ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών & Ψηφιακών Συστημάτων

ΣΧΕΔΙΑΣΗ ΣΥΣΤΗΜΑΤΟΣ ΨΗΦΙΔΑΣ ΜΕ ΔΥΝΑΤΟΤΗΤΕΣ SIMD ΔΥΝΑΜΙΚΗΣ ΕΠΑΝΑΔΙΑΤΑΞΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Σ. Αγιαννιώτης

Επιβλέπων : Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2007

(κενή σελίδα)

Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων Και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Εργαστηριο Μικροϋπολογιστων & Ψηφιακων Συστηματων

ΣΧΕΔΙΑΣΗ ΣΥΣΤΗΜΑΤΟΣ ΨΗΦΙΔΑΣ ΜΕ ΔΥΝΑΤΟΤΗΤΕΣ SIMD ΔΥΝΑΜΙΚΗΣ ΕΠΑΝΑΔΙΑΤΑΞΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Σ. Αγιαννιώτης

Επιβλέπων : Κιαμάλ Πεκμεστζή
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την --------2007.

.....	.....	.....
Κ. Πεκμεστζή	Γ. Οικονομάκος	Ν. Κοζύρης
Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2007

.....

Νικόλαος Σ. Αγιαννιώτης

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

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας διπλωματικής εργασίας είναι η παρουσίαση ενός δυναμικά επαναδιατάξιμου συστήματος επεξεργασίας. Το σύστημα αυτό σχεδιάστηκε, στα πλαίσια της εργασίας, για τη μελέτη της αποδοτικότητας και της αποτελεσματικότητας του συνδυασμού επαναδιατάξιμου υλικού με γενικού σκοπού επεξεργαστές για την υλοποίηση εφαρμογών επιπέδου λέξης και μεγάλης έντασης υπολογισμών. Το συγκεκριμένο σύστημα είναι ένα ολοκληρωμένο και επαναδιατάξιμο σύστημα σε ψηφίδα (*System-On-Chip*, SoC), με βαθμό επαναδιατάξης επιπέδου λέξης και πλάτος χειριστή δεδομένων (data-path) ίσο με 16 bits. Στοχεύει στην απεικόνιση εφαρμογών με υψηλό ρυθμό εξόδου και παραλληλία δεδομένων. Αποτελεί επέκταση ενός SoC (με την ονομασία MicroSoc([1],[2])) που έχει ήδη σχεδιαστεί στο "Εργαστήριο Μικρουπολογιστών και Ψηφιακών Συστημάτων" του Ε.Μ.Π, με την προσθήκη σε αυτό μιας περιφερειακής μονάδας επαναδιατάξιμης λογικής. Για το λόγο αυτό, στο εξής θα αναφερόμαστε στο επαναδιατάξιμο σύστημα που περιγράφουμε με το όνομα rMicroSoc (reconfigurable MicroSoc). Το rMicroSoc αποτελείται από έναν πυρήνα επεξεργασίας αρχιτεκτονικής ARM[3], ένα σύστημα μνημών με μνήμες RAM, ROM και ένα αρχείο καταχωρητών, κυκλώματα για την υλοποίηση του διαδρόμου επικοινωνίας αρχιτεκτονικής AMBA ASB[4] και ένα περιφερειακό σύστημα επαναδιατάξιμης λογικής. Το περιφερειακό αυτό απαρτίζεται -στη βάση του- από 16 επαναδιατάξιμες μονάδες επεξεργασίας, οργανωμένες σε έναν 4x4 πίνακα ακολουθώντας το μοντέλο επεξεργασίας SIMD([5],[6]). Άλλα συστατικά στοιχεία του περιφερειακού είναι δύο μνήμες -μία μνήμη δεδομένων επεξεργασίας και μία μνήμη δεδομένων διαμόρφωσης-, κυκλώματα που υλοποιούν τη διεπαφή σύνδεσης του περιφερειακού σαν "σκλάβου" (slave) στο σύστημα διαδρόμων AMBA ASB του rMicroSoc και διάφορες άλλες βοηθητικές μονάδες. Η αρχιτεκτονική και ο τρόπος λειτουργίας του περιφερειακού αυτού βασίζονται στο σύστημα MorphoSys[7], το οποίο είναι επίσης ένα επαναδιατάξιμο SoC, με βαθμό επαναδιάταξης επιπέδου λέξης. Το MorphoSys αναπτύχθηκε στην αρχική του μορφή από μια ομάδα επιστημόνων του τμήματος Ηλεκτρολόγων μηχανικών και μηχανικών Η/Υ του πανεπιστημίου της Καλιφόρνια και του πανεπιστημίου του Ρίο Ντε Τζανέιρο.

Στο πρώτο κομμάτι της εργασίας παρουσιάζονται κάποια θεωρητικά στοιχεία, τα οποία συμβάλλουν στην καλύτερη κατανόηση της σχεδίασης και της αρχιτεκτονικής του rMicroSoc. Συγκεκριμένα, αρχικά δίνονται πληροφορίες γενικά για τα επαναδιατάξιμα συστήματα επεξεργασίας και ακολούθως για το μοντέλο επεξεργασίας SIMD. Στη συνέχεια αναλύεται η αρχιτεκτονική και η λειτουργία του επαναδιατάξιμου συστήματος MorphoSys -πάνω στο οποίο βασίστηκε και η αρχιτεκτονική του rMicroSoc-, καθώς και του MicroSoc. Μάλιστα, στα πλαίσια της περιγραφής του τελευταίου συστήματος, δίνονται πληροφορίες για την αρχιτεκτονική ARM επεξεργαστών και η αρχιτεκτονική του συστήματος διαδρόμων επικοινωνίας AMBA. Έπειτα, ακολουθεί η αναλυτική παρουσίαση όλων των χαρακτηριστικών, της αρχιτεκτονικής και της λειτουργίας του συστήματος rMicroSoc. Στο τέλευταίο κομμάτι του κειμένου παρουσιάζονται οι τρεις εφαρμογές που απεικονίστηκαν πειραματικά στο επαναδιατάξιμο σύστημα (φίλτρο FIR 4-tap, φίλτρο FIR 16-tap και 1-D μετασχηματισμός DCT σε μπλοκ εικόνας 8x8), καθώς και τα αποτελέσματα συγκριτικών μετρήσεων μεταξύ των απεικονίσεων αυτών και αντίστοιχων απεικονίσεων υλοποιημένων σε λογισμικό.

<u>Λέξεις Κλειδιά</u>
επαναδιάταξη, επαναδιατάξιμα συστήματα, ολοκληρωμένα κυκλώματα ειδικών κυκλωμάτων, βαθμός επαναδιάταξης επιπέδου λέξης, SIMD, MorphoSys, MicroSoc, AMBA, ARM, πίνακας επαναδιατάξιμων κελιών, λέξη διαμόρφωσης, φίλτρο πεπερασμένης απόκρισης, διακριτός συνημιτονικός μετασχηματισμός

ABSTRACT

In this diploma thesis, a dynamic reconfigurable processing system is presented, which is called rMicroSoc. It was designed to study the efficiency and effectiveness of combining reconfigurable hardware with general-purpose processors for implementing word level and computation-intensive applications. This system is reconfigurable System-On-Chip (SoC), with coarse-grained granularity and a data-path of 16 bits. It`s target applications are those with data parallel computations and high throughput requirements. This system is the extension of another SoC (called MicroSoc), which is already designed at Verilog level at the "Laboratory of Microprocessors and Digital Systems" of the National Technical University of Athens, by the addition of a slave reconfigurable module. rMicroSoc is composed of an ARM architecture processing core, a memory system with RAM, ROM and a register file, hardware that implements the communication bus system AMBA ASB and slave reconfigurable module. This module is mainly composed of 16 reconfigurable processing units (the reconfigurable cells-RCs), which form a 4x4 array according to the SIMD computing model. Other components of the module are two memories -a context memory and a data memory-, hardware for the implementation of the interface that connects the module as slave to the AMBA ASB of the rMicroSoc and other complementary components. The architecture and the operation of this module are based on MorphoSys, which is another coarse-grained reconfigurable SoC. MorphoSys was developed by some engineers of the electrical and computer engineering department at the university of California, Irvine and the Federal University of Rio de Janeiro, Brazil.

At the first section of the diploma thesis is presented some theory, which helps the better understanding of the architecture and designng of rMicroSoc. Specifically, some information about the reconfigurable processing systems and the SIMD processing model are provided. Afterwards, it is described the architecture and the operation of MorphoSys and MicroSoc and some information about the ARM and the AMBA architectures are given. Then follows the detailed presentation of the features, the architecture and the operation of rMicroSoc. At the end, the three applications, which were experimentaly mapped at the reconfigurable system (one 4-tap FIR filter, one 16-tap FIR filter and a 1-D DCT on a 8x8 image block), are detailed described, along with the results of the comparison between these mappings and respective mappings implemented with software.

Keywords
reconfigurability, reconfigurable systems, ASICs, coarse-grained granularity, SIMD, MorphoSys, MicroSoc, AMBA, ARM, reconfigurable-cell array, context word, FIR filter, DCT

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον Επιβλέποντα της παρούσας διπλωματικής εργασίας κ.κ. Πεκμεστζή Κιαμάλ, τακτικό Καθηγητή της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Π., για την εμπιστοσύνη που μου έδειξε με την ανάθεση του θέματος αυτού και για την ευκαιρία να ασχοληθώ με ένα ιδιαίτερα ενδιαφέρον τεχνολογικό και θεματικό αντικείμενο. Οι συμβουλές του βοήθησαν στην εξέλιξη και ολοκλήρωση της εργασίας.

Επίσης θα ήθελα να εκφράσω τις θερμές ευχαριστίες μου στον υποψήφιο Δρ. Ξύδη Σωτήριο για την άψογη συνεργασία μεταξύ μας, αλλά και για την πολύ σημαντική βοήθεια και υποστήριξη που μου παρείχε κατά τη διάρκεια της εργασίας. Χωρίς τη συμβολή του το αποτέλεσμα δε θα ήταν το ίδιο.

Ένα ακόμη ευχαριστώ απευθύνεται στους υποψήφιους Δρ. Σιδέρη Ισίδωρο, Μπεκιάρη Δημήτρη και Πιλίτσο Δημήτρη του “Εργαστηρίου Μικροϋπολογιστών και Ψηφιακών Συστημάτων” για το εποικοδομητικό κλίμα συνεργασίας και κατανόησης που διαμορφώθηκε.

Στο τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την απέραντη αγάπη που μου δείχνουν, την κατανόηση που έχουν απέναντί μου και την υποστήριξη που μου παρέχουν ως σήμερα, καθώς και εκείνους από το κοινωνικό μου περιβάλλον που -φανερά ή όχι- με βοήθησαν δίνοντάς μου κουράγιο όποτε χρειάστηκε.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΚΕΦΑΛΑΙΟ 1
Επαναδιατάξιμα Συστήματα Υπολογισμού

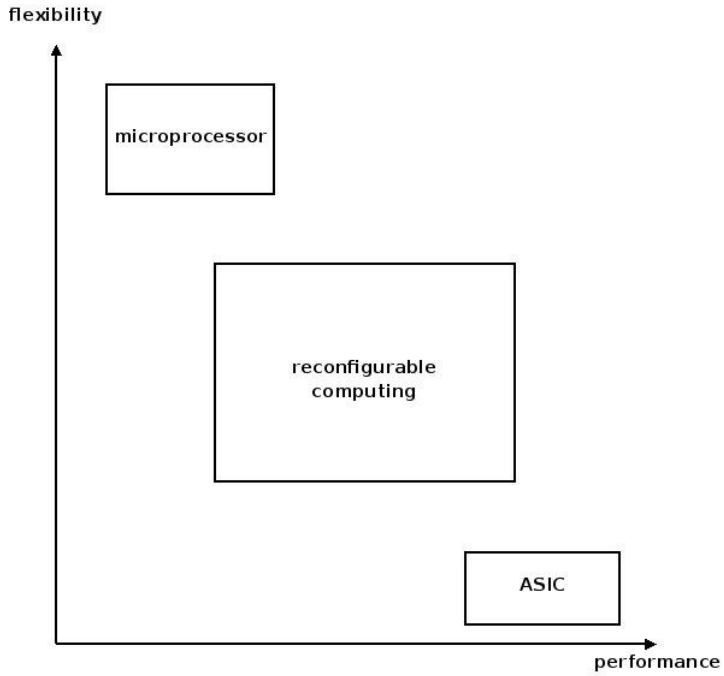
1.1. Γενικά

Εξαιτίας της μεγάλης και ραγδαίας ανάπτυξης της τεχνολογίας VLSI τα τελευταία χρόνια, όπου έχουν σημειωθεί τεράστια άλματα τόσο από πλευράς μείωσης της επιφάνειας των κυκλωμάτων όσο και από πλευράς τελειοποίησης των τεχνικών σχεδίασης, έχει γίνει εφικτή η επέκταση της επιστημονικής έρευνας και μελέτης σε τομείς, στους οποίους κατά το παρελθόν το ενδιαφέρον και οι δυνατότητες ήταν περιορισμένα. Ένας από αυτούς είναι και τα επαναδιατάξιμα συστήματα.

Η έρευνα στον τομέα των επαναδιατάξιμων συστημάτων έχει φτάσει σε πολύ σημαντικά επίπεδα, με τάσεις να αυξάνονται ολοένα και περισσότερο. Ο λόγος για αυτή τη στροφή είναι η μεγάλη προοπτική των επαναδιατάξιμων συστημάτων στο να επιταχύνουν σε σημαντικό βαθμό μια μεγάλη ποικιλία εφαρμογών. Το χαρακτηριστικό-κλειδί αυτών των συστημάτων είναι η ικανότητά τους να εκτελούν πολύπλοκους αριθμητκούς υπολογισμούς κατευθείαν πάνω στο υλικό (hardware), κάτι που αυξάνει τις επιδόσεις τους σε ταχύτητα υπολογισμού. Ταυτόχρονα εμφανίζουν χαρακτηριστικά που συναντώνται στο λογισμικό (software), όπως η ευελιξία μέσω της δυνατότητας προγραμματισμού. Παρακάτω, γίνεται μια πιο αναλυτική αναφορά στα χαρακτηριστικά των επαναδιατάξιμων συστημάτων.

Τα επαναδιατάξιμα συστήματα αποτελούν συστήματα υπολογισμού, των οποίων ο πυρήνας επεξεργασίας μπορεί να προσαρμόσει τις εσωτερικές διασυνδέσεις μεταξύ των στοιχείων του και τη λειτουργία του, ανάλογα με την εφαρμογή που υλοποιείται. Το χαρακτηριστικό αυτό επιτρέπει τη βέλτιστη προσαρμογή του συστήματος στις ιδιαίτερες υπολογιστικές απαιτήσεις του εκάστοτε αλγορίθμου.

Τα επαναδιατάξιμα συστήματα παρουσιάζουν μια προσέγγιση σχεδίασης υπολογιστικών συστημάτων, η οποία βρίσκεται ενδιάμεσα των δύο ακραίων προσεγγίσεων σχεδίασης, αυτής με ολοκληρωμένα κυκλώματα ειδικών εφαρμογών (*Application-Specific Integrated Circuit*, ASIC[8]) και αυτής με γενικού σκοπού επεξεργαστές (Σχήμα 1.1). Γενικά, ένα επαναδιατάξιμο σύστημα παρουσιάζει πιο ευρείς χρήσεις σε σχέση με ένα ASIC και έχει πολύ καλύτερες επιδόσεις σε σχέση με ένα σύστημα βασισμένο αποκλειστικά σε γενικού σκοπού επεξεργαστή. Παρακάτω, γίνεται μια σύντομη αναφορά στις δύο αυτές επικρατέστερες τεχνικές σχεδίασης, ώστε να φανερωθούν τα πλεονεκτήματα των επαναδιατάξιμων συστημάτων.



Σχήμα 1.1: Γεφυρώνοντας το χάσμα

1.2. Τεχνικές Σχεδίασης Συστημάτων

Κάθε αλγόριθμος που πρέπει να υλοποιηθεί σε ένα ψηφιακό σύστημα μπορεί να παρασταθεί ως ένας συνδυασμός αφηρημένων γράφων ροής δεδομένων και ελέγχου, με κάθε κόμβο του γράφου να παριστάνει κάποια πρωτογενή λειτουργία, όπως πρόσθεση μεταξύ ακεραίων ή σύγκριση. Η κυριότερη λειτουργία ενός υπολογιστή είναι να αξιολογήσει και να υλοποιήσει τέτοιους γράφους ώστε να επιτύχει κάποιο στόχο. Ασφαλώς, οι σύγχρονοι υπολογιστές δεν εργάζονται σε τόσο αφηρημένο επίπεδο, απευθείας πάνω στους γράφους αυτούς. Τα προγράμματα κωδικοποιούνται ως ένα σύνολο εντολών μηχανής που μπορούν να εκτελεστούν η μία μετά την άλλη σε μια συγκεκριμένη σειρά. Σημασία έχει ότι -ανεξάρτητα από το πως ένα πρόγραμμα κωδικοποιείται σε φυσικό επίπεδο- αυτοί οι γράφοι παριστάνουν τους πραγματικούς υπολογισμούς που εκτελούνται. Στη συνέχεια δίνονται λεπτομέρειες για τις δύο βασικές τεχνικές σχεδίασης συστημάτων.

1.2.1. Χρήση ASIC

Μια μέθοδος για την εκτέλεση αλγορίθμων είναι η αποκλειστική χρήση υλικού, η οποία υλοποιείται συνήθως με τη χρήση ενός ASIC. Τα κυκλώματα αυτά είναι ολοκληρωμένα κυκλώματα (*Integrated Circuit*, IC) που έχουν προσαρμοστεί για μία πολύ συγκεκριμένη χρήση, δηλαδή για την υλοποίηση ενός συγκεκριμένου αλγορίθμου. Σε ένα ASIC υπάρχουν, δηλαδή, λειτουργικές μονάδες που είναι

αφιερωμένες σε μεμονωμένες λειτουργίες του προγράμματος και διασυνδέονται μεταξύ τους με τέτοιον τρόπο ώστε να υλοποιηθεί ένας συγκεκριμένος αλγόριθμος. Όπως είναι λογικό, αυτό το χαρακτηριστικό καθιστά τα ASIC πολύ γρήγορα και αποδοτικά στην εκτέλεση του συγκεκριμένου αλγορίθμου για τον οποίο έχουν σχεδιαστεί. Άλλο πλεονεκτήματα τέτοιου είδους κυκλωμάτων σε σύγκριση με συστήματα βασισμένα σε επεξεργαστή είναι η ελαχιστοποίηση του χρόνου αδράνειας των λειτουργικών μονάδων. Αυτό οφείλεται στο ότι η λειτουργία κάθε λειτουργικής μονάδας είναι γνωστή και σχεδιάζεται εκ των προτέρων. Εξάλλου σε έναν επεξεργαστή, ορισμένα είδη λειτουργικών μονάδων μπορεί να μη χρησιμοποιηθούν ποτέ από μια συγκεκριμένη εφαρμογή.

Υπάρχουν ασφαλώς και οι αρνητικές πλευρές της χρήσης των ASIC. Το γεγονός ότι τα κυκλώματα αυτά κατασκευάζονται για την αποδοτική υλοποίηση ενός συγκεκριμένου αλγορίθμου και μόνο σημαίνει ότι δεν μπορεί να χρησιμοποιηθεί (είτε καθόλου είτε μη αποδοτικά) για άλλους αλγορίθμους. Δεν μπορούν να θεωρηθούν δηλαδή τα ASIC ως γενικού σκοπού αρχιτεκτονικές. Επιπλέον, το κύκλωμα που σχεδιάζεται δεν είναι δυνατόν να διαφοροποιηθεί μετά την κατασκευή του. Αυτό υποχρεώνει κάποιον να το σχεδιάσει και να το κατασκευάσει εξ` ολοκλήρου από την αρχή, σε περίπτωση που απαιτείται κάποια αλλαγή στο κύκλωμα. Η διαδικασία αυτή είναι χρονοβόρα και δαπανηρή, αν αναλογιστεί κανείς την περίπτωση που χρειάζεται να αντικατασταθεί ένα συγκεκριμένο ASIC σε ένα μεγάλο αριθμό συστημάτων ήδη σε λειτουργία.

1.2.2. Χρήση μικροεπεξεργαστών γενικού σκοπού

Η δεύτερη μέθοδος για την υλοποίηση αλγορίθμων είναι η χρήση μονάδων με μικροεπεξεργαστές ή αυτόνομους μικροελεγκτές, οι οποίοι μπορούν να προγραμματιστούν μέσω λογισμικού. Οι επεξεργαστές περιλαμβάνουν μία ή περισσότερες λειτουργικές μονάδες, κάθε μία από τις οποίες είναι ικανή να υλοποιήσει μια συγκεκριμένη ομάδα λειτουργιών. Ένας απλός επεξεργαστής συνήθως έχει μία μοναδική γενικού σκοπού λειτουργική μονάδα, που είναι γνωστή ως *αριθμητική-λογική μονάδα* (Arithmetic-Logical Unit, ALU), η οποία μπορεί να εκτελέσει μία μόνο λειτουργία τη φορά. Πιο εξελιγμένοι επεξεργαστές, αντιθέτως, (superscalar, VLIW) προσπαθούν να χρησιμοποιήσουν ταυτόχρονα πολλαπλές λειτουργικές μονάδες διαφορετικών ειδών για να εκτελούν προγράμματα ταχύτερα. Σε κάθε κύκλο ρολογιού γίνεται προσπάθεια να εκτελεστούν όσο το δυνατόν περισσότερες πρωτογενείς λειτουργίες στις λειτουργικές μονάδες που είναι διαθέσιμες. Το ιδανικό είναι να χρησιμοποιούνται όλες οι λειτουργικές μονάδες σε κάθε κύκλο ρολογιού, όμως εξαιτίας της εμφάνισης εξαρτήσεων δεδομένων και ελέγχου στους πρακτικούς αλγόριθμους κάτι τέτοιο δεν είναι εφικτό. Στην ουσία οι λειτουργικές μονάδες είναι το μόνο που χρειάζεται ένας υπολογιστής για να υλοποιήσει τις λειτουργίες που περιλαμβάνονται σε ένα γράφο ροής δεδομένων, ωστόσο στην πράξη οι υπολογιστές υποστηρίζουν τη φυσική κίνηση δεδομένων μεταξύ των λειτουργικών μονάδων και μεταξύ λειτουργικών μονάδων και της μνήμης. Για

το λόγο αυτό πρέπει να υπάρχει ένα *αρχείο καταχωρητών* (register file) -η μικρότερη και ταχύτερη διαθέσιμη μνήμη- το οποίο να μπορεί να αποθηκεύει δεδομένα εξόδου των λειτουργικών μονάδων. Η ευελιξία λοιπόν των λειτουργικών μονάδων, του αρχείου καταχωρητών και των διασυνδέσεων είναι που κάνει τους επεξεργαστές να μπορούν να προγραμματιστούν και τους δίνει τη δυνατότητα να εκτελέσουν έναν οποιοδήποτε τυχαίο αλγόριθμο με ικανοποιητική απόδοση. Όπως γίνεται κατανοητό, αυτή η λύση είναι πιο ευέλικτη αφού μια αναγκαία τροποποίηση στην λειτουργικότητα του όλου κυκλώματος, απαιτεί μονάχα την τροποποίηση των εντολών που εκτελεί ο μικροεπεξεργαστής για την υλοποίηση του αλγορίθμου. Η διαδικασία αυτή αλλαγής της λειτουργικότητας του συστήματος δεν απαιτεί την αλλαγή του υλικού. Από την άλλη, η αρνητική πλευρά της ευελιξίας που μπορεί να προσφέρει αυτή η μέθοδος υλοποίησης είναι η δραματική πτώση της απόδοσης του συστήματος. Αυτό μπορεί να εκφράζεται ως πτώση του ρυθμού παραγωγής αποτελεσμάτων και το μέγεθος αυτό είναι σαφώς μικρότερο σε σχέση με το αντίστοιχο σε ένα κύκλωμα ASIC. Αυτή η πτώση στην απόδοση οφείλεται στο ότι ο επεξεργαστής πρέπει αρχικά να προσπελάσει τη μνήμη και να αναγνώσει την επόμενη προς εκτέλεση εντολή, στη συνέχεια να την αποκωδικοποιήσει για να βρει σε ποια λειτουργία αντιστοιχεί και μόνο τότε να την εκτελέσει. Τα παραπάνω αντιστοιχούν σε ένα σημαντικό μέγεθος εργασίας που πρέπει να επιτελεσθεί για την εκτέλεση μίας και μόνο εντολής. Ένα άλλο μειονέκτημα της μεθόδου αυτής είναι ότι το σύνολο εντολών που υποστηρίζει ένας επεξεργαστής -άρα και οι βασικές λειτουργίες που μπορεί να επιτελέσει- ορίζεται κατά το σχεδιασμό του και δεν μπορεί στη συνέχεια να τροποποιηθεί. Οποιαδήποτε άλλη πιο σύνθετη λειτουργία είναι αναγκαίο να υλοποιηθεί, πρέπει να βασιστεί πάνω σε αυτές τις ήδη υπάρχουσες εντολές. Κατά μία έννοια -λοιπόν- οι συμβατικοί επεξεργαστές περιορίζονται από το μικρό εύρος των εντολών που υποστηρίζονται αλλά και από διάφορους περιορισμούς κατά την εκτέλεση.

1.2.3. Σύγκριση των τεχνικών

Έχοντας μια πιο σφαιρική άποψη για τις δύο επικρατούσες τεχνικές σχεδίασης λογικών συστημάτων, μπορούν να παρουσιαστούν ορισμένα συγκριτικά συμπεράσματα. Η μία πλευρά, αυτή των ASIC, περιλαμβάνει την ανάπτυξη ενός αλγορίθμου αποκλειστικά στο υλικό. Λόγω του ότι ο αλγόριθμος εκτελείται απευθείας στο κατώτερο στρώμα του συστήματος, η απόδοση (από πλευράς ταχύτητας και εκμετάλλευσης του υλικού για το συγκεκριμένο αλγόριθμο) της μεθοδολογίας αυτής είναι πολύ μεγάλη. Αλλά όμως για την υλοποίηση ενός διαφορετικού αλγορίθμου απαιτείται εκ νέου ο σχεδιασμός και η κατασκευή ενός διαφορετικού συστήματος. Η άλλη πλευρά, αυτή των μικροεπεξεργαστών, έχει να επιδείξει μεγάλη ευελιξία, αφού οποιαδήποτε τροποποίηση είναι απαραίτητη μπορεί να γίνει αλλάζοντας απλώς το λογισμικό που εκτελεί ο επεξεργαστής, χωρίς καμία αλλαγή στο υλικό. Φυσικά η αποδοτικότητα είναι πιο μειωμένη σε σχέση με την πρώτη μεθοδολογία.

