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

Αρχιτεκτονική Σχεδίαση Ασαφούς Ελεγκτή σε VHDL και Υλοποίηση σε FPGA

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φαίδων-Ιωσήφ Ε. Νενεδάκης

Επιβλέπων : Σπύρος Γ. Τζαφέστας
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

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

Αρχιτεκτονική Σχεδίαση Ασαφούς Ελεγκτή σε VHDL και Υλοποίηση σε FPGA

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φαίδων-Ιωσήφ Ε. Νενεδάκης

Επιβλέπων : Σπύρος Γ. Τζαφέστας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 6η Ιουλίου 2005.

.....	.....	.....
Σπ. Τζαφέστας	Γ. Παπαβασιλόπουλος	Κ. Τζαφέστας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

.....
Φαίδων-Ιωσήφ Ε. Νενεδάκης

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

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Ρομποτικής και Αυτοματισμού της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Πολυτεχνείου. Ευχαριστώ τον επιβλέποντα Καθηγητή μου κ. Σπύρο Τζαφέστα για την ανάθεση του θέματος και την καθοδήγησή του σε όλη τη διάρκεια της εργασίας μου. Ευχαριστώ επίσης, τον Υποψήφιο Διδάκτορα Κυριάκο Δεληπαράσχο για την πολύτιμη βοήθειά του καθώς και την οικογένειά μου για τη στήριξή που μου προσέφερε.

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική βασίζεται στην κατασκευή ενός γρήγορου παραμετρικού ψηφιακού ασαφούς ελεγκτή *(DFLC: Digital Fuzzy Logic Controller)* τύπου *Takagi-Sugeno μηδενικής τάξης*, ο οποίος επεξεργάζεται μόνο τους ενεργούς ασαφείς κανόνες (εκείνους με μη μηδενική δύναμη για τις δοσμένες εισόδους), με υψηλή συχνότητα λειτουργίας, χωρίς μεγάλη περιπλοκότητα. Χρησιμοποιείται μια βελτιστοποιημένη μέθοδος σχεδίασης του ασαφούς ελεγκτή [33] η οποία μειώνει σημαντικά το χρόνο που απαιτείται για την επεξεργασία όλων των ενεργών κανόνων, αυξάνοντας τη συχνότητα δειγματοληψίας. Ο *DFLC* που υλοποιήθηκε επιτυγχάνει εσωτερική συχνότητα λειτουργίας τουλάχιστο *200MHz*, διαθέτοντας δύο *8-bit* εισόδους και μία *12-bit* έξοδο, με εφτά το πολύ τραπεζοειδείς συναρτήσεις συμμετοχής για κάθε είσοδο και με *49* ασαφείς κανόνες. Όλα τα στοιχεία του *DFLC* μοντελοποιήθηκαν αρχικά μέσω του *Simulink* ώστε να επιβεβαιωθεί η σωστή λειτουργία τους και να παραχθούν *διανύσματα δοκιμής (test vectors)* για χρήση σε μετέπειτα προσομοιώσεις. Η αρχιτεκτονική του *DFLC* υλοποιήθηκε σε ένα *Field Programmable Gate Array (FPGA) chip* χρησιμοποιώντας τη γλώσσα περιγραφής υλικού *VHDL (Very high speed integrated circuits Hardware Description Language)* και προηγμένα εργαλεία *θέσης και δρομολόγησης (place and route)*.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Ασαφής, Έλεγχος, VHDL, FPGA, VLSI, pipeline, latency, προσομοίωση, place n’ route, σύνθεση, Takagi-Sugeno.

ABSTRACT

This diploma thesis is based on the implementation of a *Takagi-Sugeno zero-order* type fast parameterized digital fuzzy logic controller *(DFLC: Digital Fuzzy Logic Controller)*, processing only the active rules (rules that give a non-zero contribution for a given input data set), at high frequency of operation, without significant increase in hardware complexity. An improved method of designing the fuzzy controller model [33] is used that significantly reduces the time required to process the active rules and effectively increases the input data processing rate. The *DFLC* implemented in this thesis achieves an internal core processing speed of at least *200MHz*, featuring two *8-bit* inputs and one *12-bit* output, with up to seven trapezoidal shape membership functions per input and a rule base of up to *49* rules. All the elements of the *DFLC* were first modeled using *Simulink* to ensure correct functionality and to help produce *test vectors* for later simulation use. The *DFLC* architecture was implemented in a *Field Programmable Gate Array (FPGA) chip* with the use of a Very high-speed integrated-circuits Hardware-Description-Language *(VHDL)* and advanced synthesis and *place and route* tools.

KEYWORDS

Fuzzy, Control, VHDL, FPGA, VLSI, pipeline, latency, simulation, place n' route, synthesis, Takagi-Sugeno.

ΠΡΟΛΟΓΟΣ

Η ύπαρξη κάθε είδους αυτοματισμού είναι στενότατα συνυφασμένη με τη δυνατότητα πετυχημένου ελέγχου. Ο πιο σημαντικός ελεγκτής είναι ο κλασσικός *PID ελεγκτής (ελεγκτής τριών όρων)* [6]. 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 [7-13]. Η υπεροχή αυτών έναντι του κλασσικού *PID ελεγκτή* είναι τόσο πιο έκδηλη όσο η πολυπλοκότητα και το εύρος εισόδων του εξεταζόμενου συστήματος αυξάνουν.

Η θεωρία των ασαφών συνόλων θεμελιώθηκε στην παρούσα της μορφή περίπου το 1965 από τον Καθηγητή αυτομάτου ελέγχου Lotfi Zadeh στο UC-Bercley [1]. 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 σύνολο [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 (απ’ευθείας) , χωρίς τη μεσολάβηση κάποιας τεχνικής παράστασης, όπως λ.χ. ένας ακριβής μαθηματικός τύπος. Ο Zadeh ανέπτυξε την 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 λόγο της ιδιοσυγκρασίας των Ιαπώνων. Σαν αποτέλεσμα, παρατηρήθηκε ραγδαία αύξηση της έρευνας και των εφαρμογών της ασαφούς λογικής στον τομέα του αυτομάτου ελέγχου *(Πίνακες Παραρτήματος)*. Η πρώτη πρακτική εφαρμογή έγινε από τον Mamdani το 1974 στο Queen Mary College στον έλεγχο της λειτουργίας μιας ατμομηχανής. Ακολούθησε η πρώτη βιομηχανική εφαρμογή του ασαφούς ελεγκτή το 1982 στην Κοπενχάγη, από τον Δανό κατασκευαστή τσιμέντου F.L. Smidth στον έλεγχο της ποιότητας παραγωγής. 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. "*νευρο-ασαφής ελεγκτής*"), αξιοποιώντας τις δυνατότητες του καθενός.

Ένας ασαφής ελεγκτής αποτελείται από ένα *chip (ασαφές chip)* το οποίο έχει τη δυνατότητα να αποθηκεύει και να επεξεργάζεται *ασαφείς κανόνες*. Τα *ασαφή chip* μπορεί να είναι *αναλογικά* ή *ψηφιακά*. Το πρώτο ψηφιακό ασαφές chip κατασκευάστηκε το 1984 στα AT & T Bell Laboratories από τους Masaki Togai και Hiroyuki Watanabe [14]. Επεξεργαζόταν *16* απλούς κανόνες σε *12.5us*. Η κατασκευή αναλογικών *ασαφών chip* άρχισε με τον Yamakawa το 1987 [15-16].

Στην παρούσα διπλωματική, εξετάζεται η κατασκευή ενός *ψηφιακού ασαφούς chip*. 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 chip* είναι ότι δεν έχουν ανάγκη από *A/D* και *D/A μετατροπείς* που περιορίζουν την ταχύτητα ενός *ψηφιακού chip* και ανεβάζουν το κόστος αυτού. Χρησιμοποιώντας *ψηφιακή σχεδίαση* και *FPGA chip* δίνεται η δυνατότητα της εύκολης κατασκευής ενός πρωτότυπου (*prototyping)* με μικρό κόστος, χωρίς την ανάγκη παραγωγής *μητρών* όπως στην περίπτωση του *ASIC*. Η περιγραφή του κυκλώματος έγινε σε γλώσσα *VHDL* [26-27], η οποία προσφέρει δυνατότητα συγγραφής παραμετρικού κώδικα και υποστηρίζεται από τα περισσότερα εργαλεία σύνθεσης.

Στο **κεφάλαιο 1 (ΕΙΣΑΓΩΓΗ)** τίθεται το γενικότερο πρόβλημα του αυτομάτου ελέγχου. Παράλληλα, μελετάται η γενική αρχιτεκτονική των ασαφών συστημάτων. Εν συνεχεία παρατίθενται διάφορες συνδεσμολογίες ασαφών συστημάτων για τον έλεγχο ενός συστήματος. Τέλος, εξετάζεται το μοντέλο της ασαφούς συλλογιστικής τύπου *Takagi Sugeno μηδενικής τάξης* το οποίο εν συνεχεία θα υλοποιήσουμε.

Στο **κεφάλαιο 2 (ΥΛΟΠΟΙΗΣΗ)** περιγράφεται η υλοποίηση του ψηφιακού ασαφούς ελεγκτή. Αρχικά παρουσιάζονται τα χαρακτηριστικά του. Έπειτα προτείνονται δύο αρχιτεκτονικές (μία με αριθμητική συνάρτηση συμμετοχής, και μία με συνάρτηση συμμετοχής βασισμένη σε ROM) και αναλύονται τα επιμέρους δομικά στοιχεία *(blocks)* από τα οποία αποτελούνται. Τέλος, περιγράφονται τα διάφορα βήματα της σχεδίασης του τελικού κυκλώματος.

Στο **κεφάλαιο 3 (ΣΥΜΠΕΡΑΣΜΑΤΑ)** παρουσιάζονται τα συμπεράσματα που προέκυψαν από το σχεδιασμό και την υλοποίηση του ασαφούς ελεγκτή στο προηγούμενο κεφάλαιο.

Στο **κεφάλαιο 4 (ΠΡΟΤΕΙΝΟΜΕΝΗ ΣΥΝΕΧΕΙΑ)** προτείνονται διάφορα θέματα ως συνέχεια της διπλωματικής εργασίας.

Στο **ΠΑΡΑΡΤΗΜΑ** παρατίθενται πίνακες για την ιστορία και τις εφαρμογές των ασαφών συστημάτων καθώς και παραδείγματα VHDL κώδικα που χρησιμοποιήθηκαν για την περιγραφή του ασαφούς ελεγκτή

ΠΕΡΙΕΧΟΜΕΝΑ

ΣΧΗΜΑΤΑ

ΠΙΝΑΚΕΣ

x

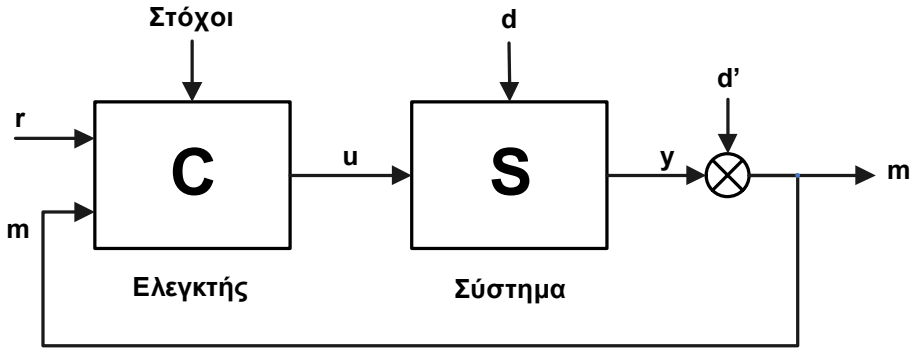
1. ΕΙΣΑΓΩΓΗ

Στο κεφάλαιο αυτό παρουσιάζεται το θεωρητικό υπόβαθρο της εφαρμογής. Αρχίζει με μια συνοπτική περιγραφή των στόχων του *Αυτομάτου Ελέγχου* δίνοντας έμφαση στις δυσκολίες για την επίτευξη τους. Έπειτα, παρουσιάζεται η γενική αρχιτεκτονική των *Ασαφών συστημάτων* και τα συμπεράσματα που αυτή συνεπάγεται. Στη συνέχεια, δίνονται διάφορες *αρχιτεκτονικές Ασαφούς Ελέγχου*. Τέλος, αναλύεται το *μοντέλο Ασαφούς Ελεγκτή τύπου Takagi-Sugeno*.

1.1 Αυτόματος Έλεγχος

Στον αυτόματο έλεγχο, το γενικό πρόβλημα είναι η επίτευξη μιας συνολικής προκαθορισμένης συμπεριφοράς *(r)* ενός συστήματος *(S)* (εντός ορισμένων ορίων διαταραχών εισόδου *(d)*) με την ταυτόχρονη ικανοποίηση συγκεκριμένων απαιτήσεων *(Στόχοι)*, όπως: ευστάθεια, προσαρμοστικότητα, μικρή ευαισθησία, σθεναρότητα κλπ.

Για την επίτευξη της επιθυμητής συμπεριφοράς, μετράμε *(m)* διάφορες μεταβλητές κατάστασης ή εξόδου *(y)* του συστήματος και τις χρησιμοποιούμε για τον προσδιορισμό και υπολογισμό των μεταβλητών (σημάτων) ελέγχου *(u)*. Το γενικό σχήμα ελέγχου έχει τη μορφή του σχήματος ***Σχ1***:



r : σήμα αναφοράς
u : σήμα ελέγχου
y : έξοδος συστήματος
d : διαταραχή
d': θόρυβος μέτρησης
m: μετρούμενο σήμα

Σχ1: *Γενικό δομικό σχήμα αυτομάτου ελέγχου*

Όλες οι μέθοδοι ελέγχου βασίζονται στο συνδυασμό τριών στοιχείων:

1.1.1 ενός **μοντέλου** *(αντικατοπτρίζει την πληροφορία για τη λειτουργία του συστήματος και τις διαταραχές)*

1.1.2 ενός **κριτηρίου** *(καθορίζει την επιθυμητή συμπεριφορά του συστήματος είτε υπό μορφή προδιαγραφών της εξόδου είτε υπό μορφή συνάρτησης κόστους/συμπεριφοράς)*

1.1.3 μιας **στρατηγικής ελέγχου** *(καθορίζει τον τρόπο ικανοποίησης των απαιτήσεων του κριτηρίου όταν είναι γνωστό το μοντέλο του συστήματος και των διαταραχών)*

1.1.1 Μοντέλο

Η διαδικασία μοντελοποίησης συνίσταται στη δημιουργία ενός παραμετρικού μοντέλου το οποίο έχει την ίδια δυναμική συμπεριφορά με το φυσικό σύστημα. Ωστόσο, σε περίπτωση που το σύστημα είναι πολύπλοκο, είναι πολύ δύσκολο να προκύψουν οι διάφοροι μαθηματικοί ή φυσικοί νόμοι που περιγράφουν τη συμπεριφορά του. Ένα άλλο θέμα

είναι η μοντελοποίηση της ασάφειας στο σύστημα που προέρχεται τόσο από το θόρυβο των μετρήσεων όσο και από το αβέβαιο περιβάλλον στο οποίο λειτουργεί το σύστημα, αλλά και από το ίδιο το μοντέλο του συστήματος, σε περίπτωση που αυτό προήλθε από την εξέταση πεπερασμένων ζευγών εισόδων-εξόδων. Υπό αυτό το πρίσμα, ακρίβεια στη μοντελοποίηση δεν είναι μόνο η ικανότητα παραγωγής μιας αριθμητικής τιμής η οποία ελαχιστοποιεί (για παράδειγμα) το μέσο τετραγωνικό σφάλμα αλλά και η ικανότητα να είμαστε ακριβείς και δίκαιοι όσον αφορά την αβεβαιότητά της.

1.1.2 Κριτήριο

Το κριτήριο (συνάρτηση συμπεριφοράς ή κριτήριο συμπεριφοράς) μπορεί να έχει ποικίλες μορφές, από καθαρά μαθηματικές μέχρι καθαρά λογικές και γλωσσικές. Γενικά, εκφράζει ένα δείκτη ή μέτρο της ποιότητας του αποτελέσματος (προϊόντος, εξόδου) που δίνει το σύστημα, κάτω από την επίδραση της στρατηγικής ή νόμου ελέγχου (ελεγκτή).

1.1.3 Στρατηγική ελέγχου

Δυστυχώς, τα εργαλεία που εξασφαλίζουν έλεγχο με αλγοριθμικό τρόπο είναι εφαρμόσιμα σε γραμμικά μόνο μοντέλα συστημάτων. Ωστόσο, τις περισσότερες φορές, προκύπτει η ανάγκη ελέγχου πολύπλοκων συστημάτων με μη γραμμική συμπεριφορά.

Τα παραπάνω προβλήματα οδήγησαν τον Zadeh στην ανάπτυξη των ασαφών συστημάτων (1965) [1].

1.2 Γενική Αρχιτεκτονική Ασαφών Συστημάτων

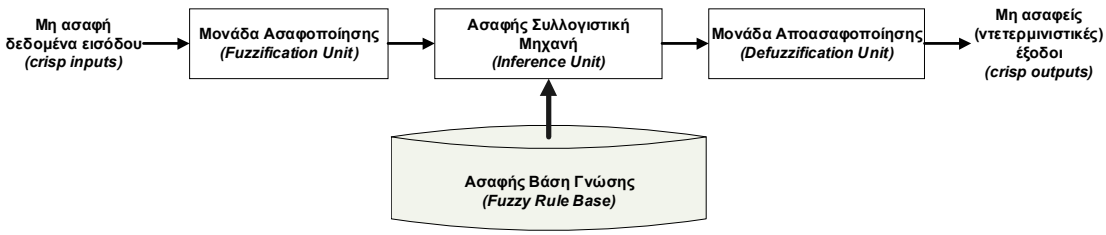
Η γενική αρχιτεκτονική (δομή) των ασαφών συστημάτων εικονίζεται στο ***Σχ2***, και περιλαμβάνει τις παρακάτω μονάδες:

1.2.1 μία **βάση ασαφών κανόνων** της μορφής ΕΑΝ–ΤΟΤΕ *(IF-THEN rules) (η οποία αντιπροσωπεύει τη γνώση της λειτουργίας του συστήματος)*

1.2.2 μία **μονάδα ασαφοποίησης** *(η οποία μετατρέπει τα δεδομένα εισόδου σε ασαφή σύνολα)*

1.2.3 μία **ασαφή συλλογιστική μηχανή** *(η οποία αποτελεί το μηχανισμό εξαγωγής ασαφών συμπερασμάτων)*

1.2.4 μια **μονάδα απο-ασαφοποίησης** *(η οποία μετατρέπει τα ασαφή συμπεράσματα/αποφάσεις σε σαφώς καθορισμένη μορφή)*



Σχ2: *Γενική αρχιτεκτονική ασαφούς Συστήματος*

1.2.1 Ασαφής βάση γνώσης

Πρακτικά, η μονάδα αυτή αποτελεί ένα τρόπο αναπαράστασης της λειτουργίας του συστήματος. Πιο αναλυτικά, αποτελείται από ένα σύνολο κανόνων της μορφής:

$R^l : IF\ x_1\ is\ A_1^l\ AND\ ...\ AND\ x_n\ is\ A_n^l\ THEN\ y\ is\ B^l$

(πχ.ΕΑΝ x_1 είναι "Μικρό" ΚΑΙ x_1 είναι "Μεγάλο" ΤΟΤΕ y είναι "Μεσαίο")

όπου τα A_i^l και B^l είναι ασαφή σύνολα *(fuzzy sets)* επί των $X_i \subset R$ και $Y \subset R$ αντίστοιχα, και $x = [x_1,...,x_n] \subset X_1 \times ... \times X_n$, $y \subset Y$ είναι οι

γλωσσικές μεταβλητές *(linguistic variables)*. Οι αριθμητικές τιμές των γλωσσικών μεταβλητών προκύπτουν από τις συναρτήσεις συμμετοχής *(membership functions)* των αντίστοιχων ασαφών συνόλων. Ο δείκτης n αναφέρεται στον αριθμό των υποθέσεων *(antecedents)* του κανόνα. Ένας κανόνας για να έχει νόημα πρέπει να έχει τουλάχιστον μία υπόθεση αλλά όχι περισσότερες από τις εισόδους του ασαφούς συστήματος. Πρέπει, επίσης, να έχει τουλάχιστον ένα συμπέρασμα *(consequent)* αλλά όχι περισσότερα από τις εξόδους του ασαφούς συστήματος. Ο τύπος του κανόνα που παρουσιάστηκε έχει απλοϊκή μορφή χωρίς ωστόσο να χάνει τη γενικότητά του καθώς πολύπλοκες μορφές κανόνων (πχ. *multiple-consequents, incomplete, mixed, statement, comparative, unless, quantifier rules*) μπορούν να αναχθούν σε αυτή. Ο δείκτης l, τέλος, αναφέρεται στον αριθμό του κανόνα ο οποίος δεν μπορεί να ξεπερνάει το αριθμό των δυνατών συνδυασμών μεταξύ των ασαφών συνόλων των εισόδων.

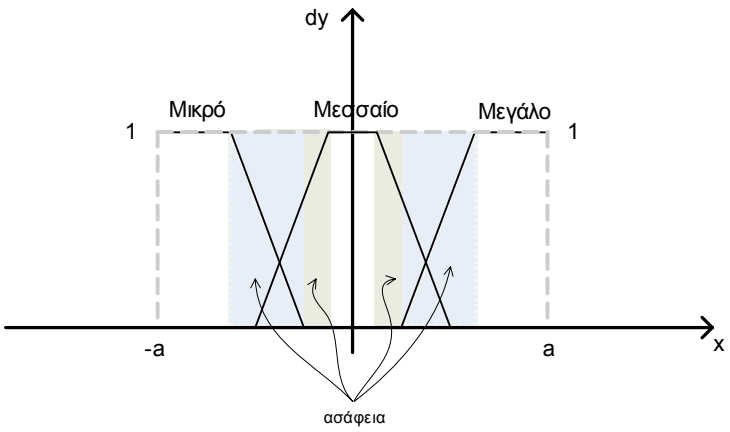
Οι κανόνες μπορούν να προκύψουν είτε από τη γνώση/εμπειρία του χειριστή *(model-based approach)* είτε από την εξέταση ζευγών εισόδων-εξόδων *(model-free approach)*.

1.2.2 Μονάδα ασαφοποίησης

Πρακτικά, η μονάδα αυτή χρειάζεται ώστε να ενεργοποιεί τους κανόνες. Συγκεκριμένα, απεικονίζει κάθε πραγματική είσοδο $x_i \in X_i \subset R$ *(crisp input)* σε ένα αριθμό ασαφών συνόλων A_{ij}, $\text{supp}(A_{ij}) = X_i$ με τον ορισμό ισάριθμων συναρτήσεων συμμετοχής *(membership functions)* $\mu_{ij}(x_i)$. Ο δείκτης j αναφέρεται στον αριθμό των ασαφών συνόλων που αντιστοιχούν σε μία πραγματική είσοδο του συστήματος. Ένας τυπικός διαχωρισμός μιας εισόδου ορισμένης στο διάστημα [-a,a] φαίνεται στο ***Σχ3***. Παρατηρούμε ότι η μοντελοποίηση της ασάφειας γίνεται μέσω της συνάρτησης συμμετοχής στις περιοχές όπου αυτή παίρνει τιμές διάφορες του 0 $(x_i \notin A_{ij})$ και του 1 $(x_i \in A_{ij})$. Σε αυτές τις περιοχές λέμε ότι το x_i ανήκει στο A_{ij} σε βαθμό $\mu_{ij}(x_i) \in [0,1]$. Μέσω του χωρισμού του

χώρου των εισόδων σε ασαφή σύνολα μετατρέπουμε τις αριθμητικές εισόδους σε λεκτικές τις οποίες μπορεί να επεξεργαστεί η ασαφής συλλογιστική μηχανή.

