Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Σχεδίαση και υλοποίηση σε FPGA του μικροεπεξεργαστή PicoBlaze με την τεχνική Triple Modular Redundancy**
Ονοματεπώνυμο Φοιτητή	**Μακαντάση Βασιλική του Γεωργίου**
Αριθμός Μητρώου	**ΜΠΣΠ/07052**
Κατεύθυνση	**Τεχνολογία Ενσωματωμένων Υπολογιστικών Συστημάτων**
Επιβλέπων	**Μιχάλης Ψαράκης, Λέκτορας**

Ημερομηνία Παράδοσης **Οκτώβριος 2010**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Μιχάλης Ψαράκης Λέκτορας	Δημήτριος Γκιζόπουλος Αναπληρωτής Καθηγητής	Χαράλαμπος Κωνσταντόπουλος Λέκτορας

Περίληψη

Σκοπός αυτής της μεταπτυχιακής διατριβής είναι να παρουσιάσει την τεχνική τριπλού αρθρωτού πλεονασμού (Triple Modular Redundancy - TMR), τόσο σε θεωρητικό επίπεδο όσο και σε επίπεδο εφαρμογής. Αρχικά γίνεται μια εισαγωγή στις διάφορες τεχνικές ανοχής σφαλμάτων και στις επιδράσεις των σφαλμάτων στις διάφορες εφαρμογές. Παρατίθεται μια λεπτομερής περιγραφή της μεθόδου TMR και του PicoBlaze (Xilinx), της CPU που τροποποιήθηκε σύμφωνα με τις προτάσεις της Xilinx για την υλοποίηση της ενισχυμένης, TMR, εκδοχής της, καθώς και όλα τα βήματα που ακολουθήθηκαν για την επίτευξη αυτού του σκοπού. Στη συνέχεια γίνεται εισαγωγή σφαλμάτων στη σχεδίαση, μόνιμων και προσωρινών, και μελετάμε τη συμπεριφορά της TMR έκδοσης του PicoBlaze για να αποδείξουμε ότι είναι ανεκτικός σε σφάλματα. Επίσης υπολογίζουμε τις επιπτώσεις της εφαρμογής της τεχνικής TMR στον επεξεργαστή PicoBlaze όσον αφορά την απόδοσή του και τους πόρους της συσκευής FPGA που καταλαμβάνει. Τέλος, γίνονται προτάσεις για μελλοντικές εφαρμογές.

Abstract

The objective of this thesis is to present the Triple Modular Redundancy (TMR) technique, in theory and in practice. Initially, several fault tolerant techniques are described, and the impact of upsets in various applications is discussed. A detailed description of the TMR method is given as well as of the PicoBlaze microcontroller, which is the CPU that was adapted according to Xilinx's suggestions for the Triple Modular Redundancy implementation. The steps that resulted in the accomplishment of this goal are thoroughly described. Then, we inject both permanent and temporary faults in the TMR-version of PicoBlaze to prove its fault-tolerant capability. Also, we calculate the impact of Triple Modular Redundancy on the FPGA-based PicoBlaze in terms of performance and device resources utilization. Finally, future implementations are suggested.

Περιεχόμενα

Κατάλογος εικόνων

Κατάλογος πινάκων

Εισαγωγή

Η ανάγκη να προστατεύσουμε τα ολοκληρωμένα κυκλώματα ενάντια στα σφάλματα γίνεται συνεχώς όλο και πιο απαραίτητη. Έτσι βασιζόμενοι στον ορισμό της ανοχής σφαλμάτων, σκοπός μας είναι να διατηρήσουμε τη σωστή λειτουργία του ολοκληρωμένου κυκλώματος παρόλη την ύπαρξη σφαλμάτων.

Η ανάπτυξη των τεχνικών ανοχής σφαλμάτων όμως είναι άμεσα συνδεδεμένη με την συσκευή στην οποία θα υλοποιηθεί μια εφαρμογή. Για κάθε τύπο κυκλώματος υπάρχει ένα σύνολο κατάλληλων λύσεων. Η βιομηχανία ολοκληρωμένων κυκλωμάτων έχει σχεδιάσει πολύπλοκες αρχιτεκτονικές με σκοπό να βελτιώσει την απόδοση και να μειώσει το κόστος. Τέτοιου είδους αρχιτεκτονικές περιλαμβάνουν ASICs (Application Specific Integrated Circuits), FPGAs (Field Programmable Gate Array) ακόμα και System-on-a-Chip (SOC) αποτελούμενα από ενσωματωμένους επεξεργαστές και μνήμες. Αυτές οι αρχιτεκτονικές έχουν μεγάλη επίδραση στον τρόπο που σχεδιάζονται οι διάφορες εφαρμογές, αφού επιτρέπουν σε ένα μόνο τσιπ να επεξεργάζεται μεγάλη ποσότητα πληροφορίας. Συγκεκριμένα, τα FPGAs έχουν βελτιστοποιήσει κατά πολύ το σχεδιασμό εφαρμογών καθώς είναι επαναπρογραμματιζόμενα, μειώνοντας έτσι το χρόνο προς πώληση και αυξάνοντας την προσαρμοστικότητα της σχεδίασης.

Η συνεχής πρόοδος της τεχνολογίας τα τελευταία χρόνια, σε συνδυασμό με το νόμο του Moore, συνέβαλαν στη μείωση του χάσματος μεταξύ των FPGAs και των ASICs όσον αφορά την απόδοσή τους. Επιπλέον το κόστος ανάπτυξης των FPGAs σε σχέση με αυτό για τα ASICs έχει μειωθεί. Έτσι, ο χρόνος προς πώληση και η ανάγκη να επιταχύνουμε τη διαδικασία σχεδίασης έχουν αυξήσει την ανάγκη για χρήση προγραμματιζόμενης λογικής.

Τα ASICs επομένως αντικαθίστανται από τα FPGAs σε πολλές εφαρμογές. Επιπλέον, τα τελευταία χρόνια προστίθενται στα FPGAs πολλές πολύπλοκες δομές, αυξάνοντας έτσι την απόδοσή τους. Σήμερα τα FPGAs αντικαθιστούν μνήμες ή ακόμα και μικροεπεξεργαστές καθώς αυτά είναι πλέον στοιχεία της μήτρας του FPGA. Ως αποτέλεσμα αυτού, η αναγκαιότητα των FPGAs αυξάνεται λόγω της προσαρμοστικότητάς τους, της υψηλής τους απόδοσης, του χαμηλού κόστους ανάπτυξης και της δυνατότητας επαναπρογραμματισμού τους.

Υπάρχουν πολλοί τύποι FPGA, ένας από τους οποίους χρησιμοποιεί μνήμη SRAM δίνοντάς μας τη δυνατότητα να τροποποιήσουμε εφαρμογές που έχουν ήδη εγκατασταθεί. Έτσι, τα FPGA με βάση SRAM είναι πολύτιμα σε εφαρμογές που χειριζόμαστε εξ' αποστάσεως, όπως διαστημικές, επιτρέποντας μας να κάνουμε αλλαγές σε αυτές με σκοπό να διορθώσουμε τυχόν σφάλματα, με αποτέλεσμα τη μείωση του κόστους της διαδικασίας αυτής.

Τα πλεονεκτήματα των FPGA με βάση SRAM σε τέτοιου είδους εφαρμογές και η αύξηση της πολυπλοκότητας της προγραμματίζόμενης λογικής με όλο και περισσότερες ενσωματωμένες μνήμες και μικροεπεξεργαστές, κάνουν αναγκαία την έρευνα για νέες τεχνικές άμβλυνσης SEU (SEU mitigation techniques). Σε αυτή την μεταπτυχιακή διατριβή θα ασχοληθούμε με την τεχνική τριπλού αρθρωτού πλεονασμού (Triple Modular Redundancy - TMR).

Σε πρώτη φάση παρουσιάζονται τα σφάλματα που μπορεί να προκύψουν σε διάφορες εφαρμογές, όπως οι διαστημικές. Οι όροι Single Event Upset (SEU) και Single Transient Effect (SET) εξηγούνται στο πρώτο κεφάλαιο, σφάλματα τα οποία καλούμαστε να εξαλείψουμε. Επιπλέον αναλύονται διάφορες τεχνικές ανοχής σφαλμάτων που έχουν προταθεί στη βιβλιογραφία [1]. Παρατίθενται τα πλεονεκτήματα και τα μειονεκτήματά τους καθώς επίσης και που υπερτερεί η κάθε μία όσον αφορά την επιφάνεια, το κόστος και την απόδοση συγκριτικά με τις υπόλοιπες.

Στο δεύτερο κεφάλαιο γίνεται μια λεπτομερής ανάλυση της τεχνικής TMR και πως προτείνεται η εφαρμογή της σε διάφορα κυκλώματα από τη Xilinx [2]. Έτσι βλέπουμε πως για να ενδυναμώσουμε σημαντικά ένα κύκλωμα, όπως μια CPU, ενάντια στα σφάλματα θα πρέπει η εφαρμογή της Triple Modular Redundancy να γίνει εσωτερικά. Εξηγείται αναλυτικότερα πως αυτή πρέπει να εφαρμοστεί σε κάθε επιμέρους μονάδα του κυκλώματος ανάλογα με το είδος λογικής του.

Στο τρίτο κεφάλαιο ακολουθεί μια αναλυτική παρουσίαση του PicoBlaze [3], ενός 8-bit επεξεργαστή, ιδανικό για να εφαρμόσουμε σε αυτόν την τεχνική TMR. Συγκεκριμένα, παρουσιάζονται οι λειτουργικές μονάδες που τον αποτελούν, το σύνολο εντολών που υποστηρίζει και τον τρόπο με τον οποίο μπορούμε να τον ενσωματώσουμε σε μια σχεδίαση χρησιμοποιώντας τη μονάδα KCPSM3 [4].

Στο τέταρτο κεφάλαιο έχουμε την υλοποίηση της νέας ενισχυμένης CPU σε μια συσκευή FPGA Spartan-3, XC3S200. Εδώ παρουσιάζονται όλες οι σχεδιαστικές αποφάσεις και τα βήματα που ακολουθήθηκαν για τη δημιουργία της. Επιπρόσθετα στο κεφάλαιο αυτό παρουσιάζεται ο συμβολομεταφραστής (assembler) KCPSM3, ο οποίος χρησιμοποιείται για τη μεταγλώττιση των προγραμμάτων που θα εκτελέσει η νέα CPU ώστε να επαληθεύσουμε τη σωστή της λειτουργία. Τέλος, μέσα από αυτά τα παραδείγματα και με εισαγωγή σφαλμάτων στη σχεδίασή μας, παρουσιάζουμε την ισοδυναμία μεταξύ των δύο CPU, της ενισχυμένης και της αρχικής, και τις συγκρίνουμε σε επίπεδο απόδοσης και πόρων συσκευής που καταλαμβάνουν. Εν κατακλείδι, παρατίθενται γενικά συμπεράσματα και προτάσεις για την εφαρμογή της τεχνικής TMR σε άλλες εφαρμογές.

1 Τεχνικές Ανοχής Σφαλμάτων

Η ανοχή σφαλμάτων σε συσκευές ημιαγωγών είναι ένα θέμα σημασίας από τότε που για πρώτη φορά συναντήθηκαν σφάλματα σε διαστημικές εφαρμογές. Το ενδιαφέρον για μελέτη τεχνικών ανοχής σφαλμάτων [1] έχει αυξηθεί ώστε τα ολοκληρωμένα κυκλώματα (ICs) να παραμένουν λειτουργικά σε τέτοια εχθρικά περιβάλλοντα.

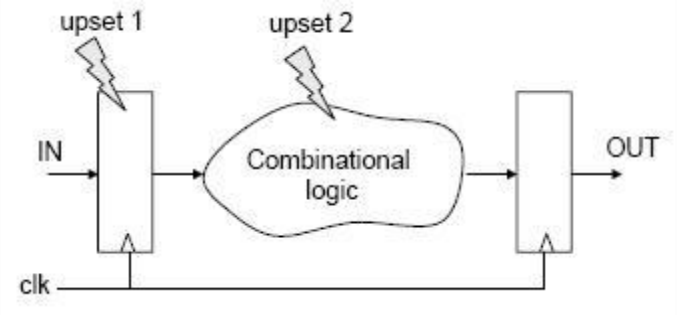
Τα φαινόμενα μονού γεγονότος (Single Event Effects - SEE), είναι η βασικότερη ανησυχία στις διαστημικές εφαρμογές καθώς οι πιθανές τους συνέπειες συμπεριλαμβάνουν απώλεια πληροφοριών και αποτυχία της λειτουργίας της εφαρμογής. Συμβαίνουν όταν φορτισμένα σωματίδια προσκρούουν στο ολοκληρωμένο κύκλωμα, μεταφέροντας αρκετή ενέργεια ώστε να προκληθεί σφάλμα στο σύστημα.

Τα SEE μπορούν να είναι καταστροφικά ή παροδικά, ανάλογα με το πόση ενέργεια βρίσκεται στα φορτισμένα σωματίδια και το σημείο όπου προσέκρουσαν στη συσκευή. Η κυριότερη συνέπεια των παροδικών SEE, που ονομάζονται διαταραχή μονού γεγονότος (Single Event Upset - SEU), είναι αντιστροφή των bits στα στοιχεία μνήμης. Η συχνότητα εμφάνισης SEU αυξάνεται συνεχώς κατά την πάροδο των χρόνων, αφού υπάρχουν όλο και πιο πολύπλοκες αρχιτεκτονικές, με περισσότερες ενσωματωμένες μνήμες. Έτσι, η ανάγκη να προστατεύσουμε τα ολοκληρωμένα κυκλώματα ενάντια στα σφάλματα γίνεται όλο και πιο απαραίτητη.

Βασιζόμενοι στον ορισμό της ανοχής σφαλμάτων, σκοπός μας είναι να διατηρήσουμε τη σωστή λειτουργία του ολοκληρωμένου κυκλώματος παρόλο την ύπαρξη σφαλμάτων.

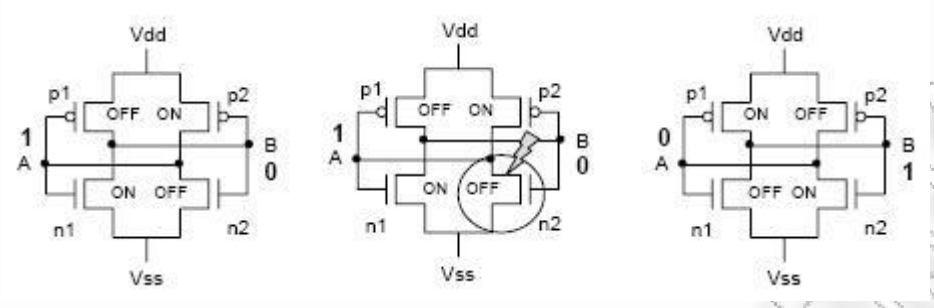
1.1 Επίδραση της ακτινοβολίας στα ολοκληρωμένα κυκλώματα

Ένα σωματίδιο μπορεί να επηρεάσει είτε την συνδυαστική είτε την ακολουθιακή λογική. Στην εικόνα 1.1 τα δεδομένα από τον πρώτο μανδαλωτή φεύγουν προς την συνδυαστική λογική στην ακμή του ρολογιού. Η έξοδος της συνδυαστικής λογικής φτάνει στο δεύτερο μανδαλωτή λίγο πριν την επόμενη ακμή ρολογιού. Σε αυτή την ακμή, οποιαδήποτε δεδομένα υπάρχουν στην είσοδό του αποθηκεύονται στο μανδαλωτή.



Εικόνα 1.1 : Upset σε συνδυαστική και ακολουθιακή λογική

Όταν ένα φορτισμένο σωματίδιο προσκρούσει σε έναν ευαίσθητο κόμβο κελιού μνήμης (memory cell), όπως στην υποδοχή (drain) ενός τρανζίστορ σε κατάσταση "off" παράγεται ένας "πρόσκαιρος παλμός" (transient current pulse) στην πύλη (gate) του αντίθετου τρανζίστορ. Αυτό μπορεί να δημιουργήσει μια αντιστροφή στην αποθηκευμένη τιμή, δηλαδή ένα bit flip στο κελί μνήμης. Τα κελιά μνήμης έχουν δύο καταστάσεις, μια για την τιμή "0" και μια για την τιμή "1". Σε κάθε κατάσταση, δύο τρανζίστορ είναι σε κατάσταση "on" και δύο σε κατάσταση"off". Ένα bit flip σε ένα στοιχείο μνήμης, συμβαίνει όταν ένα φορτισμένο σωματίδιο προκαλέσει την αντιστροφή της κατάστασης του τρανζίστορ στο κύκλωμα (εικόνα 1.2). Αυτή η επίδραση ονομάζεται διαταραχή μονού γεγονότος (Single Event Upset - SEU) και είναι ένας από τους βασικούς προβληματισμούς στα ψηφιακά κυκλώματα.



Εικόνα 1.2 : SEU σε ένα κελί SRAM μνήμης

Όταν ένα φορτισμένο σωματίδιο προσκρούσει στο μπλοκ συνδυαστικής λογικής, δημιουργεί και πάλι έναν πρόσκαιρο παλμό. Αυτό το φαινόμενο ονομάζεται μονό παροδικό φαινόμενο (Single Transient Effect - SET). Αν η λογική είναι αρκετά γρήγορη για να διαδώσει τον παλμό που προκλήθηκε, τότε το SET θα εμφανιστεί τελικά στην είσοδο του δεύτερου latch, όπου θα το εκλάβει ως έγκυρο σήμα. Αν το SET θα αποθηκευτεί ως πραγματικό δεδομένο εξαρτάται από την προσωρινή σχέση μεταξύ του χρόνου αφίξεως και την ανοδική ή καθοδική ακμή του ρολογιού.

1.2 Τεχνικές άμβλυνσης SEU

Η πρώτη τεχνική SEU, η οποία χρησιμοποιείται εδώ και πολλά χρόνια, είναι η θωράκιση (shielding). Η μέθοδος αυτή ελαττώνει στο ελάχιστο τη ροή σωματιδίων χωρίς να τα εξαλείψει τελείως. Ήταν επαρκής για πολλά χρόνια, αλλά λόγω της συνεχής εξέλιξης της τεχνολογίας, είναι απαραίτητες καινούργιες τεχνικές για την αποφυγή των επιδράσεων της ακτινοβολίας (radiation effects).

Τα τελευταία χρόνια, έχουν προταθεί πολλές τεχνικές άμβλυνσης SEU με σκοπό την αποφυγή σφαλμάτων στα ψηφιακά κυκλώματα, συμπεριλαμβανομένων και αυτών που υλοποιούνται σε προγραμματιζόμενη λογική. Μπορούν να κατηγοριοποιηθούν ως εξής :

- Τεχνικές βασισμένες στη διαδικασία κατασκευής (Fabrication process-based techniques), όπως:
    1. Epitaxial CMOS
    2. Προηγμένες διαδικασίες, όπως silicon-on-insulator (SOI)
- Τεχνικές βασισμένες στη σχεδίαση (Design-based techniques), όπως:
    1. Τεχνικές Ανίχνευσης (Detection techniques):
        i. Πλεονασμός υλικού (Hardware redundancy)
        ii. Πλεονασμός χρόνου (Time redundancy)
        iii. Κώδικες ανίχνευσης σφαλμάτων (Error Detection Coding - EDC)
        iv. Τεχνικές Αυτόνομου ελέγχου (Self-checker)
    2. Τεχνικές Άμβλυνσης (Mitigation techniques):
        i. Τριπλού Αρθρωτού Πλεονασμού (Triple Modular Redundancy - TMR)
        ii. Πολλαπλός πλεονασμός και εκλογή (Multiple redundancy with voting)
        iii. Κώδικες ανίχνευσης και διόρθωσης σφαλμάτων (Error detection and correction coding - EDAC)
        iv. Ενίσχυση σε επίπεδο κελιού μνήμης (Hardened memory cell level)
    3. Τεχνικές Ανάκτησης (Recovery Techniques), που εφαρμόζονται μόνο σε προγραμματιζόμενη λογική, όπως :
        i. Επαναρχικοποίηση (Reconfiguration)
        ii. Μερική Επαναρχικοποίηση (Partial configuration)
        iii. Επαναδρομολόγηση σχεδίασης (Rerouting design)

Οι τεχνικές που βασίζονται στη διαδικασία κατασκευής είναι γνωστές και ως τεχνολογικές τεχνικές (technological techniques). Μειώνουν σε αποδεκτό επίπεδο κάποιες από τις επιπτώσεις της ακτινοβολίας, όπως την ολική δόση ακτινοβολίας (Total Ionization Dose - TID) και το μονό γεγονός latch-up (single event latch-up - SEL), ωστόσο δεν εξαλείφουν τελείως τις επιδράσεις των διαταραχών (upsets), όπως τα SEUs και SET. Είναι ακριβή μέθοδος, επομένως δεν χρησιμοποιείται συχνά, ιδίως αν πρόκειται για μικρή παραγωγή.

Οι τεχνικές που βασίζονται στη σχεδίαση, είναι γνωστές και ως τεχνικές αρχιτεκτονικής (architectural techniques). Είναι κοινώς αποδεκτές μιας και μπορούν να χρησιμοποιηθούν σε διάφορα επίπεδα της σχεδίασης χωρίς να αλλάξουμε κάτι στη διαδικασία κατασκευής. Μπορούν να σχεδιαστούν έτσι ώστε να ανιχνεύουν μόνο την ύπαρξη σφαλμάτων στο σύστημα ή μπορούν να είναι πιο περίπλοκες ώστε να ανιχνεύουν και να διορθώνουν τα σφάλματα στο σύστημα κατά την ύπαρξη κάποιας διαταραχής. Όλες αυτές οι τεχνικές αποτελούνται από κάποιου είδους επανάληψης-πλεονασμού (redundancy), που δίδεται από κάποια επιπλέον στοιχεία (hardware redundancy) ή από επιπλέον χρόνο εκτέλεσης ή στιγμιότυπα δεδομένων (instants of data) (time redundancy).

Η Hardware redundancy βασίζεται στην επανάληψη της λογικής, και χαρακτηρίζεται από επιπλέον στοιχεία ή μονοπάτια που επιτρέπουν στη σχεδίαση να συνεχίσει να λειτουργεί ακόμα και αν κάποια μονοπάτια αποτυγχάνουν. Οι κώδικες ανίχνευσης και διόρθωσης σφαλμάτων (EDAC) μπορούν να θεωρηθούν και ως hardware redundancy μιας και δημιουργούν πλεονάζοντα bits για να εντοπίσουν και να διορθώσουν τις διαταραχές. Χρησιμοποιούνται όμως και στις τεχνικές time redundancy.

Οι τεχνικές ανάκτησης χρησιμοποιούνται κυρίως για στοιχεία προγραμματιζόμενης λογικής, όπως σε FPGA με βάση SRAM. Σκοπός είναι να ανακτήσουμε την αρχικά προγραμματισμένη πληροφορία μετά από μια διαταραχή. Τέτοιου είδους τεχνικές είναι οι επαναρχικοποίηση (τεχνική ανάγνωσης / επανεγγραφής - scrubbing), η μερική επαναρχικοποίηση και η επαναδρομολόγηση σχεδίασης. Μπορούν να εξαλείψουν ένα σφάλμα από τη μήτρα σε πολύ μικρό χρονικό διάστημα. Χρησιμοποιούνται συνήθως για να αποφύγουμε τη συσσώρευση σφαλμάτων.

Όπως καταλαβαίνουμε, το να βρούμε την καταλληλότερη τεχνική είναι μια πρόκληση όπου πρέπει να συνδυάσουμε χαμηλό κόστος, υψηλή απόδοση και υψηλή αξιοπιστία. Ένα επαρκές σύνολο τεχνικών θα πρέπει να χειρίζεται επιτυχώς τα SET που συμβαίνουν στη συνδυαστική λογική και τα SEU στα κελιά μνήμης. Με αυτόν τον τρόπο τα πρόσκαιρα σφάλματα δε θα αποθηκευτούν ποτέ στα κελιά μνήμης και τα bit flip δε θα συμβούν ποτέ ή θα διορθωθούν αμέσως.

Στην επόμενη ενότητα θα δώσουμε ιδιαίτερη έμφαση στις τεχνικές που βασίζονται στη σχεδίαση αφού αυτές είναι που χρησιμοποιούνται ευρέως.

1.3 Τεχνικές βασισμένες στη σχεδίαση

Οι τεχνικές time και hardware redundancy ποικίλουν, μπορούν να είναι απλή ανίχνευση σφαλμάτων ή εκλογή διαταραχών (upset voting) και διόρθωση. Μερικές φορές είναι απαραίτητο να ενημερώσουμε την εφαρμογή για την ύπαρξη σφάλματος με ένα Interrupt ενώ άλλες φορές είναι απαραίτητο να αποφύγουμε τελείως τα interrupts, εξασφαλίζοντας πλήρη αξιοπιστία. Στην ενότητα αυτή περιγράφονται αναλυτικότερα οι διάφορες τεχνικές που ανήκουν σε αυτή την κατηγορία.

1.3.1 Τεχνικές Εντοπισμού

Σε αυτή την κατηγορία ανήκουν οι τεχνικές time και hardware redundancy και οι αλγόριθμοι ανίχνευσης σφαλμάτων.

Οι τεχνικές που βασίζονται σε time redundancy χρησιμοποιούνται συνήθως για να εντοπίσουν ένα SET στη συνδυαστική λογική. Σκοπός είναι να εκμεταλλευτούμε τα χαρακτηριστικά του πρόσκαιρου παλμού που δημιουργείται από το σωματίδιο και να συγκρίνουμε τα σήματα εξόδου σε δύο διαφορετικές χρονικές στιγμές.

Στην περίπτωση της hardware redundancy, ο διπλασιασμός με σύγκριση (duplication with comparison - DWC) μπορεί να χρησιμοποιηθεί στη συνδυαστική και στην ακολουθιακή λογική για να διορθώσει τυχόν SET και SEU αντίστοιχα. Και στις δύο περιπτώσεις (time και hardware redundancy), είναι σημαντικό να λάβουμε υπόψη μας τη διάρκεια του SET.

Ένα άλλο παράδειγμα τεχνικής ανίχνευσης σφαλμάτων στην ακολουθιακή λογική είναι οι αλγόριθμοι ανίχνευσης σφαλμάτων (error-detecting codes) όπως αυτός που χρησιμοποιεί parity bits. Στην περίπτωση αυτή, το parity bit μιας ομάδος από bits υπολογίζεται και συγκρίνεται συνεχώς με ένα νέο parity bit που υπολογίζουμε. Στην περίπτωση που έχει συμβεί SEU μπορεί να διορθωθεί. Αυτή η μέθοδος χρησιμοποιείται ευρέως σε μνήμες. Ωστόσο, για να επιτύχουμε μεγαλύτερη αξιοπιστία, δεν είναι πάντα αρκετό να εντοπίσουμε απλά το σφάλμα στο σύστημα αλλά χρειάζεται να εξασφαλίσουμε τη σωστή του λειτουργία κατά την ύπαρξη του σφάλματος αυτού. Για αυτό το λόγο είναι σημαντικό να εμβαθύνουμε στις τεχνικές έμβλυνσης SEU που παρέχουν την απαιτούμενη αξιοπιστία.

1.3.2 Τεχνικές Άμβλυνσης

Στις τεχνικές αυτές υπάγονται η τεχνική TMR, αλγόριθμοι για τον εντοπισμό και τη διόρθωση σφαλμάτων και η ενίσχυση των κελιών μνήμης.

Υπάρχουν δύο τύποι πλήρους πλεονασμού (full redundancy), πλήρης πλεονασμός χρόνου (full time redundancy) και πλήρης πλεονασμός υλικού (full hardware redundancy). Η χρήση της full time redundancy στη συνδυαστική λογική επιτρέπει την επιλογή, μέσω ψηφοφορίας, της σωστής τιμής της εξόδου κατά την ύπαρξη SET. Το όνομα full redundancy προέρχεται από το n-modular redundancy, εδώ n=3 οπότε έχουμε την triple modular redundancy. Σε αυτή την περίπτωση η έξοδος της συνδυαστικής λογικής υπολογίζεται σε τρεις διαφορετικές χρονικές στιγμές, όπου η ακμή του ρολογιού για το δεύτερο δείγμα έχει χρόνο καθυστέρησης = d και για το τρίτο 2d. Ένας εκλογέας (voter) χρησιμοποιείται για την επιλογή της σωστής τιμής.

Στην περίπτωση της full hardware redundancy, γνωστή ως Triple Modular Redundancy - TMR, η λογική τριπλασιάζεται και στην έξοδο τοποθετούνται εκλογείς για να αναγνωρίσουν τη σωστή τιμή. Μια πρώτη προσέγγιση είναι ο τριπλασιασμός όλης της συσκευής. Ένας εκλογέας χρησιμοποιείται ως τέταρτο στοιχείο, ενώ είναι απαραίτητες επιπλέον συνδέσεις με αποτέλεσμα η τελική σχεδίαση να παρουσιάζει επιβάρυνση επιφάνειας. Αν παρουσιαστεί σφάλμα σε μια από τις τρεις συσκευές, ο εκλογέας θα επιλέξει τη σωστή τιμή. Προστατεύει εξίσου τη συνδυαστική και ακολουθιακή λογική από σφάλματα. Ωστόσο αν προκύψει σφάλμα στον εκλογέα ο TMR δεν είναι αποτελεσματικός και θα δώσει λάθος τιμή στην έξοδο. Ένα άλλο πρόβλημα αυτής της μεθόδου είναι η συσσώρευση σφαλμάτων, για αυτό και είναι απαραίτητος ένας επιπλέον μηχανισμός για να διορθώσει το σφάλμα σε κάθε συσκευή προτού συμβεί το επόμενο SEU.

Μια πιο αποτελεσματική προσέγγιση της τεχνικής TMR είναι να την υλοποιήσουμε για τα επιμέρους στοιχεία της σχεδίασης που θέλουμε και να χρησιμοποιήσουμε πολλαπλούς εκλογείς. Αυτός ο τρόπος δεν αποτρέπει τη συσσώρευση σφαλμάτων στην ακολουθιακή λογική και ο εκλογέας είναι ευαίσθητος σε σφάλματα. Για να μπορέσουμε να ανακτήσουμε τη σωστή τιμή θα χρησιμοποιήσουμε τρεις εκλογείς με ανάδραση. Τα σφάλματα θα διορθωθούν και η συσσώρευση σφαλμάτων θα αποφευχθεί. Παρόλο που αυτή η προσέγγιση της TMR παρουσιάζει μεγάλη επιβάρυνση επιφάνειας σε σύγκριση με την time redundancy, μιας και τριπλασιάζεται όλη η λογική, προστατεύει αποτελεσματικά τη λογική ενάντια σε SEU και SET και αποφεύγει τη συσσώρευση σφαλμάτων. Επιπλέον δεν έχει μεγάλη απώλεια απόδοσης, μόνο το χρόνο διάδοσης (propagation time) για τους εκλογείς.

Μια άλλη μέθοδος για να εξαλείψουμε τα SEU στη συνδυαστική λογική είναι αυτή που βασίζεται στον διπλασιασμό και τη χρήση στοιχείων CWSP (Code Word State Preserving). Εδώ δε χρειάζονται εκλογείς, ο διπλασιασμός μπορεί να αντικατασταθεί με time redundancy, που μειώνει σημαντικά την επιβάρυνση επιφάνειας. Βασικό κομμάτι αυτής της μεθόδου είναι το CWSP στάδιο, όπου αντικαθίστανται οι τελευταίες πύλες του κυκλώματος με μια συγκεκριμένη τοπολογία πυλών που είναι ικανή να ξεχωρίσει τη σωστή τιμή της συνδυαστικής λογικής κατά την ύπαρξη SET.

Μια ακόμα τεχνική που βασίζεται στη σχεδίαση, είναι η τεχνική EDAC, η οποία χρησιμοποιείται συνήθως σε μνήμες. Υπάρχουν πολλοί αλγόριθμοι που μπορούν να χρησιμοποιηθούν για την προστασία του κυκλώματος από SEU, όπως ο κώδικάς Hamming στην πιο απλή του μορφή. Ο κώδικας Hamming προτείνεται κυρίως για συστήματα με μικρές πιθανότητες πολλαπλών σφαλμάτων και μπορεί να προστατέψει δομές όπως καταχωρητές, αρχεία καταχωρητών και μνήμες. Αυξάνει η επιφάνεια αφού χρειάζεται επιπλέον κελιά αποθήκευσης, τα μπλοκ του κωδικοποιητή (encoder) και του αποκωδικοποιητή (decoder). Η καθυστέρηση των encoder και decoder προστίθεται στο κρίσιμο μονοπάτι. Το μειονέκτημά του όμως είναι πως δεν μπορεί να διορθώσει διπλές διαταραχές σε bits (double bit upsets). Υπάρχουν όμως άλλοι αλγόριθμοι ικανοί να αντιμετωπίσουν πολλαπλές

διαταραχές bit, όπως ο Reed-Solomon, ο οποίος χρησιμοποιείται για τη διόρθωση σφαλμάτων σε πολλές εφαρμογές όπως συσκευές αποθήκευσης και ασύρματες επικοινωνίες. Είναι αποτελεσματικός στην προστασία μνημών ενάντια σε πολλαπλά SEU.

Μια διαφορετική μέθοδος άμβλύνσης των SEU, είναι κελιά μνήμης που αποτελούνται από επιπλέον στοιχεία, τα οποία μπορεί να είναι ρεζίστορ ή τρανζίστορ, ικανά να ανακτήσουν την αποθηκευμένη τιμή αν ένα σφάλμα προκληθεί σε ένα από τα drain ενός τρανζίστορ σε κατάσταση "off". Αυτά τα κελιά ονομάζονται Hardened Memory Cells και μπορούν να αποφύγουν τη δημιουργία ενός SEU κατά τη σχεδίαση, σύμφωνα με τη ροή και το φορτίο του σωματιδίου. Η πρώτη προσέγγιση της μεθόδου ήταν κελιά μνήμης ανθεκτικά σε SEU, προστατευόμενα από ρεζίστορ. Αυτά παρέχουν υψηλή πυκνότητα σε πυρίτιο (high silicon density) αλλά μειονεκτούν γιατί είναι ευαίσθητα στη θερμότητα, ευπαθή σε χαμηλές θερμοκρασίες και χρειάζονται μια επιπλέον μάσκα κατά την κατασκευή για την πύλη του ρεζίστορ.

Τα κελιά μνήμης μπορούν να προστατευτούν επίσης με κατάλληλη ανάδραση για να ανακτήσουν δεδομένα όταν φθαρούν από φορτισμένα σωματίδια. Το βασικό πρόβλημα αυτής της προσέγγισης είναι τα επιπλέον τρανζίστορ που απαιτούνται για την ανάδραση και η επίδραση. Τα κυριότερα πλεονεκτήματα αυτής της προσέγγισης είναι η απουσία ευαισθησίας στις μεταβολές θερμοκρασίας, η ανεξαρτησία της παροχής τάσης και τεχνολογικής επεξεργασίας (technology process independence), και η καλή ανοσία σε SEU. Βασικό της μειονέκτημα είναι η επιβάρυνση επιφάνειας λόγω των επιπλέον τρανζίστορ. Παραδείγματα αυτής της μεθόδου είναι τα κελιά μνήμης Canaris. Η προσέγγιση αυτή μπορεί να χρησιμοποιηθεί για να εξισώσει τη συνδυαστική και ακολουθιακή λογική όταν τα κελιά μνήμης υλοποιούνται χρησιμοποιώντας συνδυαστικές πύλες με ανοσία σε SEU. Το συνδυαστικό κομμάτι του κυκλώματος μπορεί να ομαδοποιηθεί σε περίπλοκες λογικές συναρτήσεις με κάθε μια από αυτές να έχει δύο επιπλέον τρανζίστορ που χωρίζουν την έξοδό τους. Λόγω του διπλασιασμού του αριθμού εξόδων, το πλήθος των εσωτερικών συνδέσεων μπορεί να αυξηθεί σύμφωνα με την υλοποίηση της αρχιτεκτονικής. Το κύριο μειονέκτημα των κελιών μνήμης Canaris είναι ο μεγάλος χρόνος ανάκτησης μετά από ένα σφάλμα.

Μια άλλη προσέγγιση άμβλυνσης είναι η αποθήκευση δεδομένων σε δύο διαφορετικές τοποθεσίες στο κελί, με τέτοιο τρόπο ώστε το φθαρμένο κομμάτι να μπορεί να ανακτηθεί. Βασικά πλεονεκτήματα αυτής της μεθόδου είναι η θερμοκρασία, η ανεξαρτησία της παροχής τάσης και τεχνολογικής επεξεργασίας, η καλή ανοσία σε SEU, και η υψηλή απόδοση (χρόνος ανάγνωσης / εγγραφής). Παράδειγμα αυτής της μεθόδου είναι τα κελιά DICE.