Στο σημείο αυτό διακρίνεται ένα κενό μεταξύ των δύο αυτών τεχνικών σχεδίασης, αφού η μία παρουσιάζει υψηλή αποδοτικότητα με χαμηλή ευελιξία ενώ η άλλη υψηλή ευελιξία με χαμηλή αποδοτικότητα. Αυτό ακριβώς το κενό ήρθε να καλύψει η τεχνολογία των επαναδιατάξιμων σύστημάτων, στοχεύοντας στην επίτευξη μεγαλύτερης αποδοτικότητας σε σχέση με το λογισμικό, διατηρώντας παράλληλα ένα μεγαλύτερο επίπεδο ευελιξίας σε σχέση με το υλικό. Αυτή είναι η λογική στην οποία κινήθηκαν από την αρχή τα επαναδιατάξιμα κυκλώματα. Αποτελούν λοιπόν ένα υβριδικό μοντέλο σχεδίασης των δύο προαναφερθέντων τεχνικών.

1.3. Η Εμφάνιση των Επαναδιατάξιμων Συστημάτων

Η έννοια της δυναμικής επαναδιάταξης στη σχεδίαση συστημάτων σαν ιδέα υπάρχει αρκετό καιρό. Παρόλα αυτά, ο όρος επαναδιατάξιμη λογική, όπως χρησιμοποιείται στην παρούσα εργασία και γενικά στη σχετική βιβλιογραφία, αναφέρεται σε συστήματα που ενσωματώνουν κάποια μορφή προγραμματισμού του υλικού -διαμορφώνοντας το πως το υλικό χρησιμοποιείται μέσω ενός αριθμού σημείων ελέγχου. Αυτά τα σημεία ελέγχου θα πρέπει να μπορούν να αλλάζουν περιοδικά, ώστε να εκτελεστούν διαφορετικές εφαρμογές πάνω στο ίδιο υλικό.

Για να επιτευχθεί ο στόχος της κάλυψης του κενού που υπάρχει μεταξύ των δύο προυπάρχοντων τεχνικών σχεδίασης, οι επαναδιατάξιμες συσκευές (ένα χαρακτηριστικό τέτοιο παράδειγμα είναι τα FPGAs[8]) παρουσιάζουν μια κοινή εσωτερική δομή. Το βασικό χαρακτηριστικό της είναι ότι περιλαμβάνει συνήθως έναν πίνακα από όμοιες υπολογιστικές μονάδες (λογικά μπλοκ), των οποίων η λειτουργικότητα καθορίζεται από έναν αριθμό προγραμματιζόμενων bits (*configuration bits*). Με τον όρο προγραμματιζόμενα, υποννοείται ότι ο προγραμματισμός γίνεται με παρέμβαση του λογισμικού. Αυτές οι λογικές μονάδες συνδέονται μεταξύ τους -εσωτερικά στον πίνακα- χρησιμοποιώντας διάφορα κυκλώματα διασύνδεσης, τα οποία είναι επίσης προγραμματιζόμενα μέσω λογισμικού. Όπως γίνεται φανερό, αν ένα επαναδιατάξιμο σύστημα ακολουθεί τη βασική αυτή εσωτερική δομή τότε είναι δυνατόν να απεικονιστεί σε αυτό κάθε είδους ψηφιακό κύκλωμα εκτελώντας δύο ενέργειες. Αρκεί να υλοποιηθούν οι λογικές συναρτήσεις του -προς απεικόνιση- κυκλώματος παραμετροποιώντας κατάλληλα τις λογικές μονάδες που απαρτίζουν τον πίνακα και να ρυθμιστούν οι εσωτερικές διασυνδέσεις όπως ακριβώς απαιτείται, ώστε η γενικότερη συνεργασία των λογικών μονάδων να διαμορφώσει τις εξόδους του κυκλώματος.

Σύμφωνα με τα παραπάνω, στα επαναδιατάξιμα συστήματα το υλικό παραμένει σταθερό από την κατασκευή του και μετά και για την απεικόνιση του οποιουδήποτε λογικού κυκλώματος πρέπει να προγραμματιστούν -μέσω λογισμικού- κατάλληλα τόσο η εσωτερική λειτουργία των λογικών μονάδων που απαρτίζουν το υλικό όσο και οι διασυνδέσεις. Με αυτό τον τρόπο επιτυγχάνεται

διπλό όφελος. Αφενός, το υλικό του συστήματος παραμένει στην ουσία σταθερό και η λειτουργία του μπορεί να αλλάξει μέσω του προγραμματισμού, οπότε είναι δυνατή η απεικόνιση οποιουδήποτε κυκλώματος, χωρίς να απαιτείται η από την αρχή κατασκευή του υλικού. Αφετέρου η εκτέλεση των υπολογισμών γίνεται απευθείας στο υλικό, δίνοντας έτσι στα επαναδιατάξιμα συστήματα αυξημένη απόδοση και ταχύτητα.

Επεκτείνοντας λίγο την ανάλυση στο σημείο αυτό, θα αναφερθούν κάποια επιπλέον πλεονεκτήματα που μπορεί ένας σχεδιαστής να αποκομίσει χρησιμοποιώντας τα επαναδιατάξιμα συστήματα. Κατ`αρχήν, η λογική ενός τέτοιου συστήματος μπορεί να αλλάξει όταν και αν είναι αυτό απαραίτητο. Για παράδειγμα, αν διαπιστωθεί κάποιο σφάλμα ή χρειάζεται να γίνει κάποια αναβάθμιση αυτό μπορεί να γίνει με την ίδια ευκολία με την οποία γίνονται οι αντίστοιχες αλλαγές σε κάποια εφαρμογή λογισμικού. Παρομοίως, εύκολα γίνεται η προσαρμογή του συστήματος σε έναν καινούριο αλγόριθμο που πρέπει να απεικονιστεί. Αυτά μεταφράζονται και σε οικονομικό κέρδος, το οποίο διαδραματίζει πρωτεύοντα ρόλο σε περιπτώσεις εμπορικών συστημάτων. Εξάλλου, αισθητά μειωμένος είναι και ο χρόνος που απαιτείται για να περάσει ένα εμπορικό σύστημα υλοποιημένο σε επαναδιατάξιμη λογική στην αγορά, σε σχέση με την αποκλειστική σχεδίαση σε υλικό, όπως στην περίπτωση των ASIC. Στα ASIC άλλωστε απαιτούνται πολλοί και χρονοβόροι κύκλοι σχεδίασης και πρωτυποποίησης πριν την τελική παραγωγή.

Ένα άλλο πλεονέκτημα που παρουσιάζουν τέτοιου είδους συστήματα είναι η δυνατότητα που έχουν να προσαρμόζουν τη λογική τους κατά τη διάρκεια της λειτουργίας τους (at run-time, σύμφωνα με την επικρατούσα ορολογία). Η σχεδίαση του υλικού μπορεί να αλλάζει για να ανταποκριθεί το σύστημα στις ανάγκες που ανακύπτουν κατά την εκτέλεση. Με βάση αυτό το χαρακτηριστικό το επαναδιατάξιμο σύστημα λειτουργεί σαν μια μηχανή εκτέλεσης πολλών διαφορετικών συναρτήσεων υλικού, όπως αντιστοίχως η CPU μπορεί να θεωρηθεί σα μια μηχανή εκτέλεσης πολλών διαφορετικών νημάτων λογισμικού. Για τον παραπάνω λόγο τα επαναδιατάξιμα συστήματα μπορούν να ονομαστούν και επαναδιατάξιμες μονάδες επεξεργασίας (RPU).

Μια άλλη παράμετρος που πρέπει να τονιστεί, είναι ότι με τα επαναδιατάξιμα συστήματα ο σχεδιαστής μπορεί δυνητικά να εκτελέσει παραπάνω λογική από αυτή που υποστηρίζει ο αριθμός των λογικών πυλών του συστήματος. Το γεγονός αυτό είναι ακόμα πιο αποδοτικό όταν υπάρχουν κομμάτια του υλικού που περιστασιακά παραμένουν ανενεργά. Για παράδειγμα αν θεωρήσουμε ένα φανταστικό "έξυπνο" κινητό τηλέφωνο, το οποίο έχει τη δυνατότητα να υποστηρίξει πολλαπλά πρωτόκολλα δεδομένων και επικοινωνίας (ένα τη φορά φυσικά). 'Οταν η συσκευή περάσει από μια γεωγραφική περιοχή που υποστηρίζεται από ένα πρωτόκολλο σε μια περιοχή που υποστηρίζεται από ένα άλλο, τότε το υλικό μπορεί να επαναδιαταχθεί αυτόματα. Με αυτό τον τρόπο είναι δυνατόν να σχεδιαστούν συστήματα που υποστηρίζουν περισσότερες λειτουργίες, κοστίζουν λιγότερο και απαιτούν λιγότερη διαδικασία σχεδιασμού και υλοποίησης. Σε αυτή την περίπτωση, το κόστος που έχει η υποστήριξη

πολλαπλών λειτουργιών, δεν επιβαρύνει το μέγεθος του υλικού αυτού καθεαυτού, αλλά περιορίζεται στο μέγεθος της μνήμης που απαιτείται για να αποθηκευτούν οι διαφορετικές λογικές.

1.4. Επαναδιατάξιμα Συστήματα – Επεξεργαστές SIMD Πίνακα

Τα επαναδιατάξιμα συστήματα, γενικά, επενδύσανε πάνω σε ιδέες που υπήρχαν ήδη στο πεδίο της αρχιτεκτονικής υπολογιστών. Για παράδειγμα, πολλά συστήματα ([9], [10], [11], [12]) έχουν πίνακες από μονάδες επεξεργασίας οργανωμένες είτε με το SIMD (Single-Instruction Multiple-Data) είτε με το MIMD (Multiple-Instruction Multiple-Data) μοντέλο. Τα συστήματα αυτά άντλησαν διάφορα στοιχεία από την έρευνα που έχει γίνει για τους επεξεργαστές με πίνακα SIMD. Υπάρχουν πολλοί λόγοι για την επανεμφάνιση της SIMD προσέγγισης (μετά την αποτυχία των αρχικών SIMD επεξεργαστών να αποκτήσουν ευρεία χρήση). Τα τελευταία χρόνια έκαναν την εμφάνισή τους ως κυρίαρχες εφαρμογές πολλές εργασίες με εντατικούς υπολογισμούς και υψηλό εύρος ζώνης (όπως επεξεργασία εικόνων, βίντεο). Αυτές οι εργασίες μπορούν να υλοποιηθούν αποδοτικά σε αρχιτεκτονικές SIMD. Ταυτόχρονα, η ανάπτυξη της τεχνολογίας VLSI επιτρέπει σε συστήματα που στο παρελθόν η ανάπτυξή τους είχε υψηλό -μη αποδεκτό- κόστος, σήμερα να παράγονται φτηνά σε ένα μόνο κομμάτι πυριτίου.

Παρόλα αυτά, υπάρχουν ακόμα πολλές διαφορές μεταξύ των πρωτοεμφανιζόμενων επεξεργαστών με πίνακα SIMD και των επαναδιατάξιμων συστημάτων που βασίζονται στο μοντέλο υπολογισμού SIMD. Μια σημαντική διαφορά είναι ότι τα επαναδιατάξιμα συστήματα είναι διαμορφώσιμα από την άποψη της λειτουργικότητας και των διασυνδέσεων.

1.5. Δυναμικά Επαναδιατάξιμες Αρχιτεκτονικές Επεξεργαστή

1.5.1. Γενικά

Τα FPGAs παρουσιάστηκαν ως μια τεχνολογική εξέλιξη των παλαιότερων PLD (προγραμματιζόμενη λογική μονάδα, *Programmable Logic Device*) και PAL (λογική προγραμματιζόμενου πίνακα, *Programmable Array Logic*) επαναδιατάξιμων διατάξεων. Παρουσίασαν εξαιρετικές επιδόσεις από πλευράς επιτάχυνσης και ρυθμού απόδοσης σε μια μεγάλη ποικιλία εφαρμογών, όπως για παράδειγμα η κρυπτογράφηση/αποκρυπτογράφηση δεδομένων. Παρόλα αυτά η ανάπτυξη με συστήματα FPGA παρουσιάζει κάποια μειονεκτήματα, με βασικότερο απ`όλα ότι τα FPGA είναι μεν επαναδιατάξιμα αλλά όχι δυναμικά (run-time) επαναδιατάξιμα. Αυτό σημαίνει ότι η επαναδιάταξη του υλικού δεν

μπορεί να γίνει κατά τη διάρκεια της εκτέλεσης ενός αλγορίθμου στο FPGA, αλλά αντίθετα θα πρέπει να σταματήσει η εκτέλεση και να γίνει η επαναδιάταξη. Παρόλα αυτά, πρέπει να σημειωθεί ότι τα τελευταίας τεχνολογίας FPGAs επιτρέπουν τη δυναμική επαναδιατάξη. Επιπλέον, τα κλασσικά FPGAs διαβάζουν τη διαμόρφωσή τους από μία σειριακή μνήμη EEPROM, ένα bit τη φορά. Και αυτή η διαδικασία απαιτεί να έχει ενεργοποιηθεί ένα σήμα επαναφοράς (reset). Αυτό σημαίνει ότι το FPGA μπορεί να επαναπρογραμματιστεί μόνο στην ολότητά του και όχι μερικώς και επιπλέον η προηγούμενη κατάστασή του δεν μπορεί να διατηρηθεί πριν την αλλαγή.

Επιπροσθέτως, ένα σημαντικό αρνητικό στοιχείο είναι η χωρητικότητα των FPGAs σε αριθμό απεικονιζόμενων πυλών. Μέχρι μερικά χρόνια πριν, οι αλγόριθμοι που μπορούσαν να υλοποιηθούν σε ένα μοναδικό FPGA ήταν σημαντικά μικροί, ώστε να μπορέσουν τα FPGAs να ανταγωνιστούν με σχετική αποδοτικότητα τους superscalar επεξεργαστές, ωστόσο αυτή η σχέση αλλάζει ήδη. Το 1995, για παράδειγμα, το μεγαλύτερο FPGA μπορούσε να προγραμματιστεί για κυκλώματα περίπου 15000 λογικών πυλών το μέγιστο. Από τη στιγμή που ένας γρήγορος αθροιστής των 32 bit απαιτεί μερικές εκατοντάδες πύλες, οι δυνατότητες τέτοιων συσκευών ήταν περιορισμένες. Τελευταία, ωστόσο, τα FPGA έχουν φτάσει σε μέγεθος όπου είναι δυνατό να υλοποιήσουν κάποια σημαντικά υποτμήματα μιας εφαρμογής σε ένα και μόνο ολοκληρωμένο. Για την ακρίβεια, τα μικροηλεκτρονικά κυκλώματα συνεχίζουν να γίνονται μικρότερα και ταχύτερα και τα FPGAs έχουν εκμεταλλευτεί το γεγονός αυτό στο έπακρο, καθώς εκτός από πιο γρήγορα (λόγω ταχύτερων transistor και διασυνδέσεων) το μικρότερο μέγεθος των κυκλωμάτων αυξάνει την πυκνότητα των στοιχείων που μπορούν να τοποθετηθούν σε μια συγκεκριμένη επιφάνεια πυριτίου.

Αν ένας επεξεργαστής μπορούσε να περιλαμβάνει μία ή παραπάνω συσκευές σαν τα FPGA, θα μπορούσε θεωρητικά να υποστηρίξει ένα κύκλωμα εξειδικευμένο σε μια συγκεκριμένη εφαρμογή για κάθε πρόγραμμα ή ακόμα για κάθε στάδιο της εκτέλεσης ενός προγράμματος. Οι απεριόριστη επαναδιαταξιμότητα ενός FPGA επιτρέπει τη συνεχή εφαρμογή μιας σειράς κυκλωμάτων ειδικής σχεδίασης, το κάθε ένα εξειδικευμένο στην εργασία της στιγμής.

Σήμερα πολλά ενσωματωμένα συστήματα παρουσιάζονται ως συνδυασμός ενός κομματιού επαναδιατάξιμης λογικής και ενός γενικού σκοπού μικροεπεξεργαστή. Σε ένα τέτοιο σύστημα ο πυρήνας τον υπολογισμών, που αποτελεί και το πιο απαιτητικό -υπολογιστικά- κομμάτι της εφαρμογής, απεικονίζεται στο επαναδιατάξιμο υλικό. Από την άλλη, ο επεξεργαστής αναλαμβάνει την υλοποίηση των λειτουργιών εκείνων που δεν μπορούν να εκτελεστούν αποδοτικά στην επαναδιατάξιμη λογική, όπως προσπελάσεις στη μνήμη καθώς και η ροή ελέγχου. Επίσης, είναι υπεύθυνος για τον έλεγχο της επαναδιατάξιμης λογικής. Αυτού του τύπου οι αρχιτεκτονικές ονομάζονται δυναμικά επαναδιατάξιμες αρχιτεκτονικές επεξεργαστή. Η επαναδιατάξιμη λογική τέτοιων συστημάτων μπορεί να αποτελείται είτε από ένα ή περισσότερα εμπορικά

διαθέσιμα FPGAs είτε από κάποιο επαναδιατάξιμο κομμάτι κυκλώματος ειδικής αρχιτεκτονικής αρχιτεκτονικής, που έχει σχεδιαστεί αποκλειστικά για το σκοπό αυτό. Στην πρώτη περίπτωση, ωστόσο, η περιορισμένη χωρητικότητα της συσκευής και το μεγάλο μέγεθος του κυκλώματος διασυνδέσεων που απαιτούνται μπορεί να είναι σημαντικός αρνητικός παράγοντας. Το επαναδιατάξιμο υλικό συνήθως λειτουργεί σα συν-επεξεργαστής με τον κεντρικό επεξεργαστή. Λόγω του ότι η παρούσα εργασία αφορά στη σχεδίαση ενός τέτοιου επαναδιατάξιμου συστήματος, στη συνέχεια θα αναλυθούν ορισμένα βασικά χαρακτηριστικά των αρχιτεκτονικών τέτοιου είδους.

1.5.2. Ιδιότητες Σχεδίασης

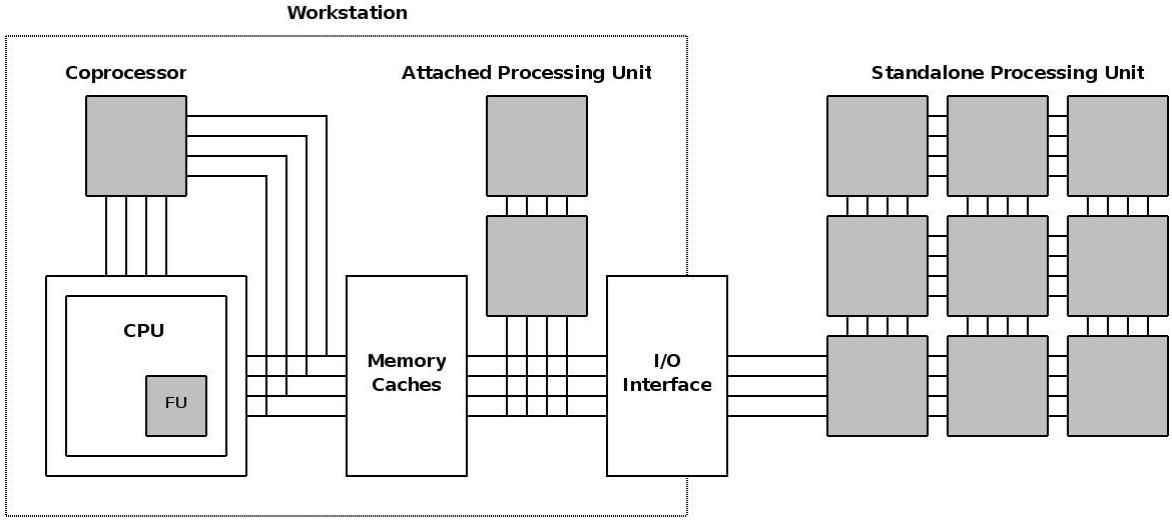
Τέσσερα είναι τα βασικά σημεία της σχεδίασης επαναδιατάξιμων συστημάτων, στα οποία πρέπει να δοθεί ιδιαίτερη σημασία.

Αυτά είναι :

- το ιεραρχικό επίπεδο συνεργασίας μεταξύ του επεξεργαστή γενικού σκοπού και του επαναδιατάξιμου τμήματος του κυκλώματος
- η πολυπλοκότητα των δομικών μονάδων της επαναδιατάξιμης λογικής
- το κύκλωμα που επιτελεί τις διασυνδέσεις μεταξύ των λογικών μονάδων και
- το αν θα υπάρχει η δυνατότητα για δυναμική (run-time) επαναδιάταξη.

1.5.2.1. Συνεργασία Επαναδιατάξιμης Λογικής-Επεξεργαστή

Όπως αναφέρθηκε, στα επαναδιατάξιμα συστήματα οι παραδοσιακοί μικροεπεξεργαστές χρησιμοποιούνται για να εκτελείται μια εφαρμογή πιο αποδοτικά. Κι αυτό γιατί έχει διαπιστωθεί πως η προγραμματιζόμενη λογική τείνει να είναι μη αποδοτική όταν υλοποιεί συγκεκριμένους τύπους λειτουργιών, όπως οι μεταβλητού μήκους βρόχοι. Έτσι, συγκεκριμένα κομμάτια του προγράμματος που δεν μπορούν να απεικονιστούν εύκολα στην επαναδιατάξιμη λογική, εκτελούνται στον μικροεπεξεργαστή. Ωστόσο, υπάρχουν διάφοροι τρόποι όσον αφορά στο πώς θα συνεργάζονται οι δύο διαφορετικές λειτουργικές μονάδες, το επαναδιατάξιμο υλικό και ο μικροεπεξεργαστής. Το Σχήμα 1.2 παρέχει ένα κατατοπιστικό σχεδιάγραμμα, για τις επιλογές που υπάρχουν για τα επίπεδα ιεράρχησης.



Σχήμα 1.2: Διαφορετικά επίπεδα συνεργασίας σε ένα επαναδιατάξιμο σύστημα. Η επαναδιατάξιμη λογική είναι σκιασμένη.

- Πρώτον, επαναδιατάξιμο υλικό μπορεί να χρησιμοποιηθεί αποκλειστικά για να παρέχει επαναδιατάξιμες λειτουργικές μονάδες μέσα στον κεντρικό μικροεπεξεργαστή. Με τον τρόπο αυτό υπάρχει η δυνατότητα για προσθήκη εντολών που να μεταβάλλουν τη λειτουργία τους με την πάροδο του χρόνου. Αποτελεί δε το χαμηλότερο επίπεδο συνεργασίας, στο οποίο το επαναδιατάξιμο υλικό ενσωματώνεται σα ξεχωριστή λειτουργική μονάδα στο datapath του κεντρικού μικροεπεξεργαστή.
- Δεύτερον, η επαναδιατάξιμη μονάδα μπορεί να χρησιμοποιηθεί ως συν-επεξεργαστής με τον κεντρικό μικροεπεξεργαστή. Ένας συν-επεξεργαστής είναι γενικά μεγαλύτερος μιας λειτουργικής μονάδας και είναι ικανός να εκτελεί υπολογισμούς χωρίς το συνεχή έλεγχο και την παρέμβαση του κεντρικού επεξεργαστή. Σε αυτή την περίπτωση συνεργασίας ο κεντρικός επεξεργαστής αρχικοποιεί το επαναδιατάξιμο υλικό και παρέχει τα δεδομένα που απαιτούνται. Στη συνέχεια, το επαναδιατάξιμο υλικό εκτελεί τους υπολογισμούς ανεξάρτητα από τον κεντρικό επεξεργαστή και επιστρέφει πίσω το τελικό (ή και τα ενδιάμεσα) αποτέλεσμα. Το πλεονέκτημα αυτού του τύπου της ιεράρχησης είναι ότι δίνει στην επαναδιατάξιμη λογική τη δυνατότητα να εκτελεί μεγάλο αριθμό κύκλων χωρίς την παρέμβαση του κεντρικού επεξεργαστή και ότι επιτρέπει μεγάλο βαθμό παραλληλίας. Με τον τρόπο αυτό μειώνονται σημαντικά τα μειονεκτήματα αυτονομίας που εισάγονται από την επαναδιατάξιμη λογική, σε σύγκριση με την προηγούμενη περίπτωση όπου η επαναδιατάξιμη λειτουργική μονάδα πρέπει να επικοινωνεί με τον κεντρικό επεξεργαστή, κάθε φορά που υπάρχει μια "επαναδιατάξιμη" εντολή προς εκτέλεση.
- Στην επόμενη -προς τα πάνω- ιεραρχική οργάνωση, η επαναδιατάξιμη μονάδα επεξεργασίας προσκολλάται στη διεπαφή εισόδου-εξόδου και συμπεριφέρεται σαν ένας επιπλέον επεξεργαστής σε ένα σύστημα πολλαπλών επεξεργαστών. Σε αυτή την περίπτωση εμφανίζεται αυξημένη καθυστέρηση στην επικοινωνία μεταξύ του κεντρικού επεξεργαστή και του επαναδιατάξιμου υλικού, διότι η εσωτερική μνήμη του πρώτου δεν είναι

άμεσα προσπελάσιμη από το δεύτερο. Ωστόσο αυτός ο τύπος ιεράρχησης παρέχει μεγάλο βαθμό ανεξαρτησίας, καθώς είναι δυνατόν να μετακινηθούν μεγάλες ποσότητες υπολογισμών προς εκτέλεση στο επαναδιατάξιμο κομμάτι του συστήματος.