Η μέθοδος ασαφοποίησης συνίσταται στην επιλογή των συναρτήσεων συμμετοχής. Τυπικές επιλογές τέτοιων συναρτήσεων είναι: τριγωνικές *(Triangular)*, τραπεζοειδείς *(Trapezoidal)* και γκαουσιανές *(Gaussian)* . Γενικά μας ενδιαφέρει να έχουν όσο το δυνατόν μικρότερο υπολογιστικό κόστος και να καλύπτουν μεγάλο εύρος περιπτώσεων χρησιμοποιώντας λίγες παραμέτρους. Στις εφαρμογές, συνήθως προτιμάται η τραπεζοειδής η οποία εμπεριέχει την τριγωνική, είναι εύκολα υπολογίσιμη και η εμπειρία έχει δείξει ότι οδηγεί σε καλύτερα αποτελέσματα από την γκαουσιανή.



Σχ3: *Τυπικός διαχωρισμός εισόδου σε ασαφή σύνολα*

1.2.3 Ασαφής Συλλογιστική μηχανή

Πρακτικά, η μονάδα αυτή αποτελεί τον πυρήνα του ασαφούς συστήματος και περιέχει τη λογική λήψης αποφάσεων. Πιο αναλυτικά, είναι μια απεικόνιση από τα ασαφή σύνολα του χώρου των εισόδων $X_1 \times ... \times X_n$ του συστήματος στα ασαφή σύνολα του χώρου της εξόδου Y. Η απεικόνιση αυτή γίνεται χρησιμοποιώντας τις αρχές της ασαφούς λογικής και είναι μη γραμμική.

Η διαδικασία εξαγωγής συμπεράσματος από το σύνολο των κανόνων χωρίζεται σε τρεις φάσεις, εκ των οποίων η τελευταία είναι προαιρετική καθώς σχετίζεται με τη μεθοδολογία απο-ασαφοποίησης:

I. ***(aggregation)*** Πρώτα, υπολογίζουμε την προσαρμοστικότητα/δύναμη *(firing strength)* του κάθε κανόνα:

$$\mu_l(x_1,...,x_n) = \mu_{A_1^l}(x_1) * ... * \mu_{A_n^l}(x_n)$$

Ο τελεστής "$*$ " αποτελεί τη μαθηματική υλοποίηση του "AND" και είναι μια $t-norm$ (είναι μια συνάρτηση στο [0,1] που ικανοποιεί την αντιμεταθετική και προσεταιριστική ιδιότητα, είναι μονότονη με ουδέτερο στοιχείο το 1). Το αποτέλεσμα αντιπροσωπεύει το βαθμό με τον οποίο ικανοποιείται η υπόθεση του l κανόνα και είναι πραγματικός αριθμός. Τα πιο γνωστά "μέτρα" *("νόρμες")* που έχουν προταθεί είναι:

- Łukasiewicz t-norm: $x *_L y = \max(0, x + y - 1)$
- Gödel t-norm *(νόρμα ελαχίστου)*: $x *_G y = \min(x, y)$
- Product t-norm *(νόρμα γινομένου)*: $x *_\Pi y = x \cdot y$

Στις εφαρμογές συνήθως χρησιμοποιείται η Gödel t-norm η οποία έχει το λιγότερο υπολογιστικό κόστος και είναι η μόνη από τις τρεις που είναι *idempotent*.

II. ***(implication)*** Έπειτα, από κάθε κανόνα προκύπτει το συνεπαγόμενο ασαφές σύνολο *(implied fuzzy set)*:

$$\mu_{\hat{B}^l}(y) = \mu_l(x_1,...,x_n) * \mu_{B^l}(y)$$

Ο τελεστής "$*$" είναι και εδώ μια $t-norm$. Στις εφαρμογές συνήθως χρησιμοποιείται η *Product t-norm*. Σημειώνεται ότι η απαίτηση για χρησιμοποίηση $t-norm$, δηλαδή για υλοποίηση του λογικού "AND" μεταξύ της πίστης της υπόθεσης ($\mu_l(x_1,...,x_n)$) και του συμπεράσματος ($\mu_{B^l}(y)$) προέρχεται από τον *"γενικευμένο κανόνα του θέτειν" (Generalized Modus Ponens)*. Το συνεπαγόμενο ασαφές σύνολο $\hat{B}^l$ έχει στήριγμα *(support)* το πεδίο ορισμού $Y \subset R$ της εξόδου y.

III. ***(overall implication)*** Συνδυάζουμε όλα τα συνεπαγόμενα ασαφή σύνολα κάθε κανόνα σε ένα, δημιουργώντας το ολικό συνεπαγόμενο ασαφές σύνολο *(overall implied fuzzy set)*:

$\mu_{\hat{B}}(y) = \mu_{\hat{B}^1}(y) \square \ldots \square \mu_{B^k}(y)$, όπου k ο αριθμός των κανόνων.

Ο τελεστής "$\square$" αποτελεί τη μαθηματική υλοποίηση του "OR" και είναι μια $t-conorm$ (ή $s-norm$) (είναι μια συνάρτηση στο [0,1] που ικανοποιεί την αντιμεταθετική και προσετεριστική ιδιότητα, είναι μονότονη με ουδέτερο στοιχείο το 0). Οι πιο γνωστές νόρμες που έχουν προταθεί είναι:

- Bounded sum t-conorm: $x \square_L y = \min(1, x + y)$
- Maximum t-conorm: $x \square_G y = \max(x, y)$
- Probabilistic sum t-conorm: $x \square_{\Pi} y = x + y - x \cdot y$

Στις εφαρμογές συνήθως χρησιμοποιείται η *Bounded sum t-conorm*.

1.2.4 Μονάδα απο-ασαφοποίησης

Πρακτικά, διαμορφώνει την έξοδο του συστήματος. Πιο αναλυτικά, αποτελεί μια απεικόνιση από το χώρο των συνεπαγόμενων ασαφών συνόλων ή του ολικού συνεπαγόμενου ασαφούς συνόλου, στο πραγματικό χώρο κάθε εξόδου. Έχουν προταθεί διάφοροι αλγόριθμοι απο-ασαφοποίησης χωρίς ωστόσο να υπάρχουν επιστημονικά κριτήρια που να επιβάλουν συγκεκριμένη επιλογή αλγορίθμου. Χωρίζονται σε δύο κατηγορίες:

1η κατηγορία

Αλγόριθμοι οι οποίοι χρησιμοποιούν τα συνεπαγόμενα ασαφή σύνολα. (Συνεπώς, παραλείπεται το *3ο* βήμα στην ασαφή συλλογιστική μηχανή). Παραδείγματα τέτοιων αλγορίθμων είναι οι εξής:

- Center of Gravity (COG)

$$y = \frac{\sum_{l=1}^{k} b_l \int_Y \mu_{\hat{B}^l}(y)dy}{\sum_{l=1}^{k} \int_Y \mu_{\hat{B}^l}(y)dy} \quad , \; b_l \text{: το κέντρο της περιοχής του } B^l$$

- Center-Average

$$y = \frac{\sum_{l=1}^{k} b_l \sup_Y \{\mu_{\hat{B}^l}(y)\}}{\sum_{l=1}^{k} \sup_Y \{\mu_{\hat{B}^l}(y)\}} \quad , \; b_l \text{: το κέντρο της περιοχής του } B^l$$

2η κατηγορία

Αλγόριθμοι οι οποίοι χρησιμοποιούν το ολικό συνεπαγόμενο ασαφές σύνολο. (Συνεπώς, συμπεριλαμβάνεται το *3ο* βήμα στην ασαφή συλλογιστική μηχανή). Παραδείγματα τέτοιων αλγορίθμων είναι οι εξής:

- Max criterion

$$y = \sup_Y \{\mu_{\hat{B}}(y)\}$$

- Mean of Maximum

$$y = \frac{\int_Y y\mu_{\hat{B}^*}(y)dy}{\int_Y \mu_{\hat{B}^*}(y)dy} \quad , \quad \mu_{\hat{B}^*}(y) = \begin{cases} 1 & \mu_{\hat{B}}(y) = \sup_Y \{\mu_{\hat{B}}(y)\} \\ 0 & otherwise \end{cases}$$

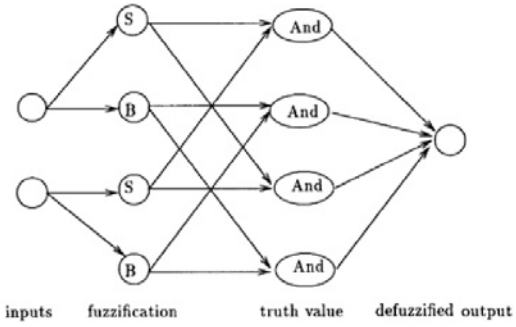
- Center of Area (COA)

$$y = \frac{\int_Y y\mu_{\hat{B}}(y)dy}{\int_Y \mu_{\hat{B}}(y)dy}$$

(Στην περίπτωση ψηφιακών ελεγκτών τα ολοκληρώματα των τύπων απο-ασαφοποίησης μετατρέπονται σε αθροίσματα)

Από την περιγραφή της δομής τους συμπεραίνουμε τα εξής για τα ασαφή συστήματα:

- χωρίζουν το χώρο κάθε εισόδου σε περιοχές ανάλογα με την επιλογή των ασαφών συνόλων
- αποτελούν μη γραμμικές απεικονίσεις από το χώρο των εισόδων στο χώρο των εξόδων.
- προσφέρονται πάρα πολλές εναλλακτικές απεικονίσεις με διαφορετική επιλογή των συναρτήσεων συμμετοχής των ασαφών συνόλων των εισόδων του συστήματος, των *t-norm* του *aggregation* και *implication*, της *t-conorm* της *overall implication*, των *IF-THEN* κανόνων και της μεθόδου απο- ασαφοποίησης.
- είναι καθολικοί προσεγγιστές *(universal approximators)*, πράγμα το οποίο πρακτικά σημαίνει ότι με κατάλληλη επιλογή παραμέτρων *(tuning)* μπορεί να επιτευχθεί η επιθυμητή συμπεριφορά (χωρίς ωστόσο να υπάρχει καθορισμένος τρόπος επιλογής των παραμέτρων)
- μοντελοποιούν την ασάφεια με μαθηματικά εύκολο τρόπο χρησιμοποιώντας ασαφή σύνολα, χωρίς να αυξάνουν την υπολογιστική πολυπλοκότητα
- είναι υβριδικά συστήματα καθώς χρησιμοποιούν τόσο αριθμητικά δεδομένα *(numerical data)* όσο και γλωσσική γνώση *(linguistic knowledge)* για την περιγραφή του συστήματος
- ισοδυναμούν με ένα νευρωνικό δίκτυο με δύο κρυμμένα στρώματα όπως φαίνεται στο ***Σχ4***

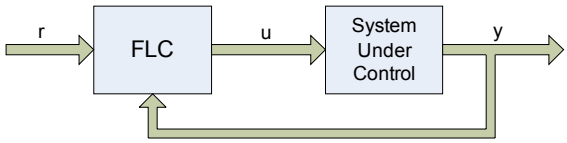


Σχ4: *Νευρωνικό ισοδύναμο Ασαφούς Συστήματος*

1.3 Αρχιτεκτονικές Ασαφούς Ελέγχου

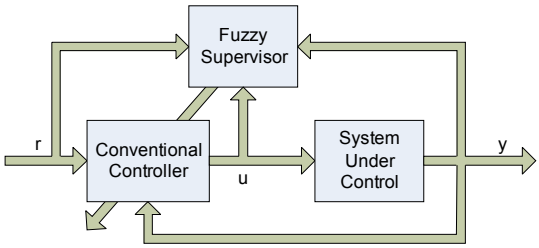
Ένα ασαφές σύστημα μπορεί να χρησιμοποιηθεί για έλεγχο ενός συστήματος με τους ακόλουθους τρόπους:

1. σαν κομμάτι του κλειστού συστήματος *(Fuzzy Logic Controller – FLC)*



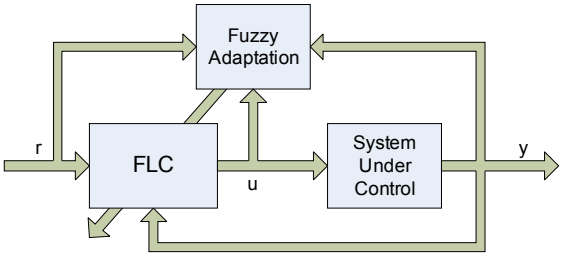
Σχ5: *Το ασαφές σύστημα αποτελεί κομμάτι του κλειστού συστήματος*

2. σαν επιτηρητής ενός συμβατικού μη-ασαφούς ελεγκτή *(Fuzzy supervisor)*



Σχ6: *Το ασαφές σύστημα επιτηρεί/ρυθμίζει ένα συμβατικό ελεγκτή*

3. σαν προσαρμοστικός ελεγκτής *(Fuzzy Adaptation)* αποτελούμενος από ένα βασικό ασαφές σύστημα, κατευθυνόμενο/προσαρμοζόμενο από έναν ασαφή επιτηρητή/επόπτη *(πχ control rule modifier)*



Σχ7: *Ο ασαφής ελεγκτής κατευθύνεται/ρυθμίζεται από ένα ασαφή προσαρμοστικό μηχανισμό*

1.4 Μοντέλο Ασαφούς Ελεγκτή Τύπου Takagi-Sugeno

Οι δύο πιο πετυχημένοι μηχανισμοί ασαφούς συλλογισμού είναι:

- Η μέθοδος *Mamdani* [2]
- Η μέθοδος *Takagi-Sugeno* [3-4]

Παρακάτω θα αναλυθεί η δεύτερη η οποία έχει προσελκύσει περισσότερο το ενδιαφέρον τα τελευταία χρόνια λόγω της απλότητας αλλά και της αποτελεσματικότητάς της.

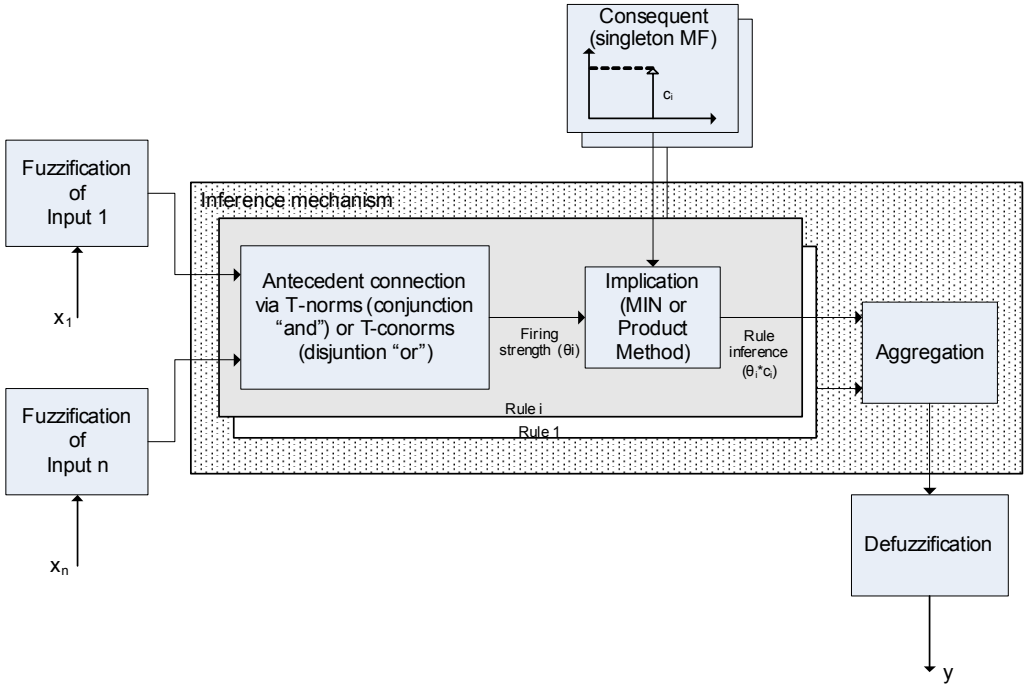
Η μέθοδος *Takagi-Sugeno* ονομάζεται αλλιώς και *"συναρτησιακή συλλογιστική"* αφού το συμπέρασμα των κανόνων δίνεται με τη μορφή γραμμικών συναρτήσεων:

$$R^l:\ IF\ \ x_1\ \ is\ \ A_1^l\ \ AND\ \ ...\ \ AND\ \ x_n\ \ is\ \ A_n^l\ \ THEN\ \ \ y\ \ is\ \ c_0^l + c_1^l x_1 + ... + c_n^l x_n$$

Στην ειδική περίπτωση όπου χρησιμοποιούμε μόνο το σταθερό όρο c_0^l *(singleton)* στην έξοδο των κανόνων λέμε ότι έχουμε την *"απλοποιημένη συναρτησιακή συλλογιστική Takagi-Sugeno μηδενικού βαθμού"* ή αλλιώς *"Takagi-Sugeno zero-order"* :

$$R^l:\ IF\ \ x_1\ \ is\ \ A_1^l\ \ AND\ \ ...\ \ AND\ \ x_n\ \ is\ \ A_n^l\ \ THEN\ \ \ y\ \ is\ \ c_0^l$$

Η μέθοδος *Takagi-Sugeno μηδενικού βαθμού* φαίνεται γραφικά στο ***Σχ8***:



Σχ8: *Μέθοδος Takagi-Sugeno μηδενικού βαθμού*

2. ΥΛΟΠΟΙΗΣΗ

Στο κεφάλαιο αυτό περιγράφεται η υλοποίηση ενός ασαφούς ελεγκτή τύπου *Takagi-Sugeno μηδενικού βαθμού*. Στο πρώτο κεφάλαιο ορίζονται οι προδιαγραφές του. Έπειτα, δίνονται δύο εναλλακτικές δομές για την υλοποίησή του και περιγράφονται αναλυτικά. Τέλος, εξηγείται η διαδικασία που ακολουθήθηκε βήμα προς βήμα μέχρι τη φόρτωση του κυκλώματος στο *FPGA chip* της πλακέτας.

2.1 Χαρακτηριστικά Ελεγκτή

Παρακάτω θα υλοποιήσουμε έναν ασαφή ελεγκτή που χρησιμοποιεί το μοντέλο *Takagi-Sugeno μηδενικού βαθμού*, με τα χαρακτηριστικά του ***Πίνακα1***.

Η Μέθοδος απο-ασαφοποίησης *Weighted average* έχει τον τύπο:

$$y = \frac{\sum_{l=1}^{m} \mu_l(x_1,...,x_n) \cdot c_0^l}{\sum_{l=1}^{m} \mu_l(x_1,...,x_n)}$$

όπου $\mu_l(x_1,...,x_n)$ είναι η δύναμη του l κανόνα, που στη συγκεκριμένη περίπτωση δίνεται από τον τύπο $\mu_l(x_1,...,x_n) = \min_{k=1}^{n}\left(\mu_{A_k^l}(x_k)\right)$, n ο αριθμός υποθέσεων του κανόνα, m ο αριθμός των κανόνων και c_0^l το συμπέρασμα κάθε κανόνα.

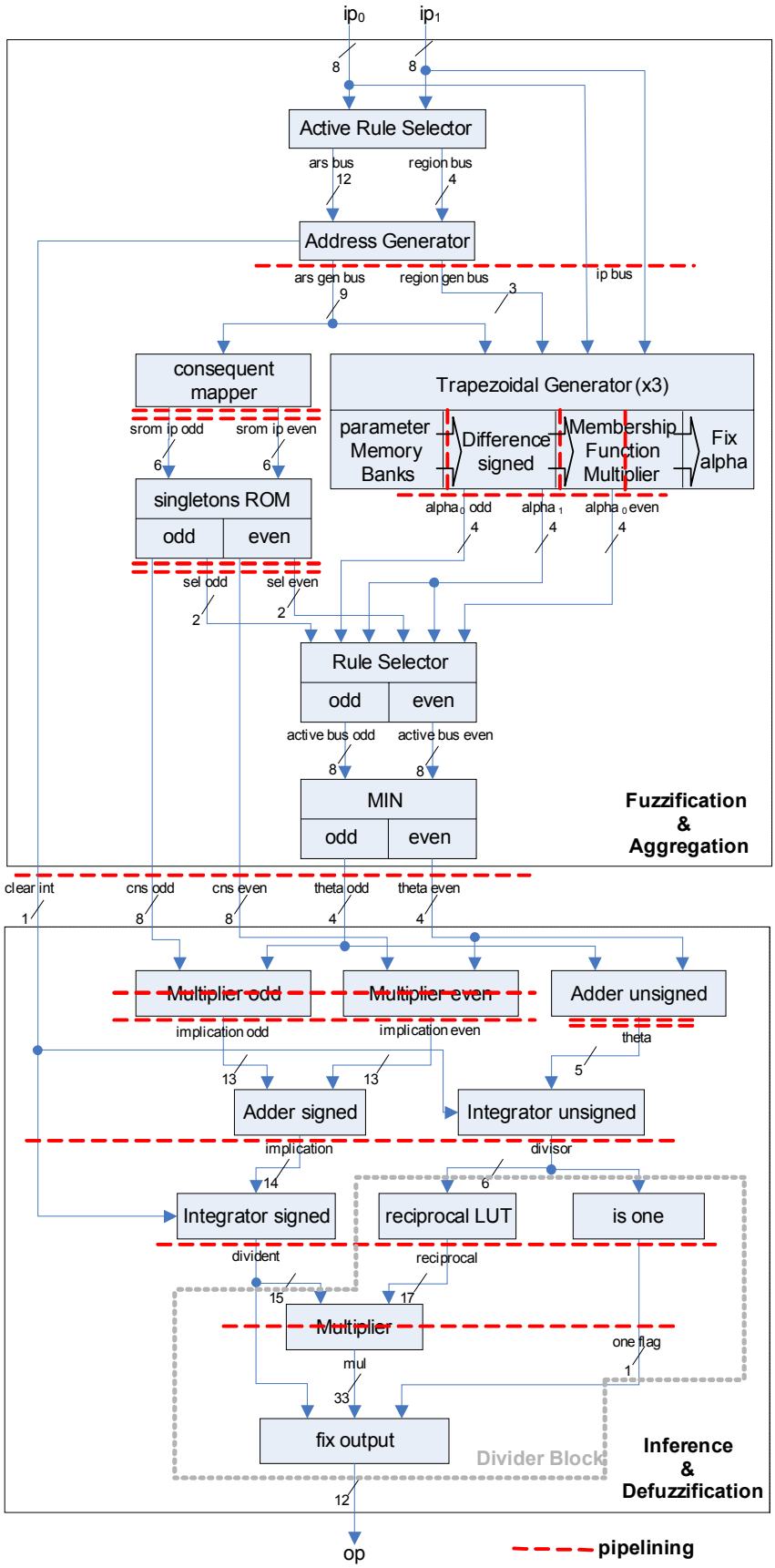
Πίνακας1: *Χαρακτηριστικά Ασαφούς Ελεγκτή*