Πίνακας 1.1 : Σύγκριση τεχνικών άμβλυνσης

Τεχνική Άμβλυνσης SEU	**Κελιά μνήμης ανθεκτικά σε SEU**	**Κώδικας Hamming**	**TMR**
Επιφάνεια	Συνήθως διπλασιάζει την επιφάνεια του κάθε κελιού. Εξαρτάται από τα τρανζίστορ.	Εξαρτάται από τον αριθμό των bits που προστατεύει. Έχει επιπλέον συνδυαστική και ακολουθιακή λογική.	Καταλαμβάνει κάτι παραπάνω από την τριπλή επιφάνεια της αρχικής σχεδίασης εξαιτίας των εκλογέων.
Απόδοση	Δεν επηρεάζεται αν τα επιπλέον τρανζίστορ ή ρεζίστορ (μονοπάτι καθυστέρησης) λειτουργούν μόνο όταν το κελί είναι σε αναμονή (hold).	Τα μπλοκ των encoder και decoder επηρεάζουν την απόδοση.	Δεν επηρεάζεται ιδιαίτερα. Η μόνη αιτία καθυστέρησης είναι οι εκλογείς.
Διόρθωση σφαλμάτων	Αποφεύγει το σφάλμα με μια καθυστέρηση στο βρόχο μνήμης (πλεονασμός / ανάκτηση)	Συνήθως διορθώνει μια διαταραχή ανά λέξη (one single upset per word), αλλά για να μπορέσει να ανανεώσει την αποθηκευμένη τιμή	Δεν διορθώνει τις διαταραχές. Αυτές συσσωρεύονται αν δεν υπάρχει επιπλέον λογική για ανανέωση.

Τεχνική Άμβλυνσης SEU	Κελιά μνήμης ανθεκτικά σε SEU	Κώδικας Hamming	TMR
		χρειάζεται ένα επιπλέον μονοπάτι (scrubbing)	
Πολλαπλές Διαταραχές	Αποτελεσματικό ως τρίτου βαθμού πολλαπλά σφάλματα καθώς κάθε κελί προστατεύει τον εαυτό του.	Μη αποτελεσματικό για πολλαπλές διαταραχές στην ίδια λέξη. Είναι όμως αποτελεσματικό για πολλαπλές διαταραχές σε διαφορετικά σημεία του κυκλώματος	Μπορεί να αντιμετωπίσει πολλαπλές διαταραχές σε διαφορετικά σημεία του κυκλώματος αλλά όχι στο ίδιο TMR σήμα.

1.3.3 Τεχνικές Ανάκτησης

Πολλές τεχνικές ανοχής σφαλμάτων, για FPGA με βάση, βασίζονται στην επαναδρομολόγηση (re-routing) και την εναλλακτική αρχικοποίηση. Οι τεχνικές αυτές αποφεύγουν τις διαταραχές στα χρησιμοποιούμενα CLBs. Το πρώτο πρόβλημα που συναντάμε όταν διορθώνουμε σφάλματα με επαναρχικοποίηση, και χωρίς τη χρήση κάποιου πλεονασμού, είναι ποια μέθοδο θα χρησιμοποιήσουμε για να βρούμε τα σφάλματα στη μήτρα.

Μια προσέγγιση ανάκτησης μετά από σφάλμα (fault recovery) που βασίζεται σε επαναρχικοποίηση και επαναδρομολόγηση, είναι να χωρίσουμε τη φυσική σχεδίαση σε τμήματα. Το σημαντικότερο σημείο αυτής της προσέγγισης είναι να επαναρχικοποιήσουμε μερικώς (partially reconfiguring) το FPGA με μια εναλλακτική αρχικοποίηση ως απάντηση στο σφάλμα. Αν η νέα αρχικοποίηση υλοποιεί την ίδια λειτουργία με την αρχική, ενώ παράλληλα αποφεύγει το προβληματικό μπλοκ, τότε το σύστημα μπορεί να ξαναρχίσει. Η πρόκληση εδώ είναι να βρούμε μια αποτελεσματική εναλλακτική αρχικοποίηση και να έχουμε μικρό χρόνο εντοπισμού σφαλμάτων.

Η τεχνική Partial reconfiguration είναι ένας αξιόπιστος τρόπος για να ανιχνεύσουμε και να διορθώσουμε σφάλματα στα ενσωματωμένα στο τσιπ δεδομένα αρχικοποίησης. Το πρόβλημα εδώ είναι η συνοχή στη μνήμη κατά την μερική επαναρχικοποίηση. Μιας και τα LUTs μπορούν να υλοποιήσουν μονάδες μνήμης για εφαρμογές του χρήστη, έχουμε προβλήματα συνοχής της μνήμης, όπως αλλαγή των δεδομένων κατά την διαδικασία ανάκτησης δεδομένων.

Στο σημείο αυτό έχουμε παρουσιάσει τις πιο σημαντικές τεχνικές ανοχής σφαλμάτων, και τις συγκρίναμε σε σχέση με την επιφάνεια, την απόδοση και την αποτελεσματικότητά τους, Στη συνέχεια θα γίνει μια λεπτομερής περιγραφή του τρόπου εφαρμογής της τεχνικής TMR, που είναι και το αντικείμενο αυτής της μεταπτυχιακής διατριβής.

2 Triple Modular Redundancy

Στο κεφάλαιο αυτό θα ασχοληθούμε με την τεχνική Triple Modular Redundancy, γνωστή ως TMR. Θα αναλύσουμε τον τρόπο με τον οποίο πρέπει να εφαρμοστεί στα διάφορα κυκλώματα, όπως προτείνεται από τη Xilinx [2].

Σε αυτό το σημείο είναι βασικό να πούμε πως τα στατικά σφάλματα (static errors) στη μνήμη αρχικοποίησης (configuration memory) δεν είναι απαραίτητα και λειτουργικά σφάλματα. Παρόλα αυτά, τα εύπλαστα – προσωρινά σφάλματα (soft error) είναι εξ' ορισμού λειτουργικά σφάλματα και η συσσώρευση σφαλμάτων στη μνήμη αρχικοποίησης θα οδηγήσει σε αποτυχία της λειτουργίας (functional failure). Οι τεχνικές άμβλυνσης, μπορούν να ενδυναμώσουν τη σχεδίαση ενάντια στα SEU και τα SET. Παράλληλα, τα SEU διορθώνονται έτσι ώστε τα στατικά σφάλματα να μη συσσωρεύονται και τα εύπλαστα σφάλματα να μη διαδίδονται.

Στα ASICs, τα τριπλά πλεονάζοντα κυκλώματα προστατεύουν μόνο τα flip-flops από SEU, καθώς τα λογικά μονοπάτια μεταξύ αυτών είναι συνδεδεμένες, μη αρχικοποιημένες πύλες. Για να αποφύγουμε και τα SET χρειαζόμαστε πλήρη αρθρωτό πλεονασμό (full module redundancy).

2.1 Πλήρης Αρθρωτός Πλεονασμός

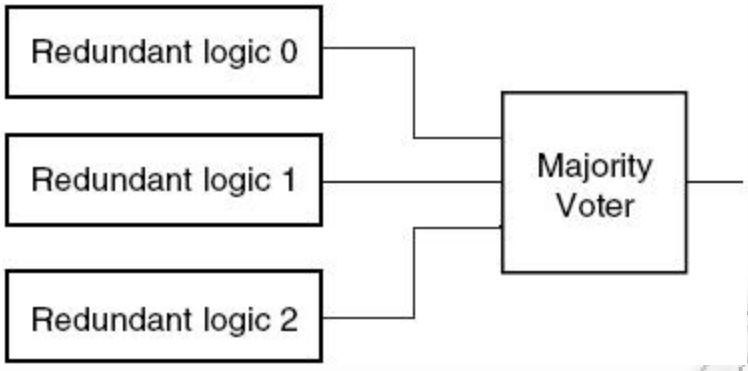
Η τεχνική full module redundancy είναι η κατάλληλη υλοποίηση της TMR στα FPGAs αφού όλα τα λογικά μονοπάτια και όχι μόνο τα flip-flops είναι επιρρεπή σε SEU. Τρία πλήρη αντίγραφα της βασικής σχεδίασης θα υλοποιηθούν για να προστατέψουν τη λειτουργικότητα του κυκλώματος από SEU και SET. Η μέθοδος για τη δημιουργία TMR σε FPGA αρχιτεκτονικής Virtex παρέχει επιπλέον πλεονεκτήματα, πλήρη διατήρηση των δεδομένων και αυτόνομη ανάκτηση.

Η σωστή εφαρμογή του TMR εξαρτάται από τον τύπο δομής του κυκλώματος στο οποίο θέλουμε να εξομαλύνουμε τα σφάλματα. Έχουμε τέσσερις κατηγορίες:

- **Λογική Διεκπεραίωσης (Throughput) :** είναι μια λογική μονάδα οποιουδήποτε μεγέθους και λειτουργικότητας, σύγχρονη ή ασύγχρονη, όπου όλα τα λογικά μονοπάτια κατευθύνονται από τις εισόδους προς τις εξόδους της, χωρίς να δημιουργούνται λογικοί βρόχοι (loops). Δηλαδή, μονάδες στις οποίες οι λογικές καταστάσεις δεν εξαρτώνται από προηγούμενες καταστάσεις, παραδείγματος χάριν ένας αθροιστής.
- **Μηχανές καταστάσεων :** δομή όπου κάποια έξοδός της (σε οποιοδήποτε στάδιο του module) τροφοδοτείται σε κάποιο προηγούμενο στάδιο μέσα στο module δημιουργώντας λογικό βρόχο, όπως accumulators, state-machines και state-sequencers όπου η εκάστοτε κατάσταση των εσωτερικών καταχωρητών εξαρτάται από την προηγούμενη κατάστασή τους.
- **Λογική I/O :** είσοδοι και έξοδοι της FPGA σχεδίασης. Συγκεκριμένα χαμηλής τάσης TTL (LVTTL) και χαμηλής τάσης CMOS (LVCMOS) I/O. Δεν χρησιμοποιούνται διπλής κατεύθυνσης ή κυκλώματα διαφορικών εισόδων - εξόδων (IOB).
- **Ιδιαίτερες μονάδες :** Block RAMs και DLLs.

2.2 Τριπλός Πλεονασμός και Εκλογή

Η βασική ιδέα για να ενδυναμώσουμε ένα ευαίσθητο κύκλωμα ενάντια στα SEU, είναι να υλοποιήσουμε τρία αντίγραφα του ίδιου κυκλώματος και στην έξοδο του τριπλασιασμένου αυτού κυκλώματος να εκτελέσουμε μια bit προς bit "ψηφοφορία πλειοψηφίας" (majority voting). Το εκάστοτε λογικό κύκλωμα μπορεί να είναι ένα απλό flip-flop ή ένα ολοκληρωμένο λογικό κύκλωμα.



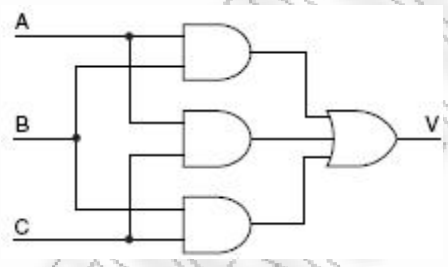
Εικόνα 2.1 : Τριπλός πλεονασμός με Ψηφοφορία πλειοψηφίας

Η λειτουργία του εκλογέα πλειοψηφίας είναι να δώσει στην έξοδό του τη λογική τιμή "0" ή "1" που ανταποκρίνεται σε τουλάχιστον δύο από τις τρεις εισόδους του. Αν θεωρήσουμε ότι οι είσοδοι του εκλογέα είναι οι A, B και C και η έξοδός του η V, τότε η Boolean εξίσωσή που τον χαρακτηρίζει είναι η εξής : $V = AB + AC + BC$.

Πίνακας 2.1 : Πίνακας αληθείας του Εκλογέα Πλειοψηφίας

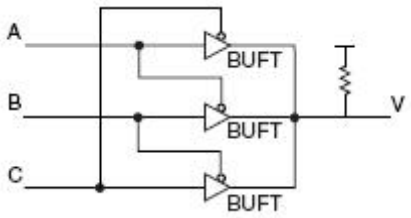
A	B	C	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Η υλοποίηση ενός εκλογές με πύλες απεικονίζεται στην εικόνα 2.2 :



Εικόνα 2.2 : Κύκλωμα εκλογέα πλειοψηφίας

Σε σχεδιάσεις που υπάρχει περιορισμός σε λογικούς πόρους, οι εκλογείς μπορούν να υλοποιηθούν χρησιμοποιώντας τους εσωτερικούς 3-state buffers του Virtex αντί για Look-Up Tables (LUTs) που χρησιμοποιούνται για την υλοποίηση όλων των εξισώσεων Boolean. Η κατασκευή του με αυτό τον τρόπο φαίνεται στην εικόνα 2.3 :



Εικόνα 2.3 : Κύκλωμα majority voter με BUFT

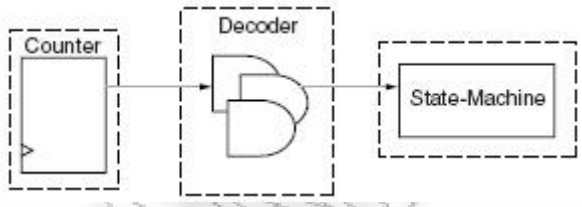
Για σχεδιάσεις FPGA που δεν περιορίζονται από τους διαθέσιμους λογικούς πόρους αλλά απαιτούν τη γρηγορότερη δυνατόν χρονική επίδοση, η υλοποίηση των εκλογέων πλειοψηφίας στα LUTs μπορεί να μας δώσει μια γρηγορότερη υλοποίηση κυκλώματος. Τα CLB LUTs χρησιμοποιούνται γενικά στην υλοποίηση όλων των συνδυαστικών λογικών στη σχεδίαση. Επομένως, η χρήση LUTs στην υλοποίηση των εκλογέων επιτρέπει στη λογική του εκλογέα να απλοποιηθεί σε γενική συνδυαστική λογική με μικρότερη καθυστέρηση διάδοσης.

2.3 Υλοποίηση TMR για λογική διεκπεραίωσης

Για την υλοποίηση του TMR σε μια δομή λογικής διεκπεραίωσης απλά δημιουργούμε τρία αντίγραφα της αρχικής μονάδας. Δημιουργούνται επίσης τρία αντίγραφα κάθε εισόδου και κάθε εξόδου αυτού. Αυτό ισχύει για κάθε λογική δομή καθώς η μεθοδολογία του TMR απαιτεί για όλα τα λογικά μονοπάτια να έχουν τριπλό πλεονασμό σε όλη τη σχεδίαση, ώστε να αποφευχθεί ένα σημείο μοναδικής αστοχίας (single-point-of-failure).

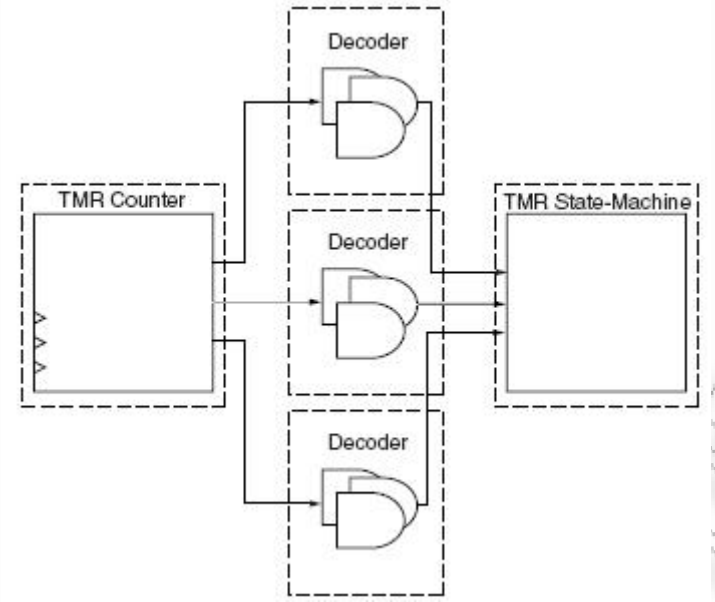
Η λογική διεκπεραίωσης δε αποτελεί συνήθως μια ολόκληρη μονάδα από μόνη της. Μια μονάδα περιέχει τις περισσότερες φορές διάφορες μηχανές καταστάσεων (state-machines) με μονοπάτια συνδυαστικής λογικής ανάμεσά τους. Έτσι η υλοποίηση TMR μπορεί να απλοποιηθεί χωρίζοντας τη μονάδα σε δομές μηχανών καταστάσεων και συνδυαστική λογική. Τις μηχανές καταστάσεων μπορούμε να τις αντιμετωπίσουμε ως μεμονωμένες μονάδες υλοποιώντας την TMR σε κάθε μια από αυτές, και αντιγράφουμε απλά τη συνδυαστική λογική που συνδέει την προηγούμενη μηχανή καταστάσεων είτε στην επόμενη είτε σε I/O λογική.

Στην εικόνα 2.4 φαίνεται ένα τυχαίο κύκλωμα το οποίο αποτελείται από έναν μετρητή, του οποίου η έξοδος αφού αποκωδικοποιηθεί γίνεται είσοδος σε μια μηχανή καταστάσεων. Τα διακεκομμένα κουτιά δείχνουν πώς χωρίζεται η μονάδα σε ξεχωριστές μονάδες σύμφωνα με το είδος λογικής τους.



Εικόνα 2.4 : Τυχαίο λογικό μονοπάτι

Η TMR έκδοση αυτής της μονάδας φαίνεται στην εικόνα 2.5. Η υλοποίηση της TMR για τις μηχανές καταστάσεων δομές δε φαίνεται σε αυτό το σημείο ενώ τα συνδυαστικά μονοπάτια έχουν τριπλασιαστεί. Σε αυτά δεν είναι απαραίτητοι εκλογείς καθώς η "ψηφοφορία" θα γίνει εσωτερικά στις TMR μηχανές καταστάσεων.

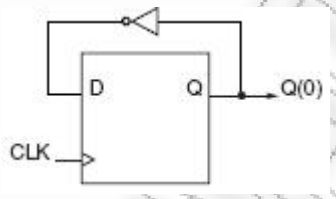


Εικόνα 2.5 : TMR έκδοση του τυχαίου λογικού μονοπατιού

Τα λογικά μονοπάτια μεταξύ των δύο μηχανών καταστάσεων δεν εξαρτώνται από προηγούμενες καταστάσεις (δεν είναι state-dependable). Οποιοδήποτε εύπλαστο σφάλμα στα λογικά μονοπάτια διαδίδεται χωρίς να "εγκλωβίζεται" σε λογικούς βρόχους. Ο μοναδικός σκοπός των τριπλασιασμένων μονοπατιών είναι να μεταφέρουν την τριπλασιασμένη έξοδο της προηγούμενης μονάδας στις τριπλασιασμένες εισόδους της επόμενης χωρίς να δημιουργήσουν σημεία μοναδικής αστοχίας. Μιας και τα τρία αντίγραφα δε χρειάζεται να περάσουν από εκλογέα, δεν υπάρχει ανάγκη για σύνδεση μεταξύ των τριών αντιγράφων.

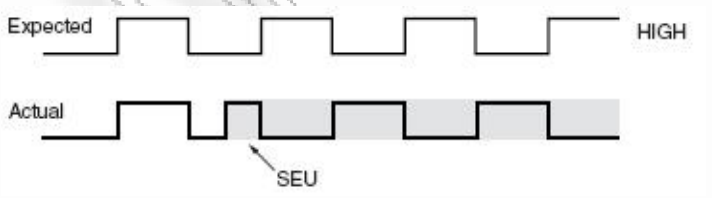
2.4 Υλοποίηση TMR για μηχανές καταστάσεων

Καθώς η λογική μηχανών καταστάσεων είναι εξαρτώνται εξ' ορισμού από προηγούμενες καταστάσεις τους, είναι απαραίτητο η ψηφοφορία να γίνεται εσωτερικά παρά εξωτερικά. Ένα πολύ απλό αλλά παράλληλα ιδανικό παράδειγμα είναι αυτό του 1-bit μετρητή (εικόνα 2.6).



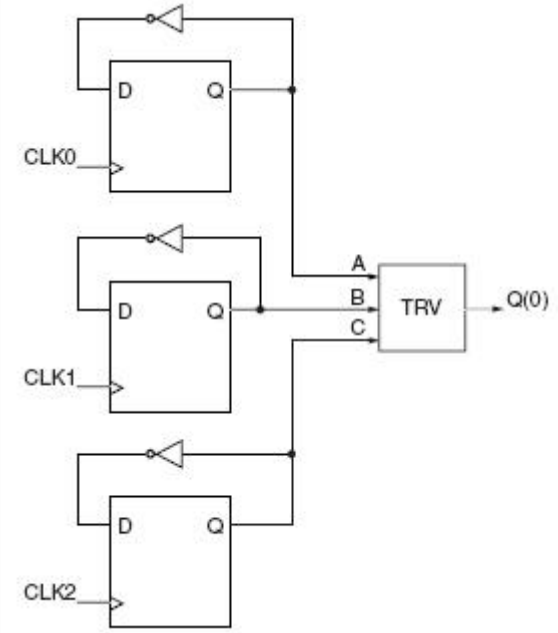
Εικόνα 2.6 : 1-Bit Μετρητής

Ο 1-bit μετρητής θα φορτώσει την αντίθετη τιμή από αυτή της προηγούμενης κατάστασής του σε κάθε ανοδική ακμή του ρολογιού. Καθώς όμως η έξοδός του είναι είτε "1" είτε "0" είναι πιθανό να υπάρχουν και άλλα κυκλώματα που βασίζονται στον απλό συγχρονισμό του. Η εικόνα 2.7 δείχνει τα αποτελέσματα που μπορεί να έχει ένα SEU σε αυτόν τον συγχρονισμό.



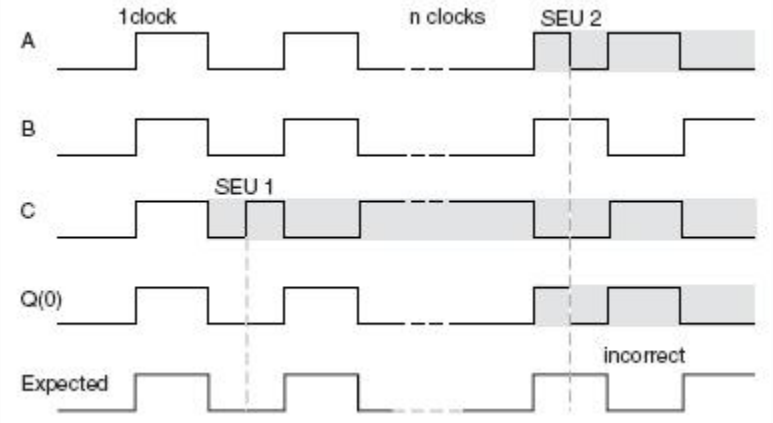
Εικόνα 2.7 : Έξοδος του 1-Bit Μετρητή με SEU

Ο μετρητής τώρα έχει κλειδώσει σε μια λανθασμένη κατάσταση επειδή είναι εκτός συχνότητας / συγχρονισμού και θα παραμείνει έτσι έως ότου γίνει reset. Τριπλασιάζοντας το κύκλωμα και βάζοντας έναν εκλογέα στην έξοδο, όπως φαίνεται στην εικόνα 2.8, η ψηφισθέντα έξοδος θα εξαλείψει το σφάλμα στο συγκεκριμένο αντίγραφο.



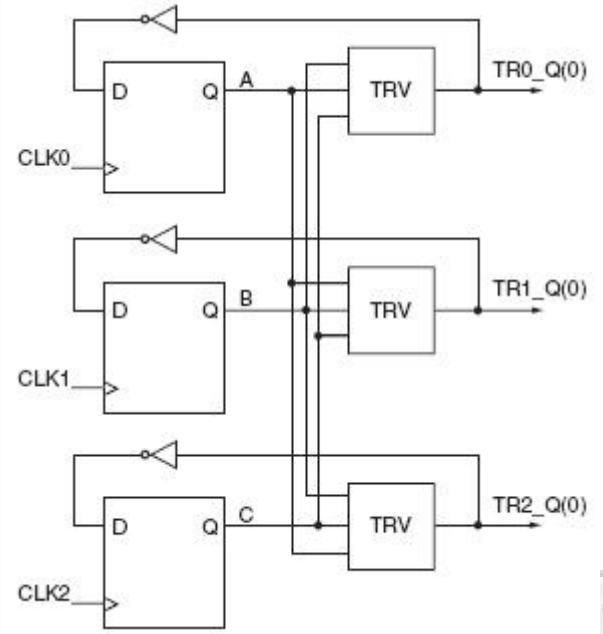
Εικόνα 2.8 : TMR έκδοση του 1-Bit Μετρητή

Αυτό το κύκλωμα δίνει σωστή τιμή στην έξοδό του, παρόλη την ύπαρξη ενός SEU. Ωστόσο, ένα αντίγραφο παραμένει εκτός συχνότητας / συγχρονισμού από τα άλλα. Αν άλλο ένα αντίγραφο επηρεαστεί από ένα SEU τότε η ψηφισθέντα έξοδος γίνεται και αυτή μόνιμα λανθασμένη, εικόνα 2.9. Έτσι, στην περίπτωση που δύο από τα τρία αντίγραφα έχουν σφάλμα, το κύκλωμα θα δίνει μόνιμα λάθος έξοδο και ένα εξωτερικό reset είναι απαραίτητο για να επανέλθει.



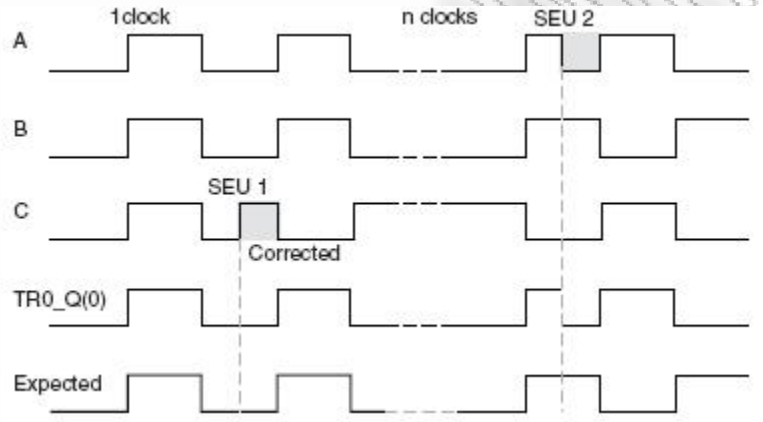
Εικόνα 2.9 : Συμπεριφορά του TMR Μετρητή με έναν εκλογέα, όπου προκύπτουν διαδοχικά SEU

Για να μπορέσει να επανέλθει αυτόνομα το κύκλωμα χωρίς εξωτερικό reset, πρέπει να ενσωματώσουμε την εκλεγόμενη τιμή στο λογικό μονοπάτι ανάδρασης. Συγκεκριμένα στο παράδειγμα αυτό θα πρέπει να χρησιμοποιήσουμε τριπλά πλεονάζοντες εκλογείς (triple redundant voters) σε κάθε μονοπάτι ανάδρασης (εικόνα 2.10).



Εικόνα 2.10 : TMR εκδοχή του 1-Bit Counter με τρεις εκλογείς

Όπως βλέπουμε και στην εικόνα 2.11, τα εύπλαστα σφάλματα φιλτράρονται σε κάθε κύκλο ρολογιού, επιτρέποντας στο κύκλωμα να επανέλθει προτού υπάρξει άλλο SEU.



Εικόνα 2.11 : Συμπεριφορά του TMR μετρητή με τρεις εκλογείς, όπου προκύπτουν διαδοχικά SEU

Επομένως, για τη δημιουργία της TMR εκδοχής μιας μηχανής καταστάσεων πρέπει να τριπλασιάσουμε όλα τα κυκλώματα και να εισάγουμε έναν εκλογέα πλειοψηφίας σε κάθε βρόχο ή μονοπάτι ανάδρασης. Η χρήση τριών εκλογέων πλειοψηφίας αποκλείει τα σημεία μοναδικής αστοχίας και παρέχει εξόδους τριπλής λογικής που συνδέονται στις τριπλασιασμένες εισόδους της επόμενης μονάδας.

Μιας και στα αντίγραφα απαιτείται επικοινωνία μεταξύ των εκλογέων, η TMR εκδοχή μιας μηχανής καταστάσεων επιτυγχάνεται καλύτερα αν απλοποιήσουμε το κύκλωμα εξαιρώντας την λογική διεκπεραίωσης από την προς υλοποίηση μονάδα.

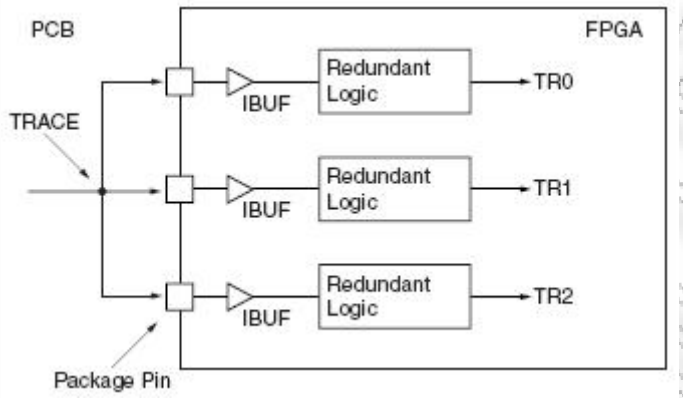
Τελικά, τα τριπλασιασμένα μονοπάτια πρέπει να "ενωθούν" ώστε να σχηματίσουν ένα μονοπάτι που δεν δημιουργεί σημεία μοναδικής αστοχίας. Αυτό επιτυγχάνεται στην TMR εκδοχή της I/O λογικής.

2.5 Υλοποίηση TMR για λογική I/O

2.5.1 TMR Είσοδοι

Πρωταρχικός σκοπός της χρήσης της τεχνικής TMR, είναι να εξαλείψουμε όλα τα σημεία μοναδικής αστοχίας από τη σχεδίαση. Αυτό ξεκινά με τις εισόδους του FPGA. Αν μια μόνο είσοδος είναι συνδεδεμένη σε όλα τα αντίγραφα μέσα στο FPGA, τότε ένα σφάλμα στην είσοδο θα προκαλέσει την επανάληψη του σφάλματος σε όλα τα αντίγραφα, με αποτέλεσμα το σφάλμα να μην εξαλειφθεί.

Έτσι, κάθε αντίγραφο της σχεδίασης που χρησιμοποιεί εισόδους του FPGA θα πρέπει να έχει το δικό του σύνολο εισόδων (εικόνα 2.12) έτσι ώστε εάν υπάρξει σφάλμα σε μια είσοδο, να επηρεαστεί μόνο ένα αντίγραφο.



Εικόνα 2.12 : Τριπλές πλεονάζουσες είσοδοι FGPA

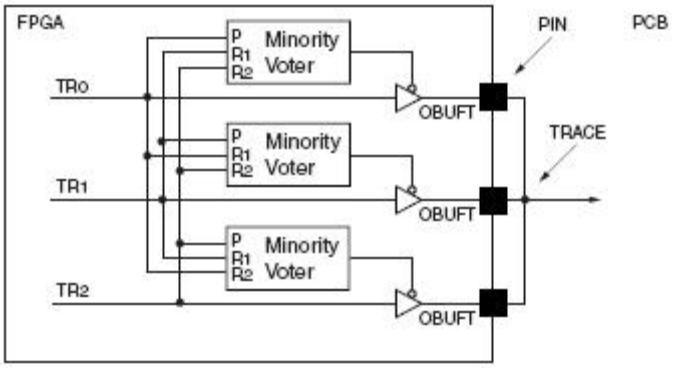
Στην περίπτωση όμως που η βασική σχεδίαση απαιτεί περισσότερες I/O από το 1/3 των διαθέσιμων I/O της συσκευής, τότε θα πρέπει είτε να μην τριπλασιάσουμε τις I/O είτε να χωρίσουμε τη σχεδίαση σε πολλές συσκευές.

2.5.2 TMR Έξοδοι

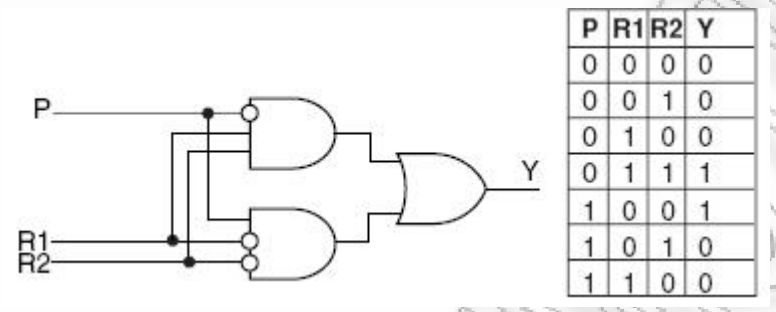
Οι έξοδοι είναι το κλειδί στη διαδικασία υλοποίησης της τεχνικής TMR. Μιας και η full triple redundancy αντιγράφει κάθε λογικό μονοπάτι τρεις φορές, πρέπει να υπάρχει τελικά μια μέθοδος όπου θα φέρνει αυτά τα τρία μονοπάτια σε ένα μοναδικό μονοπάτι που δεν θα δημιουργεί σημεία μοναδικής αστοχίας. Αυτό επιτυγχάνεται με τις TMR εξόδους.

Μια TMR έξοδος δημιουργείται χρησιμοποιώντας τα πρωταρχικά στοιχεία (primitives) OBUFT της βιβλιοθήκης. Κάθε τριπλασιασμένο μονοπάτι που εξέρχεται του FPGA σε μια έξοδο, το κάνει μέσω ενός OBUFT. Το “enable” (Tpin) κάθε OBUFT ελέγχεται από ένα κύκλωμα εκλογέα μειοψηφίας (minority voter). Ο εκλογέας μειοψηφίας υποδεικνύει αν το βασικό μονοπάτι (primary path) συμφωνεί με κάποιο από τα άλλα δύο. Αν αυτό συμφωνεί με έστω ένα από τα δύο αντίγραφα θεωρείται μέρος της πλειοψηφίας. Αν διαφωνεί και με τα δύο είναι μειοψηφία.

Αν το βασικό μονοπάτι είναι μέρος της πλειοψηφίας, τότε ο εκλογέας μειοψηφίας θα ενεργοποιήσει τον αντίστοιχο OBUFT (ενεργά χαμηλό) επιτρέποντας στα δεδομένα του βασικού μονοπατιού να οδηγηθούν μέσω του OBUFT στο Pad-Pin. Αν όχι, τότε ο OBUFT απενεργοποιείται θέτοντας την έξοδό του σε υψηλή εμπέδηση, επιτρέποντας στα τριπλασιασμένα μονοπάτια να οδηγήσουν τα σωστά δεδομένα.



Εικόνα 2.13 : TMR έξοδοι του FPGA που περνούν από εκλογείς μειοψηφίας



P	R1	R2	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0

Εικόνα 2.14 : Κύκλωμα εκλογέα μειοψηφίας και ο πίνακας αληθείας του

Εξωτερικά του FPGA, οι τρεις έξοδοι συνδέονται πάνω στην πλακέτα. Αυτή η δομή δεν δημιουργεί επίμαχες καταστάσεις εφόσον μόνο τα μονοπάτια που συμφωνούν μεταξύ τους οδηγούνται ενεργά. Αυτή η τεχνική έχει το επιπλέον πλεονέκτημα να διπλασιάζει ή / και να τριπλασιάζει τις υπάρχουσες δυνατότητες του sink και source της εξόδου όσον αφορά τα υπόλοιπα στοιχεία της πλακέτας που είναι συνδεδεμένα σε αυτή τη σχεδίαση. Το πρωταρχικό πλεονέκτημα όμως αυτής της μεθόδου είναι πως δε χρειάζονται επιπλέον εξωτερικές συσκευές για να ολοκληρώσουν την τριπλή πλεονάζουσα ψηφοφορία (triple redundant voting), όπως στην περίπτωση που θα χρησιμοποιούσαμε τριπλά πλεονάζοντα FPGAs αντί για εσωτερικό πλεονασμό μέσα σε ένα FPGA.

2.5.3 Συνδέσεις μεταξύ FPGA

Για τα σήματα εξόδου από ένα FPGA σε άλλο, η δομή των εκλογέων μπορεί να παραβλεφθεί και τρεις ξεχωριστές πλεονάζουσες έξοδοι μπορούν να κατευθυνθούν στο γειτονικό FPGA. Πλεονέκτημα αυτού είναι πως δεν υπάρχει περιορισμός στη χρήση IOB καταχωρητών εξόδου, συνεπώς έχουμε καλύτερη clock-to-output επίδοση. Ένα άλλο πλεονέκτημα είναι πως μπορούν να χρησιμοποιηθούν άλλα I/O στάνταρ όπως GTL ή LVDS για υψηλής ταχύτητας επικοινωνία από τσιπ σε τσιπ.

2.6 Υλοποίηση TMR για ιδιαίτερες μονάδες

Παρόλο που η πλειοψηφία των λογικών σχεδιασμών μπορεί να υλοποιηθεί με Look-Up Tables, flip flops και δρομολόγηση, υπάρχουν και άλλα ειδικά χαρακτηριστικά που επιτρέπουν υλοποιήσεις αποτελεσματικότερες και πιο αποδοτικές. Αυτά τα χαρακτηριστικά συμπεριλαμβάνουν block Ram, Lut Ram, καταχωρητές ολίσθησης, αριθμητικά, και DLL ρολογιού. Καθώς οι μέθοδοι άμβλυνσης σφαλμάτων SEU είναι υπό ανάπτυξη και δοκιμή, τα παρακάτω είναι προτάσεις για κάποια από αυτά τα χαρακτηριστικά.