- Στο τελευταίο επίπεδο ιεράρχησης σχεδιάζεται ένα επαναδιατάξιμο κομμάτι υλικού το οποίο λειτουργεί σαν μια ανεξάρτητη (stand-alone) μονάδα επεξεργασίας. Σε αυτή την περίπτωση η επικοινωνία του επαναδιατάξιμου υλικού με τον κεντρικό επεξεργαστή είναι αραιή. Το μοντέλο αυτό υπολογισμού μοιάζει με την περίπτωση των σταθμών εργασίας σε ένα δίκτυο, όπου η επεξεργασία μπορεί να διαρκέσει μεγάλο χρονικό διάστημα χωρίς να απαιτείται ενδιάμεσα κάποιου είδους επικοινωνία.

Συνεπώς, υπάρχουν πολλές μέθοδοι για την οργάνωση της ιεράρχησης μεταξύ του προγραμματιζόμενου και μη κομματιού ενός επαναδιατάξιμου συστήματος, κάθε μία από τις οποίες έχει συγκεκριμένα πλεονεκτήματα και μειονεκτήματα. Όσο χαμηλότερα -ιεραρχικά- είναι η ενσωμάτωση της επαναδιατάξιμης λογικής με τον κεντρικό επεξεργαστή, τόσο πιο συχνά μπορεί να της παραχωρείται κάποια λειτουργία μιας εφαρμογής για εκτέλεση, λόγω του ότι υπάρχει λιγότερο κόστος -χρονικό κυρίως- κατά την επικοινωνία της με τον κεντρικό επεξεργαστή. Το μειονέκτημα της οργάνωσης αυτής είναι ότι η επαναδιατάξιμη λογική εξαρτάται πολύ από τις παρεμβάσεις του κεντρικού επεξεργαστή, με αποτέλεσμα η επαναδιατάξιμη μονάδα να μένει πολλές φορές σε άεργη κατάσταση αναμένοντας κάποιου είδους διαδραστικότητα με τον κεντρικό επεξεργαστή. Επίσης, το μέγεθος της επαναδιατάξιμης λογικής που είναι διαθέσιμο είναι σημαντικά μικρό. Αντιθέτως, όσο πιο απομακρυσμένη είναι η διασύνδεση των δύο μονάδων, τόσο μεγαλύτερη παραλληλία εκτέλεσης υποστηρίζεται αλλά από την άλλη η επικοινωνία μεταξύ των μονάδων είναι πιο χρονοβόρα. Μάλιστα το τελευταίο στοιχείο είναι τόσο σημαντικό, που πολλές εφαρμογές που χρειάζονται μεγάλο επικοινωνιακό φόρτο μπορεί να επιταχύνονται ελάχιστα ή και καθόλου από την απεικόνισή τους σε επαναδιατάξιμα συστήματα.

1.5.2.2. Βαθμός Επαναδιάταξης (Logic Block Granularity)

Όπως αναφέρθηκε παραπάνω, τα περισσότερα επαναδιατάξιμα συστήματα αποτελούνται από ένα σύνολο μονάδων υπολογισμού, οι οποίες επαναλαμβάνονται και διαμορφώνουν έναν επαναδιατάξιμο πίνακα. Αυτές οι μονάδες, από τις οποίες δομούνται τα επαναδιατάξιμα συστήματα, ονομάζονται συνήθως λογικά μπλοκ ή κελιά. Η πολυπλοκότητα των κελιών αυτών μπορεί να διαφέρει σημαντικά από περίπτωση σε περίπτωση και ποικίλει μεταξύ ενός πολύ μικρού και απλού κελιού που μπορεί να υπολογίσει μια απλή boolean συνάρτηση τριών εισόδων, έως μια δομή που να παρομοιάζεται με τη λειτουργία μιας ALU ή ακόμη κι ενός μικροεπεξεργαστή. Επίσης, η έννοια της επαναδιαταξιμότητας εμφανίζεται με διπλή σημασία, καθώς αφενός ένας τύπος κελιών μπορεί να είναι

επαναδιατάξιμος με την έννοια ότι η λειτουργία που επιτελούν καθορίζεται από τα δεδομένα διαμόρφωσης. Αφετέρου, άλλα κελιά έχουν σταθερή δομή και η επαναδιαταξιμότητά τους έγκειται στη διαφοροποίηση των μεταξύ τους διασυνδέσεων. Ο όρος βαθμός επαναδιατάξης (granularity) για ένα κελί, συνεπώς, αναφέρεται στο μέγεθος και στην πολυπλοκότητα των βασικών λειτουργικών μονάδων ενός επαναδιατάξιμου συστήματος.

Στην πραγματικότητα, τα επαναδιατάξιμα συστήματα διαχωρίζονται σε δύο κατηγορίες, αναφορικά με το βαθμό επαναδιάταξης. Οι δύο αυτές κατηγορίες είναι τα συστήματα με βαθμό επαναδιάταξης επιπέδου bit (fine-grained) και τα συστήματα με βαθμό επαναδιάταξης επιπέδου λέξης (coarse-grained). Παρακάτω θα γίνει σύντομη περιγραφή των δύο αυτών διαφορετικών κατηγοριών, θα παρουσιαστούν τα πλεονεκτήματα/μειονεκτήματα καθενός από αυτά και θα δοθούν μερικά παραδείγματα συστημάτων που έχουν αναπτυχθεί σε κάθε περίπτωση.

Αρχιτεκτονικές επιπέδου bit

Τα αρχικά επαναδιατάξιμα συστήματα βασίζονταν σε FPGAs, είτε αυτά ήταν ανεξάρτητα συστήματα ή το επαναδιατάξιμο κομμάτι σε αρχιτεκτονικές με την παρουσία γενικού σκοπού μικροεπεξεργαστή. Ο προγραμματισμός γίνεται στο επίπεδο του bit και για το λόγο αυτό τα συγκεκριμένα συστήματα αποτελούν μια αρχιτεκτονική επιπέδου λέξης. Ένα χαρακτηριστικό παράδειγμα στην κατηγορία αυτή είναι η σειρά 6200 της Xilinx, όπου οι δομικές λειτουργικές μονάδες μπορούν να υλοποιήσουν οποιαδήποτε λογική πύλη δύο εισόδων (και κάποιες τριών εισόδων). Λόγω και της παραπάνω απλότητας, κυκλώματα τα οποία είναι εύκολο να απεικονιστούν με έναν λογικό αριθμό δομικών μονάδων που ελέγχονται bit προς bit εμφανίζουν μεγάλη αποδοτικότητα. Ωστόσο, με την εμφάνιση πιο πολύπλοκων λειτουργιών που έπρεπε να απεικονιστούν (όπως για παράδειγμα ο πολλαπλασιασμός ή οι γεννήτριες καταστάσεων), οι αρχιτεκτονικές επιπέδου bit άρχισαν να αποκτούν σημαντικά αρνητικά στοιχεία, καθώς οι συγκεκριμένες σύνθετες λειτουργίες δεν υλοποιούνται αποδοτικά σε τέτοια συστήματα. Μεταξύ των άλλων, κάποια αρνητικά στοιχεία των αρχιτεκτονικών επιπέδου bit είναι ότι :

- Απαιτείται μεγάλη ποσότητα δεδομένων διαμόρφωσης, το οποίο κάνει τη γρήγορη διαμόρφωση πολύ δύσκολη να υλοποιηθεί και αυξάνει την κατανάλωση ισχύος κατά τη διάρκεια της διαμόρφωσης και
- Καθώς οι λειτουργίες που απεικονίζονται αποτελούνται από πολλές λογικές μονάδες, χάνεται το στοιχείο της ομοιομορφίας και εμφανίζεται επιπλέον απώλεια χρόνου (απόδοσης) λόγω των μακρινών διασυνδέσεων. Αυτές οι έξτρα διασυνδέσεις που απαιτούνται αυξάνουν επίσης την κατανάλωση ισχύος κατά τη διάρκεια λειτουργίας του κυκλώματος.

Αρχιτεκτονικές επιπέδου λέξης

Οι αρχιτεκτονικές επιπέδου λέξης παρουσιάζουν έντονο ερευνητικό ενδιαφέρον τα τελευταία χρόνια. Δίνουν τη δυνατότητα στα συστήματα που εφαρμόζουν τη

λογική αυτή να είναι περισσότερο επαρκή για υπολογιστικές εργασίες, εξαιτίας της καλύτερης αποδοτικότητάς τους και της μεγαλύτερης ταχύτητας. Ωστόσο, εξαιτίας του ότι στα συστήματα επιπέδου λέξης η ευελιξία του κυκλώματος είναι μειωμένη, είναι δύσκολο να διαμορφωθεί μια γενικευμένη αρχιτεκτονική για τέτοιου είδους συστήματα.

Τα συστήματα αυτά είναι ικανά να υλοποιούν υψηλού επιπέδου λειτουργίες, προσφέροντας ένα χειριστή δεδομένων (datapath) με πλάτος μεγαλύτερου του ενός bit. Γενικά, τα συστήματα με αρχιτεκτονική επιπέδου λέξης υλοποιούν μεγάλους και πολύπλοκους υπολογισμούς, με ταχύτητα και αποδοτικότητα, καταναλώνοντας μικρότερη επιφάνεια πυριτίου, αφού τα λογικά κελιά τους είναι βελτιστοποιημένα για τέτοιου είδους υπολογισμούς. Οι αρχιτεκτονικές επιπέδου λέξης προσέφεραν λύση στα περισσότερα αρνητικά σημεία που εμφανίζουν οι αντίστοιχες επιπέδου bit.

Ορισμένα, λοιπόν, από τα πλεονεκτήματα που εμφανίζουν οι αρχιτεκτονικές επιπέδου λέξης είναι:

- Απαιτούνται λιγότερα δεδομένα διαμόρφωσης, καθώς τυπικά παρέχονται εσωτερικά πολύ λιγότεροι -αλλά συνάμα πολύ ισχυρότεροι- τελεστές σε σύγκριση π.χ. με ένα FPGA. Αυτό οδηγεί και στη μείωση του απαιτούμενου μεγέθους της μνήμης διαμόρφωσης.

- Τα κυκλώματα διασύνδεσης μεταφέρουν λέξεις αντί για bit, οι οποίες απαιτούν λιγότερα σήματα ελέγχου αλλά και χώρο αποθήκευσης αφιερωμένο αποκλειστικά για τα δεδομένα διαμόρφωσης. Επίσης, ο μικρότερος αριθμός δεδομένων διαμόρφωσης υποστηρίζει τη δυνατότητα γρήγορης φόρτωσης και δυναμικής μερικής επαναδιάταξης.

- Οι αρχιτεκτονικές επιπέδου λέξης συνήθως απαιτούν λιγότερη επιφάνεια υλικού για να υλοποιηθούν, είναι δηλαδή περισσότερο αποδοτικές απο πλευράς χώρου και αυτό προκύπτει τόσο από τα μειωμένα κυκλώματα για τις διασυνδέσεις, όσο και για αυτούς καθεαυτούς του λογικούς τελεστές. Είναι χαρακτηριστικό ότι οι ισχυροί τελεστές, όπως κυκλώματα αριθμητικές-λογικές μονάδες (ALU) ή πολλαπλασιαστές-συσσωρευτές (Multiplier-ACummulator, MAC), υλοποιούνται απευθείας στο πυρίτιο, αντί να απεικονιστούν πάνω σε λογική χαμηλού επιπέδου bit.

- Τέλος, τα συστήματα αρχιτεκτονικής επιπέδου λέξης ταιριάζουν καλύτερα στην τεχνική ανάπτυξης εφαρμογών μέσω μιας υψηλού επιπέδου γλώσσας.

Ωστόσο, υπάρχουν κάποια σημαντικά προβλήματα που πρέπει να λυθούν, ώστε να ξεπεραστεί η μειωμένη ευελιξία σε σύγκριση με λύσεις σχεδίασης που βασίζονται σε FPGAs. Τέτοια είναι :

- Ενώ το datapath πολαπλών bits εφαρμόζεται καλά σε τελεστές

προερχόμενους από τις υψηλού επιπέδου γλώσσες, άλλες λειτουργίες που δουλεύουν με μικρότερο μήκος λέξης ή χρειάζονται να εκτελέσουν τροποποιήσεις σε επίπεδο bit, είτε υποστηρίζονται ελάχιστα είτε απαιτούν μια πολύπλοκη αρχιτεκτονική για να υλοποιηθούν. Όταν εμφανίζονται τέτοιες λειτουργίες, όπως για παράδειγμα στους αλγορίθμους συμπίεσης δεδομένων, αυτό έχει ως αποτέλεσμα να ανακύπτουν δύσκολες απαιτήσεις από πλευράς αρχιτεκτονικής ή να απαιτούνται πολλά στοιχεία επεξεργασίας.

- Παρόλο που η κατανάλωση ισχύος που προκαλείται από τα κυκλώματα διασύνδεσης είναι σαφώς μικρότερη από εκείνη στα FPGAs, παραμένει ακόμα το πρόβλημα για μια προσεκτική αρχιτεκτονική σχεδίαση του δικτύου διασυνδέσεων -εργασία όχι τόσο απλή- που να παρέχει χαμηλή ισχύ και επαρκή ευελιξία.

Ένα χαρακτηριστικό παράδειγμα συστήματος με αρχιτεκτονική επιπέδου λέξης είναι η αρχιτεκτονική RaPiD[13] και η αρχιτεκτονική Chameleon[10]. Κάθε ένα από τα συστήματα αυτά αποτελείται από αθροιστές, πολλαπλασιαστές και καταχωρητές μεγέθους λέξης. Αν για παράδειγμα σε κάποιον υπολογισμό απαιτούνται μόλις τρεις τιμές του 1-bit, τότε η χρήση τέτοιων αρχιτεκτονικών πάσχει από αχρησιμοποίητη επιφάνεια πυριτίου και σημαντικές απώλειες ταχύτητας, δεδομένου ότι θα υπολογιστούν τα πλήρη bit της λέξης στα αποτελέσματα, παρόλο που δεν χρησιμοποιούνται όλα. Παρόλα αυτά, οι αρχιτεκτονικές αυτές είναι περισσότερο αποδοτικές από αρχιτεκτονικές επιπέδου bit για την υλοποίηση συναρτήσεων που απαιτούν αποτελέσματα πιο κοντά στο πλήρες εύρος bit τους.

1.5.2.3. Κυκλώματα Διασύνδεσης

Τα κυκλώματα διασύνδεσης σε μια επαναδιατάξιμη αρχιτεκτονική χρησιμοποιούνται προκειμένου να συνδεθούν μεταξύ τους οι προγραμματιζόμενες λογικές μονάδες του συστήματος. Αυτά τα κυκλώματα συνήθως έχουν τη δυνατότητα να διαταχθούν κατά το δοκούν, ανάλογα με τις απαιτήσεις του εκάστοτε κυκλώματος που πρέπει να υλοποιηθεί. Ο καθορισμός των διαδρομών που πρέπει να ακολουθήσουν τα διάφορα σήματα μέσα στο σύστημα πραγματοποιείται κατά τη διάρκεια της λειτουργίας του κυκλώματος ή κατά τη μεταγλώττιση και όχι στη φάση της κατασκευής. Αυτή η ευελιξία των διασυνδέσεων επιτρέπει την απεικόνιση μιας μεγάλης ποικιλίας δυνατών κυκλωματικών διατάξεων.

Γενικά απαιτείται μια προσεκτική επιλογή του τύπου των κυκλωμάτων διασύνδεσης, καθώς όσο πιο πολύπλοκη είναι η λογική τόσο πιο μεγάλη είναι η επιφάνεια του πυριτίου που πρέπει να χρησιμοποιηθεί. Αν τα διαθέσιμα καλώδια είναι πολύ μακρύτερα απ`ότι χρειάζεται για την οδήγηση ενός σήματος, το παραπάνω μήκος καλωδίου σπαταλάται. Από την άλλη αν τα καλώδια είναι

κοντύτερα από τα απαιτούμενα, τα σήματα θα πρέπει αναγκαστικά να περάσουν μέσα από μεταγωγείς (σαν τα switches στα δίκτυα υπολογιστών) που συνδέουν τα κοντά καλώδια μεταξύ τους, κάτι το οποίο εισάγει επιπλέον καθυστερήσεις και μειώνει την απόδοση του όλου κυκλώματος.

Τέλος θα αναφέρουμε τους δύο βασικούς τρόπους που υπάρχουν για την οργάνωση των -τοπικών και μη- διασυνδέσεων.

- Πρώτον, η *τμηματοποιημένη διασύνδεση* (segmented routing), όπου υπάρχουν καλώδια κοντινών αποστάσεων για να εξυπηρετήσουν την τοπική κίνηση. Αυτά τα καλώδια συνδέονται μεταξύ τους σε κάποια σημεία που παίζουν το ρόλο μεταγωγέων, έτσι ώστε να δημιουργηθούν μακρύτερα καλώδια.

- Δεύτερον, η *ιεραρχική διασύνδεση* (hierarchical routing) για την παροχή τοπικής και μη επικοινωνίας. Το κύκλωμα των λογικών μονάδων του συστήματος χωρίζεται νοητά (αλλά ανάλογα γίνεται και η χωρική τοποθέτησή τους πάνω στο πυρίτιο) σε ομάδες και οι ομάδες στο μεγαλύτερο επίπεδο ιεράρχησης σε δέσμες ομάδων. Μέσα σε μια ομάδα οι λογικές μονάδες διασυνδέονται μόνο στο τοπικό επίπεδο, δηλαδή μια λογική μονάδα συνδέεται μόνο με λογικές μονάδες της ίδιας ομάδας. Στα όρια των ομάδων υπάρχουν ωστόσο κάποια μακρύτερα καλώδια που συνδέουν διαφορετικές ομάδες μεταξύ τους. Η ίδια λογική επαναλαμβάνεται και σε μεγαλύτερο επίπεδο. Η ιδέα πίσω από την ιεραρχική διασύνδεση είναι ότι δεδομένου μιας προσεκτικής τοποθέτησης των διασυνδέσεων και μιας προσεκτικής οργάνωσης, η περισσότερη κίνηση πρέπει να μείνει στο τοπικό επίπεδο και μόνο μια περιορισμένη ποσότητα κίνησης να μετακινείται σε μεγάλες αποστάσεις.

1.5.2.4. Δυναμική (run-time) Επαναδιάταξη

Επειδή τα κομμάτια ενός προγράμματος τα οποία μπορούν να επιταχυνθούν με την απεικόνισή τους σε επαναδιατάξιμο υλικό είναι -συνήθως- μεγάλα και πολύπλοκα για να φορτωθούν ταυτόχρονα στο διαθέσιμο υλικό, είναι χρήσιμο να υπάρχει η δυνατότητα να εναλλάσσονται διαφορετικές διαμορφώσεις του υλικού κατά τη διάρκεια της εκτέλεσης του προγράμματος, ανάλογα με τις απαιτήσεις. Αυτή η ιδέα είναι γνωστή σαν δυναμική επαναδιάταξη (run-time reconfiguration, RTR).

Η τεχνική αυτή βασίζεται στην ιδέα του "εικονικού υλικού" (virtual hardware), που είναι παρόμοια με την εικονική μνήμη. Σύμφωνα με αυτή το φυσικό υλικό είναι πολύ μικρότερο από το άθροισμα των πόρων που απαιτούνται από κάθε διαμόρφωση. Παρόλα αυτά, αντί να μειωθεί ο αριθμός των διαμορφώσεων που απεικονίζονται ή να αυξηθεί η επιφάνεια του υλικού, αυτό που συμβαίνει είναι η εναλλαγή των διαφόρων διαμορφώσεων κατά τη διάρκεια της εκτέλεσης του

προγράμματος στο υλικό. Επειδή λοιπόν αυτή η τεχνική επιτρέπει να απεικονίζονται στο υλικό περισσότερα κομμάτια ενός προγράμματος από αυτά που μπορούν να "χωρέσουν" σε ένα επαναδιατάξιμο σύστημα που δεν υποστηρίζει δυναμική επαναδιάταξη, είναι δυνατόν να επιτευχθεί μεγαλύτερη επιτάχυνση του προγράμματος, άρα και βελτίωση της απόδοσης της υλοποίησης και καλύτερη χρησιμοποίηση του υλικού.

Όταν χρησιμοποιείται αυτή η τεχνική στα επαναδιατάξιμα συστήματα είναι πολύ πιθανό κάποιες διαμορφώσεις να χρειάζονται πρόσβαση σε αποτελέσματα άλλων διαμορφώσεων, οι οποίες συνήθως είναι ενεργές σε διαφορετικές χρονικές περιόδους. Για το λόγο αυτό είναι αναγκαία η χρήση κάποιων καταχωρητών που να μένουν αμετάβλητοι μεταξύ των διαμορφώσεων. Αυτοί οι καταχωρητές μπορούν να είναι είτε μέσα στον επαναδιατάξιμο πίνακα είτε εξωτερικοί. Είναι δυνατόν αντί για καταχωρητές να χρησιμοποιείται η εξωτερική μνήμη του συστήματος.

Να σημειωθεί ότι υπάρχουν πολλά μοντέλα για να εφαρμοστεί η τεχνική που περιγράφεται στην ενότητα αυτή, όπως η διαμόρφωση μοναδικού περιεχομένου, η μερική επαναδιαταξιμότητα ή η διαμόρφωση πολλαπλού περιεχομένου, χωρίς ωστόσο να προχωρήσει η ανάλυση σε παραπάνω λεπτομέρειες[11].

1.6. Κατηγοριοποίηση Επαναδιατάξιμων Συστημάτων και Προηγούμενη Δουλειά

Η ενότητα αυτή εισάγει ένα σύνολο κριτηρίων που χρησιμοποιούνται συχνά για να χαρακτηρίσουν τη σχεδίαση ενός επαναδιατάξιμου συστήματος υπολογισμού. Αυτά τα κριτήρια είναι ο *βαθμός επαναδιάταξης* (granularity), το *βάθος της δυνατότητας προγραμματισμού* (depth of programmability), η *επαναδιαταξιμότητα* (reconfigurability), το *σύστημα διεπαφής* (interface) και το *μοντέλο του υπολογισμού* (model of computation).

1. Βαθμός Επαναδιατάξης : Αυτός ο όρος αναφέρεται στο μέγεθος των δεδομένων που χρησιμοποιούνται στις λειτουργίες της επαναδιατάξιμης μονάδας επεξεργασίας (Reconfigurable Processing Unit, RPU) ενός συστήματος. Μια επαναδιατάξιμη μονάδα επεξεργασίας είναι ένα λογικό μπλοκ από προγραμματιζόμενη λειτουργικότητα. Στα συστήματα αρχιτεκτονικής επιπέδου bit (*fine-grained*) τα στοιχεία επεξεργασίας στο RPU είναι συνήθως λογικές πύλες, flip-flops και πίνακες αναζήτησης (look-up table, LUT). Αυτά λειτουργούν στο επίπεδο του bit, υλοποιώντας μια boolean συνάρτηση. Από την άλλη, στα συστήματα αρχιτεκτονικής επιπέδου λέξης (*coarse-grained*) τα στοιχεία επεξεργασίας στο RPU μπορεί να περιέχουν πολύπλοκες λειτουργικές μονάδες, όπως ALUs ή/και πολλαπλασιαστές, οι οποίες λειτουργούν με λέξεις πολλαπλών bits. Υπάρχουν και συστήματα όπου συμπλέκονται και οι δύο παραπάνω κατηγορίες βαθμού επαναδιάταξης και οι οποίες χαρακτηρίζονται ως

μικτού βαθμού επαναδιατάξιμες αρχιτεκτονικές (mixed-grained).

2. Βάθος Δυνατότητας Προγραμματισμού : Αναφέρεται στον αριθμό των προγραμμάτων διαμόρφωσης (ή αλλιώς *contexts*) που αποθηκεύονται μέσα στο RPU. Ένα RPU μπορεί να έχει ένα μοναδικό ή πολλαπλά περιεχόμενα διαμόρφωσης. Στα συστήματα μοναδικού περιεχομένου διαμόρφωσης (*single-context*), μόνο ένα πρόγραμμα διαμόρφωσης υπάρχει πάνω στο RPU. Επομένως, η λειτουργικότητα του RPU περιορίζεται στο πρόγραμμα διαμόρφωσης που έχει ήδη φορτωθεί. Από την άλλη, ένα RPU πολλαπλών περιεχομένων διαμόρφωσης (*multiple-context*) έχει διάφορα προγράμματα διαμόρφωσης αποθηκευμένα στο σύστημα. Αυτό επιτρέπει την εκτέλεση διαφορετικών εργασιών, απλώς αλλάζοντας το πρόγραμμα διαμόρφωσης που εκτελείται, χωρίς να χρειάζεται αυτό να φορτωθεί στο σύστημα.

3. Επαναδιαταξιμότητα : Ένα RPU μπορεί να επαναδιατάσσεται συχνά για την εκτέλεση διαφορετικών εφαρμογών. Η επαναδιάταξη είναι η διαδικασία της επαναφόρτωσης προγραμμάτων διαμόρφωσης. Αυτή η διαδικασία είναι είτε στατική (η εκτέλεση πρέπει να διακοπεί προσωρινά) είτε δυναμική (σε παραλληλία με την εκτέλεση). Συνήθως, ένα RPU μοναδικού περιεχομένου διαμόρφωσης υποστηρίζει στατική επαναδιάταξη. Η δυναμική επαναδιάταξη ταιριάζει περισσότερο στα RPU πολλαπλού περιεχομένου διαμόρφωσης. Εδώ υποννοείται πως ένα τέτοιο RPU μπορεί να εκτελέσει ένα κομμάτι του προγράμματος διαμόρφωσής του, ενώ το άλλο υπόκειται σε αλλαγές. Αυτό το χαρακτηριστικό μειώνει σημαντικά τις χρονικές καθυστερήσεις που εισάγει η διαδικασία της επαναδιάταξη.