Είσοδοι (Inputs)	**2**
Έξοδοι (Outputs)	**1**
Συναρτήσεις συμμετοχής (Membership Functions)	**7 Trapezoidal για κάθε είσοδο**
Ανάλυση Εισόδου (Input Resolution)	**8bit (signed)**
Ανάλυση Εξόδου (Output Resolution)	**12bit (signed)**
Ανάλυση συνάρτησης συμμετοχής (Membership Function degree of truth Resolution)	**4bit (unsigned)**
Ανάλυση Συμπεράσματος (Consequent Resolution)	**8bit (signed)**
Μέγιστος αριθμός κανόνων (Maximum number of inference rules)	**49 (= 7^2)**
Μέγιστος βαθμός επικάλυψης των ασαφών συνόλων (Membership function overlapping degree)	**2**
Τρόπος εξαγωγής της δύναμης του κανόνα (Aggregation Method)	**Gödel t-norm (min)**
Τρόπος εξαγωγής του συνεπαγόμενου ασαφούς συνόλου (Implication Method)	**Product t-norm**
Μέθοδος απο-ασαφοποίησης (Defuzzification method)	**Weighted average**

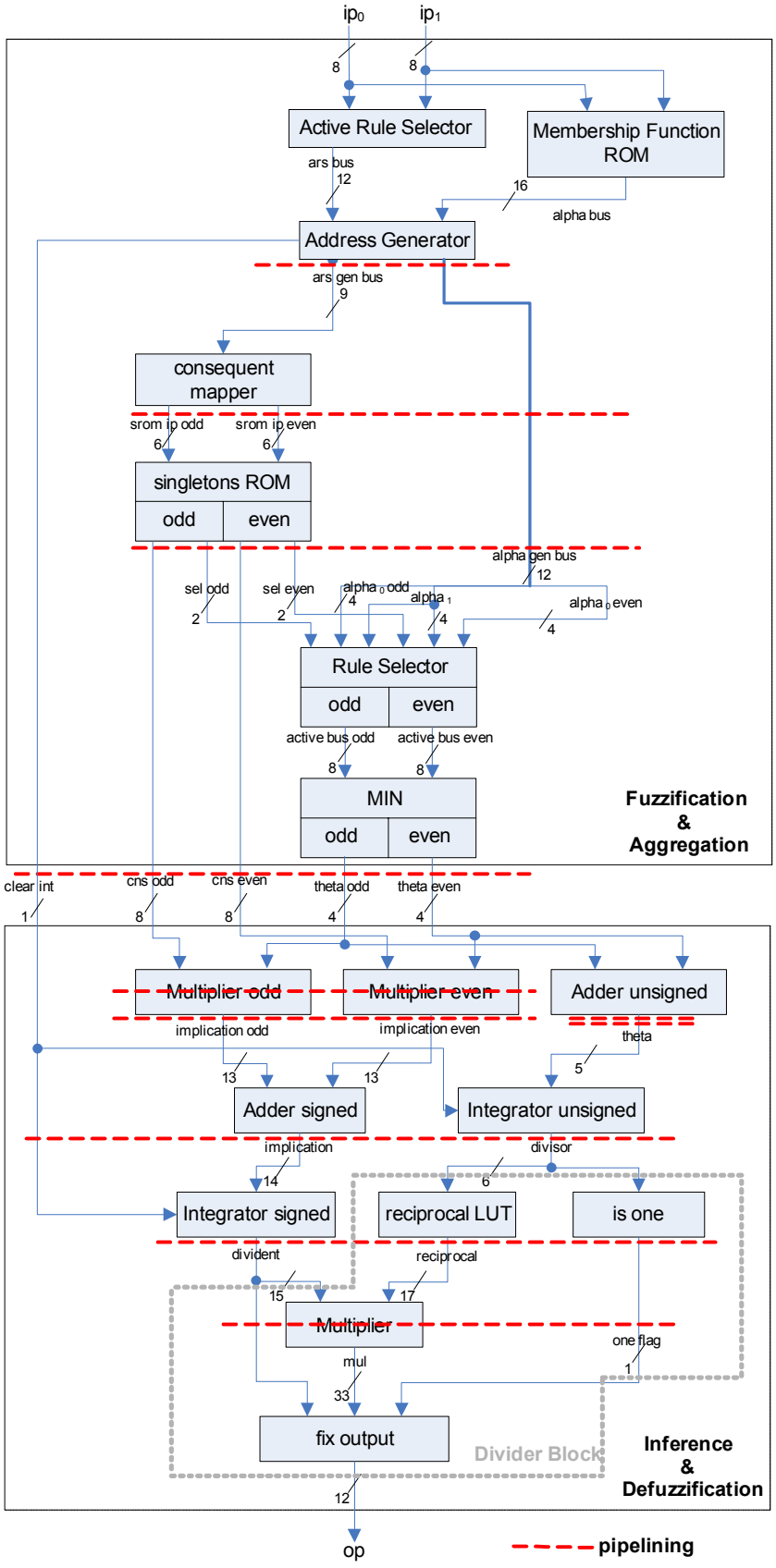
2.2 Δομή Ελεγκτή

Στα σχήματα ***Σχ9,10*** παρουσιάζονται οι δύο εναλλακτικές αρχιτεκτονικές που χρησιμοποιήθηκαν για την κυκλωματική υλοποίηση του ασαφούς ελεγκτή. Σε αυτές φαίνονται:

- οι δομικές μονάδες *(blocks)* από τις οποίες αποτελείται (με τετράγωνα)
- τα διάφορα σήματα (με βέλη), καθώς και τα ονόματα και τα πλάτη αυτών
- τα *pipeline stages* (με στικτές γραμμές)



Σχ9: *Μοντέλο Ασαφούς Ελεγκτή με Αριθμητική Συνάρτηση Συμμετοχής*



Σχ10: *Μοντέλο Ασαφούς Ελεγκτή με Συνάρτηση Συμμετοχής βασισμένη σε ROM*

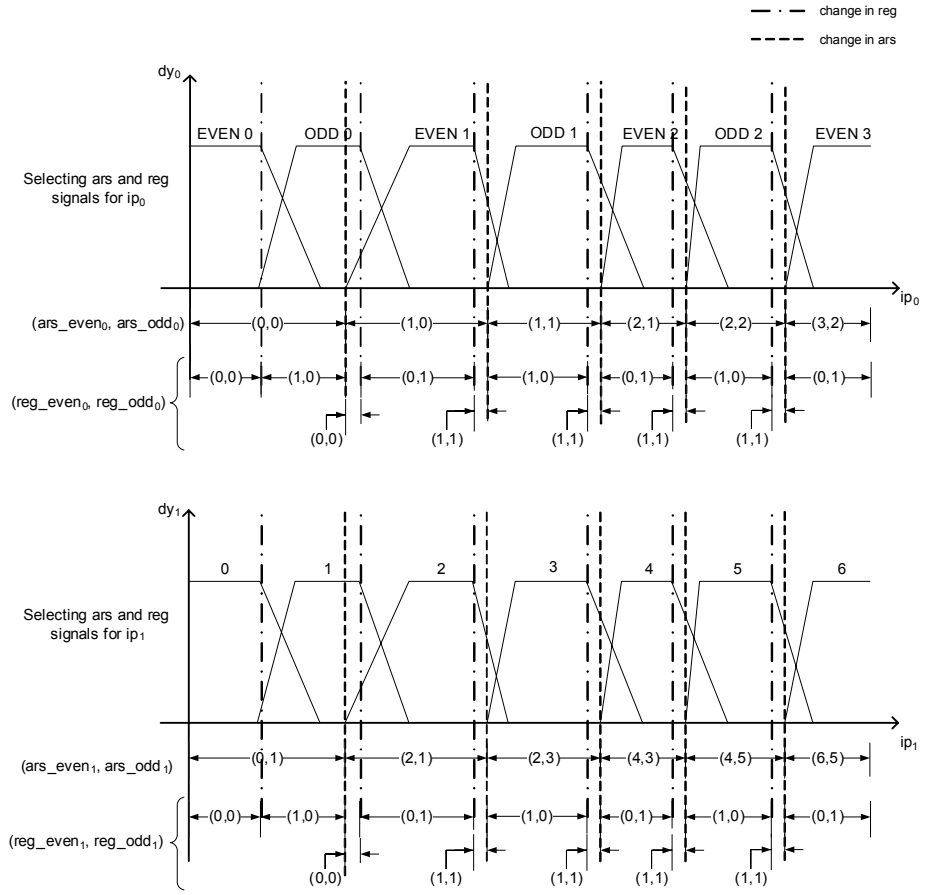
Στη συνέχεια, αναλύονται εκτενώς τα διάφορα *δομικά στοιχεία (blocks)* της $1^{ης}$ αρχιτεκτονικής του ***Σχ9***, ενώ στο τέλος εξετάζονται μόνο τα σημεία της $2^{ης}$ αρχιτεκτονικής στα οποία αυτή διαφοροποιείται της $1^{ης}$.

2.2.1 Επιλογέας Ενεργών Κανόνων *(Active Rule Selector)*

Για κάθε είσοδο πρέπει να εξετάσουμε όλους τους κανόνες (*49* τον αριθμό) ώστε να σχηματίσουμε την έξοδο. Η μεθοδολογία αύτη φυσικά δεν είναι αποδοτική. Ωστόσο, αν ξέραμε ποιοι από τους κανόνες είναι ενεργοί τότε θα εξετάζαμε μόνο αυτούς, μειώνοντας έτσι τους κύκλους υπολογισμού. Ένας κανόνας είναι ενεργός όταν έχει μη μηδενική δύναμη *(firing strength)*. Εμείς έχουμε υποθέσει ότι τα ασαφή σύνολα στα οποία χωρίζουμε τον χώρο των εισόδων έχουν επικάλυψη δύο. Αυτή η υπόθεση είναι μάλλον ρεαλιστική παρά περιοριστική. Για παράδειγμα, ένας άνθρωπος λέμε ότι είναι κοντός προς μεσαίο με κάποια αβεβαιότητα μεταξύ των δύο, ωστόσο αποκλείουμε την πιθανότητα να είναι ψηλός. Δηλαδή, τις περισσότερες φορές, το δίλημμα της κατηγοριοποίησης περιέχει δύο γειτονικές κατηγορίες. Με την υπόθεση αυτή, σε κάθε είσοδο αντιστοιχούν δύο το πολύ γειτονικά ασαφή σύνολα με μη μηδενικές συναρτήσεις συμμετοχής *(ενεργά ασαφή σύνολα)*. Και επειδή στην συγκεκριμένη υλοποίηση έχουμε δύο εισόδους, ο αριθμός των ενεργών κανόνων είναι $2^2=4$. Το πρόβλημα λοιπόν έγκειται στον εντοπισμό των ενεργών ασαφών συνόλων κάθε εισόδου. Η λειτουργία αύτή επιτελείται στον *Active Rule Selector*.

Για κάθε είσοδο $(ip_{0,1})$, για την εύρεση του ζευγαριού των ενεργών ασαφών συνόλων $(ars\_odd_{0,1}, ars\_even_{0,1})$, για κάθε *fuzzy set* συγκρίνουμε την είσοδο με την ελάχιστη τιμή εισόδου για την οποία έχει μη μηδενική συμμετοχή $(rise\_start_{0,1})$. Τα σήματα που προκύπτουν από τα αποτελέσματα των συγκρίσεων $(sel\_r_{0,1})$ καθορίζουν το ζεύγος των ενεργών ασαφών συνόλων με τον τρόπο που φαίνεται στο ***Σχ12***. Φυσικά, η σύγκριση δεν είναι απαραίτητη για το πρώτο και το δεύτερο *fuzzy set* μιας και είναι το αρχικά επιλεχθέν ζεύγος για τις μικρές τιμές εισόδου μέχρις ότου φτάσουμε στο *rise_start* του τρίτου. Έτσι χρειαζόμαστε *7-2=5* συγκρίσεις. Για να γίνεται κάθε φορά η μετάβαση όταν $ip_{0,1}=rise\_start_{0,1}$ και όχι όταν $ip_{0,1}>rise\_start_{0,1}$

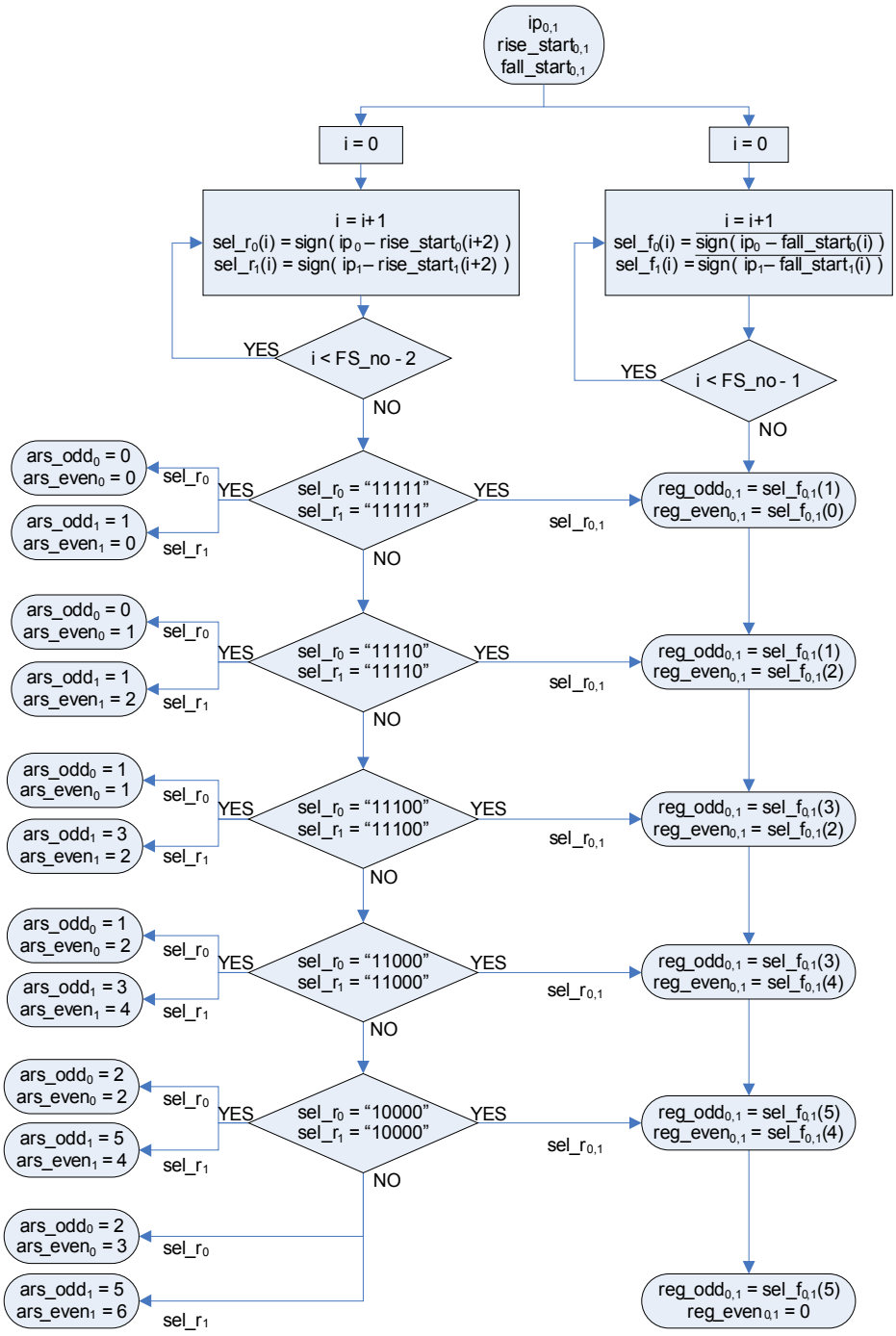
χρησιμοποιούμε για σήμα επιλογής το πρόσημο της αφαίρεσης $ip_{0,1}$-$rise\_start_{0,1}$,για κάθε *fuzzy set*.



Σχ11: *Ονοματολογία των ασαφών συνόλων και έξοδοι του Επιλογέα Ενεργών κανόνων*

Η κωδικοποίηση των *ασαφών συνόλων* (*fuzzy sets)* (***Σχ11***) παίζει βασικό ρόλο στη σχεδίαση των μετέπειτα δομικών μονάδων *(blocks)*. Πρέπει να είναι τέτοια ώστε να βελτιστοποιεί το συνολικό κύκλωμα από πλευράς χρόνου χωρίς να θυσιάζει πολύ χώρο (απλούστευση πράξεων, μικρό *width*, σπάσιμο μνημών). Υπό αυτό το σκεπτικό, η επιλεχθείσα κωδικοποίηση επιτρέπει το σπάσιμο της *μνήμης αποθήκευσης των παραμέτρων (parameter Memory Bank)* που αντιστοιχεί στην ip_0 και της *μνήμης μονοσυνόλου* (*singletons ROM)*, σε *ODD* και *EVEN* (γλιτώνοντας τους μισούς κύκλους ρολογιού και θυσιάζοντας ταυτόχρονα λίγο χώρο λόγω αύξησης της παραλληλίας), ενώ οδηγεί σε απλό *απεικονιστή αποτελέσματος (consequent*

mapper) και *γεννήτρια διευθύνσεων (Address Generator)* όπως φαίνεται παρακάτω.



Σχ12: *Διάγραμμα Ροής του Επιλογέα Ενεργών Κανόνων*

Αυξάνοντας λίγο ακόμα την παραλληλία του *Επιλογέα Ενεργών Κανόνων (Active Rule Selector)* ώστε να απλοποιήσουμε τη λειτουργία της

Γεννήτριας Τραπεζοειδών Συναρτήσεων (Trapezoidal Generator), τον επιφορτίζουμε για κάθε είσοδο, με την εύρεση, για κάθε ενεργό *fuzzy set* του ζεύγους, την περιοχή *{rising ή falling} = {0 ή 1}* στην οποία βρίσκεται η είσοδος $(reg\_odd_{0,1}, reg\_even_{0,1})$. Ο τρόπος φαίνεται στο ***Σχ12***. Για τη δημιουργία των σημάτων $sel\_f_{0,1}$ συγκρίνουμε την είσοδο με το $fall\_start_{0,1}$ που είναι το σημείο μετά το οποίο η συνάρτηση συμμετοχής παρουσιάζει αρνητική κλίση. Η σύγκριση δεν είναι αναγκαία για το τελευταίο *fuzzy set* κάθε εισόδου μιας και αυτό δεν έχει *falling* περιοχή. Άρα, έχουμε συνολικά *7-1=6* συγκρίσεις.

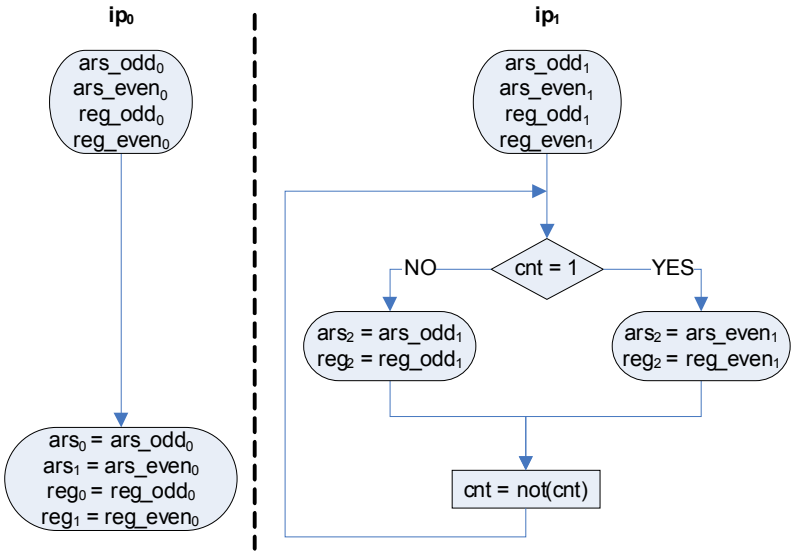
Αξίζει τέλος να σημειωθεί ότι οι αφαιρέσεις που χρησιμοποιούνται για τις συγκρίσεις είναι ουσιαστικά πιο γρήγορες από τις κοινές αφαιρέσεις αφού ο αφαιρετέος είναι σταθερός αριθμός. Άρα, αντί για κοινούς αφαιρέτες έχουμε *special purpose* αφαιρέτες. Ωστόσο, δεν μπορούμε να μεταβάλουμε τα *rising* και τα *falling parts* των *fuzzy sets*. Αν ο προσαρμοστικός αλγόριθμος επιβάλει το αντίθετο τότε πρέπει να χρησιμοποιήσουμε αναγκαστικά κοινούς αφαιρέτες.

Από τα παραπάνω γίνεται κατανοητό ότι ο *Active Rule Selector* αποτελεί το κλειδί για την υλοποίηση ενός γρήγορου *Fuzzy Controller*.

2.2.2 Γεννήτρια Διευθύνσεων *(Address Generator)*

Η μονάδα αυτή επιλέγει ανάμεσα στις εξόδους του *Active Rule Selector*, 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 *fuzzy sets* κάθε εισόδου, για όλες τις εισόδους. Ο αριθμός των δυνατών συνδυασμών είναι λοιπόν $2^2=4$ και αντιστοιχεί στον αριθμό των ενεργών κανόνων. Μπορούμε ωστόσο να μειώσουμε τους κύκλους ρολογιού στο μισό αφού, όπως θα φανεί παρακάτω, μπορούμε να εξετάσουμε παράλληλα τους συνδυασμούς που προκύπτουν αν επιλέξουμε το *ODD fuzzy set* για την πρώτη είσοδο με αυτούς που προκύπτουν αν επιλέξουμε το *EVEN fuzzy set* για την πρώτη είσοδο. 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) 1bit*, τον οποίο χρησιμοποιούμε και σαν σήμα μηδενισμού των ολοκληρωτών, το περιεχόμενο του οποίου αντιστρέφουμε σε κάθε *rising edge* του ρολογιού. Το διάγραμμα ροής φαίνεται στο ***Σχ13***:



Σχ13: *Διάγραμμα Ροής της 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 επεξεργασία ενός καινούριου ζεύγους εισόδων κάθε *2* κύκλους ρολογιού. Φυσικά, η μέθοδος σχεδίασης αυτή, την οποία ονομάσαμε *"ODD-EVEN" method*, βρίσκει εφαρμογή και για περισσότερες εισόδους του ασαφούς ελεγκτή. Στη γενική περίπτωση, λοιπόν, η συχνότητα επεξεργασίας της εισόδου είναι $1/2^{n-1}$, όπου n ο αριθμός των εισόδων του ασαφούς ελεγκτή.

