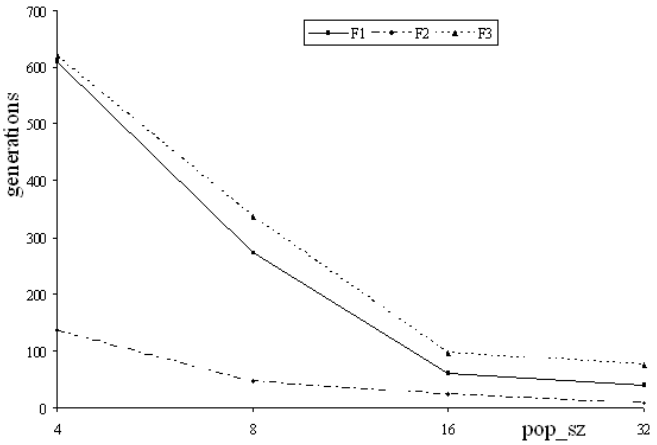


Σχήμα 9.19: Συναρτήσεις αξιολόγησης (α) Zhang and Zhang (F1), (β) Rastrigin (F2), (γ) Easom (F3)

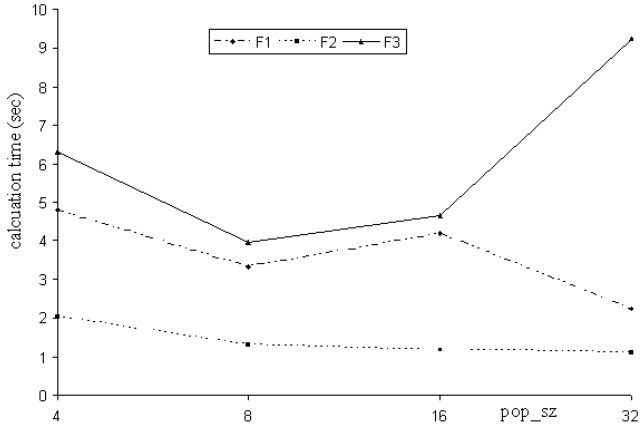
Ακολουθούν διάφορες μετρήσεις αξιολόγησης του ΓΑ που έγιναν με τη χρήση αυτών των συναρτήσεων. Στο Σχήμα 9.20 παρουσιάζεται η επίδραση που έχει το μήκος (σε bit) του γονιδίου (*genom_lngt*) στη βέλτιστη λύση στην οποία καταλήγει ο αλγόριθμος. Η επίδραση που έχει το μέγεθος του πληθυσμού (*pop_sz*) στις απαιτούμενες γενιές για σύγκλιση (*generations*) απεικονίζεται στο Σχήμα 9.21, ενώ τέλος στο Σχήμα 9.22 παρουσιάζεται η επιρροή που έχει το μέγεθος του πληθυσμού (*pop_sz*) στον απαιτούμενο χρόνο υπολογισμού (*calculation time*).



Σχήμα 9.20: Βέλτιστη λύση σε αντιδιαστολή με το Μήκος γονιδίου



Σχήμα 9.21: Απαιτούμενες Γενιές σε αντιδιαστολή με το Μέγεθος πληθυσμού



Σχήμα 9.22: Χρόνος υπολογισμού σε αντιδιαστολή με το Μέγεθος πληθυσμού

Σε αυτό το σημείο πρέπει να αναφερθεί ότι ο συγκεκριμένος γενετικός αλγόριθμος βρίσκει τη βέλτιστη τιμή και των τριών συναρτήσεων, όμως η ακριβής τιμή του εξαρτάται από την ανάλυση σε bit που επιλέγεται για την παράσταση κάθε σημείου μέσα στο γονίδιο. Συγκεκριμένα, πειραματικά καταλήξαμε ότι ένα μήκος γονιδίου 16-bit είναι κατάλληλο για συναρτήσεις μίας μεταβλητής (F1) και 32-bit για συναρτήσεις δύο μεταβλητών (F2 και F3) προκειμένου να πετύχουμε ταυτόχρονα και ικανοποιητική ακρίβεια αλλά και σχετικά μικρό χρόνο υπολογισμού.

9.9 Συμπεράσματα

Στο κεφάλαιο αυτό παρουσιάστηκε η σχεδίαση και υλοποίηση ενός πλήρως παραμετρικού πυρήνα γενετικού αλγορίθμου σε ένα ολοκληρωμένο κύκλωμα προγραμματίζομενης λογικής FPGA. Η παραμετρικότητα του πυρήνα αφορά στις διάφορες παραμέτρους του γενετικού αλγορίθμου, όπως για παράδειγμα, το μέγεθος του πληθυσμού, το μέγιστο όριο γενεών εκτέλεσης, το μέγιστο όριο τιμής προσαρμογής, την πιθανότητα μετάλλαξης, τις μεθόδους διασταύρωσης και μετάλλαξης, το μέγιστο αριθμό γενεών και τον αριθμό των ελίτ απογόνων κάθε γενιάς, κ.τ.λ., που υιοθετούνται στο κύκλωμα με τη χρήση ενός αρχείου παραμέτρων που τροφοδοτεί τον κώδικα VHDL. Επιπλέον, η παραμετρική σχεδίαση επιτρέπει την προσαρμογή του ΓΑ σε διαφορετικές προδιαγραφές χωρίς την ανάγκη αλλαγών στην ψηφιακή σχεδίαση του πυρήνα.

Η υλοποίηση του πυρήνα του γενετικού αλγορίθμου που παρουσιάστηκε λειτουργεί σε συχνότητα ρολογιού ίση με 92 MHz και 80 MHz αντίστοιχα για τον προσαρμοσμένο πυρήνα στο πρόβλημα του Πλανόδιου Πωλητή και επιτυγχάνει μία αξιοσημείωτη επιτάχυνση όταν συγκρίνεται με την αντίστοιχη υλοποίησή του σε λογισμικό.

Επιπρόσθετα, οι πόροι που καταλαμβάνει στο τσιπ η παρούσα υλοποίηση όσον αφορά τον αριθμό πυλών και τα μπλοκ μνήμης RAM, είναι πολύ λίγοι σύμφωνα με τα αποτελέσματα από το εργαλείο θέσης και δρομολόγησης. Συγκρινόμενο με άλλες υλοποιήσεις γενετικών αλγορίθμων σε υλικό που αναφέρονται στη βιβλιογραφία [97][98][99][100], η υλοποίηση που παρουσιάζεται εδώ λειτουργεί σε υψηλότερη συχνότητα ρολογιού και υλοποιεί περισσότερες της μίας μεθόδου διασταύρωσης και μετάλλαξης, οι οποίες μπορούν να αλλάξουν σε πραγματικό χρόνο κατά την εκτέλεση του αλγορίθμου. Τέλος, η παρούσα σχεδίασή κάνει χρήση περισσότερων παραμέτρων και αξιολογείται όχι μόνο μέσω της χρήσης συναρτήσεων σύγκρισης αλλά και με τη χρήση του προβλήματος του Πλανόδιου Πωλητή.

Κεφάλαιο 10

Συμπεράσματα και Περαιτέρω Έρευνα

Κύριος σκοπός της παρούσας διατριβής ήταν η ανάπτυξη αποδοτικών αρχιτεκτονικών ευφυών αλγορίθμων σε κυκλώματα προγραμματιζόμενης λογικής και η ενσωμάτωση αυτών σε κινητά ρομπότ, με τη βοήθεια γλωσσών περιγραφής και εργαλεία αυτοματοποίησης της σχεδίασης.

Στο πεδίο της ασαφούς λογικής υλοποιήθηκαν δύο παραμετρικοί πυρήνες ασαφών ελεγκτών. Οι πυρήνες είναι διαμορφώσιμοι ως προς τον αριθμό των εισόδων, εξόδων, συναρτήσεων συμμετοχής και τη μέθοδο συνάθροισης και συνεπαγωγής επιτρέποντας έτσι την προσαρμογή τους σε εφαρμογές με διαφορετικές απαιτήσεις χωρίς την ανάγκη επανασχεδίασης τους. Πιο συγκεκριμένα, η αρχιτεκτονική του ασαφούς ελεγκτή έχει τη δυνατότητα να επεξεργάζεται μόνο τους ενεργούς κανόνες, δηλαδή του κανόνες εκείνους που έχουν μη μηδενική συμβολή στο τελικό αποτέλεσμα, αυξάνοντας σημαντικά με αυτό τον τρόπο το ρυθμό επεξεργασίας δεδομένων του πυρήνα.

Η πρώτη αρχιτεκτονική υλοποίηση του πυρήνα στο ολοκληρωμένο κύκλωμα προγραμματιζόμενης λογικής, διαμορφωμένη για 4 εισόδους των 12-bit και 2401 κανόνες, επιτυγχάνει υψηλή συχνότητα λειτουργίας και ρυθμό επεξεργασίας δεδομένων 2^n για n αριθμό εισόδων του ασαφούς ελεγκτή.

Η δεύτερη αρχιτεκτονική υλοποίηση διαφοροποιείται υιοθετώντας μια τεχνική αύξησης του ρυθμού επεξεργασίας των δεδομένων στην αρχιτεκτονική του πυρήνα του ασαφούς ελεγκτή, που αναφέρεται ως τεχνική ODD-EVEN. Με την προαναφερθείσα τεχνική επιτυγχάνεται ρυθμός επεξεργασίας δεδομένων $2^n / 2$ για δύο εισόδους, με τον έλεγχο δύο ενεργών κανόνων σε κάθε κύκλο ρολογιού. Η μέθοδος ονομάστηκε ODD-EVEN διότι στηρίζεται στον ταυτόχρονο έλεγχο των κανόνων που αναφέρονται στο περιττό ενεργό ασαφές σύνολο της πρώτης εισόδου με αυτούς που αναφέρονται στο άρτιο ενεργό σύνολο της πρώτης

εισόδου. Η εξέταση δύο κανόνων ανά κύκλο ρολογιού επιτυγχάνεται διαχωρίζοντας τη μνήμη των κανόνων και τη μνήμη των παραμέτρων των ασαφών συνόλων της πρώτης εισόδου σε ODD και EVEN μέρη. Ο διαχωρισμός αυτός αυξάνει την παραλληλία τα αρχιτεκτονικής σχεδίασης με μικρή αύξηση της λογικής. Επιπλέον, για την αναπαράσταση των συναρτήσεων συμμετοχής παρουσιάστηκαν δύο εναλλακτικές υλοποιήσεις. Η πρώτη με αριθμητικό υπολογισμό των συναρτήσεων συμμετοχής και η δεύτερη στην οποία οι τιμές είναι προϋπολογισμένες και αποθηκεμένες σε πίνακα αναζήτησης.

Στη συνέχεια, ο πυρήνας του ασαφούς ελεγκτή εφαρμόστηκε στο έλεγχο ενός κινητού ρομπότ τύπου Pioneer P3-DX8. Συγκεκριμένα ο πυρήνας της πρώτης αρχιτεκτονικής ολοκληρώθηκε με έναν πυρήνα μαλακού επεξεργαστή τύπου Microblaze για να αποτελέσει ένα αυτόνομο σύστημα σε ψηφίδα (*System on a Chip* – SoC) για το πρόβλημα της παρακολούθησης πορείας (*path tracking*) σε κινητά ρομπότ. Η διαμορφώσιμη αρχιτεκτονική σχεδίαση του ασαφούς ελεγκτή επέτρεψε την εύκολη προσαρμογή του πυρήνα θέτοντας μόνο τις παραμέτρους στα χαρακτηριστικά του ασαφούς ελεγκτή για το πρόβλημα της παρακολούθησης πορείας, χωρίς την ανάγκη επανασχεδίασης του πυρήνα. Η ακρίβεια της ακολουθούμενης πορείας στα πειράματα που διεξήχθησαν επιβεβαιώθηκαν με την παράλληλη χρήση διαφορικού GPS για τη μέτρηση της πραγματικής πορείας του ρομπότ.

Εδώ αξίζει να σημειωθεί ότι επιπλέον πειράματα για το πρόβλημα της παρακολούθησης πορείας εκτελέστηκαν και με το ρομπότ Kephera II. Στα πειράματα αυτά η μέτρηση της πραγματικής θέσης του ρομπότ πραγματοποιήθηκε με επεξεργασία εικόνας από κάμερα [115].

Επιπλέον, παρουσιάστηκε η αρχιτεκτονική σχεδίαση και υλοποίηση ενός παραμετρικού πυρήνα γενετικού αλγορίθμου σε κύκλωμα προγραμματιζόμενης λογικής FPGA. Η διαμορφωσιμότητα του πυρήνα αφορά στις διάφορες παραμέτρους του γενετικού αλγορίθμου, όπως για παράδειγμα, το μέγεθος του πληθυσμού, το μέγιστο όριο γενεών εκτέλεσης, το μέγιστο όριο τιμής προσαρμογής, ο αριθμός "ελίτ" απογόνων, οι μέθοδοι διασταύρωσης και μετάλλαξης, κ.τ.λ. Η συγκεκριμένη υλοποίηση γενετικού αλγορίθμου επιτυγχάνει αρκετά υψηλή συχνότητα λειτουργίας και παρουσιάζει αξιοσημείωτη επιτάχυνση όταν συγκρίνεται με την αντίστοιχη υλοποίησή του σε λογισμικό. Η παρούσα σχεδίασή αξιολογήθηκε μέσω της χρήσης συναρτήσεων σύγκρισης και με την εφαρμογή του πυρήνα στην επίλυση του προβλήματος του Πλανόδιου Πωλητή για διαφορετικό αριθμό πόλεων.

Σαν προέκταση της διατριβής αυτής, προτείνονται να διερευνηθούν τα παρακάτω:

- Η αρχιτεκτονική του πυρήνα του ασαφούς ελεγκτή θα μπορούσε να διαμορφωθεί κατάλληλα ούτως ώστε να εισαχθεί ένας προσαρμοστικός αλγόριθμος που θα ανανεώνει τις παραμέτρους των συναρτήσεων συμμετοχής ή τους ασαφείς κανόνες σε πραγματικό χρόνο. Συγκεκριμένα ο πυρήνας του γενετικού αλγορίθμου που παρουσιάστηκε θα μπορούσε κάλλιστα να ενσωματωθεί κατάλληλα με τον πυρήνα του ασαφούς ελεγκτή για τη δημιουργία ενός πυρήνα προσαρμοστικού ασαφούς ελεγκτή.
- Ο πυρήνας προσαρμοστικού ασαφούς ελεγκτή θα μπορούσε να αποτελέσει ένα αυτόνομο σύστημα πραγματικού χρόνου και υψηλής ταχύτητας απόκρισης στον έλεγχο της ισορροπίας του ανάστροφου εκκρεμούς που είναι ένα πρότυπο

πρόβλημα αξιολόγησης αλγορίθμων ελέγχου σε μεγάλο εύρος διαταραχών και θέσεων ισορροπίας.

- Ο πυρήνας του γενετικού αλγορίθμου που υλοποιήθηκε μπορεί να επεκταθεί περαιτέρω ώστε να επιτευχθεί μεγαλύτερη συχνότητα λειτουργίας μέσω της μεγαλύτερης παραλληλοποίησης κάποιων διαδικασιών.

Η σελίδα αυτή αφέθηκε σκόπιμα κενή

This page was left blank intentionally

Παράρτημα Α

Ενδεικτικοί Κώδικες Προγραμμάτων

Παρακάτω παρατίθονται ενδεικτικά μέρος από τους κώδικες που αναπτύχθηκαν.