4. Σύστημα Διεπαφής : Εάν ο κεντρικός επεξεργαστής ενός επαναδιατάξιμου συστήματος δε βρίσκεται στο ίδιο κομμάτι πυριτίου με το RPU, τότε το σύστημα έχει μια απομακρυσμένη διεπαφή. Μια τοπική διεπαφή -αντίθετα- εφαρμόζεται όταν ο κεντρικός επεξεργαστής και ο συν-επεξεργαστής RPU βρίσκονται στο ίδιο ολοκληρωμένο, ή όταν το RPU είναι ενσωματωμένο στο datapath του κεντρικού υπολογιστή.

5. Μοντέλο υπολογισμού : Πολλά επαναδιατάξιμα συστήματα ακολουθούν το μοντέλο υπολογισμού του μοναδικού επεξεργαστή (*uniprocessor*). Ωστόσο, υπάρχουν πολλά άλλα που ακολουθούν το μοντέλο *SIMD* ή *MIMD*. Μερικά συστήματα μπορεί ακόμα να ακολουθούν το μοντέλο *VLIW* (Very Long Instruction Word).

Συμβατικά, οι πιο κοινές μονάδες που χρησιμοποιούνται στα επαναδιατάξιμα συστήματα επεξεργασίας είναι τα FPGAs. Τα FPGAs επιτρέπουν στους σχεδιαστές να τροποποιήσουν το κύκλωμα σε επίπεδο πύλης, όπως είναι τα flip-flops, η μνήμη και άλλες λογικές πύλες. Αυτό καθιστά τα FPGAs χρήσιμα σε πολύπλοκους υπολογισμούς που δουλεύουν σε επίπεδο bit. Μερικά παραδείγματα επαναδιατάξιμων συστημάτων που χρησιμοποιούν FPGAs είναι τα [14], [15], [16], [17]. Παρόλα αυτά -όπως έχει ήδη αναφερθεί αναλυτικά σε

προηγούμενες ενότητες- τα FPGAs έχουν σημαντικά μειονεκτήματα, με σημαντικότερο τις μη αποδοτικές επιδόσεις τους σε υπολογισμούς επιπέδου λέξης (8 bits ή περισσότερα).

Πολλοί ερευνητές πρότειναν άλλα μοντέλα επαναδιατάξιμων συστημάτων που στοχεύουν σε διαφορετικές εφαρμογές. Ορισμένα από τα πρώτα μοντέλα επαναδιατάξιμων συστημάτων υπολογισμού είναι τα PADDI[9], MATRIX[18], RaPiD[13] και REMARC[10]. Δύο παραδείγματα συστημάτων αρχιτεκτονικής επιπέδου bit, τα οποία όμως δε βασίζονται σε FPGAs, είναι τα DPGA[19] και Garp[20]. Ο Πίνακας 1.1 συνοψίζει τα χαρακτηριστικά διαφόρων από τα σημαντικότερα επαναδιατάξιμα συστήματα, σύμφωνα με τα κριτήρια που αναφέρθηκαν στην ενότητα αυτή.

Πίνακας 1.1: Σημαντικά επαναδιατάξιμα συστήματα, κατηγοριοποιημένα με βάση τα κριτήρια της ενότητας αυτής.

Σύστημα	Granularity	Δυνατότητα προγραμματισμού	Επαναδιάταξη	Διεπαφή	Μοντέλο υπολογισμού	Περιοχή εφαρμογών
DPGA [19]	Fine	Πολλαπλή	Δυναμική	Απομακρυσμένη	Uniprocessor	Υπολογισμοί επιπέδου bit
Garp [20]	Fine	Πολλαπλή	Στατική	Τοπική	Uniprocessor	Υπολογισμοί επιπέδου bit
Splash [14]	Fine	Πολλαπλή	Στατική	Απομακρυσμένη	Uniprocessor	Υπολογισμοί επιπέδου bit
DEC PeRLE 1 [15]	Fine	Μοναδική	Στατική	Απομακρυσμένη	Uniprocessor	Υπολογισμοί επιπέδου bit
Chimaera [16]	Fine	Μοναδική	Στατική	Τοπική	Uniprocessor	Υπολογισμοί επιπέδου bit
OneChip [21]	Fine	Μοναδική	Στατική	Τοπική	Uniprocessor	Ενσωματωμένοι ελεγκτές, επιταχυντές
DISC [17]	Fine	Μοναδική	Δυναμική	Τοπική	Uniprocessor	Γενικής σκοπού
PADDI [9]	Coarse	Πολλαπλή	Στατική	Απομακρυσμένη	VLIW, SIMD	Εφαρμογές DSP
MATRIX [18]	Coarse	Πολλαπλή	Δυναμική	Μη ορισμένο	MIMD	Μη ορισμένο
RaPiD [13]	Coarse	Μοναδική	Σχεδόν Στατική	Απομακρυσμένη	Linear array	Συστολικοί πίνακες
Remarc [10]	Coarse	Πολλαπλή	Στατική	Τοπική	SIMD	Εφαρμογές παραλληλίας δεδομένων
RAW [11]	Mixed	Μοναδική	Στατική	Τοπική	MIMD	Γενικού σκοπού
PipeRench [22]	Mixed	Πολλαπλή	Δυναμική	Απομακρυσμένη	Pipelined	Παραλληλία δεδομένων, εφαρμογές DSP
MorphoSys [23]	Coarse	Πολλαπλή	Δυναμική	Τοπική	SIMD	Παραλληλία δεδομένων, εφαρμογές εικόνας

1.7. Ροή Σχεδίασης Επαναδιατάξιμων Συστημάτων

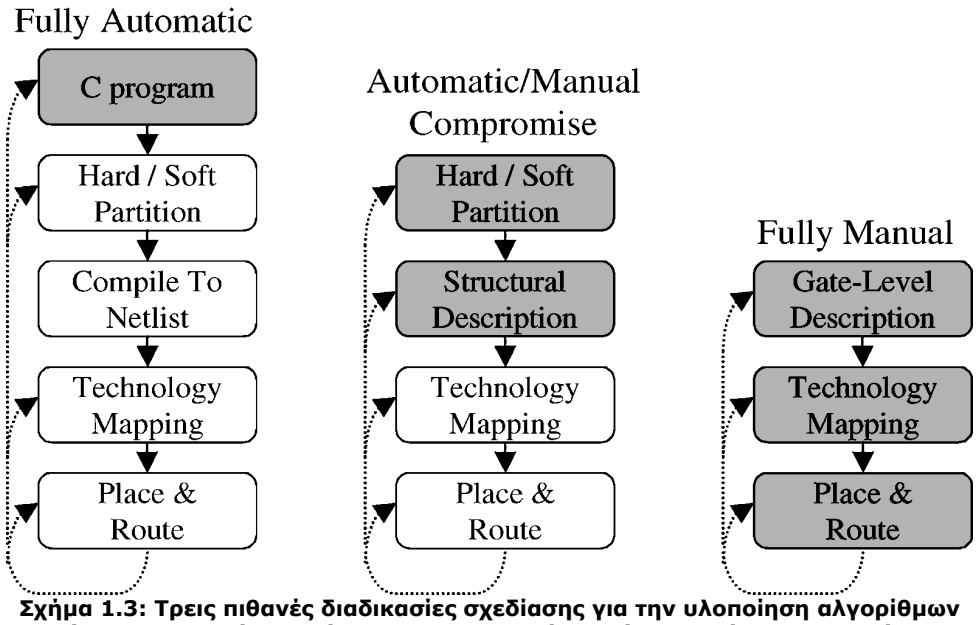
Παρόλο που το επαναδιατάξιμο υλικό έδειξε να προσφέρει σημαντικά οφέλη στην απόδοση της υλοποίησης ορισμένων εφαρμογών, μπορεί να μη χρησιμοποιηθεί στο βαθμό που θα μπορούσε από τους προγραμματιστές εφαρμογών, εκτός και αν αυτοί μπορούν να το ενσωματώσουν με ευκολία στα συστήματά τους. Αυτό απαιτεί ένα περιβάλλον σχεδίασης εφαρμογών που θα βοηθάει στη δημιουργία των περιεχομένων διαμόρφωσης για το επαναδιατάξιμο υλικό. Αυτό το λογισμικό μπορεί να κυμαίνεται από μια εφαρμογή που βοηθάει στη χειροκίνητη δημιουργία του κυκλώματος μέχρι ένα σύστημα πλήρως αυτοματοποιημένης σχεδίασης κυκλώματος. Η χειρωνακτική περιγραφή κυκλώματος είναι μια πανίσχυρη μέθοδος για τη δημιουργία σχεδιάσεων υψηλής ποιότητας. Παρόλα αυτά απαιτεί ένα μεγάλο βαθμό γνώσης του συγκεκριμένου επαναδιατάξιμου συστήματος που χρησιμοποιείται, όπως επίσης και ένα σημαντικό χρόνο σχεδίασης. Στην άλλη πλευρά, ένα αυτόματο σύστημα μεταγλώττισης παρέχει έναν εύκολο και γρήγορο τρόπο προγραμματισμού επαναδιατάξιμων συστημάτων. Για το λόγο αυτό, τέτοιες εφαρμογές κάνουν τη χρήση του επαναδιατάξιμου υλικού περισσότερο προσβάσιμη στους προγραμματιστές γενικών εφαρμογών, αλλά η ποιότητα της σχεδίασης εμφανίζεται χειρότερη.

Τόσο για τη χειρωνακτική όσο και για την αυτόματη δημιουργία κυκλώματος, η διαδικασία σχεδίασης ακολουθεί έναν αριθμό διακριτών φάσεων, όπως φαίνεται και στο Σχήμα 1.3. Προσδιορισμός του κυκλώματος είναι η διαδικασία περιγραφής των συναρτήσεων που πρέπει να απεικονιστούν στο επαναδιατάξιμο υλικό. Αυτό μπορεί να είναι πολύ απλό, όπως με τη συγγραφή ενός προγράμματος σε C που αναπαριστά τη λειτουργικότητα του αλγορίθμου που θα υλοποιηθεί στο υλικό. Αλλά μπορεί αντίθετα να είναι πολύπλοκο, όπως ο καθορισμός των εισόδων, εξόδων και της λειτουργίας καθεμιάς από τις βασικές δομικές μονάδες ενός επαναδιατάξιμου συστήματος. Υπάρχει φυσικά και η ενδιάμεση περίπτωση, όπου ο καθορισμός του κυκλώματος γίνεται χρησιμοποιώντας γενικευμένες πολύπλοκες μονάδες, όπως αθροιστές ή πολλαπλασιαστές. Για περιγραφές με γλώσσες υψηλού επιπέδου (high-level language, HLL), ο κώδικας που θα δημιουργηθεί πρέπει να μεταγλωττιστεί σε μια λίστα από μονάδες επιπέδου πύλης. Από τη στιγμή που υπάρχει μια δομική περιγραφή, είτε δημιουργημένη από μια HLL είτε καθορισμένη από το χρήστη, τότε κάθε πολύπλοκη δομή της περιγραφής αντικαθίσταται από ένα δίκτυο βασικών δομικών μονάδων που υλοποιούν την επιθυμητή συνάρτηση.

Από τη στιγμή που δημιουργηθεί μια λεπτομερής περιγραφή του κυκλώματος (σε επίπεδο πύλης ή μονάδων), τότε οι δομές της περιγραφής πρέπει να μεταφραστούν στις πραγματικές λογικές μονάδες του επαναδιατάξιμου υλικού. Αυτό το στάδιο είναι γνωστό ως απεικόνιση στην τεχνολογία (technology mapping) και εξαρτάται άμεσα από την ακριβή αρχιτεκτονική που θα

χρησιμοποιηθεί. Αφού πραγματοποιηθεί και το στάδιο της απεικόνισης, τα τελικά block που προκύπτουν πρέπει να τοποθετηθούν πάνω στο επαναδιατάξιμο υλικό. Κάθε ένα από τα block αυτά ανατίθεται σε μια συγκεκριμένη τοποθεσία πάνω στο υλικό.

Στο τελικό στάδιο των διασυνδέσεων πρέπει να συνδεθούν οι διαφορετικές επαναδιατάξιμες μονάδες πάνω στο υλικό που αποτελούν το κύκλωμα απεικόνισης της εφαρμογής. Η ανάθεση σημάτων σε συγκεκριμένα κομμάτια κυκλώματος διασύνδεσης μπορεί να γίνει δύσκολο, αν η τοποθέτηση πάνω στο υλικό έχει ως αποτέλεσμα να τοποθετηθούν σε μακρινή απόσταση η μία από την άλλη μονάδες που πρόκειται να συνδεθούν μεταξύ τους, αφού σήματα που ταξιδεύουν σε μεγάλες αποστάσεις στο κύκλωμα χρησιμοποιούν περισσότερο κύκλωμα διασύνδεσης από σήματα που ταξιδεύουν σε μικρές αποστάσεις. Για το λόγο αυτό, στη διαδικασία της διασύνδεσης είναι απαραίτητο να υπάρχει μια σωστή τοποθέτηση των μονάδων πάνω στο υλικό.



Σχήμα 1.3: Τρεις πιθανές διαδικασίες σχεδίασης για την υλοποίηση αλγορίθμων σε ένα επαναδιατάξιμο σύστημα. Τα σκιασμένα στάδια δηλώνουν χειροκίνητη προσπάθεια από την πλευρά του σχεδιαστή, ενώ τα μη σκιασμένα στάδια γίνονται αυτόματα. Οι γραμμές μεταξύ των σταδίων απεικονίζουν δρόμους για τη βελτίωση του τελικού κυκλώματος. Θα πρέπει να τονιστεί ότι η μεσαία διαδικασία σχεδίασης είναι μόνο μία από τις πιθανές περιπτώσεις όπου υπάρχει συμβιβασμός μεταξύ αυτόματης και χειροκίνητης σχεδίασης. Να σημειωθεί ότι τα τελευταία στάδια (Compile to netlist, technology mapping, place&route) αναφέρονται σε επαναδιατάξιμα συστήματα με FPGAs.

Συνοψίζοντας, τα επαναδιατάξιμα συστήματα απαιτούν λογισμικό μεταγλώττισης ώστε να είναι εφικτό οι προγραμματιστές να εκμεταλλευτούν πλήρως τα οφέλη του επαναδιατάξιμου υπολογισμού. Από τη μια μεριά του φάσματος, κυκλώματα για επαναδιατάξιμα συστήματα μπορούν να σχεδιαστούν και χειροκίνητα, κάνοντας χρήση όλων των βελτιστοποιήσεων εφαρμογής και αρχιτεκτονικής που είναι διαθέσιμες ώστε να παραχθεί τελικά μια υψηλής απόδοσης εφαρμογή. Ωστόσο, αυτό απαιτεί μια μεγάλη ποσότητα χρόνου και προσπάθειας από την

πλευρά του σχεδιαστή. Από την άλλη πλευρά όμως υπάρχει η επιλογή της πλήρους αυτοματοποιημένης μεταγλώττισης μέσω γλωσσών υψηλού επιπέδου. Χρησιμοποιώντας αυτοματοποιημένα εργαλεία, ένας προγραμματιστής εφαρμογών μπορεί να κάνει διάφανη χρήση του επαναδιατάξιμου υλικού, χωρίς δηλαδή να είναι ανάγκη να επέμβει άμεσα πάνω στο υλικό και χωρίς να είναι απαραίτητες επιπλέον γνώσεις για αυτό. Τα κυκλώματα που δημιουργούνται κάνοντας χρήση της μεθόδου αυτής, παρότι δημιουργούνται γρηγορότερα και ευκολότερα, καταλαμβάνουν γενικά περισσότερο χώρο και είναι πιο αργά από αυτά που δημιουργούνται χειροκίνητα. Τα πραγματικά εργαλεία που είναι διαθέσιμα για τον προγραμματισμό επαναδιατάξιμων συστημάτων, είναι μερικώς αυτοματοποιημένα και απαιτούν κάποιο βαθμό χειροκίνητης βοήθειας. Οι σχεδιαστές κυκλωμάτων για επαναδιατάξιμα συστήματα αντιμετωπίζουν δηλαδή το πρόβλημα της επίτευξης συμβιβασμού μεταξύ της ευκολίας της σχεδίασης και της ποιότητας του τελικού αποτελέσματος.

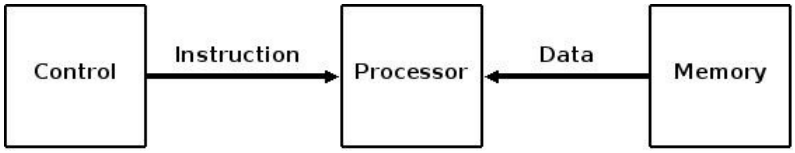
ΚΕΦΑΛΑΙΟ 2
Μοντέλα Επεξεργασίας

Οποιοσδήποτε υπολογιστής, είτε σειριακός είτε παράλληλος, λειτουργεί εκτελώντας εντολές πάνω στα δεδομένα. Μια σειρά από εντολές (ο αλγόριθμος) υποδεικνύει στον υπολογιστή τι πρέπει να εκτελεστεί σε κάθε βήμα. Μια σειρά από δεδομένα, από την άλλη (οι είσοδοι στον αλγόριθμο), είναι αυτά που επηρεάζονται από αυτές τις εντολές. Μια ευρέως χρησιμοποιούμενη κατηγοριοποίηση των παράλληλων συστημάτων, που οφείλεται στον Michael J. Flynn[24], βασίζεται στον αριθμό των ταυτόχρονων εντολών και σειρών δεδομένων που "βλέπει" ο επεξεργαστής κατά τη διάρκεια της εκτέλεσης του προγράμματος. Ανάλογα με το αν υπάρχει μία ή πολλές από αυτές τις σειρές εντολών/δεδομένων, οι υπολογιστές έχουν χωριστεί σε τέσσερις κατηγορίες:

- Μοναδικής σειράς εντολών/μοναδικής σειράς δεδομένων (Single Instruction stream/Single Data stream, SISD)
- Πολλαπλών σειράς εντολών/μοναδικής σειράς δεδομένων (Multiple Instruction stream/Single Data stream, MISD)
- Μοναδικής σειράς εντολών/πολλαπλής σειράς δεδομένων (Single Instruction stream/Multiple Data stream, SIMD)
- Πολλαπλής σειράς εντολών/μοναδικής σειράς δεδομένων(Multiple Instruction stream/Multiple Data stream, MIMD)

2.1. Υπολογιστές SISD

Ένας SISD υπολογιστής αποτελείται από μία μόνο μονάδα επεξεργασίας που λαμβάνει μια μοναδική σειρά εντολών, η οποία εργάζεται σε μια μοναδική σειρά δεδομένων (Σχήμα 2.1). Σε κάθε βήμα, η μονάδα ελέγχου βγάζει μια εντολή που επεξεργάζεται μια θέση (δεδομένα) από τη μονάδα μνήμης. Σχεδόν όλοι οι υπολογιστές που χρησιμοποιούνται σήμερα έχουν αφομοιώσει αυτό το μοντέλο, το οποίο επινοήθηκε από τον John von Neumann και ονομάστηκε προς τιμήν του (μοντέλο υπολογισμού von Neumann). Ένας αλγόριθμος που εκτελείται σε έναν υπολογιστή SISD ονομάζεται σειριακός, αφού δεν περιλαμβάνει κάποια έννοια παραλληλίας.

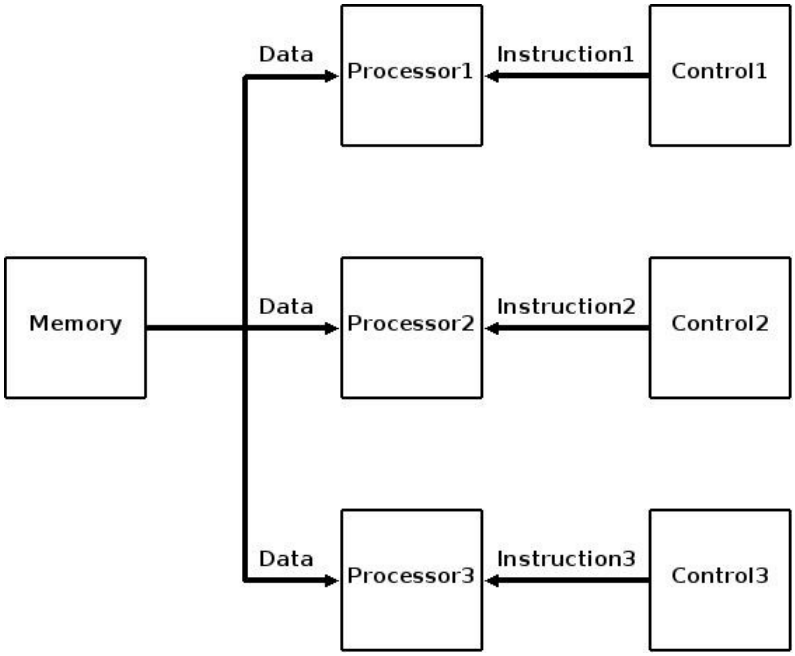


Σχήμα 2.1: Μοντέλο επεξεργασίας SISD

2.2. Υπολογιστές MISD

Στους υπολογιστές που ακολουθούν το μοντέλο MISD υπάρχουν συνολικά N σε αριθμό επεξεργαστές (Σχήμα 2.4), κάθε ένας με τη δική του μονάδα ελέγχου, οι οποίοι μοιράζονται μια κοινή μονάδα μνήμης. Σε κάθε βήμα, κάποιο δεδομένο που προέρχεται από τη μνήμη χρησιμοποιείται από όλους τους επεξεργαστές ταυτόχρονα, ο καθένας με βάση τις εντολές που λαμβάνει από τη δική του μονάδα μνήμης. Ο παραλληλισμός επιτυγχάνεται με το να επιτρέπεται στους επεξεργαστές να εκτελούν διαφορετικές εργασίες πάνω στα ίδια δεδομένα.

Αυτή η κατηγορία υπολογιστών ενδείκνυται για εκείνους τους υπολογισμούς που απαιτούν η μια και μοναδική είσοδος να χρησιμοποιείται σε πολλές διαφορετικές λειτουργίες, η κάθε μία από τις οποίες λαμβάνει την είσοδο στην αρχική της μορφή. Ωστόσο, οι υπολογισμοί που μπορούν να εκτελεστούν αποδοτικά σε υπολογιστές MISD είναι πολύ εξειδικευμένες.

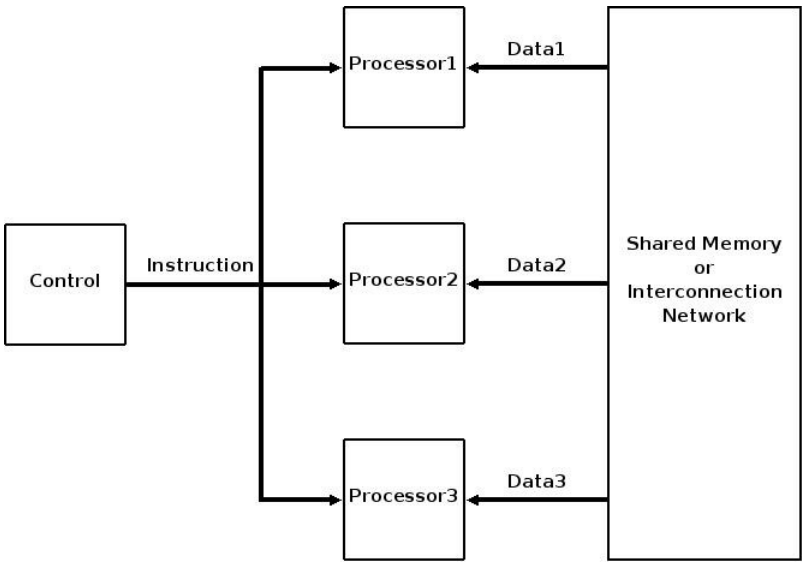


Σχήμα 2.2: Μοντέλο επεξεργασίας MISD

2.3.Υπολογιστές SIMD

Ένας SIMD υπολογιστής αποτελείται από N στον αριθμό πανομοιότυπους επεξεργαστές (Σχήμα 2.3), καθένας από τους οποίους έχει τη δική του τοπική μνήμη όπου μπορεί να αποθηκεύει δεδομένα. Όλοι οι επεξεργαστές εργάζονται υπό τον έλεγχο μια μοναδικής σειράς εντολών, οι οποίες καθορίζονται από την κεντρική μονάδα ελέγχου. Αντιθέτως, υπάρχουν N σειρές δεδομένων, μία για κάθε επεξεργαστή. Οι επεξεργαστές μπορούν να εργάζονται ταυτόχρονα: σε κάθε βήμα, όλοι οι επεξεργαστές εκτελούν την ίδια εντολή αλλά χρησιμοποιώντας διαφορετικά δεδομένα.

Οι SIMD υπολογιστές είναι πιο ευέλικτοι στη χρήση τους από τους υπολογιστές της προηγούμενης ομάδας. Με τη χρήση κατάλληλων παράλληλων αλγορίθμων σε υπολογιστές SIMD μπορεί να υλοποιηθεί μια μεγάλη ποικιλία εφαρμογών. Άλλο χαρακτηριστικό είναι ότι οι αλγόριθμοι για αυτούς τους υπολογιστές είναι σχετικά εύκολοι στη σχεδίαση, την ανάλυση και την υλοποίηση. Από την άλλη, σε υπολογιστές SIMD μπορούν να υλοποιηθούν μόνο εκείνα τα προβλήματα που μπορούν να υποδιαιρεθούν σε ένα σύνολο από πανομοιότυπα υπο-προβλήματα, τα οποία στη συνέχεια επιλύονται ταυτόχρονα από την ίδια ομάδα υπολογισμών. Αυτό δεν είναι δυνατόν να γίνει με όλους τους αλγορίθμους και υπάρχουν πολλοί υπολογισμοί που δεν ταιριάζουν σε αυτό το μοντέλο. Συνήθως όμως τέτοια προβλήματα υποδιαιρούνται σε υπο-προβλήματα που δεν είναι απαραίτητα πανομοιότυπα και τα οποία λύνονται χρησιμοποιώντας υπολογιστές της επόμενης κατηγορίας (MIMD).