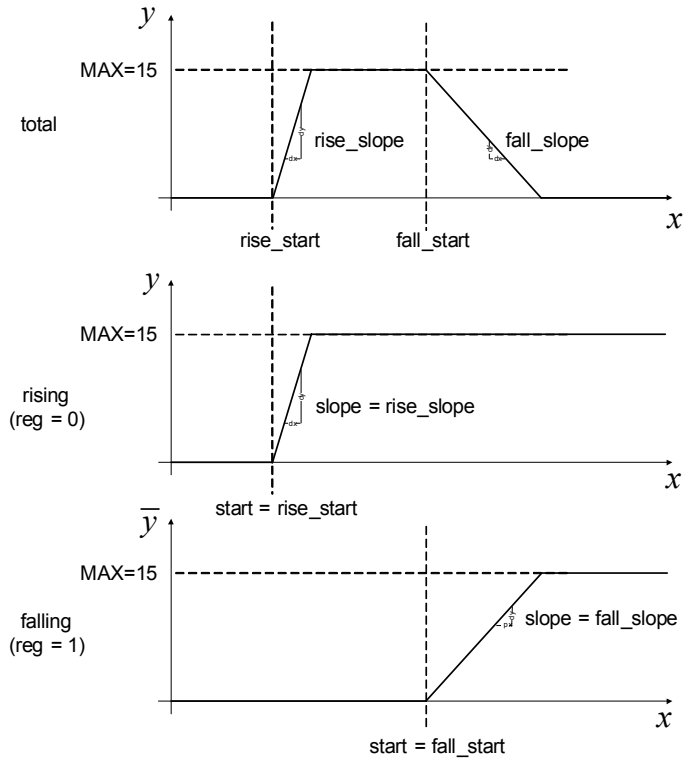
2.2.3 Γεννήτρια Τραπεζοειδών Συναρτήσεων Συμμετοχής *(Trapezoidal Generator)*

Η μονάδα αυτή κάνει την κυκλωματική *(hardware)* υλοποίηση της *Trapezoidal* συνάρτησης συμμετοχής. Για να μπορέσουμε να προτείνουμε ένα αλγόριθμο υπολογισμού πρέπει πρώτα να προτείνουμε ένα τρόπο κωδικοποίησης ενός τραπεζίου. Η κωδικοποίηση που ακολουθήθηκε φαίνεται στο ***Σχ14***.

Έτσι, για κάθε fuzzy set A_{ij} πρέπει να ξέρουμε τα εξής:

- rise_start (8bit)
- rise_slope (4bit)
- fall_start (8bit)
- fall_slope (4bit)

Τα δεδομένα είναι αποθηκευμένα στις *μνήμες παραμέτρων* (*parameter Memory Banks)* με τον τρόπο που φαίνεται στον ***Πίνακα2***, σύμφωνα με την ονοματολογία των *fuzzy sets* του ***Σχ11***.



Σχ14: *Κωδικοποίηση Τραπεζίου και χωρισμός αυτού σε περιοχές*

Πίνακας2: *Τα περιεχόμενα των μνημών με τις παραμέτρους των Ασαφών Συνόλων*

ip_0 ODD		**ip_0 EVEN**		**ip_1**	
$rise\_start_{ODD0}$	$rise\_slope_{ODD0}$	$rise\_start_{EVEN0}$	$rise\_slope_{EVEN0}$	$rise\_start_0$	$rise\_slope_0$
$fall\_start_{ODD0}$	$fall\_slope_{ODD0}$	$fall\_start_{EVEN0}$	$fall\_slope_{EVEN0}$	$fall\_start_0$	$fall\_slope_0$
$rise\_start_{ODD1}$	$rise\_slope_{ODD1}$	$rise\_start_{EVEN1}$	$rise\_slope_{EVEN1}$	$rise\_start_1$	$rise\_slope_1$
$fall\_start_{ODD1}$	$fall\_slope_{ODD1}$	$fall\_start_{EVEN1}$	$fall\_slope_{EVEN1}$	$fall\_start_1$	$fall\_slope_1$
$rise\_start_{ODD2}$	$rise\_slope_{ODD2}$	$rise\_start_{EVEN2}$	$rise\_slope_{EVEN2}$	$rise\_start_2$	$rise\_slope_2$
$fall\_start_{ODD2}$	$fall\_slope_{ODD2}$	$fall\_start_{EVEN2}$	$fall\_slope_{EVEN2}$	$fall\_start_2$	$fall\_slope_2$
		$rise\_start_{EVEN3}$	$rise\_slope_{EVEN3}$	$rise\_start_3$	$rise\_slope_3$
				$fall\_start_3$	$fall\_slope_3$
				$rise\_start_4$	$rise\_slope_4$
				$fall\_start_4$	$fall\_slope_4$
				$rise\_start_5$	$rise\_slope_5$
				$fall\_start_5$	$fall\_slope_5$
				$rise\_start_6$	$rise\_slope_6$

Από το ***Σχ14*** παρατηρούμε ότι η επιλεχθείσα κωδικοποίηση μας επιτρέπει μια ενιαία αντιμετώπιση του *rising* και του *falling* κομματιού του τραπεζίου, με μόνη διαφορά τη λογική αντιστροφή *(not)* του αποτελέσματος *(y)* στην περίπτωση του *falling*. Το γεγονός αυτό είναι πολύ ικανοποιητικό αφού μειώνει κατά πολύ την πολυπλοκότητα του κυκλώματος.

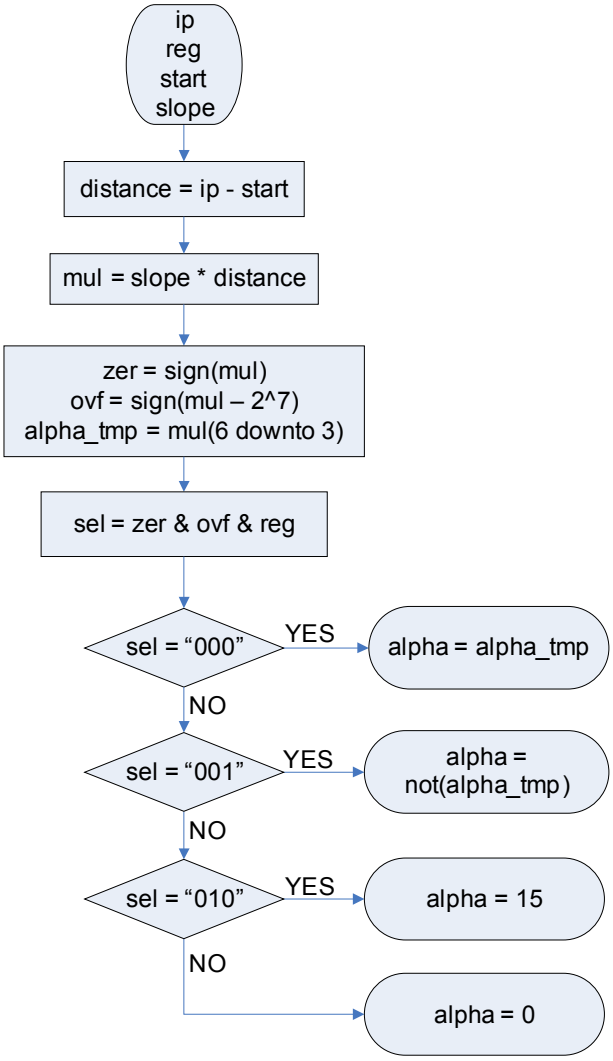
Με δεδομένη την κωδικοποίηση μπορούμε πλέον να εξετάσουμε τον αλγόριθμο υπολογισμού του οποίου το διάγραμμα ροής εικονίζεται στο ***Σχ15***.

Τα σήματα εισόδου προκύπτουν ως εξής:

- **ip:** η αντίστοιχη είσοδος *(ip_0, ip_0, ip_1)*
- **reg:** προέρχεται από την έξοδο του *Adress Generator (reg_0, reg_1, reg_2)*
- **start:** προέρχεται από την έξοδο της αντίστοιχης μνήμης *parameter Memory Bank* με είσοδο τα αντίστοιχα σήματα *ars&reg (ars_0&reg_0, ars_1&reg_1, ars_2&reg_2)*
- **slope:** προέρχεται από την έξοδο της αντίστοιχης μνήμης *parameter Memory Bank* με όμοιο τρόπο

Πρώτα υπολογίζουμε την απόσταση *(distance)* και έπειτα πολλαπλασιάζουμε με την κλήση *(mul)*. Έπειτα κάνουμε *truncate* κατά *3bit (alpha_tmp)*. Το πόσα *bit* θα κόψουμε στο *truncation* έχει να κάνει με το τι ακρίβεια θέλουμε να έχει η *slope*. Εδώ επιλέχθηκαν τα *3bit* που επιτρέπουν ομοιόμορφη ακρίβεια τόσο στις μεγάλες όσο και στις μικρές κλίσεις. Τέλος, κάνουμε διόρθωση αποτελέσματος *(alpha)* με τη βοήθεια $3^{ων}$ *flags*:

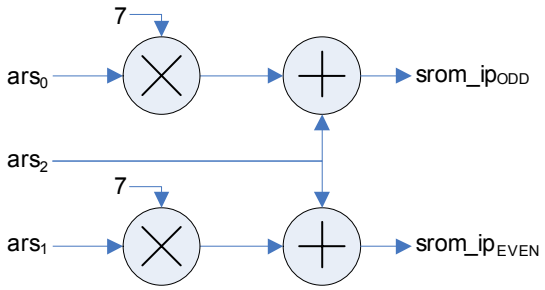
- **reg:** υποδηλώνει σε ποια περιοχή *(rising,falling)* του *trapezoidal* βρισκόμαστε
- **ovf:** ενεργοποιείται σε περίπτωση που έχουμε υπερχείλιση *(overfloat)*
- **zer:** ενεργοποιείται σε περίπτωση που η απόσταση *(distance)* είναι αρνητική



Σχ15: *Διάγραμμα Ροής της Γεννήτριας Τραπεζοειδών Συναρτήσεων*

2.2.4 Απεικονιστής Συμπεράσματος *(Consequent Mapper)*

Χρειάζεται ώστε από τα *ars* σήματα του *Address Generator* να προκύψουν οι διευθύνσεις $srom\_ip_{ODD}$ και $srom\_ip_{EVEN}$ για την *ODD* και *EVEN Singletons ROM* αντίστοιχα. Το διάγραμμα ροής φαίνεται στο ***Σχ16***. Τα σήματα ars_0 και ars_1 πολλαπλασιάζονται επί τον αριθμό των fuzzy *sets* της ip_1 που στην περίπτωσή μας είναι *7*. Στα αποτελέσματα προστίθεται το σήμα ars_2.



Σχ16: *Διάγραμμα Ροής του Απεικονιστή Συμπεράσματος*

2.2.5 Μνήμη Μονοσυνόλων *(Singletons ROM)*

Στη μνήμη αυτή, η οποία είναι σπασμένη 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 *fuzzy set* της ip_0 που εμπλέκεται στον κανόνα, ενώ ο δείκτης *j* σε αυτό της ip_1. Και οι δύο δείκτες έχουν ως τιμές τα ονόματα των *fuzzy sets* που προτάθηκαν στο ***Σχ11*** για τις δύο εισόδους.

Πίνακας3: *Τα περιεχόμενα της Μνήμης Μονοσυνόλων*

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

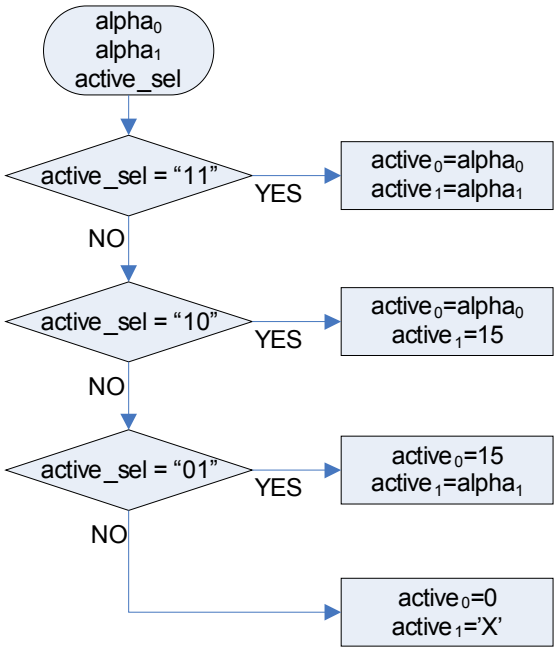
2.2.6 Επιλογέας Κανόνα *(Rule Selector)*

Η λειτουργία του *Rule Selector* είναι να αξιοποιεί την πληροφορία για το ποιες υποθέσεις του εξεταζόμενου κανόνα είναι ενεργές και να διαμορφώνει κατάλληλα τις αριθμητικές τιμές των γλωσσικών μεταβλητών. Μιας και οι εξεταζόμενοι κανόνες σε κάθε κύκλο είναι δύο, λόγω της δομής *ODD-EVEN* που εφαρμόσαμε παραπάνω, χρειαζόμαστε και δύο *Rule Selectors (ODD-EVEN)*, η λειτουργία των οποίων είναι πανομοιότυπη.

- Ο *ODD* δέχεται για είσοδο τα σήματα *alpha$_0$odd* και *alpha$_1$* των αντιστοίχων *Trapezoidal Generators* καθώς και το σήμα επιλογής *active_sel* της *ODD consequents ROM (sel odd)*:
  - alpha$_0$ = alpha$_0$odd
  - alpha$_1$ = alpha$_1$
  - active_sel = sel odd

- Ο *EVEN Rule Selector* δέχεται για είσοδο τα σήματα *alpha$_0$even* και *alpha$_1$* των αντιστοίχων *Trapezoidal Generators* και το σήμα επιλογής *active_sel* της *EVEN consequents ROM*:
  - alpha$_0$ = alpha$_0$even
  - alpha$_1$ = alpha$_1$
  - active_sel = sel even

Παρακάτω αναλύεται η λειτουργία του *Rule Selector* της οποίας το διάγραμμα ροής φαίνεται στο ***Σχ17***.

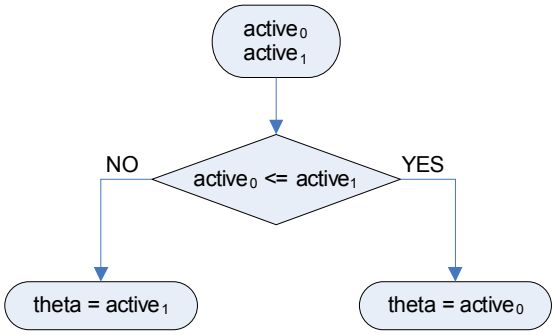


Σχ17: *Διάγραμμα Ροής του Επιλογέα Κανόνα*

Σκοπός μας είναι όταν μια υπόθεση δεν είναι ενεργή να μην συμμετέχει στη διαμόρφωση της δύναμης του κανόνα. Αφού λοιπόν η δύναμη του κανόνα ισούται με το μικρότερο από τα *alpha*, για να αγνοήσουμε ένα alpha αρκεί να του δώσουμε τη μέγιστη δυνατή τιμή, η οποία αφού έχουμε *unsigned* αριθμό *4bit* είναι $2^4-1 = 15$. Η πρακτική αυτή λειτουργεί όταν έχουμε μια τουλάχιστον ενεργή υπόθεση. Αν όλες οι υποθέσεις του κανόνα είναι ανενεργές, τότε θα πρέπει να θέσουμε τουλάχιστον ένα *alpha* ίσο με την ελάχιστη τιμή, με αποτέλεσμα η δύναμη του κανόνα να προκύψει μηδενική.

2.2.7 Μονάδα προσδιορισμού του Ελαχίστου *(MIN unit)*

Η μονάδα αυτή εξάγει τον ελάχιστο *(minimum)* μεταξύ δύο *unsigned* αριθμών, ίδιας ακρίβειας, τους οποίους δέχεται σαν είσοδο. Ουσιαστικά, αποτελεί τη *hardware* υλοποίηση της *Gödel t-norm*. Η έξοδος ισούται με τη δύναμη του κανόνα *(theta value)*. 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 *Rule Selector*. Το διάγραμμα ροής του *MIN block* φαίνεται στο ***Σχ18***:



Σχ18: *Διάγραμμα Ροής της μονάδας προσδιορισμού του Ελαχίστου*

2.2.8 Πολλαπλασιαστής Περιττού/Άρτιου *(Multiplier Odd/Even)*

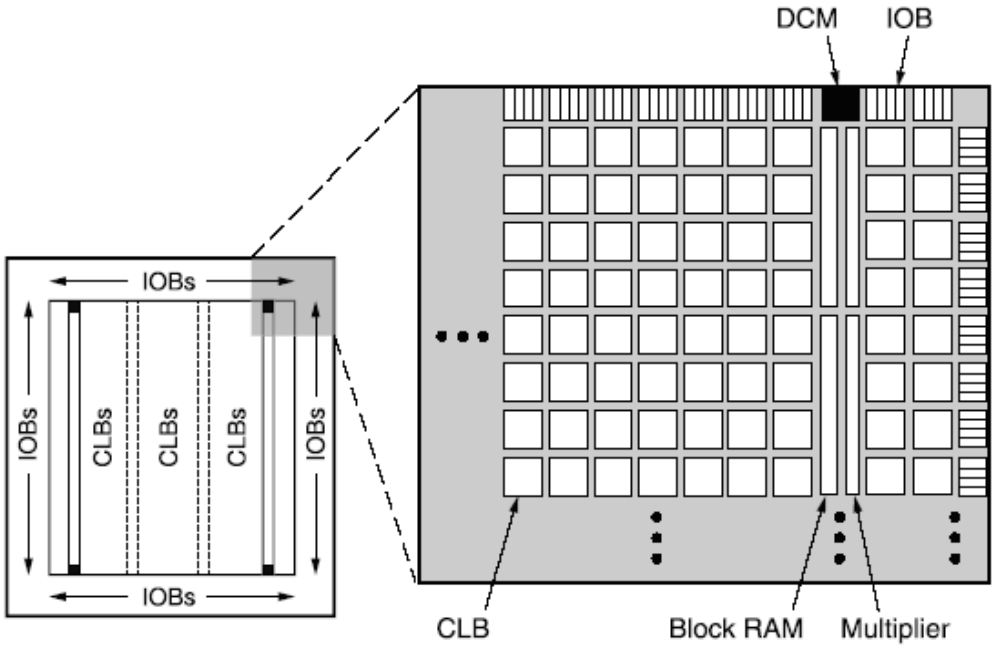
Υπολογίζει το *συνεπαγόμενο ασαφές σύνολο* του ενεργού κανόνα εκείνου που εμπλέκει το *ODD-EVEN* αντίστοιχα ασαφές σύνολο της ip_0. Αφού χρησιμοποιούμε *singleton* για το συμπέρασμα του κανόνα, το αποτέλεσμα της συνεπαγωγής θα είναι επίσης *singleton*. Για τον υπολογισμό χρησιμοποιούμε την *Product t-norm*.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

Η οικογένεια των *FPGA devices*, *Spartan 3* [28-29], 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 1500 -4 FPGA*, το οποίο επιλέχθηκε, περιέχει *32 embedded multipliers*. Αυτοί πολλαπλασιάζουν δύο αριθμούς, καθένας εκ των οποίων είναι *18-bit signed* ή *17-bit unsigned*. Ο πολλαπλασιασμός γίνεται σε αριθμητική συμπληρώματος του *2 (tow's complement)*, με εφαρμογή του *τροποποιημένου αλγορίθμου του Booth* [32], χρησιμοποιώντας πολυπλέκτες *(multiplexers)* για την δημιουργία των μερικών γινομένων *(partial products)*. Η θέση των πολλαπλασιαστών στο chip φαίνεται στο ***Σχ19***. Όπως φαίνεται στο σχήμα, υπάρχουν *CLBs (Controllable Logic Blocks)* και *RAMs* σε κοντινή απόσταση από τους πολλαπλασιαστές ώστε να μειωθεί το *critical path* των συνδέσεων των εισόδων και των εξόδων αυτών με τη λογική που παρεμβάλλεται.

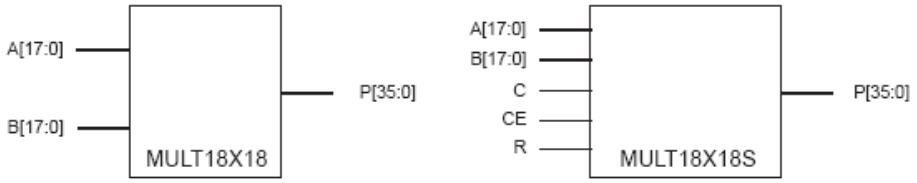
Υπάρχουν δύο τύποι πολλαπλασιαστών, ο *ασύγχρονος (MULT18X18)* και ο *σύγχρονος (MULT18X18S)*, όπώς δείχνει το ***Σχ20***, οι οποίοι ουσιαστικά αναφέρονται στον ίδιο πολλαπλασιστή. Η μόνη διαφορά μεταξύ των δύο είναι ότι στον *σύγχρονο* χρησιμοποιείται ο καταχωρητής εξόδου του πολλαπλασιαστή. Αυτός μας δίνει τη δυνατότητα προσθήκης ενός *pipe stage*, με αποτέλεσμα να μειωθεί το *critical path* 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 του *delay* κατά μία περίοδο ρολογιού. Αυτό υποδηλώνεται στα σχήματα ***Σχ9,10*** των

αρχιτεκτονικών, με την τοποθέτηση ενός *pipe stage* στο μέσο του κάθε πολλαπλασιαστή.

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



Σχ19: *Θέση των Πολλαπλασιαστών στην οικογένεια Spartan 3*



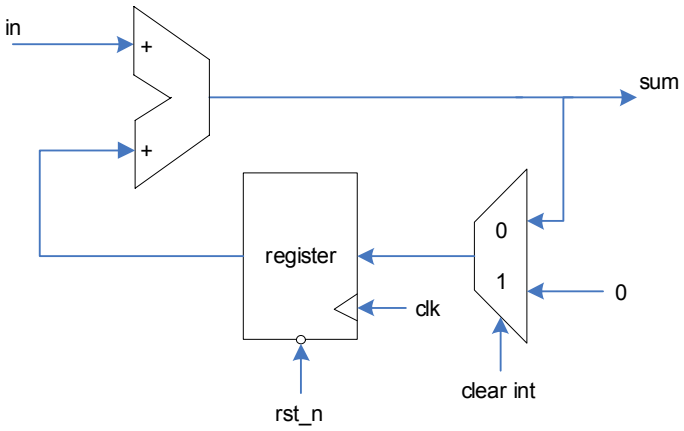
Σχ20: *Ασύγχρονος & Σύγχρονος Ενσωματωμένος Πολλαπλασιαστής στα Spartan 3 FPGA chips της Xilinx*

2.2.9 Αθροιστής Προσημασμένος/Μη προσημασμένος *(Adder Signed/Unsigned)*

Είναι ένας *unsigned/signed* αντίστοιχα *adder* ο οποίος αποτελεί μέρος της διαδικασίας απο-ασαφοποίησης.

2.2.10 Ολοκληρωτής Προσημασμένος/Μη προσημασμένος *(Integrator Signed/Unsigned)*

Είναι ένας *unsigned/signed* αντίστοιχα ολοκληρωτής ο οποίος αποτελεί μέρος της διαδικασίας απο-ασαφοποίησης. Όπως έχει αναφερθεί παραπάνω, χρειάζονται $2^2/2=2$ κύκλοι 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 *Address Generator* ***Σχ13***, 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 τον μηδενισμό του πρώτου, αφού βρίσκεται στο επόμενο *pipe stage* όπως φαίνεται στα ***Σχ9,10***.