Α.1 Ασαφής Ελεγκτής

Η παρακάτω VHDL δομική περιγραφή συνδέει τα επιμέρους υποσυστήματα που συνθέτουν τον πυρήνα του ασαφούς ελεγκτή.

```
--------------------------------------------------------------------------------
-- Title      : Fuzzy Controller Core
-- Project    : Fuzzy Controller
--------------------------------------------------------------------------------
-- File       : fc.vhd
-- Author     : Kyriakos Deliparaschos (kdelip@mail.ntua.gr)
-- Company    : NTUA
-- Created    : 15/02/05
-- Last update: 17/01/06
-- Platform   : Modelsim, Synplify, ISE
-- Standard   : VHDL'93
--------------------------------------------------------------------------------
-- Description: This is the top structural block of the Fuzzy Controller (Core)
--
--              Model description:
--
--              Fuzzy controller Sugeno-Takagi type order 0 model
--              MISO type with 4 inputs, and 1 output
--
--              The FC parameters (generics) are explained below:
--
--              ip_no    -> number of crisp inputs
--              ip_sz    -> input bus width (all crisp ip's belong to the same
--                          Universe of Discourse)
--              op_sz    -> output bus width (crisp op Universe of Discourse)
--              FS_no    -> number of fuzzy sets (under the hypothesis that all
--                          crisp ip's have the same number of fuzzy sets)
--              dy       -> MF degree of truth op range (alpha value)
--              sel_op   -> 00 for min
--                          01 for prod
--                          10 for max
--                          11 for probor (probalistic OR or algebraic sum)
--              div_type -> Divider type selection
--
--              Similarly the IO Ports of the block are described below:
--
```

```
--              clk      -> external clk
--              clkx     -> internal clk (external clk * 2^ip_no)
--              rst_n    -> active high asynchronous reset
--              ip_set   -> crisp input set. A set is defined as a
--                          collection of a number of elements (in our
--                          case inputs), defined by the (ip_no) parameter
--              op_set   -> crisp output set
-------------------------------------------------------------------------------
-- Copyright (c) 2005 NTUA
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 15/02/05    1.1      kdelip  Created
-- 29/02/06    1.2      kdelip  Changed register instantiation names
--                              Added andor_meth_p component instantiation
--                              aggr_meth_sel -> meth_sel
--                              Fixed several width assignements
--                              Updated comments
-- 02/03/06    1.3      kdelip  This version uses component andor_meth_p
--                              Generics & Ports updated
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- LIBRARIES
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library work;
use work.arith_pkg.all;
use work.fc_pkg.all;

-------------------------------------------------------------------------------
-- ENTITY
-------------------------------------------------------------------------------
entity FC is
  generic (
    ip_no   : positive := ip_no;    -- number of crisp inputs
    ip_sz   : positive := ip_sz;    -- input bus width
    op_sz   : positive := op_sz;    -- output bus width
    FS_no   : positive := FS_no;    -- number of fuzzy sets
    dy      : positive := dy;       -- MF degree of truth op
    sel_op  : std_logic_vector(1 downto 0) := sel_op; -- AND/OR method selection
    -- Path Syncronization Register (psr) between signal paths
    psr1_no  : integer := psr1_no;  -- ip_set ->psr-> trap_gen_p
    psr2_no  : integer := psr2_no;  -- s_rom  ->psr-> mult
    psr3_no  : integer := psr3_no;  -- s_rom  ->psr-> rul_sel_p
    psr4_no  : integer := psr4_no;  -- cpr5   ->psr-> int_uns
    -- Component Pipeline Register
    cpr1_no  : integer := cpr1_no;  -- addr_gen_p
    cpr2_no  : integer := cpr2_no;  -- cons_map_p
    cpr3_no  : integer := cpr3_no;  -- trap_gen_p
    cpr4_no  : integer := cpr4_no;  -- rule_sel_p
    cpr5_no  : integer := cpr5_no;  -- minmax_p
    cpr6_no  : integer := cpr6_no;  -- mult
    cpr7_no  : integer := cpr7_no;  -- int_uns
    cpr8_no  : integer := cpr8_no; -- int_sig
    cpr9_no  : integer := cpr9_no); -- div
  port (
    --clk      : in  std_logic;       -- external clk
    clkx     : in  std_logic;       -- internal clk
    rst_n    : in  std_logic;       -- reset (active low)
    ip_set   : in  std_logic_vector(ip_no*ip_sz-1 downto 0); -- input set
    op       : out std_logic_vector(op_sz-1 downto 0));      -- output
end entity FC;

-------------------------------------------------------------------------------
-- ARCHITECTURE
-------------------------------------------------------------------------------
architecture str of FC is

-- SIGNAL DECLARATION
-------------------------------------------------------------------------------
  -- input register for trap_gen_p (signed)
  signal ip_set_d           : std_logic_vector(ip_no*ip_sz-1 downto 0);
  -- starting fuzzy set address (unsigned)
  signal FS_start_addr      : std_logic_vector(ip_no*log2(FS_no-1)-1 downto 0);
  -- generated addr (unsigned)
  signal gen_addr           : std_logic_vector(ip_no*log2(FS_no)-1 downto 0);
  -- input membership function (antecedent) parameters (concatenated)(unsigned)
  signal MF_ip_param        : std_logic_vector(ip_no*(2*ip_sz+2*dy)-1 downto 0);
  -- consequent memory address (unsigned)
  signal MF_op_val_addr     : std_logic_vector(log2(FS_no**ip_no)-1 downto 0);
  -- inf_r_sel & MF_op_param (unsigned)
  signal MF_op_val_data     : std_logic_vector(ip_no+ip_sz-1 downto 0);
  -- antecedents degree of truth (alpha values) (unsigned)
  signal alpha_val          : std_logic_vector(ip_no*dy-1 downto 0);
  -- active rule alpha value (unsigned)
  signal alpha_val_active   : std_logic_vector(ip_no*dy-1 downto 0);
  -- consequents degree of truth (theta values) or firing strength (unsigned)
  signal theta_val          : std_logic_vector(dy-1 downto 0);
  -- implication result (signed)
  signal impl_val           : std_logic_vector(theta_val'length+1+ip_sz-1 downto 0);
  -- divisor (unsigned)
  signal divs               : std_logic_vector(ip_no+theta_val'length -1 downto 0);
  -- divident (signed)
  signal divd               : std_logic_vector(ip_no+impl_val'length -1 downto 0);
  -- quotient
  signal q                  : std_logic_vector(divd'length-divs'length downto 0);
```

```
  -- remainder
  signal r                 : std_logic_vector(divs'length-1 downto 0);
  signal op_cat            : std_logic_vector(q'length+r'length-1 downto 0);
  -- signal for zeroing integrators (unsigned)
  signal int_zer           : std_logic;
  -- pipeline signals
  signal gen_addr_p        : std_logic_vector(gen_addr'left downto 0);
  signal MF_op_val_addr_p  : std_logic_vector(MF_op_val_addr'left downto 0);
  signal MF_op_param_p     : std_logic_vector(ip_sz-1 downto 0);
  signal inf_r_sel_p       : std_logic_vector(ip_no-1 downto 0);
  signal alpha_val_p       : std_logic_vector(alpha_val'left downto 0);
  signal alpha_val_active_p : std_logic_vector(alpha_val_active'left downto 0);
  signal theta_val_p       : std_logic_vector(theta_val'left downto 0);
  signal theta_val_p_s     : std_logic_vector(theta_val'left downto 0);
  signal impl_val_p        : std_logic_vector(impl_val'left downto 0);
  signal divs_p            : std_logic_vector(divs'left downto 0);
  signal divd_p            : std_logic_vector(divd'left downto 0);
  signal op_c              : std_logic_vector(op_sz-1 downto 0);

begin

-- COMPONENT instantiation

  -- Active rule selector
  U0 : ars_p -- U_ars
    generic map (
      ip_no => ip_no,
      ip_sz => ip_sz,
      FS_no => FS_no)
    port map (
      ip_data       => ip_set,
      FS_start_addr => FS_start_addr);

  -- Addrgen
  U1 : addr_gen_p -- U_addr_gen
    generic map (
      ip_no            => ip_no,
      FS_no            => FS_no,
      int_zer_delay => 3)
    port map (
      clk        => clkx,
      rst_n      => rst_n,
      ip_data    => FS_start_addr,
      gen_addr   => gen_addr,
      int_zer => int_zer);

  -- Rule selector
  U2 : rule_sel_p -- U_rule_sel
    generic map (
      ip_no => ip_no,
      dy    => dy)
    port map (
      sel     => inf_r_sel_p,
      ip_data => alpha_val_p,
      op_data => alpha_val_active);

  U3 : andor_meth_p
    generic map(
      ip_no => ip_no,
      sel_op => sel_op,
      n     => dy)
    port map (
      ip_data => alpha_val_active_p,
      op_data => theta_val);

  -- Memmory mapper
  U4 : cons_map_p -- U_mem_map
    generic map (
      ip_no => ip_no,
      FS_no => FS_no)
    port map (
      addr_in  => gen_addr_p,
      addr_out => MF_op_val_addr);

  -- Singletons rom
  U5 : s_rom_p -- U_s_rom
    generic map (
      ip_no => ip_no,
      ip_sz => ip_sz,
      FS_no => FS_no)
    port map (
      Addr => MF_op_val_addr_p,
      Data => MF_op_val_data);

  -- Trapezoidal membership function parameters rom
  U6 : mf_rom_p -- U_mf_rom
    generic map (
      ip_no => ip_no,
      ip_sz => ip_sz,
      dy    => dy,
      FS_no => FS_no)
    port map (
      Addr => gen_addr_p,
      Data => MF_ip_param);

  -- Alpha value computation (MFtrapGen)
  U7 : trap_gen_p -- U_trap_gen
    generic map (
      ip_no       => ip_no,
      ip_sz       => ip_sz,
```

```
    dy          => dy,
    slope_trunc => 8)
  port map (
    ip_data   => ip_set_d,
    MF_param  => MF_ip_param,
    alpha_val => alpha_val);

-- Implication value computation (mult)
U8 : mult -- U_mult
  generic map (
    N_unsigned => theta_val_p'length,
    N_signed   => MF_op_param_p'length)
  port map (
    x_unsigned => theta_val_p,
    x_signed   => MF_op_param_p,
    y          => impl_val);

-- unsigned Integrator (This analogous to MAXing)
U9 : int_uns -- U_int_uns
  generic map (
    ip_no => ip_no,
    ip_sz => theta_val_p'length)
  port map (
    clk   => clkx,
    rst_n => rst_n,
    x     => theta_val_p_s,
    clear => int_zer,
    y     => divs);

-- signed Integrator
U10 : int_sig -- U_int
  generic map (
    ip_no => ip_no,
    ip_sz => impl_val_p'length)
  port map (
    clk   => clkx,
    rst_n => rst_n,
    x     => impl_val_p,
    clear => int_zer,
    y     => divd);

-- divider
U11 : if div_type = 1 generate -- U_div_ppa
  div_ppa_m : div_ppa
    generic map (
      divd_sz => divd'length,
      divs_sz => divs'length,
      op_sz   => op_sz)
    port map (
      divs => divs_p,
      divd => divd_p,
      op   => op_c);

end generate U11;

U12 : if div_type = 0 generate -- U_div_array
  div_array_m : div_array
    generic map (
      widthX => divd'length,
      widthY => divs'length)
    port map (
      X  => divd_p,
      Y  => divs_p,
      q => q,
      r => r);
end generate U12;

-- Path synchronization (due to blocks latency)
--------------------------------------------------------------------------------
-- Add registers to crisp input set (ip_set) to synchronize with mf_rom_p op
PSR1 : delay_regs -- U_delay_regs
  generic map (
    width   => ip_set'length,
    latency => psr1_no)
  port map(
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => ip_set,
    out_data => ip_set_d);

-- Add registers to output of s_rom_p to synchronize with minmax_p op
PSR2 : delay_regs -- PIPE_MF_op_param
  generic map (
    width   => MF_op_param_p'length,
    latency => psr2_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => MF_op_val_data(ip_sz-1 downto 0),
    out_data => MF_op_param_p);

-- Add registers to output of s_rom_p to syncronize with MFtrapGen_p op
PSR3 : delay_regs -- PIPE_inf_r_sel
  generic map (
    width   => inf_r_sel_p'length,
    latency => psr3_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => MF_op_val_data(MF_op_val_data'left downto ip_sz),
```

```
    out_data => inf_r_sel_p);

--!!This could be improved by delaying the int_zer signal instead to save flipflops!!
-- Add registers to output of CPR5 to syncronize with mult op
PSR4 : delay_regs
  generic map (
    width   => theta_val_p'length,
    latency => psr4_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => theta_val_p,
    out_data => theta_val_p_s);


-- Component pipelining
--------------------------------------------------------------------------------
CPR1 : delay_regs  -- PIPE_gen_addr
  generic map (
    width   => gen_addr'length,
    latency => cpr1_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => gen_addr,
    out_data => gen_addr_p);

CPR2 : delay_regs -- PIPE_MF_op_val_addr
  generic map (
    width   => MF_op_val_addr'length,
    latency => cpr2_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => MF_op_val_addr,
    out_data => MF_op_val_addr_p);

CPR3 : delay_regs -- PIPE_alpha_val
  generic map (
    width   => alpha_val'length,
    latency => cpr3_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => alpha_val,
    out_data => alpha_val_p);

CPR4 : delay_regs -- PIPE_alpha_val_active
  generic map (
    width   => alpha_val_active'length,
    latency => cpr4_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => alpha_val_active,
    out_data => alpha_val_active_p);

CPR5 : delay_regs -- PIPE_theta_val
  generic map (
    width   => theta_val'length,
    latency => cpr5_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => theta_val,
    out_data => theta_val_p);

CPR6 : delay_regs -- PIPE_impl_val
  generic map (
    width   => impl_val'length,
    latency => cpr6_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => impl_val,
    out_data => impl_val_p);

CPR7 : delay_regs -- PIPE_divs
  generic map (
    width   => divs'length,
    latency => cpr7_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => divs,
    out_data => divs_p);

CPR8 : delay_regs -- PIPE_divd
  generic map (
    width   => divd'length,
    latency => cpr8_no)
  port map (
    clk      => clkx,
    rst_n    => rst_n,
    in_data  => divd,
    out_data => divd_p);

CPR9 : if div_type = 1 generate
CPR9_i : delay_regs
  generic map (
```

```
      width   => op_sz,
      latency => cpr9_no)
    port map (
      clk      => clkx,
      rst_n    => rst_n,
      in_data  => op_c,
      out_data => op);
  end generate;

array_sig : if div_type = 0 generate -- U_div_array
                    op_cat <= q & r;
                    op <= op_cat(op_cat'left downto op_cat'length-ip_sz);
end generate;

end architecture str;
```