2.6.1 Block RAM

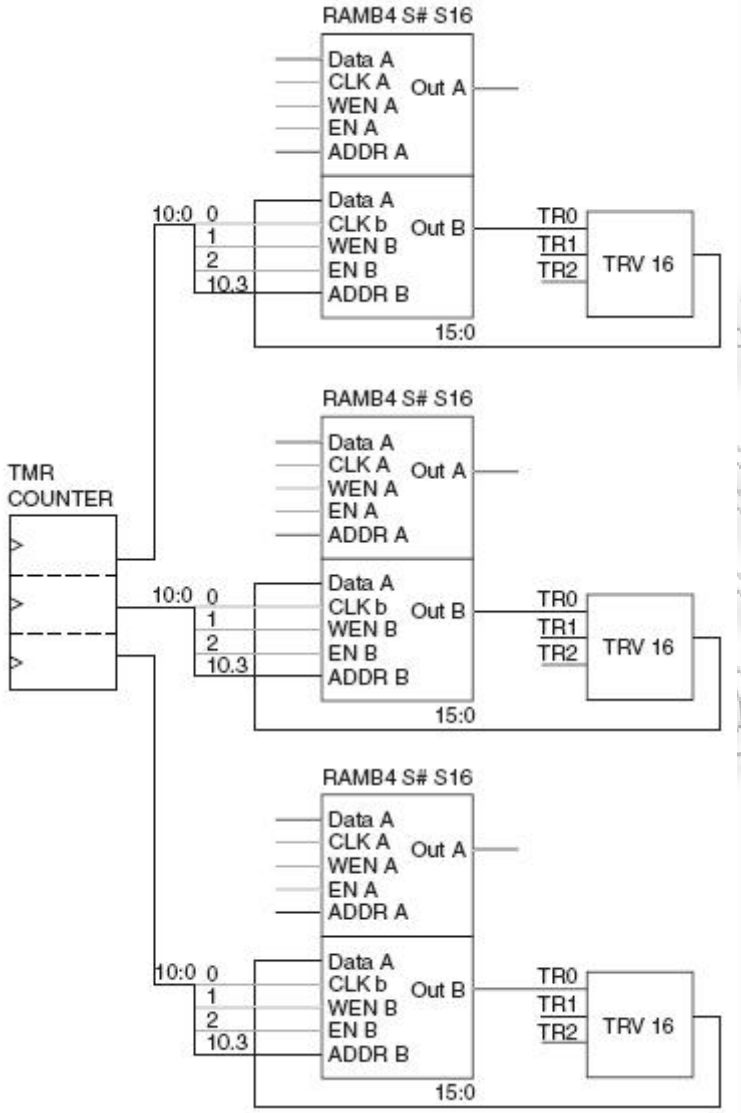
Οι Virtex Block RAMs είναι μπλοκ στατικής μνήμης, μεγέθους 4K bits το καθένα, διπλής πόρτας (true dual port) και πλήρως σύγχρονες (fully synchronous). Τα δεδομένα που εμπεριέχονται στην μνήμη μπορούν να προσπελαστούν μέσω της πόρτας αρχικοποίησης (SelectMAP), αλλά αυτή δεν είναι μια προσέγγιση που προτείνεται για μεθόδους EDAC.

Όταν το περιεχόμενο της block RAM προσπελαύνεται μέσω της πόρτας αρχικοποίησης, δεν επιτρέπεται η πρόσβαση στη block RAM μέσω της λογικής του χρήστη με αποτέλεσμα να διακόπτεται η καθορισμένη λειτουργία της. Οποιαδήποτε EDAC μέθοδος για τα περιεχόμενα της block RAM πρέπει να ενσωματωθεί στη σχεδίαση του χρήστη.

Υπάρχουν τρεις διαφορετικές μέθοδοι εφαρμογής της τεχνικής TMR στις block RAMs. Η πρώτη μέθοδος, ο Απλός Πλεονασμός (Simple Redundancy), δεν χρησιμοποιεί αλγορίθμους EDAC αλλά τριπλές πλεονάζουσες block RAM και εκλογείς πλειοψηφίας στις εξόδους. Αυτό είναι αρκετό για μια εφαρμογή όπου πιθανώς θα γραφτούν νέα δεδομένα σε όλες τις διευθύνσεις της μνήμης μέσα στο χρόνο που οι διαταραχές (upsets) είναι αναμενόμενες στις αντίστοιχες διευθύνσεις των τριπλασιασμένων block. Η μέθοδος αυτή βασίζεται στον στατιστικό ρυθμό σφαλμάτων (statistical upset rate), και δεν είναι τόσο ασφαλής και σίγουρη. Επιτρέπει όμως μέγιστη χρηστικότητα χαρακτηριστικών (max feature usability) αφού δεν έχουμε επιπλέον επιβάρυνση για να ανανεώσουμε τα μπλοκ μνήμης.

Ένας πιο αξιόπιστος τρόπος είναι να ανανεώνουμε συνεχώς τα περιεχόμενα της μνήμης, μέθοδος Πλεονασμός και Ανανέωση (Redundancy and Refresh). Η μια από τις δύο πόρτες μπορεί να χρησιμοποιηθεί για ανίχνευση και διόρθωση σφαλμάτων, EDAC, που έχει ως αποτέλεσμα η μνήμη να χρησιμοποιείται ως μνήμη μονής πόρτας (single port) από την υπόλοιπη λογική.

Για την ανανέωση της μνήμης μπορούμε να χρησιμοποιήσουμε έναν μετρητή όπου θα αυξάνει τη διεύθυνση κάθε τέσσερις κύκλους ρολογιού (εικόνα 2.15). Το Bit[0] της εξόδου του μετρητή οδηγεί το CLK της πόρτας B. Τα Bit[1] και Bit[2] οδηγούν τα WEN και EN αντίστοιχα. Για κάθε διεύθυνση τα δεδομένα ψηφίζονται και η πλειοψηφούσα ψήφος γράφεται στα κελιά.



Εικόνα 2.15 : TMR Block RAM με ανανέωση

Στο παράδειγμα αυτό, το εύρος δεδομένων (data width) της πόρτας Β τίθεται στη μεγαλύτερη δυνατή τιμή που μπορεί να πάρει, δηλαδή 16. Αυτό μειώνει το εύρος διευθύνσεων (address width) και συνεπώς και το μέγεθος του μετρητή, στη μικρότερη δυνατή τιμή, 8. Ωστόσο το εύρος δεδομένων της πόρτας Α μπορεί να τεθεί ανεξάρτητα από την πόρτα Β και να χρησιμοποιηθεί στην εφαρμογή.

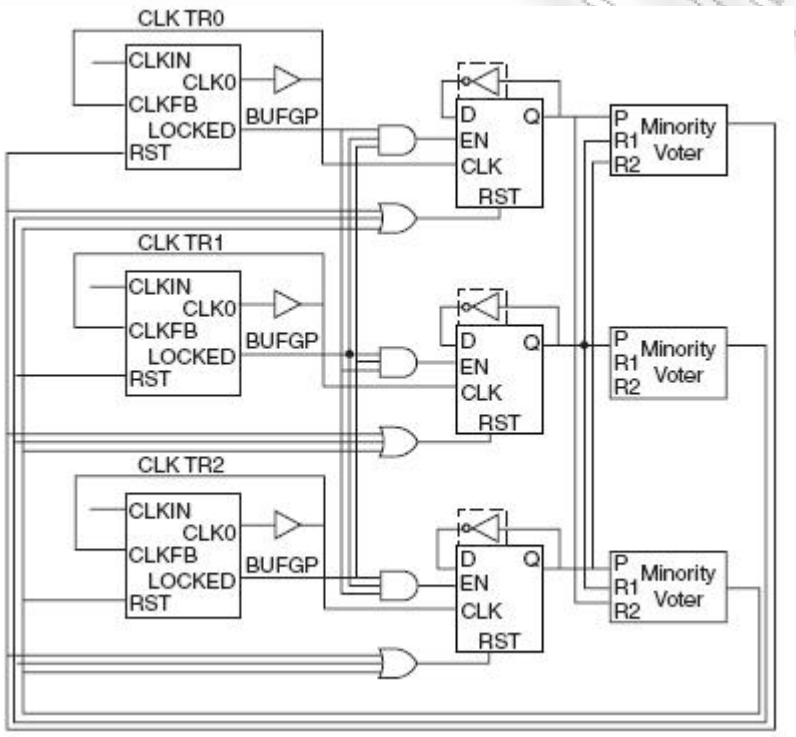
Οι έξοδοι της πόρτας Α δε χρειάζεται να περάσουν από εκλογέα πλειοψηφίας καθώς είναι ήδη τριπλασιαζσμένες. Η ψηφοφορία γίνεται στην επόμενη μηχανή κατάστασης ή I/O στάδιο. Το εύρος δεδομένων της πόρτας Α είναι ανεξάρτητο από αυτό της πόρτας Β.

Η τρίτη μέθοδος, Κωδικοποίηση Δεδομένων (Data Encryption), είναι μια μέθοδος όπου τα δεδομένα κωδικοποιούνται προτού γραφτούν στη μνήμη και αποκωδικοποιούνται όταν διαβάζονται από αυτή (μετά από ανάγνωση της μνήμης). Αλγόριθμοι EDAC, όπως ο Hamming, χρησιμοποιούνται σε αυτή τη μέθοδο. Ο αλγόριθμος κωδικοποίησης επιτρέπει στα σωστά δεδομένα να ξεχωρίσουν από τα μερικώς φθαρμένα.

2.6.2 Διαχείριση ρολογιού

Η αρχιτεκτονική Virtex έχει τέσσερις buffers ρολογιού και τέσσερα DLLs για την υλοποίηση ρολογιού στη σχεδίαση. Αυτό όμως περιορίζει κάπως μια TMR σχεδίαση αφού επιτρέπει μόνο ένα TMR πεδίο ρολογιού. Ωστόσο, στη δρομολόγηση μπορεί να χρησιμοποιηθεί οποιαδήποτε I/O ως δευτερεύουσα είσοδος ρολογιού, επιτρέποντας πολλαπλά TMR πεδία ρολογιού.

- **Buffer ρολογιού :** Οι σχεδιάσεις που χρησιμοποιούν μόνο ένα σύστημα ρολογιού μπορούν να χρησιμοποιήσουν τα πρωταρχικά στοιχεία buffer ρολογιού (BUFGP) για τη δημιουργία ενός μόνο TMR πεδίου ρολογιού. Στην ουσία είναι τρία πεδία ρολογιού που οδηγούνται εξωτερικά από το ίδιο ρολόι. Κάθε ένας από τους τρεις BUFGP (global clock buffers) συνδέεται σ' ένα ολοκληρωμένο αντίγραφο της σχεδίασης που μπορεί να υπάρχει σε αυτό το πεδίο.
- **DLL ρολογιού :** Χρησιμοποιούνται σε συνδυασμό με τους BUFGP για να επανασυγχρονίσουν το σήμα ρολογιού στο δικό του μονοπάτι απόκλισης (path skew) ή ως εξωτερική αναφορά για τη μείωση των clock-to-output καθυστερήσεων. Όταν υπάρξει σφάλμα στο DLL πρέπει να κάνουμε reset για να επανασυγχρονιστεί, με χρήση ενός κυκλώματος εντοπισμού SEU (εικόνα 2.16). Το σήμα "LOCKED", που δείχνει την κατάστασή του υπό φυσιολογική λειτουργία , δεν είναι αξιόπιστο.



Εικόνα 2.16 : TMR DLL ρολογιού με αυτόνομο έλεγχο

Τρεις 1-bit μετρητές χρησιμοποιούνται για να αποδείξουν πως όλα τα ρολόγια λειτουργούν και είναι συγχρονισμένα. Κάθε μετρητής είναι χρονισμένος από μια από τις πλεονάζοντες εξόδους ρολογιού. Το Enable στους μετρητές δεν είναι ενεργοποιημένο μέχρι και τα τρία DLLs να συγχρονιστούν και να έχουν τις εξόδους LOCKED ενεργοποιημένες. Όταν μια από τις τρεις εξόδους δε συμφωνεί με την πλειοψηφία, ο σχετικός εκλογέας μειοψηφίας κάνει reset το ανάλογο DLL, καθώς επίσης και τους τρεις καταχωρητές. Οι τρεις καταχωρητές δε θα ξεκινήσουν να χρονίζουν έως ότου και τα τρία DLLs είναι " κλειδωμένα".

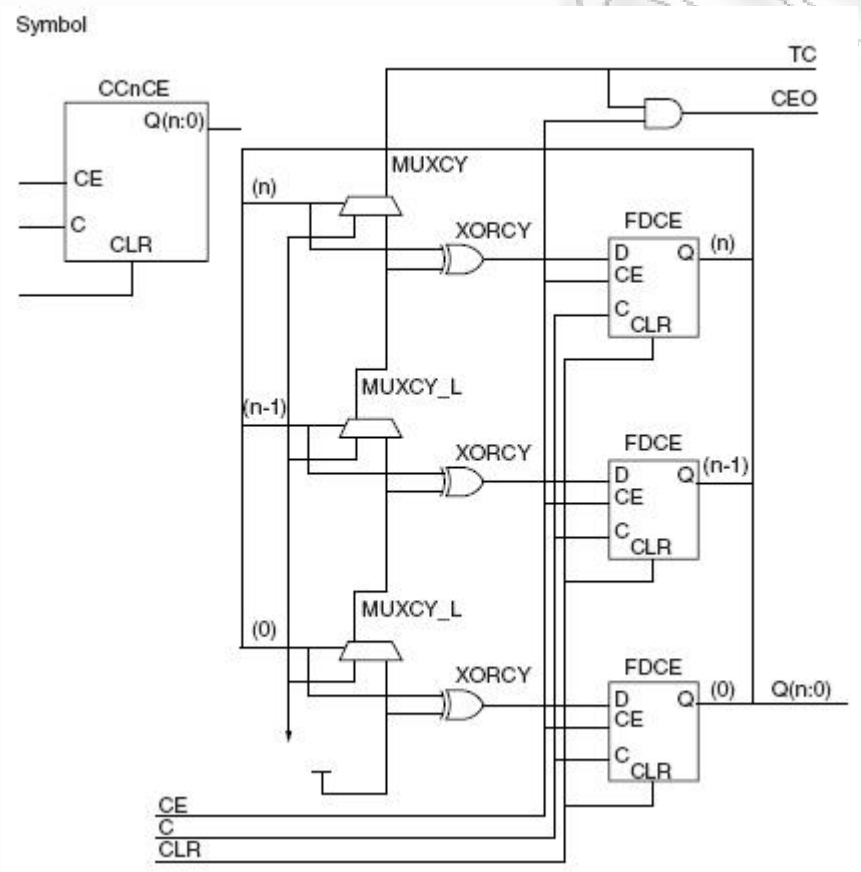
Κάθε DLL απαιτεί μια είσοδο ρολογιού στον ακροδέκτη CLK. Αυτή η είσοδος μπορεί να προέλθει μόνο από ένα dedicated clock input pad (GCLK) μέσω ενός buffer ρολογιού εισόδου (IBUFG). Οι έξοδοι ρολογιού είναι οι CLK_TR0, CLK_TR1 και CLK_TR2. Αντιπροσωπεύουν τα εσωτερικά TMR πεδία ρολογιού. Τα DLL έχουν άλλες

ταυτόχρονες εξόδους που παρέχουν μετατόπιση φάσης 90, 180 και 270, 2x πολλαπλασιασμός ρολογιού, και μια έξοδο διαίρεσης ρολογιού που καθορίζεται από το χρήστη.

2.6.3 Αριθμητικές αλυσίδες κρατουμένου

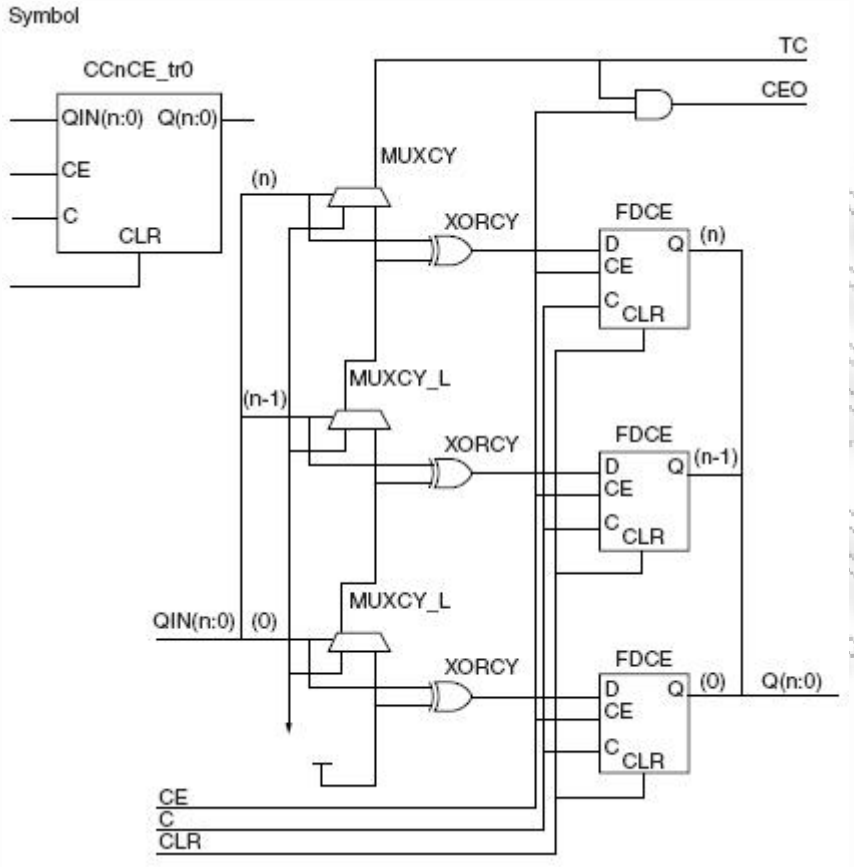
Τα αριθμητικά, όπως οι μετρητές και οι αθροιστές, υλοποιούνται καλύτερα χρησιμοποιώντας τις αλυσίδες κρατουμένου (carry-chain) που βρίσκονται στα CLBs. Συνήθως δε δημιουργούμε τέτοιες δομές σε πρωταρχικό επίπεδο αλλά μπορούμε να αρχικοποιήσουμε μονάδες της βιβλιοθήκης που χρησιμοποιούν αυτά τα χαρακτηριστικά ή να καταλήξουμε σε αυτά κατά τη σύνθεση της HDL σχεδίασης. Ωστόσο, ούτε η Xininx βιβλιοθήκη ούτε και οι βιβλιοθήκες σύνθεσης λαμβάνουν υπ' όψιν τους τις μεθόδους TMR.

Όσον αφορά τη σχηματική σχεδίαση, αντιγράφουμε μια μονάδα της Xilinx από την EDA βιβλιοθήκη σχηματικών και την τροποποιούμε ώστε να δημιουργήσουμε την TMR έκδοσή του. Η υλοποίηση ενός βασικού μετρητή με αλυσίδα κρατουμένου φαίνεται στην εικόνα 2.17. Το πρώτο στάδιο (bottom) αρχικοποιεί την αλυσίδα κρατουμένου. Το τελικό στάδιο (top) χρησιμοποιεί MUXCY αντί για MUXCY_L, έτσι ώστε το σήμα carry-out να εξαχθεί από το CLB και να χρησιμοποιηθεί ως έξοδος terminal count (TC). Οποιοσδήποτε αριθμός όμοιων σταδίων μπορεί να παρεμβληθεί ενδιάμεσα ώστε να δημιουργηθεί ένας μετρητής οποιουδήποτε μεγέθους (μόνος περιορισμός είναι ο αριθμός των CLBs / column για το εκάστοτε μέλος της οικογένειας FPGA).



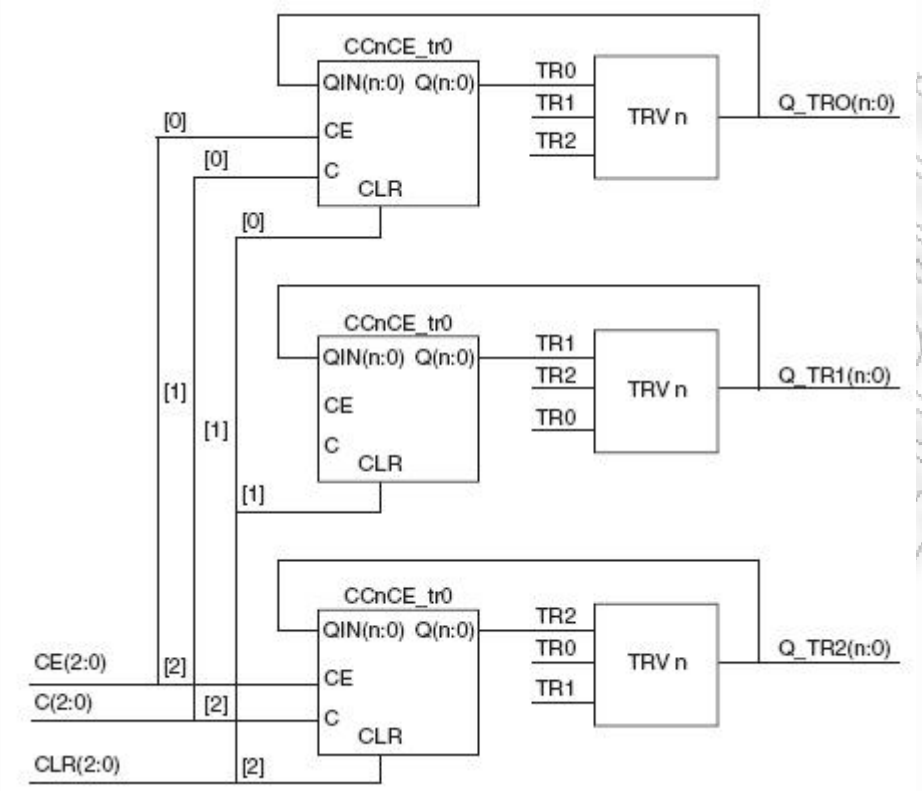
Εικόνα 2.17 : Μετρητής υλοποιημένος με λογική αλυσίδας κρατουμένου

Είναι εμφανές ο βρόχος που υπάρχει στο παραπάνω κύκλωμα. Για να δημιουργήσουμε την TMR έκδοσή του, θα πρέπει να σπάσουμε τον εσωτερικό βρόχο ώστε να εισάγουμε τους εκλογείς πλειοψηφίας (εικόνα 18).



Εικόνα 2.18 : TMR Μετρητής (υλοποιημένος με λογική αλυσίδας κρατουμένου)

Όπως φαίνεται, το μονοπάτι ανάδρασης έχει περιοριστεί και μια νέα είσοδος έχει προστεθεί (QIN(n:0)). Μπορούμε να χρησιμοποιήσουμε τρεις από αυτούς τους μετρητές ώστε να δημιουργήσουμε έναν TMR μετρητή (εικόνα 2.19).



Εικόνα 2.19 : TMR Μετρητής : CcnCE_TMR

Τα clock enable (CE), clock (C), και clear (CLR) είναι τριπλασιασμένα και ξεχωριστά για κάθε αντίγραφο. Ωστόσο, οι τριπλασιασμένες έξοδοι είναι η κάθε μια, μια ψήφος πλειοψηφίας (majority vote) των τριών μετρητών, παρέχοντας η καθεμία ένα ξεχωριστό μονοπάτι ανάδρασης στους μετρητές. Επομένως, αν ένας μετρητής αποτύχει λόγω ενός SEU, διορθώνει τον εαυτό του στον πρώτο κύκλο ρολογιού μετά τη διόρθωση του SEU (υποθέτοντας ότι το πιο πιθανό σημείο για ένα SEU είναι στη μνήμη αρχικοποίησης. SEU σε ένα από τα flip flops διορθώνεται άμεσα με τη χρήση του κυκλώματος TMR).

Όταν η σχεδίασή μας είναι σε VHDL, τότε για παράδειγμα μια συνηθισμένη συνάρτηση μετρητή μοιάζει με αυτή της εικόνας 2.20. Παρόλο που μπορούμε να χρησιμοποιήσουμε το ίδιο στυλ και για τον μετρητή της εικόνας 2.19, αφήνει άλυτα τα θέματα που αφορούν τη χρήση Vcc και GND. Για την υλοποίηση δομών αλυσίδας κρατουμένου είναι προτιμότερη η μέθοδος αρχικοποίησης μιας μονάδας από τη βιβλιοθήκη XQVR-SYN.

```
Counter: process(CLK, CLR)
Begin
        If (RST='1') Then
                Q <= null; (others => 0);
        Elsif CLK'event and CLK='1' then
                if (CE='1') then
                        Q <= Q + 1;
                endif;
        End if;
End Counter;
```

Εικόνα 2.20 : Συνάρτηση μετρητή σε VHDL

2.6.4 Κατανεμημένες RAM και καταχωρητές ολίσθησης υλοποιημένοι με LUTs

Η χρήση LUTs για την κατασκευή κατανεμημένων (distributed) RAM ή καταχωρητών ολίσθησης δεν προτείνεται, καθώς με τη χρήση αυτών τα δεδομένα αποθηκεύονται δυναμικά και χρειάζονται κελιά της μνήμης αρχικοποίησης. Αυτό είναι πρόβλημα σε εφαρμογές που χρησιμοποιούν Readback και / ή partial reconfiguration.

Τα LUTs μπορούν να χρησιμοποιηθούν σε μια σχεδίαση ως μικρά μπλοκ των στοιχείων κατανεμημένων RAM (για παράδειγμα RAMS16x1) ή δυναμικά διευθυνσιοδοτούμενοι καταχωρητές ολίσθησης. Όταν ένα LUT χρησιμοποιείται για τέτοιου τύπου λειτουργίες, τα δεδομένα του χρήστη αποθηκεύονται δυναμικά και διαχειρίζονται στα κελιά της μνήμης αρχικοποίησης. Αυτό δημιουργεί πρόβλημα σε μια εφαρμογή που σκοπεύει να χρησιμοποιήσει Readback και / ή partial reconfiguration, γι' αυτό είναι συνιστώμενο οι χρήστες να μη χρησιμοποιούν LUTs με αυτόν τον τρόπο καθώς μπορεί να προκληθεί καταστροφή των δεδομένων στην μνήμη αρχικοποίησης. Η χρήση Block RAM μνημών συνίσταται για όλες τις λειτουργίες RAM και flip flops για καταχωρητές ολίσθησης.

2.7 Vcc και GND

2.7.1 Μόνιμα σφάλματα

Τα εργαλεία PAR (Place And Route) υλοποιούν τα Vcc και GND με τέτοιο τρόπο ώστε να επιτυγχάνεται μέγιστη αξιοποίηση των πηγών της συσκευής. Αυτό γίνεται με την αξιοποίηση κυκλωμάτων διατήρησης (keeper) που βρίσκονται στα pin εισόδου των CLBs και IOBs.

Τα κυκλώματα διατήρησης βρίσκονται σε σειρά με τα κανάλια δρομολόγησης (routing channel) και τους ακροδέκτες εισόδου των logic block. Όταν το κανάλι δρομολόγησης φέρει ένα ενεργό σήμα, τότε το κύκλωμα διατήρησης μένει αδρανές. Όταν όμως το κανάλι δρομολόγησης είναι αχρησιμοποίητο, το κύκλωμα διατήρησης διατηρεί την τελευταία του τιμή, η οποία καθορίστηκε όταν η συσκευή ενεργοποιήθηκε αρχικά ή επαναρχικοποιήθηκε ενεργοποιώντας την είσοδο PROG του FPGA.

Όταν ένα λογικό στοιχείο όπως flip-flop στο λογικό μπλοκ (CLB ή IOB), απαιτεί μια λογική σταθερά, Vcc ή GND, η λογική αυτή σταθερά μπορεί να παρθεί από το κύκλωμα διατήρησης ενός αχρησιμοποίητου ακροδέκτη στο λογικό μπλοκ. Η πολικότητά του επιλέγεται με προγραμματιζόμενη αντιστροφή μέσα στο λογικό μπλοκ.

Ένα SEU μπορεί να αλλάξει ή να τροποποιήσει την κατάσταση του κυκλώματος διατήρησης, άμεσα με ιονισμό ή έμμεσα συνδέοντας ένα ενεργό κανάλι δρομολόγησης στην είσοδό του. Το αποτέλεσμα είναι διαταραχή της λειτουργίας, η οποία δε μπορεί να ανιχνευτεί με readback ή να διορθωθεί με partial reconfiguration. Αυτό είναι ένα μόνιμο σφάλμα (persistent error) και μπορεί μόνο να διορθωθεί με πλήρη επαναρχικοποίηση του FPGA.

Αφαιρώντας τις λειτουργικές εξαρτήσεις (functional dependencies) από τα Vcc και GND ελαχιστοποιούμε την ευαισθησία σε αυτά τα σφάλματα.

2.7.2 Vcc και GND

Τα Vcc και GND υπάρχουν στη Xilinx βιβλιοθήκη ως πρωταρχικά στοιχεία για όλες τις οικογένειες. Δεν πρέπει να χρησιμοποιούνται σε σχεδιάσεις που έχουν να κάνουν με την άμβλυνση SEU. Κάποιες φορές μπορούμε να προκαλέσουμε τη χρήση τους άθελά μας, όταν θέτουμε μια σταθερή τιμή σ' ένα σήμα ή μια πόρτα, χωρίς να καθορίσουμε σαφή χρήση των εισόδων ενός πρωταρχικού στοιχείου, ή συμπεραίνοντας αριθμητικά που εν τέλει υλοποιούνται με λογική αλυσίδας κρατουμένου.

Το πρωταρχικό CLB flip flop αρχιτεκτονικής Virtex έχει Clock Enable (CE) καθώς επίσης εισόδους INIT και REV (ένας καταχωρητής IOB έχει CE και INIT αλλά όχι REV). Οι είσοδοι INIT και REV χρησιμοποιούνται για την υλοποίηση σύγχρονων και / ή ασύγχρονων λειτουργιών SET και RESET για τον καταχωρητή. Οι είσοδοι CE και REV πρέπει να οδηγηθούν ώστε να λειτουργήσει ο καταχωρητής, ακόμα και αν αυτή η σύνδεση δεν έχει καθοριστεί στη σχεδίαση.

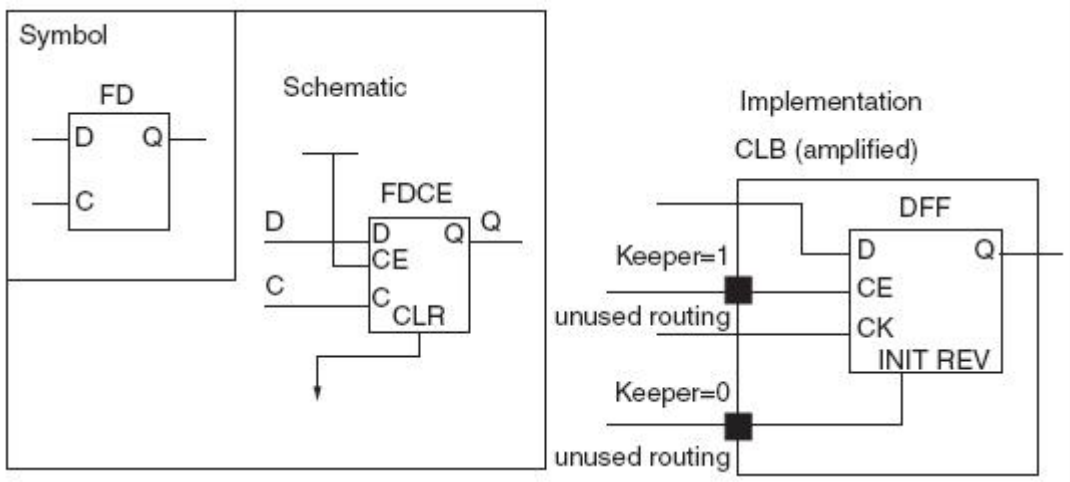
Για παράδειγμα, αν το σχηματικό αρχικοποιεί ένα FD (D flip flop), έχουμε στην ουσία αρχικοποιήσει μια μονάδα της βιβλιοθήκης που υλοποιεί ένα FDCE στο οποίο ο ακροδέκτης CE είναι καρφωμένος στο Vcc και ο ακροδέκτης CLR στο GND (το CLR τελικά γίνεται INIT).

Θα πρέπει να είμαστε πολύ προσεκτικοί και να εξετάζουμε την πρωταρχική υλοποίηση όλων των μονάδων της βιβλιοθήκης που πιθανώς περιέχουν καταχωρητές, προτού τις χρησιμοποιήσουμε. Ακόμα και αν η μονάδα παρέχει ακροδέκτες clock enable και reset στο υψηλότερο επίπεδο σχεδίασης (top level), η πρωταρχική υλοποίηση μπορεί να είναι διαφορετική από την αναμενόμενη.

Ομοίως, αν περιγράψουμε μια σύγχρονη διαδικασία σε VHDL χωρίς να καθορίσουμε λειτουργίες clock-enable ή αρχικοποίησης, τότε τα εργαλεία σύνθεσης υλοποιούν αυτή την λειτουργία χρησιμοποιώντας πρωταρχικά στοιχεία (primitives) και καρφώνοντας όλους τους αχρησιμοποίητους ακροδέκτες στις σωστές λογικές σταθερές, δημιουργώντας τα Vcc και GND. Το VHDL παράδειγμα της εικόνας 2.21 δημιουργεί ένα τέτοιο πρόβλημα όπως φαίνεται και στην εικόνα 2.22.

```
register: process (CLK)
Begin
        If CLK'event and CLK='1' then
                Q <= D;
        End if;
End register;
```

Εικόνα 2.21 : Παράδειγμα σε VHDL



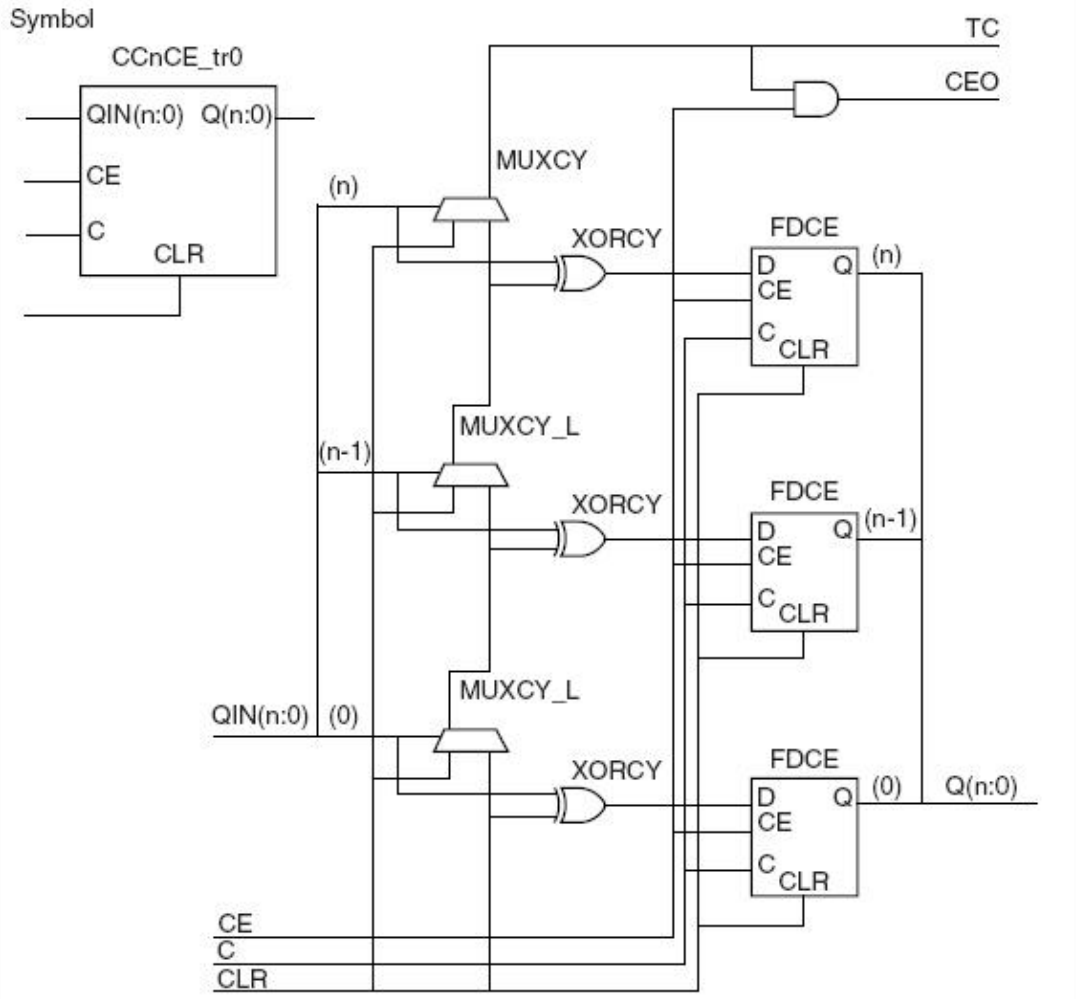
Εικόνα 2.22 : Πρωταρχική υλοποίηση ενός D Flip Flop

Κατά τη σύνθεση θα πρέπει πάντα να συμπεριλαμβάνουμε σύγχρονα clock enable και σύγχρονες ή / και ασύγχρονες συνθήκες αρχικοποίησης για όλες τις σύγχρονες διαδικασίες και τις αρχικοποιήσεις μονάδων (component instantiations) :

```
register: process(CLK, CLR)
Begin
        If (CLR='1') Then
                Q <= 0;
        Elsif CLK'event and CLK='1' then
                if (CE='1') then
                        Q <= D;
                end if;
        End If;
End register;
```

Εικόνα 2.23 : Αρχικοποίηση καταχωρητή

Παρατηρώντας ξανά την εικόνα 2.18, βλέπουμε πως η αλυσίδα κρατουμένου αρχικοποιείται με Vcc και GND. Μια απλή μέθοδος για να αφαιρεθούν αυτά στο σχηματικό φαίνεται στην εικόνα 2.24. Μιας και το σήμα CE πρέπει είναι ενεργά υψηλό για να αυξηθεί ο μετρητής, το Vcc μπορεί να αντικατασταθεί συνδέοντάς το στο σήμα αυτό. Ομοίως, μιας και το σήμα CLR πρέπει να είναι ενεργά χαμηλό για να αυξηθεί ο μετρητής, το GND μπορεί να αντικατασταθεί συνδέοντας το στο σήμα CLR

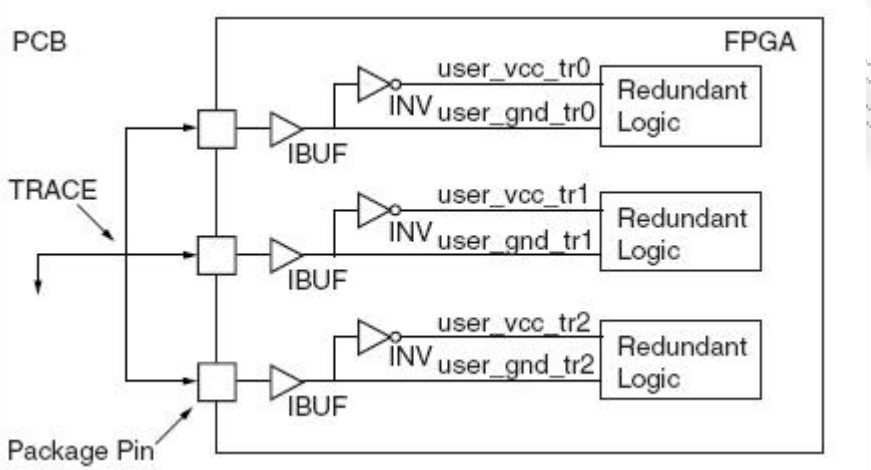


Εικόνα 2.24 : TMR Μετρητής (υλοποιημένος με λογική αλυσίδας κρατουμένου) με δέσμευση των Vcc και GND (Power Tie Down)

Ο μοναδικός τρόπος να επιτευχθεί μια τέτοια υλοποίηση στη σύνθεση είναι να αρχικοποιήσουμε μόνοι μας την πρωταρχική απεικόνιση της λογικής αλυσίδας κρατουμένου ή να χρησιμοποιήσουμε μια από τις μονάδες που βρίσκονται στη βιβλιοθήκη XVRWARE (για FPGA αρχιτεκτονικής Virtex μόνο). Η δημιουργία αριθμητικών με περιγραφή συμπεριφοράς δημιουργεί πάντα ενδεχόμενη χρήση των Vcc και GND κατά τη σύνθεση. Αυτό πέρα από άβολο, έχει και μια πολύ σημαντική επίδραση στον αναμενόμενο ρυθμό σφαλμάτων SEU της σχεδίασής μας.