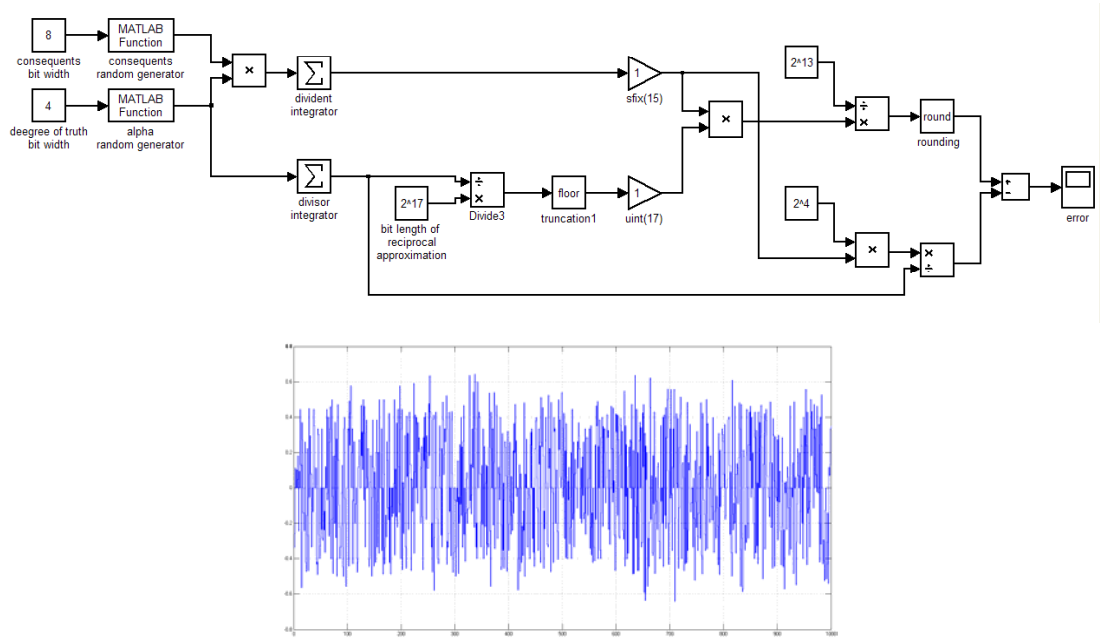
Σχ21: *Κυκλωματικό διάγραμμα Ολοκληρωτή*

2.2.11 Ανάστροφος Πίνακας Επισκόπησης *(reciprocal LUT)*

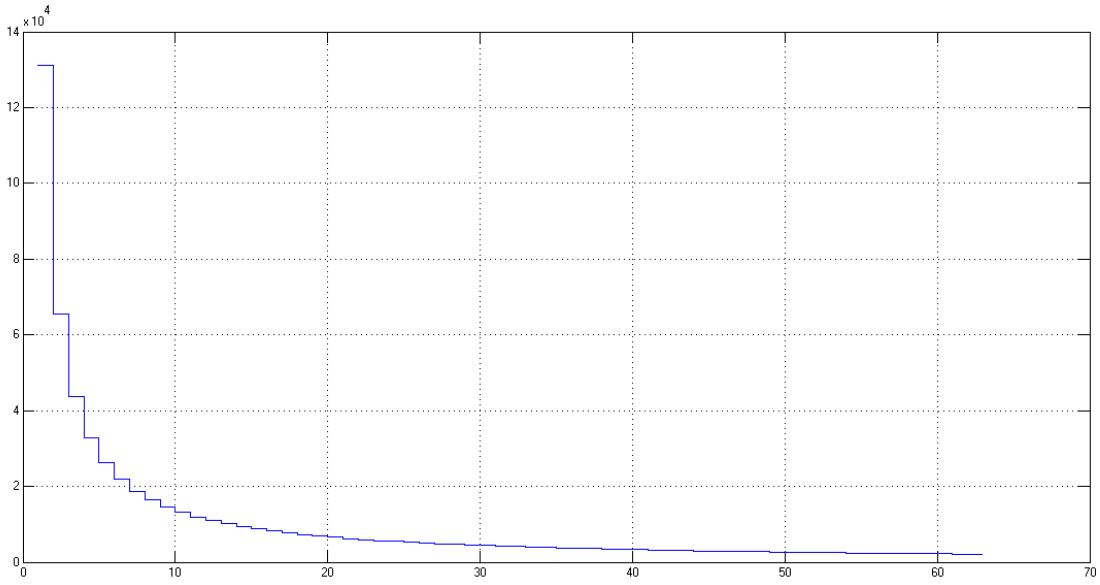
Είναι μια μνήμη που αποτελεί πίνακα τιμών *(look up table)* της συνάρτησης *1/x* και χρησιμοποιείται στη διαδικασία απο-ασαφοποίησης. Η ελάχιστη επιθυμητή ακρίβεια υπολογισμού του *1/x* επιλέχθηκε πειραματικά ίση με *17bit*, με τη βοήθεια του *MATLAB-SIMULINK* ώστε να μην έχουμε σφάλμα στον υπολογισμό της *12bit* εξόδου του κυκλώματος. Το κύκλωμα που δημιουργήθηκε για τον πειραματικό έλεγχο φαίνεται στο ***Σχ22*** μαζί με το μετρούμενο σφάλμα στην έξοδο.

Όπως φαίνεται και από το ***Σχ22***, υπολογίστηκε το σφάλμα της διαίρεσης του τύπου απο-ασαφοποίησης για όλες τις δυνατές τιμές των παραμέτρων του. Το σφάλμα που προέκυψε χρησιμοποιώντας ακρίβεια *17bit* στον υπολογισμό του αντιστρόφου του διαιρέτη είναι μικρότερο της μονάδας. Συνεπώς, δεν υπεισέρχονται σφάλματα στα *bit* της εξόδου.

Τα περιεχόμενα της μνήμης φαίνονται στο ***Σχ23***. Η διαίρεση με μηδέν δεν έχει πρακτική αξία μιας και εμφανίζεται μόνο στην περίπτωση που οι ενεργοί κανόνες έχουν μηδενική δύναμη. Ωστόσο, μπορεί να συμβεί αν έχουμε απενεργοποιήσει όλα τα ορίσματα αυτών στις *singletons ROMs*. Στην περίπτωση αυτή επιλέγουμε ως έξοδο τη μηδενική ($1/0 \equiv 0$) με την έννοια ότι για τις δεδομένες εισόδους δεν υπάρχει ανάγκη ελέγχου.



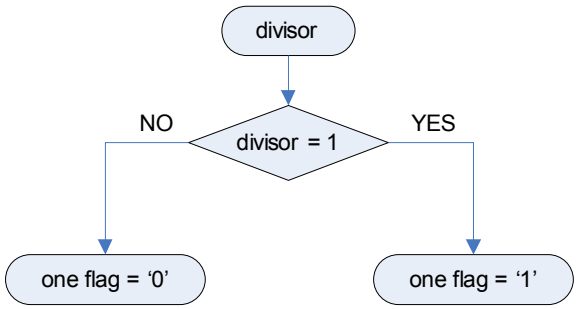
Σχ22: *Πειραματικός έλεγχος της ελάχιστης απαιτούμενης ακρίβειας του 1/x*



Σχ23: *Περιεχόμενα του ανάστροφου πίνακα επισκόπησης*

2.2.12 Έλεγχος του μοναδιαίου παρονομαστή *(Is One)*

Η μονάδα αυτή 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 *18bit* ακρίβειας στο *reciprocal LUT* αυξάνοντας, κατά πολύ το μέγεθος της απαιτούμενης μνήμης ενώ παράλληλα θα απαιτούσε και μεγαλύτερο πολλαπλασιαστή για τον υπολογισμό της εξόδου. Το διάγραμμα ροής του *block* φαίνεται στο ***Σχ24***:



Σχ24: *Διάγραμμα Ροής του ελεγκτή μοναδιαίου παρονομαστή*

2.2.13 Πολλαπλασιαστής *(Multiplier)*

Είναι ένας πολλαπλασιαστής που εκτελεί τον πολλαπλασιασμό μεταξύ του διαιρετέου του τύπου απο-ασαφοποίησης *(6-bit unsigned)* και της προσέγγισης του αντιστρόφου του διαιρέτη που προκύπτει από την έξοδο του *reciprocal LUT (17-bit unsigned)*. Όπως και στην περίπτωση των *ODD-EVEN Multipliers (ενότητα 2.2.8)*, έτσι και για την υλοποίηση αυτού του *Multiplier* χρησιμοποιείται ένας *embedded σύγχρονος multiplier MULT18X18S [29]*.

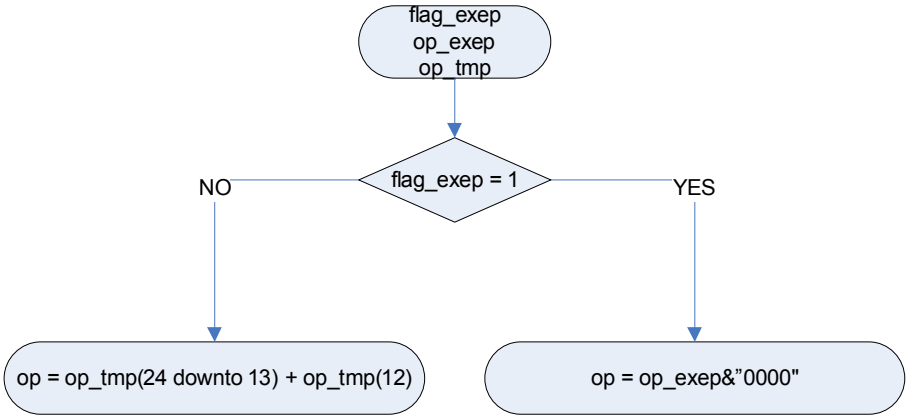
2.2.14 Παραγωγή τελικής εξόδου *(Fix Output)*

Στη μονάδα αυτή γίνεται διόρθωση του αποτελέσματος του πολλαπλασιασμού και παράγεται η τελική έξοδος. Το διάγραμμα ροής φαίνεται στο ***Σχ25***.

Δέχεται για είσοδο τα εξής σήματα:

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

Στην περίπτωση που ο διαιρέτης είναι ίσος με τη μονάδα, τότε περνάμε κατευθείαν στην έξοδο την τιμή του διαιρετέου *(8bit)* και προσθέτουμε και τα *4bit* 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*.



Σχ25: *Διάγραμμα Ροής της παραγωγής τελικής εξόδου*

2.2.15 Εναλλακτική Υλοποίηση με Συνάρτηση Συμμετοχής βασισμένη σε ROM

Μετά από την ολοκλήρωση της περιγραφής των *blocks* της $1^{ης}$ αρχιτεκτονικής του ***Σχ9*** εξετάζουμε τα σημεία στα οποία διαφοροποιείται η $2^{η}$ αρχιτεκτονική του ***Σχ10***. Αυτά είναι:

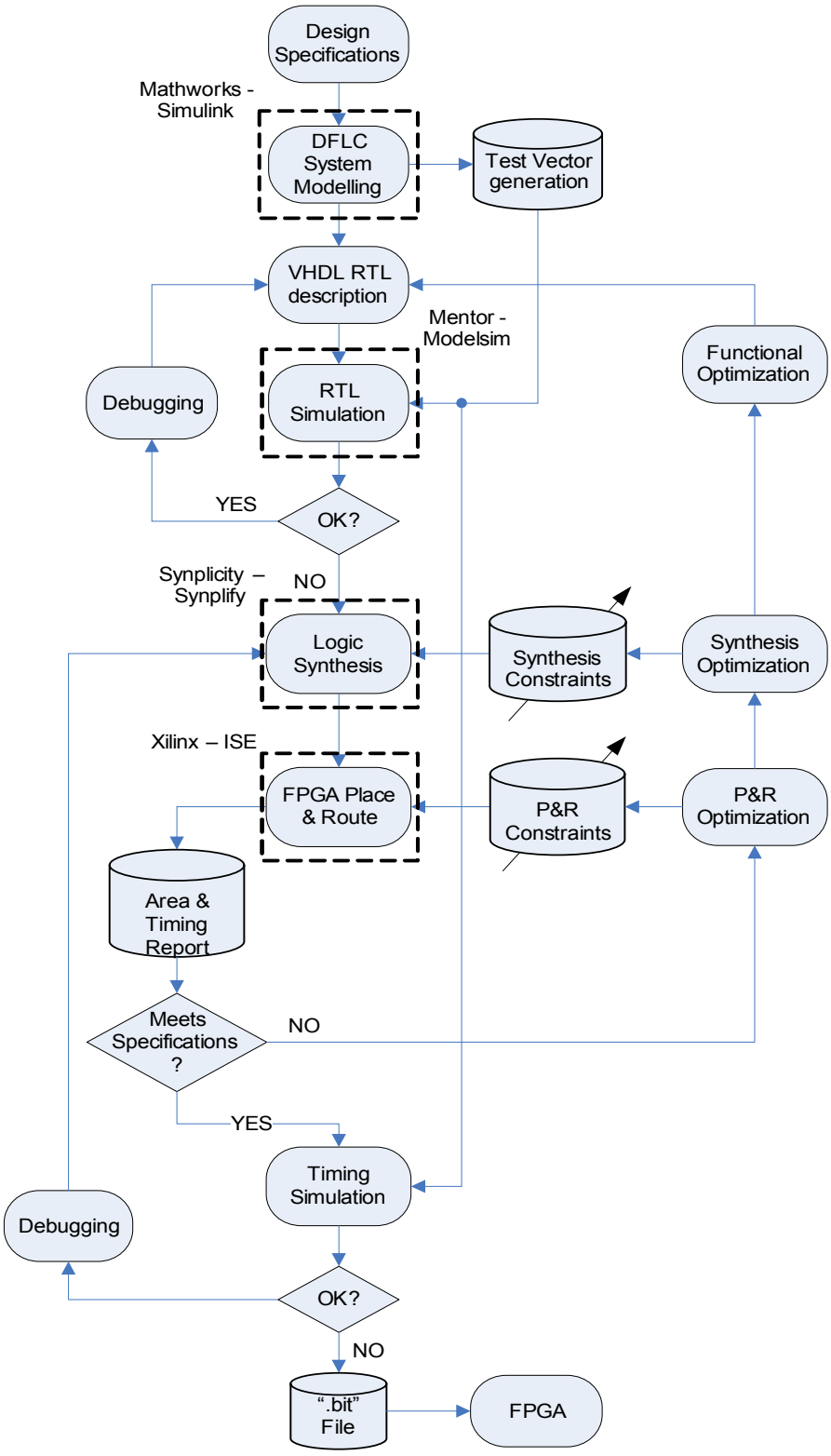
- Αντί του *Trapezoidal Generator* υπάρχει η *μνήμη συναρτήσεων συμμετοχής (Membership Function ROM).* Αυτή παίρνει ως διευθύνσεις τις εισόδους ip_0 και ip_1 και βγάζει στην έξοδο τις τιμές των συναρτήσεων συμμετοχής των ενεργών *fuzzy sets* κάθε εισόδου στο σημείο που υποδεικνύει η αντίστοιχη είσοδος. Τα περιεχόμενά της φαίνονται στον ***Πίνακα4***.
- Δεν χρειάζεται η έξοδος *region bus* του *Active Rule Selector*.
- Αντικαθιστούμε την είσοδο *region bus* του *Address Generator* με την *alpha bus*. Στην έξοδο, συνεπώς, θα έχουμε το σήμα *alpha gen bus* αντί του *region gen bus*. Δεν αλλάζει η λειτουργία του *block*.

Πίνακας4: *Τα περιεχόμενα της μνήμης συναρτήσεων συμμετοχής*

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

2.3 Διαδικασία Σχεδίασης

Η διαδικασία σχεδίασης που ακολουθήθηκε φαίνεται στο ***Σχ26***:



Σχ26: *Διάγραμμα Ροής Σχεδίασης*

Παρακάτω αναλύονται τα διάφορα βήματα

2.3.1 Προδιαγραφές Σχεδίασης *(Design Specifications)*

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 ***Πίνακα1***.

2.3.2 Μοντελοποίηση συστήματος ψηφιακού ασαφούς ελεγκτή *(DFLC System Modeling)*

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 στο *MATLAB-Simulink*. Το αποτέλεσμα φαίνεται στα ***Σχ27-30***.

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

2.3.3 Περιγραφή σε γλώσσα VHDL σε RTL επίπεδο *(VHDL RTL description)*

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 *(behavioral, rtl, gate level)*. Εμείς χρησιμοποιήσαμε την *rtl (register transfer level)* περιγραφή για

τον ασαφή ελεγκτή ο οποίος γίνεται *synthesized*, ενώ για τα *testbenches* την *behavioral* περιγραφή (*wait statements* δεν είναι *synthesized*) .

Όπως φαίνεται και από τους ενδεικτικούς κώδικες που παρουσιάζονται στο *ΠΑΡΑΡΤΗΜΑ*, πραγματοποιήθηκε ο διαχωρισμός του συστήματος σε *blocks*, σύμφωνα με τις αρχιτεκτονικές που παρουσιάστηκαν στα σχήματα ***Σχ9*** και ***Σχ10***. Σε κάθε *block* 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 *generics*. Επίσης, με *generics* καθορίστηκαν και τα μεγέθη των διαφόρων σημάτων.

2.3.4 Εξομοίωση του RTL *(RTL Simulation)*

Μετά την ολοκλήρωση της περιγραφής του συστήματος σε *VHDL*, ακολούθησε η αποσφαλμάτωση *(debugging)* αυτού. Για τη διαπίστωση της ορθότητας των αποτελεσμάτων χρησιμοποιήθηκαν τα *test vectors* 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 προγράμματος *Modelsim*. Στα ***Σχ31-32*** παρουσιάζονται οι κυματομορφές που προέκυψαν από την *rtl* προσομοίωση για τις δύο αρχιτεκτονικές των σχημάτων ***Σχ9-10*** αντίστοιχα. Τα ονόματα και οι χρονισμοί των σημάτων στο data flow βρίσκονται σε πλήρη συμφωνία με αυτά των ***Σχ9-10***.

Όπως γίνεται φανερό από το ***Σχ31***, χρησιμοποιούμε δύο ρολόγια για το χρονισμό του κυκλώματος. Το γρήγορο-εσωτερικό *(clkx)* δουλεύει στα *200MHz (5ns)* και αποτελεί το ρολόι εσωτερικής λειτουργίας του ασαφούς ελεγκτή. Το αργό-εξωτερικό ρολόι *(clk)* δουλεύει στα *100MHz (10ns)* και αποτελεί το ρολόι των καταχωρητών εισόδου και εξόδου του ασαφούς ελεγκτή. Όπως αποδείχτηκε στην *ενότητα 2.2.2*, αρκούν δύο κύκλοι ρολογιού του *Address Generator* ώστε να παραχθούν όλοι οι δυνατοί συνδυασμοί των ενεργών κανόνων που υποδεικνύει ασύγχρονα ο *Active Rule Selector*. 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 τον *DCM (Digital Control Manager)* και βρίσκονται σε φάση μεταξύ τους.

Ένα νέο ζεύγος εισόδων αποθηκεύεται στον καταχωρητή εισόδου, στην αρνητική ακμή *(falling edge)* του εξωτερικού ρολογιού. Ο *Address Generator* παράγει τους συνδυασμούς των ενεργών κανόνων στο *$1^{ο}$-$2^{ο}$ pipe stage*. Κάθε ένας από τους τρεις *Trapezoidal Generators* χρειάζονται *4 pipe stages* για να υπολογίσει την τιμή της τραπεζοειδούς συνάρτησης συμμετοχής *(alpha)*. Ο υπολογισμός γίνεται παράλληλα με τη διευθυνσιοδότηση *($3^{ο}$-$4^{ο}$ pipe stage)* και την ανάγνωση *($5^{ο}$-$6^{ο}$ pipe stage)* των *Singletons ROMs*. Ένα κύκλο εσωτερικού ρολογιού αργότερα *($6^{ο}$-$7^{ο}$ pipe stage)*, αφότου έχει γίνει η επεξεργασία του *Rule Selector* και του *Min* για το εξεταζόμενο ζεύγος των ενεργών κανόνων, οι *theta values* αυτών (αποτέλεσμα του *Min operator* στις *alpha values*) μαζί με τα *consequents* (συμπεράσματα) και το σήμα για το μηδενισμό των ολοκληρωτών γίνονται διαθέσιμα στο *Inference & Defuzzification* μέρος του κυκλώματος. Η *implication* (συνεπαγωγή) κάθε κανόνα προκύπτει δύο κύκλους εσωτερικού ρολογιού αργότερα *($8^{ο}$-$9^{ο}$ pipe stage)*, μαζί με το άθροισμα των *theta values*. Στον επόμενο κύκλο *($9^{ο}$-$10^{ο}$ pipe stage)*, υπολογίζεται το άθροισμα των *implications*. Στη διάρκεια αυτή ο *Unsigned Integrator* εξάγει την τιμή του διαιρέτη *(divisor) ($10^{ο}$ pipe stage)*. Η τιμή του διαιρετέου *(dividend)* προκύπτει από τον *Signed Integrator* στον επόμενο κύκλο *($11^{ο}$ pipe stage)* μαζί με την εκτίμηση του *reciprocal* από τον *Reciprocal LUT*. Ο πολλαπλασιασμός του διαιρετέου με το *reciprocal* και ο έλεγχος *Is One* γίνονται στο *$12^{ο}$ pipe stage*, όπου πραγματοποιείται και η διόρθωση της εξόδου στο *FIX Output block*. Τέλος, η διορθωμένη έξοδος αποθηκεύεται στον καταχωρητή εξόδου *(output register)* στην θετική ακμή *(rising edge)* του εξωτερικού ρολογιού *($13^{ο}$ pipe stage)*.

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 pipe stages ή 65ns*. Συνεπώς, έχουμε δειγματοληψία εισόδου κάθε *10ns* και παραγωγή εξόδου

επίσης κάθε *10ns* με *65ns* καθυστέρηση *(delay)*. Τα παραπάνω ισχύουν για το *Μοντέλο Ασαφούς Ελεγκτή με Αριθμητική Συνάρτηση Συμμετοχής* (***Σχ9,31***).

Για το *Μοντέλο Ασαφούς Ελεγκτή με Συνάρτηση Συμμετοχής βασισμένη σε ROM* (***Σχ10,32***) αντίστοιχα, για την παραγωγή της εξόδου απαιτούνται *11 pipe stages ή 55ns.* Συνεπώς, έχουμε δειγματοληψία εισόδου κάθε *10ns* και παραγωγή εξόδου επίσης κάθε *10ns με 55ns* καθυστέρηση *(delay)*.

Αξίζει να σημειωθεί ότι χωρίς τη χρήση της *ODD-EVEN* αρχιτεκτονικής που προτείνεται και επιτρέπει την εξέταση δύο κανόνων σε κάθε κύκλο εσωτερικού ρολογιού, θα είχαμε *12* και *10 pipe stages* 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 *design* λόγω μείωσης της παραλληλίας, αφού θα χρησιμοποιούσαμε *2 Trapezoidal Generators* αντί για *3*, *1 Rule Selector* αντί για *2*, 1 *MIN operator* αντί για *2*, *1 Multiplier* για την παραγωγή του *implication* αντί για *2*. Ωστόσο, θα είχαμε δειγματοληψία εισόδου κάθε *20ns*. Αποφασίστηκε λοιπόν ότι ήταν προτιμότερο να θυσιαστεί λίγο μέγεθος ώστε να αυξηθεί η συχνότητα δειγματοληψίας του ασαφούς ελεγκτή. Αυτό σημαίνει ότι θα έχει τη δυνατότητα να “κλειδώνει” (ελέγχει) πιο γρήγορα συστήματα.