A.2 Βελτιστοποιημένος Ασαφής Ελεγκτής

Η παρακάτω δομική περιγραφή συνδέει τα επιμέρους υποσυστήματα που συνθέτουν τον πυρήνα του βελτιστοποιημένου ασαφούς ελεγκτή.

```
--------------------------------------------------------------------------------
-- Title      : Fuzzy Controller Core
-- Project    : Fuzzy Controller (ODD-EVEN)
--------------------------------------------------------------------------------
-- File       : FC.vhd
-- Author     : Kyriakos Deliparaschos
-- Company    : NTUA/IRAL
-- Created    : 05/06/2004
-- Platform   : Modelsim 5.4, Synplify Pro 7.5, ISE 6.3
-- Standard   : VHDL'93
--------------------------------------------------------------------------------
-- Description: This the top structural description of the FC core
--------------------------------------------------------------------------------
-- Copyright (c) 2004 NTUA
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- LIBRARIES
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library work;
use work.FPGA_FC_pkg.all;

--------------------------------------------------------------------------------
-- ENTITY
--------------------------------------------------------------------------------
entity FC is
  generic (
    aggr_sel  : boolean  := aggr_sel; -- aggregation method
    ip_no     : positive := ip_no; -- number of crisp inputs
    ip_sz     : positive := ip_sz; -- input bit length
    op_sz     : positive := op_sz; -- output bit length (DAC size)
    FS_no     : positive := FS_no; -- number of fuzzy sets for each input
    dy        : positive := dy); -- MF degree of truth output
  port (
    clk       : in  std_logic; -- internal clk
    rst_n     : in  std_logic; -- reset (active low)
    ip_bus    : in  std_logic_vector(ip_no*ip_sz-1 downto 0); -- input set
    op        : out std_logic_vector(op_sz-1 downto 0)); -- fuzzy output
end entity FC;

--------------------------------------------------------------------------------
-- ARCHITECTURE
--------------------------------------------------------------------------------
architecture structural of FC is

-- SIGNAL DECLARATION
--------------------------------------------------------------------------------
  signal clear_int         : std_logic;
  signal theta_active_odd  : std_logic_vector(dy-1 downto 0);
  signal theta_active_even : std_logic_vector(dy-1 downto 0);
  signal cns_odd           : std_logic_vector(ip_sz-1 downto 0);
  signal cns_even          : std_logic_vector(ip_sz-1 downto 0);

  -- pipeline signals
  signal clear_int_p         : std_logic;
  signal theta_active_odd_p  : std_logic_vector(theta_active_odd'left downto 0);
  signal theta_active_even_p : std_logic_vector(theta_active_even'left downto 0);
  signal cns_odd_p           : std_logic_vector(cns_odd'left downto 0);
  signal cns_even_p          : std_logic_vector(cns_even'left downto 0);

begin -- structural

-- COMPONENTS INSTANTIATION
--------------------------------------------------------------------------------

  U_Fuz_Aggr: Fuz_Aggr
    generic map (
```

```
      ip_no => ip_no,
      ip_sz => ip_sz,
      FS_no => FS_no,
      dy    => dy)
    port map (
      clk               => clk,
      rst_n             => rst_n,
      ip_bus            => ip_bus,
      clear_int         => clear_int,
      theta_active_odd  => theta_active_odd,
      theta_active_even => theta_active_even,
      cns_odd           => cns_odd,
      cns_even          => cns_even);

  U_Inf_Defuz: Inf_Defuz
    generic map (
      aggr_sel => aggr_sel,
      ip_no    => ip_no,
      ip_sz    => ip_sz,
      op_sz    => op_sz,
      dy       => dy)
    port map (
      clk               => clk,
      rst_n             => rst_n,
      clear_int         => clear_int_p,
      theta_active_odd  => theta_active_odd_p,
      theta_active_even => theta_active_even_p,
      cns_odd           => cns_odd_p,
      cns_even          => cns_even_p,
      op                => op);

-- COMPONENT PIPELINING
--------------------------------------------------------------------------------
  PIPE_clear_int : delay_unit
    generic map (
      clk_EN  => 0,
      edge    => 0,
      width   => 1,
      latency => 1)
    PORT MAP (
      clk      => clk,
      rst_n    => rst_n,
      enable   => '0',
      in_data  => clear_int,
      out_data => clear_int_p);

  PIPE_theta_active_odd : delay_regs
    generic map (
      clk_EN  => 0,
      edge    => 0,
      width   => theta_active_odd'length,
      latency => 1)
    PORT MAP (
      clk      => clk,
      rst_n    => rst_n,
      enable   => '0',
      in_data  => theta_active_odd,
      out_data => theta_active_odd_p);

  PIPE_theta_active_even : delay_regs
    generic map (
      clk_EN  => 0,
      edge    => 0,
      width   => theta_active_even'length,
      latency => 1)
    PORT MAP (
      clk      => clk,
      rst_n    => rst_n,
      enable   => '0',
      in_data  => theta_active_even,
      out_data => theta_active_even_p);

  PIPE_cns_odd : delay_regs
    generic map (
      clk_EN  => 0,
      edge    => 0,
      width   => cns_odd'length,
      latency => 1)
    PORT MAP (
      clk      => clk,
      rst_n    => rst_n,
      enable   => '0',
      in_data  => cns_odd,
      out_data => cns_odd_p);

  PIPE_cns_even : delay_regs
    generic map (
      clk_EN  => 0,
      edge    => 0,
      width   => cns_even'length,
      latency => 1)
    PORT MAP (
      clk      => clk,
      rst_n    => rst_n,
      enable   => '0',
      in_data  => cns_even,
      out_data => cns_even_p);


end architecture structural;
```

```
--------------------------------------------------------------------------------
-- Title      : FPGA Fuzzy Controller Package
-- Project    : Fuzzy Controller
--------------------------------------------------------------------------------
-- File       : FPGA_FC_pkg.vhd
-- Author     : Kyriakos Deliparaschos
-- Company    : NTUA/IRAL
-- Created    : 05/06/2004
-- Platform   : Modelsim 5.4, Synplify Pro 7.5, ISE 6.3
-- Standard   : VHDL'93
--------------------------------------------------------------------------------
-- Description: Package definition file
--------------------------------------------------------------------------------
-- Copyright (c) 2004 NTUA
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- LIBRARIES
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

--------------------------------------------------------------------------------
-- PACKAGE DECLARATION
--------------------------------------------------------------------------------
package FPGA_FC_pkg is

-- CONSTANTS DECLARATION
--------------------------------------------------------------------------------
  constant aggr_sel  : boolean  := false; -- aggregation method select (min/prod)
  constant ip_no     : positive := 2; -- number of crisp inputs
  constant ip_sz     : positive := 8; -- input bit length
  constant op_sz     : positive := 12; -- output bit length (DAC size)
  constant FS_no     : positive := 7; -- number of fuzzy sets for each input
  constant dy        : positive := 4; -- MF degree of truth output
  constant slp_no    : positive := 4; -- fixed slopes repertoire
  constant slp_sz    : positive := 4; -- slope bit length
  constant slp_trunc : positive := 3; -- slope truncation

-- FUNCTIONS DECLARATION
--------------------------------------------------------------------------------
  function log2 (no : in integer) return integer;

-- TYPES DECLARATION
--------------------------------------------------------------------------------
  subtype ARSrom_wordsize_r is std_logic_vector(ip_sz-1 downto 0);
  type    ARSrom_contents_r is array(0 to ip_no*(FS_no-2)-1) of ARSrom_wordsize_r;

  subtype ARSrom_wordsize_f is std_logic_vector(ip_sz-1 downto 0);
  type    ARSrom_contents_f is array(0 to ip_no*(FS_no-1)-1) of ARSrom_wordsize_f;

  subtype MFrom_wordsize_slop is std_logic_vector(slp_sz-1 downto 0);
  type    MFrom_contents_slop is array(0 TO (ip_no+1)*slp_no-1) of MFrom_wordsize_slop;

-- COMPONENTS DECLARATION
--------------------------------------------------------------------------------
  component DCM_XC3S200_VQ100_5_CLK0_CLKFX2 is
    port (
      CLKIN_IN   : in    std_logic;
      RST_IN     : in    std_logic;
      CLK2X_OUT  : out   std_logic;
      CLK0_OUT   : out   std_logic;
      LOCKED_OUT : out   std_logic);
  end component;

  component delay_regs
    generic (
      clk_EN   : integer;
      edge     : integer;
      width    : integer;
      latency  : integer);
    port(
      clk      : in  std_logic;
      rst_n    : in  std_logic;
      enable   : in  std_logic;
      in_data  : in  std_logic_vector(width-1 downto 0);
      out_data : out std_logic_vector(width-1 downto 0));
  end component;

  component delay_unit is
    generic (
      clk_EN  : integer;
      edge    : integer;
      width   : integer;
      latency : integer);
    port (
      clk      : in  std_logic;
      rst_n    : in  std_logic;
      enable   : in std_logic;
      in_data  : in  std_logic;
      out_data : out std_logic);
  end component;

  component ARS is
    generic (
      ip_no : integer;
      ip_sz : integer;
      FS_no : integer);
```

```
  port (
    ip_bus  : in  std_logic_vector(ip_no*ip_sz-1 downto 0);
    ars_bus : out std_logic_vector(ip_no*2*log2(FS_no)-1 downto 0);
    reg_bus : out std_logic_vector(ip_no*2-1 downto 0));
end component;

component AddrGen is
  generic (
    ip_no     : integer;
    sz0       : integer;
    sz1       : integer;
    clr_delay : integer);
  port (
    clk       : in  std_logic;
    rst_n     : in  std_logic;
    x2_0_bus  : in  std_logic_vector(ip_no*2*sz0-1 downto 0);
    x2_1_bus  : in  std_logic_vector(ip_no*2*sz1-1 downto 0);
    x1_0_bus  : out std_logic_vector((ip_no+1)*sz0-1 downto 0);
    x1_1_bus  : out std_logic_vector((ip_no+1)*sz1-1 downto 0);
    clr_int   : out std_logic);
end component;

component MFProm is
  generic (
    fs     : boolean;
    ip_no  : integer;
    ip0_sz : integer;
    ip1_sz : integer;
    op0_sz : integer;
    op1_sz : integer);
  port (
    ip0 : in  std_logic_vector(ip_no*ip0_sz-1 downto 0);
    ip1 : in  std_logic_vector(ip_no*ip1_sz-1 downto 0);
    op0 : out std_logic_vector(ip_no*op0_sz-1 downto 0);
    op1 : out std_logic_vector(ip_no*op1_sz-1 downto 0));
end component;

component MFrom is
  generic (
    ip_no : integer;
    ip_sz : integer;
    op_sz : integer);
  port (
    ip : in  std_logic_vector(ip_no*ip_sz-1 downto 0);
    op : out std_logic_vector(ip_no*op_sz-1 downto 0));
end component;

component cns_Mapper
  generic (
    ip_no : integer;
    FS_no : integer);
  port (
    addr_gen_bus : in  std_logic_vector((ip_no+1)*log2(FS_no)-1 downto 0);
    sromip_odd   : out std_logic_vector(log2(FS_no**ip_no)-1 downto 0);
    sromip_even  : out std_logic_vector(log2(FS_no**ip_no)-1 downto 0));
end component;

component Dif_signed is
  generic (
    ip_no : integer;
    ip_sz : integer);
  port (
    ip0_bus : in  std_logic_vector(ip_no*ip_sz-1 downto 0);
    ip1_bus : in  std_logic_vector(ip_no*ip_sz-1 downto 0);
    op_bus  : out std_logic_vector(ip_no*(ip_sz+1)-1 downto 0));
end component;

component S_ROM is
  generic (
    ip_sz  : integer;
    op0_sz : integer;
    op1_sz : integer);
  port (
    ip0      : in  std_logic_vector(ip_sz-1 downto 0);
    ip1      : in  std_logic_vector(ip_sz-1 downto 0);
    op0_odd  : out std_logic_vector(op0_sz-1 downto 0);
    op0_even : out std_logic_vector(op0_sz-1 downto 0);
    op1_odd  : out std_logic_vector(op1_sz-1 downto 0);
    op1_even : out std_logic_vector(op1_sz-1 downto 0));
end component;

component MFmult is
  generic (
    fs     : boolean;
    ip_no  : integer;
    ip0_sz : integer;
    ip1_sz : integer;
    op_sz  : integer;
    slp_no : integer);
  port (
    clk     : in  std_logic;
    rst_n   : in  std_logic;
    ip0_bus : in  std_logic_vector(ip_no*ip0_sz-1 downto 0);
    ip1_bus : in  std_logic_vector(ip_no*ip1_sz-1 downto 0);
    op_bus  : out std_logic_vector(ip_no*op_sz-1 downto 0));
end component;

component FIXalpha is
  generic (
    ip_no     : integer;
```

```
    ip_sz     : integer;
    op_sz     : integer;
    slp_trunc : integer);
  port (
    reg_gen_bus : in  std_logic_vector(ip_no-1 downto 0);
    ip_bus      : in  std_logic_vector(ip_no*ip_sz-1 downto 0);
    op_bus      : out std_logic_vector(ip_no*op_sz-1 downto 0));
end component;

component RuleSelector is
  generic (
    ip_no : integer;
    dy    : integer);
  port (
    alpha_bus  : in  std_logic_vector(ip_no*dy-1 downto 0);
    active_sel : in  std_logic_vector(ip_no-1 downto 0);
    active_bus : out std_logic_vector(ip_no*dy-1 downto 0));
end component;

component Minimum is
  generic (
    ip_no : integer;
    dy    : integer);
  port (
    active_bus : in  std_logic_vector(ip_no*dy-1 downto 0);
    theta      : out std_logic_vector(dy-1 downto 0));
end component;

component mult IS
  GENERIC (
    pipe        : boolean;
    N_unsigned  : integer;
    N_signed    : integer);
  PORT (
    clk        : in  std_logic;
    rst_n      : in  std_logic;
    x_unsigned : in  std_logic_vector(N_unsigned-1 downto 0);
    x_signed   : in  std_logic_vector(N_signed-1 downto 0);
    y          : out std_logic_vector(N_unsigned+N_signed downto 0));
end component;

component Adder_unsigned is
  generic (
    ip0_sz : positive;
    ip1_sz : positive;
    op_sz  : positive);
  port (
    ip0 : in  std_logic_vector(ip0_sz-1 downto 0);
    ip1 : in  std_logic_vector(ip1_sz-1 downto 0);
    op  : out std_logic_vector(op_sz-1 downto 0));
end component;

component Adder_signed is
  generic (
    ip0_sz : positive;
    ip1_sz : positive;
    op_sz  : positive);
  port (
    ip0 : in  std_logic_vector(ip0_sz-1 downto 0);
    ip1 : in  std_logic_vector(ip1_sz-1 downto 0);
    op  : out std_logic_vector(op_sz-1 downto 0));
end component;

component INT_signed is
  generic (
    ip_sz : integer;
    op_sz : integer);
  port (
    clk   : in  std_logic;
    rst_n : in  std_logic;
    clear : in  std_logic;
    ip    : in  std_logic_vector(ip_sz-1 downto 0);
    op    : out std_logic_vector(op_sz-1 downto 0));
end component;

component INT_unsigned is
  generic (
    ip_sz : integer;
    op_sz : integer);
  port (
    clk   : in  std_logic;
    rst_n : in  std_logic;
    clear : in  std_logic;
    ip    : in  std_logic_vector(ip_sz-1 downto 0);
    op    : out std_logic_vector(op_sz-1 downto 0));
end component;

component reciprocal_lut is
  generic (
    ip_sz : integer;
    op_sz : integer);
  port (
    Addr : in  std_logic_vector(ip_sz-1 downto 0);
    Data : out std_logic_vector(op_sz-1 downto 0));
end component;

component is_one is
  generic (
    ip_sz  : positive);
  port (
```

```
      ip : in  std_logic_vector(ip_sz-1 downto 0);
      op : out std_logic);
  end component;

  component FIXoutput is
    generic (
      exep_sz  : positive;
      divd_sz  : positive;
      recip_sz : positive;
      op_sz    : positive);
    port (
      flag_exep : in  std_logic;
      op_exep   : in  std_logic_vector(exep_sz-1 downto 0);
      op_tmp    : in  std_logic_vector(divd_sz+recip_sz downto 0);
      op        : out std_logic_vector(op_sz-1 downto 0));
  end component;

  component Fuz_Aggr is
    generic (
      ip_no : positive;
      ip_sz : positive;
      FS_no : positive;
      dy    : positive);
    port (
      clk              : in  std_logic;
      rst_n            : in  std_logic;
      ip_bus           : in  std_logic_vector(ip_no*ip_sz-1 downto 0);
      clear_int        : out std_logic;
      theta_active_odd  : out std_logic_vector(dy-1 downto 0);
      theta_active_even : out std_logic_vector(dy-1 downto 0);
      cns_odd          : out std_logic_vector(ip_sz-1 downto 0);
      cns_even         : out std_logic_vector(ip_sz-1 downto 0));
  end component;

  component Inf_Defuz is
    generic (
      aggr_sel : boolean;
      ip_no    : positive;
      ip_sz    : positive;
      op_sz    : positive;
      dy       : positive);
    port (
      clk              : in  std_logic;
      rst_n            : in  std_logic;
      clear_int        : in  std_logic;
      theta_active_odd  : in  std_logic_vector(dy-1 downto 0);
      theta_active_even : in  std_logic_vector(dy-1 downto 0);
      cns_odd          : in  std_logic_vector(ip_sz-1 downto 0);
      cns_even         : in  std_logic_vector(ip_sz-1 downto 0);
      op               : out std_logic_vector(op_sz-1 downto 0));
  end component;

  component FC is
    generic (
      aggr_sel : boolean;
      ip_no    : positive;
      ip_sz    : positive;
      op_sz    : positive;
      FS_no    : positive;
      dy       : positive);
    port (
      clk    : in  std_logic;
      rst_n  : in  std_logic;
      ip_bus : in  std_logic_vector(ip_no*ip_sz-1 downto 0);
      op     : out std_logic_vector(op_sz-1 downto 0));
  end component;

-- ROMS DECLARATION
-------------------------------------------------------------------------------
  constant ARSrom_rise : ARSrom_contents_r := ARSrom_contents_r'(
-- rise_start (we don't need the first two)
  conv_std_logic_vector(-78,ARSrom_wordsize_r'length),  -- ip0
  conv_std_logic_vector(-38,ARSrom_wordsize_r'length),
  conv_std_logic_vector(-2,ARSrom_wordsize_r'length),
  conv_std_logic_vector(43,ARSrom_wordsize_r'length),
  conv_std_logic_vector(87,ARSrom_wordsize_r'length),
  conv_std_logic_vector(-81,ARSrom_wordsize_r'length),  -- ip1
  conv_std_logic_vector(-30,ARSrom_wordsize_r'length),
  conv_std_logic_vector(-7,ARSrom_wordsize_r'length),
  conv_std_logic_vector(43,ARSrom_wordsize_r'length),
  conv_std_logic_vector(75,ARSrom_wordsize_r'length));


  constant ARSrom_fall : ARSrom_contents_f := ARSrom_contents_f'(
-- fall_start (we don't need the last one)
  conv_std_logic_vector(-110,ARSrom_wordsize_f'length),  -- ip0
  conv_std_logic_vector(-65,ARSrom_wordsize_f'length),
  conv_std_logic_vector(-48,ARSrom_wordsize_f'length),
  conv_std_logic_vector(-8,ARSrom_wordsize_f'length),
  conv_std_logic_vector(43,ARSrom_wordsize_f'length),
  conv_std_logic_vector(100,ARSrom_wordsize_f'length),
  conv_std_logic_vector(-110,ARSrom_wordsize_f'length),  -- ip1
  conv_std_logic_vector(-65,ARSrom_wordsize_f'length),
  conv_std_logic_vector(-48,ARSrom_wordsize_f'length),
  conv_std_logic_vector(-8,ARSrom_wordsize_f'length),
  conv_std_logic_vector(43,ARSrom_wordsize_f'length),
  conv_std_logic_vector(105,ARSrom_wordsize_f'length));

  constant MFrom_slop : MFrom_contents_slop := MFrom_contents_slop'(
-- fixed slope repertoire  (2ip)
```

```
  conv_std_logic_vector(0,MFrom_wordsize_slop'length),  -- ip0
  conv_std_logic_vector(5,MFrom_wordsize_slop'length),
  conv_std_logic_vector(8,MFrom_wordsize_slop'length),
  conv_std_logic_vector(14,MFrom_wordsize_slop'length),
  conv_std_logic_vector(0,MFrom_wordsize_slop'length),  -- ip0
  conv_std_logic_vector(5,MFrom_wordsize_slop'length),
  conv_std_logic_vector(8,MFrom_wordsize_slop'length),
  conv_std_logic_vector(14,MFrom_wordsize_slop'length),
  conv_std_logic_vector(0,MFrom_wordsize_slop'length),  -- ip1
  conv_std_logic_vector(5,MFrom_wordsize_slop'length),
  conv_std_logic_vector(8,MFrom_wordsize_slop'length),
  conv_std_logic_vector(14,MFrom_wordsize_slop'length));

end package FPGA_FC_pkg;

-------------------------------------------------------------------------------
-- PACKAGE BODY DECLARATION
-------------------------------------------------------------------------------
package body FPGA_FC_pkg is

-- log2 function
  function log2(no : integer) return integer is
    variable i : integer;
  begin
    i := 1;
    for val in 1 to no-1 loop
      if val = 2**i then
        i := i+1;
      end if;
    end loop;
    return i;
  end log2;

end package body FPGA_FC_pkg;
```

Ακολουθεί η RTL περιγραφή της μονάδας παραγωγής διευθύνσεων.

```
-------------------------------------------------------------------------------
-- Title      : Address Generator
-- Project    : Fuzzy Controller (ODD-EVEN)
-------------------------------------------------------------------------------
-- File       : AddrGen.vhd
-- Author     : Kyriakos Deliparaschos
-- Company    : NTUA/IRAL
-- Created    : 05/06/2004
-- Platform   : Modelsim 5.4, Synplify Pro 7.5, ISE 6.3
-- Standard   : VHDL'93
-------------------------------------------------------------------------------
-- Description: Parameterized RTL description of the Address Generator block
-------------------------------------------------------------------------------
-- Copyright (c) 2004 NTUA
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- LIBRARIES
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-------------------------------------------------------------------------------
-- ENTITY
-------------------------------------------------------------------------------
entity AddrGen is
  generic (
    ip_no     : integer := 2;
    sz0       : integer := 6;
    sz1       : integer := 4;
    clr_delay : integer := 1);
  port (
    clk       : in  std_logic;
    rst_n     : in  std_logic;
    x2_0_bus  : in  std_logic_vector(ip_no*2*sz0-1 downto 0);
    x2_1_bus  : in  std_logic_vector(ip_no*2*sz1-1 downto 0);
    x1_0_bus  : out std_logic_vector((ip_no+1)*sz0-1 downto 0);
    x1_1_bus  : out std_logic_vector((ip_no+1)*sz1-1 downto 0);
    clr_int   : out std_logic);
end entity AddrGen;

-------------------------------------------------------------------------------
-- ARCHITECTURE
-------------------------------------------------------------------------------
architecture rtl of AddrGen is

begin -- architecture rtl

COUNTER: if (ip_no > 2) generate

  signal CNS           : std_logic_vector(ip_no-2 downto 0);
  signal counter_bus   : std_logic_vector(ip_no-2 downto 0);
  signal counter_bus_i : std_logic_vector(ip_no-2 downto 0);

begin -- generate

  A: for i in 1 to ip_no-1 generate
```

```
  signal sel : std_logic;
  signal x2_0 : std_logic_vector(2*sz0-1 downto 0);
  signal x2_1 : std_logic_vector(2*sz1-1 downto 0);
  signal x1_0 : std_logic_vector(sz0-1 downto 0);
  signal x1_1 : std_logic_vector(sz1-1 downto 0);

  begin -- generate

  -- input data select
    sel <= counter_bus(i-1);
    x2_0 <= x2_0_bus((i+1)*x2_0'length -1 downto i*x2_0'length);
    x2_1 <= x2_1_bus((i+1)*x2_1'length -1 downto i*x2_1'length);

  -- select
    x1_0 <= x2_0(x1_0'left downto 0) when sel = '0' else
            x2_0(x2_0'left downto x1_0'length);
    x1_1 <= x2_1(x1_1'left downto 0) when sel = '0' else
            x2_1(x2_1'left downto x1_1'length);

  -- output
    x1_0_bus((i+2)*x1_0'length -1 downto (i+1)*x1_0'length) <= x1_0;
    x1_1_bus((i+2)*x1_1'length -1 downto (i+1)*x1_1'length) <= x1_1;

  end generate;

  -- direct output
    x1_0_bus(2*sz0-1 downto 0) <= x2_0_bus(2*sz0-1 downto 0);
    x1_1_bus(2*sz1-1 downto 0) <= x2_1_bus(2*sz1-1 downto 0);

  -- counter circular incrementation
    counter_bus <= counter_bus_i + '1';
    incr_proc : process (clk,rst_n)
      begin -- process select
        if (rst_n = '0') then
                                    counter_bus_i <= (others => '0');
        elsif rising_edge(clk) then
          counter_bus_i <= counter_bus;
        end if;
    end process incr_proc;

  -- constant
    CNS <= conv_std_logic_vector(clr_delay,counter_bus_i'length);

  -- clear integrators signal
    clr_int <= '1' when counter_bus_i = CNS else '0';

end generate;


--------------------------------------------------------------------------------


INVERTER: if (ip_no = 2) generate

  signal CNS           : std_logic;
  signal counter_bus   : std_logic;
  signal counter_bus_i : std_logic;

begin -- generate

  A: for i in 1 to ip_no-1 generate

  signal sel : std_logic;
  signal x2_0 : std_logic_vector(2*sz0-1 downto 0);
  signal x2_1 : std_logic_vector(2*sz1-1 downto 0);
  signal x1_0 : std_logic_vector(sz0-1 downto 0);
  signal x1_1 : std_logic_vector(sz1-1 downto 0);

  begin -- generate

  -- input data select
    sel <= counter_bus;
    x2_0 <= x2_0_bus((i+1)*x2_0'length -1 downto i*x2_0'length);
    x2_1 <= x2_1_bus((i+1)*x2_1'length -1 downto i*x2_1'length);

  -- select
    x1_0 <= x2_0(x1_0'left downto 0) when sel = '0' else
            x2_0(x2_0'left downto x1_0'length);
    x1_1 <= x2_1(x1_1'left downto 0) when sel = '0' else
            x2_1(x2_1'left downto x1_1'length);

  -- output
    x1_0_bus((i+2)*x1_0'length -1 downto (i+1)*x1_0'length) <= x1_0;
    x1_1_bus((i+2)*x1_1'length -1 downto (i+1)*x1_1'length) <= x1_1;

  end generate;

  -- direct output
    x1_0_bus(2*sz0-1 downto 0) <= x2_0_bus(2*sz0-1 downto 0);
    x1_1_bus(2*sz1-1 downto 0) <= x2_1_bus(2*sz1-1 downto 0);

  -- counter circular incrementation
    counter_bus <= not counter_bus_i;
    incr_proc : process (clk,rst_n)
      begin -- process select
        if (rst_n = '0') then
                                    counter_bus_i <= '0';
        elsif rising_edge(clk) then
          counter_bus_i <= counter_bus;
```

```
        end if;
    end process incr_proc;

  -- constant
    CNS <= '0' when clr_delay = 0 else '1';

  -- clear integrators signal
    clr_int <= '1' when counter_bus_i = CNS else '0';

end generate;

end architecture rtl;
```