Αν η σχεδίαση δεν έχει σήμα Global Clock Enable ή Global Reset, ή δεν μπορεί να προσαρμοστεί ώστε να δημιουργηθεί ένα, τότε ένα νέο ενεργό και δρομολογήσιμο σήμα πρέπει να δημιουργηθεί ώστε να παράσχει ένα σημείο σύνδεσης για τις απαραίτητες λογικές σταθερές (Power Tie Down).

Αν κάποιο από αυτά τα σήματα δεν συνδεθεί στη σχεδίαση, η βελτιστοποίηση οδηγεί είτε σε υπερβολική διαγραφή λογικής, είτε εισάγει ξανά τα Vcc και GND. Μια απλή μέθοδος για να παρέχουμε μονόπλευρη λειτουργία των Vcc και GND είναι να αφιερώσουμε έναν ακροδέκτη I/O (τρία για τριπλό πλεονασμό) που θα εισάγει μια πραγματική γείωση από την πλακέτα.



Εικόνα 2.25 : Δεσμευμένες τριπλά πλεονάζουσες Vcc και GND

2.8 Βιβλιοθήκη XVMWARE

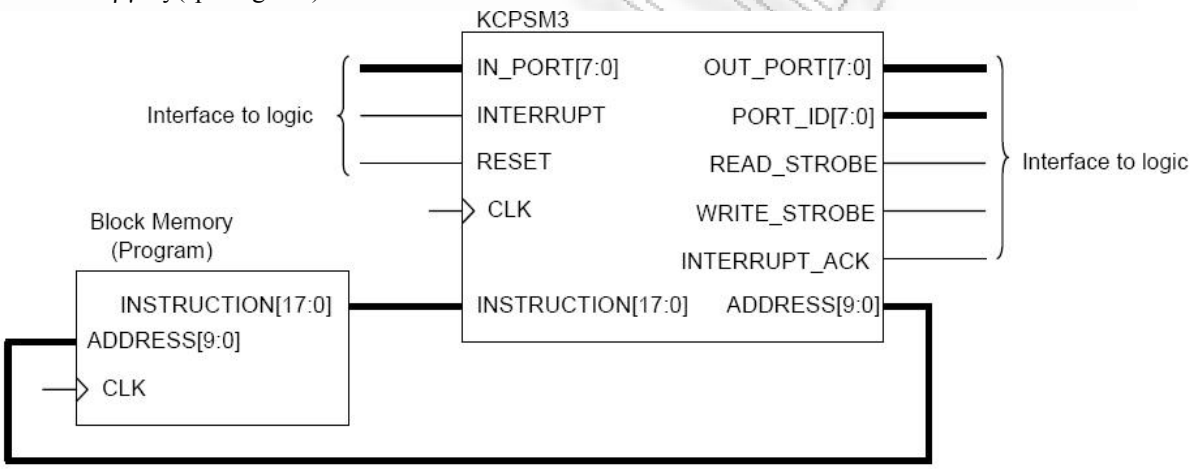
Τέλος, αναφέρουμε τη βιβλιοθήκη XVMWARE, μια βιβλιοθήκη με έτοιμες μονάδες και παραδείγματα σύνθεσης για την υλοποίηση TMR κυκλωμάτων σε VHDL για αρχιτεκτονική Virtex. Δεν θα την αναλύσουμε περαιτέρω όμως καθώς για την υλοποίηση μας θα χρησιμοποιήσουμε ένα Spartan-3 FPGA.

3 PicoBlaze

Σε αυτό το κεφάλαιο θα αναλύσουμε τον επεξεργαστή που θα χρησιμοποιήσουμε στην εφαρμογή μας, τον PicoBlaze [3].

Ο PicoBlaze είναι ένας πολύ απλός, αποδοτικός και χαμηλού κόστους 8-bit μικροελεγκτής, σχεδιασμένος κυρίως για συσκευές Spartan-3. Μπορεί να χρησιμοποιηθεί για την επεξεργασία δεδομένων αλλά και σε εφαρμογές που απαιτούν μια πολύπλοκη αλλά όχι χρονικά κρίσιμη state-machine.

- Η μονάδα KCPSM3 (**C**onstant **C**oded **P**rogrammable **S**tate **M**achine) [4] υλοποιεί τον PicoBlaze και καταλαμβάνει στο Spartan-3 μόνο 96 slices.
- Μια μονής θύρας block RAM χρησιμοποιείται για να δημιουργήσει μια μνήμη εντολών (ROM store) για ένα πρόγραμμα μέχρι 1024 εντολών.
- Έχει απόδοση 43 με 66 MIPS, ανάλογα με τον τύπο συσκευής και το βαθμό ταχύτητας λειτουργίας (speed grade).

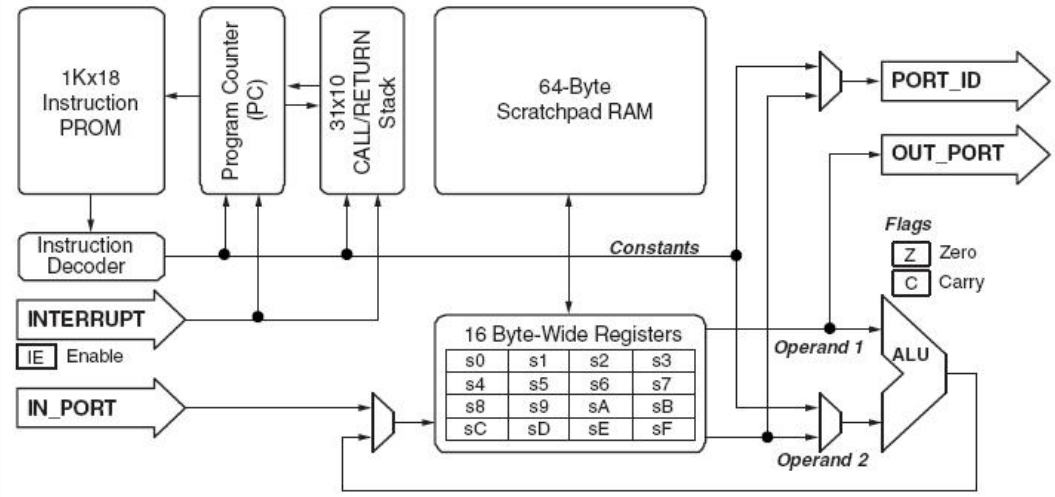


Εικόνα 3.1 : KCPSM3

- Ο KCPSM3 ενσωματώνεται πλήρως στη συσκευή και δε χρειάζεται εξωτερική υποστήριξη.
- Οποιαδήποτε λογική μπορεί να συνδεθεί στη μονάδα μέσα στη συσκευή, που σημαίνει ότι οποιοδήποτε επιπλέον χαρακτηριστικό μπορεί να προστεθεί για μεγαλύτερη ευλυγισία.
- Υπάρχει η δυνατότητα να τοποθετήσουμε πολλαπλούς KCPSM3 σε μια συσκευή
- Σε μια συσκευή XC3S200 Spartan-3 η μονάδα KCPSM3 καταλαμβάνει το 5% της συσκευής (96 slices, ~ 87MHz σε –4 Speed grade, ~43,5 MIPS).

3.1 Λειτουργικές μονάδες

Στην εικόνα 3.2 φαίνεται το διάγραμμα μονάδων του PicoBlaze, οι μονάδες του οποίου αναλύονται στη συνέχεια.



Εικόνα 3.2 : Διάγραμμα μονάδων του μικροεπεξεργαστή PicoBlaze

3.1.1 16 8-bit καταχωρητές γενικού σκοπού (s0 - sF)

Κανένας από αυτούς δεν κρατείται για ειδική χρήση και κανένας δεν έχει υψηλότερη προτεραιότητα από τους άλλους, με αποτέλεσμα να υπάρχει μεγάλη ευλυγισία στη χρήση τους. Δεν υπάρχει συσσωρευτής (accumulator) καθώς οποιοσδήποτε καταχωρητής μπορεί να χρησιμοποιηθεί για αυτή τη λειτουργία. Μπορούν να μετονομαστούν για μεγαλύτερη ευκολία ανάγνωσης του προγράμματος χρησιμοποιώντας ψευδοεντολές του assembler.

3.1.2 Program store μεγέθους1024 εντολών

Ο PicoBlaze υποστηρίζει προγράμματα μήκους 1024 εντολών χρησιμοποιώντας συνήθως μια block μνήμη. Κάθε εντολή έχει μέγεθος 18-bit. Μεταγλωττίζονται κατά τη σχεδίαση (με χρήση ενός assembler) και φορτώνονται αυτόματα κατά την αρχικοποίηση του FPGA. Για μεγαλύτερα προγράμματα χρησιμοποιούμε πολλαπλούς KCPSM3, ο καθένας συνδεδεμένος με μια block μνήμη για να διανείμει τις διάφορες εργασίες.

3.1.3 ALU

8-bit μονάδα επεξεργασίας που εκτελεί όλες τις λειτουργίες ενός μικροελεγκτή, βασικές αριθμητικές, λογικές, σύγκρισης και ελέγχου bit προς bit καθώς και ολίσθησης και περιστροφής. Όλες οι λειτουργίες κάνουν χρήση ενός τελεστή (operand), την τιμή του οποίου παίρνουμε από έναν καταχωρητή sX και το αποτέλεσμα αποθηκεύεται στον ίδιο καταχωρητή. Αν χρειάζεται δεύτερος τελεστής μπορούμε να χρησιμοποιήσουμε έναν καταχωρητή sY ή μια 8-bit σταθερά (kk). Το γεγονός ότι καθορίζει την τιμή μιας σταθεράς χωρίς να επιβαρύνει το μέγεθος του προγράμματος ή την επίδοσή του ενισχύει την απλότητα του συνόλου εντολών του.

3.1.4 Δείκτες κατάστασης και έλεγχος λειτουργίας του προγράμματος

Οι λειτουργίες της ALU επηρεάζουν τους δείκτες κατάστασης (flags) ZERO και CARRY. Το zero flag υποδεικνύει αν η τελευταία λειτουργία της ALU είχε μηδενικό αποτέλεσμα ενώ το carry flag υποδεικνύει διάφορες συνθήκες (αν για παράδειγμα έχουμε υπερχείλιση) ανάλογα με την τελευταία λειτουργία που εκτελέστηκε. Το INTERRUPT_ENABLE flag ενεργοποιεί την είσοδο INTERRUPT.

3.1.5 Μνήμη Scratchpad

Εσωτερική μνήμη γενικού σκοπού, 64-byte, άμεσης ή έμμεσης διευθυνσιοδότησης. Τα δεδομένα οποιουδήποτε από τους 16 καταχωρητές μπορούν να γραφτούν σε οποιαδήποτε από τις 64 θέσεις της χρησιμοποιώντας μια εντολή Store. Η συμπληρωματική της εντολή, Fetch, επιτρέπει στα δεδομένα οποιασδήποτε από τις 64 θέσεις της μνήμης να γραφτούν σε έναν από τους 16 καταχωρητές. Η 6-bit διεύθυνση που καθορίζει μια τοποθεσία στην μνήμη scratchpad μπορεί να προσδιοριστεί άμεσα στον κώδικα με μια σταθερά (ss), ή έμμεσα ως περιεχόμενο οποιουδήποτε από τους 16 καταχωρητές (sY). Μόνο τα χαμηλά 6-bits της διεύθυνσης χρησιμοποιούνται, οπότε πρέπει να προσέξουμε να μην ξεπεράσουμε το εύρος $00-3F_{16}$ της διαθέσιμης μνήμης.

3.1.6 Input / Output

Ο KCPSM3 υποστηρίζει έως 256 πόρτες εισόδου και 256 πόρτες εξόδου ή συνδυασμό αυτών. Οι I/O πόρτες διευρύνουν τις ικανότητες του επεξεργαστή επιτρέποντάς του να συνδεθεί με περιφερειακές μονάδες ή ακόμα και άλλη FPGA λογική.

Η πόρτα που θα προσπελαστεί υποδεικνύεται με μια 8-bit διεύθυνση στην 'PORT_ID'. Η διεύθυνση πόρτας μπορεί να καθοριστεί άμεσα ως μια σταθερά (pp), ή έμμεσα ως περιεχόμενο οπουδήποτε από τους 16 καταχωρητές (sY).

- Κατά την λειτουργία εισόδου (input) διαβάζει δεδομένα από την πόρτα εισόδου, IN_PORT[7:0], σ' έναν από τους 16 καταχωρητές. Μια τέτοια λειτουργία υποδεικνύεται από έναν παλμό στο READ_STROBE (1 κατά τον δεύτερο κύκλο), το οποίο βεβαιώνει ότι τα δεδομένα έχουν αποκτηθεί από τον επεξεργαστή.
- Κατά την λειτουργία εξόδου (output), το περιεχόμενο οποιουδήποτε από τους 16 καταχωρητές γράφεται στην πόρτα εξόδου, OUT_PORT[7:0]. Ένας παλμός στο WRITE_STROBE υποδεικνύει μια τέτοια λειτουργία. Το σήμα αυτό χρησιμοποιείται από το interface για να διασφαλίσει ότι μόνο έγκυρα δεδομένα περνούν στα εξωτερικά συστήματα. Τυπικά το WRITE_STROBE (1 κατά τον δεύτερο κύκλο) χρησιμοποιείται ως clock enable ή write enable.

3.1.7 Program Counter (PC)

Είναι μεγέθους 10-bit και υποστηρίζει κώδικα μεγέθους 1024 εντολών ($000-3FF_{hex}$). Αν φτάσει στην τελευταία θέση μνήμης (3FF), τότε αναδιπλώνεται στη θέση 000.

3.1.8 CALL / RETURN Stack

Αποθηκεύει έως 31 διευθύνσεις εντολών ενεργοποιώντας ένθετες CALL, βάθους 31 επιπέδων. Μιας και χρησιμοποιείται και κατά τη διάρκεια της λειτουργίας interrupt, τουλάχιστον ένα από αυτά τα επίπεδα θα πρέπει να μη χρησιμοποιείται ώστε να είναι ελεύθερο όταν τα interrupts είναι ενεργά.

Υλοποιείται ως ξεχωριστός κυκλικός buffer. Όταν είναι γεμάτος γράφει πάνω στην παλιότερη τιμή και δεν υπάρχουν εντολές που να ελέγχουν αυτόν και τον stack pointer.

3.1.9 Διακοπές - Interrupt

Ο επεξεργαστής παρέχει ένα μοναδικό INTERRUPT σήμα, επιτρέποντάς του να χειριστεί εξωτερικά ασύγχρονα σήματα. Αντιμετωπίζει τα interrupts γρήγορα, σε 5 κύκλους ρολογιού. Τα interrupts είναι εξ' ορισμού απενεργοποιημένα και μετά ενεργοποιούνται και απενεργοποιούνται υπό τον έλεγχο του προγράμματος. Έτσι, αφού μετά από reset η είσοδος INTERRUPT είναι απενεργοποιημένη, για να την ενεργοποιήσουμε ξανά χρησιμοποιούμε την εντολή ENABLE INTERRUPT. Για να την απενεργοποιήσουμε σε οποιοδήποτε σημείο του κώδικα χρησιμοποιούμε την εντολή DISABLE INTERRUPT.

Από τη στιγμή που ενεργοποιείται, το σήμα INTERRUPT θα πρέπει να παραμείνει ενεργό για τουλάχιστον δύο κύκλους ρολογιού ώστε να είναι σίγουρο πως θα αναγνωριστεί, δημιουργώντας ένα συμβάν διακοπής (interrupt event).

Ένα ενεργό interrupt αναγκάζει τον επεξεργαστή να εκτελέσει μια εντολή CALL 3FF (κλήση υπορουτίνας στην τελευταία διεύθυνση μνήμης) αφού ολοκληρώσει την εντολή που ήδη εκτελεί. Εκεί καθορίζεται που πρέπει να χειριστεί το interrupt, συνήθως με μια εντολή Jump σε μια Ρουτίνα Εξυπηρέτησης Διακοπής (Interrupt Service Routine - ISR).

Τα ZERO και CARRY flags φυλάσσονται αυτόματα και οποιαδήποτε περαιτέρω interrupts απενεργοποιούνται. Ο τωρινός PC αποθηκεύεται στον Call / Return Stack. Σε αυτό το σημείο ένας παλμός δημιουργείται στην έξοδο INTERRUPT_ACK (κατά τον δεύτερο κύκλο του interrupt event) για να βεβαιώσει ότι το interrupt έχει αναγνωριστεί.

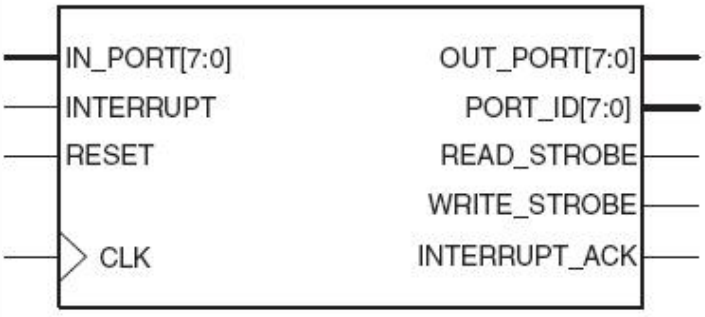
Η εντολή RETURNI εξασφαλίζει ότι το τέλος της ISR επαναφέρει την κατάσταση των flags και καθορίζει αν τα μελλοντικά interrupts είναι ενεργά ή όχι. Ο PC στον Call / Return Stack φορτώνεται στον PC, τα zero και carry flags επαναφέρονται, και το πρόγραμμα συνεχίζει από την επόμενη εντολή από αυτή που εκτελούνταν όταν έγινε το interrupt.

3.1.10 Reset

Επαναφέρει τον επεξεργαστή στην αρχική του κατάσταση. Το πρόγραμμα θα εκτελεστεί από τη διεύθυνση ‘000’ και τα interrupts θα απενεργοποιηθούν. Οι δείκτες κατάστασης και ο Call / Return stack θα γίνουν reset. Τα περιεχόμενα των καταχωρητών και της μνήμης scratchpad δεν επηρεάζονται.

3.2 Σήματα στο υψηλότερο επίπεδο σχεδίασης

Στην εικόνα 3.3 φαίνονται τα σήματα του PicoBlaze στο υψηλότερο επίπεδο σχεδίασης.



Εικόνα 3.3 : Σήματα σύνδεσης του PicoBlaze

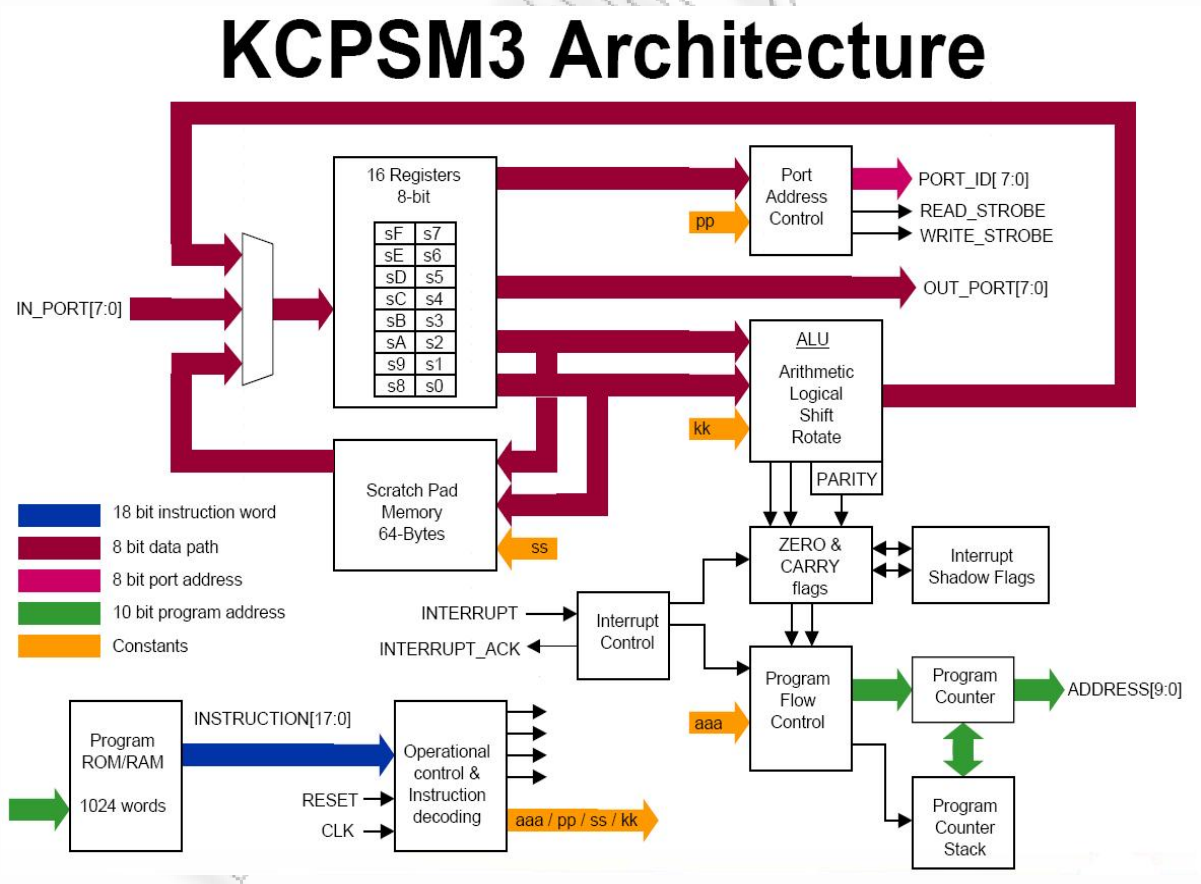
- **IN_PORT[7:0] :** Δίνει έγκυρα δεδομένα στην πόρτα κατά την εντολή Input. Τα δεδομένα “αιχμαλωτίζονται” κατά την άνοδο του ρολογιού.
- **INTERRUPT :** Αν το INTERRUPT_ENABLE flag είναι “1”, δημιουργεί ένα interrupt event θέτοντας αυτή την είσοδο 1 για τουλάχιστον 2 κύκλους ρολογιού. Αν το INTERRUPT_ENABLE flag είναι “0”, αυτή η είσοδος αγνοείται.
- **RESET :** Για να κάνουμε reset τον PicoBlaze και να δημιουργήσουμε ένα συμβάν reset θέτουμε 1 αυτή την είσοδο για τουλάχιστον 1 κύκλο ρολογιού. Ένα συμβάν reset δημιουργείται αυτόματα μετά την αρχικοποίηση του FPGA.
- **CLK :** Η συχνότητα μπορεί να κυμαίνεται από DC στη μέγιστη λειτουργική συχνότητα που αναφέρει το ISE της Xilinx. Όλα τα σύγχρονα στοιχεία του PicoBlaze χρονίζοντα από την ανοδική ακμή του ρολογιού. Δεν υπάρχουν απαιτήσεις κύκλου εργασιών του ρολογιού (clock duty-cycle) πέρα από τις ελάχιστες απαιτήσεις εύρους παλμού του FPGA.
- **OUT_PORT[7:0] :** Τα δεδομένα εξόδου εμφανίζονται σε αυτήν την πόρτα για 2 κύκλους ρολογιού κατά την εντολή Output. Τα δεδομένα αιχμαλωτίζονται στο FPGA όταν το WRITE_STROBE είναι 1.

- **PORT_ID[7:0] :** Η διεύθυνση της I/O πόρτας που θα χρησιμοποιηθεί κατά την Input ή Output εντολή, εμφανίζεται σε αυτή την πόρτα για 2 κύκλους ρολογιού.
- **READ_STROBE :** Όταν είναι 1, το σήμα υποδεικνύει ότι τα δεδομένα εισόδου στην IN_PORT[7:0] πόρτα έχουν τοποθετηθεί στον καταχωρητή δεδομένων (κατά την Input εντολή). Το σήμα αυτό ενεργοποιημένο στον δεύτερο κύκλο ρολογιού της εντολής Input.
- **WRITE_STROBE :** Όταν είναι 1, επιβεβαιώνει τα δεδομένα εξόδου στην πόρτα OUT_PORT[7:0] (κατά την εντολή Output). Είναι ενεργοποιημένο στον δεύτερο κύκλο ρολογιού της εντολής Output. Λαμβάνει τα δεδομένα εξόδου στο FPGA όταν το WRITE_STROBE είναι 1.

3.3 Αρχιτεκτονική του KCPSM3

Ο KCPSM3 έχει τα εξής επιπλέον χαρακτηριστικά από την προγενέστερή του έκδοση, τον KCPSM :

- Οι εντολές Compare και Test επιτρέπουν στους καταχωρητές να είναι υπό ερώτηση χωρίς να μεταβάλλεται το περιεχόμενό τους
- Η εντολή Test υπολογίζει επίσης το Parity
- Μια εσωτερική 64-byte scratch pad memory επιτρέπει την αποθήκευση περισσοτέρων μεταβλητών με αποτέλεσμα να γράφονται περισσότερο διαισθητικά προγράμματα.
- Παρέχει σήμα αναγνώρισης διακοπής (interrupt acknowledgement).



Εικόνα 3.4 : Αρχιτεκτονική KCPSM3

Ο KCPSM3 μπορεί να θεωρηθεί ως μια μηχανή βασισμένη σε σταθερές (Constant(k) Coded).

Μπορούμε να καθορίσουμε σταθερές και να τις χρησιμοποιήσουμε στις περισσότερες φάσεις ενός προγράμματος.

- Σταθερές για δεδομένα (constant data value), χρησιμοποιούνται σε λειτουργίες της ALU
- Σταθερές για διεύθυνση πόρτας (constant port address), χρησιμοποιούνται για να προσπελάσουμε μια συγκεκριμένη πληροφορία ή να ελέγξουμε εξωτερική λογική του KCPSM3.
- Σταθερές για διευθύνσεις (constant address values), χρησιμοποιούνται κατά την προσπέλαση της εσωτερικής μνήμης scratchpad και για να ελέγξουμε την αλληλουχία του προγράμματός μας.

Το σύνολο εντολών του KCPSM3 έχει σχεδιαστεί ώστε να επιτρέπει στις σταθερές να καθορίζονται σε επίπεδο εντολής. Επομένως, η χρήση μιας σταθεράς δεν επιφέρει επιπλέον επιβάρυνση στο μέγεθος του προγράμματος ή της εκτέλεσής του. Αυτό έχει ως αποτέλεσμα την επέκταση του συνόλου εντολών που υποστηρίζει σ' ένα μεγαλύτερο εύρος με εικονικές εντολές.

Είναι σημαντικό πως όλες οι εντολές, υπό οποιεσδήποτε συνθήκες, εκτελούνται σε 2 κύκλους ρολογιού. Αυτός ο σταθερός ρυθμός εκτέλεσης είναι μεγάλης αξίας όταν καθορίζουμε το χρόνο εκτέλεσης ενός προγράμματος, ιδίως όταν είναι ενσωματωμένο σε συστήματα πραγματικού χρόνου.

Το μήκος προγράμματος είναι 1024 εντολές και επομένως ακολουθεί το πρότυπο $1024x8$ μιας μόνο Spartan-3, Virtex-II ή Virtex-IIPRO Block RAM. Αυτό σημαίνει πως όλες οι διευθύνσεις ορίζονται ως 10-bits που εμπεριέχονται στις εντολές (ο assembler υποστηρίζει ετικέτες (line labels) για να απλοποιήσει τη γραφή του προγράμματος). Το συγκεκριμένο μέγεθος της μνήμης καθορίζει ένα σταθερό επίπεδο απόδοσης της μονάδας.

3.4 Ενσωμάτωση του KCPSM3 στη σχεδίαση

Η μονάδα KCPSM3 δίδεται ως πηγαίος κώδικας σε VHDL (αρχείο kcpsm3.vhd) και είναι έτσι γραμμένη ώστε να παρέχει τη βέλτιστη υλοποίηση σε ένα Spartan-3 ή Virtex-II(PRO). Ο κώδικας αυτός είναι κατάλληλος για υλοποίηση και προσομοίωση της μονάδας.

```
component kcpsm3
  Port (      address : out std_logic_vector(9 downto 0);
          Instruction : in std_logic_vector(17 downto 0);
              port_id : out std_logic_vector(7 downto 0);
         write_strobe : out std_logic;
             out_port : out std_logic_vector(7 downto 0);
         read_strobe : out std_logic;
              in_port : in std_logic_vector(7 downto 0);
            interrupt : in std_logic;
        interrupt_ack : out std_logic;
                reset : in std_logic;
                  clk : in std_logic);
  end component;
```

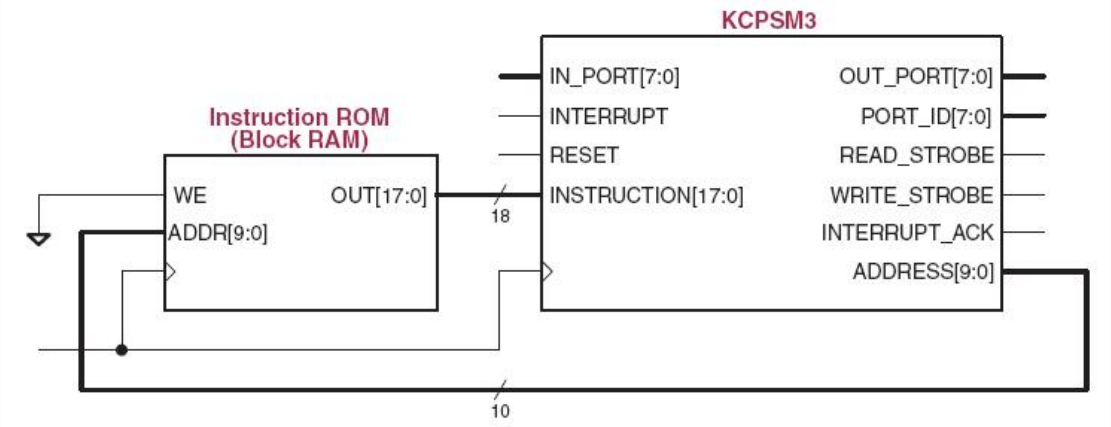
Εικόνα 3.5 : Δήλωση της μονάδας KCPSM3

```
processor: kcpsm3
  port map(      address => address,
             instruction => instruction,
                 port_id => port_id,
            write_strobe => write_strobe,
                out_port => out_port,
             read_strobe => read_strobe,
                 in_port => in_port,
               interrupt => interrupt,
           interrupt_ack => interrupt_ack,
                   reset => reset,
                     clk => clk);
```

Εικόνα 3.6 : Αρχικοποίηση της μονάδας KCPSM3

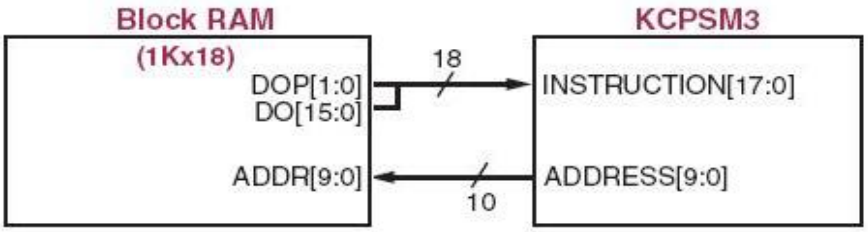
3.5 Μνήμη εντολών

Ο PicoBlaze εκτελεί ένα πρόγραμμα που βρίσκεται αποθηκευμένο στο FPGA. Η μονάδα KCPSM3 παρέχει την PicoBlaze ALU, register file, μνήμη scratchpad κτλ. Ένα είδος εσωτερικής μνήμης, συνήθως μια block ROM, παρέχει στον PicoBlaze την μνήμη εντολών. Για να δημιουργήσουμε αποτελεσματικά μια ενσωματωμένη στο τσιπ ROM, ο ακροδέκτης write enable, WE, παραμένει LOW απενεργοποιώντας όποια πιθανή λειτουργία εγγραφής.



Εικόνα 3.7 : Συνήθης υλοποίηση με χρήση μιας Single 1Kx18 Block RAM ως instruction store

Στις περισσότερες εφαρμογές το πρόγραμμα που θα εκτελέσει ο PicoBlaze αποθηκεύεται σε μια FPGA Block RAM, σχηματίζοντας μια 1Kx18 ROM. Ο κώδικας της εφαρμογής έχει μεταφραστεί και μεταγλωττίζεται ως μέρος της FPGA σχεδίασης. Η μνήμη εντολών φορτώνεται αυτόματα στην προσαρτημένη RAM κατά τη διαδικασία αρχικοποίησης του FPGA.



Εικόνα 3.8 : Αρχικοποίηση με μια Single 1Kx18 Block RAM

3.5.1 Σύνδεση της μνήμης εντολών

Ο KCPSM3 assembler θα δημιουργήσει ένα VHDL αρχείο, στο οποίο θα ορίζει μια block RAM και τα αρχικά της περιεχόμενα. Μπορεί να χρησιμοποιηθεί για υλοποίηση και προσομοίωση του επεξεργαστή. Στις επόμενες εικόνες (3.9 και 3.10) έχουμε τη δήλωση και την αρχικοποίηση της μνήμης στο υψηλότερο επίπεδο σχεδίασης.

```
component prog_rom
  Port (      address : in std_logic_vector(9 downto 0);
         instruction : out std_logic_vector(17 downto 0);
                 clk : in std_logic);
end component;
```

Εικόνα 3.9 : Δήλωση της μνήμης εντολών στο υψηλότερο επίπεδο σχεδίασης

```
program: prog_rom
  port map(      address => address,
              instruction => instruction,
                      clk => clk);
```

Εικόνα 3.10 : Αρχικοποίηση της μνήμης εντολών στο υψηλότερο επίπεδο σχεδίασης

Το όνομα της μνήμης εντολών εξαρτάται από το όνομα του προγράμματος, δηλαδή αν το αρχείο προγράμματος ονομάζεται “foo.psm”, τότε ο assembler θα δημιουργήσει ένα αρχείο για τη μνήμη εντολών με όνομα “foo.vhd”. Επομένως, για κάθε διαφορετική εφαρμογή που θα χρησιμοποιήσουμε τον PicoBlaze, θα έχουμε και ένα διαφορετικό αρχείο μνήμης. Έτσι, για το παράδειγμα του προγράμματος foo, στο υψηλότερο επίπεδο σχεδίασης θα έχουμε τη μνήμη foo και όχι prog_rom.