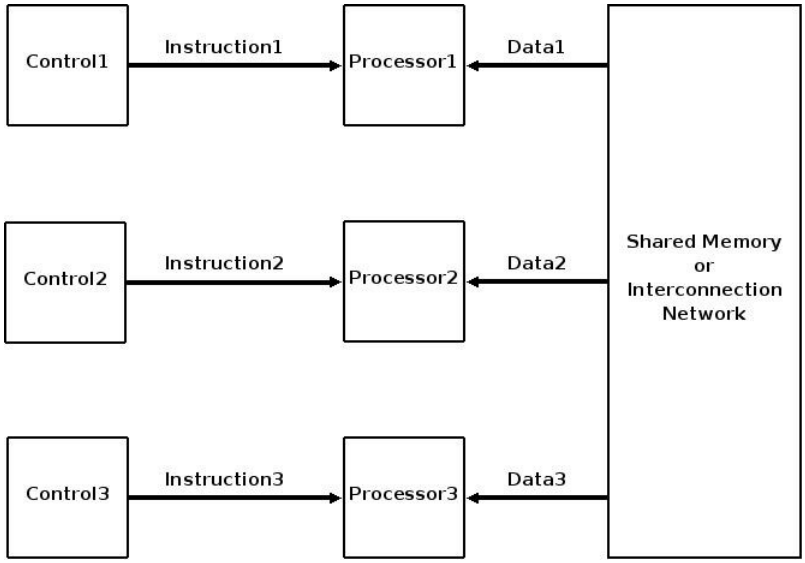
Σχήμα 2.3: Μοντέλο επεξεργασίας SIMD

2.4. Υπολογιστές MIMD

Αυτή η ομάδα παράλληλων υπολογιστών είναι η πιο γενική και πιο ισχυρή -υπολογιστικά- στην κατηγοριοποίηση κατά Flynn. Εδώ υπάρχουν N επεξεργαστές, N σειρές εντολών και N σειρές δεδομένων (Σχήμα 2.4). Κάθε επεξεργαστής έχει, δηλαδή, τη δική του μονάδα ελέγχου και τη δική του τοπική μνήμη, το οποίο καθιστά τους επεξεργαστές πιο ισχυρούς από αυτούς που χρησιμοποιούνται στους SIMD υπολογιστές. Κάθε επεξεργαστής λειτουργεί κάτω από τον έλεγχο μιας σειράς εντολών που υπάρχουν στη δική του μονάδα ελέγχου. Για το λόγο αυτό, οι επεξεργαστές -θεωρητικά- εκτελούν όλοι διαφορετικά προγράμματα σε διαφορετικά δεδομένα, επιλύοντας διαφορετικά υπο-προβλήματα ενός μοναδικού προβλήματος. Αυτό σημαίνει ότι οι επεξεργαστές δουλεύουν ασύγχρονα.

Το μοντέλο MIMD παράλληλων υπολογισμών είναι, όπως αναφέρθηκε, το πιο

γενικό και ισχυρό και οι υπολογιστές σε αυτή την κατηγορία χρησιμοποιούνται για την επίλυση -σε παραλληλία- εκείνων των προβλημάτων που στερούνται μιας επαναλαμβανόμενης δομής, πράγμα που απαιτείται από το μοντέλο SIMD. Ωστόσο, οι ασύγχρονοι αλγόριθμοι είναι αρκετά δύσκολο να σχεδιαστούν, αναλυθούν και υλοποιηθούν.



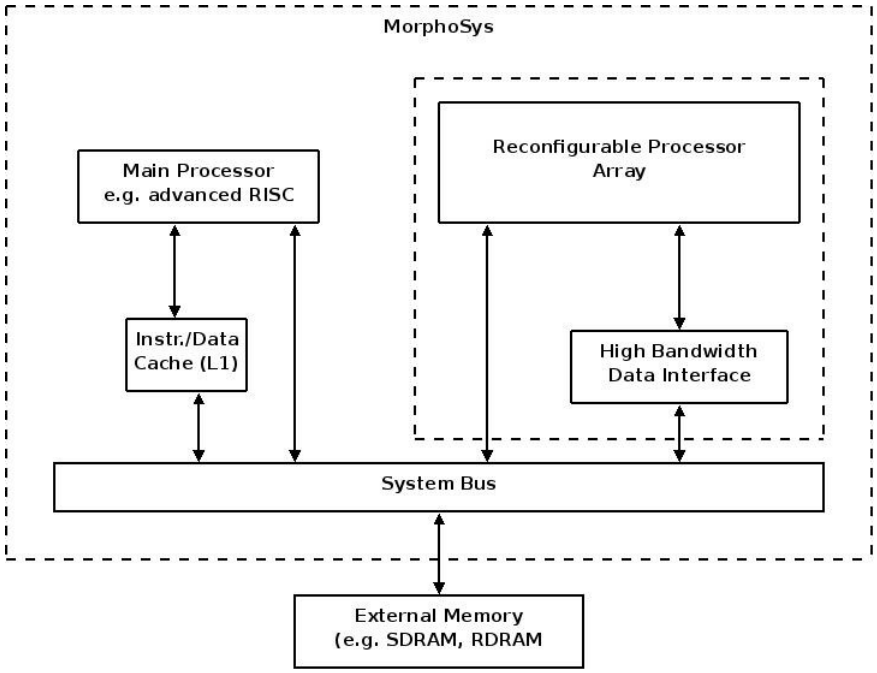
Σχήμα 2.4: Μοντέλο επεξεργασίας MIMD

ΚΕΦΑΛΑΙΟ 3
Το Επαναδιατάξιμο Σύστημα MorphoSys

3.1. Μοντέλο Συστήματος του MorphoSys

Το MorphoSys([25], [7], [23]) είναι ένα επαναδιατάξιμο υπολογιστικό σύστημα που αναπτύχθηκε από μια ομάδα ερευνητών με σκοπό τη μελέτη της αποδοτικότητας της συνένωσης επαναδιατάξιμου υλικού με γενικού σκοπού επεξεργαστές. Συγκεκριμένα, πρόκειται για ένα ενσωματωμένο και επαναδιατάξιμο SoC αρχιτεκτονικής επιπέδου λέξης, με τον κάθε χειριστή δεδομένων του να έχει μέγεθος 16-bit. Το MorphoSys έχει ως στόχο την αποδοτικότερη υλοποίηση εφαρμογών με απαιτήσεις για υψηλό ρυθμό απόδοσης και μεγάλη παραλληλία, όπως για παράδειγμα ο διακριτός συνημιτονικός μετασχηματισμός, η επεξεργασία εικόνας και γραφικών, η κρυπτογράφηση δεδομένων και οι μετασχηματισμοί DSP.

Στο Σχήμα 3.1 φαίνεται ένα γενικό διάγραμμα των βασικών στοιχείων της αρχιτεκτονικής MorphoSys.

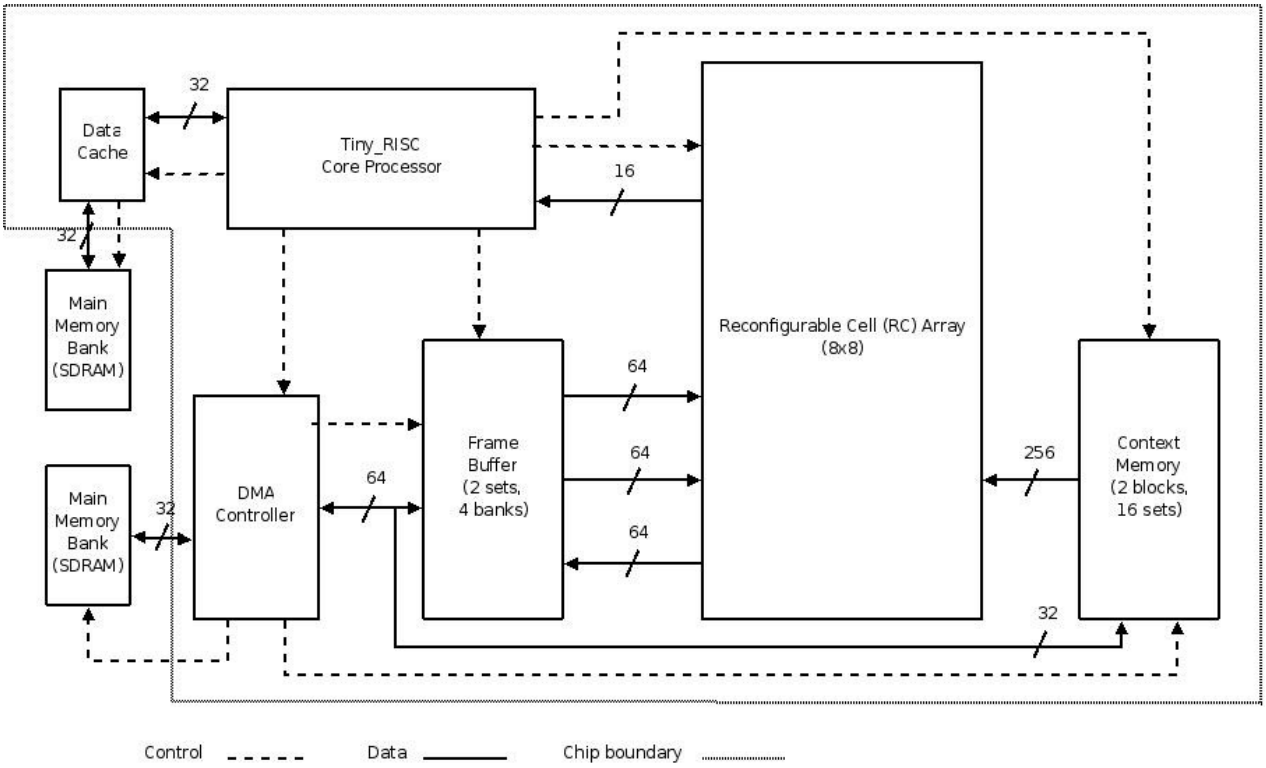


Σχήμα 3.1: Γενικό διάγραμμα της αρχιτεκτονικής MorphoSys

Η αρχιτεκτονική του MorphoSys αποτελείται, όπως φαίνεται και από το Σχήμα 3.1, από μια επαναδιατάξιμη μονάδα επεξεργασίας, έναν πυρήνα γενικού σκοπού επεξεργαστή ελέγχου και μια διεπαφή μνήμης υψηλού εύρους ζώνης, όλα ενσωματωμένα σε ένα ολοκληρωμένο. Ο πυρήνας του επαναδιατάξιμου κομματιού έχει οργανωθεί σε έναν 8x8 πίνακα από επαναδιατάξιμα κελιά (*Reconfigurable Cells – RCs*) που λειτουργεί σύμφωνα με τη λογική του SIMD

μοντέλου υπολογισμού, χρησιμοποιώντας πολλαπλές λέξεις διαμόρφωσης. Η αρχιτεκτονική και η λειτουργία των κελιών αυτών παρουσιάζουν βαθμό επαναδιάταξης επιπέδου λέξης. Ο επεξεργαστής του συστήματος, που είναι αρχιτεκτονικής RISC, ελέγχει όλες τις λειτουργίες του επαναδιατάξιμου πίνακα (RC Array), αποτελεί δηλαδή τη μονάδα ελέγχου του συστήματος. Η υψηλού εύρους ζώνης διεπαφή μνήμης αποτελείται από έναν εξειδικευμένο ενταμιευτή (buffer) και ελεγκτή ώστε να εξυπηρετούνται οι μεταφορές δεδομένων μεταξύ της εξωτερικής μνήμης και του πίνακα επαναδιατάξιμων κελιών.

Στο Σχήμα 3.2 φαίνεται μια πιο αναλυτική περιγραφή της αρχιτεκτονικής, όπου τα γενικά μπλοκ από το Σχήμα 3.1 έχουν αντικατασταθεί από συγκεκριμένες κυκλωματικές μονάδες.



Σχήμα 3.2: Συστατικά στοιχεία της υλοποίησης του MorphoSys (M1 chip)

Από το Σχήμα 3.2 φαίνονται αναλυτικά τα 5 κύρια στοιχεία από τα οποία αποτελείται η αρχιτεκτονική MorphoSys. Αυτά είναι ο πίνακας επαναδιατάξιμων κελιών (RC Array) με μια μνήμη δεδομένων διαμόρφωσης (Context Memory), ένας επεξεργαστής ελέγχου (TinyRISC), ενταμιευτής δεδομένων (Frame Buffer) και ένας ελεγκτής DMA. Συγκριτικά, ο πίνακας RC με την μνήμη διαμόρφωσης αποτελούν τον επαναδιατάξιμο πίνακα στο Σχήμα 3.2, ο TinyRISC αντιστοιχεί στον κύριο επεξεργαστή και ο Frame Buffer και ο ελεγκτής DMA υλοποιούν τη διεπαφή μνήμης υψηλού εύρους ζώνης στο Σχήμα 3.1. Ακολούθως περιγράφονται το μοντέλο του συστήματος και οι λεπτομέρειες της αρχιτεκτονικής της πρώτης υλοποίησης του MorphoSys.

3.1.1. Επισκόπηση του Συστήματος

3.1.1.1. Πίνακας Επαναδιατάξιμων Κελιών (RC Array)

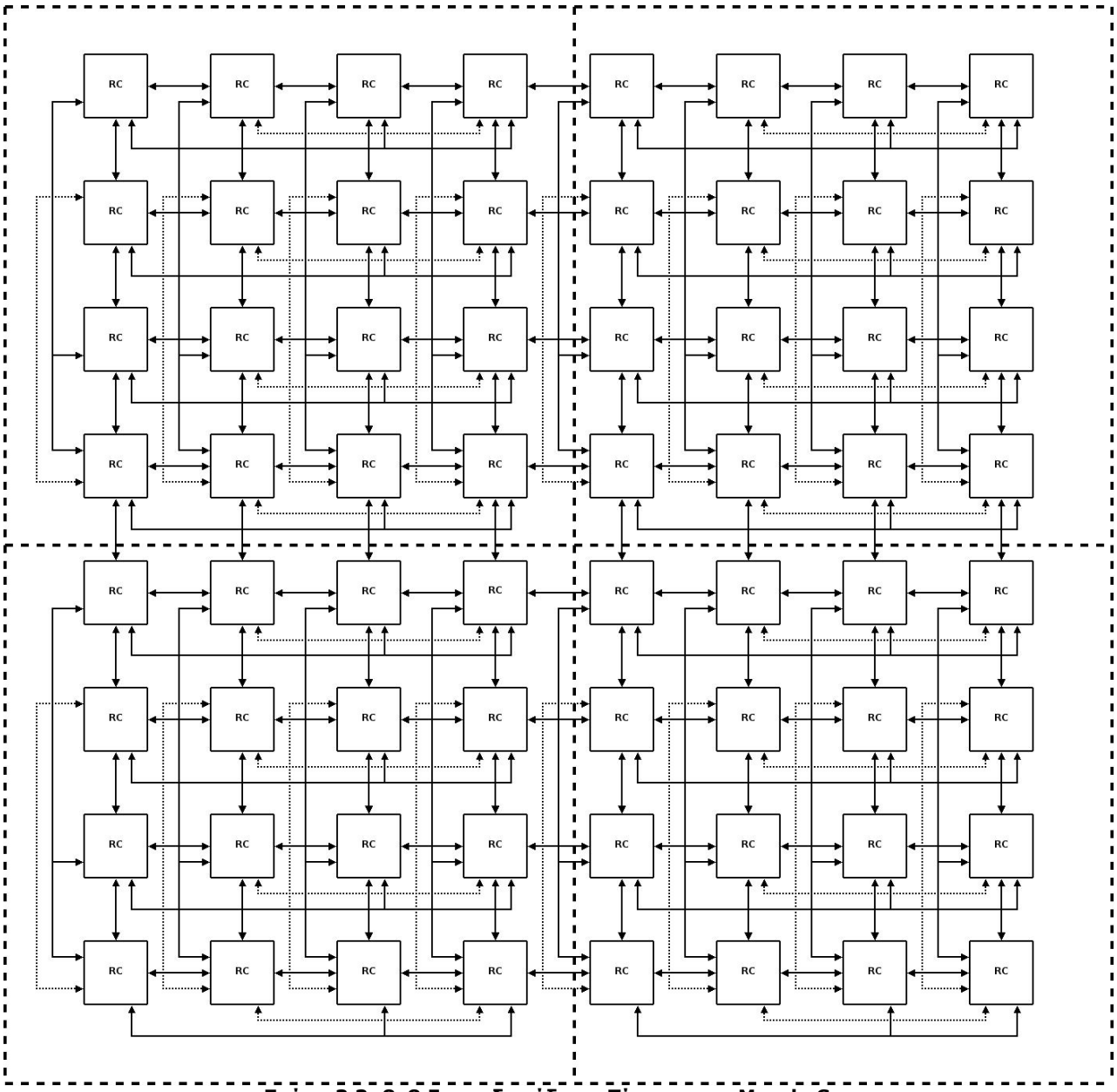
Η κύρια μονάδα του MorphoSys είναι ο Re-configurable Cell (RC) Array (Σχήμα 3.3), ο οποίος αποτελείται από 64 επαναδιατάξιμα κελιά, οργανωμένα σε ένα πίνακα διαστάσεων 8 x 8. Κάθε επαναδιατάξιμο κελί, με τη σειρά του, αποτελείται από μια ALU που υποστηρίζει υπολογισμούς σταθερής υποδιαστολής, έναν πολλαπλασιαστή και ένα αρχείο καταχωρητών (με χειριστή δεδομένων 16-bit). Η λειτουργικότητα και το δίκτυο διασυνδέσεων μεταξύ των κελιών διαμορφώνονται από λέξεις διαμόρφωσης μήκους 32bit. Οι λέξεις που προορίζονται για διαμόρφωση των κελιών αποθηκεύονται στη μνήμη διαμόρφωσης σε δύο μπλοκ, ένα για τις γραμμές και ένα για τις στήλες, με κάθε μπλοκ να αποτελείται από 8 σύνολα των 16 θέσεων μνήμης. Επομένως, δεδομένα διαμόρφωσης από τη μνήμη διαμόρφωσης εκπέμπονται στον επαναδιατάξιμο πίνακα είτε ανά στήλη είτε ανά γραμμή. Όταν έχουμε εκπομπή ανά γραμμή (στήλη), τότε και τα 8 κελιά της ίδιας γραμμής (στήλης) διαμορφώνονται με βάση την ίδια λέξη.

3.1.1.2. Επεξεργαστής Ελέγχου

Η μονάδα ελέγχου του MorphoSys είναι ένας επεξεργαστής 32bits, που ονομάζεται TinyRISC και βασίζεται στην αρχιτεκτονική RISC. Ο επεξεργαστής αυτός χειρίζεται τις γενικού σκοπού λειτουργίες και ελέγχει τη γενική λειτουργία του επαναδιατάξιμου πίνακα (χωρίς να επεμβαίνει ωστόσο εσωτερικά, όπου τον έλεγχο αναλαμβάνει η μνήμη διαμόρφωσης) και τις μεταφορές δεδομένων από και προς τον πίνακα. Επιπλέον αρχικοποιεί τις μεταφορές δεδομένων από ή προς το Frame Buffer καθώς και τη φόρτωση του προγράμματος διαμόρφωσης στη μνήμη διαμόρφωσης. Για να υλοποιηθούν οι επιπλέον αυτές λειτουργίες, έχουν προστεθεί ορισμένες ειδικές εντολές στο σύνολο εντολών του Tiny RISC.

3.1.1.3. Frame Buffer

Ο Frame Buffer (FB) είναι ένα σημαντικό στοιχείο του MorphoSys, καθώς είναι αντίστοιχο με μια προσωρινή μνήμη (cache) δεδομένων. Αποτελείται από δύο σύνολα, κάθε ένα από τα οποία έχει δύο ομάδες (bank) μνήμης. Καθιστά τις προσπελάσεις στη μνήμη διάφανες στον επαναδιατάξιμο πίνακα, καθώς επικαλύπτει τη φόρτωση και την αποθήκευση δεδομένων, χρησιμοποιώντας ανάλογα τα δύο σύνολα. Η απόδοση του MorphoSys βελτιώνεται σημαντικά από αυτή την επικάλυψη.



Σχήμα 3.3: 8x8 Επαναδιατάξιμος Πίνακας του MorphoSys

3.1.2. Ροή Προγράμματος

Το MorphoSys λειτουργεί ως ακολούθως: Ο επεξεργαστής Tiny RISC φορτώνει αρχικά τα δεδομένα διαμόρφωσης από την κεντρική μνήμη του συστήματος στη μνήμη διαμόρφωσης μέσω του ελεγκτή DMA. Έπειτα, ενεργοποιεί τη μεταφορά των δεδομένων προς επεξεργασία από την κεντρική μνήμη στη μονάδα Frame Buffer. Η μεταφορά αυτή πραγματοποιείται επίσης μέσω της μονάδας DMA και μετά την ολοκλήρωσή της, τόσο τα δεδομένα διαμόρφωσης όσο και τα δεδομένα προς επεξεργασία έχουν φορτωθεί στις μνήμες του επαναδιατάξιμου πίνακα. Έπειτα, ο επεξεργαστής μεταφέρει ορισμένες οδηγίες στον πίνακα, οι οποίες αφορούν στην εκτέλεση των περιεχομένων διαμόρφωσης. Αυτές οι οδηγίες

καθορίζουν ποιο περιεχόμενο διαμόρφωσης (μεταξύ των πολλαπλών διαμορφώσεων της μνήμης διαμόρφωσης) πρέπει να εκτελεστεί. Επιπλέον, ο Tiny RISC επιτρέπει -κατά περίπτωση- την επιλεκτική λειτουργία μιας γραμμής/στήλης και μπορεί να προσπελάσει δεδομένα από επιλεγμένες εξόδους κελιών.

3.1.3. Χαρακτηριστικά του MorphoSys

Ο επαναδιατάξιμος πίνακας διαμορφώνεται μέσω των λέξεων διαμόρφωσης, οι οποίες μεταφέρονται από τη μνήμη διαμόρφωσης. Κάθε λέξη διαμόρφωσης προσδιορίζει τον κωδικό μιας εντολής για τα επαναδιατάξιμα κελιά. Επειδή ο επαναδιατάξιμος πίνακας ακολουθεί το μοντέλο SIMD στους υπολογισμούς, όλα τα κελιά στην ίδια γραμμή ή στήλη μοιράζονται την ίδια λέξη διαμόρφωσης, αλλά παρόλα αυτά κάθε κελί λειτουργεί με διαφορετικά δεδομένα. Το χαρακτηριστικό αυτό, ότι οι λέξεις διαμόρφωσης μοιράζονται ανά στήλη ή ανά γραμμή, είναι πολύ χρήσιμο για εφαρμογές που περιλαμβάνουν μεγάλο αριθμό λειτουργιών με παραλληλία δεδομένων, δηλαδή τις εφαρμογές τις οποίες στοχεύει να απεικονίσει με αποδοτικότητα το MorphoSys.

Συνοπτικά, τα σημαντικά χαρακτηριστικά του MorphoSys είναι :

- *Βαθμός επαναδιάταξης επιπέδου λέξης* : Το MorphoSys έχει σχεδιαστεί για να λειτουργεί με δεδομένα των 16 bits, το οποίο εξασφαλίζει ταχύτερη απόδοση για λειτουργίες επιπέδου λέξης, σε σύγκριση με τα FPGA. Δε συναντώνται, επίσης, μεταβλητές καθυστερήσεις διάδοσης στα καλώδια, χαρακτηριστικό των FPGA. Εξάλλου, κάθε κελί του επαναδιατάξιμου πίνακα διαμορφώνεται από τη λέξη διαμόρφωσης. Η λέξη αυτή καθορίζει μία από τους πολλαπλούς κωδικούς εντολών για τον επαναδιατάξιμο πίνακα και παρέχει bits ελέγχου για τους πολυπλέκτες εισόδου. Ακόμα, καθορίζει κάποια δεδομένα- "σταθερές" (constants) που -πιθανώς- χρειάζονται στους υπολογισμούς.

- *Δυνατότητα δυναμικής επαναδιάταξης* : Επιτυγχάνεται με τη δυνατότητα που υπάρχει να φορτωθούν -με νέα δεδομένα διαμόρφωσης- ορισμένα κομμάτια της μνήμης διαμόρφωσης που είναι μη-ενεργά, ενώ ταυτόχρονα ο επαναδιατάξιμος πίνακας εκτελεί λέξεις διαμόρφωσης από ένα άλλο κομμάτι της μνήμης, χωρίς να διακόπτονται οι τρέχουσες λειτουργίες του πίνακα. Για παράδειγμα, ενώ ο επαναδιατάξιμος πίνακας λειτουργεί εκτελώντας τις 16 λέξεις διαμόρφωσης σε κατάσταση λειτουργίας ανά γραμμές, οι υπόλοιπες 16 που αφορούν τη λειτουργία ανά στήλες μπορούν να φορτωθούν με νέα δεδομένα. Η φόρτωση των δεδομένων διαμόρφωσης στη μνήμη καθορίζεται από εντολές του TinyRISC και υλοποιείται από τον ελεγκτή DMA.

- *Μεγάλο βάθος δυνατότητας προγραμματισμού* : Η μνήμη διαμόρφωσης

μπορεί να αποθηκεύσει μέχρι 32 ενότητες διαμόρφωσης (16 που αναφέρονται σε μια συγκεκριμένη γραμμή και 16 που αναφέρονται σε μια συγκεκριμένη στήλη) και παρέχει δύο μεθόδους μετάδοσης, μία ανά στήλες και μία ανά γραμμές.

- *Στενά συνδυασμένη διεπαφή κεντρικού επεξεργαστή και κύριας μνήμης* : Ο επεξεργαστής TinyRISC και ο επαναδιατάξιμος πίνακας βρίσκονται στο ίδιο κομμάτι πυριτίου, κάτι το οποίο αποτρέπει διάφορους περιορισμούς εισόδου/εξόδου να επηρεάσουν την απόδοση του συστήματος. Επιπροσθέτως, ο ελεγκτής DMA που υπάρχει πάνω στο ολοκληρωμένο επιτρέπει τις γρήγορες μεταφορές δεδομένων μεταξύ της κύριας (εξωτερικής) μνήμης και του Frame Buffer. Ακόμη, η μονάδα DMA επιταχύνει το χρόνο φόρτωσης των δεδομένων διαμόρφωσης.