Ακολουθεί ένα μέρος από το testbench του ασαφούς ελεγκτή σε γλώσσα TCL που χρησιμοποιήθηκε για RTL και Timing simulation του πυρήνα. Το testbench χρησιμοποιεί διανύσματα ελέγχου που έχουν παραχθεί από το Simulink μοντέλο του ελεγκτή.

```
##########################################################################
# Title        : Fuzzy Logic Controller-2 inputs normal model            #
# Project  : Fuzzy Controller (ODD-EVEN)                              #
# File           : FC_ROM.do                                                 #
# Author   : Kyriakos Deliparaschos                                   #
# Description  : RTL & back-anotation Testbench                        #
#                Depends on test vectors generated by Simulink model    #
# Date         : 31/03/2005                                            #
##########################################################################

############################
#  SETUP PARAMETERS
############################
set no_lines $1
set sim_type $2

############################
#  SIM TYPE ( rtl , target)
############################
if [string match $sim_type target] {set rtl false; set target true} else {set rtl true; set target false}

############################
#  WORKING DIRECTORY
############################
set wddir ..

############################
#  INPUTS DIRECTORY
############################
set bindir /tmp

############################
#  WORK library
############################
if [ string match $sim_type rtl ] {
  set work_dir                work
  set reso ps
} else {
  set work_dir                target
  set reso ps
  set sdf_file "$wddir/par/FC_ROM_s/FC_ROM_s_timesim.sdf"
}

############################
#  VSIM INVOCATION
############################
if [ string match $sim_type  rtl ]      {
 vsim  -t $reso -lib $work_dir FPGA_tb -wlf /wlf/fuzzycontroller.wlf
} else {
 vsim  -t $reso -lib $work_dir FPGA_tb -wlf /wlf/fuzzycontroller.wlf
}

############################
#  SET FILE PARAMETERS
############################
set x0_path $bindir/ip0
set x1_path $bindir/ip1
set ars_path $bindir/ars_bus
set reg_path $bindir/reg_bus
set ars_gen_path $bindir/ars_gen_bus
set reg_gen_path $bindir/reg_gen_bus
set start_path $bindir/start_bus
set slpcode_path $bindir/slpcode_bus
set sromip_odd_path $bindir/sromip_odd
set sromip_even_path $bindir/sromip_even
set cns_odd_path $bindir/cns_odd
set cns_even_path $bindir/cns_even
set distance_path $bindir/distance
set alpha_mul_path $bindir/alpha_mul
set alpha_bus_path $bindir/alpha_bus
set active_bus_odd_path $bindir/active_bus_odd
set active_bus_even_path $bindir/active_bus_even
set theta_active_odd_path $bindir/theta_active_odd
set theta_active_even_path $bindir/theta_active_even
set theta_active_path $bindir/theta_active
set impl_path $bindir/impl
set divs_path $bindir/divs
```

```
set divd_path $bindir/divd
set op_path $bindir/op

#### INPUTS ####
set x0_f [open $x0_path]
set x1_f [open $x1_path]
set ars_f [open $ars_path]
set reg_f [open $reg_path]
set ars_gen_f [open $ars_gen_path]
set reg_gen_f [open $reg_gen_path]
set start_f [open $start_path]
set slpcode_f [open $slpcode_path]
set sromip_odd_f [open $sromip_odd_path]
set sromip_even_f [open $sromip_even_path]
set cns_odd_f [open $cns_odd_path]
set cns_even_f [open $cns_even_path]
set distance_f [open $distance_path]
set alpha_mul_f [open $alpha_mul_path]
set alpha_bus_f [open $alpha_bus_path]
set active_bus_odd_f [open $active_bus_odd_path]
set active_bus_even_f [open $active_bus_even_path]
set theta_active_odd_f [open $theta_active_odd_path]
set theta_active_even_f [open $theta_active_even_path]
set theta_active_f [open $theta_active_path]
set impl_f [open $impl_path]
set divs_f [open $divs_path]
set divd_f [open $divd_path]
set op_f [open $op_path]

##############################
#  LOAD WAVE FILE
##############################
add log -r /*

if [string match $sim_type rtl] {
  do ../sim/FC_ROM_s_rtl_wave.do
} else {
  do ../sim/FC_ROM_s_target_wave.do
}

##############################
#  SET INPUT PRECISIONS
##############################
set x0_prec 8
set x1_prec 8

##############################
#  SET CLOCK PERIOD
##############################
set clk_period 10000
set clkx4_period 5000
set delay 400

##############################
#  SET RESET
##############################
set rst_time 40000
force rst_n 1 0 $reso
force rst_n 0 $rst_time $reso

##############################
#  ECHO TB PARAMETERS
##############################
echo
echo ******** TB parameters ********
echo "Simulation Type       : $sim_type"
echo "Number of Lines       : $no_lines"
echo "clk_period            : $clk_period $reso"
echo "clkx4_period          : $clkx4_period $reso"
echo "Simulation Time       : [expr $no_lines*$clk_period] $reso"
echo "Simulation Resolution : $reso"
echo "Working Directory     : $wddir"
echo "Inputs Directory      : $bindir"
echo "Working Library       : $work_dir"
if [string match $sim_type rtl] {

} else {
echo "SDF file              : $sdf_file"
}
echo "Input_x0              : $x0_path"
echo "Input_x1              : $x1_path"
echo "x0 input prec         : $x0_prec bit"
echo "x0 input prec         : $x1_prec bit"
echo ********************************************************************************
echo

##############################
#  READ INPUTS
##############################

for {set i 0} {$i< [expr $no_lines]} {incr i} {
          scan "[gets $x0_f]" %f x0_val_rl($i)
          set x0_val($i) [expr int($x0_val_rl($i))]
  scan "[gets $x1_f]" "%f" x1_val_rl($i)
          set x1_val($i) [expr int($x1_val_rl($i))]
}

close $x0_f
close $x1_f
```

```
############################
#  RUN SCRIPT
############################
force clk 1 0 -repeat $clk_period $reso
force clk 0 [expr $clk_period/2] $reso -repeat $clk_period $reso

for {set i 0} {$i < [expr $no_lines]} {incr i} {
  if {$x0_val($i) < 0} {
                force ip0 10#[expr [expr int([expr pow(2,$x0_prec)])] + $x0_val($i)] [expr $i*$clk_period] $reso
          }
          if {$x0_val($i) >= 0} {
                force ip0 10#$x0_val($i) [expr $i*$clk_period] $reso
          }

          if {$x1_val($i) < 0} {
                force ip1 10#[expr [expr int([expr pow(2,$x1_prec)])] + $x1_val($i)] [expr $i*$clk_period] $reso
          }
          if {$x1_val($i) >= 0} {
                force ip1 10#$x1_val($i) [expr $i*$clk_period] $reso
          }
}

run [expr ($no_lines-1)*$clk_period] $reso

############################
#  EXAMINE OUTPUT
############################
echo ******** ars_bus (rtl) ********
for {set i 1} {$i < [expr $no_lines]} {incr i} {
          scan "[gets $ars_f]" "%f" tv_val
          if [ string match $sim_type  rtl ] {
            set sim_ars  [exa -time [expr $i*$clkx4_period] $reso -unsigned /fpga_tb/u_fpga/u_fc/u_fuz_aggr/ars_bus_p]
    } else {
#   set sim_val [exa -time [expr $i*$clkx4_period+$delay] $reso -unsigned /fpga_tb/Ieq]
          }

          if [ string match $sim_type  rtl ] {
          if {$tv_val != $sim_ars} {
          if {$sim_ars != "X"} {
                                echo "Test vector value = $tv_val WHILE simulation value = $sim_ars AT $i line, diff = [expr
$tv_val - $sim_ars]"
                    }
          }
  }
}
echo ********************************************************************************
close $ars_f

echo ******** reg_bus (rtl) ********
for {set i 1} {$i < [expr $no_lines]} {incr i} {
          scan "[gets $reg_f]" "%f" tv_val
          if [ string match $sim_type  rtl ] {
            set sim_reg  [exa -time [expr $i*$clkx4_period] $reso -unsigned /fpga_tb/u_fpga/u_fc/u_fuz_aggr/reg_bus_p]
    } else {
#           set      sim_reg            [exa      -time      [expr      $i*$clkx4_period+$delay]      $reso      -unsigned
/fpga_tb/u_fpga/u_fc_u_fuz_aggr_reg_bus_p]
          }

  if [ string match $sim_type  rtl ] {
            if {$tv_val != $sim_reg} {
          if {$sim_reg != "X"} {
                                echo "Test vector value = $tv_val WHILE simulation value = $sim_reg AT $i line, diff = [expr
$tv_val - $sim_reg]"
                    }
            }
          }
}
echo ********************************************************************************
close $reg_f

echo ******** ars_gen_bus (rtl) ********
for {set i 1} {$i < [expr $no_lines]} {incr i} {
          scan "[gets $ars_gen_f]" "%f" tv_val
          if [ string match $sim_type  rtl ] {
            set      sim_ars_gen            [exa      -time      [expr      $i*$clkx4_period]      $reso      -unsigned
/fpga_tb/u_fpga/u_fc/u_fuz_aggr/ars_gen_bus_p]
    } else {
#   set sim_val [exa -time [expr $i*$clkx4_period+$delay] $reso -unsigned /CMAMMA_EQ/Ieq]
          }

          if [ string match $sim_type  rtl ] {
          if {$tv_val != $sim_ars_gen} {
          if {$sim_ars_gen != "X"} {
                                echo "Test vector value = $tv_val WHILE simulation value = $sim_ars_gen AT $i line, diff =
[expr $tv_val - $sim_ars_gen]"
                    }
          }
  }
}
echo ********************************************************************************
close $ars_gen_f

echo ******** theta_active (rtl,target) ********
for {set i 1} {$i < [expr $no_lines]} {incr i} {
          scan "[gets $theta_active_f]" "%f" tv_val
          if [ string match $sim_type  rtl ] {
            set      sim_theta_active           [exa      -time      [expr      $i*$clkx4_period]      $reso      -unsigned
/fpga_tb/u_fpga/u_fc/u_inf_defuz/theta_active_p]
    } else {
```

```
            set     sim_theta_active          [exa     -time     [expr     $i*$clkx4_period+$delay]     $reso     -unsigned
/fpga_tb/u_fpga/u_fc_u_inf_defuz_theta_active_p]
          }

          if {$tv_val != $sim_theta_active} {
          if {$sim_theta_active != "X"} {
                              echo "Test vector value = $tv_val WHILE simulation value = $sim_theta_active AT $i line,
diff = [expr $tv_val - $sim_theta_active]"
                    }
          }
}
echo ********************************************************************************
close $theta_active_f

echo ******** impl (rtl,target) ********
for {set i 1} {$i < [expr $no_lines]} {incr i} {
          scan "[gets $impl_f]" "%f" tv_val
          if [ string match $sim_type  rtl ] {
            set sim_impl  [exa -time [expr $i*$clkx4_period] $reso -dec /fpga_tb/u_fpga/u_fc/u_inf_defuz/impl_p]
    } else {
            set sim_impl  [exa -time [expr $i*$clkx4_period+$delay] $reso -dec /fpga_tb/u_fpga/u_fc_u_inf_defuz_impl_p]
          }

          if {$tv_val != $sim_impl} {
          if {$sim_impl != "X"} {
                              echo "Test vector value = $tv_val WHILE simulation value = $sim_impl AT $i line, diff =
[expr $tv_val - $sim_impl]"
                    }
          }
}
echo ********************************************************************************
close $impl_f

echo ******** op (rtl,target) ********
for {set i 1} {$i < [expr $no_lines]} {incr i} {
          scan "[gets $op_f]" "%f" tv_val
          if [ string match $sim_type  rtl ] {
            set sim_op  [exa -time [expr $i*$clkx4_period] $reso -dec /fpga_tb/op]
    } else {
            set sim_op  [exa -time [expr $i*$clkx4_period+$delay] $reso -dec /fpga_tb/op]
          }

          if {$tv_val != $sim_op} {
          if {$sim_op != "X"} {
                              echo "Test vector value = $tv_val WHILE simulation value = $sim_op AT $i line, diff = [expr
$tv_val - $sim_op]"
                    }
          }

}
echo ********************************************************************************
close $op_f
```

A.3 SoC

Παράδειγμα σύνδεσης του πυρήνα Microblaze με τον πυρήνα του ασαφούς ελεγκτή για την ολοκλήρωση του SoC.

```
 microblaze_0 : microblaze_0_wrapper
    port map (
      CLK => sys_clk_s,
      RESET => mb_opb_OPB_Rst,
      INTERRUPT => Interrupt,
      DEBUG_RST => Debug_Rst,
      EXT_BRK => Ext_BRK,
      EXT_NM_BRK => Ext_NM_BRK,
      DBG_STOP => net_gnd0,
      INSTR => dlmb_LMB_ReadDBus,
      I_ADDRTAG => open,
      IREADY => dlmb_LMB_Ready,
      IWAIT => net_gnd0,
      INSTR_ADDR => dlmb_M_ABus,
      IFETCH => dlmb_M_ReadStrobe,
      I_AS => dlmb_M_AddrStrobe,
      DATA_READ => ilmb_LMB_ReadDBus,
      DREADY => ilmb_LMB_Ready,
      DWAIT => net_gnd0,
      DATA_WRITE => ilmb_M_DBus,
      DATA_ADDR => ilmb_M_ABus,
      D_ADDRTAG => open,
      D_AS => ilmb_M_AddrStrobe,
      READ_STROBE => ilmb_M_ReadStrobe,
      WRITE_STROBE => ilmb_M_WriteStrobe,
      BYTE_ENABLE => ilmb_M_BE,
      DM_ABUS => mb_opb_M_ABus(0 to 31),
      DM_BE => mb_opb_M_BE(0 to 3),
      DM_BUSLOCK => mb_opb_M_busLock(0),
      DM_DBUS => mb_opb_M_DBus(0 to 31),
      DM_REQUEST => mb_opb_M_request(0),
      DM_RNW => mb_opb_M_RNW(0),
      DM_SELECT => mb_opb_M_select(0),
      DM_SEQADDR => mb_opb_M_seqAddr(0),
```

```
DOPB_DBUS => mb_opb_OPB_DBus,
DOPB_ERRACK => mb_opb_OPB_errAck,
DOPB_MGRANT => mb_opb_OPB_MGrant(0),
DOPB_RETRY => mb_opb_OPB_retry,
DOPB_TIMEOUT => mb_opb_OPB_timeout,
DOPB_XFERACK => mb_opb_OPB_xferAck,
IM_ABUS => mb_opb_M_ABus(32 to 63),
IM_BE => mb_opb_M_BE(4 to 7),
IM_BUSLOCK => mb_opb_M_busLock(1),
IM_DBUS => mb_opb_M_DBus(32 to 63),
IM_REQUEST => mb_opb_M_request(1),
IM_RNW => mb_opb_M_RNW(1),
IM_SELECT => mb_opb_M_select(1),
IM_SEQADDR => mb_opb_M_seqAddr(1),
IOPB_DBUS => mb_opb_OPB_DBus,
IOPB_ERRACK => mb_opb_OPB_errAck,
IOPB_MGRANT => mb_opb_OPB_MGrant(1),
IOPB_RETRY => mb_opb_OPB_retry,
IOPB_TIMEOUT => mb_opb_OPB_timeout,
IOPB_XFERACK => mb_opb_OPB_xferAck,
DBG_CLK => DBG_CLK_s,
DBG_TDI => DBG_TDI_s,
DBG_TDO => DBG_TDO_s,
DBG_REG_EN => DBG_REG_EN_s,
DBG_CAPTURE => DBG_CAPTURE_s,
DBG_UPDATE => DBG_UPDATE_s,
VALID_INSTR => open,
PC_EX => open,
REG_WRITE => open,
REG_ADDR => open,
MSR_REG => open,
NEW_REG_VALUE => open,
PIPE_RUNNING => open,
INTERRUPT_TAKEN => open,
JUMP_TAKEN => open,
PREFETCH_ADDR => open,
MB_Halted => open,
Trace_Branch_Instr => open,
Trace_Delay_Slot => open,
Trace_Data_Address => open,
Trace_AS => open,
Trace_Data_Read => open,
Trace_Data_Write => open,
Trace_DCache_Req => open,
Trace_DCache_Hit => open,
Trace_ICache_Req => open,
Trace_ICache_Hit => open,
Trace_Instr_EX => open,
FSL0_S_CLK => open,
FSL0_S_READ => i_fsl_v20_0_FSL_S_Read,
FSL0_S_DATA => i_fsl_v20_0_FSL_S_Data,
FSL0_S_CONTROL => i_fsl_v20_0_FSL_S_Control,
FSL0_S_EXISTS => i_fsl_v20_0_FSL_S_Exists,
FSL0_M_CLK => open,
FSL0_M_WRITE => i_fsl_v20_1_FSL_M_Write,
FSL0_M_DATA => i_fsl_v20_1_FSL_M_Data,
FSL0_M_CONTROL => i_fsl_v20_1_FSL_M_Control,
FSL0_M_FULL => i_fsl_v20_1_FSL_M_Full,
FSL1_S_CLK => open,
FSL1_S_READ => open,
FSL1_S_DATA => net_gnd32,
FSL1_S_CONTROL => net_gnd0,
FSL1_S_EXISTS => net_gnd0,
FSL1_M_CLK => open,
FSL1_M_WRITE => open,
FSL1_M_DATA => open,
FSL1_M_CONTROL => open,
FSL1_M_FULL => net_gnd0,
FSL2_S_CLK => open,
FSL2_S_READ => open,
FSL2_S_DATA => net_gnd32,
FSL2_S_CONTROL => net_gnd0,
FSL2_S_EXISTS => net_gnd0,
FSL2_M_CLK => open,
FSL2_M_WRITE => open,
FSL2_M_DATA => open,
FSL2_M_CONTROL => open,
FSL2_M_FULL => net_gnd0,
FSL3_S_CLK => open,
FSL3_S_READ => open,
FSL3_S_DATA => net_gnd32,
FSL3_S_CONTROL => net_gnd0,
FSL3_S_EXISTS => net_gnd0,
FSL3_M_CLK => open,
FSL3_M_WRITE => open,
FSL3_M_DATA => open,
FSL3_M_CONTROL => open,
FSL3_M_FULL => net_gnd0,
FSL4_S_CLK => open,
FSL4_S_READ => open,
FSL4_S_DATA => net_gnd32,
FSL4_S_CONTROL => net_gnd0,
FSL4_S_EXISTS => net_gnd0,
FSL4_M_CLK => open,
FSL4_M_WRITE => open,
FSL4_M_DATA => open,
FSL4_M_CONTROL => open,
FSL4_M_FULL => net_gnd0,
FSL5_S_CLK => open,
FSL5_S_READ => open,
```

```
      FSL5_S_DATA => net_gnd32,
      FSL5_S_CONTROL => net_gnd0,
      FSL5_S_EXISTS => net_gnd0,
      FSL5_M_CLK => open,
      FSL5_M_WRITE => open,
      FSL5_M_DATA => open,
      FSL5_M_CONTROL => open,
      FSL5_M_FULL => net_gnd0,
      FSL6_S_CLK => open,
      FSL6_S_READ => open,
      FSL6_S_DATA => net_gnd32,
      FSL6_S_CONTROL => net_gnd0,
      FSL6_S_EXISTS => net_gnd0,
      FSL6_M_CLK => open,
      FSL6_M_WRITE => open,
      FSL6_M_DATA => open,
      FSL6_M_CONTROL => open,
      FSL6_M_FULL => net_gnd0,
      FSL7_S_CLK => open,
      FSL7_S_READ => open,
      FSL7_S_DATA => net_gnd32,
      FSL7_S_CONTROL => net_gnd0,
      FSL7_S_EXISTS => net_gnd0,
      FSL7_M_CLK => open,
      FSL7_M_WRITE => open,
      FSL7_M_DATA => open,
      FSL7_M_CONTROL => open,
      FSL7_M_FULL => net_gnd0,
      ICACHE_FSL_IN_CLK => open,
      ICACHE_FSL_IN_READ => open,
      ICACHE_FSL_IN_DATA => net_gnd32,
      ICACHE_FSL_IN_CONTROL => net_gnd0,
      ICACHE_FSL_IN_EXISTS => net_gnd0,
      ICACHE_FSL_OUT_CLK => open,
      ICACHE_FSL_OUT_WRITE => open,
      ICACHE_FSL_OUT_DATA => open,
      ICACHE_FSL_OUT_CONTROL => open,
      ICACHE_FSL_OUT_FULL => net_gnd0,
      DCACHE_FSL_IN_CLK => open,
      DCACHE_FSL_IN_READ => open,
      DCACHE_FSL_IN_DATA => net_gnd32,
      DCACHE_FSL_IN_CONTROL => net_gnd0,
      DCACHE_FSL_IN_EXISTS => net_gnd0,
      DCACHE_FSL_OUT_CLK => open,
      DCACHE_FSL_OUT_WRITE => open,
      DCACHE_FSL_OUT_DATA => open,
      DCACHE_FSL_OUT_CONTROL => open,
      DCACHE_FSL_OUT_FULL => net_gnd0
    );

  flc_ip_0 : flc_ip_0_wrapper
    port map (
      FSL_Clk => sys_clk_s,
      FSL_Rst => i_fsl_v20_0_OPB_Rst,
      FSL_S_Clk => open,
      FSL_S_Read => i_fsl_v20_1_FSL_S_Read,
      FSL_S_Data => i_fsl_v20_1_FSL_S_Data,
      FSL_S_Control => i_fsl_v20_1_FSL_S_Control,
      FSL_S_Exists => i_fsl_v20_1_FSL_S_Exists,
      FSL_M_Clk => open,
      FSL_M_Write => i_fsl_v20_0_FSL_M_Write,
      FSL_M_Data => i_fsl_v20_0_FSL_M_Data,
      FSL_M_Control => i_fsl_v20_0_FSL_M_Control,
      FSL_M_Full => i_fsl_v20_0_FSL_M_Full
    );
```