2.3.5 Σύνθεση σε λογικό επίπεδο – επίπεδο πυλών *(Logic Synthesis)*

Αφού διαπιστώθηκε η ορθότητα της *VHDL* περιγραφής, ακολούθησε η μετάβαση από το επίπεδο κώδικα στο επίπεδο λογικών πυλών (παραγωγή *netlist*). Για το σκοπό αυτό χρησιμοποιήθηκε το πρόγραμμα σύνθεσης *Synplify* της *Synplicity*. Εκεί, τέθηκαν επιπλέον περιορισμοί *(constrains)* για τη βελτίωση χαρακτηριστικών του κυκλώματος, όπως αύξηση της συχνότητας λειτουργίας. Τα αποτελέσματα φαίνονται στα ***Σχ33-39*** .

2.3.6 Θέση και Δρομολόγηση FPGA *(FPGA Place & Route)*

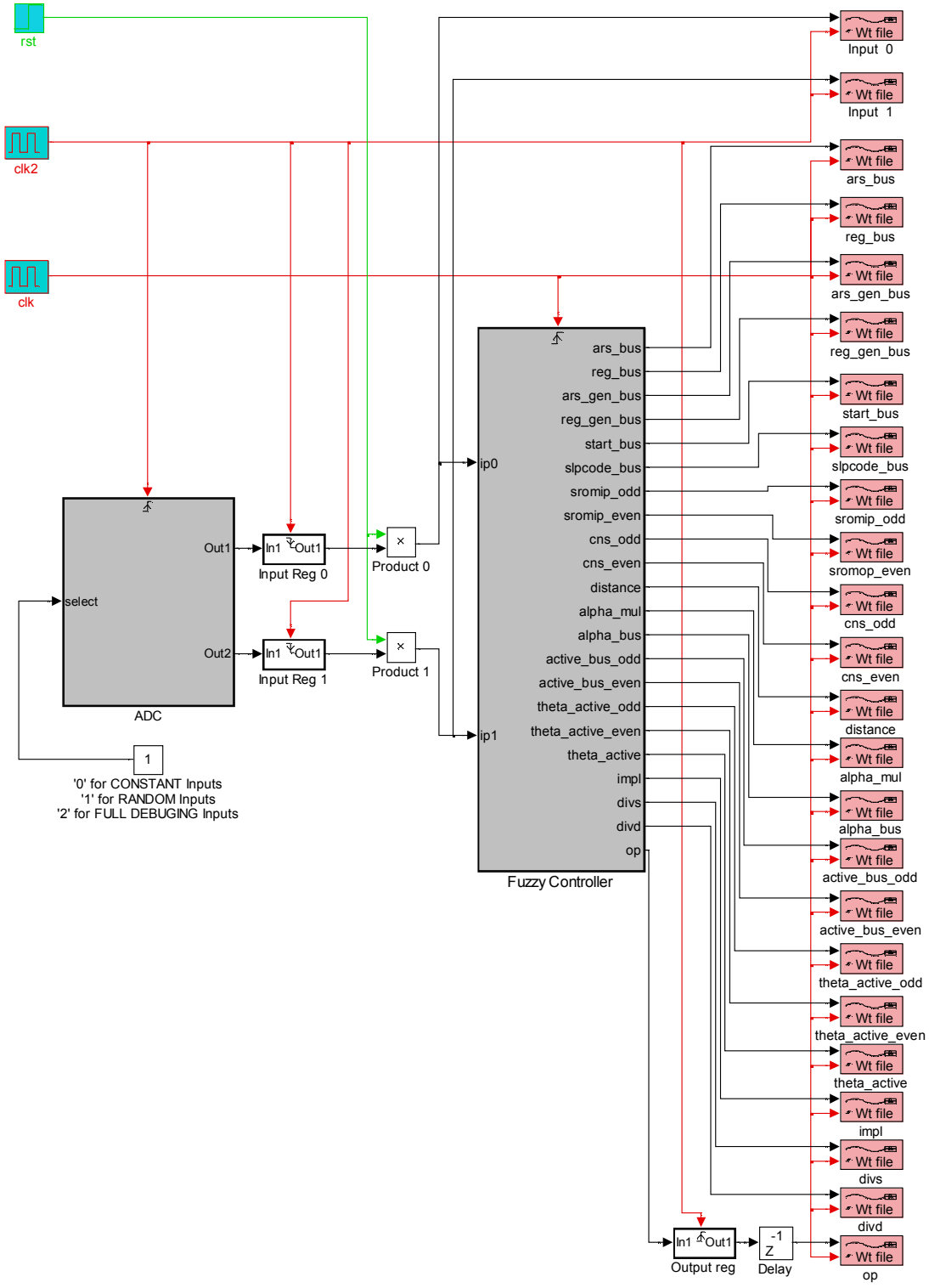
Το αρχείο περιγραφής του συστήματος σε *netlist* που προέκυψε από τον *synthesizer* έπρεπε να μετατραπεί σε κώδικα του κατασκευαστή του chip. Μιας και το chip στο οποίο προοριζόταν να γίνει *target* το *design* ήταν το *1500 Spartan-3 fg676 -4* της *Xilinx*, για το *place and root* χρησιμοποιήθηκε ο *project navigator της Xilinx*. Τα αποτελέσματα για τις δύο αρχιτεκτονικές φαίνονται στους ***Πίνακες5-6***.

2.3.7 Προσομοίωση του Χρονισμού *(Timing Simulation – Back annotation)*

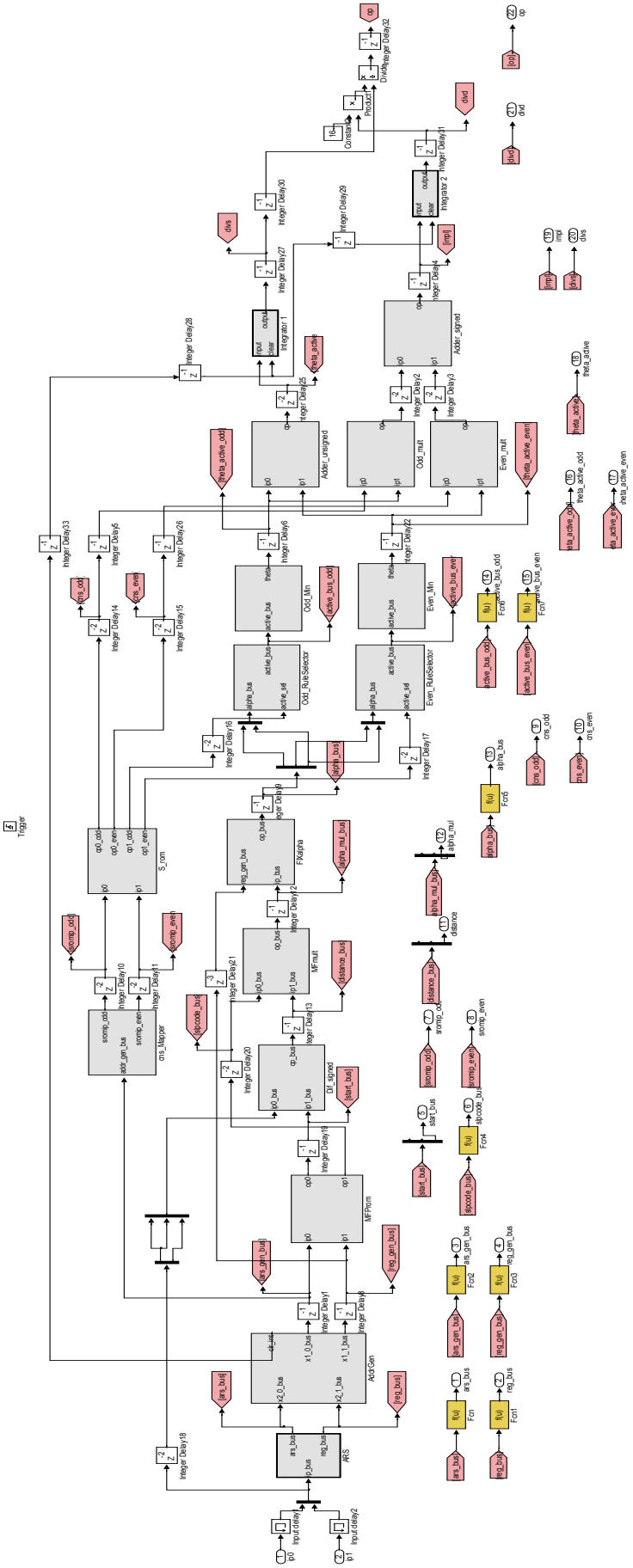
Τέλος, ελέγχθηκε η ορθότητα του κυκλώματος που προέκυψε από το *Place and Route* χρησιμοποιώντας πάλι σαν αναφορά τα *test vectors* που παρήχθησαν από το αντίστοιχο μοντέλο στο *Simulink*. Η ανάγκη για επανέλεγχο του τελικού κυκλώματος προκύπτει από το γεγονός ότι σε *rtl* επίπεδο δεν συνυπολογίζονται 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 από το *place and route*, ακριβώς επειδή έχει υπολογίσει τις θέσεις των διαφόρων πυλών στο επιλεχθέν *FPGA chip*, είναι σε θέση να γνωρίζει τις καθυστερήσεις των σημάτων, και άρα αυτές λαμβάνονται υπ'όψιν στη χρονική προσομοίωση. Έτσι, σε *rtl* επίπεδο χρησιμοποιούνται *delta unit times* για τις καθυστερήσεις, ενώ στο *back annotation* εμπεριέχονται τα πραγματικά *delay times* των πυλών.

2.3.8 FPGA

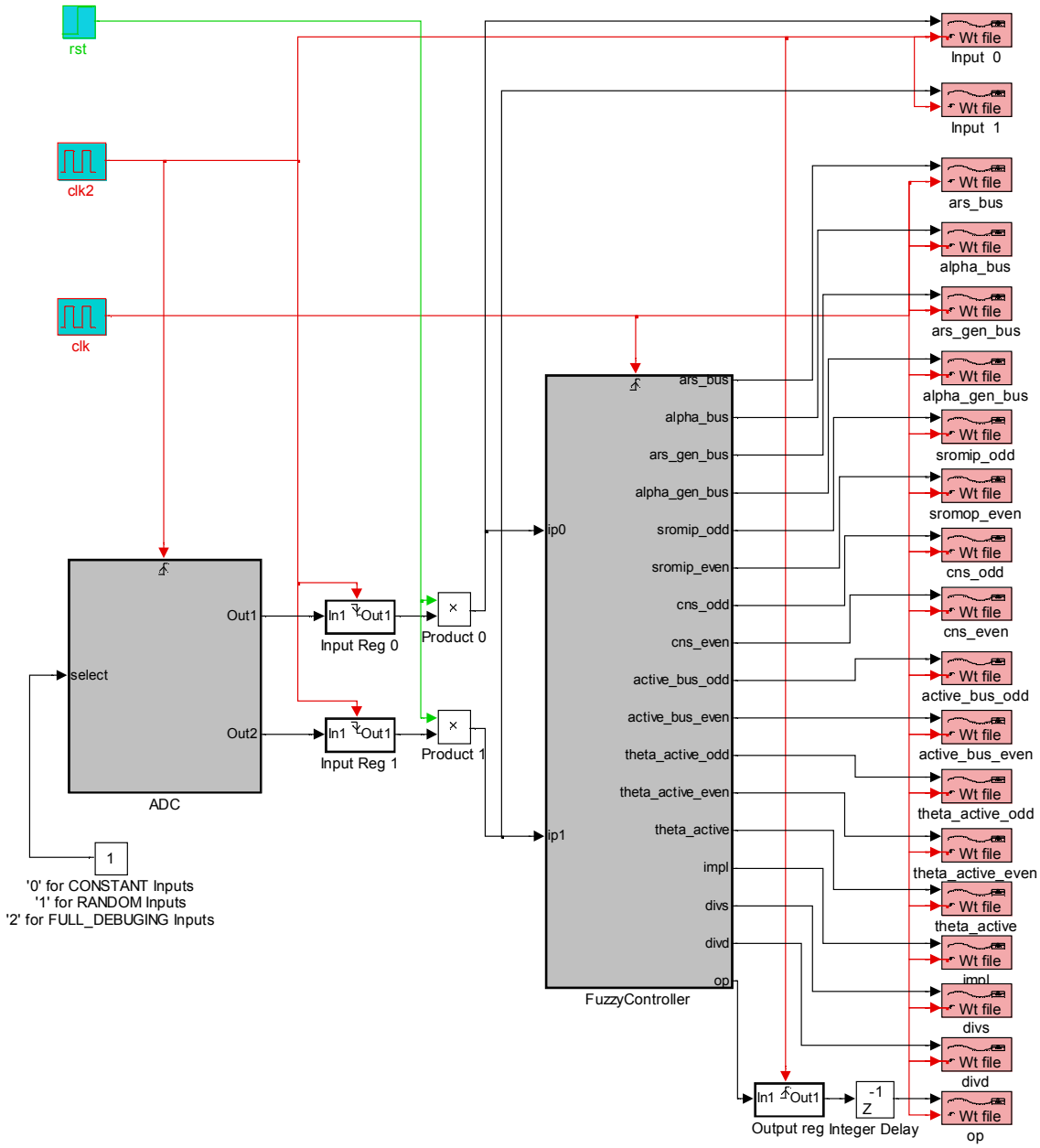
Τελευταίο βήμα είναι η παραγωγή του αρχείου *bitstream* το οποίο θα "φορτωθεί" (*download)* στο chip. Φυσικά, μετά την φόρτωση του αρχείου στο chip και τη δημιουργία του πραγματικού κυκλώματος μέσα σε αυτό, είναι πλέον δυνατός ο έλεγχος αυτού στον πάγκο για τυχόν λάθη *(bench testing)* αλλά και κάτω από τις πραγματικές συνθήκες εργασίας.



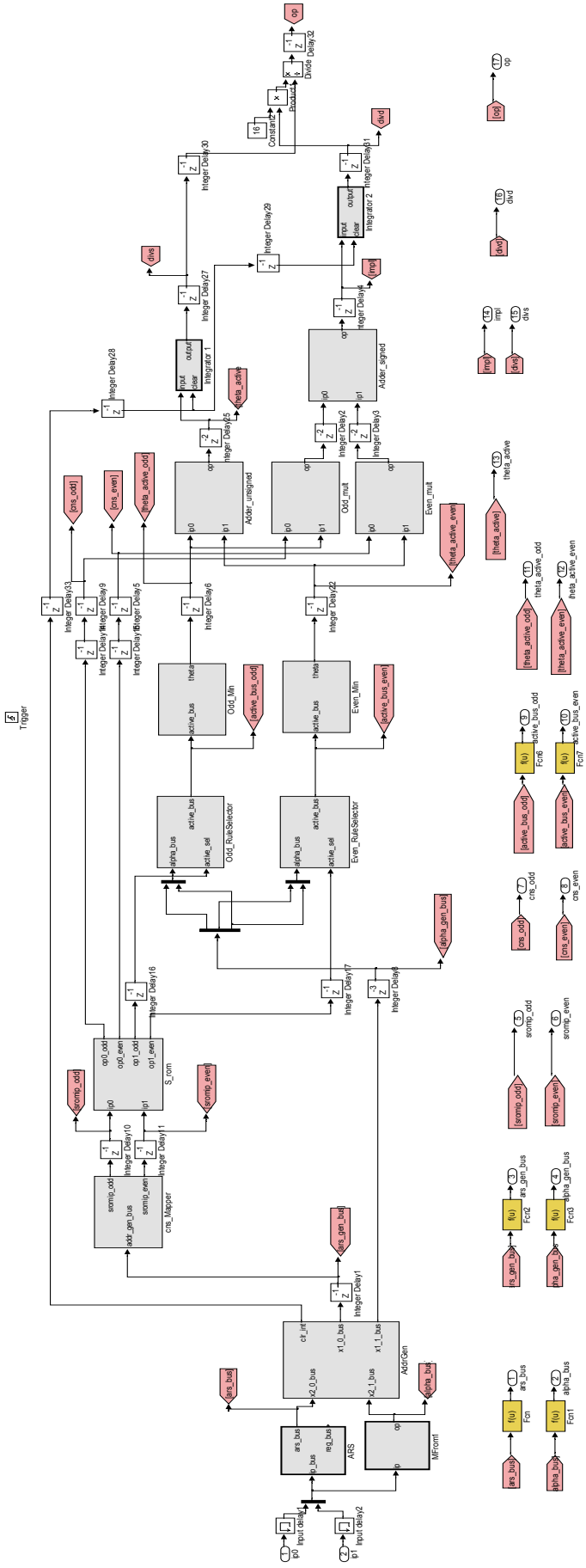
Σχ27: *Simulink: Testbench Ασαφούς Ελεγκτή με Αριθμητική Συνάρτηση Συμμετοχής*



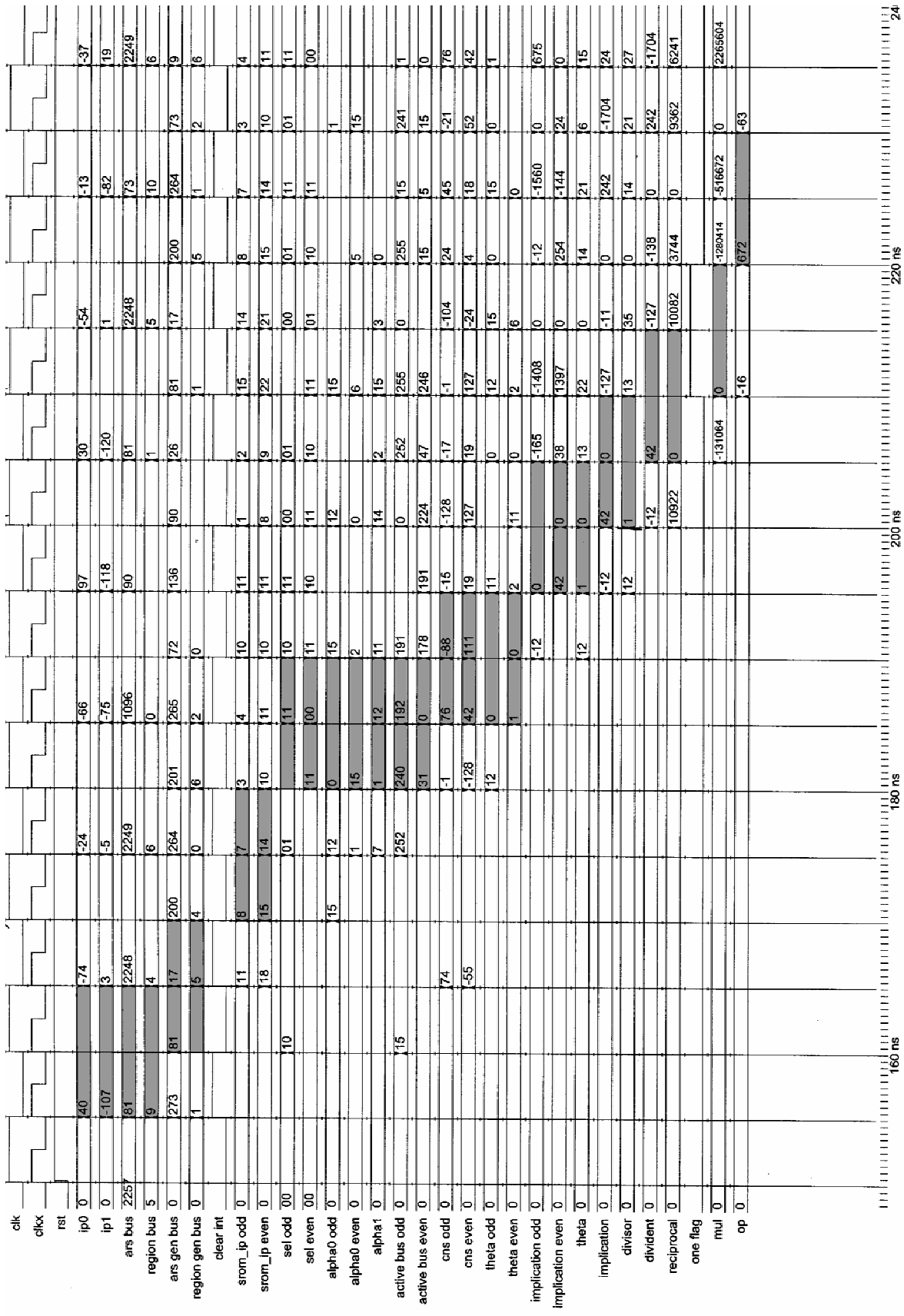
Σχ28: *Simulink: Μοντέλο Ασαφούς Ελεγκτή με Αριθμητική Συνάρτηση Συμμετοχής*



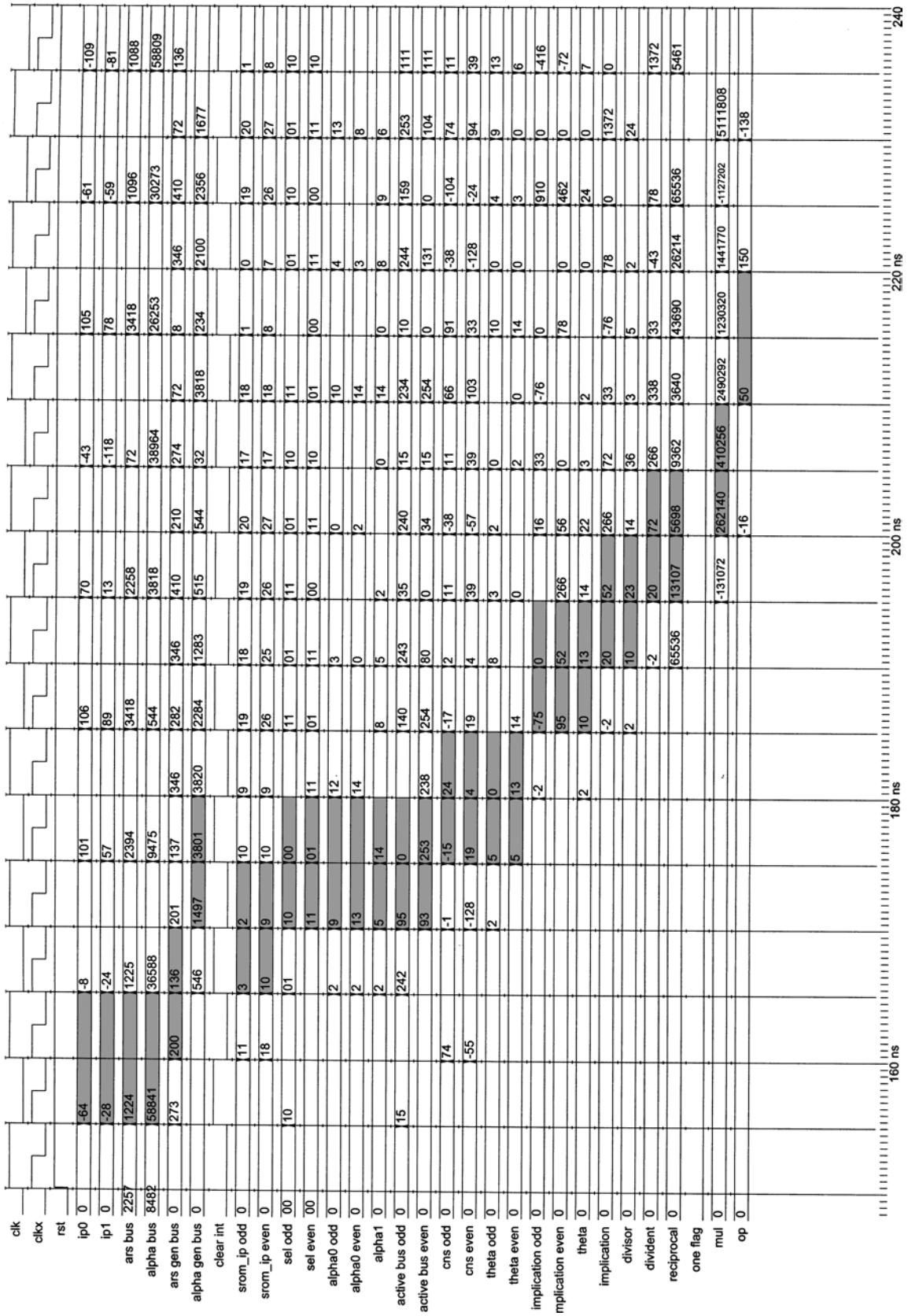
Σχ29: *Simulink: Testbench Ασαφούς Ελεγκτή με Συνάρτηση Συμμετοχής βασισμένη σε ROM*