3.1.4. Εντολές TinyRISC για το MorphoSys

Έχουν εισαχθεί αρκετές νέες εντολές (Πίνακας 3.1) στο σύνολο εντολών του TinyRISC για τον αποτελεσματικό έλεγχο των λειτουργιών του επαναδιατάξιμου πίνακα του MorphoSys. Αυτές οι εντολές ενεργοποιούν της μεταφορά δεδομένων μεταξύ της κεντρικής μνήμης του συστήματος (SDRAM) και της μονάδας Frame Buffer, τη φόρτωση δεδομένων διαμόρφωσης από την κεντρική μνήμη στη μνήμη διαμόρφωσης και τον έλεγχο των λειτουργιών του πίνακα.

Πίνακας 3.1: Νέες εντολές του TinyRISC

CBCAST	Εκτέλεση συγκεκριμένης λέξης διαμόρφωσης στον πίνακα
DBCB	Εκτέλεση περιεχομένου διαμόρφωσης στον πίνακα, ανάγνωση δύο δεδομένων από το Frame Buffer στον RC Array
LDCTXT	Φόρτωση περιεχομένου διαμόρφωσης στη Context Memory
LDFB	Φόρτωση δεδομένων στο Frame Buffer από την κεντρική μνήμη
RCRISC	Εγγραφή δεδομένων από το RC Array στο Tiny RISC
SBCB	Εκτέλεση περιεχομένου διαμόρφωσης στον πίνακα, ανάγνωση ενός δεδομένου στο RC Array από το Frame Buffer
STFB	Αποθήκευση δεδομένων από το Frame Buffer στην κεντρική μνήμη
WFB	Εγγραφή δεδομένων από μια συγκεκριμένη στήλη του πίνακα στο Frame Buffer

Οι εντολές αυτές είναι χωρισμένες σε δύο κατηγορίες : εντολές DMA και εντολές RC. Τα πεδία των εντολών DMA καθορίζουν τη φόρτωση/αποθήκευση, την (έμμεση) διεύθυνση μνήμης, τον αριθμό των bytes/λέξεων διαμόρφωσης που θα μεταφερθούν και τη διεύθυνση στο Frame Buffer ή στη μνήμη διαμόρφωσης (ανάλογα με την περίπτωση). Τα πεδία των εντολών RC καθορίζουν τη διεύθυνση στη μνήμη διαμόρφωσης του περιεχομένου διαμόρφωσης που θα εκτελεστεί, τη διεύθυνση του Frame Buffer (αν ο επαναδιατάξιμος πίνακας χρειάζεται δεδομένα εγγραφής/ανάγνωσης) και τον τρόπο λειτουργίας (ανά στήλη ή ανά γραμμή).

3.2. Αρχιτεκτονική του Επαναδιατάξιμου Πίνακα

Στην ενότητα αυτή περιγράφονται τρία κύρια χαρακτηριστικά του MorphoSys. Αρχικά, δίνονται οι λεπτομέρειες της αρχιτεκτονικής καθενός από τα επαναδιατάξιμα κελιά (Σχήμα 3.4) με περιγραφή των διαφορετικών μονάδων λειτουργίας, αποθήκευσης και ελέγχου. Έπειτα, συζητείται η μνήμη διαμόρφωσης, η οργάνωσή της και ο μηχανισμός διάδοσης δεδομένων προς τον πίνακα. Τέλος, περιγράφεται το δίκτυο διασύνδεσης τριών επιπέδων ιεραρχίας του επαναδιατάξιμου πίνακα.

3.2.1. Αρχιτεκτονική Επαναδιατάξιμου Κελιού

Ο πίνακας επαναδιατάξιμων κελιών είναι ο προγραμματιζόμενος πυρήνας του MorphoSys. Αποτελείται από 64 πανομοιότυπες μονάδες επεξεργασίας που ονομάζονται επαναδιατάξιμα κελιά (Reconfigurable Cell, RC) και τα οποία έχουν διαταχθεί ώστε να σχηματίσουν έναν πίνακα 8x8 (Σχήμα 3.3). Κάθε κελί είναι η βασική μονάδα της επαναδιάταξης (Σχήμα 3.4) και το μοντέλο λειτουργίας του είναι παρόμοιο με το χειριστή δεδομένων ενός συμβατικού μικροεπεξεργαστή, χωρίς το στάδιο της διευθυνσιοδότησης, του forwarding και άλλες πολύπλοκες λειτουργίες των σύγχρονων επεξεργαστών. Ο έλεγχος, ωστόσο, των κελιών ακολουθεί το μοντέλο των bits διαμόρφωσης που χρησιμοποιείται στα ολοκληρωμένα τύπου FPGA. Όπως φαίνεται στο Σχήμα 3.4, ένα RC αποτελείται από μια μονάδα ALU-πολλαπλασιαστή, μια μονάδα ολίσθησης και δύο πολυπλέκτες για την επιλογή των εισόδων της ALU. Υπάρχει επίσης ένας καταχωρητής εξόδου και ένα αρχείο καταχωρητών με τέσσερις καταχωρητές. Μια λέξη διαμόρφωσης, που φορτώνεται από τη μνήμη διαμόρφωσης και αποθηκεύεται στον καταχωρητή διαμόρφωσης (Context Register) του κάθε κελιού, καθορίζει τη λειτουργία της ALU και την κατεύθυνση/μέγεθος της ολίσθησης στην έξοδο. Η λέξη παρέχει bits ελέγχου στους πολυπλέκτες εισόδου και καθορίζει ποιοι καταχωρητές εγγράφονται μετά από μια λειτουργία. Επιπλέον, η λέξη διαμόρφωσης -που βρίσκεται αποθηκευμένη στον καταχωρητή

διαμόρφωσης- μπορεί επίσης να καθορίσει μια άμεση τιμή ως δεδομένο επεξεργασίας (στο οποίο αναφερόμαστε ως μια "σταθερά").

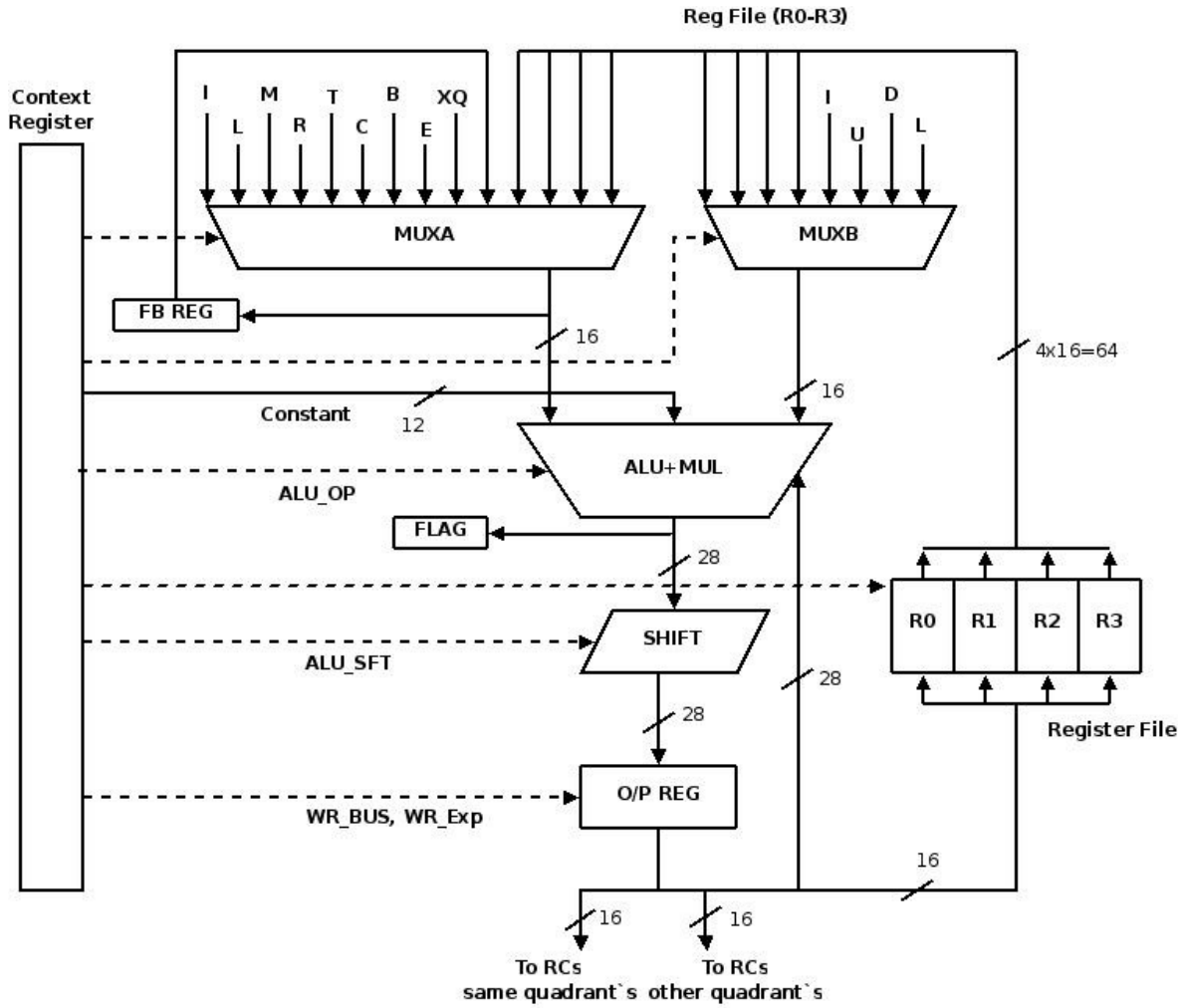
3.2.1.1. Μονάδα ALU-πολλαπλασιαστή

Η ALU σταθερής υποδιαστολής έχει εισόδους των 16-bits και ο πολλαπλασιαστής έχει δύο εισόδους των 16 και των 12 bits, παράγοντας επομένως αποτελέσματα εξόδου των 28 bits. Εξωτερικά, η μονάδα ALU-πολλαπλασιαστής έχει τέσσερις θύρες εισόδου. Δύο θύρες, η θύρα A και η θύρα B υπάρχουν για δεδομένα από τις εξόδους των πολυπλεκτών εισόδου. Μια τρίτη θύρα εισόδου (των 12 bits) λαμβάνει τιμή από το πεδίο σταθεράς του καταχωρητή διαμόρφωσης. Η τέταρτη θύρα εισόδου λαμβάνει είσοδο από τον καταχωρητή εξόδου του κελιού.

Η ALU υποστηρίζει τις κλασσικές λογικές και αριθμητικές λειτουργίες. Συνολικά υποστηρίζονται 25 λειτουργίες. Ωστόσο, μεταξύ των αριθμητικών λειτουργιών της ALU -στις οποίες περιλαμβάνονται η πρόσθεση και η αφαίρεση- υπάρχει και μία λειτουργία για τον υπολογισμό της απόλυτης τιμής της διαφοράς δύο αριθμών. Η ALU έχει, επίσης, λειτουργίες που λαμβάνουν το ένα όρισμα από τη θύρα A και το άλλο από τη θύρα εισόδου της "σταθεράς". Η μονάδα πραγματοποιεί, ακόμα, λειτουργία πολλαπλασιασμού-συσσώρευσης (MAC, Multiply-ACcumulate) σε ένα μόνο κύκλο λειτουργίας. Με τη λειτουργία αυτή δύο δεδομένα πολλαπλασιάζονται και προστίθενται στην προηγούμενη τιμή εξόδου. Ο αθροιστής της ALU έχει σχεδιαστεί για εισόδους των 28 bits, κάτι το οποίο αποτρέπει την απώλεια ακρίβειας κατά τη διάρκεια της λειτουργίας MAC, παρόλο που κάθε έξοδος του πολλαπλασιαστή μπορεί να είναι πολύ μεγαλύτερη από 16 bits (το μέγιστο μέγεθος είναι τα 28 bits).

3.2.1.2. Πολυπλέκτες εισόδου

Οι δύο πολυπλέκτες εισόδου επιλέγουν μία από τις πολλαπλές εισόδους για τη μονάδα του ALU-πολλαπλασιαστή, ανάλογα με τα bits ελέγχου από τη λέξη διαμόρφωσης που υπάρχει στον καταχωρητή διαμόρφωσης. Ο πολυπλέκτης A είναι ένας 16-σε-1 πολυπλέκτης, ενώ ο πολυπλέκτης B είναι ένας 8-σε-1 πολυπλέκτης. Ο πολυπλέκτης A παρέχει εισόδους από τα τέσσερα γειτονικά κελιά και από τα υπόλοιπα κελιά στην ίδια γραμμή και στήλη που βρίσκονται στην ίδια τετράδα. Ακόμα, παρέχεται μια είσοδος από την ταχεία γραμμή δεδομένων (express lane) -όπως περιγράφεται παρακάτω στην υποενότητα για το δίκτυο διασύνδεσης-, μια είσοδος από το διάδρομο δεδομένων του πίνακα, μια είσοδος ανάδρασης, μια είσοδος εκτός της τετράδας (cross-quadrant) και τέσσερις είσοδοι από το αρχείο καταχωρητών. Ο πολυπλέκτης B παρέχει τέσσερις εισόδους από το αρχείο καταχωρητών, είσοδο από το διάδρομο δεδομένων του πίνακα και εισόδους τριών από τα γειτονικά κελιά.



Σχήμα 3.4: Αρχιτεκτονική του Επαναδιατάξιμου Κελιού

3.2.1.3. Τοπικοί Καταχωρητές

Το αρχείο καταχωρητών αποτελείται από τέσσερις καταχωρητές (των 16 bits), οι οποίοι αποδεικνύονται επαρκείς για τις περισσότερες εφαρμογές. Ο καταχωρητής εξόδου έχει πλάτος 32 bits, ώστε να μπορεί να αποθηκεύσει ενδιάμεσα αποτελέσματα των εντολών MAC. Η μονάδα ολίσθησης έχει επίσης πλάτος 32 bits και μπορεί να υλοποιήσει λογικές ολισθήσεις προς τα δεξιά ή τα αριστερά, με μέγεθος 1 έως 15 bits. Ένας ειδικός καταχωρητής του συστήματος προσδιορίζει το πρόσημο του παράγοντα εισόδου στη θύρα A της ALU. Παίρνει την τιμή 0 για θετικούς παράγοντες και 1 για αρνητικούς παράγοντες. Ο καταχωρητής αυτός είναι χρήσιμος όταν η λειτουργία που υλοποιείται εξαρτάται από το πρόσημο του παράγοντα εισόδου, όπως για παράδειγμα στο στάδιο κβαντοποίησης κατά τη διάρκεια της συμπίεσης εικόνας. Τέλος, υπάρχει διαθέσιμος ένας καταχωρητής ανάδρασης σε περίπτωση που κάποιος παράγοντας πρέπει να επαναχρησιμοποιηθεί σε συνεχόμενους κύκλους, όπως σε εφαρμογές εκτίμησης της κίνησης.

3.2.2. Μνήμη Διαμόρφωσης (Context Memory)

Οι πληροφορίες διαμόρφωσης για τον πίνακα αποθηκεύονται στη μνήμη διαμόρφωσης (Context Memory). Επίσης, κάθε επαναδιατάξιμο κελί έχει έναν καταχωρητή διαμόρφωσης, στον οποίο αποθηκεύεται η τρέχουσα λέξη διαμόρφωσης. Η μνήμη διαμόρφωσης είναι οργανωμένη σε δύο μπλοκ, με κάθε μπλοκ να αποτελείται από 8 σύνολα των 16 λέξεων διαμόρφωσης.

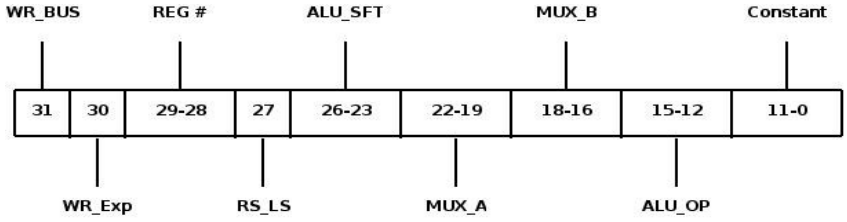
3.2.2.1. Καταχωρητής Διαμόρφωσης

Αυτός ο καταχωρητής με μέγεθος 32 bits περιέχει τη λέξη διαμόρφωσης για το εκάστοτε κελί. Αποτελεί τμήμα καθενός από τα κελιά του επαναδιατάξιμου πίνακα, ενώ η μνήμη διαμόρφωσης είναι ξεχωριστή από τον πίνακα RC.

Τα διαφορετικά πεδία της λέξης διαμόρφωσης φαίνονται στο Σχήμα 3.5. Υπάρχουν δύο παραλλαγές που διαφέρουν στη χρήση του πρώτου πεδίου, του ονομαζόμενου Constant. Για λειτουργίες που περιλαμβάνουν σταθερά ορίσματα, αυτό το πεδίο μπορεί να χρησιμοποιηθεί για να παρέχει αυτά τα ορίσματα στη μονάδα ALU-πολλαπλασιαστή σε κάθε κελί. Ένα παράδειγμα είναι ο πολλαπλασιασμός επί μιας σταθεράς όπου το σταθερό όρισμα μπορεί να δοθεί από τη λέξη διαμόρφωσης. Για τη δεύτερη περίπτωση, κάποια bits του πεδίου Constant χρησιμοποιούνται στο πεδίο SUB_OP, το οποίο επιτρέπει την επέκταση του συνόλου λειτουργιών της ALU.

Το πεδίο ALU_OP καθορίζει τη λειτουργία της μονάδας του ALU-πολλαπλασιαστή. Τα MUX_A και MUX_B καθορίζουν τα bits ελέγχου για τους πολυπλέκτες εισόδου του RC. Άλλα πεδία καθορίζουν την κατεύθυνση (RS_LS) και το μέγεθος της ολίσθησης (ALU_SFT) που εφαρμόζεται στην έξοδο της ALU, καθώς και τους καταχωρητές στους οποίους θα εγγραφεί το αποτέλεσμα μιας λειτουργίας (REG #).

Η λέξη διαμόρφωσης καθορίζει ακόμα αν ένα συγκεκριμένο RC θα γράψει στη δική του ταχεία γραμμή στήλης/γραμμής (express lane) μέσω του πεδίου Write_EXPR. Τέλος το αν ο πίνακας RC θα γράψει το αποτέλεσμα στο Frame Buffer, καθορίζεται επίσης από τα δεδομένα διαμόρφωσης (WR_BUS). (Σχήμα 3.5)



Σχήμα 3.5: Ορισμός bits για τη Λέξη Διαμόρφωσης (RC Context Word)

Ο προγραμματισμός του δικτύου διασύνδεσης επιτελείται επίσης από τη λέξη διαμόρφωσης. Ανάλογα με το περιεχόμενο της λέξης, ένα RC μπορεί να προσπελάσει την είσοδο οποιουδήποτε από τα άλλα RCs στη γραμμή ή στη στήλη του μέσα στην ίδια τετράδα, ή διαφορετικά να επιλέξει μία είσοδο από το τοπικό αρχείο καταχωρητών.

3.2.2.2. Οργάνωση της Μνήμης Διαμόρφωσης

Η μνήμη διαμόρφωσης έχει σχεδιαστεί σαν μία SRAM, με συνολική χωρητικότητα 256 (16x16) λέξεις των 32 bit η κάθε μια. Το περιεχόμενο διαμόρφωσης μπορεί να μεταδοθεί είτε κατά γραμμές είτε κατά στήλες. Επομένως, χρειάζονται 16 σύνολα διαμόρφωσης χωρισμένα σε δύο μπλοκ, ένα μπλοκ με 8 σύνολα για μετάδοση ανά γραμμή στις 8 γραμμές και ένα μπλοκ με 8 σύνολα για μετάδοση ανά στήλη στις 8 στήλες του πίνακα. Όπως έχει ήδη αναφερθεί, το υπολογιστικό μοντέλο του RC υποστηρίζει πολλαπλό περιεχόμενο διαμόρφωσης. Με άλλα λόγια, υπάρχουν πολλαπλές λέξεις διαμόρφωσης στη μνήμη για κάθε ένα από τα 16 σύνολα που αναφέραμε παραπάνω. Έχει διαπιστωθεί ότι ένα βάθος 16 λέξεων διαμόρφωσης για κάθε σύνολο διαμόρφωσης είναι επαρκές για τις περισσότερες εφαρμογές που μελετήθηκαν.

Επιγραμματικά, κάθε λέξη διαμόρφωσης έχει μέγεθος 32 bits, ενώ υπάρχουν 16 λέξεις σε ένα σύνολο διαμόρφωσης της μνήμης και υπάρχουν συνολικά 16 σύνολα διαμόρφωσης.

Το επίπεδο διαμόρφωσης του πίνακα RC (το σύνολο των λέξεων περιεχομένου για τον προγραμματισμό ολόκληρου του επαναδιατάξιμου πίνακα για έναν κύκλο) αποτελείται από 8 λέξεις διαμόρφωσης (μία από κάθε σύνολο) από ένα μπλοκ στήλης ή γραμμής. Έτσι είναι δυνατό να αποθηκευτούν συνολικά 16 επίπεδα διαμόρφωσης σε κάθε block της μνήμης, καθιστώντας το συνολικό αριθμό τους ίσο με 32.

3.2.2.3. Μετάδοση Λέξεων Διαμόρφωσης

Για το MorphoSys, το κύριο βάρος έχει δοθεί σε εφαρμογές με παραλληλία δεδομένων που εμφανίζουν μια δεδομένη κανονικότητα. Με βάση την ιδέα της παραλληλίας και της κανονικότητας, κάθε λέξη διαμόρφωσης μεταδίδεται σε μια στήλη ή γραμμή του RC. Έτσι και τα 8 κελιά σε μια γραμμή ή στήλη, αντίστοιχα, μοιράζονται την ίδια λέξη διαμόρφωσης και επιτελούν την ίδια λειτουργία. Για παράδειγμα ο DCT[26] περιλαμβάνει τον υπολογισμό 8 1-D DCTs, κατά μήκος 8 γραμμών. Η υλοποίηση του αλγορίθμου αυτού επιτυγχάνεται με τη χρήση μόνο 8 λέξεων διαμόρφωσης για κάθε στάδιο του υπολογισμού (με συνολικά 10 στάδια να υπάρχουν)

3.2.2.4. Δυναμική Επαναδιάταξη

Η μνήμη διαμόρφωσης υποστηρίζει τη δυναμική επαναδιάταξη. Κατά τη διάρκεια που ο επαναδιατάξιμος πίνακας εκτελεί μια σειρά λέξεων διαμόρφωσης, ένα άλλο σύνολο δεδομένων διαμόρφωσης μπορεί να φορτωθεί από την κεντρική μνήμη μέσω του ελεγκτή DMA. Το βάθος διαμόρφωσης (32 επίπεδα διαμόρφωσης) επιτρέπει στον πίνακα RC να λειτουργεί συνεχώς, ακόμα κι αν η μνήμη διαμόρφωσης πρέπει να αλλάξει. Δηλαδή, η δυναμική επαναδιάταξη επιτρέπει τη μείωση του ενεργού χρόνου επαναδιάταξης στο μηδέν.

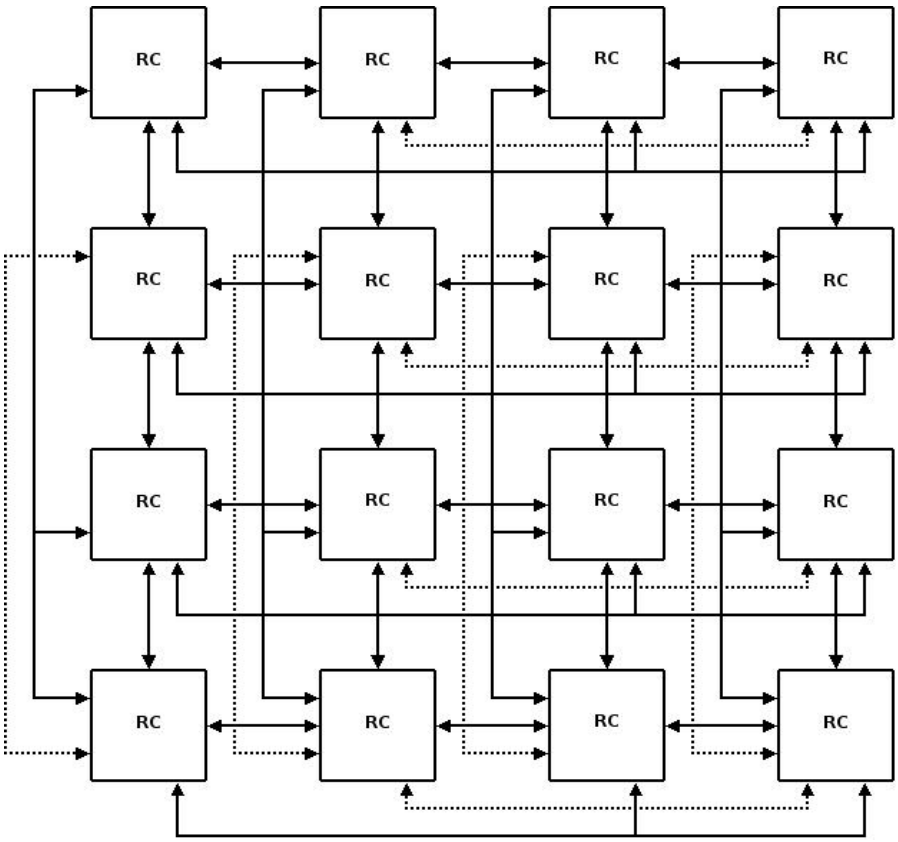
3.2.3. Δίκτυο Διασύνδεσης

Το δίκτυο διασύνδεσης του πίνακα RC αποτελείται από τρία ιεραρχικά επίπεδα.

Πλέγμα πίνακα RC: Το πρώτο επίπεδο διασυνδέσεων στον επαναδιατάξιμο πίνακα είναι ένα πλήρες 2-D πλέγμα. Αυτό παρέχει συνδεσιμότητα καθενός κελιού με κάθε γειτονικό του, προς όλες τις κατευθύνσεις. Δηλαδή κάθε κελί είναι συνδεδεμένο με τα γειτονικά του προς Βορρά, Νότο, Ανατολή και Δύση (Σχήμα 3.6).