3.6 Σύνολο εντολών του KCPSM3

Οι εντολές που υποστηρίζει ο KCPSM3 είναι μεγέθους 18 bits και μπορούμε να τις κατηγοριοποιήσουμε σε επτά ομάδες ανάλογα με τη λειτουργία τους.

- Ελέγχου προγράμματος (Program Control)
- Αριθμητικές (Arithmetic)
- Διακοπής (Interrupt)
- Λογικές (Logical)
- Αποθήκευσης (Storage)
- Ολίσθησης και Περιστροφής (Shift & Rotate)
- Εισόδου / Εξόδου (Input / Output)

Πίνακας 3.1 : Σύνολο εντολών που υποστηρίζει ο KCPSM3 (1)

Program Control	Arithmetic	Logical	Shift & Rotate
JUMP aaa	ADD sX, kk	LOAD sX, kk	SR0 sX
JUMP Z, aaa	ADDCY sX, kk	AND sX, kk	SR1 sX
JUMP NZ, aaa	SUB sX, kk	OR sX, kk	SRX sX
JUMP C, aaa	SUBCY sX, kk	XOR sX, kk	SRA sX
JUMP NC, aaa	COMPARE sX, kk	TEST sX, kk	RR sX
CALL aaa	ADD sX, sY	LOAD sX, sY	SL0 sY

Program Control	Arithmetic	Logical	Shift & Rotate
CALL Z, aaa	ADDCY sX, sY	AND sX, sY	SL1 sY
CALL NZ, aaa	SUB sX, sY	OR sX, sY	SLX sY
CALL C, aaa	SUBCY sX, sY	XOR sX, sY	SLA sY
CALL NC, aaa	COMPARE sX, sY	TEST sX, sY	RL sY
RETURN			
RETURN Z			
RETURN NZ			
RETURN C			
RETURN NC			

Πίνακας 3.2 : Σύνολο εντολών που υποστηρίζει ο KCPSM3 (2)

Interrupt	Storage	Input / Output
RETURNI ENABLE	STORE sX, ss	INPUT sX, pp
RETURNI DISABLE	STORE sX, (sY)	INPUT sX, (sY)
	FETCH sX, ss	OUTPUT sX, pp
ENABLE INTERRUPT	FETCH sX, (sY)	OUTPUT sX, (sY)
DISABLE INTERRUPT		

Τα 'X' και 'Y' αναφέρονται στους καταχωρητές s με εύρος 0 έως F.
Το 'kk' αντιπροσωπεύει μια σταθερά εύρους 00 έως FF.
Το 'aaa' αντιπροσωπεύει μια διεύθυνση εύρους 000 έως 3FF.
Το 'pp' αντιπροσωπεύει μια διεύθυνση πόρτας εύρους 00 έως FF.
Το 'ss' αντιπροσωπεύει μια εσωτερική διεύθυνση αποθήκευσης εύρους 00 έως 3F.

3.7 Μέγεθος και απόδοση

Ο παρακάτω πίνακας δείχνει την απόδοση του PicoBlaze για διαφορετικές οικογένειες FPGA και διαφορετικό speed grade.

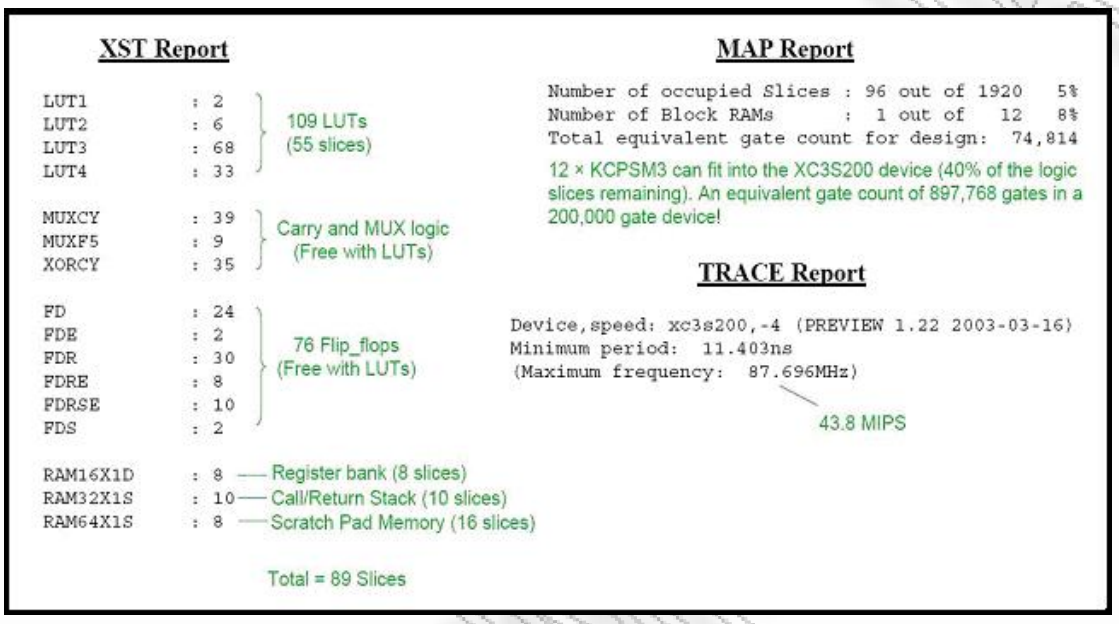
Πίνακας 3.3 : Απόδοση PicoBlaze χρησιμοποιώντας το ελάχιστο Speed Grade

Οικογένεια FPGA (Speed Grade)	Μέγιστη Συχνότητα Ρολογιού	Μέγιστη Απόδοση Εκτέλεσης
Spartn-3 (-4) FPGA	88 MHz	44 MIPS
Virtex-II (-6) FPGA	152 MHz	76 MIPS
Virtex-II Pro (-7) FPGA	200 MHz	100 MIPS

Δεν είναι πάντα απαραίτητο να λειτουργούμε τον PicoBlaze στη μέγιστη συχνότητα ρολογιού, εκτός και αν η τελική εφαρμογή το απαιτεί. Στην ουσία, το να λειτουργεί σε χαμηλότερες συχνότητες έχει πλεονεκτήματα. Συχνά χειρίζεται πιο αργές περιφερειακές λειτουργίες όπως σειριακές επικοινωνίες, όπου δεν ορίζουν την επίδοση του FPGA. Μια πιο αργή συχνότητα ρολογιού μειώνει τον

αριθμό των αδρανών κύκλων εντολής καθώς επίσης και την κατανάλωση ισχύος του τελικού συστήματος.

Τα παρακάτω στοιχεία προκύπτουν από τις αναφορές στο ISE για τη μονάδα KCPSM3 σε μια συσκευή XC3S200.



Εικόνα 3.11 : Αναφορές από το ISE για τη μονάδα KCPSM 3 σε μια συσκευή XC3S200

Αφού ολοκληρώσαμε την παρουσίαση του PicoBlaze, στη συνέχεια θα παρουσιάσουμε τα στάδια υλοποίησης της εφαρμογής της τεχνικής TMR σε αυτόν. Στο επόμενο κεφάλαιο περιγράφονται αναλυτικά όλες οι σχεδιαστικές αποφάσεις που πάρθηκαν για την επίτευξη του ενισχυμένου, μέσω της τεχνικής TMR, PicoBlaze.

4 Εφαρμογή – Υλοποίηση

Από τον αρχικό κώδικα του PicoBlaze σε VHDL, δημιουργήθηκε μια νέα, με ακριβώς ίδια λειτουργικότητα, CPU, όπου κάθε εσωτερική μονάδα της προστατεύεται από σφάλματα με χρήση της τεχνικής TMR. Αυτή είναι μια καλύτερη προσέγγιση από το να τριπλασιάσουμε απλά την αρχική CPU και να προσθέσουμε voters στις εξόδους, καθώς παρέχει καλύτερη προστασία. Η υλοποίηση έγινε σύμφωνα με τις προτάσεις της Xilinx, όπως αυτές περιγράφηκαν στο δεύτερο κεφάλαιο ενώ όλα τα βήματα που ακολουθήθηκαν περιγράφονται αναλυτικά σε αυτό.

Ο πηγαίος κώδικας του PicoBlaze (ο KCPSM3), βρίσκεται στην ιστοσελίδα της Xilinx, http://www.xilinx.com/products/ipcenter/picoblaze-S3-V2-Pro.htm. Για την σχεδίαση, σύνθεση και υλοποίηση του ενισχυμένου PicoBlaze χρησιμοποιήσαμε το περιβάλλον της Xilinx, ISE (Integrated Software Environment) 9.2.04i ενώ η υλοποίησή μας έγινε σε μια συσκευή FPGA Spartan-3, XC3S200.

Από τα αρχεία πηγαίου κώδικα που βρίσκονται στο φάκελο του PicoBlaze, θα χρησιμοποιήσουμε τα εξής :

- **kcpsm3.vhd :** είναι το χαμηλότερο επίπεδο σχεδίασης του KCPSM3 και είναι το αρχείο πάνω στο οποίο θα βασιστεί όλη μας η σχεδίαση.
- **embedded_kcpsm3.vhd :** όπου είναι το υψηλότερο επίπεδο σχεδίασης, στο οποίο γίνεται και η σύνδεση του KCPSM3 με τη μνήμη εντολών.

4.1 Αναγνώριση και ταξινόμηση

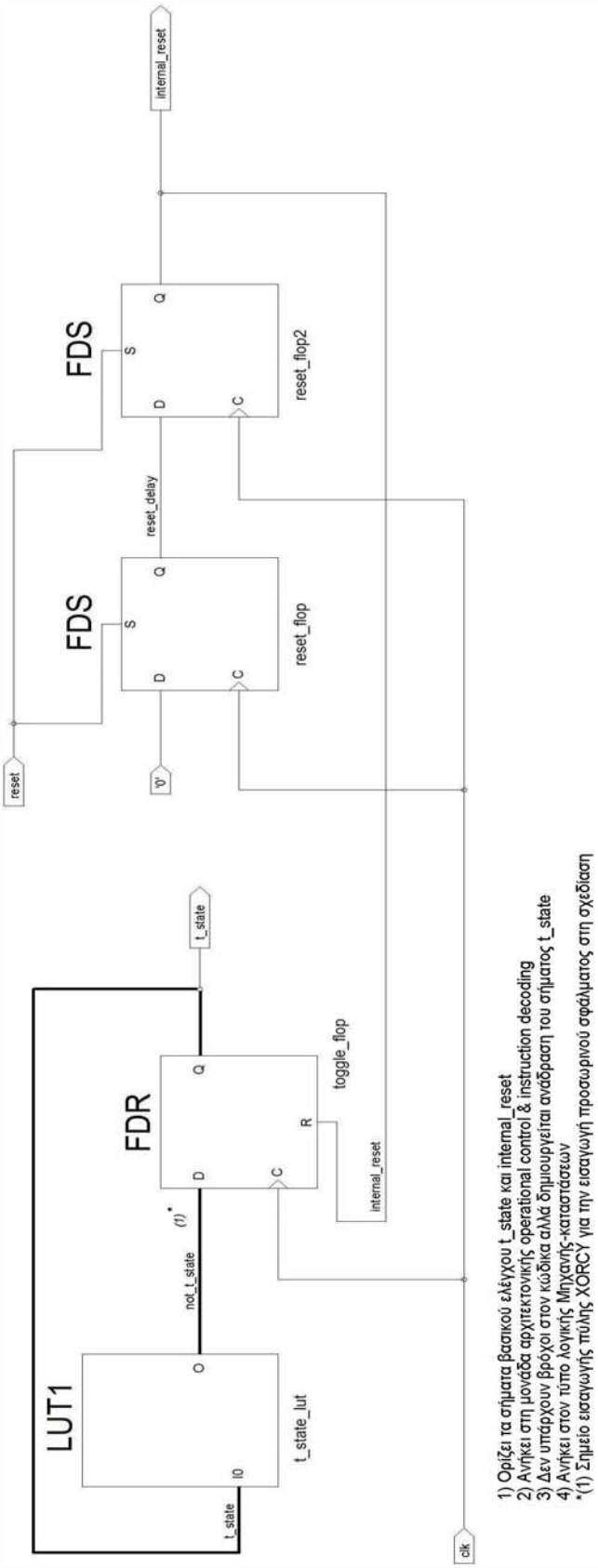
Αρχικό βήμα για την υλοποίηση της TMR εκδοχής του PicoBlaze είναι η αναγνώριση των μονάδων που τον αποτελούν (ο KCPSM3 είναι υλοποιημένος με πρωταρχικά στοιχεία της βιβλιοθήκης [5]) και η ταξινόμησή τους σύμφωνα με τον τύπο λογικής που υλοποιούν. Αυτό είναι και το πιο βασικό βήμα καθώς ο τρόπος εφαρμογής της τεχνικής TMR για κάθε μονάδα θα επιλεχθεί βάση αυτής της πληροφορίας.

Όπως είπαμε και προηγουμένως, ο συνθέσιμος κώδικας παρέχεται από το αρχείο kcpsm3.vhd, ενώ η μνήμη εντολών από τον assembler, διαφορετική για κάθε εφαρμογή που θα εκτελέσει ο PicoBlaze. Σύμφωνα με αυτό το αρχείο ο KCPSM3 αποτελείται από τις εξής επιμέρους μονάδες (οι ονομασίες των μονάδων παραθέτονται σύμφωνα με το πώς αναφέρονται στον κώδικα) :

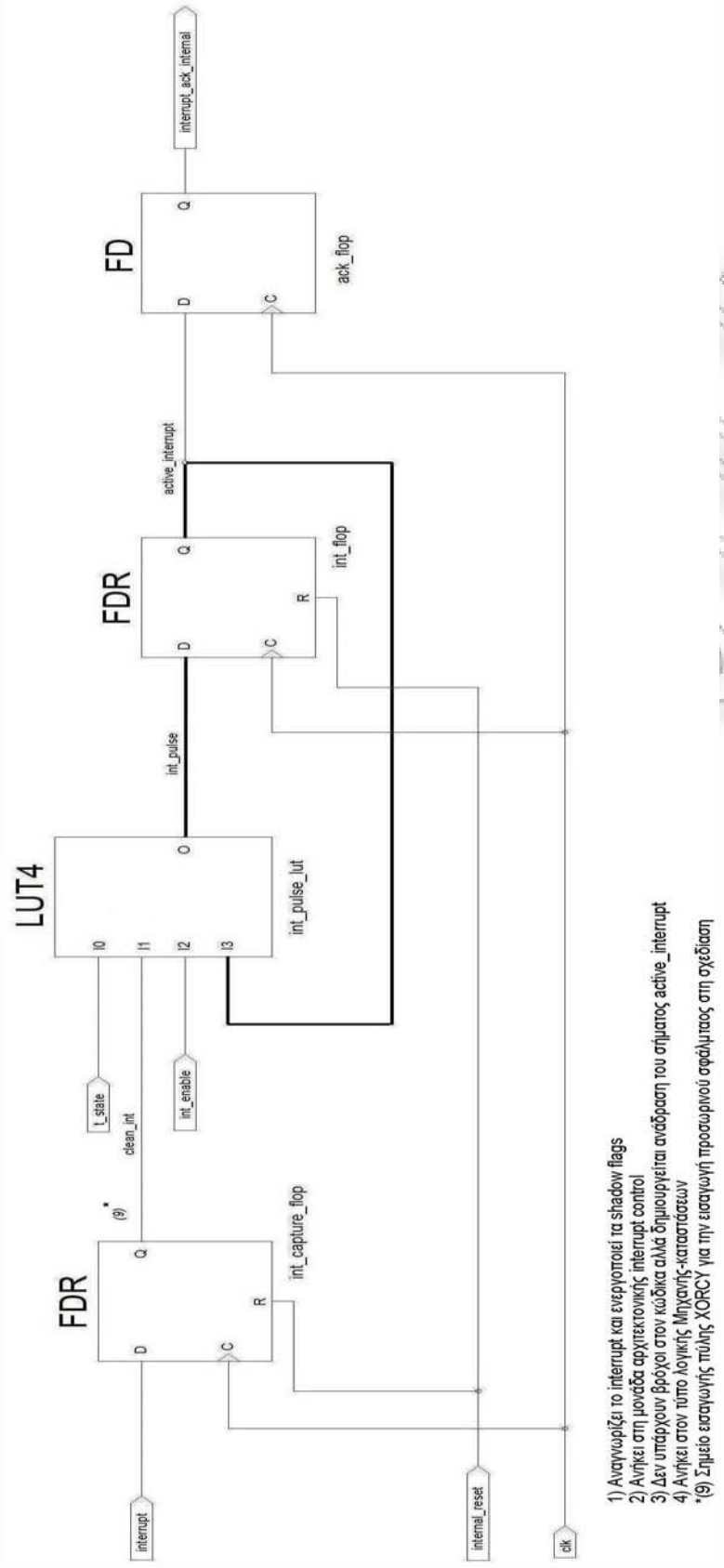
- T-state and internal_reset
- Interrupt capture
- Shadow flags
- Decode instructions that set or reset interrupt enable
- Decodes for the control of the program counter and Call / Return Stack
- Enable for flags
- Zero flag
- Parity detection
- Carry flag selection
- Program counter
- Register bank and 2nd operand selection
- Store memory
- Logical operations
- Shift-rotate operations
- Arithmetic operations
- ALU multiplexer
- Read, Write Strobes

- Program Call / Return Stack
- Stack address pointer

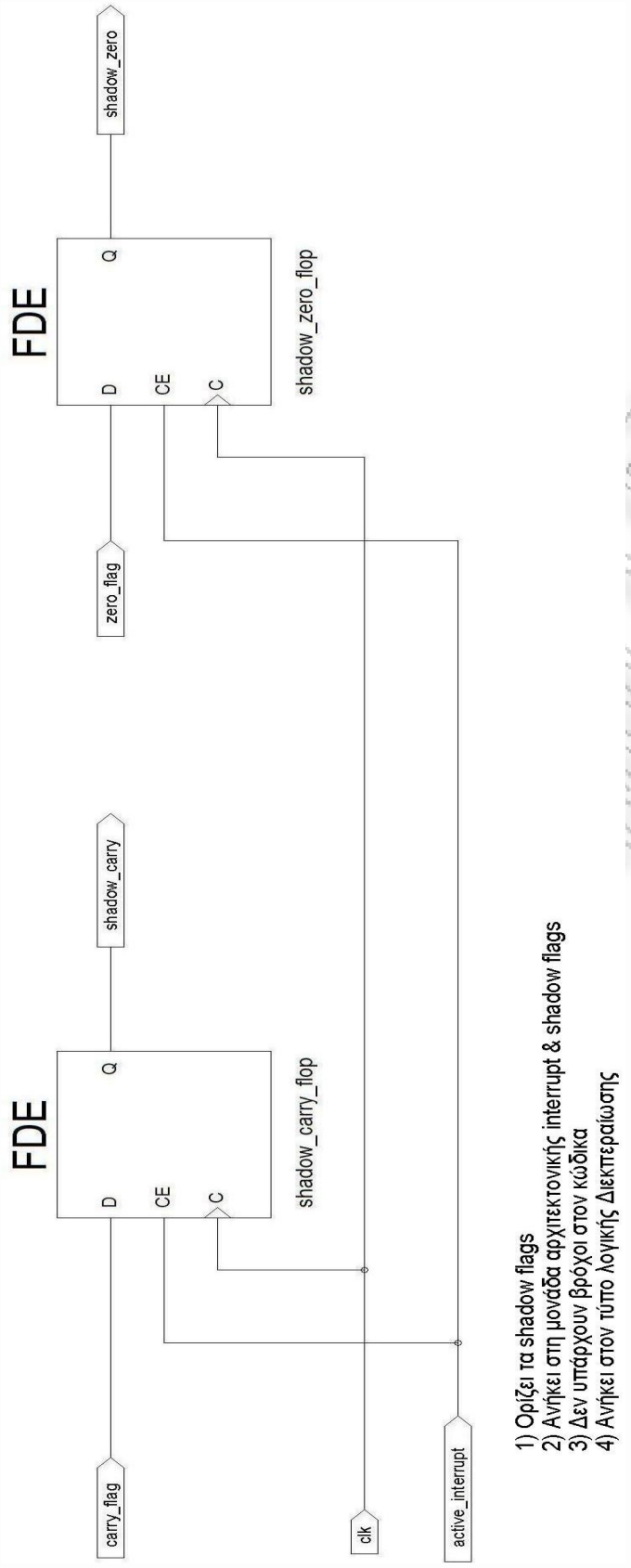
Για το σκοπό της ταξινόμησης, από τον VHDL κώδικα του KCPSM3 δημιουργήσαμε το σχηματικό της κάθε μονάδας (χρησιμοποιώντας πρωταρχικά στοιχεία της βιβλιοθήκης σχηματικών [6]) ώστε να μπορέσουμε να την τοποθετήσουμε στην κατάλληλη κατηγορία. Οι εικόνες 4.1-4.19 είναι τα σχηματικά αυτά, απλουστευμένα σε κάποιες περιπτώσεις ώστε να γίνει πιο κατανοητό για ποιο λόγο η κάθε μονάδα ανήκει στην εκάστοτε ομάδα. Σε αυτά υπάρχει μια σύντομη περιγραφή της λειτουργίας τους και η μονάδα αρχιτεκτονικής στην οποία ανήκουν σύμφωνα με το διάγραμμα μονάδων της εικόνας 4.3. Επισημαίνονται οι βρόχοι που υπάρχουν στον κώδικα και οι πιθανές αναδράσεις σημάτων που δημιουργούνται καθώς επίσης ο τύπος λογικής στην οποία ανήκουν. Οι βρόχοι που υπάρχουν στον κώδικα (π.χ το pc_loop στην εικόνα 4.10) επισημαίνονται με διακεκομμένα πλαίσια ενώ με έντονες γραμμές οι αναδράσεις - ανατροφοδοτήσεις που δημιουργούνται (π.χ στην εικόνα 4.1). Στα σχηματικά 4.1, 4.2, 4.5, 4.8, 4.10, 4.11, 4.12, 4.16 και 4.17 δηλώνονται τα σημεία εισαγωγής πυλών XOR, αριθμημένα σύμφωνα με τους πίνακες 4.3 και 4.4 της ενότητας 4.4.2, με τη βοήθεια των οποίων θα εισάγουμε στη σχεδίαση προσωρινά σφάλματα.



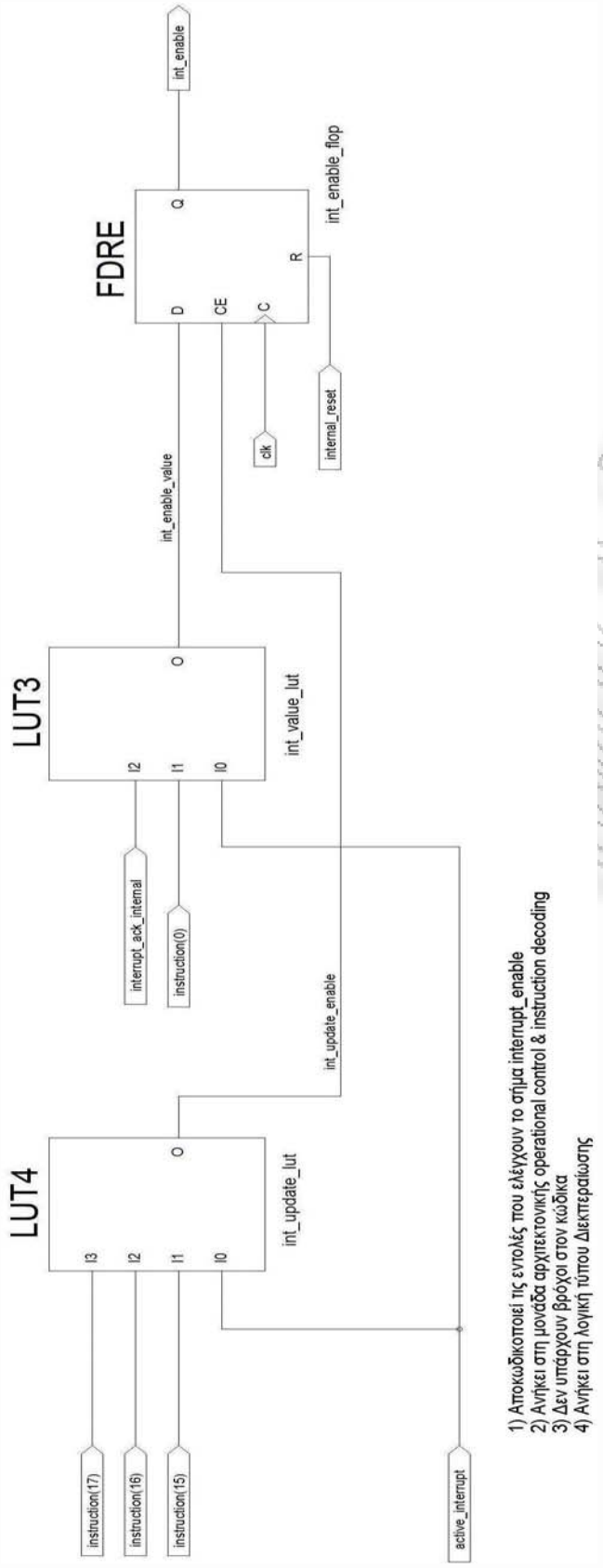
Εικόνα 4.1 : T_state & internal_reset



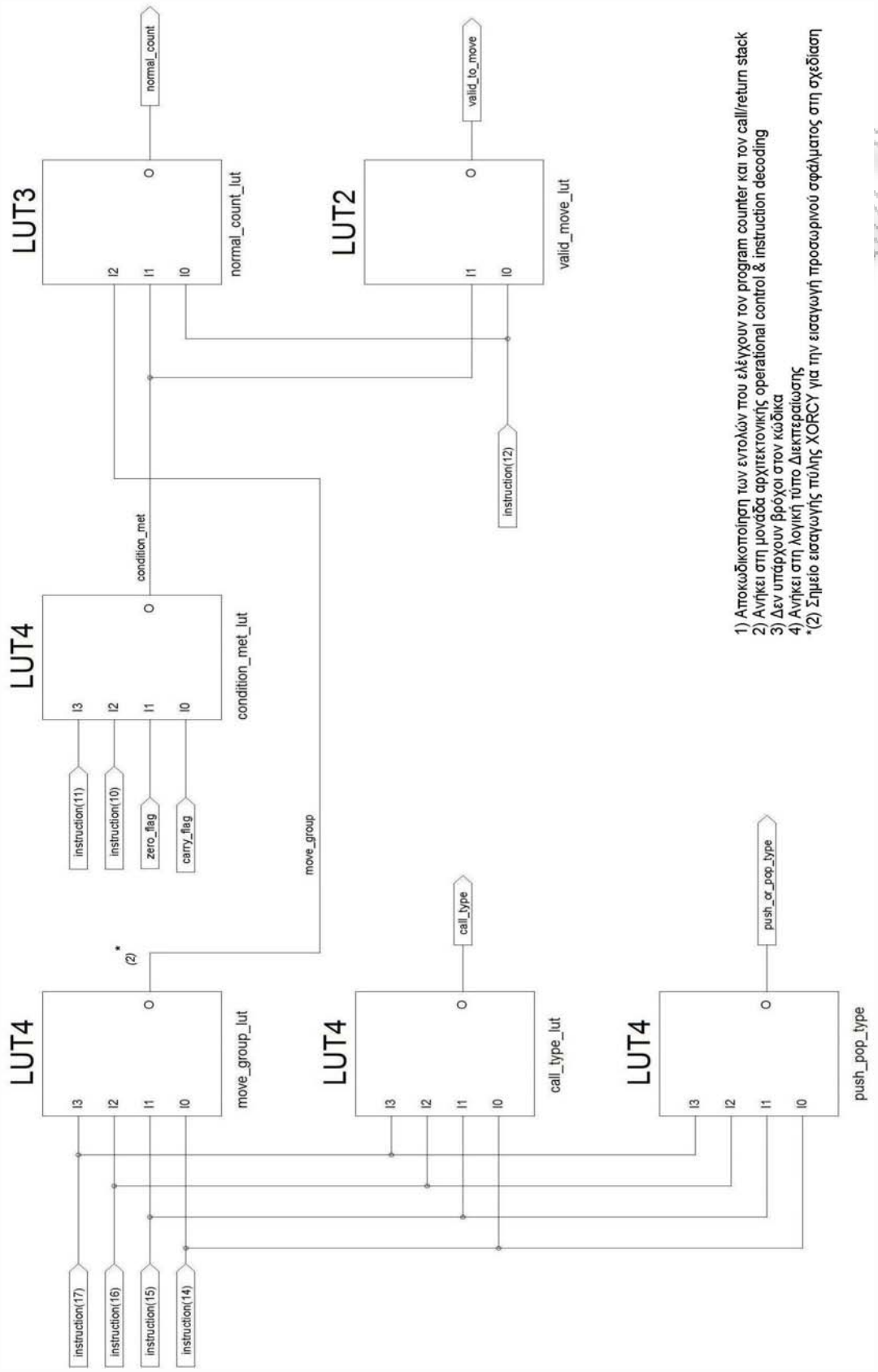
Εικόνα 4.2 : Interrupt Capture



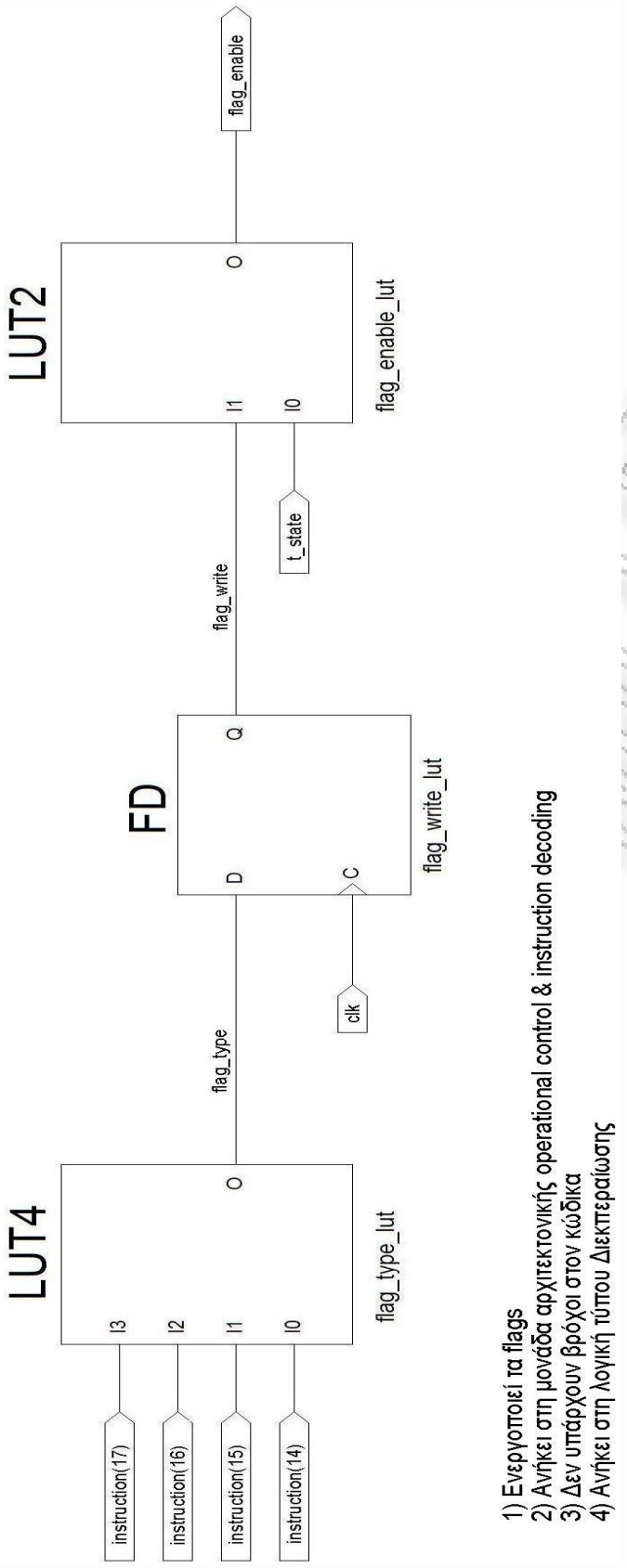
Εικόνα 4.3 : Shadow flags



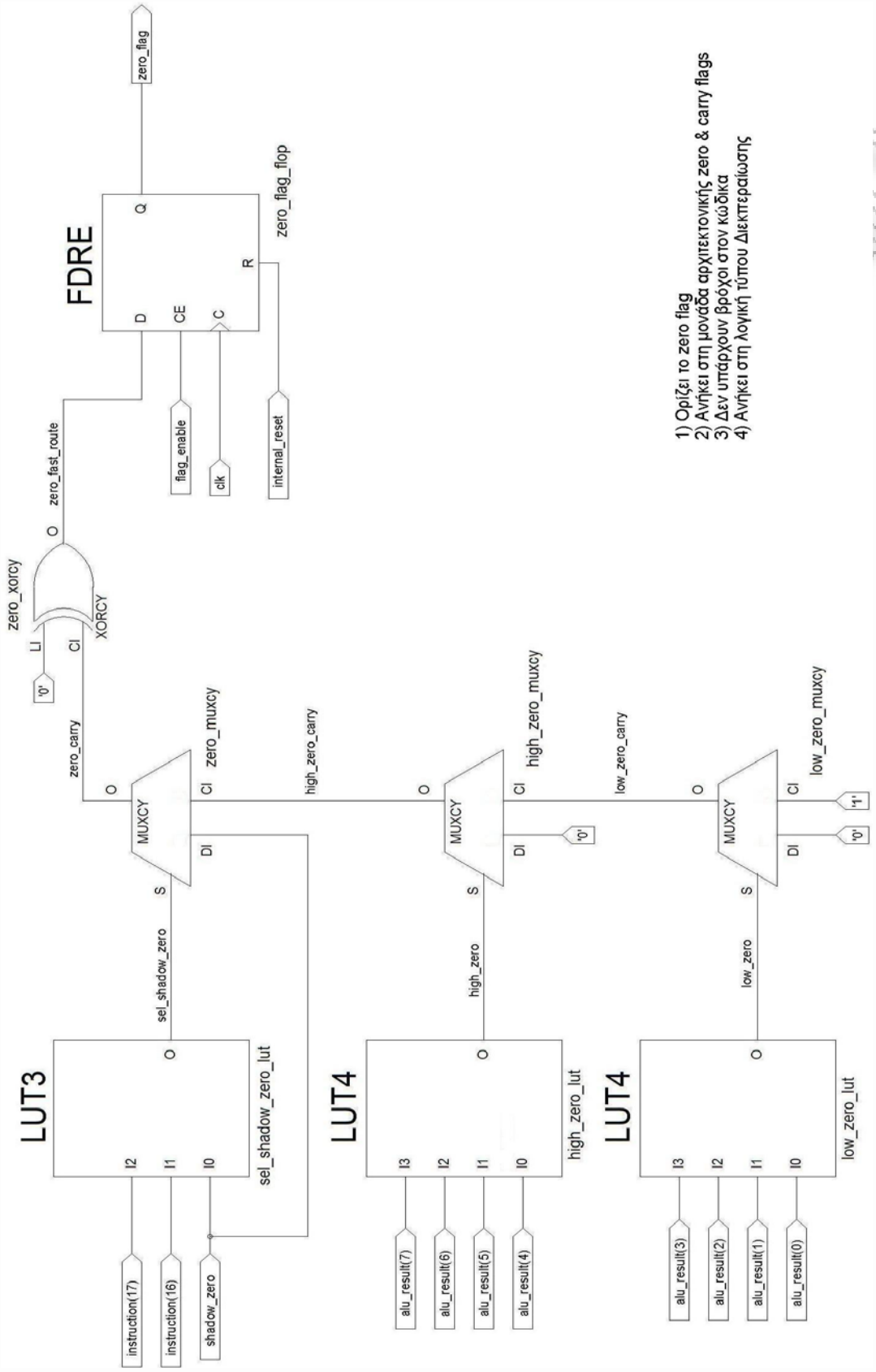
Εικόνα 4.4 : Decode instructions that set or reset interrupt enable