Συναρτήσεις ελέγχου (C κώδικας) που εκτελούνται στον Microblaze για την επικοινωνία του SoC με το Pioneer P3 ρομπότ και τον υπολογισμό παραμέτρων για το πρόβλημα της παρακολούθησης δρόμου.

```
#include "xuartlite_l.h"
#include "xintc_l.h"
#include "xintc.h"
#include "xtmrctr_l.h"
#include "xtmrctr.h"
#include "xgpio_l.h"
#include "xparameters.h"
#include "xbasic_types.h"
#include "xstatus.h"
#include "flc_ip.h"
#define PI 3.14159265
#define PIBY2 1.5707963

#define LED7SEG1_BASEADDR XPAR_LED_7SEGMENT1_BASEADDR
#define LED7SEG2_BASEADDR XPAR_LED_7SEGMENT2_BASEADDR
#define RS232_BASEADDR XPAR_RS232_BASEADDR
#define USB_BASEADDR XPAR_USB_UART_BASEADDR
#define FLC_IP_0_INPUT_SLOT_ID  XPAR_FSL_FLC_IP_0_INPUT_SLOT_ID
#define FLC_IP_0_OUTPUT_SLOT_ID  XPAR_FSL_FLC_IP_0_OUTPUT_SLOT_ID

#define true 1
```

```
//*********************************************************************
// Send pulse
//*********************************************************************
void send_pulse(){
          unsigned int pulse_packet[]={0xFA,0xFB,0x03,0x00,0x00,0x00};
          int i=0;
          unsigned int *pulse_ptr;
          pulse_ptr=&pulse_packet[0];
          for (i=0; i<6; i++) {
                    XUartLite_SendByte(RS232_BASEADDR, *pulse_ptr++);
          }
}

//*********************************************************************
//        The following function generates a variable delay
//*********************************************************************
void _wait(int loop_count){
int       sum;
volatile int data;
sum       = 0;
for (data = 0; data < loop_count; data++) {
          sum = (data << 8);}
  return;
}

//*********************************************************************
// Echo test for sync packets
//*********************************************************************
unsigned int sync_echo_test (unsigned int sync[],unsigned int byteCount) {
          unsigned int echo_sync[byteCount];
          unsigned int while_flag=1;
          unsigned int chkCounter=0;
          int i; // declare index
          while(while_flag){
                    chkCounter=0;
                    for (i=0; i<byteCount; i++) {
                              echo_sync[i] = XUartLite_RecvByte(RS232_BASEADDR);
                              if(echo_sync[i]==sync[i]){
                                        chkCounter++;
                              }
                    }
                    if(chkCounter==byteCount){while_flag=0;}
          }
          return(1);
}


//*********************************************************************
// Open Server
//*********************************************************************
void send_open(){
          unsigned int open_packet[]={0xFA,0xFB,0x03,0x01,0x00,0x01};
          int i=0;
          unsigned int *open_ptr;
          open_ptr=&open_packet[0];
          for (i=0; i<6; i++) {
                    XUartLite_SendByte(RS232_BASEADDR, *open_ptr++);
          }
}

//*********************************************************************
// Checksum Calculation
//*********************************************************************
int calc_chksum(unsigned int *ptr){
          int n;
          int c=0;

          n=*(ptr++);
          n-=2;
          while(n>1){
                    c+=(*(ptr)<<8) | *(ptr+1);
                    c=c & 0xffff;
                    n-=2;
                    ptr+=2;
          }
          if(n>0)
                    c=c^(int)*(ptr++);
          return(c);
}

unsigned short int calc_Path_chksum(unsigned int *ptr){
          short int length;
          unsigned short int c=0;
          int i=1;
          length=(*ptr)*2;

          for(i=1;i<=length;i++){
                    c+=*(ptr+i);
          }
          return(c);
}

//*********************************************************************
// Synchronization sequence
//*********************************************************************
int do_sync(){
          unsigned int sync0[]={0xFA,0xFB,0x03,0x00,0x00,0x00}; // sync0 array variable
          unsigned int sync1[]={0xFA,0xFB,0x03,0x01,0x00,0x01}; // sync1 array variable
          unsigned int sync2[]={0xFA,0xFB,0x03,0x02,0x00,0x02}; // sync2 array variable
          unsigned int autoConfigSync[]={0xFA,0xFB,0x1C,0x02,0x41,0x74,0x68,0x65,
```

```
          0x6E,0x73,0x5F,0x32,0x30,0x35,0x38,0x00,0x50,0x69,0x6F,0x6E,0x65,
          0x65,0x72,0x00,0x50,0x33,0x44,0x58,0x00,0x80,0x08};
          unsigned int *sync0_ptr; // pointer to sync0 array
          unsigned int *sync1_ptr;
          unsigned int *sync2_ptr;
          int i; // declare index
          int status=0;

          sync0_ptr = &sync0[0]; // point to the 1st element of sync0
          sync1_ptr = &sync1[0];
          sync2_ptr = &sync2[0];

          unsigned int firstTst=0;
          unsigned int secTst=0;
          unsigned int thirdTst=0;


          for (i=0; i<6; i++) {
                    XUartLite_SendByte(RS232_BASEADDR, *sync0_ptr++);
          }
          firstTst=sync_echo_test(sync0,6);

          if(firstTst==1) {
                    status++;
                    _wait(1000000);
                    for (i=0; i<6; i++) {
                              XUartLite_SendByte(RS232_BASEADDR, *sync1_ptr++);
                    }
          }
          secTst=sync_echo_test(sync1,6);

          if(secTst==1) {
                    status++;
                    _wait(1000000);
                    for (i=0; i<6; i++) {
                              XUartLite_SendByte(RS232_BASEADDR, *sync2_ptr++);
                    }
          }

          thirdTst=sync_echo_test(autoConfigSync,31);
          if(thirdTst==1) {
                    status++;
                    _wait(1000000);
                    send_open();
          }
          return(status);
}

//*************************************************************************
// Send command
//*************************************************************************
void send_command(unsigned int commNum, unsigned int hasArg,int arg){
          int i=0;
          unsigned int byteCount=3;
          int chksum;
          int *ip;

          if(hasArg!=0){    //argType=0 => No argument
                    byteCount+=3;
          }
          ip=malloc((byteCount+3) * sizeof(int));
          *ip=0xFA;
          *(ip+1)=0xFB;
          *(ip+2)=byteCount;
          *(ip+3)=commNum;

          if(hasArg!=0){
                    if(arg>0){
                              *(ip+4)=0x3B; }
                    else {
                              *(ip+4)=0x1B;
                              arg=-arg;
                               }
                    *(ip+5)=arg;
                    *(ip+6)=arg>>8;
          }
          chksum=calc_chksum(ip+2);
          *(ip+byteCount+2)=chksum;
          *(ip+byteCount+1)=chksum>>8;
          for (i=0; i<(byteCount+3); i++) {
                              XUartLite_SendByte(RS232_BASEADDR, *ip++);
                    }
          free(ip);
}

double fast_atan2f( double y, double x )
{
          if ( x == 0.0 )
          {
                    if ( y > 0.0 ) return PIBY2;
                    if ( y == 0.0 ) return 0.0;
                    return -PIBY2;
          }
          double atan;
          double z = y/x;
          if ( fabsf( z ) < 1.0 )
          {
                    atan = z/(1.0 + 0.28*z*z);
                    if ( x < 0.0 )
                    {
```

```
                        if ( y < 0.0 ) return atan - PI;
                        return atan + PI;
                }
        }
        else
        {
                atan = PIBY2 - z/(z*z + 0.28);
                if ( y < 0.0 ) return atan - PI;
        }
        return atan;
}

//**********************************************************************
// Extract input
//**********************************************************************
void extract_input(double xx,double yy,double th, int *path, int pnum,double *out){
        int minInd=0;
        int windowOrder=3;  //minimum 1
        int windowOffset=5;
        int windowStep=5;  //minimum 0
        int i=0;
        long long int cur_dist=0;
        long long int prev_dist=0;
        long long int x=(long long int)xx;
        long long int y=(long long int)yy;
        double theta1=0;
        double theta2=0;
        path++;

        prev_dist=((long long int)*path-x)*((long long int)*path-x)+((long long int)*(path+pnum)-y)*((long long
int)*(path+pnum)-y);
        for(i=1;i<pnum;i++){
                cur_dist=((long long int)*(path+i)-x)*((long long int)*(path+i)-x)+((long long int)*(path+i+pnum)-
y)*((long long int)*(path+i+pnum)-y);
                if(cur_dist<prev_dist){
                        minInd=i;
                        prev_dist=cur_dist;
                }
        }

        minInd+=windowOffset;  // shift window by windowOffset
        if((minInd+windowOrder*windowStep)>=pnum-1){
                minInd=pnum-3;
                windowOrder=1;
                windowStep=0;
                *(out+3)=0;   //stop (velocity=0)
        }
        *out=0;
        *(out+1)=0;
        for(i=0;i<windowOrder;i++){
                minInd+=windowStep;
                *out=fast_atan2f((double)*(path+minInd+pnum)-yy,(double)*(path+minInd)-xx)-th;
                *(out+1)=fast_atan2f((double)(*(path+minInd+pnum+1)-*(path+minInd+pnum)),(double)(*(path+minInd+1)-
*(path+minInd)))-th;

                double tmpOut1=*out-((int)(*out/(2*PI)))*2*PI;
                double tmpOut2=*(out+1)-((int)(*(out+1)/(2*PI)))*2*PI;

                if(tmpOut1<-PI){
                        *out=tmpOut1+2*PI;}
                else if(tmpOut1>PI){
                        *out=tmpOut1-2*PI;}
                else{*out=tmpOut1;}
                if(tmpOut2<-PI){
                        *(out+1)=tmpOut2+2*PI;}
                else if(tmpOut2>PI){
                        *(out+1)=tmpOut2-2*PI;}
                else{*(out+1)=tmpOut2;}
                theta1+=(*out);
                theta2+=(*(out+1));
        }
        *out=theta1/windowOrder;
        *(out+1)=theta2/windowOrder;
        *(out+2)=(double)minInd;
}

//**********************************************************************
// Receive Path
//**********************************************************************
int *receive_Path(){
        unsigned int headerByte1=0x00;
        unsigned int headerByte2=0x00;
        unsigned int timerCycles=0;
        int i=0;
        int byteCount=0;
        int byteCountMSB=0;
        int byteCountLSB=0;
        int dataMSB=0;
        int dataLSB1=0;
        int dataLSB2=0;
        int dataLSB3=0;
        int foundHeader=0;
        int *ip;
    unsigned short int chk;

        while(1){

                headerByte1=XUartLite_RecvByte(USB_BASEADDR);
                if(headerByte1==0xFA){  //Check first header byte
                        headerByte2=XUartLite_RecvByte(USB_BASEADDR);
```

```
                        if(headerByte2==0xFB){ //Check second header byte
                                foundHeader=1;
                                break;
                        }
                }
        }//end while

        if(foundHeader){
                byteCountLSB=XUartLite_RecvByte(USB_BASEADDR);
                byteCountMSB=XUartLite_RecvByte(USB_BASEADDR);
                byteCount=(byteCountMSB<<8) | byteCountLSB;
                ip=malloc((1+(byteCount-2)/4) * sizeof(int)); //
                *ip=(byteCount-2)/8;  //number of points
                for(i=1;i<=(int)((byteCount-2)/4);i++){
                        dataLSB1=XUartLite_RecvByte(USB_BASEADDR);
                        dataLSB2=XUartLite_RecvByte(USB_BASEADDR);
                        dataLSB3=XUartLite_RecvByte(USB_BASEADDR);
                        dataMSB=XUartLite_RecvByte(USB_BASEADDR);
                        *(ip+i)=(((dataMSB<<24) | (dataLSB3<<16)) | (dataLSB2<<8)) | dataLSB1;
                }
                dataLSB1=XUartLite_RecvByte(USB_BASEADDR);
                dataLSB2=XUartLite_RecvByte(USB_BASEADDR);
                *(ip+1+(byteCount-2)/4)= (dataLSB2<<8) | dataLSB1;
                chk=calc_Path_chksum(ip);
                if(chk==(unsigned short int)*(ip+1+(byteCount-2)/4)){
                        XUartLite_SendByte(USB_BASEADDR,248);
                        XUartLite_SendByte(USB_BASEADDR,249);
                        XUartLite_SendByte(USB_BASEADDR,250);
                        XUartLite_SendByte(USB_BASEADDR,251);
                        XGpio_mSetDataReg(LED7SEG2_BASEADDR, 2, 0x08);
                }else {
                        free(ip);
                        XUartLite_SendByte(USB_BASEADDR,240);
                        XUartLite_SendByte(USB_BASEADDR,241);
                        XUartLite_SendByte(USB_BASEADDR,242);
                        XUartLite_SendByte(USB_BASEADDR,243);
                        XGpio_mSetDataReg(LED7SEG2_BASEADDR, 2, 0x30);
                }
        return(ip);
        }
}
//*************************************************************************
// Receive SIP
//*************************************************************************
int *receive_SIP(){
        unsigned int headerByte1=0x00;
        unsigned int headerByte2=0x00;
        unsigned int timerCycles=0;
        int i=0;
        int byteCount=0;
        int foundHeader=0;
        static short int firstCall=1;
        static int *ip;

        while(1){
                headerByte1=XUartLite_RecvByte(RS232_BASEADDR);
                if(headerByte1==0xFA){  //Check first header byte
                        headerByte2=XUartLite_RecvByte(RS232_BASEADDR);
                        if(headerByte2==0xFB){ //Check second header byte
                                foundHeader=1;
                                break;
                        }
                }
        }//end while

        if(foundHeader){
                byteCount=XUartLite_RecvByte(RS232_BASEADDR);
                if(firstCall){
                        //ip=malloc((byteCount+3) * sizeof(int));
                        firstCall=0;
                        ip=malloc(200 * sizeof(int));  //sip packet =>200 int's
                        *ip=0xFA;
                        *(ip+1)=0xFB;
                }

                *(ip+2)=byteCount;
                XUartLite_SendByte(USB_BASEADDR,*ip);
                XUartLite_SendByte(USB_BASEADDR,*(ip+1));
                XUartLite_SendByte(USB_BASEADDR,*(ip+2));
                for(i=1;i<=byteCount;i++){
                        *(ip+i+2)=XUartLite_RecvByte(RS232_BASEADDR);
                        XUartLite_SendByte(USB_BASEADDR,*(ip+i+2));
                }
        return(ip);
        }
}

//*************************************************************************
// FLC interface
//*************************************************************************
unsigned int FLC(double in1,double in2){
        short int inn1;
        short int inn2;
    unsigned short int innc1;
        unsigned short int innc2;

        if (in1 < 0) {
                inn1=(short int)(-1+in1*4096/360);
                innc1 = inn1 + 4096;}
        else{
```

```
                inn1=(short int)(in1*4096/360);
                innc1=inn1;
        }

        if (in2 < 0) {
                inn2=(short int)(-1+in2*4096/360);
                innc2 = inn2 + 4096;}
        else{
                inn2=(short int)(in2*4096/360);
                innc2=inn2;
        }
        unsigned int input_0=0;
        unsigned int output_0=0;
        short int output=0;

        input_0=(unsigned int)innc2<<16;
        input_0=input_0|(unsigned int)innc1;
        write_into_fsl(input_0,FLC_IP_0_INPUT_SLOT_ID);
        read_from_fsl(output_0,FLC_IP_0_OUTPUT_SLOT_ID);
        output=output_0;
        return(output);
}
```