Συνδεσιμότητα εντός τετράδας (interquadrant) (πλήρης σε γραμμές/στήλες): Το δεύτερο επίπεδο διασύνδεσης βρίσκεται νοητά στο επίπεδο της τετράδας, με μια τετράδα να είναι ένα 4x4 κομμάτι του πίνακα. Στην τρέχουσα έκδοση του MorphoSys που περιγράφεται, ο επαναδιατάξιμος πίνακας έχει τέσσερις τετράδες (Σχήμα 3.3). Μέσα σε μια τετράδα, κάθε κελί έχει πλήρη συνδεσιμότητα προς τις δύο κατευθύνσεις, όπως φαίνεται στο Σχήμα 3.6. Δηλαδή κάθε κελί μπορεί να προσπελάσει τις εξόδους κάθε ενός κελιού στην ίδια γραμμή και στην ίδια στήλη με αυτό.

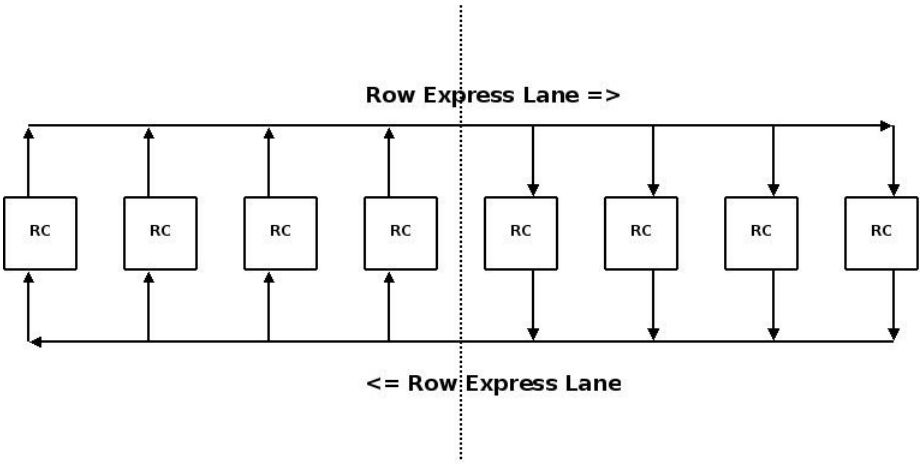
Συνδεσιμότητα εκτός τετράδας (intraquadrant) (express lane): Στο υψηλότερο -ιεραρχικά- επίπεδο οργάνωσης, υπάρχουν οριζόντιοι και κατακόρυφοι διάδρομοι, που ονομάζονται "Express Lanes" και μεταδίδουν δεδομένα μεταξύ γειτονικών τετράδων. Το Σχήμα 3.7 δείχνει δύο "Express Lanes" που πηγαίνουν και προς τις δύο κατευθύνσεις κατά μήκος μιας γραμμής. Αυτοί οι διάδρομοι παρέχουν δεδομένα από οποιοδήποτε κελί (από τα τέσσερα) σε μια γραμμή ή στήλη μιας τετράδας προς άλλα κελιά μιας γειτονικής τετράδας αλλά της ίδιας γραμμής ή στήλης. Αυτό σημαίνει ότι μέχρι και 4 κελιά σε μια γραμμή (στήλη) μπορούν να προσπελάσουν την τιμή εξόδου οποιουδήποτε από τα 4 κελιά στην ίδια γραμμή (στήλη), αλλά σε μια γειτονική τετράδα. Οι express lanes βελτιώνουν σημαντικά τη γενική συνδεσιμότητα του πίνακα και με την παρουσία τους μπορούν να χειριστούν αποτελεσματικά ακόμα και μη συνηθισμένα σχήματα επικοινωνίας, που απαιτούν εκτεταμένες διασυνδέσεις. Για παράδειγμα ένα σχήμα χιαστί διασύνδεσης 8-σημείων ("butterfly" interconnection) μπορεί να υλοποιηθεί σε τρεις μόλις κύκλους.



Σχήμα 3.6: Συνδεσμολογίες εντός μιας τετράδας

Διάδρομος δεδομένων: Ένας διάδρομος δεδομένων από το Frame Buffer προς τον επαναδιατάξιμο πίνακα -με πλάτος 128 bits- συνδέεται στα στοιχεία των στηλών του πίνακα. Παρέχει δύο παράγοντες των 8 bits σε κάθε ένα από τα 8 κελιά μιας στήλης. Είναι δυνατόν να φορτωθούν δύο παράγοντες δεδομένων (στις θύρες A και B) σε μια ολόκληρη στήλη σε έναν κύκλο. Έτσι, χρειάζονται μόνο 8 κύκλοι για τη φόρτωση ολόκληρου του πίνακα RC. Οι έξοδοι των κελιών κάθε στήλης εγγράφονται πάλι πίσω στο Frame Buffer μέσω της θύρας A.

Διάδρομος διαμόρφωσης: Όταν μια εντολή του Tiny RISC καθορίσει ότι ένα συγκεκριμένο σύνολο από λέξεις διαμόρφωσης πρέπει να εκτελεστεί, τότε αυτές οι λέξεις πρέπει να διανεμηθούν από τη μνήμη διαμόρφωσης σε κάθε κελί. Ο διάδρομος διαμόρφωσης μεταφέρει αυτά τα δεδομένα διαμόρφωσης στον καταχωρητή διαμόρφωσης σε κάθε κελί μιας γραμμής (στήλης). Κάθε λέξη διαμόρφωσης έχει πλάτος 32 bits και υπάρχουν 8 γραμμές (στήλες), οπότε ο διάδρομος δεδομένων έχει πλάτος 256 bits. Όταν τα δεδομένα διαμόρφωσης για γραμμές (στήλες) ενεργοποιούνται, τότε η ίδια λέξη διαμόρφωσης μεταδίδεται σε όλα τα κελιά μιας συγκεκριμένης γραμμής (στήλης).



Σχήμα 3.7: Συνδεσμολογία ταχέων διαδρόμων (express lane) – μεταξύ δύο ομάδων κελιών στην ίδια γραμμή αλλά διπλανές τετράδες

3.2.4. Frame Buffer

Το Frame Buffer (FB) είναι μια μνήμη δεδομένων οργανωμένη σε δύο σύνολα (set), τα Set0 και Set1, με κάθε ένα από τα σύνολα να περιέχει δύο ενότητες μνήμης (memory banks). Αυτή η δομή των δύο συνόλων παρέχει επικάλυψη των μεταφορών δεδομένων και των υπολογισμών. Το ένα σύνολο παρέχει τα δεδομένα για τους υπολογισμούς του πίνακα RC και επίσης αποθηκεύει δεδομένα που έχουν επεξεργαστεί στον επαναδιατάξιμο πίνακα. Το άλλο σύνολο αποθηκεύει στην κεντρική μνήμη δεδομένα που είχαν επεξεργαστεί προηγουμένως, μέσω ενός ελεγκτή DMA και φορτώνει δεδομένα για τον επόμενο γύρο υπολογισμών. Αυτές οι λειτουργίες εκτελούνται ταυτόχρονα. Μια ενότητα του FB έχει 64 σειρές των 8 bytes η κάθε μία.

ΚΕΦΑΛΑΙΟ 4
Το Σύστημα MicroSoc

4.1. Γενικά

Το επαναδιατάξιμο σύστημα που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής ακολουθεί τις αρχές των συστημάτων που περιγράφηκαν στην εισαγωγή. Πρόκειται για ένα SoC, στο οποίο υπάρχει συνεργασία μεταξύ ενός κομματιού επαναδιατάξιμης λογικής και ενός επεξεργαστή ARM, με την παρουσία των κατάλληλων εξωτερικών μνημών και των περιφερειακών εκείνων που είναι απαραίτητα για τη σωστή λειτουργία του συστήματος.

Το συγκεκριμένο SoC -χωρίς την επαναδιατάξιμη λογική- είχε ήδη σχεδιαστεί στο παρελθόν (MicroSoC ([1],[2])) σε επίπεδο περιγραφής σε γλώσσα Verilog. Η παρούσα διλωματική αφορά στη σχεδίαση της επαναδιατάξιμης λογικής και στην ενσωμάτωση της στο υπόλοιπο σύστημα. Παρακάτω, θα γίνει μια αναλυτική παρουσίαση του MicroSoC και των χαρακτηριστικών του, ώστε στη συνέχεια να αναλυθεί ο τρόπος σύνδεσης του επαναδιατάξιμου περιφερειακού στο σύστημα αυτό.

Το MicroSoc είναι ένα κλασσικό σύστημα σε ψηφίδα (System on Chip) που περιλαμβάνει έναν 32-bit μικροεπεξεργαστή ARM7, εσωτερική μνήμη SRAM, ROM και σειριακές διεπαφές όπως SPI, I2C, UART. Στόχος της αρχικής σχεδίασής του ήταν η εγκατάσταση μιας πλατφόρμας τόσο για εκπαίδευση σε τεχνικές συσχεδίασης Υλικού/Λογισμικού HW/SW (HardWare/SoftWare Codesign) όσο και για την υλοποίηση ερευνητικών και μη υποσυστημάτων που μπορούν να αναπτυχθούν σε HW, SW ή και συνδυασμό των δύο. Στο Σχήμα 4.1 φαίνεται η δομή του συστήματος

Το σύστημα αυτό δεν έχει υλοποιηθεί πάνω σε ψηφίδα πυριτίου, αλλά έχει σχεδιαστεί πλήρως σε γλώσσα περιγραφής υλικού Verilog, λαμβάνοντας υπόψη τα συνθέσιμα χαρακτηριστικά της γλώσσας. Για το λόγο αυτό, παρότι φαίνεται στο παραπάνω διάγραμμα ότι υπάρχει μια μεγάλη ποικιλία εξωτερικών interfaces (JTAG, SPI, I2C, UART κλπ), στην ειδική περίπτωση της σχεδίασης του επαναδιατάξιμου συστήματος rMicroSoc δεν χρησιμοποιήθηκαν αυτές οι δυνατότητες.

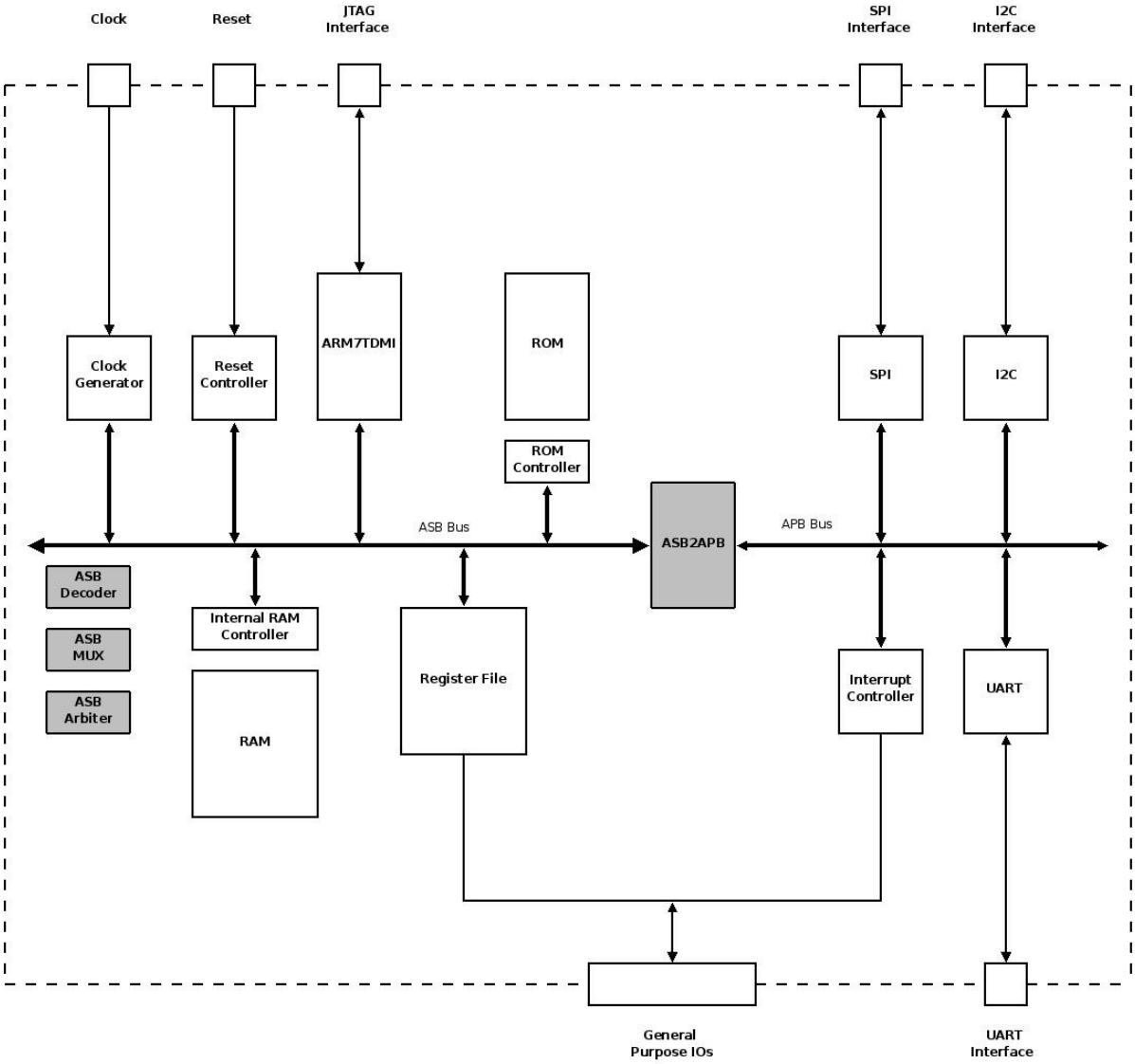
Τα βασικά στοιχεία που απαρτίζουν το MicroSoC είναι :

- *ARM7* – Το βασικό στοιχείο του συστήματος είναι ο ARM7 ο οποίος υποστηρίζει το πρωτόκολλο επικοινωνίας διαδρόμων AMBA-ASB. (περισσότερες πληροφορίες για αυτό παρακάτω). Πρόκειται για έναν 32-

bit RISC επεξεργαστή με αρχιτεκτονική von-Neumann.

- *Γεννήτρια Σήματος Ρολογιού* – Το μπλοκ αυτό δέχεται το ρολόι από το αντίστοιχο PAD (ή PLL) και είναι υπεύθυνο για τη δημιουργία των διάφορων συχνοτήτων που χρειάζεται το σύστημα για να λειτουργήσει. Είναι, επίσης, υπεύθυνο για την ενεργοποίηση/απενεργοποίηση των ρολογιών κάθε επιμέρους μπλοκ. Περιλαμβάνει καταχωρητές για τον προγραμματισμό της διαίρεσης του αρχικού ρολογιού και καταχωρητές για τον προγραμματισμό της ενεργοποίησης του ρολογιού. Υποστηρίζει κι αυτό το πρωτόκολλο επικοινωνίας AMBA-ASB.

- *Γεννήτρια Σήματος Επαναφοράς (reset)* – Το μπλοκ αυτό δέχεται σήμα επαναφοράς (reset) από το αντίστοιχο PAD και είναι υπεύθυνο για την ομαλή μετάβαση του συστήματος από κατάσταση επαναφοράς σε ανενεργή κατάσταση. Επίσης, περιλαμβάνει καταχωρητές για τον προγραμματισμό του reset ανά μπλοκ (software reset). Υποστηρίζει επίσης το πρωτόκολλο επικοινωνίας AMBA-ASB.



Σχήμα 4.1: Γενικό διάγραμμα του MicroSoc

- *Διάδρομος AMBA ASB* – Πρόκειται για ένα ευέλικτο πρωτόκολλο επικοινωνίας με ισχυρές δυνατότητες. Είναι η βάση της επικοινωνίας όλων των επιμέρους υποσυστημάτων με τον ARM. Περισσότερες πληροφορίες θα δοθούν σε αντίστοιχη ενότητα, παρακάτω.

- *Εσωτερικός Ελεγκτής RAM / RAM* – Το μπλοκ αυτό είναι ένα ακόμα περιφερειακό στο διάδρομο ASB. Είναι υπεύθυνο για την υλοποίηση της διεπαφής του επεξεργαστή με την εσωτερική μνήμη SRAM. Υποστηρίζει το πρωτόκολλο επικοινωνίας AMBA-ASB.

- *Αρχείο Καταχωρητών* – Το μπλοκ αυτό περιλαμβάνει καταχωρητές γενικού σκοπού. Οι καταχωρητές ελέγχουν κάποια PADs όσον αφορά στην κατευθυντικότητά τους και τα δεδομένα που μπορούν να εγγραφούν ή να αναγνωστούν.

- *Ελεγκτής ROM / ROM* – Το μπλοκ αυτό είναι ένα ακόμα περιφερειακό στο διάδρομο ASB. Είναι υπεύθυνο για την υλοποίηση της διεπαφής του επεξεργαστή με την εσωτερική μνήμη ROM. Υποστηρίζει το πρωτόκολλο επικοινωνίας AMBA-ASB.

- *ASB2APB* – Το μπλοκ αυτό υλοποιεί την μετατροπή του διαδρόμου ASB σε APB, σύμφωνα πάντα με τις προδιαγραφές των προδιαγραφών του πρωτοκόλλου AMBA. Η μετατροπή της πρόσβασης από τον επεξεργαστή σε κάποιο APB περιφερειακό είναι διάφανης.

- *Ελεγκτής Διακοπών* – Το μπλοκ αυτό είναι υπεύθυνο για τη συλλογή όλων των πηγών διακοπής του συστήματος και την αποστολή του στον ARM. Περιλαμβάνει καταχωρητές για τον έλεγχο, την εφαρμογή "μάσκας" και τον καθαρισμό των πηγών διακοπής.

Στη συνέχεια θα περιγραφούν αναλυτικά τα συστατικά που αποτελούν τον πυρήνα του SoC, δηλαδή η αρχιτεκτονική ARM και η αρχιτεκτονική των διαδρόμων AMBA. Ο επεξεργαστής ARM έχει το γενικό έλεγχο και το σύστημα διαδρόμων AMBA αναλαμβάνει τη σύνδεση και την επικοινωνία μεταξύ του επεξεργαστή και όλων των υπολοίπων μονάδων του SoC. Τέλος, θα γίνει μια αναφορά στον τρόπο που χρησιμοποιείται το MicroSoC στην πράξη, δηλαδή η τεχνική συγγραφής λογισμικού για εκτέλεση από το MicroSoC και σχεδίασης επιπλέον περιφερειακών για επέκταση της λειτουργικότητας του συστήματος.

4.2. Αρχιτεκτονική ARM

4.2.1. Γενικά

Οι ARM[3] είναι μια οικογένεια 32-bit RISC αρχιτεκτονικών, οι οποίες μοιράζονται τις ίδιες αρχές σχεδίασης και ένα κοινό σύνολο εντολών. Έχουν αναπτυχθεί από την εταιρεία ARM Limited. Σήμερα χρησιμοποιούνται ευρύτατα σε ένα μεγάλο αριθμό ολοκληρωμένων συστημάτων. Συγκεκριμένα, υπολογίζεται ότι η οικογένεια ARM συμμετέχει σε περίπου 75% των ενσωματωμένων επεξεργαστών τύπου RISC 32-bit, καθιστώντας την την πιο παραγωγική 32-bit αρχιτεκτονική παγκοσμίως. Στην αγορά των ενσωματωμένων συστημάτων η συγκεκριμένη οικογένεια παίζει κυρίαρχο ρόλο, δεδομένου ότι τα μέλη της παρουσιάζουν χαμηλή κατανάλωση ισχύος. Συνοπτικά, κάποια από τα χαρακτηριστικά που έχουν συμβάλλει στην ευρύτατη εξάπλωση της χρήσης των ARM επεξεργαστών είναι η χαμηλή κατανάλωση ισχύος, η καλή συμβατότητα προς προηγούμενες εκδόσεις, το χαμηλό κόστος και η μεγάλη ποικιλία διαθέσιμων πυρήνων.

Οι επεξεργαστές τύπου ARM, ανάλογα με την εφαρμογή, διατίθονται με διαφορετικά μεγέθη κρυφής μνήμης (cache), διαφορετικά πλάτη διαδρόμων και μεταβλητές συχνότητες λειτουργίας. Ακόμη, υποστηρίζονται και διαφορετικά μοντέλα αρχιτεκτονικής, όπως για παράδειγμα von Neumann (ARM7) ή Harvard (ARM9). Παρόλα αυτά η γλώσσα μηχανής παραμένει η ίδια, ανεξάρτητα από το μοντέλο αρχιτεκτονική του επεξεργαστή.

4.2.2. Χαρακτηριστικά

Ένα βασικό χαρακτηριστικό της οικογένειας ARM είναι ότι ανήκουν στην κατηγορία αρχιτεκτονικών RISC, οι οποίες παρουσιάστηκαν τη δεκαετία του `80 από το πανεπιστήμιο Berkeley. Ορισμένα βασικά χαρακτηριστικά των αρχιτεκτονικών αυτού του τύπου είναι :

- Σταθερό μέγεθος εντολών στα 32-bit με πολύ λίγους διαφορετικούς τύπους εντολών (δηλαδή λίγες διαφορετικές μορφοποιήσεις εντολών)
- Αρχιτεκτονική μοντέλου φόρτωσης-αποθήκευσης (load-store όπως αποκαλείται), στην οποία οι εντολές που επεξεργάζονται δεδομένα εφαρμόζονται μόνο σε καταχωρητές και είναι διαφορετικές από τις εντολές που έχουν πρόσβαση στη μνήμη.
- Ένα μεγάλο αρχείο καταχωρητών αποτελούμενο από 32 καταχωρητές των 32-bit, κάθε ένας από τους οποίους μπορεί να χρησιμοποιηθεί για οποιοδήποτε σκοπό, ώστε να υποστηριχθεί αποδοτικά το μοντέλο της load-store αρχιτεκτονικής.

Αυτά τα χαρακτηριστικά-κλειδιά των RISC επεξεργαστών απλοποίησαν σημαντικά τη σχεδίαση των επεξεργαστών και επέτρεψαν στους σχεδιαστές να υλοποιήσουν την αρχιτεκτονική χρησιμοποιώντας οργανωτικά χαρακτηριστικά που συνέβαλαν με τη σειρά τους στην αύξηση της απόδοσης. Τέτοια είναι η αποκωδικοποίηση των εντολών χρησιμοποιώντας καλωδιωμένη λογική (hard-wired) και η εκτέλεση τους με χρήση τεχνικών διοχέτευσης (pipeline).

Τα βασικά πλεονεκτήματα της RISC αρχιτεκτονικής είναι τρία :

- *Μικρότερη επιφάνεια πυριτίου*. Ένας επεξεργαστής αυτής της αρχιτεκτονικής απαιτεί λιγότερα transistors και μικρότερη επιφάνεια πυριτίου. Συνεπώς, από τη στιγμή που τοποθετηθεί μια CPU τύπου RISC σε ένα ολοκληρωμένο, θα υπάρχει περισσότερος χώρος (σε σχέση με άλλες αρχιτεκτονικές) για την ανάπτυξη χαρακτηριστικών βελτίωσης της απόδοσης, όπως είναι η εσωτερική προσωρινή μνήμη, υλικό για υπολογισμούς μεταβλητής υποδιστολής κλπ.
- *Λιγότερος χρόνος ανάπτυξης.* Ένας επεξεργαστής RISC απαιτεί λιγότερη προσπάθεια σχεδίασης και συνεπώς έχει χαμηλότερο κόστος σχεδίασης.
- *Αυξημένη απόδοση.* Αυτό σχετίζεται με το γεγονός ότι ένας απλός επεξεργαστής (λόγω ακριβώς της απλότητάς του), όπως ένας επεξεργαστής RISC, επιτρέπει την εφαρμογή υψηλών συχνοτήτων ρολογιού.

Η οικογένεια ARM υποστηρίζει τα ακόλουθα χαρακτηριστικά των RISC επεξεργαστών:

- Αρχιτεκτονική load/store
- Περιορισμένο σύνολο εντολών
- Μεγάλο αρχείο καταχωρητών (16x32-bit), με καταχωρητές γενικού σκοπού
- Σταθερό μέγεθος εντολών στα 32 bits ώστε να διευκολυνθεί η αποκωδικοποίηση και η διοχέτευση, με το κόστος της μειωμένης πυκνότητας του κώδικα
- Οι περισσότερες εντολές εκτελούνται σε ένα κύκλο

4.2.3. Βασικές Αρχές Προγραμματισμού

Στη συνέχεια παρουσιάζονται κάποια στοιχεία χρήσιμα για τον προγραμματισμό των ARM επεξεργαστών. Στη λεγόμενη user mode (κατάσταση λειτουργίας χρήστη) οι διαθέσιμοι ενεργοί καταχωρητές γενικού σκοπού που μπορούν να χρησιμοποιηθούν είναι 17, 16 από αυτούς είναι καταχωρητές δεδομένων και ο ένας καταχωρητής κατάστασης. Ο τελευταίος (που ονομάζεται CPSR), χρησιμοποιείται για να υπάρχει έλεγχος και παρακολούθηση των εσωτερικών λειτουργιών. Επιπλέον, από τους 16 καταχωρητές δεδομένων, οι τρεις τελευταίοι έχουν μια ειδική λειτουργία, αφού πρόκειται για τον καταχωρητή-δείκτη στοίβας (*stack pointer*), τον καταχωρητή σύνδεσης (*link register*) και τον μετρητή προγράμματος (*program counter*). Οι υπόλοιποι καταχωρητές μπορούν

να αποθηκεύσουν είτε δεδομένα είτε διευθύνσεις, χωρίς κάποιο περαιτέρω διαχωρισμό. Να σημειωθεί ότι υπάρχουν κι άλλοι καταχωρητές (συνολικά 37 μαζί με τους αυτούς που αναφέρθηκαν ήδη), ωστόσο αυτοί είναι μη ενεργοί και μη διαθέσιμοι στον προγραμματισμό του επεξεργαστή.