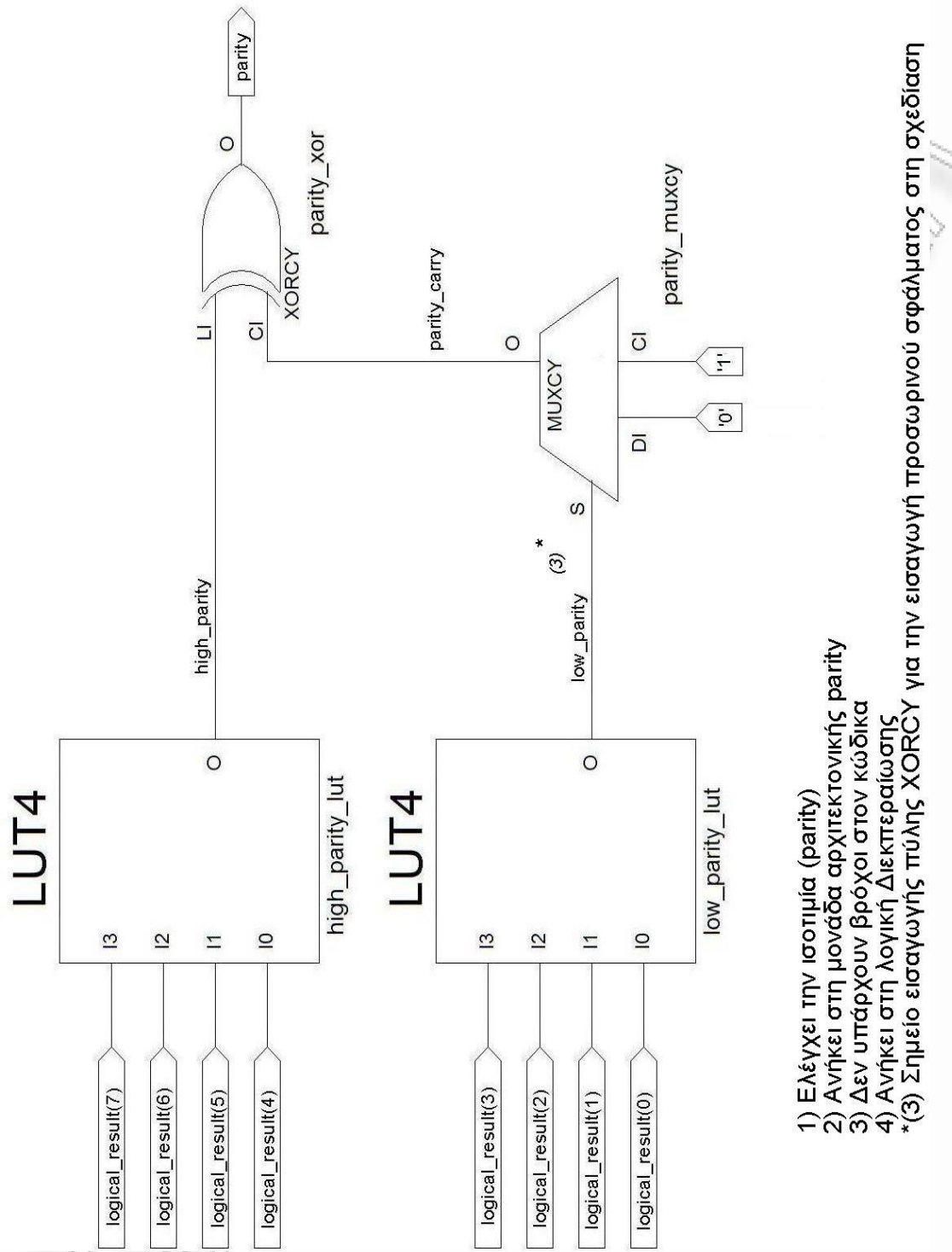
Εικόνα 4.5 : Decodes for the control of the program counter & Call / Return Stack



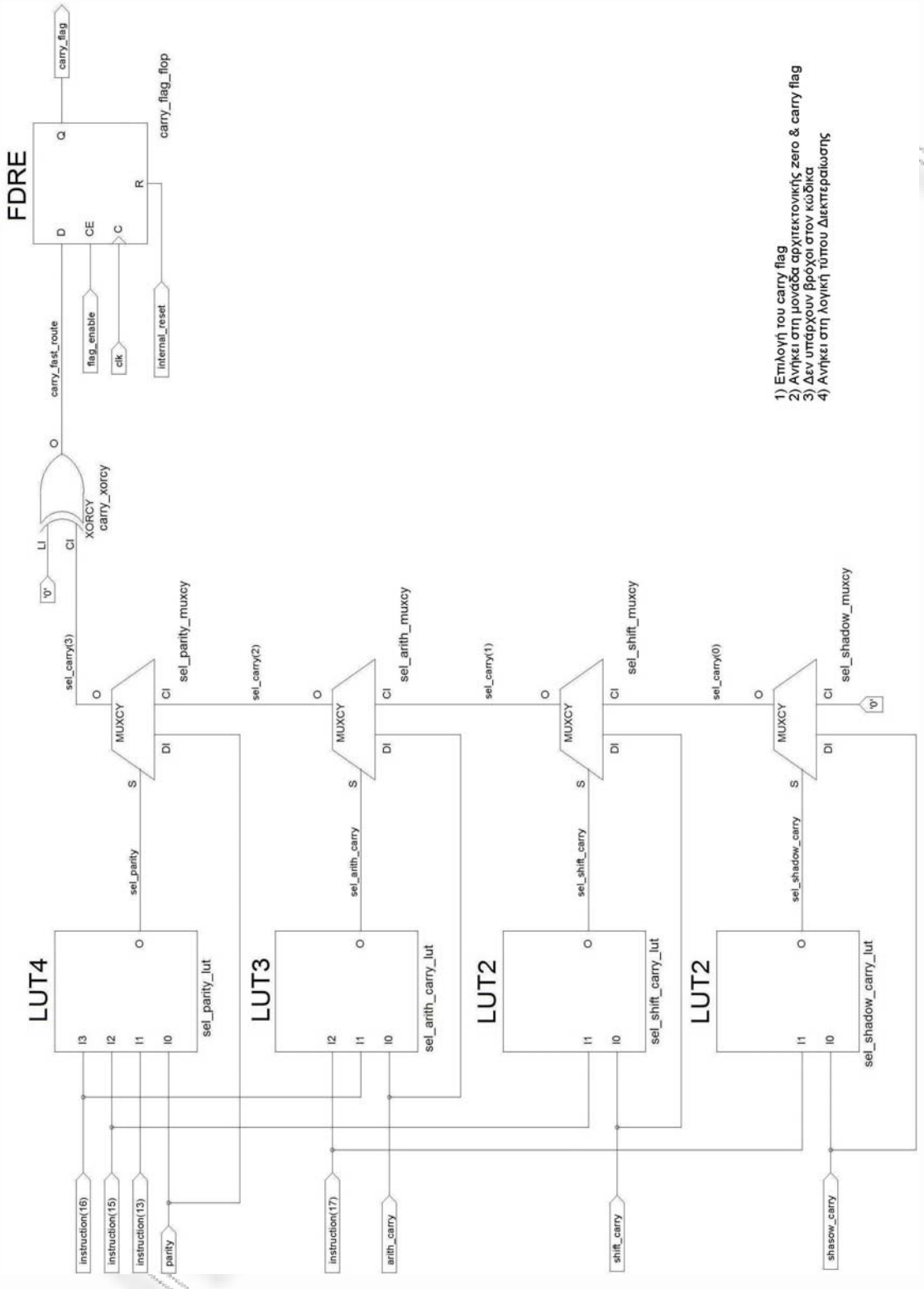
Εικόνα 4.6 : Enable for flags



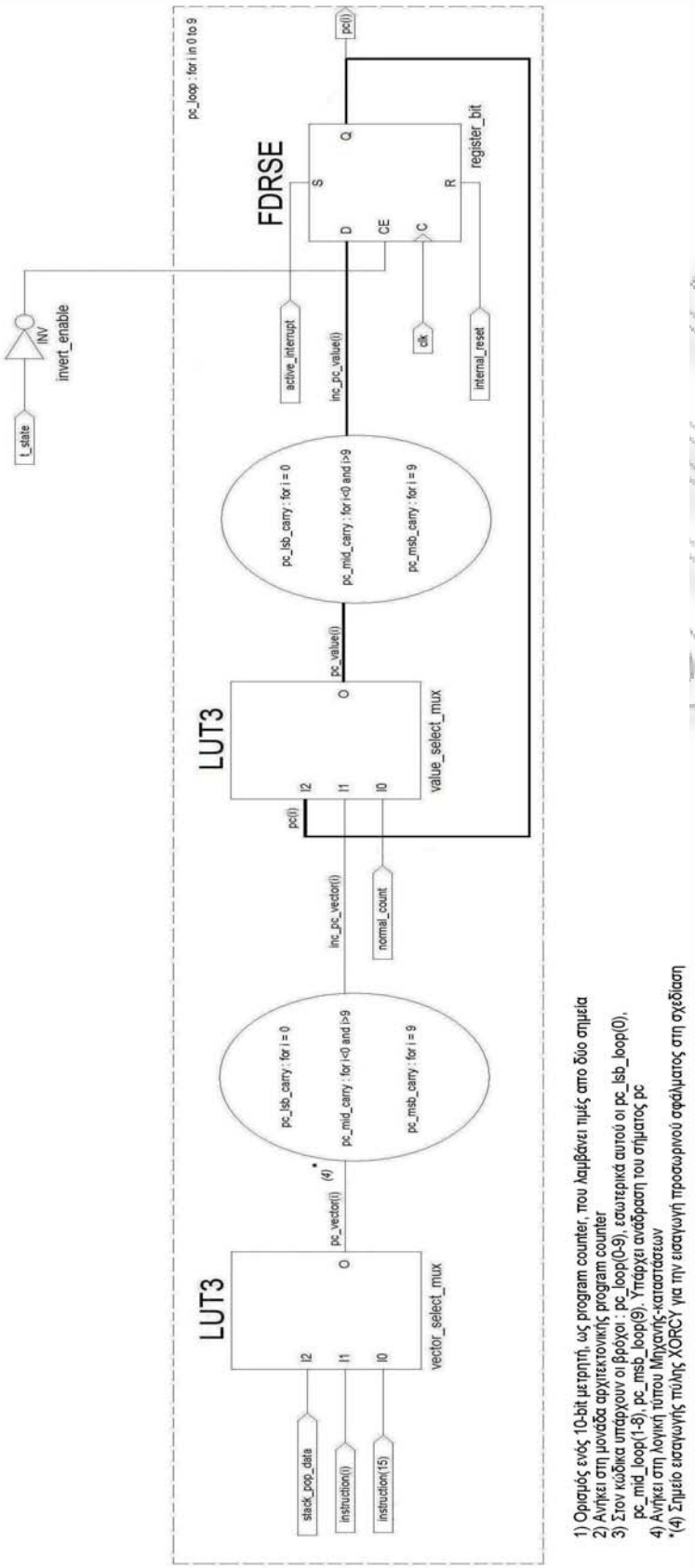
Εικόνα 4.7 : Zero flag



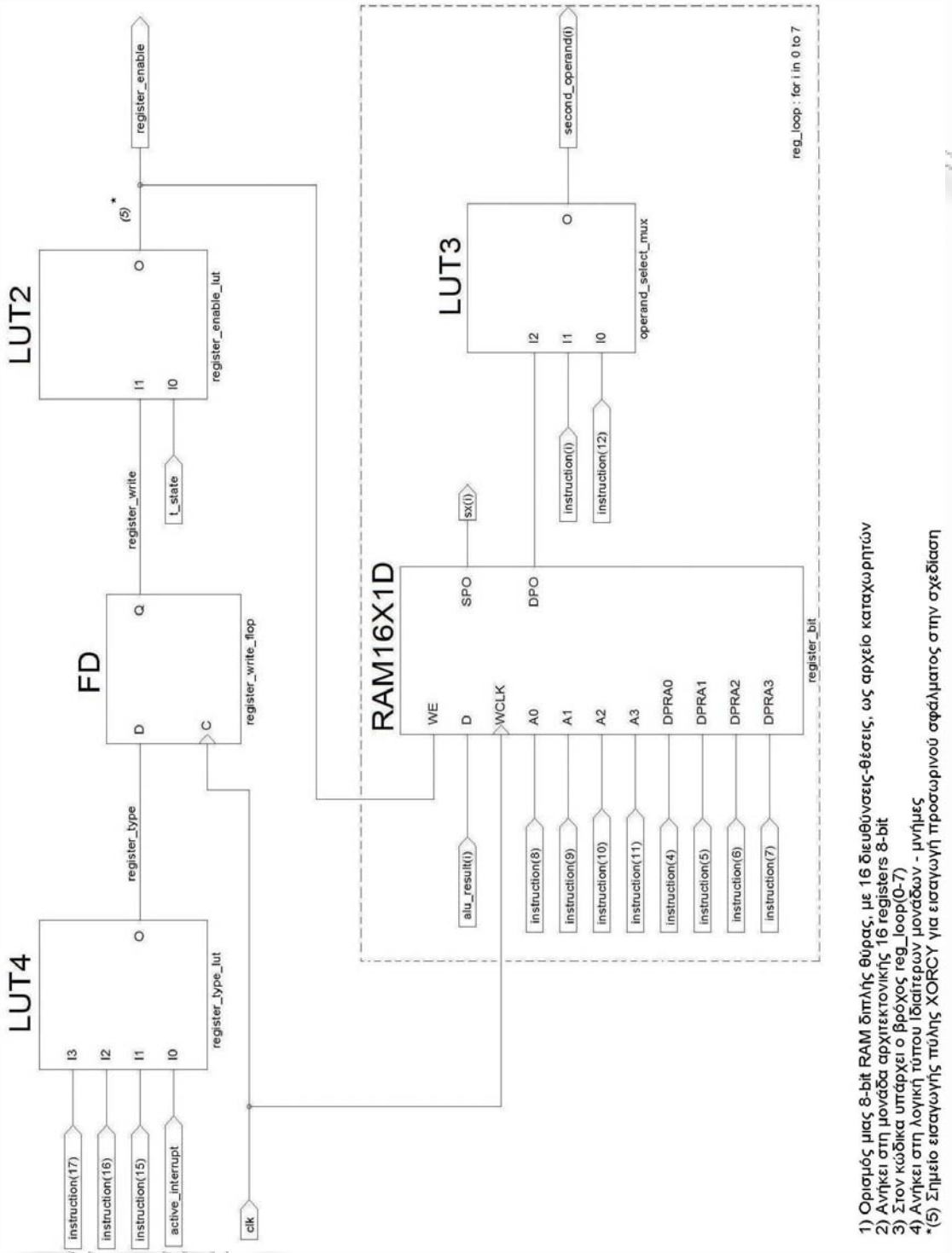
Εικόνα 4.8 : Parity detection



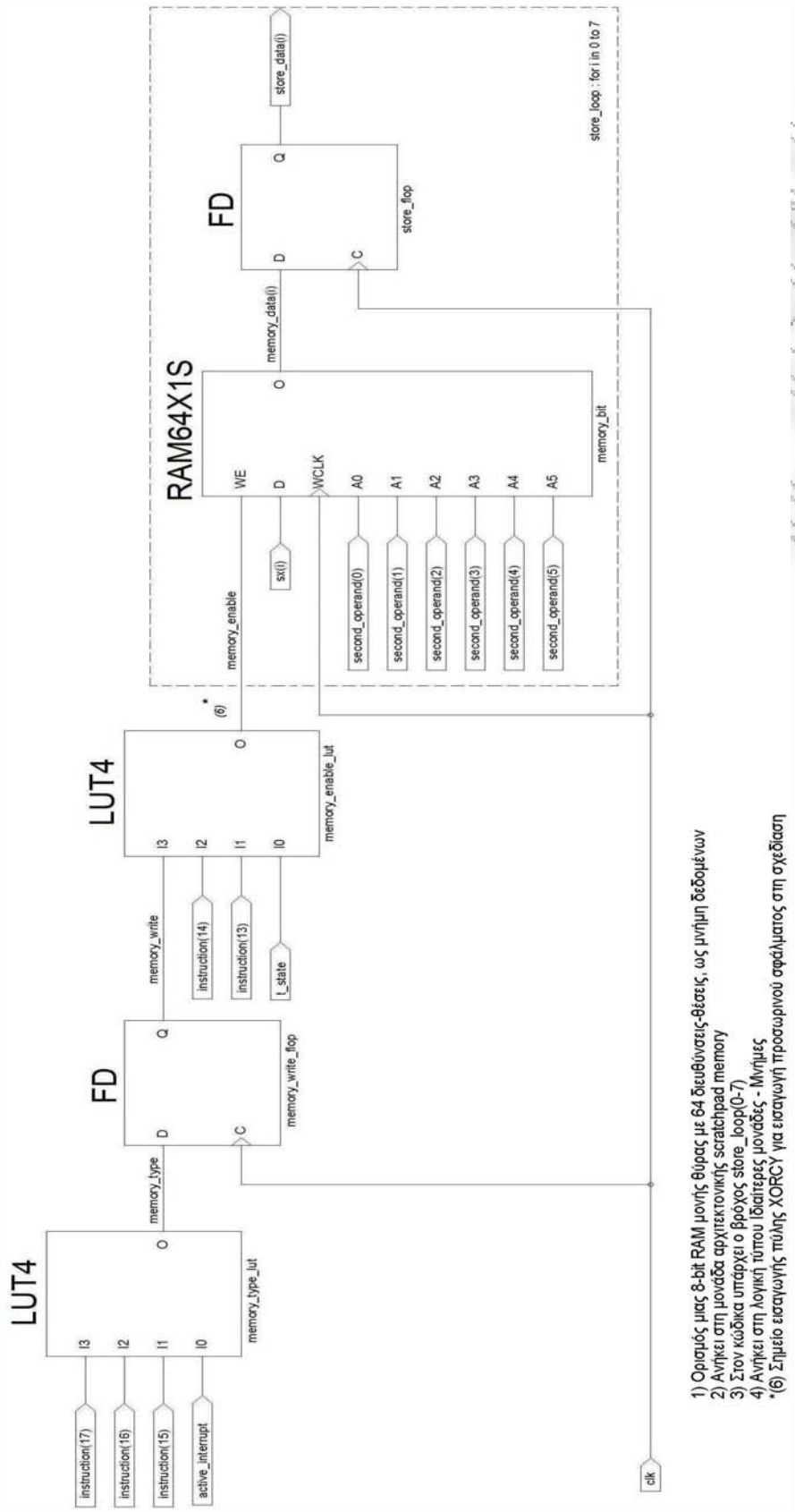
Εικόνα 4.9 : Carry flag selection



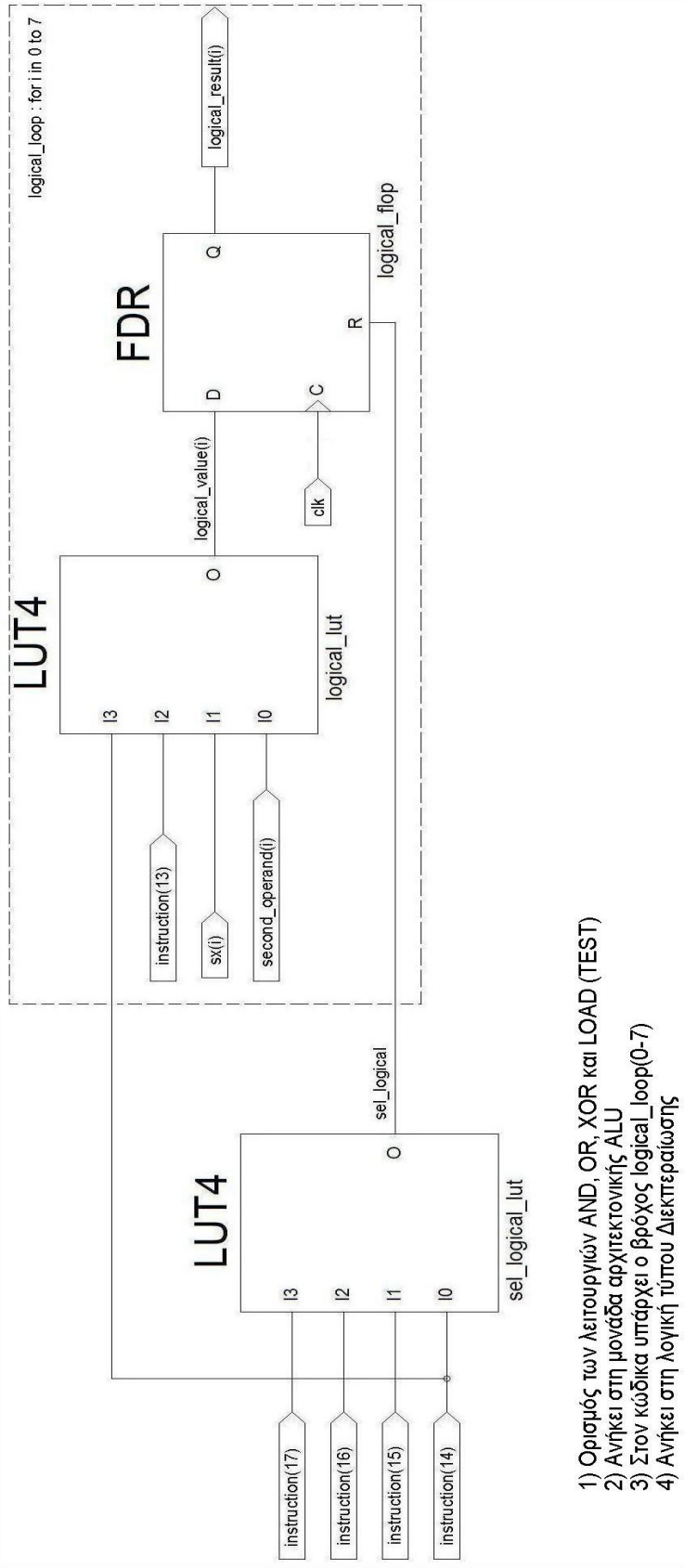
Εικόνα 4.10 : Program Counter



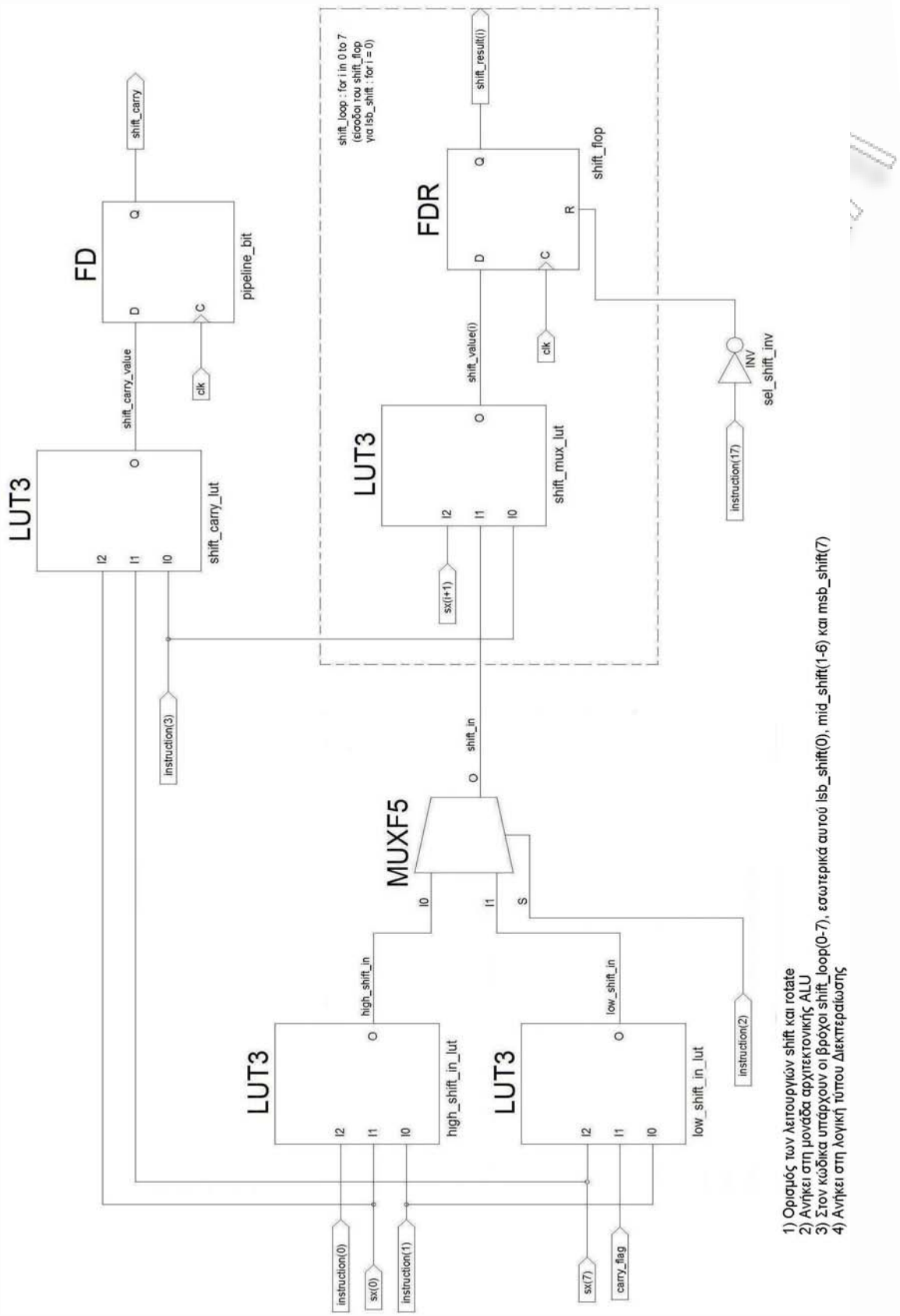
Εικόνα 4.11 : Register bank & 2nd operand selection