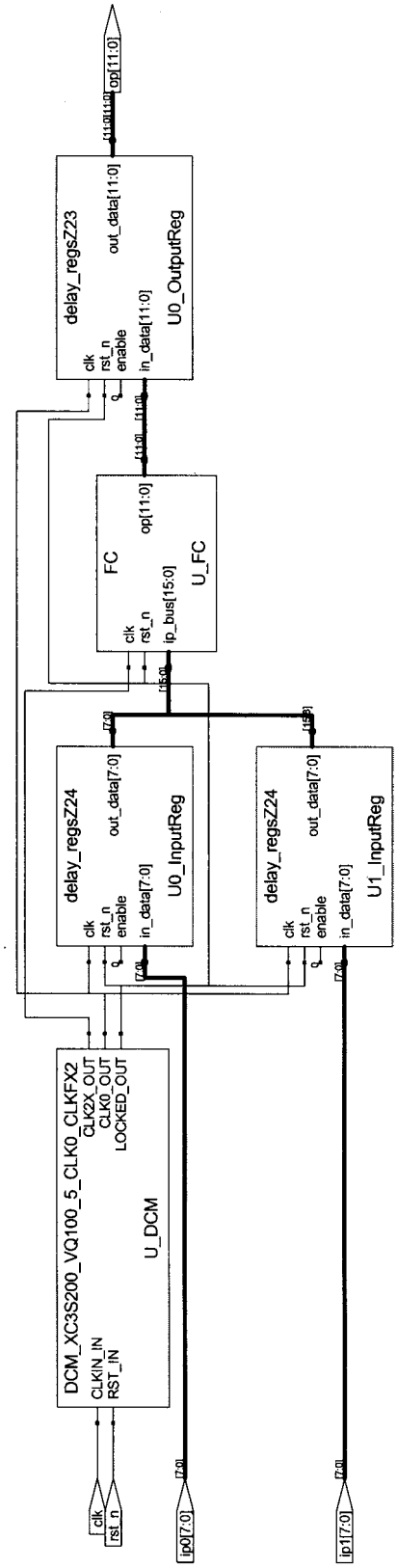
Σχ30: *Simulink: Μοντέλο Ασαφούς Ελεγκτή με Συνάρτηση Συμμετοχής βασισμένη σε ROM*



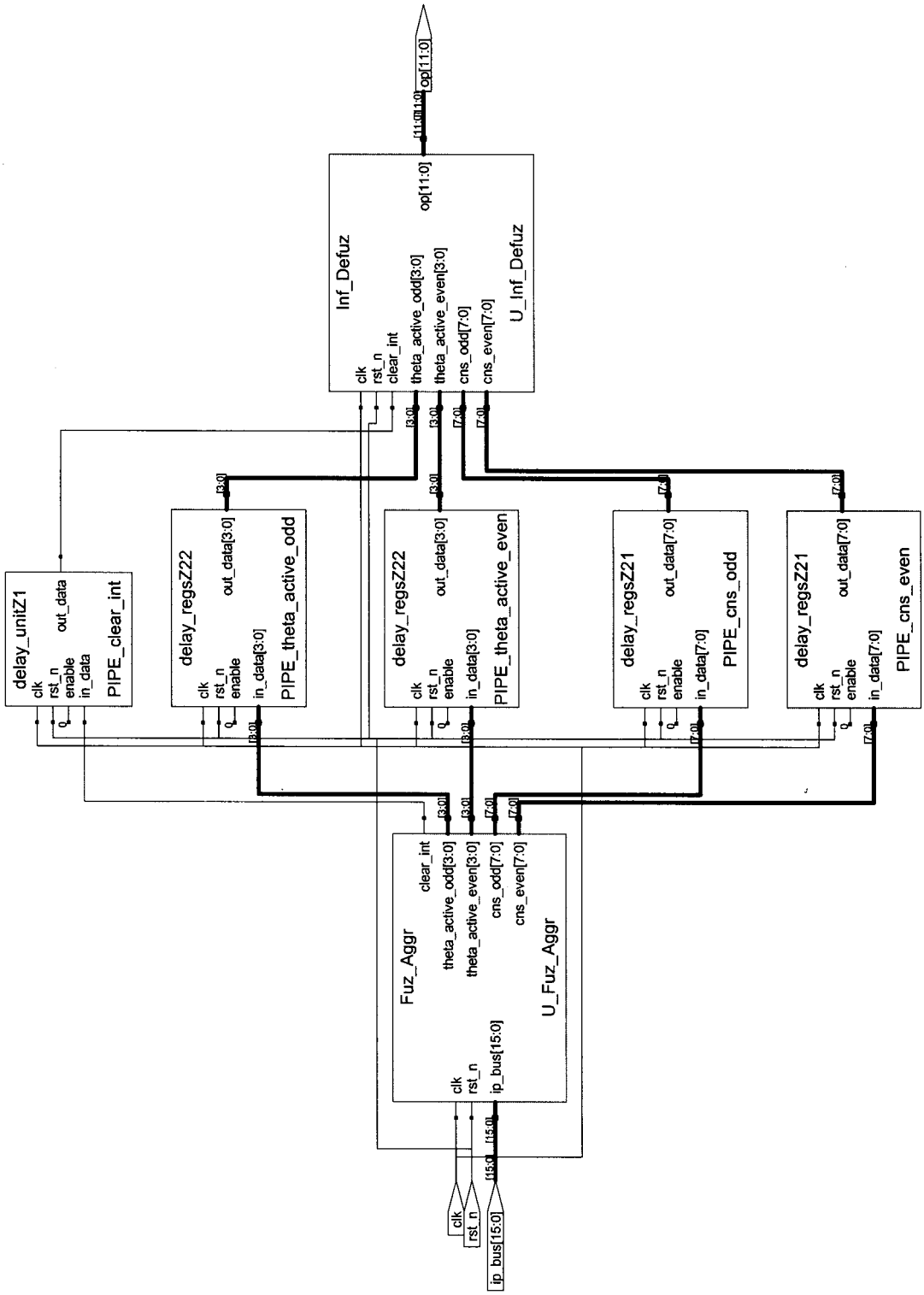
Σχ31: *Modelsim: rtl simulation: Για το μοντέλο με Αριθμητική Συνάρτηση Συμμετοχής*



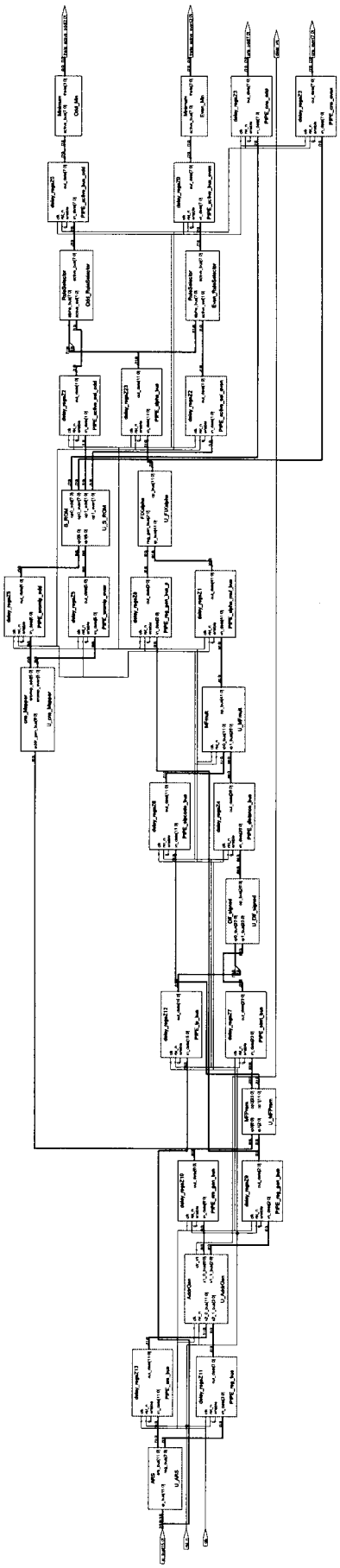
Σχ32: *Modelsim: rtl simulation: Για το μοντέλο με Συνάρτηση Συμμετοχής βασισμένη σε ROM*



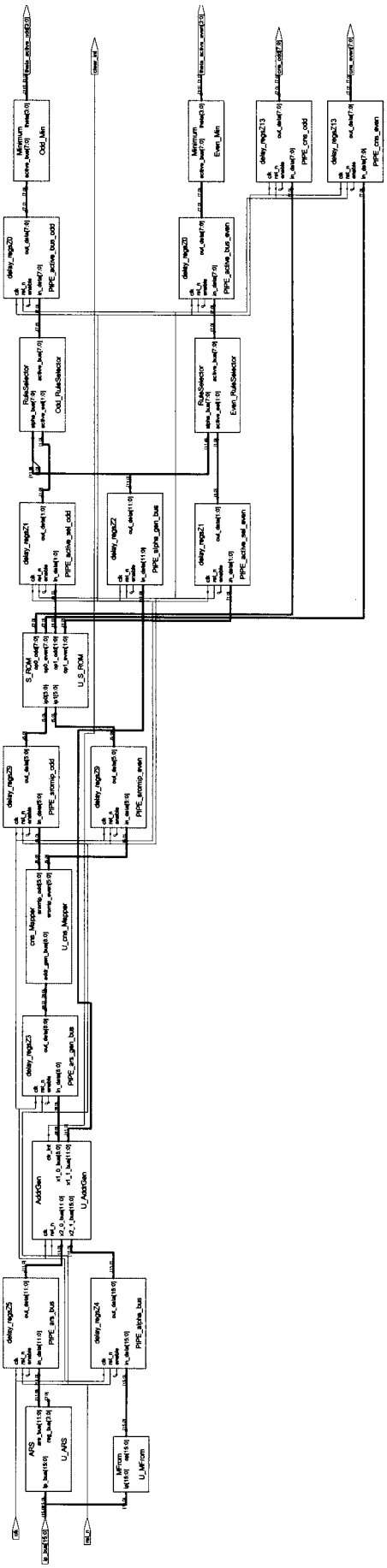
Σχ33: *Sinplify: rtl netlist: FPGA*



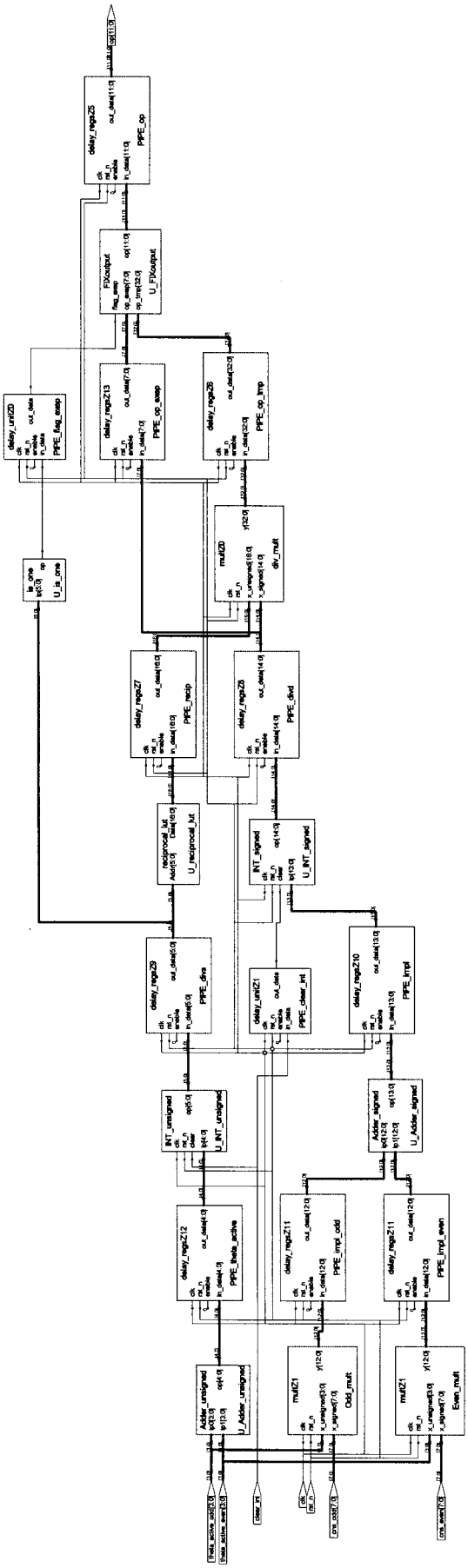
Σχ34: *Sinplify: rtl netlist: Ασαφής Ελεγκτής*



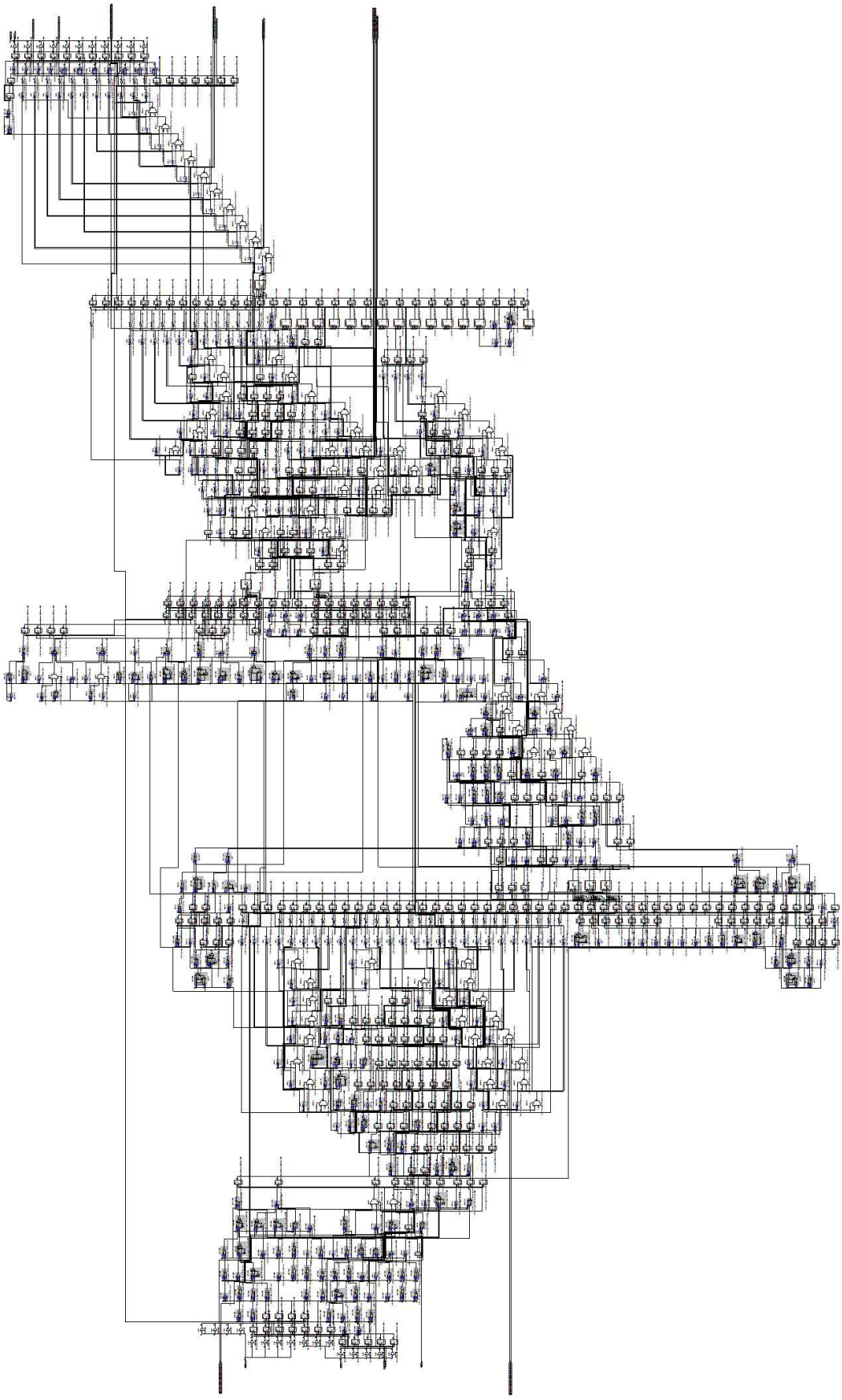
Σχ35: *Sinplify: rtl netlist: Fuzzification & Aggregation μέρος με Αριθμητική Συνάρτηση Συμμετοχής*



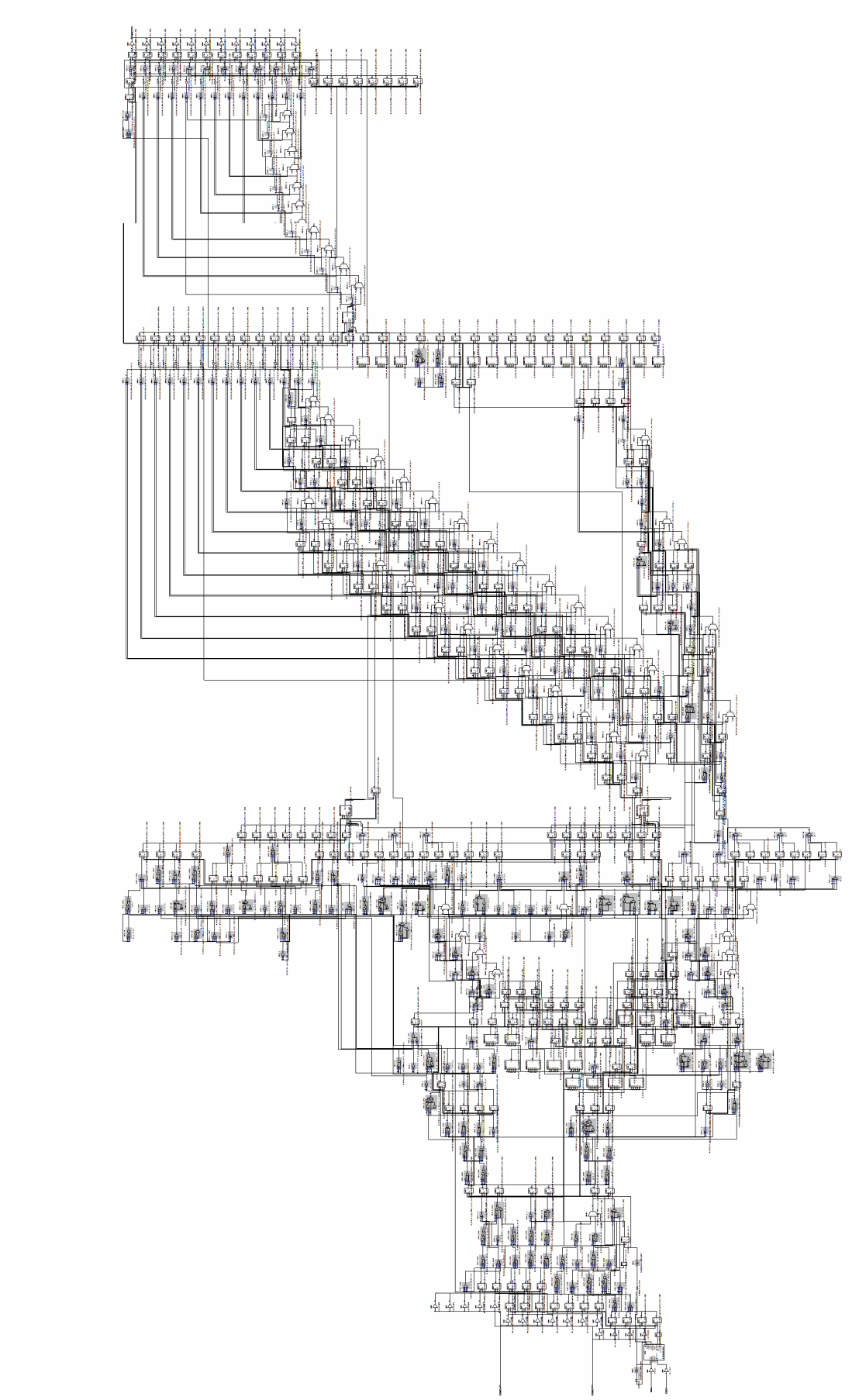
Σχ36: *Sinplify: rtl netlist: Fuzzification & Aggregation μέρος με Συνάρτηση Συμμετοχής βασισμένη σε ROM*



Σχ37: *Sinplify: rtl netlist: Inference & Defuzzification μέρος*



Σχ38: *Sinplify: gate netlist: Μοντέλο με Αριθμητική Συνάρτηση Συμμετοχής*



Σχ39: *Sinplify: gate netlist: Μοντέλο με Συνάρτηση Συμμετοχής βασισμένη σε ROM*

Πίνακας5: *Xilinx: Device Utilization summary & Constrains για το μοντέλο με Αριθμητική συνάρτηση συμμετοχής*

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

Πίνακας6: *Xilinx: Device Utilization summary & Constrains για το μοντέλο με συνάρτηση συμμετοχής βασισμένη σε ROM*

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

3. ΣΥΜΠΕΡΑΣΜΑΤΑ

Σκοπός της Διπλωματικής εργασίας ήταν η σχεδίαση ενός ασαφούς ελεγκτή τύπου *Takagi-Sugeno μηδενικής τάξης* σε *VHDL* και η υλοποίηση αυτού σε *FPGA chip*. Ιδιαίτερο βάρος δόθηκε στο σχεδιασμό του ελεγκτή ώστε να είναι παραμετρικός και να επεξεργάζεται δεδομένα με μεγάλη συχνότητα.

Η παραμετρικότητα επιτεύχθηκε με τον χωρισμό του ελεγκτή σε ανεξάρτητες δομικές μονάδες *(blocks)* με διακριτές λειτουργίες και την περιγραφή καθενός από αυτά σε γλώσσα υλικού *VHDL*, η οποία παρέχει τη δυνατότητα παραμετρικότητας μέσω των *"Generics"*.