Μέρος από το περιβάλλον (MATLAB κώδικας) της γραφικής εφαρμογής σχεδιασμού και παρακολούθησης της πορείας του ρομπότ.

```
% Main GUI code (depends on .fig which calls other functions as callbacks)
% Kyriakos Deliparaschos, 2007
clear all; clc

global robotBlob
global robotHeadingLine
global hli
global timestamp

%%%% Local Coordinates of Robot Pose %%%%
global xRobLocal
global yRobLocal
global thRobLocal

%%%% Global Coordinates of Robot Pose %%%%
global xRobGlobal
global yRobGlobal
global thRobGlobal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global xInterp
global yInterp

global Robottheta1
global Robottheta2
global Matlabtheta1
global Matlabtheta2

global MlabminPathInd
global RminPathInd
global hli

global y0
global x0
global th0

global newRun
newRun=0;

global axh

y0=1000;  %mm
x0=1000;  %mm
th0=0;  %degrees

%%%Pose initialization
xRobGlobal=x0;
yRobGlobal=y0;
thRobGlobal=th0*pi/180;

xRobLocal=0;
yRobLocal=0;
thRobLocal=0;
%%%%%%%%%%%%%%%%%%%%%%%%

delete(instrfind);

s = serial('COM9');
set(s,'tag','usbComObj');
s.BytesAvailableFcnCount = 1;
s.BytesAvailableFcnMode = 'byte';
s.BytesAvailableFcn = @rcvUsbByte;
s.OutputBufferSize=32000;
s.OutputEmptyFcn=@pathSent;
s.BaudRate=9600;

figh=openfig('mainGUI.fig');
```

```
axh=get(figh,'CurrentAxes');

robotBlob=rectangle('position',[x0-500 y0-500 1000 1000],'curv',[1 1],'parent',axh,'tag','robotBlob');
robotHeadingLine=line([x0,x0+700*cos(th0*pi/180)],[y0,y0+700*sin(th0*pi/180)]);  %theta=th0
hli=line([0,0],[0,0]);
set(hli,'marker','.','color',[0 1 0],'MarkerSize',10);
```

A.4 Γενετικός Αλγόριθμος

Στον VHDL κώδικα που ακολουθεί δηλώνονται τα επιμέρους υποσυστήματα που συνθέτουν τον πυρήνα του γενετικού αλγορίθμου καθώς και άλλες παράμετροι.

```
--------------------------------------------------------------------------------
-- Title      : Genetic Algorithm Package
-- Project    : Genetic Algorithm
--------------------------------------------------------------------------------
-- File       : ga_pkg.vhd
-- Author     : Kyriakos Deliparaschos (kdelip@mail.ntua.gr)
-- Company    : NTUA/IRAL
-- Created    : 23/03/06
-- Last update: 12/01/08
-- Platform   : Modelsim 6.1c, Synplicity 8.1, ISE 8.1
-- Standard   : VHDL'93
--------------------------------------------------------------------------------
-- Description: Package definition file
--------------------------------------------------------------------------------
-- Copyright (c) 2006 NTUA
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 23/03/06    1.1      kdelip  Created
-- 16/11/06    1.2      kdelip  Update delay_regs (width-1 -> width) and all
--                              necessary signals. Add rng_128 component
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
-- LIBRARIES
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library work;
use work.arith_pkg.all;

--------------------------------------------------------------------------------
-- PACKAGE DECLARATION
--------------------------------------------------------------------------------
package ga_pkg is

-- CONSTANTS DECLARATION
--------------------------------------------------------------------------------
  type int_array is array (integer range <>) of integer;

  constant pool : int_array(1 to 7) := (1, 2, 3, 4, 5, 6, 7);

-- COMPONENTS DECLARATION
--------------------------------------------------------------------------------

  -- Random Number Generator
  component rng is
    generic(
      n : positive);                    -- length of pseudo-random sequence
    port (
      clk          : in  std_logic;     -- clock
      rst_n        : in  std_logic;     -- reset (active low)
      load         : in  std_logic;     -- load (active high)
      seed         : in  std_logic_vector(n-1 downto 0);
      run          : in  std_logic;
      parallel_out : out std_logic_vector(n-1 downto 0));  -- parallel data out
  end component rng;

  component rng_128 is
    generic(
      n : positive);                    -- length of pseudo-random sequence
    port (
      clk          : in  std_logic;     -- clock
      rst_n        : in  std_logic;     -- reset (active low)
      load         : in  std_logic;     -- load (active high)
      seed         : in  std_logic_vector(n-1 downto 0);
      run          : in  std_logic;
      parallel_out : out std_logic_vector(n-1 downto 0));  -- parallel data out
  end component rng_128;

  component fitness_calc is
    generic(
      genom_lngt : positive;
      score_sz   : integer;
      pop_sz     : positive);
    port (
      clk        : in  std_logic;       -- clock
```

```
    rst_n      : in  std_logic;       -- reset (active low)
    decode     : in  std_logic;
    valid      : in  std_logic;
    in_genes   : in  std_logic_vector(2*genom_lngt-1 downto 0);
    gene_score : out std_logic_vector(genom_lngt+score_sz-1 downto 0);
    fit        : out std_logic_vector(score_sz-1 downto 0) := (others => '0');
    ready_out  : out std_logic);
end component fitness_calc;

component fix_elite is
  generic(
    genom_lngt : positive;
    score_sz   : integer;
    pop_sz     : positive;
    elite      : positive);
  port (
    clk           : in  std_logic;    -- clock
    rst_n         : in  std_logic;    -- reset (active low)
    decode        : in  std_logic;
    valid         : in  std_logic;
    elite_null    : in  std_logic;
    index         : in  integer;
    fit           : in  std_logic_vector(score_sz-1 downto 0);
    count_parents : in  integer;
    ready_in      : in  std_logic;
    elite_offs    : out int_array(0 to elite-1);
    fit_sum       : out std_logic_vector(score_sz+log2(pop_sz)-1 downto 0);
    max_fit       : out std_logic_vector(score_sz-1 downto 0);
    rd            : out std_logic);
end component fix_elite;

component fitness_eval is
  generic (
    genom_lngt : positive;            -- townres*(num_towns-1)
    score_sz   : integer;
    pop_sz     : integer;
    elite      : integer);
  port (
    clk           : in  std_logic;
    rst_n         : in  std_logic;
    decode        : in  std_logic;
    valid         : in  std_logic;
    in_genes      : in  std_logic_vector(2*genom_lngt-1 downto 0);
    elite_null    : in  std_logic;
    index         : in  integer;
    count_parents : in  integer;
    gene_score    : out std_logic_vector(genom_lngt+score_sz-1 downto 0);
    elite_offs    : out int_array(0 to elite-1);
    fit_sum       : out std_logic_vector(score_sz+log2(pop_sz)-1 downto 0);
    max_fit       : out std_logic_vector(score_sz-1 downto 0);
    rd            : out std_logic);
end component fitness_eval;

component selection is
  generic(
    genom_lngt         : positive;
    pop_sz             : positive;
    elite              : integer;
    score_sz           : integer;
    scaling_factor_res : integer);
  port (
    clk        : in  std_logic;       -- clock
    rst_n      : in  std_logic;       -- reset (active low)
    inGene     : in  std_logic_vector(genom_lngt+score_sz-1 downto 0);
    rng        : in  std_logic_vector(scaling_factor_res-1 downto 0);
    fitSum     : in  std_logic_vector(score_sz+log2(pop_sz)-1 downto 0);
    data_valid : in  std_logic;
    next_gene  : in  std_logic;
    selParent  : out std_logic_vector(genom_lngt-1 downto 0);
    rd         : out std_logic);
end component selection;

component mutation is
  generic(
    genom_lngt : positive;
    mr         : integer;
    mut_res    : integer);
  port (
    clk       : in  std_logic;        -- clock
    rst_n     : in  std_logic;        -- reset (active low)
    mutPoint  : in  std_logic_vector(log2(genom_lngt)-1 downto 0);
    mutMethod : in  std_logic_vector(1 downto 0);
    cont      : in  std_logic;
    flag      : in  std_logic;
    rng       : in  std_logic_vector(genom_lngt+mut_res-1 downto 0);
    inGene    : in  std_logic_vector(genom_lngt-1 downto 0);
    rd        : out std_logic;
    mutOffspr : out std_logic_vector(genom_lngt-1 downto 0));
end component mutation;

component crossover is
  generic(
    genom_lngt : positive);
  port (
    clk          : in  std_logic;     -- clock
    rst_n        : in  std_logic;     -- reset (active low)
    cont         : in  std_logic;
    crossPoints  : in  std_logic_vector(2*log2(genom_lngt)-1 downto 0);
    crossMethod  : in  std_logic_vector(1 downto 0);
    rng          : in  std_logic_vector(genom_lngt-1 downto 0);
```

```
      inGene1      : in  std_logic_vector(genom_lngt-1 downto 0);
      inGene2      : in  std_logic_vector(genom_lngt-1 downto 0);
      rd           : out std_logic;
      crossOffspr1 : out std_logic_vector(genom_lngt-1 downto 0));
  end component crossover;

  component spram is
    generic (
      add_width  : integer;
      data_width : integer);
    port (
      clk      : in  std_logic;         -- write clock
      rst_n    : in  std_logic;         -- system reset
      add      : in  std_logic_vector(add_width downto 0);
      data_in  : in  std_logic_vector(data_width -1 downto 0);
      data_out : out std_logic_vector(data_width -1 downto 0);
      wr       : in  std_logic);        -- read/write enable
  end component spram;

  component obs is
    generic(
      score_sz  : positive;
      fit_limit : positive);
    port (clk       : in  std_logic;    -- clock
          rst_n     : in  std_logic;    -- reset (active low)
          max_fit   : in  std_logic_vector(score_sz-1 downto 0);
          fitlim_rd : out std_logic;
          rd        : out std_logic);
  end component obs;

  component control is
    generic(
      genom_lngt : integer;
      max_gen    : positive;
      pop_sz     : integer;
      score_sz   : integer;
      elite      : integer);
    port (
      clk             : in  std_logic;
      rst_n           : in  std_logic;
      done            : in  std_logic;
      fit_eval_rd     : in  std_logic;
      sel_rd          : in  std_logic;
      cross_rd        : in  std_logic;
      mut_rd          : in  std_logic;
      term_rd         : in  std_logic;
      run_ga          : in  std_logic;
      elite_offs      : in  int_array(0 to elite-1);
      data_in_ram2    : in  std_logic_vector(genom_lngt-1 downto 0);
      mut_method      : in  std_logic_vector(1 downto 0);
      data_out_cross1 : out std_logic_vector(genom_lngt-1 downto 0);
      data_out_cross2 : out std_logic_vector(genom_lngt-1 downto 0);
      addr_1_c        : out integer;
      addr_2_c        : out integer;
      cnt_parents     : out integer;
      we1_c           : out std_logic;
      we2_c           : out std_logic;
      data_valid      : out std_logic;
      next_gene       : out std_logic;
      ga_fin          : out std_logic;
      cross_out       : out std_logic;
      valid           : out std_logic;
      elite_null      : out std_logic;
      index           : out integer range 0 to pop_sz+1;
      mut_out         : out std_logic;
      flag            : out std_logic;
      decode          : out std_logic;
      sel_out         : out std_logic;
      term_out        : out std_logic;
      run             : out std_logic;
      run1            : out std_logic;
      run2            : out std_logic;
      run3            : out std_logic;
      load            : out std_logic);
  end component control;

  component delay_regs is
    generic (
      width   : integer;               -- data width
      latency : integer);              -- number of delay elements
    port (
      clk      : in  std_logic;         -- clock
      rst_n    : in  std_logic;         -- reset (active low)
      in_data  : in  std_logic_vector(width downto 0);   -- input data
      out_data : out std_logic_vector(width downto 0));  -- ouput data
  end component delay_regs;

--------------------------------------------------------------------------------
-- Components for TSP problem evaluation
--------------------------------------------------------------------------------

  component crossover_TSP is
    generic(
      genom_lngt : positive;
      num_towns  : positive);
    port (
      clk          : in  std_logic;     -- clock
      rst_n        : in  std_logic;     -- reset (active low)
      cont         : in  std_logic;
      crossPoints  : in  std_logic_vector(2*log2(num_towns)-1 downto 0);
```

```
    inGene1      : in  std_logic_vector(genom_lngt-1 downto 0);
    inGene2      : in  std_logic_vector(genom_lngt-1 downto 0);
    rd           : out std_logic;
    crossOffspr1 : out std_logic_vector(genom_lngt-1 downto 0));
end component crossover_TSP;

component mutation_tsp is
  generic(
    genom_lngt : positive;
    mr         : integer;
    mut_res    : integer;
    num_towns  : integer);
  port (
    clk       : in  std_logic;        -- clock
    rst_n     : in  std_logic;        -- reset (active low)
    mutPoint  : in  std_logic_vector(2*log2(num_towns)-1 downto 0);
    cont      : in  std_logic;
    flag      : in  std_logic;
    rng       : in  std_logic_vector(mut_res-1 downto 0);
    inGene    : in  std_logic_vector(genom_lngt-1 downto 0);
    rd        : out std_logic;
    mutOffspr : out std_logic_vector(genom_lngt-1 downto 0));
end component mutation_tsp;

component fitness_calc_tsp is
  generic(
    genom_lngt : positive;
    score_sz   : integer;
    pop_sz     : positive;
    townres    : positive;
    num_towns  : positive);
  port (
    clk        : in  std_logic;       -- clock
    rst_n      : in  std_logic;       -- reset (active low)
    decode     : in  std_logic;
    valid      : in  std_logic;
    in_genes   : in  std_logic_vector(2*genom_lngt-1 downto 0);
    data_in    : in  std_logic_vector(2*townres-1 downto 0);
    gene_score : out std_logic_vector(genom_lngt+score_sz-1 downto 0);
    fit        : out std_logic_vector(score_sz-1 downto 0) := (others => '0');
    addr_rom   : out integer;
    ready_out  : out std_logic);
end component fitness_calc_tsp;

component fix_elite_tsp is
  generic(
    genom_lngt : positive;
    score_sz   : integer;
    pop_sz     : positive;
    elite      : positive);
  port (
    clk           : in  std_logic;    -- clock
    rst_n         : in  std_logic;    -- reset (active low)
    decode        : in  std_logic;
    valid         : in  std_logic;
    elite_null    : in  std_logic;
    index         : in  integer;
    fit           : in  std_logic_vector(score_sz-1 downto 0);
    count_parents : in  integer;
    ready_in      : in  std_logic;
    elite_offs    : out int_array(0 to elite-1);
    fit_sum       : out std_logic_vector(score_sz+log2(pop_sz)-1 downto 0);
    max_fit       : out std_logic_vector(score_sz-1 downto 0);
    rd            : out std_logic);
end component fix_elite_tsp;

component coordinates_rom is
  generic (
    townres : integer;
    pop_sz  : integer);
  port (
    addr     : in  integer;
    data_out : out std_logic_vector(2*townres-1 downto 0));
end component coordinates_rom;

component init_generation_rom is
  generic (
    townres   : integer;
    pop_sz    : integer;
    num_towns : integer);
  port (
    addr     : in  integer;
    data_out : out std_logic_vector((num_towns-1)*townres-1 downto 0));
end component init_generation_rom;

component fitness_eval_tsp is
  generic (
    genom_lngt : positive;
    score_sz   : integer;
    pop_sz     : integer;
    elite      : integer;
    townres    : integer;
    num_towns  : integer);
  port (
    clk           : in  std_logic;
    rst_n         : in  std_logic;
    decode        : in  std_logic;
    valid         : in  std_logic;
    in_genes      : in  std_logic_vector(2*genom_lngt-1 downto 0);
    elite_null    : in  std_logic;
```

```
      index         : in  integer;
      count_parents : in  integer;
      gene_score    : out std_logic_vector(genom_lngt+score_sz-1 downto 0);
      elite_offs    : out int_array(0 to elite-1);
      fit_sum       : out std_logic_vector(score_sz+log2(pop_sz)-1 downto 0);
      max_fit       : out std_logic_vector(score_sz-1 downto 0);
      rd            : out std_logic);
  end component fitness_eval_tsp;

  component control_tsp is
    generic(
      genom_lngt : integer;
      max_gen    : positive;
      pop_sz     : integer;
      score_sz   : integer;
      elite      : integer);
    port (
      clk             : in  std_logic;
      rst_n           : in  std_logic;
      fit_eval_rd     : in  std_logic;
      sel_rd          : in  std_logic;
      cross_rd        : in  std_logic;
      mut_rd          : in  std_logic;
      run_ga          : in  std_logic;
      elite_offs      : in  int_array(0 to elite-1);
      data_in_ram2    : in  std_logic_vector(genom_lngt-1 downto 0);
      data_out_cross1 : out std_logic_vector(genom_lngt-1 downto 0);
      data_out_cross2 : out std_logic_vector(genom_lngt-1 downto 0);
      addr_1_c        : out integer;
      addr_2_c        : out integer;
      cnt_parents     : out integer;
      we1_c           : out std_logic;
      we2_c           : out std_logic;
      data_valid      : out std_logic;
      next_gene       : out std_logic;
      ga_fin          : out std_logic;
      cross_out       : out std_logic;
      valid           : out std_logic;
      elite_null      : out std_logic;
      index           : out integer range 0 to pop_sz+1;
      mut_out         : out std_logic;
      flag            : out std_logic;
      decode          : out std_logic;
      sel_out         : out std_logic;
      addr_rom        : out integer;
      run1            : out std_logic;
      run2            : out std_logic;
      run3            : out std_logic;
      load            : out std_logic);
  end component control_tsp;

end package ga_pkg;

-------------------------------------------------------------------------------
-- PACKAGE BODY DECLARATION
-------------------------------------------------------------------------------
package body ga_pkg is
-- empty
end package body ga_pkg;

end architecture str;
```