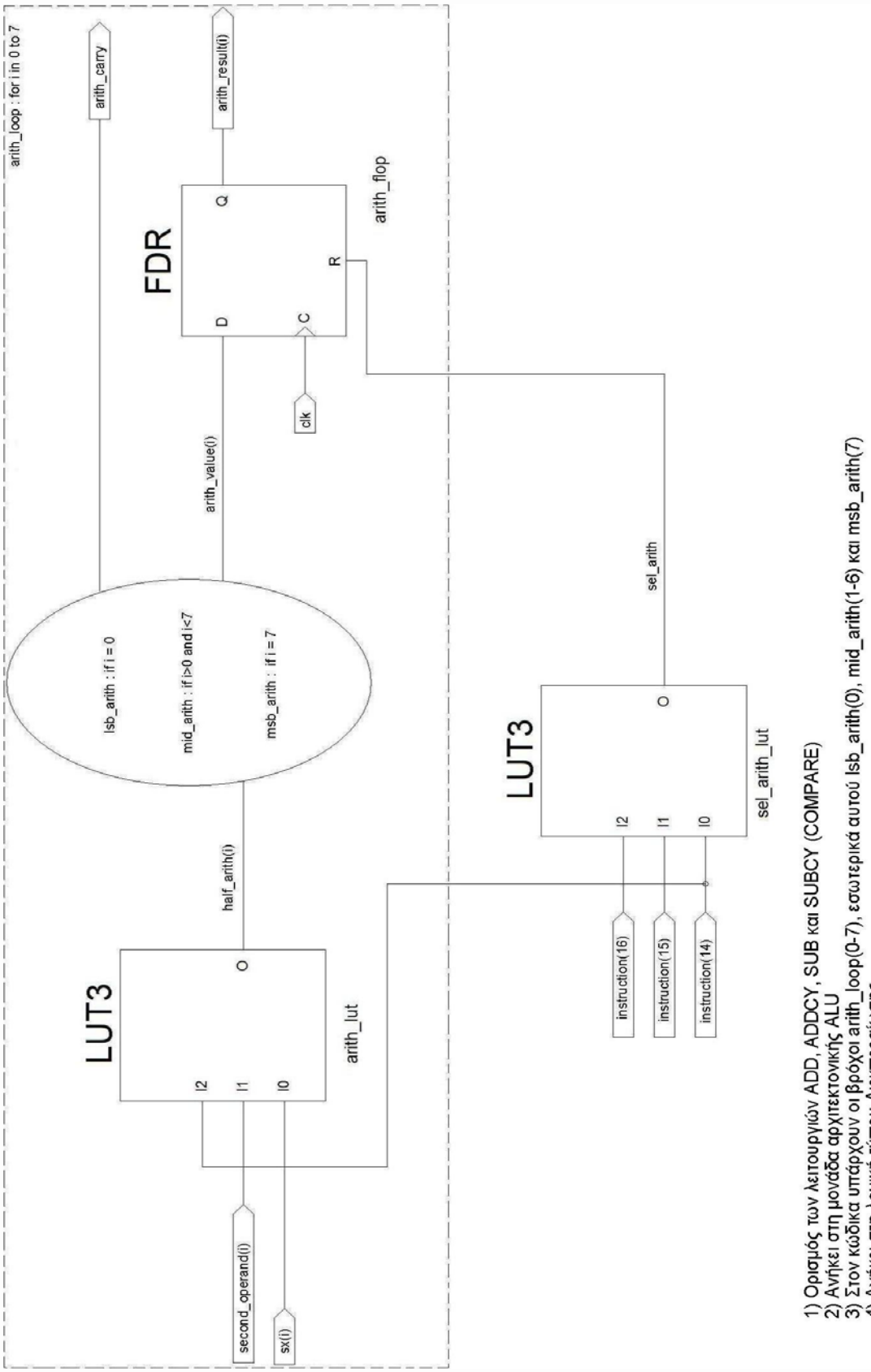
Εικόνα 4.12 : Store memory



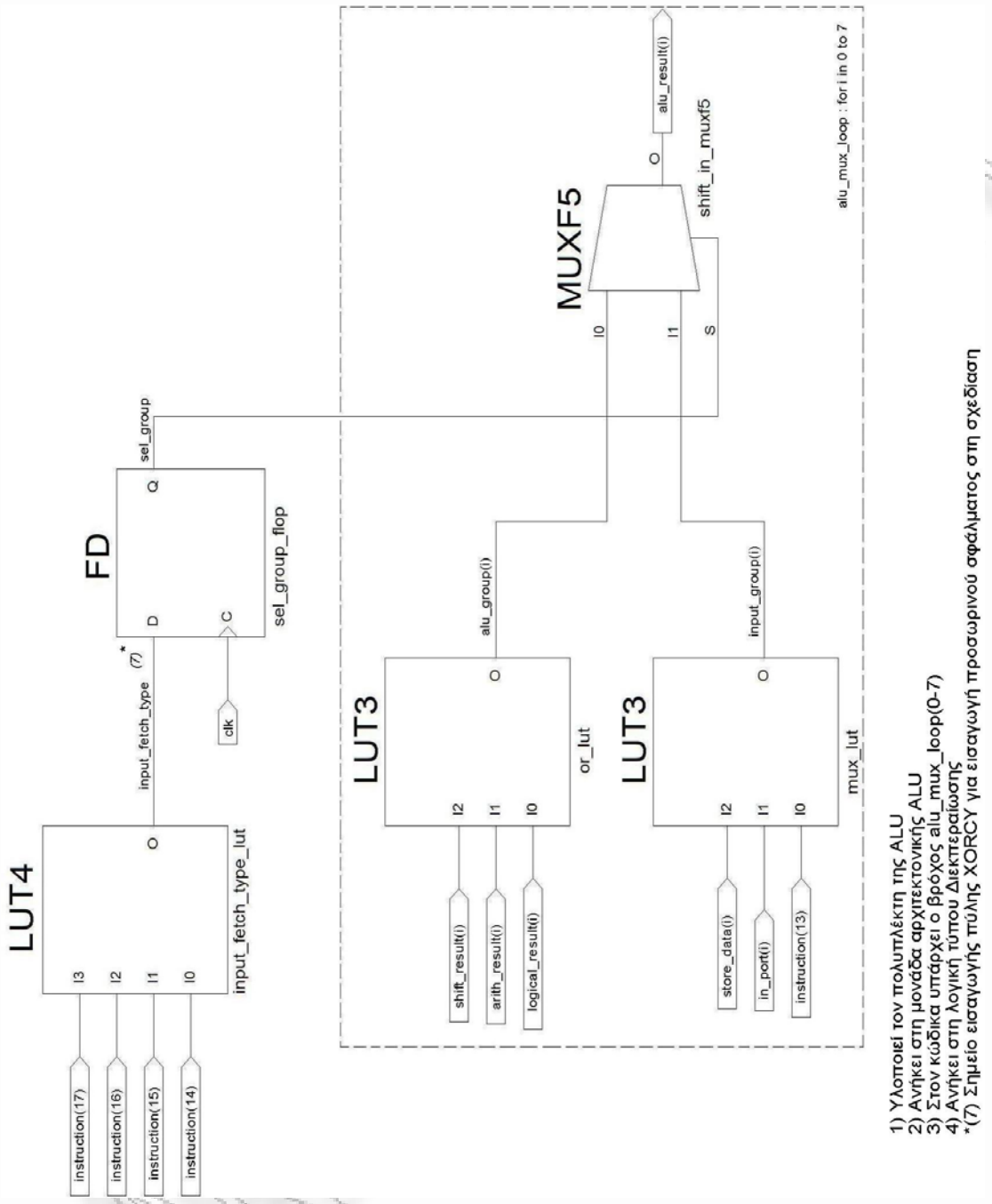
Εικόνα 4.13 : Logical operations



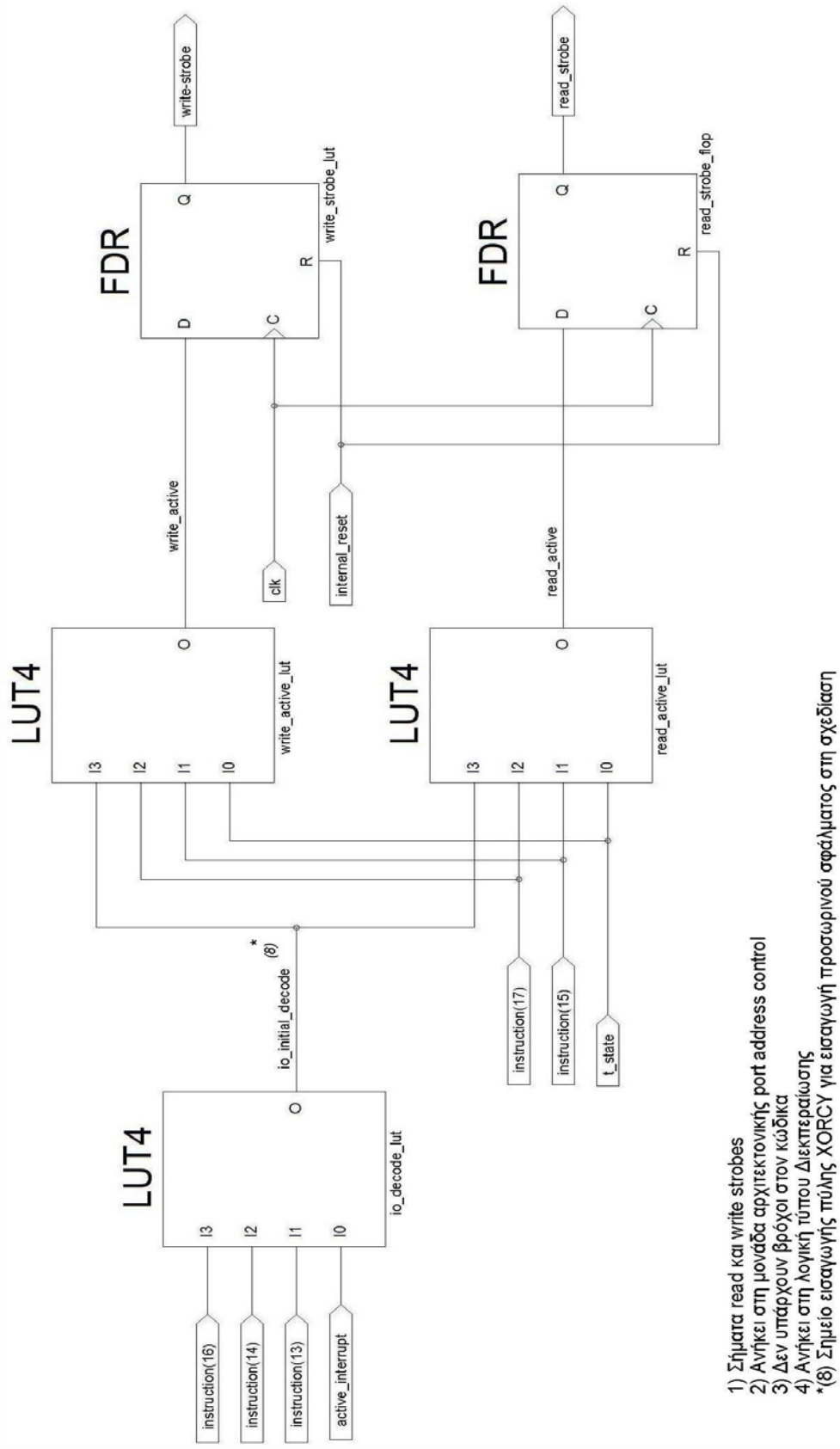
Εικόνα 4.14 : Shift – Rotate operations



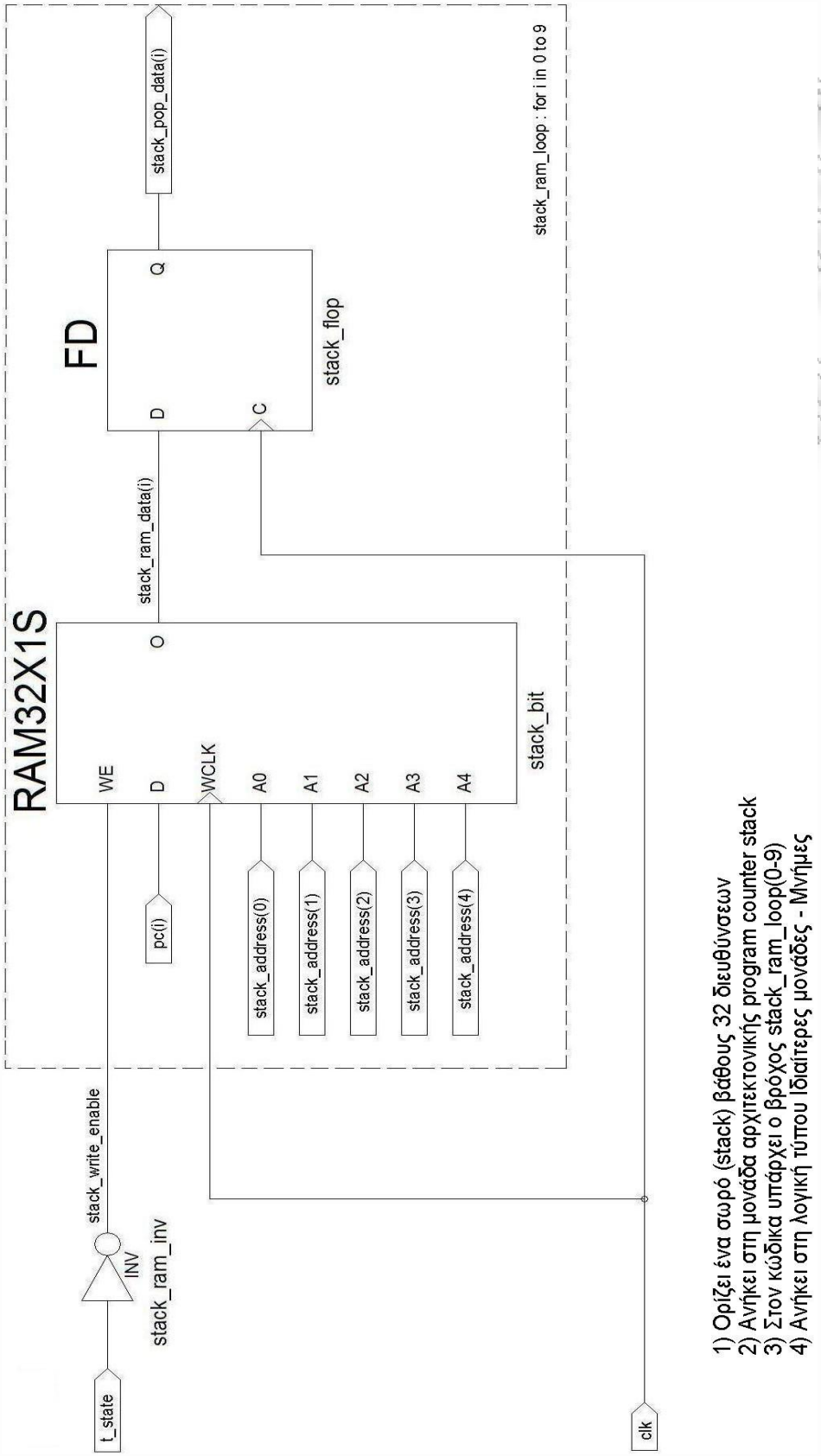
Εικόνα 4.15 : Arithmetic operations



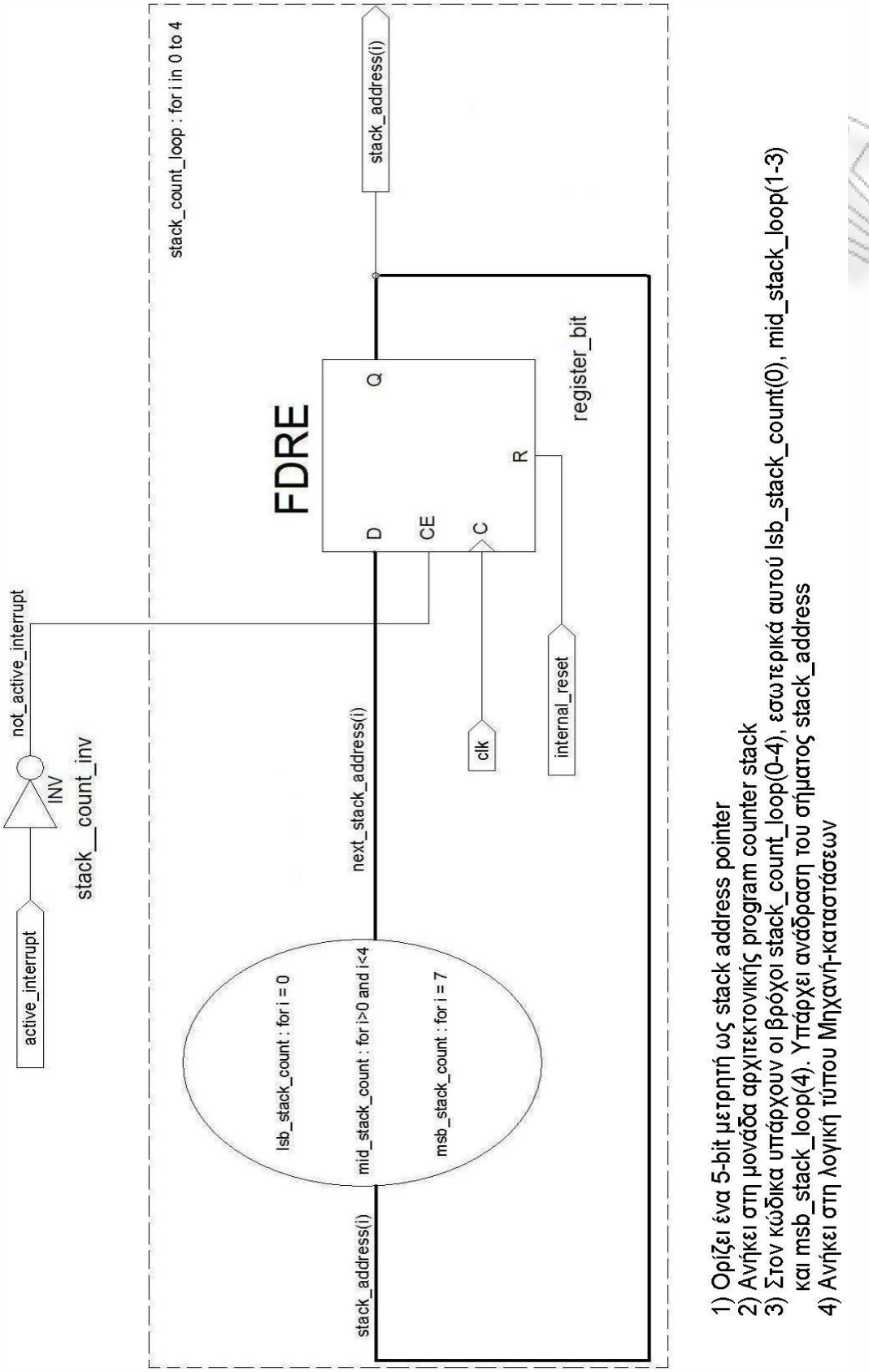
Εικόνα 4.16 : ALU multiplexer



Εικόνα 4.17 : Read – Write strobes



Εικόνα 4.18 : Program Call / Return Stack



Εικόνα 4.19 : Stack address pointer

Ο παρακάτω πίνακας δείχνει συγκεντρωτικά τις μονάδες που αποτελούν τον KCPSM3 και τις ομάδες στις οποίες ανήκουν.

Πίνακας 4.1 : Μονάδες που αποτελούν τον KCPSM3 ανά κατηγορία τύπου λογικής

Λογική Διεκπεραίωσης	• Shadow flags • Decode instructions that set or reset interrupt enable • Decodes for the control of the program counter and Call / Return Stack • Enable for flags • Zero flag • Parity detection • Carry flag selection • Logical operations • Shift-rotate operations • Arithmetic operations • ALU multiplexer • Read, Write Strobes
Μηχανές καταστάσεων	• t-state and internal_reset • Interrupt capture • Program counter • Stack address pointer
Μνήμες	• Register bank and 2nd operand selection • Store memory • Program Call / Return Stack

4.2 Υλοποίηση

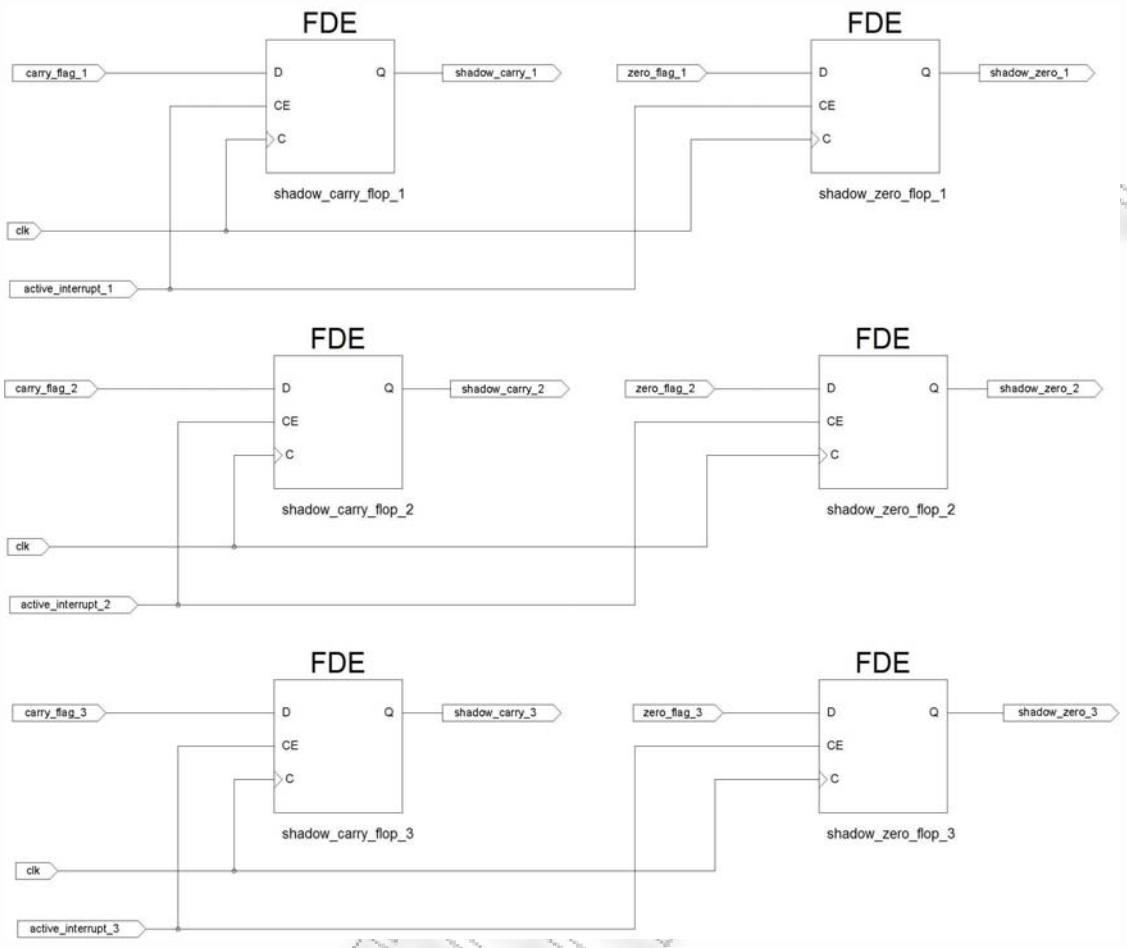
Στη συνέχεια θα περιγράψουμε αναλυτικά πως έγινε ο εσωτερικός τριπλασιασμός του PicoBlaze για κάθε επιμέρους μονάδα, σύμφωνα με την ομάδα στην οποία ανήκουν. Θα εξηγήσουμε κάποιες σχεδιαστικές αποφάσεις που πάρθηκαν και θα παρουσιάσουμε πως υλοποιήθηκαν οι voters.

Αρχικά θα διατηρήσουμε τη μορφή του KCPSM3, σε ένα αρχείο, και δεν θα χωρίσουμε τις επιμέρους μονάδες του σε ξεχωριστά αρχεία χαμηλού επιπέδου σχεδίασης. Καθώς ο τριπλασιασμένος μας κώδικας βρίσκεται σε ένα μόνο αρχείο για να είναι πιο ευανάγνωστος, θα ονομάσουμε τα αντίγραφα της εκάστοτε μονάδας replica 1, replica 2 και replica 3 (διακρίνονται με σχόλια μέσα στον κώδικα).

Όλα τα εσωτερικά σήματα του KCPSM3 τριπλασιάζονται. Έτσι όταν στον αρχικό KCPSM3 έχουμε για παράδειγμα το εσωτερικό σήμα t_state, στην TMR εκδοχή του θα έχουμε τα σήματα t_state_1, t_state_2 και t_state_3 για τα αντίγραφα replica 1, replica 2 και replica 3 αντιστοίχως.

4.2.1 Μονάδες λογικής Διεκπεραίωσης

Οι μονάδες που ανήκουν σε αυτήν την κατηγορία απλά τριπλασιάζονται. Έτσι έχουμε τρία πανομοιότητα μονοπάτια (μονάδες) που μεταφέρουν το σήμα από την προηγούμενη λογική στην επόμενη. Σε αυτές τις μονάδες δεν χρειαζόμαστε εκλογείς στις εξόδους τους καθώς η τιμή θα ψηφιστεί στην επόμενη μονάδα μηχανής καταστάσεων.



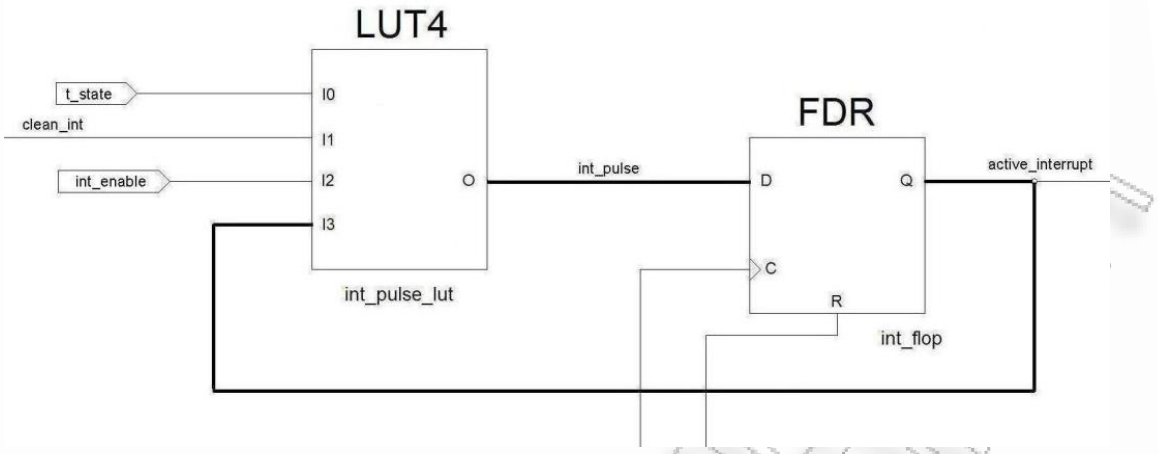
Εικόνα 4.20 : TMR εκδοχή της μονάδας λογικής διεκπεραίωσης, shadow flags

Όπως βλέπουμε και από το σχηματικό (εικόνα 4.20) δεν υπάρχει σύνδεση μεταξύ των τριών αντιγράφων μιας μονάδας λογικής διεκπεραίωσης. Τα σήματα εξόδου μεταφέρονται στις επόμενες μονάδες και η εκλογή θα γίνει στο αμέσως επόμενο στάδιο μηχανής καταστάσεων.

4.2.2 Μονάδες λογικής μηχανής καταστάσεων

Οι μονάδες αυτές τριπλασιάζονται και σε κάθε λογικό βρόχο που δημιουργείται τοποθετούμε εκλογείς, έναν σε κάθε αντίγραφο. Είσοδοι των εκλογέων είναι τα σήματα εξόδου του βρόχου. Έτσι στην έξοδό τους έχουμε τη σωστή-διορθωμένη τιμή η οποία ανατροφοδοτείται στο βρόχο. Επομένως ακόμα και στην περίπτωση που υπάρξει σφάλμα σε κάποιο από τα τρία αντίγραφα, η σωστή τιμή θα ανατροφοδοτηθεί στο βρόχο εξαλείφοντας το σφάλμα.

Για παράδειγμα, στη μονάδα Interrupt capture δημιουργείται ο παρακάτω βρόχος :



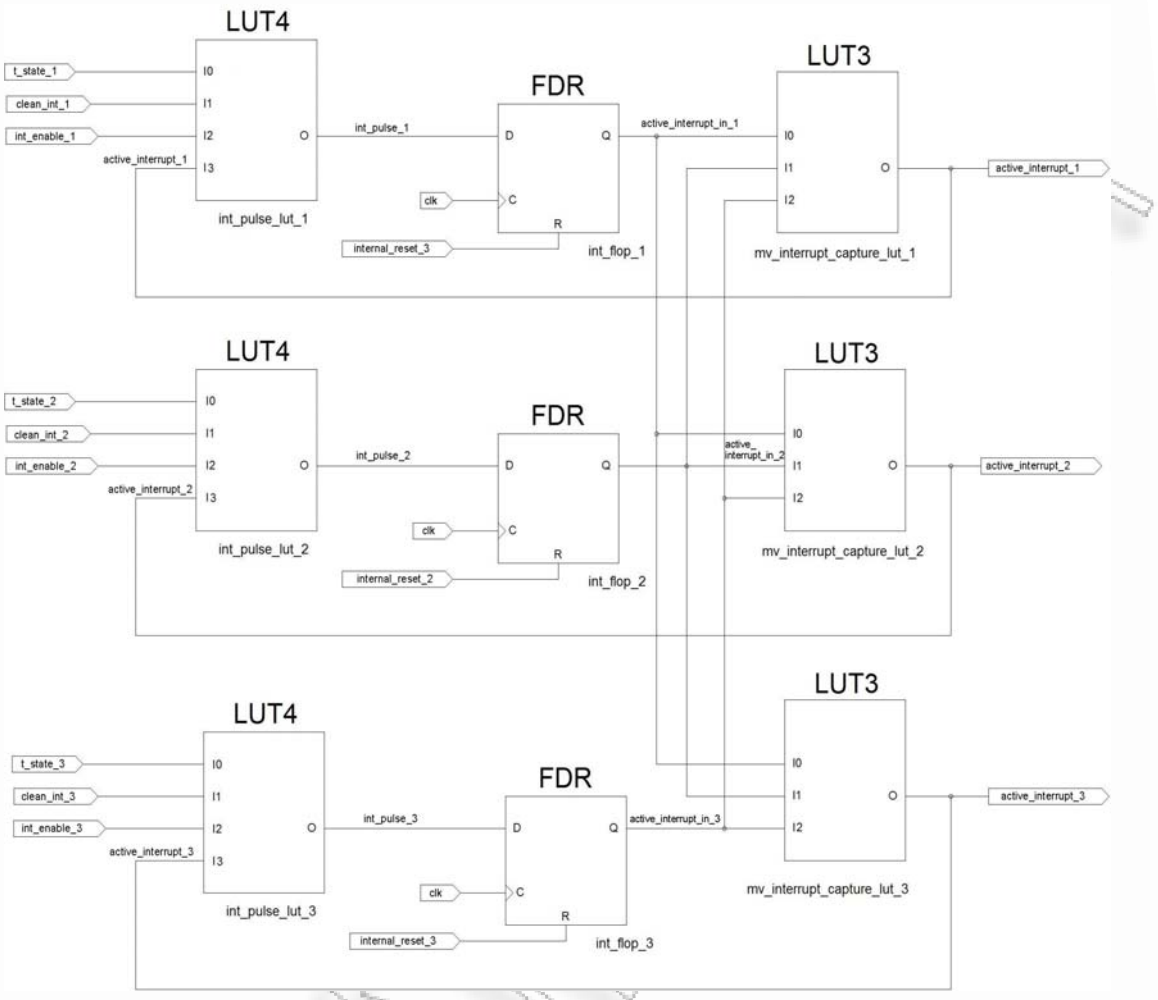
Εικόνα 4.21 : Λογικός βρόχος στη μονάδα interrupt capture

```
int_pulse_lut: LUT4
--synthesis translate_off
  generic map (INIT => X"0080")
--synthesis translate_on
port map( I0 => t_state,
          I1 => clean_int,
          I2 => int_enable,
          I3 => active_interrupt,
           O => int_pulse );

int_flop: FDR
port map ( D => int_pulse,
           Q => active_interrupt,
           R => internal_reset,
           C => clk);
```

Εικόνα 4.22 : Ο κώδικας που δημιουργεί το βρόχο στη μονάδα interrupt capture

Αντιμετωπίζουμε το τμήμα αυτό ως μηχανή καταστάσεων, οπότε τοποθετούμε τρεις voters, έναν σε κάθε αντίγραφο, πριν την ανάδραση του σήματος active_interrupt. Το υπόλοιπο τμήμα αντιμετωπίζεται ως λογική throughput και απλά τριπλασιάζεται. Στην εικόνα 4.23 φαίνεται σχηματικά η TMR εκδοχή αυτού του τμήματος και είναι σαφή τα ακριβή σημεία εισαγωγής των εκλογέων.



Εικόνα 4.23 : TMR εκδοχή της λογικής μηχανής καταστάσεων, της μονάδας Interrupt capture

Όπως βλέπουμε και από το σχηματικό (εικόνα 4.23) είναι απαραίτητο να προσθέσουμε ένα ακόμα εσωτερικό σήμα για κάθε αντίγραφο, active_interrupt_in_1, active_interrupt_in_2 και active_interrupt_in_3. Αυτά τα σήματα αντικαθιστούν τα σήματα εξόδου των int_flop_1, int_flop_2 και int_flop_3, και θα γίνουν οι είσοδοι των τριών voters (mv_interrupt_capture_lut_1, mv_interrupt_capture_lut_2, mv_interrupt_capture_lut_3), οι οποίοι στην έξοδό τους θα μας δώσουν τα ζητούμενα σήματα (active_interrupt_1, active_interrupt_2, active_interrupt_3). Στην εικόνα 4.24 φαίνεται ο κώδικας του ενός από τα τρία αντίγραφα της μονάδας interrupt capture, και ο κώδικας που υλοποιεί τους τρεις εκλογείς.

```
 --replica 3

int_capture_flop_3: FDR
port map ( D => interrupt,
           Q => clean_int_3,
           R => internal_reset_3,
           C => clk);

int_pulse_lut_3: LUT4
--synthesis translate_off
  generic map (INIT => X"0080")
--synthesis translate_on
port map( I0 => t_state_3,
          I1 => clean_int_3,
          I2 => int_enable_3,
          I3 => active_interrupt_3,
           O => int_pulse_3 );

int_flop_3: FDR
port map ( D => int_pulse_3,
           Q => active_interrupt_in_3,
           R => internal_reset_3,
           C => clk);

ack_flop_3: FD
port map ( D => active_interrupt_3,
           Q => interrupt_ack_internal_3,
           C => clk);
```

```
 -- interrupt capture majority voters

   mv_interrupt_capture_lut_1: LUT3
--synthesis translate_off
  generic map (INIT => X"E8")
--synthesis translate_on
port map( I0 => active_interrupt_in_1,
          I1 => active_interrupt_in_2,
          I2 => active_interrupt_in_3,
           O => active_interrupt_1 );

   mv_interrupt_capture_lut_2: LUT3
--synthesis translate_off
  generic map (INIT => X"E8")
--synthesis translate_on
port map( I0 => active_interrupt_in_1,
          I1 => active_interrupt_in_2,
          I2 => active_interrupt_in_3,
           O => active_interrupt_2 );

   mv_interrupt_capture_lut_3: LUT3
--synthesis translate_off
  generic map (INIT => X"E8")
--synthesis translate_on
port map( I0 => active_interrupt_in_1,
          I1 => active_interrupt_in_2,
          I2 => active_interrupt_in_3,
           O => active_interrupt_3 );
```

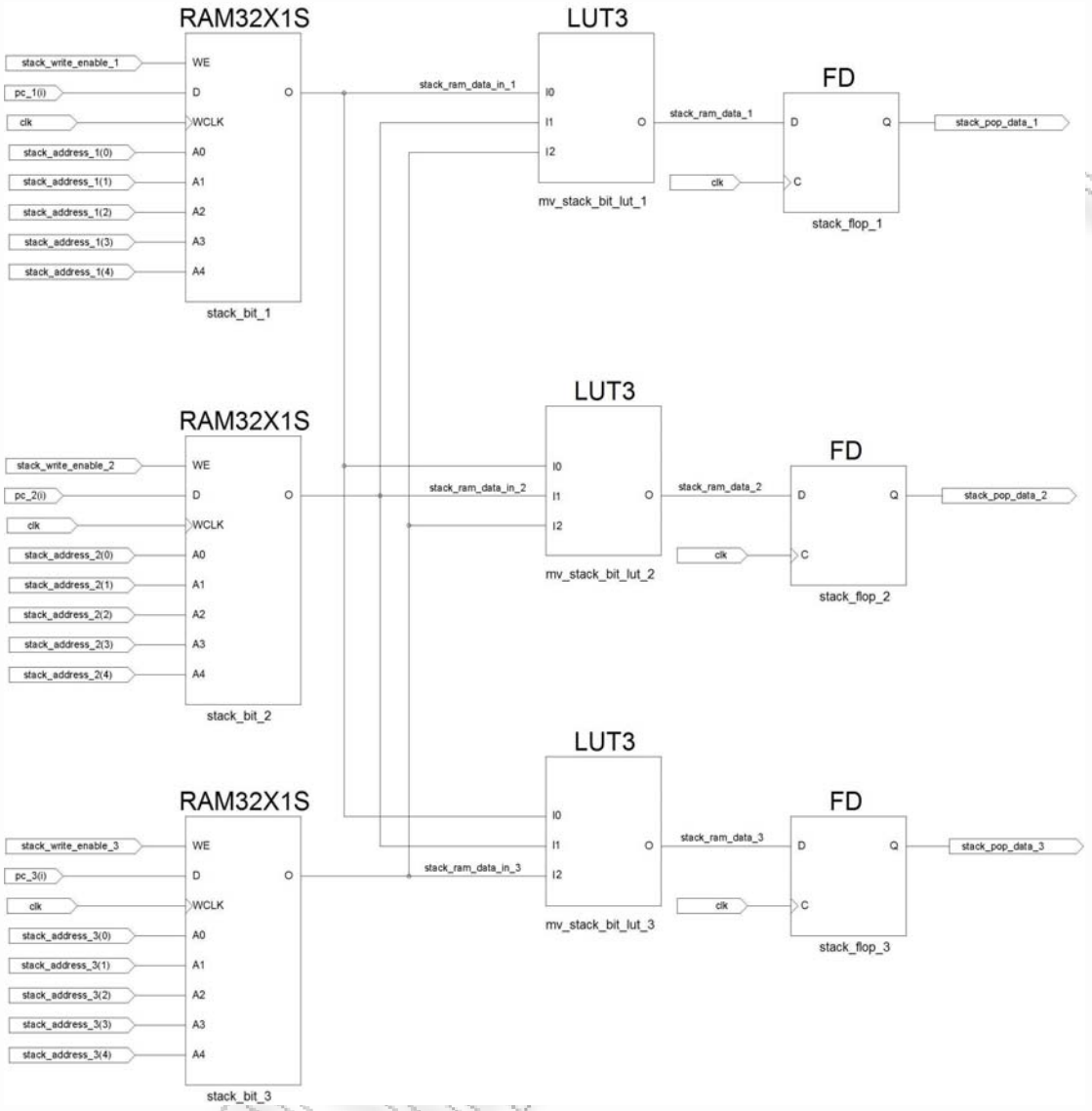
Εικόνα 4.24 : Τρίτο αντίγραφο της μονάδας Interrupt capture και οι απαραίτητοι εκλογείς

Η υλοποίηση των εκλογέων θα περιγράφει αναλυτικά στην συνέχεια αυτής της ενότητας.

4.2.3 Ιδιαίτερες μονάδες – Μνήμες

Για την TMR εκδοχή των μονάδων που ανήκουν σε αυτήν την κατηγορία εφαρμόσαμε την προσέγγιση Simple Redundancy όπως αυτή περιγράφεται στην ενότητα 2.6.1. Αυτή η μέθοδος προτιμήθηκε σε αντίθεση με την μέθοδο Redundancy & Refresh καθώς οι μνήμες που χρησιμοποιούνται στον κώδικα του KCPSM3 είναι κατανεμημένες RAM και όχι Block RAMs. Μιας και δεν θα γίνει χρήση Readback ή partial reconfiguration στη συγκεκριμένη υλοποίηση, δε θεωρήθηκε απαραίτητο να αντικατασταθούν οι μνήμες αυτές με άλλες Block RAMs, μιας και κάτι τέτοιο θα άλλαζε την αρχική σχεδίαση του KCPSM3, κάτι που δεν είναι επιθυμητό.

Επομένως, για την TMR εκδοχή των μνημών, τριπλασιάζουμε τις μνήμες και στην έξοδο κάθε αντιγράφου τοποθετούμε έναν εκλογέα. Στην εικόνα 4.25 έχουμε τη σχηματική απεικόνιση της τριπλασιασμένης μονάδας Program Call / Return Stack (το τμήμα που υλοποιεί το stack_ram_loop). Και σε αυτή την περίπτωση προσθέτουμε τρία επιπλέον εσωτερικά σήματα (stack_ram_data_in_1, stack_ram_data_in_2 και stack_ram_data_in_3) που αποτελούν τις εξόδους των μνημών. Οι είσοδοι των εκλογέων είναι οι έξοδοι αυτοί, ενώ οι έξοδοί τους μας δίνουν την πλειοψηφούσα τιμή.



Εικόνα 4.25 : TMR εκδοχή μιας μνήμης, της μονάδας program Call / Return stack

4.2.4 Είσοδοι – Έξοδοι

Στην συγκεκριμένη υλοποίηση αντιμετωπίζουμε τον KCPSM3 ως ένα μαύρο κουτί. Αυτό σημαίνει πως οι είσοδοι και οι έξοδοι της μονάδας KCPSM3 δεν τριπλασιάζονται. Ο τριπλασιασμός είναι μόνο εσωτερικός. Έτσι η κάθε είσοδος δίδεται χωρίς τριπλασιασμό και στα τρία αντίγραφα, ενώ τα σήματα που καταλήγουν σε εξόδους της μονάδας περνούν πρώτα από εκλογείς, χωρίς αυτές να τριπλασιάζονται.

Ενδεικτικά, η είσοδος interrupt, δίδεται και στα τρία αντίγραφα της μονάδας Interrupt Capture ως έχει, δηλαδή ένα σήμα ίδιο και στα τρία, όπως φαίνεται και παραπάνω, στην εικόνα 4.24 που απεικονίζει τον vhdl κώδικα για το τρίτο αντίγραφο της μονάδας interrupt capture. Η έξοδος write_strobe, δίδεται από την έξοδο ενός εκλογέα (mv_write_strobe_lut), ο οποίος έχει ως εισόδους του τις εξόδους των τριών αντιγράφων της μονάδας Read - Write Strobes, και δίνει στην έξοδό του την πλειοψηφούσα τιμή της εξόδου της μονάδας (KCPSM3).

```
-- majority voters for write & read strobes

  mv_write_strobe_lut: LUT3
  --synthesis translate_off
    generic map (INIT => X"E8")
  --synthesis translate_on
  port map( I0 => write_strobe_1,
            I1 => write_strobe_2,
            I2 => write_strobe_3,
             O => write_strobe );

  mv_read_strobe_lut: LUT3
  --synthesis translate_off
    generic map (INIT => X"E8")
  --synthesis translate_on
  port map( I0 => read_strobe_1,
            I1 => read_strobe_2,
            I2 => read_strobe_3,
             O => read_strobe );
```

Εικόνα 4.26 : Εκλογείς για τα σήματα εξόδου write_strobe και read_strobe

4.2.5 Μνήμη Εντολών

Στη μνήμη εντολών, από την οποία ο KCPSM3 διαβάζει το προς εκτέλεση πρόγραμμα, δεν εφαρμόζουμε την τεχνική TMR μιας και οι μνήμες συνήθως προστατεύονται από κώδικες ανίχνευσης και διόρθωσης σφαλμάτων. Στην περίπτωση που υλοποιούνταν θα έπρεπε να αλλάξει και ολόκληρη η δομή του assembler.

4.2.6 Εκλογείς πλειοψηφίας

Για την υλοποίηση των εκλογέων ακολουθήσαμε την ίδια λογική με την οποία είναι υλοποιημένος ο KCPSM3. Έτσι, καθώς ο KCPSM3 είναι υλοποιημένος με πρωταρχικά στοιχεία (primitives), για την υλοποίηση των εκλογέων θα χρησιμοποιήσαμε ένα πρωταρχικό στοιχείο, ένα LUT3, που θα εκτελεί τη συνάρτηση $V = AB + AC + BC$, όπως είδαμε στην παράγραφο 2.2. Έτσι θα είναι της παρακάτω μορφής (εικόνα 4.27) :

```
mojority_voter_lut: LUT3
--synthesis translate_off
  generic map (INIT => X"E8")
--synthesis translate_on
port map( I0 => input_1,
          I1 => input_2,
          I2 => input_3,
           O => output );
```

Εικόνα 4.27 : Εκλογέας πλειοψηφίας

Επομένως, για τα 1-bit σήματα, ο εκλογέας που υλοποιούμε σε VHDL είναι της παραπάνω μορφής, όπως για παράδειγμα οι εκλογείς στην εικόνα 4.26, ενώ για τα n-bit σήματα θα είναι της μορφής στην εικόνα 4.28.

```
-- majority voter for program counter - address

   address_loop: for i in 0 to 9 generate
  --
  -- Attribute to define LUT contents during implementation
  -- The information is repeated in the generic map for functional simulation
  --
  attribute INIT : string;
  attribute INIT of mv_address_lut : label is "E8";
  --
  begin

  mv_address_lut: LUT3
  --synthesis translate_off
    generic map (INIT => X"E8")
  --synthesis translate_on
  port map( I0 => pc_1(i),
            I1 => pc_2(i),
            I2 => pc_3(i),
             O => pc(i) );

   end generate address_loop;
```

Εικόνα 4.28 : Εκλογέας για n-bit σήματα (σήμα pc, n=10)

4.3 Assembler

Σε αυτό το σημείο έχουμε ήδη αναλύσει πως προέκυψε και υλοποιήθηκε η TMR έκδοση του PicoBlaze και είναι σημαντικό να δούμε λίγο πιο αναλυτικά πως λειτουργεί ο assembler από τον οποίο και προκύπτει το αρχείο μνήμης που θα συνδέσουμε στον KCPSM3..

Για τον PicoBlaze παρέχονται τρία διαφορετικά περιβάλλοντα ανάπτυξης. Δύο από αυτά μας τα προσφέρει η Xilinx, το Xilinx KCPSM3 και Xilinx System Generator, ενώ το τρίτο από την Mediatronix, το Mediatronix pBlazIDE. Έχουμε επιλέξει να χρησιμοποιήσουμε το περιβάλλον της Xilinx, KCPSM3 το οποίο είναι και ο assembler που έχει δημιουργηθεί από τον δημιουργό του PicoBlaze (ο KCPSM3 assembler παρέχεται στο φάκελο με τον κώδικα του KCPSM3).

4.3.1 KCPSM3 Assembler

Ο KCPSM3 assembler παρέχεται ως ένα απλό εκτελέσιμο DOS αρχείο και συνοδεύεται από τρία αρχεία φόρμες. Τα αρχεία αυτά, KCPSM3.EXE, ROM_form.vhd, ROM_form.v και ROM_form.coe πρέπει να αντιγραφούν στο φάκελο εργασίας μας.