Για την μείωση των κύκλων υπολογισμού υποθέσαμε επικάλυψη μεταξύ των γειτονικών ασαφών συνόλων, το πολύ μέχρι δύο 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 [19]. 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 $1/2^n$.

Στην παρούσα διπλωματική, 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 σπάζοντας τη μνήμη των κανόνων *(singletons ROM)* και τη μνήμη των παραμέτρων των τραπεζοειδών ασαφών συνόλων της πρώτης εισόδου *(parameter Memory Bank)* σε μέρη *ODD* και *EVEN*. Το σπάσιμό των μνημών είναι θεμιτό 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 *(singletons ROM)* 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 *Trapezoidal Generator*, ενός *Rule Selector*, ενός *Min*, ενός πολλαπλασιαστή και δύο αθροιστών. Αξίζει δε να σημειωθεί ότι η αύξηση της λογικής λόγω του μοντέλου *ODD-EVEN* οφείλεται αποκλειστικά και μόνο στην τοποθέτηση των προαναφερθέντων *blocks* ανεξάρτητα από τον αριθμό των εισόδων του ασαφούς ελεγκτή. Επιπλέον, η επεξεργασία κάθε κανόνα γίνεται ανεξάρτητα, και μόνο στους δύο *Adders (Signed-Unsigned)* γίνεται συνδυασμός των αποτελεσμάτων τους. Άρα, η εισαγωγή του μοντέλου *ODD-EVEN* προκαλεί μια αύξηση της συνολικής καθυστέρησης *(delay)* μόνο κατά *1 pipe stage,* εξ'αιτίας της παρεμβολής των *Adders* πριν από τους ολοκληρωτές. Το συμπέρασμα, συνεπώς, που προκύπτει από την εξέταση του μοντέλου *ODD-EVEN* 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, προτείναμε δύο εναλλακτικές υλοποιήσεις. Μία με αριθμητικό υπολογισμό των συναρτήσεων συμμετοχής (με χρήση του *Trapezoidal Generator*) 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 *chip* 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 *blocks*. 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 απο-ασαφοποίησης. Η διαίρεση είναι μία από τις πιο δύσκολες και απαιτητικές αριθμητικές πράξεις όσον αφορά την κυκλωματική της υλοποίηση. Για το θέμα αυτό έχουν προταθεί πολλές μέθοδοι. 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 [30]. Υπάρχει και μια άλλη μέθοδος η οποία χρησιμοποιεί το εσωτερικό ενός πολλαπλασιαστή για να πραγματοποιήσει την πράξη της διαίρεσης [31]. 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 εκφυλίσουμε 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 ***Πίνακα1***. Η σύνθεση έγινε με τη βοήθεια του *Sinplicity* της *Synplify*, και το place and route σε ένα *FPGA chip Spartan-3 1500* 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 chip)*. 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 chip)*. Οι δύο αρχιτεκτονικές ελέγχθηκε ότι λειτουργούν για την περίπτωση $3^{ων}$ εισόδων καθώς και για ανάλυση συνάρτησης συμμετοχής *8-bit*.

4. ΠΡΟΤΕΙΝΟΜΕΝΗ ΣΥΝΕΧΕΙΑ

Στο κεφάλαιο αυτό παρουσιάζονται ορισμένες προτάσεις για συνέχιση της δουλειάς που πραγματοποιήθηκε στα πλαίσια της διπλωματικής.

4.1 Γεννήτρια VHDL κώδικα Ασαφούς Ελεγκτή (Fuzzy Controller VHDL Core Generator)

Σαν συνέχεια της διπλωματικής εργασίας έχει αρχίσει η κατασκευή ενός *core generator* για την κατασκευή *ασαφών ελεγκτών*. Ένα πρόγραμμα δηλαδή το οποίο θα διαθέτει κατάλληλο γραφικό περιβάλλον επιτρέποντας στο χρήστη να διαλέξει τα χαρακτηριστικά του προς υλοποίηση ελεγκτή. Στη συνέχεια, επεξεργάζοντας κατάλληλα το *VHDL* κώδικα, θα παράγει σαν έξοδο την περιγραφή του ζητούμενου ελεγκτή σε *VHDL*. Τον παραγόμενο κώδικα θα μπορεί έπειτα ο χρήστης να τον μετατρέψει σε επίπεδο λογικών πυλών με κάποιο πρόγραμμα σύνθεσης (πχ. *Sinplicity*). Το *netlist* που θα προκύψει από τη σύνθεση θα μπορεί έπειτα να περάσει από τη διαδικασία του *Place and Route*, για να φορτωθεί τέλος σε ένα *FPGA chip*.

Η δημιουργία ενός τέτοιου *core generator* έχει ήδη αρχίσει να γίνεται σε *Java*. Με την ολοκλήρωσή του, θα δοθεί η δυνατότητα ελέγχου της παραμετρικότητας του *VHDL* κώδικα, η οποία στην παρούσα φάση είναι δύσκολη αφού απαιτείται η φόρτωση των ROM εκ νέου και ο υπολογισμός της κατάλληλης ακρίβειας του ανάστροφου του διαιρέτη (*reciprocal)*, σε κάθε αλλαγή των χαρακτηριστικών του ελεγκτή. Επίσης, κάποια σημεία κώδικα τα οποία δεν μπορούν να περιγραφούν παραμετρικά σε *VHDL*, θα αναλάβει να τα συμπληρώνει το πρόγραμμα του core generator. Θα δοθεί τέλος η δυνατότητα σε κάποιον ο οποίος δεν γνωρίζει *VHDL*, να κατασκευάσει ένα ασαφή ελεγκτή σε *VHDL*. Η δημιουργία του *core generator* θα γίνει με τέτοιο τρόπο ώστε να είναι εύκολα επεκτάσιμος με την προσθήκη διαφορετικών

αρχιτεκτονικών του ασαφούς ελεγκτή ή των επιμέρους *δομικών μονάδων (blocks)*.

4.2 Προσαρμοστικός Ασαφής Ελεγκτής (Adaptive Fuzzy Controller)

Μια επέκταση τέτοια στην αρχιτεκτονική του ασαφούς ελεγκτή μπορεί να γίνει με την εισαγωγή ενός *προσαρμοστικού αλγόριθμου*. Ο αλγόριθμος αυτός θα ανανεώνει κατάλληλα της *μνήμες αποθήκευσης παραμέτρων (parameter memory banks)* ή τη μνήμη των κανόνων *(singletons RAM)*. Με τον τρόπο αυτό, θα προκύψει ένας *προσαρμοστικός ασαφής ελεγκτής (πχ. "νευροασαφής" ελεγκτής)*.

4.3 Σύγκριση Ασαφούς Ελεγκτή (Fuzzy Controller) με PID Ελεγκτή

Τέλος, προτείνεται η σύγκριση των δύο αρχιτεκτονικών του *ασαφούς ελεγκτή* που παρουσιάστηκαν, σε σχέση με ένα κλασσικό *PID ελεγκτή* για τον έλεγχο ενός πραγματικού συστήματος (πχ. *ανάστροφο εκρεμές*). Με αυτό τον τρόπο θα φανεί η πρακτική αξία της υλοποίησης.

ΑΝΑΦΟΡΕΣ

[1] L. A. Zadeh. Fuzzy Sets, *Inf. and Control*, 8:338-353, 1965

[2] Mamdani E. H. “Application of Fuzzy Logic to Approximate Reasoning using Linguistic Synthesis”, IEEE Trans. On Computers, Vol.C-26, No.12, 1977

[3] T. Takagi & M. Sugeno, “Derivation of Fuzzy Control Rules from Human Operator’s Control Actions”, Proc. of the IFAC Symp. on Fuzzy Information, Knowledge Representation and Decision analysis, pp.55-60, July 1983

[4] T. Takagi & M. Sugeno, “Fuzzy Identification of Systems and Its Application to Modeling and Control”, IEEE Trans. Syst. Man & Cybern,, Vol.20, No.2, pp.116-132, 1985

[5] Amine Trabelsi, Frederic Lafont, Mohamed Kamoun, Gilles Enea, “Identification of Nonlinear Multivariable Systems by Adaptive Fuzzy Takagi-Sugeno Model”, *International Journal of Computational Cognition*, Vol.2, No.3, pp.137-153, September 2004

[6] Σπύρος Γ. Τζαφέστας, “Αυτόματος Έλεγχος Γραμμικών και Μη Γραμμικών Συστημάτων Συνεχούς και Διακριτού χρόνου”, 2004

[7] Σπύρος Γ. Τζαφέστας, “Υπολογιστική Νοημοσύνη”, 2002

[8] Σπύρος Γ. Τζαφέστας, “Methods and Applications of Intelligent Control”, Vol.16, Kluwer Academic Publishers, 1997

[9] Kevin M. Passino, Stephen Yurkovich, “Fuzzy Control”, Addison-Wesley, 1997

[10] Jerry M. Mendel, “Fuzzy Logic Systems for Engineering: A Tutorial”, IEEE, Vol.83, No.3, March 1995

[11] Abraham Kandel, Gideon Langholz, “Fuzzy Control Systems”, CRC Press, ISBN: 0849344964, January 1993

[12] Godjevac J., “Neuro-Fuzzy Controllers Design and Application”, Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1997

[13] Kazuo Tanaka, Hua O. Wang, “Fuzzy Control Systems Design and Analysis: A Linear Matrix Approach”, John Wiley & Sons Inc., 2001

[14] M. Togai, H. Watanabe, “A VLSI Implementation of a Fuzzy Inference Engine: Toward an Expert System on a chip”, *Inform. Sci.*, Vol.38, pp.147-163, 1986

[15] T. Yamakawa and T. Miki, “The Current Mode Fuzzy Logic Integrated Circuits Fabricated by the Standard CMOS Process”, *IEEE Trans. Comput.*, Vol.C-35, pp.161-167, February 1986

[16] T. Yamakawa, “High Speed Fuzzy Controller Hardware System: The mega-FLIPS Machine in Fuzzy Computing”, Inf. Sci. M. M. Gupta and T.Yamakawa, Eds. Amsterdam, The Netherlands: Elsevier, 1988, Vol.45, No.1, pp.113-128

[17] H. Watanabe, W. D. Dettloff, K. E. Yount, “A VLSI Fuzzy Logic Controller with Reconfigurable, Cascadable Architecture”, IEEE J. Solid-State Circuits, Vol.25, No.2, pp.376-382, April 1990

[18] K. Shimuzu, M. Osumi, F. Imae, “Digital Fuzzy Processor FP-5000”, Proc. 2nd *Int. Conf. Fuzzy Logic Neural Networks*, Iizuka, Japan, pp.539-542, July 1992

[19] Alessandro Gabrielli, E.Gandolfi, “A Fast Digital Fuzzy Processor”, *IEEE Micro*, Vol.17, pp.68-79, 1999

[20] E. Lago, M. A. Hinojosa, C. J. Jiménez, A. Barriga, S. Sánchez-Solano, “FPGA Implementation Of Fuzzy Controllers”, *XII Conference on*

Design of Circuits and Integrated Systems (DCIS'97), pp.715-720, Sevilla, Noviembre 18-21, 1997

[21] C. J. Jiménez, A. Barriga, S. Sánchez-Solano, "Digital Implementations of SISC Fuzzy Controllers", *Proc. Int. Conf. on Fuzzy Logic, Neural Nets and Soft Computing*, pp.651-652, Iizuka, 1994

[22] M. J. Patyra, J. L. Grantner, K. Koster, "Digital Fuzzy Logic Controller: Design and Implementation", *IEEE Trans. Fuzzy Systems*, Vol.4, No.4, pp.439-459, November 1996

[23] H. Eichfeld, T. Künemund, M. Menke, "A 12b General-Purpose Fuzzy Logic Controller Chip", *IEEE Trans. Fuzzy Systems*, Vol.4, No.4, pp.460-475, November 1996

[24] Μπούρας, "Mixed Analog-Digital Fuzzy Logic Controller with Continuous Amplitude Fuzzy Inferences and Defuzzification", IEEE Transactions on Fuzzy Systems, Vol.6, No.2, pp. 205-215, May 1998

[25] Carlos Dualibe, Michel Verleysen and Paul G.A. Jespers, "Design of Analog Fuzzy Logic Controllers in CMOS Technologies – Implementation, Test and Application", Kluwer Academic Publishers, 2003

[26] S. Sjoholm, L. Lindh, "VHDL for Designers", 1st Edition, Prentice Hall PTR, January 10, 1991

[27] Douglas L. Perry, "VHDL: Programming by Example", 4th Edition, McGraw-Hill, 2002

[28] Xilinx, "Spartan-3 FPGA Family: Complete Data Sheet", DS099 December 24, 2003

[29] Xilinx, "Using Embedded Multipliers in Spartan-3 FPGAs", XAPP467 (v1.1), May 13, 2003

[30] Derek Wong, Michael Flynn, “Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations”, *IEEE Transactions on Computers*, Vol. 41, No. 8, August 1992

[31] Eric M. Schwarz, Michael J. Flynn, “Using a Floating-Point Multiplier’s Internals for High-Radix Division and Square Root”, Technical Report CSL-TR-93-554, Computer Systems Laboratory, Stanford University, January 1993

[32] P. E. Madrid, B. Millar, and E. E. Swartzlander, “Modified Booth algorithm for high radix fixed-point multiplication” *IEEE Trans. VLSI,* Vol.1 , No.2, pp.164-167, June 1993.

[33] K. M. Deliparaschos, F. I. Nenedakis, S. G. Tzafestas, “A Fast Digital Fuzzy Logic Controller: FPGA Design and Implementation”, *ETFA 2005*, *Automation Systems & Control*, under review.

ΠΑΡΑΡΤΗΜΑ

Οι παρακάτω πίνακες προσφέρουν πληροφορίες για την ιστορία και τις εφαρμογές των ασαφών συστημάτων και έχουν ληφθεί από το βιβλίο *«Design of Analog Fuzzy Logic Controllers in CMOS Technologies – Implementation, Test and Application»* [25].

Table 1.1. The historical development of Fuzzy Logic until 1993. From [KaLa98], © 1998 Kluwer Academic Publishers, reprinted with permission.

Year	Event
1961	Lotfi Zadeh claims in his paper for a new kind of "fuzzy" mathematics.
1965	Lotfi Zadeh introduces the concept of fuzzy sets.
1969	Marinos (Duke University) conducts the first research aiming at hardware implementation of Fuzzy Logic.
1972	M. Sugeno presents the idea of fuzzy measures.
1974	E. Mamdani presents a fuzzy application to control a steam machine in an academic framework.
1980	Yamakawa build the first fuzzy circuit with discrete bipolar components [Yama80].
1982	The first industrial application in a cement kiln in Denmark [HoOs82].
1984	Togai and Watanabe present the first VLSI implementation of Fuzzy Controllers [ToWa85].
1986	Hitachi puts into operation a fuzzy controlled subway system.
1987	Yamakawa presents the first analog Fuzzy Controller.
1988	Togai implements the first digital fuzzy processors.
1989	LIFE: Laboratory for International Fuzzy Engineering Research starts in Japan.
1990	Yamakawa establishes the Fuzzy Logic Systems Institute (FLSI) in Japan.
1992	The first IEEE International Conference on Fuzzy Systems.
1993	The first issue of the IEEE Transactions on Fuzzy Systems.

Table 1.2. Main R&D areas in Fuzzy Logic and their major topics (FUZZ-IEEE' 2001).

R&D Area	Main Topics
Fuzzy Mathematics:	Foundations of Fuzzy Logic, approximate reasoning, evolutionary computation, identification and learning algorithms, rule base optimization.
Control Systems:	Fuzzy Control theory and applications, process and environmental control, stability criterions issues, multi-level supervisory control.
Pattern Recognition and Image Processing:	Supervised and unsupervised learning, classifiers design and integration, signal/image processing and analysis, computer vision, multimedia applications.
Soft Computing and Hybrid Systems:	Intelligent information systems, data base systems, data mining, intelligent agents, reliability engineering, Neuro-Fuzzy Systems, Internet computing, networks traffic modeling and control.
Electronic Systems:	Fuzzy hardware implementation and embedded applications.
Robotics and Automation:	Fuzzy Logic in robotics, industrial automation and other industrial applications.

Table 1.3. Some reported applications of Fuzzy Logic in different domains.

Application	Reference
Process Control	
5MW Nuclear Reactor Control	[Bern88]
Cement Kiln control	[HoOs82]
Sake Brewing Manufacturing	[OiTo91]
Activated Sludge Wastewater Plant	[ToBe80]
Water Purification Plant	[Yalt85]
Multilayer Waste Incinerator	[SuKa86]
Environmental Emissions Control	[TaSa94]
Aero-Space Applications	
Aircraft Carrier Landing System	[Stie93]
Attitude Control of a Flexible Satellite	[DaGi89]
Flexible Wing Roll Control	[ChCh91]
Pitch Control of an Interceptor Missile	[Chri93]
Robotics and Automation	
Mobile Robot Control	[SaRu93]
Robot Arm Control	[ScMa85]
Container Crane Control	[YaHa86]
Industrial Automotive	
Car Modeling in Extreme Situations	[AlKr92]
Automotive Automatic Transmission	[SaSa93]
Automotive Anti-Skid Braking	[LaPa93]
Autonomous Vehicle Controller	[HaRe89]
Automatic Train Operation	[YaMi85]
Car Parking Modeling	[SuMu85]
Automotive Engine Idle Control	[VaFa93]
Image Processing	
Real-Time Motion Estimation	[ChJo99]
Image Color Correction Systems	[HuKe00]
Facsimile Image Fuzzy Processor	[LeLe93]
Image Processing in IQTV	[MaDa95]
Fuzzy Stack Filters for Image Processing	[YuCh96]
Signal Processing	
Non-linear Channel Equalization	[WaMe94]
AGC Control for Radio Communications	[BaDi00]
Decision Feedback Equalizer	[LiMe00]
Magnetic Recording Equalization	[Mage94]
Analog Filter Tuning	[MoPi94]
Adaptive Noise Cancellation	[PaKo00]
Real-Time Source Separation	[ToHa98]
Fuzzy Digital PLL Filter	[SiEl95]
Earthquake Precursors Detection	[ItWa92]
Power Electronics	
Control of DC/DC Converters	[CrLi98]
Control of PWM DC Converters	[FrMa98]
Embedded Low Cost Control for DC Motors	[OeGr96]
Fuzzy Control of Induction AC Motors	[ClTu92]

Table 1.4. Main advantages and disadvantages for each hardware implementation option.

Alternative	Advantages	Disadvantages
1. General-Purpose Processors:	- Complete flexibility - Short prototyping time - For high complexity FLCs - Availability of development systems [Moto92] [Info94]	- Low speed (control systems with slow dynamics: 0.1-1 KHz)
2. General-Purpose Processors with Dedicated Fuzzy Instructions:	- Complete flexibility - Short prototyping time - For high complexity FLCs - Higher speed compared with 1: 1-10KHz (CISC), 1-50KHz (RISC) - Simple extension of existing CPU cores [Info94]	- Higher cost than 1 (due to small production volume) - Low performance for high-end applications
3. Special-Purpose Coprocessors:	- Higher speed than 1 and 2 (i.e. 10-100KHz) - For moderate complexity FLCs (256-rule, 16-I/16-O) - Short prototyping time - Modularity	- Need of general-purpose processor to support non-fuzzy operations - Limited flexibility [ShOs92] [PaPo92]
4. Dedicated ASICs:	- Fast processing, tuned to the application (i.e. 0.1-1MHz) - Availability of high-level synthesis tools [FrMa98] [HoHa96] - Low cost in terms of die area - Available as a subsystem block for embedded applications - Cost-effective for high production volumes	- Fixed applications - Low flexibility (minimal programmability) - Limited complexity FLCs - Only for ASIC or FPGA based systems - Large prototyping time

Παρακάτω παρουσιάζονται δύο αρχεία VHDL κώδικα που χρησιμοποιήθηκαν για την περιγραφή του ασαφούς ελεγκτή :

```
--------------------------------------------------------------------------------
-- Title      : Fuzzy Controller
-- Project    : Fuzzy Controller
--------------------------------------------------------------------------------
-- File       : FC.vhd
-- Author     : Faidon-Iosif Nenedakis & Kyriakos Deliparaschos
-- Company    : NTUA/IRAL
-- Created    : 05/06/2004
-- Platform   : Modelsim 5.4 & FPGA Compiler-II 3.5
--------------------------------------------------------------------------------
-- Description: Το αρχείο αυτό περιράφει τη δομή του Fuzzy Controller.Σε αυτό
--              γίνεται η σύνδεση μεταξύ των επιμέρους blocks, ενώ παρεμβάλονται
--              μεταξύ των συνδέσεων και τα pipe stages.Το block είναι πλήρως
--              παραμετρικό.
--------------------------------------------------------------------------------
-- 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      : Fix alpha
-- Project    : Fuzzy Controller
-------------------------------------------------------------------------------
-- File       : FIXalpha.vhd
-- Author     : Faidon-Iosif Nenedakis & Kyriakos Deliparaschos
-- Company    : NTUA/IRAL
-- Created    : 05/06/2004
-- Platform   : Modelsim 5.4 & FPGA Compiler-II 3.5
-------------------------------------------------------------------------------
-- Description: Το αρχείο αυτό περιγράφει τη λειτουργία του Fix alpha.Δεν υπάρχει
--              delay στην έξοδο.Είναι πλήρως παραμετρικό.
-------------------------------------------------------------------------------
-- 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 FIXalpha is
  generic (
    ip_no     : integer := 3;
    ip_sz     : integer := 14;
    op_sz     : integer := 4;
    slp_trunc : integer := 3);
  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 entity FIXalpha;

-------------------------------------------------------------------------------
-- ARCHITECTURE
-------------------------------------------------------------------------------
architecture rtl of FIXalpha is

begin -- architecture rtl

  A: for i in 0 to ip_no-1 generate

  signal reg      : std_logic;
  signal ovf      : std_logic;
  signal zer      : std_logic;
  signal sel      : std_logic_vector(2 downto 0);
  signal ip       : std_logic_vector(ip_sz-1 downto 0);
  signal op_i     : std_logic_vector(op_sz-1 downto 0);
  signal op       : std_logic_vector(op_sz-1 downto 0);
  signal zeros    : std_logic_vector(ip_sz-1-op_sz-slp_trunc-1 downto 0);

  begin -- generate

  -- input data select
    reg <= reg_gen_bus(i);
    ip <= ip_bus((i+1)*ip'length -1 downto i*ip'length);

  -- constant
    zeros <= (others => '0');

  -- overfloat
```

```
    ovf <= '0' when ip(ip'left -1 downto op_sz+slp_trunc) = zeros else '1';

  -- zero
    zer <= ip(ip'left);

  -- truncation
    op_i <= ip(op_sz+slp_trunc-1 downto slp_trunc);

  -- correct
    sel <= zer & ovf & reg;

    correct_proc : process (sel,op_i)
      begin -- process select
          case sel is
            when "000" =>
              op <= op_i;
            when "001" =>
              op <= not op_i;
            when "010" =>
              op <= (others => '1');
            when others =>
              op <= (others => '0');
          end case;
      end process correct_proc;

  -- output
    op_bus((i+1)*op'length -1 downto i*op'length) <= op;

  end generate;

end architecture rtl;
```