VHDL RTL κώδικας που υλοποιεί την πράξη της μετάλλαξης.

```
-------------------------------------------------------------------------------
-- Title      : Mutation block
-- Project    : Genetic Algorithm
-------------------------------------------------------------------------------
-- File       : mutation.vhd
-- Author     : Kyriakos Deliparaschos
-- Company    : NTUA/IRAL
-- Created    : 23/03/06
-- Last update: 20/11/06
-- Platform   : Modelsim & Synplify & Xilinx ISE
-- Standard   : VHDL'93
-------------------------------------------------------------------------------
-- Description: This block implements the mutation algorithm
-------------------------------------------------------------------------------
-- Copyright (c) 2006 NTUA
-------------------------------------------------------------------------------
-- revisions  :
-- date        version  author  description
-- 23/03/06    1.1      kdelip  created
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- LIBRARIES
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library work;
use work.ga_pkg.all;
```

```
use work.arith_pkg.all;

-------------------------------------------------------------------------------
-- ENTITY
-------------------------------------------------------------------------------
entity mutation is
  generic(
    genom_lngt : positive := 8;
    mr         : integer  := 25;
    mut_res    : integer  := 8);
  port (
    clk       : in  std_logic;          -- clock
    rst_n     : in  std_logic;          -- reset (active low)
    mutPoint  : in  std_logic_vector(log2(genom_lngt)-1 downto 0);
    mutMethod : in  std_logic_vector(1 downto 0);
    cont      : in  std_logic;
    flag      : in  std_logic;
    rng       : in  std_logic_vector(genom_lngt + mut_res-1 downto 0);
    inGene    : in  std_logic_vector(genom_lngt-1 downto 0);
    rd        : out std_logic;
    mutOffspr : out std_logic_vector(genom_lngt-1 downto 0));
end entity mutation;

-------------------------------------------------------------------------------
-- ARCHITECTURE
-------------------------------------------------------------------------------
architecture rtl of mutation is

  constant WeirdTRUE : std_logic_vector(1 downto 0)                := "01";
  signal   mutout    : std_logic_vector(genom_lngt-1 downto 0)     := (others => '0');
  signal   mutout_p1 : std_logic_vector(genom_lngt-1 downto 0)     := (others => '0');
  signal   count     : std_logic_vector(log2(genom_lngt) downto 0) := (others => '0');
  signal   done      : std_logic                                   := '0';
  signal   done_p    : std_logic                                   := '0';

begin

  process (clk, rst_n)
  begin
    if (rst_n = '0') then
      mutout_p1 <= (others => '0');
      count     <= (others => '0');
      done_p    <= '0';
    elsif clk = '1' and clk'event then
      mutout_p1 <= mutout;
      if flag = '1' then
        done_p <= '0';
      else
        done_p <= done;
      end if;

      if done = '1' or flag = '1' or cont = '0' then
        count <= (others => '0');
      else
        count <= count + 1;
      end if;

    end if;
  end process;

  mutOffspr <= mutout_p1;
  rd        <= done_p;
  mutation : process (mutMethod, mutPoint, rng, inGene, mutout_p1, count, cont, flag, done_p)
  begin

    case done_p is

      when '0' =>
        case cont is

          when '1' =>                   -- mutation block enabled

            case mutMethod is

              when "00" =>              -- one Point mutation
                if rng(mut_res-1 downto 0) > conv_std_logic_vector(mr, mut_res) then
                  mutout <= inGene;
                  done   <= '1';
                else
                  mutout <= inGene xor shl(ext(WeirdTRUE, genom_lngt), mutPoint);
                  done   <= '1';
                end if;

              when "01" =>              -- XOR (masked) mutation

                if rng(mut_res-1 downto 0) > conv_std_logic_vector(mr, mut_res) then
                  mutout <= inGene;
                  done   <= '1';
                else
                  mutout <= inGene xor rng(genom_lngt+mut_res-1 downto mut_res);
                  done   <= '1';
                end if;

              when "10" =>              -- uniform mutation

                if rng(mut_res-1 downto 0) < conv_std_logic_vector(mr, mut_res) and count = ext("00", count'length) then
                  mutout <= inGene xor shl(ext(WeirdTRUE, genom_lngt), count);
                elsif rng(mut_res-1 downto 0) < conv_std_logic_vector(mr, mut_res) and count /= ext("00", count'length)
then
                  mutout <= mutout_p1 xor shl(ext(WeirdTRUE, genom_lngt), count);
```

```
                elsif rng(mut_res-1 downto 0) >= conv_std_logic_vector(mr, mut_res) and count = ext("00", count'length)
then
                  mutout <= inGene;
                else
                  mutout <= mutout_p1;
                end if;

                if count = conv_std_logic_vector(genom_lngt, log2(genom_lngt)+1)-1 then
                  done <= '1';
                else
                  done <= '0';
                end if;

              when others =>
                mutout <= mutout_p1;
                done   <= '0';
            end case;


          when '0' =>                   -- mutation block disabled
            mutout <= mutout_p1;
            if flag = '1' then
              done <= '0';
            else
              done <= done_p;
            end if;

          when others =>

        end case;


      when '1' =>
        mutout <= mutout_p1;
        done   <= done_p;

      when others =>

    end case;
  end process mutation;

end rtl;
```

Μέρος του MATLAB κώδικα που δημιουργήθηκε για τη μοντελοποίηση τον πυρήνα του γενετικού αλγορίθμου.

```
% top m-file for the GA core
% depends on other functions
% Kyriakos Deliparaschos, 2006

clear all;
clc;
tic
global popSz
global elite
global genomlngt
global nParents
global factor
global mut_prob
global fit_limit
global max_gen
global mut_res

%% Generics - Constants
popSz = 8;
elite = 2;
genomlngt = 8;
nParents = 2*(popSz-elite);
max_gen = 100;
fit_limit = 511; % According to fitness evaluation function
factor = 4;
mut_prob = 150; % according to mut_res from VHDL design
mut_res = 8;
cross_method =2;
mut_method =2;
%% Produce random seeds
[seed,seed1,seed2,seed3] = seeds_generator();
%load seeds_tsp
%% Initializations
fit_limit_reached = 0;
max_gen_reached = 0;
current_gen = 0;
genes = cell(1,max_gen+1);
parents = cell(1,max_gen);
offspring = cell(1,max_gen);
offsprings = cell(1,max_gen);
fit_sum = cell(1,max_gen);
max_fit = cell(1,max_gen);
best_fit = cell(1,max_gen);
mutation_rng = cell(1,max_gen);
scale = cell(1,max_gen);
cross_points = cell(1,max_gen);
mut_point = cell(1,max_gen);
best_fit = cell(1,max_gen);
index = 1;
```

```
LFSR_reg = ones(1,genomlngt);
LFSR_reg_1 = ones(1,factor);
LFSR_reg_2 = ones(1,2*log2(genomlngt));
LFSR_reg_3 = ones(1,genomlngt+mut_res);
delete('D:\Designs\GA_eval\sim\ip_vec_ga.vec');
delete('D:\Designs\GA_eval\sim\op_vec_ga.vec');

%% RNG : Creation of popSz genes (ram1) randomly -- Insert various seeds
load_var = 1;
run = 1;

for i=1:popSz
    [genes{index}(i,1:genomlngt)] = rng(load_var,run,seed,genomlngt,LFSR_reg);
    LFSR_reg = genes{index}(i,1:genomlngt);
    load_var = 0;
end

%% Fitness evaluation of first generation
[fit_sum{index},max_fit{index},elite_indexs,best_fit{index},genes{index}] = initial_fit_evaluation(genes{index});
current_gen = 1;

while fit_limit_reached == 0 && max_gen_reached == 0
%% Input test vectors creation
    inputtestvectors(1)= bin2dec(num2str(seed));
    inputtestvectors(2)= bin2dec(num2str(seed1{index}));
    inputtestvectors(3)= bin2dec(num2str(seed2{index}));
    inputtestvectors(4)= binary2integer(seed3{index});
    inputtestvectors(5)= 1;
    inputtestvectors(6)= cross_method;
    inputtestvectors(7)= mut_method;
    dlmwrite('D:\Designs\GA_eval\sim\ip_vec_ga.vec', inputtestvectors, 'delimiter', '\t', 'precision', '%.0f','-append')

%% RNG : Creation of scale matrix for selection (factor random bits =>
%% converted to dec) Insert various seeds
load_var = 1;
run = 1;

if isempty(find(seed1{index})) == 1 && current_gen ~= 1 % Manipulation of zero seed input
    LFSR_reg_1 = temp_scale(nParents,:);
    note = 1;
end

for i=1:nParents+1
    [temp_scale(i,:)] = rng(load_var,run,seed1{index},factor,LFSR_reg_1);
    LFSR_reg_1 = temp_scale(i,:);
    load_var = 0;
end

if current_gen == 1
    scale{index} = bin2dec(num2str(temp_scale(2:nParents+1,:))); % Array of decs
else
    scale{index} = bin2dec(num2str(temp_scale(1:nParents,:))); % Array of decs
end

%% Selection on current generation (parents array contains the selected parents from genes-ram1)
[parents{index}] = selection(genes{index}, scale{index}, fit_sum{index});

%% RNG : Creation of random crosspoints/mutpoint -- Insert various seeds
load_var = 1;
run = 1;

if isempty(find(seed2{index})) == 1 && current_gen ~= 1 % Manipulation of zero seed input
    LFSR_reg_2 = temp_cros_mut_points(nParents/2+1,:);
end

for i=1:nParents/2+1
    [temp_cros_mut_points(i,:)] = rng(load_var,run,seed2{index},2*log2(genomlngt),LFSR_reg_2);
    LFSR_reg_2 = temp_cros_mut_points(i,:);
    cros_mut_points(i,1) = bin2dec(num2str(temp_cros_mut_points(i,1:log2(genomlngt))));
    cros_mut_points(i,2) = bin2dec(num2str(temp_cros_mut_points(i,log2(genomlngt)+1:2*log2(genomlngt))));
    load_var = 0;
    if i~=1
        cross_points{index}(i-1,:) = cros_mut_points(i,:);
        mut_point{index}(i-1) = cros_mut_points(i,2);
    end
end

%% RNG : Creation of random masks -- Insert various seeds
if mut_method ~= 2

    load_var = 1;
    run = 1;
    k=0;
    l=0;

    if isempty(find(seed3{index})) == 1 && current_gen ~= 1 % Manipulation of zero seed input
        LFSR_reg_3 = temp_masks(nParents+1,:);
    end

    for i=1:nParents+1
        [temp_masks(i,:)] = rng(load_var,run,seed3{index},genomlngt + mut_res,LFSR_reg_3);
        LFSR_reg_3 = temp_masks(i,:);
        load_var = 0;
        if i~=1
            %if isequal(rem(i,2),0)
                k=k+1;
                cross_mask{index}(k,:) = temp_masks(i,1:genomlngt);
            %else
            %    l=l+1;
                mut_mask{index}(k,:) = temp_masks(i,:);
```

```
            %end
        end
    end

    for i=1:(popSz-elite)
        mutation_rng{index}(i,1:genomlngt) = mut_mask{index}(i,1:genomlngt);
        mutation_rng{index}(i,genomlngt+1) = bin2dec(num2str(mut_mask{index}(i,genomlngt+1:genomlngt+mut_res)));
    end

else % mut_method == 2
    load_var = 1;
    run = 1;
    ind=1;
    ind2=1;
    zero = 0;
    if isempty(find(seed3{index})) == 1 && current_gen ~= 1
        LFSR_reg_3 = temp_masks((nParents/2)*(genomlngt+2)+1,:);
        zero = 1;
    end
    for i=1:(nParents/2)*(genomlngt+3) + 1 % According to VHDL rng [+1 is for seed]
        [temp_masks(i,:)] = rng(load_var,run,seed3{index},genomlngt + mut_res,LFSR_reg_3);
        LFSR_reg_3 = temp_masks(i,:);
        load_var = 0;
    end
    for j=1:genomlngt+3:(nParents/2 - 1)*(genomlngt+3) + 1
        cross_mask{index}(ind,:) = temp_masks(j+1,1:genomlngt);
        ind=ind+1;
        for k=1:genomlngt
            mut_mask{index}(ind2,:) = temp_masks(j+k+1,:);
            ind2=ind2+1;
        end
    end

    for i=1:(popSz-elite)*genomlngt
        mutation_rng{index}(i,1:genomlngt) = mut_mask{index}(i,1:genomlngt);
        mutation_rng{index}(i,genomlngt+1) = bin2dec(num2str(mut_mask{index}(i,genomlngt+1:genomlngt+mut_res)));
    end

end

%% Crossover performed on parents
[offspring{index}] = crossover(parents{index}, cross_points{index}, cross_mask{index}, cross_method);

%% Mutation performed on parents
[offsprings{index}] = mutation(offspring{index}, mut_point{index}, mutation_rng{index}, mut_method);

%% Fitness evaluation of new generation - offsprings
[fit_sum{index+1},max_fit{index+1},elite_offs,best_fit{index+1},genes{index+1}]                                                    =
fit_evaluation(offsprings{index},elite_indexs,best_fit{index},genes{index});

%% Elite offsprings update
elite_indexs = elite_offs;

%% Output test vectors creation
outputtestvectors(1)=elite_offs(1);
outputtestvectors(2)=elite_offs(2);
outputtestvectors(3)=max_fit{index+1};
dlmwrite('D:\Designs\GA_eval\sim\op_vec_ga.vec', outputtestvectors, 'delimiter', '\t', 'precision', '%.0f','-append')

%% Observer
[fit_limit_reached, max_gen_reached] = observer(max_fit{index+1}, current_gen);
if max_gen_reached==1 || fit_limit_reached == 1;
    inputtestvectors(1)= 0;
    inputtestvectors(2)= 0;
    inputtestvectors(3)= 0;
    inputtestvectors(4)= 0;
    inputtestvectors(5)= 0;
    inputtestvectors(6)= 0;
    inputtestvectors(7)= 0;
    dlmwrite('D:\Designs\GA_eval\sim\ip_vec_ga.vec', inputtestvectors, 'delimiter', '\t', 'precision', '%.0f','-append');
end
index = index + 1;
current_gen = current_gen + 1;
end

%% Execution Time Calculation
toc
calc_time = toc;
```

Συνάρτηση (MATLAB κώδικας) για τη διαδικάσία της επιλογής.

```
function [parents] = selection(genes, scale, fit_sum)

% scale : 1xnParents array of decs
% fit_sum : positive integer
% genes : popSz x genomlngt+1 array
% parents : nParents x genomlngt array

global genomlngt
global popSz
global nParents
global factor

cumSum = 0;
parents = zeros(nParents,genomlngt);
```

```
for i=1:nParents % Creation of "nParents" parents (ram2)

    scalFitSum = floor(fit_sum*scale(i)/(2^factor));

    for j=1:popSz % Run through genes (ram1)
        cumSum = cumSum + genes(j,genomlngt+1);
        if cumSum > scalFitSum
            parents(i,1:genomlngt) = genes(j,1:genomlngt);
            break;
        else
        end
    end
    cumSum = 0;

end
```

Παράρτημα Β

Λεξιλόγιο Αγγλικών Όρων

Algorithmic State Machine (ASM) – Αλγοριθμικές Μηχανές Κατάστασης (ΑΜΚ)
Architecture Body – Σώμα αρχιτεκτονικής
Behavioral modelling – Συμπεριφορική περιγραφή
Block diagram – Διάγραμμα δομικών μονάδων
Cell – Κύτταρο
Chip – Τσιπ, πλακίδιο ολοκληρωμένου κυκλώματος
Circuit schematic – Κυκλωματικό διάγραμμα
CLB (Configurable Logic Block) – Διαμορφώσιμο Λογικό Μπλοκ
Clock – Ρολόι, Χρονικό βήμα
Combinational logic – Συνδιαστική λογική, λογική ταυτόχρονης εκτέλεσης
Component – Δομικά στοιχεία
Concatenated signals – Συνδεδεμένα σήματα
Core – Πυρήνας
Critical path delay – Διαδρομή με τη μέγιστη καθυστέρηση
Cut set rules – Μέθοδος των τομών
Dataflow modelling– Περιγραφή ροής δεδομένων
Declarative part – Τμήμα δήλωσης
Design units – Μονάδες οντότητας
Entity – Οντότητα
Entity Declaration – Δήλωση Οντότητας
Flip-Flop – Μανδαλωτής
Gate-level model –Μοντέλο σε επίπεδο πυλών
Hardware –Υλικό
Latch – Στοιχεία καθυστέρησης
Latency – Καθυστέρηση
Layout – Φυσικό σχέδιο

Least Significant Bit – Δεξιότερο ψηφίο με το μικρότερο βάρος ή λιγότερο σημαντικό ψηφίο
LUT: Look Up Table – Πίνακας Αναζήτησης
Module – Μονάδα
Most Significant Bit (MSB) – Αριστερότερο ψηφίο με το μεγαλύτερο βάρος
Netlist – Χωροδικτύωμα
Overfloat – Υπερχείλιση
Pipeline – Συνεχούς διοχέτευσης ή διοχέτευσης
Pipeline stage – Στάδιο αγωγού
Pipelined multiplier – Πολλαπλασιαστής διοχέτευσης
Pipelining – Τεχνική συνεχούς (διαδοχικής) διοχέτευσης δεδομένων
Placement – Θέση
Port – Θύρα
Routing – Δρομολόγηση
RTL Model – Μοντέλο Επιπέδου Καταχωρητή Μεταφοράς
Schematic – Σχηματικό
Signed digit arithmetic – Αριθμητική προσημασμένων αριθμών
Singleton – Μονότιμο Σύνολο
Statement part – Τμήμα εντολών
Structural Modelling – Περιγραφή Δομικής περιγραφής
Symbol schematic – Σχηματικό σύμβολο
Systolic – Συστολικά
Test vectors – Διανύσματα δοκιμής
Throughput – Ρυθμός λειτουργίας, Ρυθμό-απόδοση
Timing diagram – Διάγραμμα χρονισμού
Truncation – Περικοπή
Two’s complement – Μορφή συμπληρώματος ως προς δυο
VLSI – Τεχνολογία Ολοκλήρωσης Πολύ Μεγάλης Κλίμακας
Wafer – Δίσκος πυριτίου

Βιβλιογραφία

[1] Σ.Γ. Τζαφέστας, *Υπολογιστική Νοημοσύνη – Τόμος Α: Μεθοδολογίες*, Εκδόσεις Ε.Μ.Π., 2002.