Χρησιμοποιούμε μια PC-format εφαρμογή για να γράψουμε το προς εκτέλεση πρόγραμμα και το σώζουμε με την προέκταση .psm προσέχοντας το όνομα του αρχείου μας να μην ξεπερνά τους οκτώ χαρακτήρες. Στη συνέχεια ανοίγουμε ένα παράθυρο DOS, βρίσκουμε το φάκελό μας και για να μεταγλωττίσουμε το πρόγραμμά μας πληκτρολογούμε την εντολή :

kcpsm3 <όνομα>[.psm]

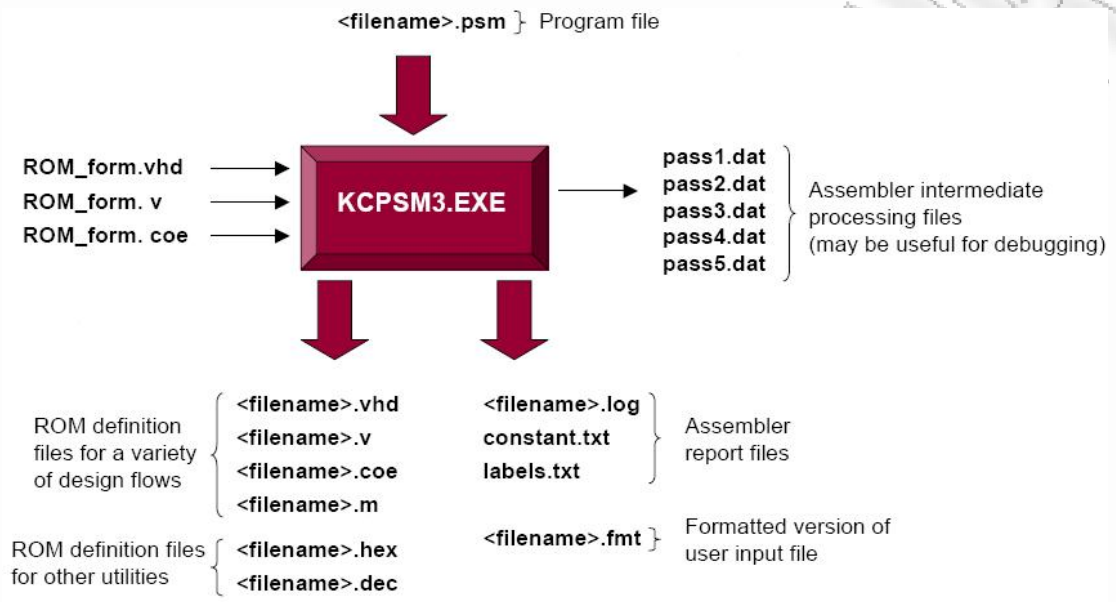
Ο assembler σε περίπτωση λάθους σταματάει και εμφανίζει ένα μικρό μήνυμα που υποδεικνύει το λάθος που υπάρχει καθώς επίσης και τη γραμμή στην οποία βρίσκεται. Έτσι διορθώνουμε το λάθος και ξανατρέχουμε τον assembler (αυτό γίνεται για κάθε λάθος).

Καθώς η εκτέλεση του προγράμματος είναι πολύ γρήγορη, δε βλέπουμε την πρόοδο του και το αποτέλεσμα είναι άμεσο. Για να δούμε ότι έχει γραφτεί από τον assembler, στέλνουμε το αποτέλεσμα της μεταγλώττισης στο αρχείο :

kcpsm3 <όνομα>[.psm] > screen_dump.txt

4.3.2 Αρχεία εισόδου και εξόδου

Ο KCPSM3 διαβάζει 4 αρχεία εισόδου και δημιουργεί 15 αρχεία εξόδου. Διαβάζει το αρχείο με το πρόγραμμα που γράψαμε (.psm) και τα αρχεία με τις φόρμες που αρχικοποιούν μια block RAM σε διάφορα πρότυπα σχεδίασης, ROM_form.*.



Εικόνα 4.29 : Αρχεία του KCPSM3 Assembler

Τα αρχεία εισόδου του assembler είναι:

- **<filename>.psm :** το πρόγραμμά μας
- **ROM_form.vhd :** μας παρέχει τη φόρμα για το vhdl αρχείο που δημιουργεί ο assembler για την μνήμη εντολών. Καθορίζει μια Block RAM μονής θύρας για Spartan-3, Virtex-II(PRO) αρχικοποιημένη ως μια ROM. Ο assembler διαβάζει τη φόρμα και αντιγράφει τις πληροφορίες στο αρχείο εξόδου <filename>.vhd. (δεν υπάρχει έλεγχος σύνταξης οπότε οποιαδήποτε αλλαγή είναι υπ' ευθύνη του χρήστη)
- **ROM_form.v :** η φόρμα για τη μνήμη σε Verilog
- **ROM_form.coe :** η φόρμα για τη μνήμη για το Core Generator

Τα αρχεία εξόδου του assembler επανεγράφονται κάθε φορά που εκτελείται, και είναι τα εξής :

- **<filename>.vhd , <filename>.v , <filename>.coe , <filename>.m :** Αν το πρόγραμμά μας δεν έχει λάθη, ο assembler δημιουργεί αυτά τα αρχεία με τον object code του προγράμματος για διάφορα πρότυπα σχεδίασης, ακολουθώντας τα αρχικά αρχεία με τις φόρμες. Αυτά τα αρχεία αρχικοποιούν τη μνήμη εντολών, εμπεριέχοντας τον object code του προγράμματος. Όπως αναφέραμε και στην παράγραφο 3.5.1, το αρχείο vhd που δημιουργεί ο assembler για τη μνήμη εντολών ονομάζεται σύμφωνα με το όνομα του αρχείου προγράμματός μας.
- **<filename>.hex & <filename>.dec :** παρέχουν τα περιεχόμενα της μνήμης εντολών σε δεκαεξαδική και δεκαδική μορφή, χρήσιμα για μετατροπή σε άλλα πρότυπα που δεν υποστηρίζονται από τον assembler.
- **<filename>.fmt :** το αρχικό πρόγραμμα σε "τακτοποιημένη" μορφή. Τυποποιεί τα label και τα σχόλια, αλλάζει όλες τις εντολές και τις δεκαεξαδικές σταθερές σε κεφαλαία και βάζει τα κατάλληλα κενά.

- **<filename>.log :** πληροφορίες για τη μεταγλώττιση. Βλέπουμε πως χρησιμοποιείται κάθε εντολή και directive. Καθορίζονται οι διευθύνσεις και ο op-code που σχετίζεται με κάθε γραμμή του προγράμματος καθώς επίσης και οι πραγματικές τιμές των διευθύνσεων και σταθερών που ορίζονται από labels.
- **constant.txt & labels.txt :** λίστες με τις σταθερές και τα labels του προγράμματος.
- **pass*.dat :** εσωτερικά αρχεία του assembler που αναπαριστούν διάφορα στάδια της μεταγλώττισης. Αγνοούνται αλλά σε κάποιες περιπτώσεις είναι χρήσιμα στο να δούμε πως ο assembler ερμηνεύει το συντακτικό του προγράμματος. Σε περίπτωση λάθους κατά τη μεταγλώττιση, τα αρχεία αυτά περιέχουν επιπλέον πληροφορίες.

4.3.3 Ψευδοεντολές

Ο assembler υποστηρίζει τρία είδη ψευδοεντολών :

- **Ψευδοεντολές σταθερών (Constant directives) :** τις χρησιμοποιούμε για να αναθέσουμε μια διψήφια δεκαεξαδική τιμή σ' ένα label. Μπορούμε να δηλώσουμε έτσι σταθερές όπως πόρτες ή διευθύνσεις αποθήκευσης και τιμές που χρησιμοποιούνται στο πρόγραμμα. Είναι global.

```
CONSTANT        name_port, 03
```

- **Ψευδοεντολές ονομάτων (Namereg directives) :** τις χρησιμοποιούμε για να αναθέσουμε νέο όνομα σε οποιονδήποτε από τους 16 καταχωρητές. (applied in-line)

```
NAMEREG         sD, new_name
```

- **Ψευδοεντολές διευθύνσεων (Address directives) :** είναι ένας τρόπος για να αναγκάσουμε τον assembler να εκτελέσει μια εντολή σε μια συγκεκριμένη διεύθυνση. Χρήσιμο για subroutines.

```
ADDRESS         3FF
```

4.4 Εφαρμογή - Έλεγχος

Για τη σύνθεση και υλοποίηση της σχεδίασης χρησιμοποιήσαμε το περιβάλλον ISE (Integrated Software Environment) 9.2.04i της Xilinx και για να αποδείξουμε την αποτελεσματικότητα του ενισχυμένου TMR PicoBlaze να προστατεύεται από σφάλματα χρησιμοποιήσαμε μια συσκευή FPGA Spartan-3, XC3S00 [7], [8], [9]. Για τον έλεγχο της ορθής λειτουργίας του χρησιμοποιήσαμε τα switches και τα leds που βρίσκονται στο Spartan-3.

4.4.1 Έλεγχος λειτουργίας

Πρώτο βήμα ήταν να ελέγξουμε τη σωστή λειτουργία της TMR εκδοχής του PicoBlaze που υλοποιήσαμε. Για το σκοπό αυτό γράψαμε τρία απλά προγράμματα σύμφωνα με το σύνολο εντολών που υποστηρίζει ο PicoBlaze και το συντακτικό του KCPSM3 assembler.

Το πρώτο πρόγραμμα (εικόνα 4.30) διαβάζει τις τιμές των switches που βρίσκονται στο Spartan-3, σε έναν καταχωρητή, s0, στην είσοδό του (in_port) και μετά διαβάζει τον καταχωρητή αυτόν ως έξοδο (out_port) στις leds.

```
         ;First example
         ;
         ;
         ; Port Definitions
         ;
         CONSTANT switches, 02       ; port for switches  ---- INPUT
         CONSTANT leds, 04           ; port for LED output  -- OUTPUT
         ;
         ;
         ; Main program
start : LOAD     s0, 00              ; initial delay
loop  : INPUT    s0, switches
        OUTPUT   S0, leds
        JUMP     Z, loop
```

Εικόνα 4.30 : Πρώτο παράδειγμα, ex1.psm

Η διαδικασία που ακολουθήσαμε για την δημιουργία του project περιγράφεται στα παρακάτω βήματα :

1. Δημιουργούμε ένα φάκελο, C:\TMR_PicoBlaze_asm, στον οποίο αντιγράφουμε τα αρχεία του assembler (KCPSM3.EXE, ROM_form.v, ROM_form.vhd, ROM_form.coe) και το πρόγραμμα προς μεταγλώττιση, ex1.psm. Κάθε φορά που μεταγλωττίζουμε το πρόγραμμα, θα αποθηκεύονται σε αυτόν το φάκελο όλα τα αρχεία που παράγει ο assembler.
2. Μεταγλωττίζουμε το πρόγραμμα. Ανοίγουμε ένα παράθυρο DOS, όπως περιγράψαμε στην ενότητα 4.3.1, και στο φάκελο εργασίας που δημιουργήσαμε προηγουμένως μεταγλωττίζουμε το πρόγραμμά μας. Καθώς ο κώδικας δεν έχει λάθη η μεταγλώττιση γίνεται κανονικά και ο assembler παράγει τη μνήμη, EX1.vhd, και τα υπόλοιπα 14 αρχεία.
3. Δημιουργούμε ένα καινούργιο project στο ISE, C:\TMR_PicoBlaze_ex1, με τα αρχεία kcpsm3.vhd, που περιέχει την TMR εκδοχή του PicoBlaze, το embedded_kcpsm3.vhd, που είναι το top level αρχείο της εφαρμογής μας όπου συνδέεται η μνήμη ROM με το KCPSM3 module και το αρχείο μνήμης που προέκυψε από τον assembler, EX1.VHD.
4. Στο top level αρχείο, embedded_kcpsm3.vhd, αλλάζουμε το όνομα της μνήμης :

```
-- declaration of program ROM
--
  component ex1
    Port (      address : in std_logic_vector(9 downto 0);
            instruction : out std_logic_vector(17 downto 0);
                    clk : in std_logic);
    end component;
--
```

Εικόνα 4.31 : Δήλωση του στοιχείου ex1 στο αρχείο embedded_kcpsm3.vhd

```
program: ex1
  port map(      address => address,
             instruction => instruction,
                     clk => clk);
```

Εικόνα 4.32 : Περιγραφή του στοιχείου ex1 στο αρχείο embedded_kcpsm3.vhd

5. Κάνουμε σύνθεση το project και παρατηρούμε πως έχουμε κάποια warnings (Xst 2591, Zst 2260) και infos (Xst 1961). Αυτά αγνοούνται οπότε για ευκολία μπορούμε να τα φιλτράρουμε ώστε να είναι πιο ευανάγνωστη η αναφορά της σύνθεσης.
6. Στο στάδιο της υλοποίησης πρέπει αρχικά να αναθέσουμε τις θύρες του υψηλότερου επιπέδου σχεδίασης στους φυσικούς ακροδέκτες του FPGA. Δημιουργούμε στο project ένα αρχείο περιορισμών χρήστη (user constraint file) με όνομα embedded_kcpsm3.ucf. Αφού αναθέσουμε τους ακροδέκτες υλοποιούμε τη σχεδίασή μας.

```
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "clk"  LOC = "t9"  ;
NET "in_port<0>"  LOC = "f12"  ;
NET "in_port<1>"  LOC = "g12"  ;
NET "in_port<2>"  LOC = "h14"  ;
NET "in_port<3>"  LOC = "h13"  ;
NET "in_port<4>"  LOC = "j14"  ;
NET "in_port<5>"  LOC = "j13"  ;
NET "in_port<6>"  LOC = "k14"  ;
NET "in_port<7>"  LOC = "k13"  ;
NET "out_port<0>"  LOC = "k12"  ;
NET "out_port<1>"  LOC = "p14"  ;
NET "out_port<2>"  LOC = "l12"  ;
NET "out_port<3>"  LOC = "n14"  ;
NET "out_port<4>"  LOC = "p13"  ;
NET "out_port<5>"  LOC = "n12"  ;
NET "out_port<6>"  LOC = "p12"  ;
NET "out_port<7>"  LOC = "p11"  ;
NET "reset"  LOC = "l14"  ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE
```

Εικόνα 4.33 : Αρχείο περιορισμών χρήστη

7. Τέλος προγραμματίζουμε το FPGA και βλέπουμε πως η σχεδίαση μας λειτουργεί κανονικά.

Στη συνέχεια υλοποιήσαμε ένα ακόμα πρόγραμμα, απλό αλλά που χρησιμοποιεί περισσότερες μονάδες του επεξεργαστή καθώς διαβάζει τις τιμές των switches στην είσοδο του σε έναν καταχωρητή, s0, τις αποθηκεύει στην μνήμη (store memory), στη θέση 02 και μετά τις διαβάζει από αυτήν για να τις δώσει στην έξοδό του μέσω του καταχωρητή s1.

```
         ;Second example
         ;
         ;
         ; Port Definitions
         ;
         CONSTANT switches, 02       ; port for switches  ---- INPUT
         CONSTANT leds, 04           ; port for LED output  -- OUTPUT
         ;
         ;
         ; Main program
start :  LOAD    s0, 00              ; reset counter
         LOAD    s2, 02
loop :   INPUT   s0, switches
         store   s0, (s2)
         fetch   s1, (s2)
         OUTPUT S1, leds
         JUMP    loop
```

Εικόνα 4.34 : Παράδειγμα 2, ex2.psm

Ακολουθούμε την ίδια διαδικασία με προηγουμένως, αλλάζοντας το όνομα της μνήμης στο αρχείο υψηλότερου επιπέδου σχεδίασης ως ex2. Το αρχείο περιορισμών χρήστη παραμένει ίδιο και μετά τον προγραμματισμό του FPGA βλέπουμε πως και σε αυτήν την περίπτωση ο επεξεργαστής λειτουργεί κανονικά.

Στο τρίτο παράδειγμα έχουμε ένα πρόγραμμα όπου τα interrupt είναι ενεργοποιημένα.

```
          ;Third example
          ;
          ;
          ; Port Definitions
          ;
          CONSTANT switches, 02       ; port for switches  ---- INPUT
          CONSTANT leds, 04           ; port for LED output  -- OUTPUT
          ;
          ;
          ; Main program
start  : LOAD   s0, 00               ; initial delay
          LOAD s2, 02
          ENABLE INTERRUPT
loop   : INPUT  s0, switches
          store s0, (s2)
          fetch s1, (s2)
          OUTPUT S1, leds
          JUMP   loop
          ;
          ;
          ADDRESS 2B0
ISR    : XOR s1, FF
loop2  : INPUT s1, switches
          XOR s1, FF
          LOAD s2, s1
          OUTPUT s2, leds
          TEST s2, FE
          JUMP NZ, loop2
          RETURNI ENABLE
          ;
          ;
          ADDRESS 3FF
          JUMP ISR
```

Εικόνα 4.35 : Τρίτο παράδειγμα, ex3.psm

Όπως και στο δεύτερο παράδειγμα, διαβάζει τις τιμές των switches, εδώ τα switches 1-7 και του button 2, στην είσοδο του σε έναν καταχωρητή, s0, τις αποθηκεύει στην μνήμη (store memory), στη θέση 02 και μετά τις διαβάζει από αυτήν για να τις δώσει στην έξοδό του μέσω του καταχωρητή s1. Όταν όμως συμβεί interrupt τότε εκτελείται η εντολή στην διεύθυνση 3FF και στη συνέχεια η ρουτίνα εξυπηρέτησης διακοπής (διεύθυνση 2B0). Εκεί αντιστρέφουμε την τιμή του καταχωρητή s1 (XOR s1, FF) και την εμφανίζουμε στα leds αφού πρώτα την έχουμε φορτώσει στον καταχωρητή s2. Στη συνέχεια συγκρίνουμε το περιεχόμενο του s2 με την τιμή FE (s1 AND FE) και αν το αποτέλεσμα είναι μηδενικό τότε επιστρέφουμε στο κυρίως πρόγραμμα. Η μοναδική περίπτωση στην οποία επιστρέφει στο κυρίως πρόγραμμα είναι όταν όλα τα switches έχουν τιμή 1 και δεν έχουμε interrupt (SW1-SW7 = 1και SW0 = 0).

Ακολουθούμε την ίδια διαδικασία με τα προηγούμενα παραδείγματα, αλλάζοντας το όνομα της μνήμης σε ex3. Στο στάδιο της υλοποίησης αλλάζουμε λίγο το αρχείο περιορισμών χρήστη. Για να μπορέσουμε να δημιουργήσουμε ένα interrupt θα χρησιμοποιήσουμε ένα από τα switches (SW0). Έτσι το switch 0 δεν διαβάζεται στην είσοδο (in_port) αλλά διαβάζεται το button 2 (BTN2) που ορίζουμε στο αρχείο περιορισμών και το οποίο θα δίνει πάντα την τιμή 0.

```
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "clk"  LOC = "t9"  ;
NET "in_port<0>"  LOC = "l13"  ;
NET "in_port<1>"  LOC = "g12"  ;
NET "in_port<2>"  LOC = "h14"  ;
NET "in_port<3>"  LOC = "h13"  ;
NET "in_port<4>"  LOC = "j14"  ;
NET "in_port<5>"  LOC = "j13"  ;
NET "in_port<6>"  LOC = "k14"  ;
NET "in_port<7>"  LOC = "k13"  ;
NET "out_port<0>"  LOC = "k12"  ;
NET "out_port<1>"  LOC = "p14"  ;
NET "out_port<2>"  LOC = "l12"  ;
NET "out_port<3>"  LOC = "n14"  ;
NET "out_port<4>"  LOC = "p13"  ;
NET "out_port<5>"  LOC = "n12"  ;
NET "out_port<6>"  LOC = "p12"  ;
NET "out_port<7>"  LOC = "p11"  ;
NET "interrupt" LOC = "f12" ;
NET "reset"  LOC = "l14"  ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE
```

Εικόνα 4.36 : Αρχείο περιορισμών χρήστη για το τρίτο παράδειγμα

4.4.2 Εισαγωγή σφαλμάτων

Αφού επιβεβαιώσαμε τη σωστή λειτουργία του ενισχυμένου επεξεργαστή, επόμενο βήμα ήταν να εισάγουμε σφάλματα στην εφαρμογή μας. Αυτό πραγματοποιήθηκε με δύο τρόπους.

Ο πρώτος τρόπος ήταν να εισάγουμε τυχαία σφάλματα στο κώδικα (μόνιμα σφάλματα) αλλάζοντας την αρχικοποίηση κάποιων στοιχείων του. Έτσι εισάγαμε στον κώδικα 8 σφάλματα σε τυχαία αλλά καίρια σημεία. Τα σημεία εισαγωγής των σφαλμάτων παρουσιάζονται στον πίνακα 4.2.

Πίνακας 4.2 : Σημεία εισαγωγής μόνιμων σφαλμάτων

Μονάδα	Αντίγραφο	Στοιχείο	Αρχική τιμή αρχικοποίησης	Εσφαλμένη τιμή
Decodes for the control of the program counter & call / return stack	Replica 1	condition_met_lut_1	5A3C	2F15
Carry flag selection	Replica 2	sel_parity_lut_2	F3FF	1F3A
Read, write strobes	Replica 3	read_active_lut_3	0100	530A
t-state & internal_reset	Replica 1	t_state_lut_1	1	F

Μονάδα	Αντίγραφο	Στοιχείο	Αρχική τιμή αρχικοποίησης	Εσφαλμένη τιμή
Store memory	Replica 2	memory_enable_lut_2	8000	1EFA
Parity selection	Replica 3	low_parity_lut_3	6996	1000
Program counter	Replica 1	vector_select_mux_1	E4	1F
Register bank & 2nd operand selection	Replica 2	register_type_lut_2	0145	00EE

Συνθέτουμε και υλοποιούμε τη σχεδίαση (και για τα τρία παραπάνω παραδείγματα) και αφού προγραμματίσουμε το FPGA βλέπουμε πως ο ενισχυμένος επεξεργαστής προστατεύεται επιτυχώς από τα σφάλματα και δεν επηρεάζουν τη λειτουργία του. Καθώς τα σημεία ήταν τυχαία, δοκιμάσαμε σφάλματα και σε διάφορα άλλα σημεία του κώδικα και πάντα η λειτουργία του τριπλασιασμένου επεξεργαστή ήταν ορθή.

Ο δεύτερος τρόπος εισαγωγής σφαλμάτων γίνεται με τη βοήθεια των switches. Για τα δύο πρώτα παραδείγματα προσθέσαμε στον κώδικα 8 πύλες XOR (XORCY) που ελέγχονται ουσιαστικά από τα switches (0-7). Τα σημεία στα οποία τοποθετήθηκαν αυτές οι πύλες φαίνονται αριθμημένα στα σχηματικά της ενότητας 4.1 (όλα βρίσκονται στο πρώτο αντίγραφο της εκάστοτε μονάδας). Τα σημεία εισαγωγής καθώς και η ονομασία των πυλών φαίνονται στον πίνακα :

Πίνακας 4.3 : Σημεία εισαγωγής σφαλμάτων με τη χρήση πυλών XOR

Σφάλμα στο σχηματικό	Μονάδα (εικόνα)	Όνομα	Είσοδος (LI)	Είσοδος (CI)	Έξοδος (O)
1	T_state & internal_reset (4.1)	t_state_xor	not_t_state_1_in	in_port(0)	not_t_state_1
2	Decodes for control of pc & CALL/RETURN stack (4.5)	move_group_xor	move_group_1_in	in_port(1)	move_group_1
3	Parity detection (4.8)	low_parity_xor	low_parity_1_in	in_port(2)	low_parity_1
4	Program Counter (4.10)	vector_selector_xor	pc_vector_1_in(i)	in_port(3)	c_vector_1(i)
5	Register Bank & second operand selection (4.11)	register_enable_xor	register_enable_1_in	in_port(4)	register_enable_1
6	Store Memory (4.12)	memory_enable_xor	memory_enable_1_in	in_port(5)	memory_enable_1
7	ALU multiplexer (4.16)	input_fetch_type_xor	input_fetch_type_1_in	in_port(6)	input_fetch_type_1
8	Read, Write Strobes (4.17)	io_decode_xor	io_initial_decode_1_in	in_port(7)	io_initial_decode_1

Για την εισαγωγή των πυλών XOR χρησιμοποιήθηκε το πρωταρχικό στοιχείο XORCY από τη βιβλιοθήκη [5],[6]. Έτσι, για παράδειγμα, η πύλη που προστίθεται στον κώδικα για το πρώτο σφάλμα είναι η εξής :

```
t_state_xor: XORCY                    --- Fault injection from switch 0
port map (LI => not_t_state_1_in,
          CI => in_port(0),
          O => not_t_state_1);
```

Εικόνα 4.37 : Πύλη XOR που εισάγει σφάλμα στη σχεδίαση

Μετά την υλοποίηση και το προγραμματισμό βλέπουμε πως η λειτουργία του επεξεργαστή δεν επηρεάζεται από τα σφάλματα που εισάγουμε.

Στην περίπτωση του τρίτου παραδείγματος το switch 0 δε θα χρησιμοποιηθεί για εισαγωγή κάποιου σφάλματος αλλά θα χρησιμοποιηθεί για να ελέγξουμε τα interrupts. Έτσι οι πύλες που θα προσθέσουμε στον κώδικα είναι 7. Θα διατηρήσουμε ίδιες τις πύλες για τα σφάλματα 3-8 και θα προσθέσουμε ακόμα μια πύλη :

Πίνακας 4.4 : Σημείο εισαγωγής σφάλματος με τη χρήση πυλών XOR, για το τρίτο παράδειγμα

Σφάλμα στο σχηματικό	Μονάδα (εικόνα)	Όνομα	Είσοδος (LI)	Είσοδος (CI)	Έξοδος (O)
9	Interrupt capture (4.2)	int_capture_xor	clean_int_1_in	in_port(1)	clean_int_1

Η λειτουργία του τριπλασιασμένου επεξεργαστή παραμένει και σε αυτή την περίπτωση ανεπηρέαστη από τα σφάλματα.

Τέλος, δοκιμάσαμε τι θα γίνει στην περίπτωση που έχουμε σφάλμα σε δύο από τα τρία αντίγραφα μιας μονάδας. Σε αυτή την περίπτωση η λειτουργία του επεξεργαστή δεν έμεινε ανεπηρέαστη καθώς σε κάποιο σημείο ο εκλογέας δεν έδωσε τη σωστή τιμή στην έξοδό του μιας και η πλειοψηφούσα τιμή ήταν εσφαλμένη. Ως αποτέλεσμα ο επεξεργαστής μας δεν λειτούργησε σωστά, όπως ήταν αναμενόμενο σύμφωνα με τη βιβλιογραφία.

4.5 Αποτελέσματα

Η TMR εκδοχή του PicoBlaze επαληθεύτηκε μετά από σύνθεση, υλοποίηση και έλεγχο σε μια συσκευή FPGA Spartan-3. Τα αποτελέσματα της σύνθεσης και της χρονικής ανάλυσης που παραθέτονται παρακάτω αναφέρονται στην σχεδίαση χωρίς την εισαγωγή σφαλμάτων μέσω πυλών XOR και λήφθηκαν για μια συσκευή XC3S200, χρησιμοποιώντας την έκδοση 9.2.04i του ISE.

4.5.1 Αξιοποίηση λογικών πόρων

Ο παρακάτω πίνακας συγκρίνει τη χρήση των λογικών πόρων από τις δύο CPU, την αρχική και την ενισχυμένη (τα αποτελέσματα αυτά είναι από τις αναφορές σύνθεσης των δύο σχεδιάσεων). Αυτό που αναμένουμε να δούμε σε αυτά τα αποτελέσματα είναι ο τριπλασιασμός των πόρων που χρησιμοποιεί η αρχική CPU. Στην πραγματικότητα όμως η ενισχυμένη CPU χρησιμοποιεί κάτι παραπάνω από τους τριπλάσιους λογικούς πόρους.

Πίνακας 4.5 : Σύγκριση πόρων συσκευής που χρησιμοποιούν οι δύο σχεδίασεις

	PicoBlaze	**TMR PicoBlaze**	**Αναλογία (PicoBlaze / TMR PicoBlaze)**
Design statistics :			
IOs	30	30	1
Cell usage :			
BELS	**200**	**778**	**3.89**
GND	1	1	1
INV	4	12	3
LUT1	2	6	3
LUT2	5	15	3
LUT3	69	389	5.64
LUT4	33	99	3
MUXCY	39	117	3
MUXF5	9	27	3
VCC	1	1	1
XORCY	37	111	3
Flip-flops - latches	**76**	**228**	**3**
FD	24	72	3
FDE	2	6	3
FDR	30	90	3
FDRE	8	24	3
FDSE	10	30	3
FDS	2	6	3
RAMs	**27**	**79**	**2.93**
RAM16x1D	8	24	3
RAM32x1S	10	30	3
RAM64x1S	8	24	3
RAM16_S18	1	1	1
Clock Buffers	**1**	**1**	**1**
BUFGP	1	1	1
IO Buffers	**29**	**29**	**1**
IBUF	10	10	1
OBUF	19	19	1
Slices	5%	19%	3.8

Αυτή η αύξηση είναι φυσιολογική καθώς στην TMR σχεδίαση έχουν προστεθεί εκ νέου στο αρχικό κύκλωμα οι εκλογείς. Έτσι, παρατηρούμε πως ενώ σχεδόν όλοι οι πόροι που χρησιμοποιεί η σχεδίαση μας είναι τριπλάσιοι (το στοιχείο που υλοποιεί την μνήμη εντολών, RAM16_S18, δεν τριπλασιάζεται καθώς δεν θα εφαρμόσουμε την TMR σε αυτήν), τα LUT3 είναι πολύ περισσότερα. Αυτό οφείλεται στη χρήση των εκλογέων, οι οποίοι υλοποιούνται με LUT3.

4.5.2 Μέγιστη συχνότητα σχεδίασης

Στη συνέχεια, από τις αναφορές χρονικής ανάλυσης των δύο σχεδιάσεων βλέπουμε ποια είναι η μέγιστη συχνότητα λειτουργίας τους. Έτσι για την αρχική σχεδίαση έχουμε ελάχιστη περίοδο 8.826 ns, οπότε έχουμε μέγιστη συχνότητα :

$$f = \frac{1}{T} = \frac{1}{8.826 * 10^{-9}\,\text{sec}} => f = 113.3MHz,$$

ενώ για την TMR σχεδίαση η ελάχιστη περίοδος είναι 12,240 ns, οπότε η μέγιστη συχνότητα είναι :

$$f = \frac{1}{T} = \frac{1}{12.240 * 10^{-9}\,\text{sec}} => f = 81.7MHz$$

Παρατηρούμε πως υπάρχει μια μείωση στη μέγιστη συχνότητα λειτουργίας της TMR σχεδίασης συγκριτικά με αυτή της αρχικής. Αυτή η διαφορά οφείλεται στην επιπλέον καθυστέρηση που επιφέρουν στο κύκλωμα οι εκλογείς.

5 Συμπεράσματα

Σκοπός της παρούσας διατριβής ήταν η παρουσίαση της τεχνικής TMR, τόσο σε θεωρητικό επίπεδο όσο και σε επίπεδο εφαρμογής, σε έναν επεξεργαστή, τον PicoBlaze.

Η διαδικασία εφαρμογής της τεχνικής TMR στον PicoBlaze παρουσιάστηκε εκτενώς. Επεξηγήθηκε αναλυτικά πως γίνεται η αναγνώριση και η ταξινόμηση των επιμέρους μονάδων που αποτελούν τον επεξεργαστή και πως πρέπει να εφαρμοστεί η τεχνική TMR σε αυτές, ανάλογα με τη λογική που υλοποιούν.

Στη συνέχεια, μέσα από παραδείγματα, δείξαμε τον ακριβή τρόπο υλοποίησης των TMR μονάδων του PicoBlaze. Μετά από μια παρουσίαση του assembler KCPSM3, που χρησιμοποιήθηκε για τη μεταγλώττιση των προγραμμάτων που καλέστηκε να εκτελέσει ο TMR PicoBlaze, έγινε έλεγχος της λειτουργίας του.

Στην πορεία έπρεπε να αποδείξουμε την ορθή λειτουργία του κατά την ύπαρξη σφαλμάτων. Για το σκοπό αυτό εισάγαμε κάποια σφάλματα στον κώδικα (μόνιμα σφάλματα) αλλάζοντας την αρχικοποίηση κάποιων στοιχείων σε διάφορες μονάδες του επεξεργαστή. Αφού αποδείξαμε πως η TMR έκδοσή μας μπορεί να αντιμετωπίσει τα σφάλματα αυτά δοκιμάσαμε έναν πιο ρεαλιστικό τρόπο εισαγωγής σφαλμάτων. Εισάγαμε στον κώδικα, σε καίρια σημεία, πύλες XOR οι οποίες ελέγχονται από τα switches που βρίσκονται στο Spartan-3. Έτσι μπορέσαμε να εισάγουμε σφάλματα σε πραγματικό χρόνο αλλάζοντας την κατάσταση των switches.

Σε αυτό το σημείο είδαμε πως η τεχνική TMR μπορεί να προστατέψει επιτυχώς τη σχεδίασή μας από τα SEU, με κόστος τους επιπλέον πόρους συσκευής που χρησιμοποιεί. Παρατηρήσαμε πως υπάρχει μεγάλη αύξηση σε επιφάνεια, περισσότερη από την αναμενόμενη τριπλάσια της αρχικής (περίπου 4 φορές). Η αρχική σχεδίαση καταλαμβάνει 5% των slices του FPGA ενώ η ενισχυμένη μέσω TMR το 19%. Ταυτόχρονα παρουσιάζεται μείωση στη μέγιστη συχνότητα λειτουργίας της σχεδίασης. Και οι δύο αυτές επιπτώσεις οφείλονται στους voters, οι οποίοι είναι επιπρόσθετοι στην αρχική σχεδίαση.

Τέλος, μπορούμε να προτείνουμε έναν διαφορετικό τρόπο εισαγωγής σφαλμάτων, συμπεριλαμβάνοντας τη μονάδα SEU Controller [10] στη σχεδίασή μας ώστε να υλοποιήσουμε μια δομή ανίχνευσης και διόρθωσης σφαλμάτων. Ο τρόπος αυτός όμως απαιτεί η σχεδίασή μας να γίνει για ένα FPGA Virtex-5, καθώς αυτά έχουν ενσωματωμένα Readback CRC κυκλώματα, που αυτοματοποιούν τον εντοπισμό των SEU και ενεργοποιούν το σύστημα ώστε να πράξει αναλόγως για τη διόρθωσή τους. Η μονάδα αυτή εξομοιώνει SEU σε ένα Virtex-5 εισάγοντας σφάλματα με έναν ελεγχόμενο και προβλέψιμο τρόπο στη μνήμη αρχικοποίησης. Τα σφάλματα σε αυτή την περίπτωση είναι προσωρινά και επηρεάζουν εξίσου λογική και συνδέσεις, ενώ έχουμε στατιστικές μετρήσεις που ανταποκρίνονται στην πραγματικότητα. Επίσης μας δίνει τη δυνατότητα να αξιολογήσουμε και να τεστάρουμε το Readback CRC κύκλωμα και τις δυνατότητες διόρθωσης σφαλμάτων της μονάδας, κάτι που είναι αδύνατο με πραγματικά SEU.

Βιβλιογραφία

[1] Lima Kastensmidt F., Carro L., Reis R. "Fault-Tolerance Techniques for SRAM-based FPGAs". Springer, 2006

[2] Xilinx, Inc. "Triple Module Redundancy Design Techniques for Virtex FPGAs". Xilinx Application Note 197, v1.0.1. Aut.: Carl Carmichael. July 6, 2006.

[3] Xilinx, Inc. "PicoBlaze 8-bit Embedded Microcontroller User Guide for Spartan-3, Virtex-II, and Virtex-II Pro FPGAs". Xilinx UG129, v1.1.2. Aut.: Ken Chapman. June 24, 2008.

[4] Xilinx, Inc. "PicoBlaze KCPSM3 8-bit micro Controller for Spartan-3, Virtex-II and Virtex-II PRO". Xilinx Ltd Rev.7. Aut.: Ken Chapman. October 2003.

[5] Xilinx, Inc. "Spartan-3 Libraries Guide for HDL Designs". Xilinx, ISE 10.1. 2008.

[6] Xilinx, Inc. "Spartan-3 Libraries Guide for Schematic Designs". Xilinx UG608, v11.3. September 16, 2009.

[7] Xilinx, Inc. "Spartan-3 FPGA Family Data Sheet". Xilinx DS099. December 4, 2009.

[8] Xilinx, Inc. "Spartan-3 FPGA Starter Kit Board User Guide". Xilinx UG130, v1.2. June 20, 2008.

[9] Xilinx, Inc. "Spartan-3 Generation FPGA User Guide. Extended Startan-3A, Spartan-3E, and Spartan-3 FPGA Families". UG331 (v1.6) December 3, 2009.

[10] Xilinx, Inc. "SEU Strategies for Virtex-5 Devices". Xilinx Application Note 864, v1.0.1. Aut.: Ken Chapman, Les Jones. March 5, 2009.

Παράρτημα Α

Στο παράρτημα αυτό παρουσιάζεται ο κατάλογος περιεχομένων του επισυναπτόμενου cd.

- Triple_Modular_Redundancy.doc
- Triple_Modular_Redundancy.pdf
- Triple_Modular_Redundancy.ppt
- Triple_Modular_Redundancy_presentation.pdf
- TMR_PicoBlaze, το project με την TMR έκδοση του PicoBlaze
- TMR_PicoBlaze_asm, ο φάκελος με τον assembler, τα αρχεία εισόδου και εξόδου του για το project TMR_PicoBlaze.
- TMR_PicoBlaze_xor, το project με την TMR έκδοση του PicoBlaze όπου έχουμε εισαγωγή σφαλμάτων μέσω των πυλών XOR.
- TMR_PicoBlaze_xor, ο φάκελος με τον assembler, τα αρχεία εισόδου και εξόδου του για το project TMR_PicoBlaze_xor.
- ex1.psm, ex2.psm και ex3.psm, τα αρχεία με τα προγράμματα που καλείται να εκτελέσει ο PicoBlaze