[2] M. Wygralak, "Triangular operations, negations, and scalar cardinality of a fuzzy set," *Computing with Words in Information/Intelligent Systems, Foundations*, Heidelberg: Physica-Verlag, 1999, pp. 326-341.

[3] L. Zadeh, "Fuzzy Sets," *Information and Control*, vol. 8, Jun. 1965, pp. 353, 338.

[4] H. Zimmermann, *Fuzzy Set Theory and Its Applications*, Kluwer Academic Publsihers, 1991.

[5] L. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. on Syst., Man, and Cyber.*, vol. SMC-3, 1973, pp. 44, 28.

[6] L.A. Zadeh, "Calculus of fuzzy restrictions," *Fuzzy Sets and Their Applications to Cognitive and Decision Processes*, Academic Press Inc.,U.S., 1975.

[7] E. Mamdani, "Application of fuzzy algorithms for control of simple dynamic plant," *Proc. of the Inst. of Elec. Eng.*, vol. 121, 1974, pp. 1585-1588.

[8] E. Czogala, J. Fodor, and J. Leski, "The Fodor fuzzy implication in approximate reasoning," *Syst. Sci.*, vol. 23, 1997, pp. 17-28.

[9] K. Tanaka, *An Introduction to Fuzzy Logic for Practical Applications*, Springer, 1996.

[10] B.D. Baets and E.E. Kerre, "The generalized modus ponens and the triangular fuzzy data model," *Fuzzy Sets and Systems*, vol. 59, 1993, pp. 305-317.

[11] R.R. Yager, "On the implication operator in fuzzy logic," *Inf. Sci.*, vol. 31, 1983, pp. 141-164.

[12] W. Bandler and L.J. Kohout, "Semantics of implication operators and fuzzy relational products," *Int. J. of Man-Mach. Stud.*, vol. 12, 1980, pp. 89-116.

[13] M. Mizumoto and H. Zimmermann, "Comparison of fuzzy reasoning methods," *Fuzzy Sets and Syst.*, vol. 8, Sep. 1982, pp. 253-283.

[14] D. Dubois and H. Prade, "Fuzzy logics and the Generalized Modus Ponens revisited," *Cyber. and Syst.*, vol. 15, 1984, p. 293.

[15] P. Magrez and P. Smets, "Fuzzy modus ponens: A new model suitable for applications in knowledge-based systems," *Int. J. of Intell. Syst.*, vol. 4, 1989, pp. 181-200.

[16] E.H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *Int. J. of Hum.-Comp. Stud.*, vol. 51, 1999, pp. 135-147.

[17] C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller. I," *IEEE Trans. on Syst., Man and Cyber.*, vol. 20, 1990, pp. 404-418.

[18] C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller. II," *IEEE Trans. on Syst., Man and Cyber.*, vol. 20, 1990, pp. 419-435.

[19] R. Giles, "Lukasiewicz logic and fuzzy set theory," *Int. J. of Man-Mach. Stud.*, vol. 8, 1976, pp. 313-327.

[20] N. Rescher, *Many-Valued Logic*, McGraw Hill, 1969.
[21] J.A. Goguen, "The logic of inexact concepts," *Synthese*, vol. 19, Apr. 1969, pp. 325-373.
[22] M. Mizumoto, S. Fukami, and K. Tanaka, "Some methods of fuzzy reasoning," *Advances in Fuzzy Set Theory and Applications*, Elsevier, 1979, pp. 117-136.
[23] R.R. Yager, "An approach to inference in approximate reasoning," *Inter. J. of Man-Mach. Stud.*, vol. 13, 1980, pp. 323-338.
[24] Z.P. Dienes, "On an implication function in many-valued systems of logic," *J. of Symb. Logic*, vol. 14, Jun. 1949, pp. 95-97.
[25] E. Mamdani, "Application of Fuzzy Algorithms for the Control of a Dynamic Plant," *IEEE Proceedings*, vol. 121, Dec. 1974, pp. 1585-1588.
[26] J. Mendel, "Fuzzy logic systems for engineering: A tutorial," *in Proc. of the IEEE*, vol. 83, 1995, pp. 345-377.
[27] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. on Syst., Man, and Cyber.*, vol. 15, Feb. 1985, pp. 116-132.
[28] J.J. Buckley, "Sugeno type controllers are universal controllers," *Fuzzy Sets and Syst.*, vol. 53, 1993, pp. 299-303.
[29] P. Bauer, S. Nouak, and R. Winkler, "A brief course in fuzzy logic and fuzzy control."
[30] J. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, University of Michigan Press, 1975.
[31] J.R. Koza, D. Andre, F.H. Bennett, and M.A. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann Publishers Inc., 1999.
[32] M. Mitchell, *An introduction to genetic algorithms*, MIT Press, 1996.
[33] J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, 1992.
[34] S. Wright, "Evolution in Mendelian populations," *Genetics*, vol. 16, Mar. 1931, pp. 97-159.
[35] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Longman Publishing Co., Inc., 1979.
[36] W.S. Scott, G. Hamachi, J. Ousterhout, and R.N. Mayo, *1985 VLSI tools: More works by the original Artists*, EECS Department, University of California, Berkeley, 1985.
[37] R.K. Brayton, A.L. Sangiovanni-Vincentelli, C.T. McMullen, and G.D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
[38] S. Sjoholm and L. Lindh, *VHDL for Designers*, Prentice Hall PTR, 1997.
[39] N. Zainalabedin, *VHDL: Analysis and Modeling of Digital Systems*, McGraw-Hill Professional, 1997.
[40] D.E. Moorby and P.R. Thomas, *The Verilog Hardware Description Language*, Kluwer Academic Publishers, Secaucus, New Jersey, U.S.A., 1996.
[41] Xilinx, Inc., "Microblaze processor reference guide," Jun. 2006.
[42] Xilinx, Inc., "Picoblaze 8-bit embedded microcontroller user guide," Nov. 2009.
[43] Altera Corp., "Nios II processor reference handbook," Nov. 2009.
[44] Xilinx, Inc., "Spartan-3 FPGA family: Complete data sheet - DS099," 2008.

[45] Σ.Γ. Τζαφέστας, *Αυτόματος Έλεγχος Γραμμικών και Μη Γραμμικών Συστημάτων Συνεχούς και Διακριτού Χρόνου*, Εκδόσεις Ε.Μ.Π., 2004.
[46] S.G. Tzafestas, *Methods and Applications of Intelligent Control*, Kluwer Academic Publishers, 1997.
[47] K.M. Passino and S. Yurkovich, *Fuzzy Control*, Addison-Wesley Longman Publishing Co., Inc., 1997.
[48] A. Kandel and G. Langholz, *Fuzzy Control Systems*, CRC-Press, 1993.
[49] J. Godjevac, *Neuro-Fuzzy Controllers: Design and Application*, Presses Polytechniques et Universitaires Romandes (PPUR), 1997.
[50] K. Tanaka and H.O. Wang, *Fuzzy Control Systems Design and Analysis: A Linear Matrix Inequality Approach*, Wiley-Interscience, 2001.
[51] L. Zadeh, "Fuzzy sets," *Inf. and Control*, vol. 8, Jun. 1965, pp. 338-353.
[52] M. Togai and H. Watanabe, "A VLSI implementation of a fuzzy-inference engine: Toward an expert system on a chip," *Inf. Sci.*, vol. 38, 1986, pp. 147-163.
[53] H. Watanabe, W. Dettloff, and K. Yount, "A VLSI fuzzy logic controller with reconfigurable, cascadable architecture," *IEEE J. of Solid-State Circ.*, vol. 25, Apr. 1990, pp. 376-382.
[54] K. Shimuzu, M. Osumi, and F. Imae, "Digital fuzzy processor FP-5000," Fuzzy Logic Systems Institute, Iizuka, Fukuoka, Japan: 1992, pp. 539-542.
[55] M. Patyra, J. Grantner, and K. Koster, "Digital fuzzy logic controller: Design and implementation," *IEEE Trans. on Fuzzy Syst.*, vol. 4, Nov. 1996, pp. 439-459.
[56] H. Eichfeld, T. Kunemund, and M. Menke, "A 12b general-purpose fuzzy logic controller chip," *IEEE Trans. on Fuzzy Syst.*, vol. 4, 1996, pp. 460-475.
[57] T. Yamakawa, "High-speed fuzzy controller hardware system: The Mega-FLIPS machine," *Inf. Sci.*, vol. 45, Jul. 1988, pp. 113-128.
[58] T. Yamakawa and T. Miki, "The current mode fuzzy logic integrated circuits fabricated by the standard cmos process," *IEEE Trans. Comput.*, vol. 35, 1986, pp. 161-167.
[59] S. Sjoholm and L. Lindh, *VHDL for Designers*, Prentice Hall PTR, 1997.
[60] D.L. Perry, *VHDL: Programming by Example*, McGraw-Hill Professional, 2002.
[61] K.M. Deliparaschos and S.G. Tzafestas, "A parameterized T-S digital fuzzy logic processor: Soft core VLSI design and FPGA implementation," *Int. J. of Factory Automation, Robotics and Soft Computing*, vol. 3, Jul. 2006, pp. 7-15.
[62] T. Takagi and M. Sugeno, "Derivation of fuzzy control rules from human operator's control actions," *in Proc. of the IFAC Symp. on Fuzzy Inf. Repr. and Dec. Anal.*, 1984, pp. 55-60.
[63] K.M. Deliparaschos, F.I. Nenedakis, and S.G. Tzafestas, "A fast digital fuzzy logic controller: FPGA design and implementation," *in Proc. of 10th IEEE Conf. on Emerging Technologies and Factory Automation, 2005 (IEEE ETFA '05)*, Catania, Sicily: 2005, pp. 259-262.
[64] A. Gabrielli and E. Gandolfi, "A fast digital fuzzy processor," *IEEE Micro*, vol. 19, 1999, pp. 68-79.
[65] D. Wong and M. Flynn, "Fast division using accurate quotient approximations to reduce the number of iterations," *IEEE Trans. Comput.*, vol. 41, 1992, pp. 981-995.
[66] C.J. Jiménez, A. Barriga, and S.S. Solano, "Digital implementation of SISC fuzzy controllers," *in Proc. 3rd Int. Conf. on Fuzzy Logic, Neural Nets and Soft Computing*, 1994, pp. 651-652.

[67] C.J. Jiménez, S. Sánchez-Solano, and A. Barriga, "Hardware implementation of a general purpose fuzzy controller," *Proc. IFSA World Congress*, 1995, pp. 185–188.
[68] K.M. Deliparaschos, F.I. Nenedakis, and S.G. Tzafestas, "Design and implementation of a fast digital fuzzy logic controller using FPGA technology," *Journal of Intelligent and Robotic Systems*, vol. 45, Jan. 2006, pp. 77-96.
[69] P. Madrid, B. Millar, and E. Swartzlander, "Modified Booth algorithm for high radix fixed-point multiplication," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, vol. 1, 1993, pp. 164-167.
[70] Xilinx, Inc, "Using embedded multipliers in Spartan-FPGAs - XAPP467 (v1.1)," May. 2003.
[71] S.F. Oberman, S.F. Oberman, M.J. Flynn, and M.J. Flynn, "An analysis of division algorithms and implementations," *IEEE Trans. on Comput.*, vol. 46, 1995, pp. 833–854.
[72] E.M. Schwarz and M.J. Flynn, *Using a floating-point multiplier's internals for high-radix division and square root*, Stanford University, 1993.
[73] G.P. Moustris and S.G. Tzafestas, "A robust fuzzy logic path tracker for non holonomic mobile robots," *Int. J. on Artif. Intell. Tools*, vol. 14, Nov. 2005, pp. 935-965.
[74] A. Kongmunvattana and P. Chongstivatana, "A FPGA-based behavioral control system for a mobile robot," *in Proc. of the 1998 IEEE Asia-Pacific Conf. on Circ. and Syst. (IEEE APCCAS '98)*, Chiangmai, Thailand: 1998, pp. 759-762.
[75] P. Leong and K. Tsoi, "Field programmable gate array technology for robotics applications," *in Proc. of the 2005 IEEE Int. Conf. on Robotics and Biomimetics (ROBIO '05)*, 2005, pp. 295-298.
[76] R. Reynolds, P. Smith, L. Bell, and H. Keller, "The design of Mars Lander cameras for Mars Pathfinder, Mars Surveyor '98 and Mars Surveyor '01," *IEEE Trans. on Instrumentation and Measurement*, vol. 50, 2001, pp. 63-71.
[77] T. Li, Shih-Jie Chang, and Yi-Xiang Chen, "Implementation of human-like driving skills by autonomous fuzzy behavior control on an FPGA-based car-like mobile robot," *IEEE Trans. on Ind. Elec.*, vol. 50, 2003, pp. 867-880.
[78] V. Salapura, "A fuzzy RISC processor," *IEEE Trans. on Fuzzy Syst.*, vol. 8, 2000, pp. 781-790.
[79] L. Fortuna, M. Lo Presti, C. Vinci, and A. Cucuccio, "Recent trends in fuzzy control of electrical drives: An industry point of view," *Proc. of the 2003 Int. Symp. on Circ. and Syst., 2003. ISCAS '03*, 2003, pp. III-459-III-461 vol.3.
[80] A. Costa, A.D. Gloria, F. Giudici, and M. Olivieri, "Fuzzy logic microcontroller," *IEEE Micro*, vol. 17, 1997, pp. 66-74.
[81] D.L. Hung, "Dedicated digital fuzzy hardware," *IEEE Micro*, vol. 15, 1995, pp. 31-39.
[82] ActivMedia Robotics, "Pioneer 3TM & Pioneer 2TM H8-series operations manual," 2003.
[83] L.E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American J. of Math.*, vol. 79, Jul. 1957, pp. 497-516.
[84] S.G. Tzafestas, K.M. Deliparaschos, and G.P. Moustris, "Fuzzy logic path tracking control for autonomous non-holonomic mobile robots: Design of system on a chip," *J. of Robotics and Autonomous Syst.*, DOI: 10.1016/j.robot.2010.03.014.

[85] Xilinx, Inc., “Embedded systems tools reference manual,” Jun. 2006.
[86] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers In, 1995.
[87] IBM, “On-chip peripheral bus, architecture specifications,” Apr. 2001.
[88] Xilinx, Inc., “Local memory bus (LMB),” Apr. 2005.
[89] Xilinx, Inc., “Fast simplex link (FSL) bus,” Jun. 2007.
[90] Xilinx, Inc., “Connecting customized IP to the microblaze soft processor using the fast simplex link (FSL) channel,” May. 2004.
[91] J. Laumond, *Robot Motion Planning and Control*, Springer, 1998.
[92] S.M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.
[93] Memec Design, “Spartan-3™ MB user's guide,” Jun. 2005.
[94] K.M. Deliparaschos, G.C. Doyamis, and S.G. Tzafestas, “A parameterised genetic algorithm IP - core: FPGA - design, implementation and performance evaluation,” *International Journal of Electronics*, vol. 95, 2008, p. 1149.
[95] K.M. Deliparaschos, G.C. Doyamis, and S.G. Tzafestas, “A parameterized genetic algorithm IP core design and implementation,” *in Proc. of the 4th Int. Conf. on Informatics in Control, Automation and Robotics (ICINCO '07)*, Angers, France: 2007, pp. 417-423.
[96] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., 1989.
[97] Z. Zhu, D. Mulvaney, and V. Chouliaras, “A novel genetic algorithm designed for hardware implementation,” *Intl. J. of Comput. Intell.*, vol. 3, 2006, pp. 281–288.
[98] C. Aporntewan and P. Chongstitvatana, “A hardware implementation of the compact genetic algorithm,” *IEEE Congress on Evolutionary Computation*, 2001, pp. 624–629.
[99] Tu Lei, Zhu Ming-cheng, and Wang Jing-xia, “The hardware implementation of a genetic algorithm model with FPGA,” *in Proc. of 2002 IEEE Int. Conf. on Field-Programmable Tech. (FPT'02)*, 2002, pp. 374-377.
[100] Wallace Tang and Leslie Yip, “Hardware implementation of genetic algorithms using FPGA,” *in Proc. of the 2004 47th Midwest Symp. on Circ. and Syst. (MWSCAS '04)*, 2004, pp. I-549-552.
[101] H. Mostafa, A. Khadragi, and Y. Hanafi, “Hardware implementation of genetic algorithm on FPGA,” *in Proc. of the 21st National Radio Sci. Conf., 2004. NRSC 2004*, 2004, pp. C9-1-9.
[102] M. Sharawi, J. Quinlan, and H. Abdel-Aty-Zohdy, “A hardware implementation of genetic algorithms for measurement characterization,” *9th Int. Conf. on Electr., Circ. and Syst., 2002*, 2002, pp. 1267-1270 vol.3.
[103] M.M. Mano, *Digital Logic and Computer Design*, Prentice Hall College Div, 1979.
[104] N. Zainalabedin, *VHDL: Analysis and Modeling of Digital Systems*, McGraw-Hill Professional, 1997.
[105] M. Mitchell, *An Introduction to Genetic Algorithms*, The MIT Press, 1998.
[106] S.G. Tzafestas, *Soft Computing in Systems and Control Technology*, World Scientific Publishing Company, 1999.
[107] D. Pham and D. Karaboga, *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*, Springer, 2000.
[108] E. Martel, “Solving travelling salesman problems using genetic algorithms,” 2007.
[109] L. Zhang and B. Zhang, “Research on the mechanism of genetic algorithms,” *J. of*

Software, vol. 11, 2000, pp. 945-952.
[110] J.G. Digalakis and K.G. Margaritis, "An experimental study of benchmarking functions for genetic algorithms," *In Proc. of IEEE Conf. on Trans., Syst., Man and Cyber.*, vol. 79, 2002, pp. 3810--3815.
[111] G. Reinelt, "TSPLIB - A traveling salesman problem library," *INFORMS J. on Computing*, vol. 3, Jan. 1991, pp. 376-384.
[112] A. Törn and A. Zilinskas, "Global optimization," *Lecture Notes in Computer Science*, vol. 350, 1989.
[113] H. Mühlenbein, D. Schomisch, and J. Born, "The parallel genetic algorithm as function optimizer," *Parallel Computing*, vol. 17, 1991, pp. 619-632.
[114] E.E. Easom, "A survey of global optimization techniques," M. Eng. Thesis, Univ. Louisville, Louisville, KY, 1990.
[115] G. Moustris and K.M. Deliparaschos, "Tracking control using the strip-wise affine transformation: An experimental SoC design," *European Control Conf. 2009 (ECC'09)*, Budapest, Hungary: 2009.