Οι διευθύνσεις που χρησιμοποιούν οι ARM έχουν μήκος 32 bits και συνήθως χρησιμοποιούνται ως δεδομένα λέξεις των 32 bits (4 bytes). Η μνήμη που υποστηρίζουν οι ARM είναι οργανωμένη σε θέσεις των 4 byte η κάθε μία.

Όπως αναφέρθηκε, οι ARM εφαρμόζουν το μοντέλο της load/store αρχιτεκτονικής. Οι διαθέσιμες εντολές επεξεργάζονται δεδομένα που είναι αποθηκευμένα σε κάποιον καταχωρητή και μόνο (ή δίνονται απευθείας από την ίδια την εντολή) και αποθηκεύουν οποιοδήποτε αποτέλεσμα της εντολής σε κάποιον καταχωρητή. Οι μόνες εντολές που σχετίζονται με τη μνήμη είναι αυτές που αντιγράφουν δεδομένα από τη μνήμη σε καταχωρητές (εντολές φόρτωσης-load) και αυτές που αντιγράφουν δεδομένα από καταχωρητές σε θέσεις της μνήμης (εντολές αποθήκευσης-store). Προκειμένου η ALU να επεξεργαστεί κάποια δεδομένα, πρέπει αυτά να μεταφερθούν σε κάποιον καταχωρητή. Υπάρχουν φυσικά εντολές για μεταφορά δεδομένων μεταξύ καταχωρητών.

Συνεπώς οι εντολές στους ARM ανήκουν σε τρεις κατηγορίες :

1. *Εντολές επεξεργασίας δεδομένων*, οι οποίες επεξεργάζονται μόνο περιεχόμενα καταχωρητών.
2. *Εντολές μεταφοράς δεδομένων*, για αμφίδρομη μεταφορά δεδομένων μεταξύ της μνήμης και των καταχωρητών.
3. *Εντολές ελέγχου ροής*, για έλεγχο της σειράς εκτέλεσης των εντολών. Να τονισθεί ότι όλες οι εντολές μπορούν να εκτελεστούν με βάση κάποια συνθήκη, κάνοντας τον έλεγχο ροής προγράμματος ευκολότερο να υλοποιηθεί.

Οι ARM επεξεργαστές χειρίζονται τα διάφορα περιφερειακά ως συσκευές με απευθείας απεικόνιση στη μνήμη και υποστήριξη διακοπών. Αυτό το χαρακτηριστικό σημαίνει ότι οι εσωτερικοί καταχωρητές των περιφερειακών αυτών εμφανίζονται στο σύστημα σαν διευθυνσιοδοτημένες θέσεις στο χάρτη μνήμης του ARM και μπορούν να εγγράφονται και να διαβάζονται χρησιμοποιώντας τις ίδιες (load/store) εντολές που χρησιμοποιούνται και για κάθε άλλη απλή θέση στη μνήμη.

Αν τα περιφερειακά χρειάζονται κάποια διαδραστική επικοινωνία με τον επεξεργαστή μπορούν να προσελκύσουν τη προσοχή του κάνοντας αίτηση για διακοπή, χρησιμοποιώντας την είσοδο είτε της κανονικής (IRQ) είτε της γρήγορης (FIQ) διακοπής. Και οι δύο αυτές είσοδοι είναι ευαίσθητες στο επίπεδο του σήματος και μπορούν να ενεργοποιηθούν/απενεργοποιηθούν. Συνήθως οι περισσότερες πηγές διακοπής χρησιμοποιούν την είσοδο IRQ, αφήνοντας την είσοδο υψηλής προτεραιότητας FIQ σε 1-2 πηγές όπου ο χρόνος απόκρισης του επεξεργαστή είναι σημαντικό να είναι μικρός. Να σημειωθεί ότι κατά την εξυπηρέτηση κάποιας διακοπής απενεργοποιούνται όλες οι είσοδοι διακοπών και επιπλέον δεν υποστηρίζονται οι φωλιασμένες διακοπές. Ορισμένα συστήματα

πιθανώς να περιλαμβάνουν υλικό (DMA) για απευθείας πρόσβαση στη μνήμη εξωτερικά του επεξεργαστή. Οι διακοπές που αναφέρθηκαν είναι μια μορφή εξαίρεσης, μεταξύ των πολλών τύπων που υποστηρίζει η αρχιτεκτονική ARM, όπως για παράδειγμα system traps, supervisor calls κ.ά. Η εξυπηρέτηση όλων των τύπων εξαιρέσεων ακολουθεί τα ίδια βασικά βήματα, που περιλαμβάνουν την αποθήκευση της τρέχουσας κατάστασης προγράμματος (στην ουσία αποθήκευση του περιεχομένου των καταχωρητών) και μετάβαση της εκτέλεσης στο σημείο μέσα στη μνήμη, όπου βρίσκεται η ρουτίνα εξυπηρέτησης της εξαίρεσης που έχει συμβεί. Περισσότερες πληροφορίες για τις διακοπές του ARM μπορούν να βρεθούν σε επόμενες ενότητες.

4.3. Πρωτόκολο Επικοινωνίας Διαδρόμου AMBA

4.3.1.Εισαγωγή

Οι προδιαγραφές της αρχιτεκτονικής *Advanced Microcontroller Bus Architecture* (AMBA)([4]) ορίζουν ένα πρωτόκολλο επικοινωνίας για συστήματα SoC με σκοπό τη σχεδίαση ενσωματωμένων συστημάτων υψηλής απόδοσης. Στις προδιαγραφές AMBA έχουν οριστεί τρεις διαφορετικοί διάδρομοι:

- ο *Advanced High-performance Bus* (AHB)
- ο *Advanced System Bus* (ASB) και
- ο *Advanced Peripheral Bus* (APB)

Πίνακας 4.1:Χαρακτηριστικά των τριών διαδρόμων του πρωτοκόλλου AMBA

AMBA AHB	AMBA ASB	AMBA APB
Υψηλή απόδοση	Υψηλή απόδοση	Χαμηλή ισχύς
Λειτουργία pipeline	Λειτουργία pipeline	Απλό interface
Πολλαπλοί master διαδρόμου	Πολλαπλοί master διαδρόμου	Κατάλληλος για πολλά περιφερειακά
Μεταφορές ριπής		Μανδαλωμένες διευθύνσεις και έλεγχος

AHB
Ο διάδρομος AMBA AHB προορίζεται για περιφερειακά συστήματος υψηλής απόδοσης και υψηλής συχνότητας ρολογιού. Ενεργεί, δηλαδή, σαν ο υψηλής απόδοσης διάδρομος (system backbone bus) του συστήματος, υποστηρίζοντας την αποδοτική σύνδεση επεξεργαστών, μνημών πάνω στο ολοκληρωμένο (on-chip) και συστήματα διεπαφής με εξωτερικές μνήμες (off-chip).

ASB
Ο διάδρομος AMBA ASB προορίζεται για περιφερειακά συστήματος με υψηλή

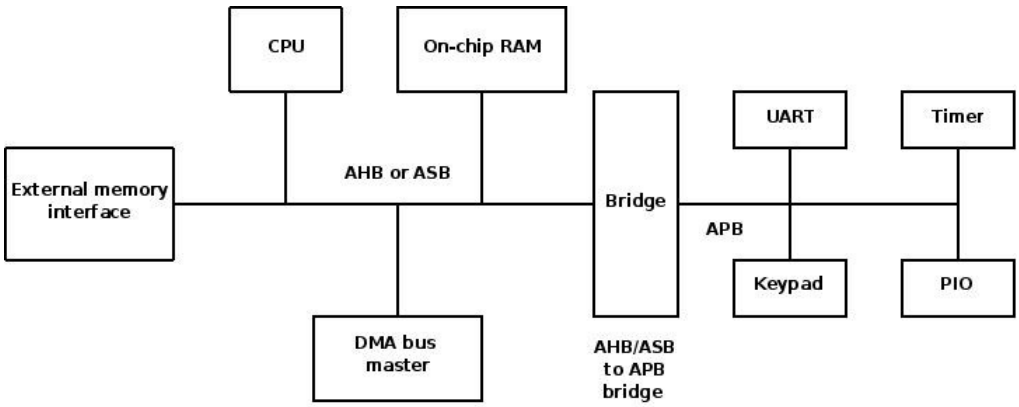
απόδοση. Στην ουσία ο AMBA ASB είναι ένας εναλλακτικός ταχύς διάδρομος κατάλληλος για χρήση όταν δεν απαιτούνται όλα τα χαρακτηριστικά υψηλής απόδοσης του AHB. Ο ASB υποστηρίζει κι αυτός με τη σειρά του την αποδοτική σύνδεση επεξεργαστών, μνημών πάνω στο ολοκληρωμένο και συστήματα διεπαφής με εξωτερικές μνήμες.

APB

Ο διάδρομος AMBA APB προορίζεται για περιφερειακά χαμηλής ισχύος. Έχει βελτιστοποιηθεί ώστε να επιτυγχάνεται χαμηλή κατανάλωση ισχύος και μειωμένη πολυπλοκότητα διεπαφής για την υποστήριξη των συναρτήσεων των περιφερειακών. Ο APB μπορεί να χρησιμοποιηθεί σε συνεργασία με οποιεσδήποτε από τις δύο διαθέσιμες εκδόσεις του διαδρόμου συστήματος (AHB ή ASB).

Τυπικό σύστημα βασισμένο σε AMBA

Ένα τυπικό σύστημα SoC συνήθως αποτελείται από ένα διάδρομο συστήματος AMBA AHB είτε τον AMBA ASB. Αυτός ο διάδρομος είναι ικανός να υποστηρίξει το απαιτούμενο εύρος ζώνης της εξωτερικής μνήμης και σε αυτόν βρίσκονται απευθείας συνδεδεμένα ο επεξεργαστής, η on-chip μνήμη και άλλες μονάδες άμεσης πρόσβασης στη μνήμη (Direct Memory Access – DMA). Αυτός ο διάδρομος παρέχει μια διεπαφή υψηλού εύρους ζώνης (για επικοινωνία υψηλής ταχύτητας) μεταξύ των στοιχείων που συμμετέχουν στην πλειοψηφία των μεταφορών. Επίσης, υπάρχει συνδεδεμένη στο διάδρομο αυτό μια γέφυρα προς το -χαμηλότερου εύρους ζώνης- διάδρομο APB, όπου υπάρχουν συνδεδεμένες οι περισσότερες περιφερειακές μονάδες του συστήματος.



Σχήμα 4.2: Τυπικό σύστημα βασισμένο σε διαδρόμους επικοινωνίας πρωτοκόλλου AMBA

Σε τέτοια συστήματα ο διάδρομος AMBA APB, σαν δευτερεύων διάδρομος σε σχέση με τον υψηλότερου εύρους ζώνης κύριο διάδρομο του συστήματος με δυνατότητες pipeline, παρέχει τη βασική υποδομή επικοινωνίας των περιφερειακών. Τέτοια περιφερειακά συνήθως:

- έχουν διεπαφές που είναι καταχωρητές με απευθείας απεικόνιση στη

μνήμη
- δεν έχουν διεπαφές υψηλού εύρους ζώνης
- προσπελαύνονται μέσω ελέγχου από το πρόγραμμα

Το σύστημα rMicroSoC έχει βασίζεται στο διάδρομο AMBA ASB ως κύριο διάδρομο συστήματος (καθώς και διάδρομο AMBA APB). Παρακάτω θα περιγραφούν συνοπτικά τα τρία είδη διαδρόμων AMBA, ωστόσο θα δοθεί μεγάλη βαρύτητα στις λεπτομέρειες του διαδρόμου AMBA ASB και των σημάτων του, καθώς αυτός αποτελεί το βασικό πόρο επικοινωνίας του MicroSoC. Εξάλλου, το περιφερειακό επαναδιατάξιμης λογικής που σχεδιάστηκε στα πλαίσια της διπλωματικής εργασίας και αποτελεί επέκταση του ήδη υπάρχοντος MicroSoc, είναι μια slave συσκευή προσκολλημένη πάνω στο διάδρομο ASB του συστήματος.

4.3.2. Συνοπτική Περιγραφή των Διαδρόμων AMBA

Διάδρομος AMBA AHB
Ο AHB είναι μια νέα γενιά διαδρόμων AMBA που προορίζεται να ανταποκριθεί στις απαιτήσεις συνθέσιμων σχεδιάσεων υψηλής απόδοσης. Πρόκειται για έναν υψηλής απόδοσης διάδρομο συστήματος που υποστηρίζει πολλαπλά "αφεντικά" (masters) του διαδρόμου και παρέχει λειτουργία υψηλού εύρους ζώνης.

Ο AMBA AHB υλοποιεί τα χαρακτηριστικά που απαιτούνται για συστήματα υψηλής απόδοσης και υψηλής συχνότητας συστήματα περιλαμβάνοντας:
- μεταφορές ριπής (burst transfers)
- διασπασμένες μεταβάσεις (split transactions)
- διαμορφώσεις για ευρύτερου πλάτους διάδρομο δεδομένων (wider data bus configuration)
- μη τρισταθή υλοποίηση (non-tristate implementation)
- λειτουργία στην ακμή του ρολογιού (clock-edge operation)

Οι πιο συνηθισμένοι AHB "σκλάβοι" (slaves) είναι η διεπαφή για την εξωτερική μνήμη, η γέφυρα APB και οποιαδήποτε εσωτερική μνήμη. Θεωρητικά οποιοδήποτε άλλο περιφερειακό θα μπορούσε να συμπεριληφθεί σαν "σκλάβος" AHB, ωστόσο τα περιφερειακά χαμηλού εύρους ζώνης συνήθως βρίσκονται στο διάδρομο APB.

Μια τυπική σχεδίαση συστήματος με AMBA AHB διάδρομο περιλαμβάνει τα ακόλουθα στοιχεία:
- *Master* – Ένας master διαδρόμου (AHB master) έχει ως αρμοδιότητες να αρχικοποιήσει λειτουργίες ανάγνωσης/εγγραφής, παρέχοντας πληροφορίες διεύθυνσης και ελέγχου. Παρόλο που μπορεί να υπάρχουν πολλοί master συνδεδεμένοι στο διάδρομο, μόνο ένας από αυτούς μπορεί ενεργά να χρησιμοποιεί το διάδρομο σε μια οποιαδήποτε χρονική στιγμή.
- *"Σκλάβος"* – Ένας "σκλάβος" διαδρόμου (AHB slave) ανταποκρίνεται σε

μια λειτουργία ανάγνωσης/εγγραφής μέσα σε ένα εύρος χώρου διευθύνσεων. Ο "σκλάβος" διαδρόμου σηματοδοτεί πίσω στον ενεργό master την επιτυχία, αποτυχία ή αναμονή της μεταφοράς δεδομένων.
- *Διαιτητής διαδρόμου* – Ο διαιτητής διαδρόμου (AHB arbiter) εξασφαλίζει ότι μόνο ένας master επιτρέπεται να αρχικοποιήσει μεταφορές δεδομένων κάθε φορά. Παρόλο που το πρωτόκολλο διαιτησίας είναι καθορισμένο, οποιοσδήποτε αλγόριθμος διαιτησίας μπορεί να υλοποιηθεί, ανάλογα με τις απαιτήσεις της εφαρμογής.
- *Αποκωδικοποιητής* – Ο αποκωδικοποιητής AHB (AHB decoder) χρησιμοποιείται για την αποκωδικοποίηση της διεύθυνσης κάθε μεταφοράς και παρέχει ένα σήμα επιλογής για τον "σκλάβο" ο οποίος εμπλέκεται στη μεταφορά. Σε κάθε υλοποίηση του AHB χρειάζεται η ύπαρξη ενός κεντρικού αποκωδικοποιητή διαδρόμου.

Διάδρομος AMBA ASB

Ο διάδρομος ASB είναι η πρώτη γενιά διαδρόμων συστήματος AMBA. Υλοποιεί χαρακτηριστικά που απαιτούνται από συστήματα υψηλής απόδοσης, συμπεριλαμβάνοντας:
- μεταφορές ριπής
- λειτουργία μεταφοράς με pipeline
- δυνατότητα για πολλαπλούς master διαδρόμου

Ένα τυπικό σύστημα βασισμένο στο διάδρομο AMBA ASB μπορεί να περιλαμβάνει έναν ή περισσότερους master διαδρόμου, για παράδειγμα τουλάχιστον τον επεξεργαστή και μια διεπαφή ελέγχου. Παρόλα αυτά είναι σύνηθες να χρησιμοποιούνται μονάδες DMA ή DSP σαν επιπλέον master του διαδρόμου.

Οι πιο κοινοί "σκλάβοι" του ASB διαδρόμου είναι η διεπαφή της εξωτερικής μνήμης, η γέφυρα APB και οποιαδήποτε εσωτερική μνήμη. Όπως αναφέρθηκε, συνήθως τα περιφερειακά χαμηλού εύρους ζώνης δεν προστίθονται στο διάδρομο συστήματος -παρότι υπάρχει αυτή η δυνατότητα- αλλά στο διάδρομο APB.

Μια τυπική σχεδίαση συστήματος με διάδρομο AMBA ASB περιλαμβάνει τα ακόλουθα στοιχεία -όπως ακριβώς στην περίπτωση του διαδρόμου AMBA AHB:
- *"Κύριος" ASB (ASB master)*
- *"Σκλάβος" ASB (ASB slave)*
- *Αποκωδικοποιητής ASB (ASB decoder)*
- *Διατητής διαδρόμου ASB (ASB arbiter)*

Διάδρομος AMBA APB

Ο διάδρομος APB είναι κομμάτι της ιεραρχίας διαδρόμων AMBA και είναι βελτιστοποιημένος ώστε να επιτυγχάνεται ελάχιστη κατανάλωση ισχύος και μειωμένη πολυπλοκότητα των διεπαφών του. Εμφανίζεται ως δευτερεύων

διάδρομος που ενσωματώνεται σαν μια συσκευή "σκλάβου" στο διάδρομο συστήματος AHB ή ASB και παρέχει μια χαμηλής ισχύος επέκταση στο διάδρομο συστήματος.

Η γέφυρα APB εμφανίζεται σαν ένα κύκλωμα "σκλάβου", το οποίο αναλαμβάνει τη χειραψία διαδρόμου και τον επαναχρονισμό των σημάτων ελέγχου από την πλευρά του τοπικού περιφερειακού διαδρόμου.

Ο διάδρομος AMBA APB θα πρέπει να χρησιμοποιείται σαν διεπαφή σε οποιοδήποτε περιφερειακό το οποίο έχει χαμηλό εύρος ζώνης και δεν απαιτεί την υψηλή απόδοση μιας διεπαφής διαδρόμου με λειτουργία pipeline.

Μια τυπική υλοποίηση με διάδρομο AMBA APB περιλαμβάνει μια μοναδική γέφυρα APB που απαιτείται για να μετατρέψει τις AHB ή ASB μεταφορές σε μια κατάλληλη μορφοποίηση για τις συσκευές του APB. Η γέφυρα επίσης παρέχει μανδάλωση όλων των σημάτων διευθύνσεων, δεδομένων ή ελέγχου και ένα δεύτερο επίπεδο αποκωδικοποίησης ώστε να παρέχονται τα σήματα επιλογής προς τις συσκευές-"σκλάβους" του APB.

Όλες οι υπόλοιπες μονάδες που βρίσκονται πάνω στον APB είναι συσκευές "σκλάβοι" του διαδρόμου, των οποίων οι διεπαφές πρέπει να έχουν τις ακόλουθες προδιαγραφές :

- οι διευθύνσεις και τα σήματα ελέγχου πρέπει να είναι διαθέσιμα σε όλη τη διάρκεια της μεταφοράς (χωρίς υποστήριξη για pipeline)
- μηδενική κατανάλωση ενέργειας κατά τη διάρκεια των περιόδων αδράνειας
- ο χρονισμός μπορεί να παρέχεται μέσω της κωδικοποίησης
- τα δεδομένα προς εγγραφή πρέπει να είναι διαθέσιμα σε όλη τη διάρκεια της μεταφοράς

4.3.3. Διάδρομος AMBA ASB

4.3.3.1. Εισαγωγή

Οι προδιαγραφές του Advanced System Bus (ASB) ορίζουν έναν υψηλής απόδοσης διάδρομο που μπορεί να χρησιμοποιηθεί στη σχεδίαση ενσωματωμένων μικροελεγκτών (16 ή 32-bit) με υψηλή απόδοση ή συστημάτων SoC.

Ο AMBA ASB υποστηρίζει την αποτελεσματική σύνδεση των επεξεργαστών, ενσωματωμένων μνημών πάνω στο ολοκληρωμένο και διεπαφών εξωτερικών μνημών με χαμηλής ισχύος περιφερειακά.

Στα τυπικά συστήματα που βασίζονται στο AMBA ASB υπάρχει ένας κεντρικός

διάδρομος αρχιτεκτονικής AMBA ASB, στον οποίο βρίσκονται ο επεξεργαστής, πιθανές μονάδες DMA και συνήθως μια γέφυρα για σύνδεση με το κομμάτι του κυκλώματος, όπου υπάρχει διάδρομος αρχιτεκτονικής AMBA APB.

Ο APB εμφανίζεται σαν ένας τοπικός δευτερεύων διάδρομος που έχει συμπτηχθεί σαν μια μοναδική μονάδα "σκλάβος" του ASB. Ο APB παρέχει μια επέκταση χαμηλής ισχύος στο διάδρομο του συστήματος που χρησιμοποιείται απευθείας με τα σήματα του ASB. Η γέφυρα APB εμφανίζεται σαν μια μονάδα "σκλάβος" που χειρίζεται τη χειραψία διαδρόμου και τον εκ νέου χρονισμό των σημάτων ελέγχου στην πλευρά του τοπικού περιφερειακού διαδρόμου APB.

Η συγκεκριμένη αρχιτεκτονική διαδρόμου AMBA αποτελεί τον κεντρικό διάδρομο του συστήματος MicroSoc και είναι το κομμάτι του συστήματος που είναι υπεύθυνο για την επικοινωνία του κεντρικού επεξεργαστή ARM με όλα τα υπόλοιπα περιφερειακά κυκλώματα. Για το λόγο αυτό, έχει μεγάλη σημασία να μελετηθούν τα σήματα που σχετίζονται με το συγκεκριμένο διάδρομο και οι χρονισμοί τους. Στον επόμενο πίνακα συνοψίζονται τα σήματα του διαδρόμου AMBA ASB και μια σύντομη περιγραφή τους. Σε επόμενες ενότητες περιγράφονται αναλυτικά όλες οι λεπτομέρειες των σημάτων αυτών.

Πίνακας 4.2: Σήματα του διαδρόμου AMBA ASB

Όνομα	**Περιγραφή**
AGNTx Bus grant	Ένα σήμα από τον διαιτητή διαδρόμου προς κάποιον master x που επισημαίνει ότι ο διάδρομος θα του παραχωρηθεί όταν το σήμα BWAIT γίνει LOW. Υπάρχει ένα σήμα AGNTx για κάθε master του συστήματος, όπως και ένα αντίστοιχο σήμα αίτησης, AREQx.
AREQx Bus request	Ένα σήμα από το master x προς το διαιτητή διαδρόμου που επισημαίνει ότι ο master αυτός ζητάει την παραχώρηση του διαδρόμου. Υπάρχει ένα σήμα AREQx από κάθε master του συστήματος, όπως και ένα αντίστοιχο σήμα παραχώρησης διαδρόμου, AGNTx.
BA[31:0] Address bus	Ο διάδρομος διευθύνσεων του συστήματος, που οδηγείται από τον ενεργό master του διαδρόμου.
BCLK Bus clock	Αυτό το σήμα ρολογιού παρέχει το χρονισμό για όλες τις μεταφορές του διαδρόμου. Τόσο η φάση LOW όσο και η φάση HIGH του BCLK χρησιμοποιούνται για τον έλεγχο των μεταφορών στο διάδρομο.
BD[31:0] Data bus	Αυτός είναι ο αμφίδρομος διάδρομος δεδομένων του συστήματος. Ο διάδρομος δεδομένων οδηγείται από τον τρέχοντα ενεργό master του διαδρόμου κατά τη διάρκεια των μεταφορών εγγραφής και από τον επιλεγμένο slave κατά τη διάρκεια των μεταφορών ανάγνωσης.
BERROR Error response	Ένα σφάλμα κατά τη μεταφορά επισημαίνεται από τον επιλεγμένο slave του διαδρόμου χρησιμοποιώντας το σήμα BERROR. Όταν το BERROR είναι στο HIGH τότε έχει συμβεί ένα σφάλμα μεταφοράς, ενώ όταν το BERROR είναι στο LOW τότε η μεταφορά είναι επιτυχής. Αυτό το σήμα χρησιμοποιείται επίσης σε συνδυασμό με το σήμα BLAST για να δηλωθεί η λειτουργία απόσυρσης (retract) του διαδρόμου. Όταν δεν έχει επιλεγεί κανένας slave το σήμα αυτό οδηγείται από τον αποκωδικοποιητή του διαδρόμου.
BLAST Last response	Αυτό το σήμα οδηγείται από τον επιλεγμένο slave του διαδρόμου για να δηλωθεί αν η παρούσα μεταφορά θα είναι η τελευταία από μια μεταφορά ριπής. Όταν το BLAST είναι στο HIGH ο αποκωδικοποιητής πρέπει να επιτρέψει επαρκή χρόνο στην αποκωδικοποίηση της διεύθυνσης. Όταν το BLAST είναι στο LOW, η επόμενη μεταφορά μπορεί να συνεχίσει μια ακολουθία ριπής. Αυτό το σήμα χρησιμοποιείται επίσης σε συνδυασμό με το σήμα BERROR για να δηλωθεί η λειτουργία απόσυρσης (retract) του διαδρόμου. Όταν δεν έχει επιλεγεί κανένας slave το σήμα αυτό οδηγείται από τον αποκωδικοποιητή του διαδρόμου.
BLOK Locked transfers	Όταν το σήμα αυτό είναι στο HIGH δηλώνει ότι η παρούσα και η επόμενη μεταφορά δε θα μπορούν να διαχωριστούν και ότι δεν πρέπει να δοθεί πρόσβαση στο διάδρομο σε κανέναν άλλο master. Το σήμα αυτό χρησιμοποιείτα από το διαιτητή του διαδρόμου και οδηγείται από τον ενεργό master.

BnRES Reset	Το σήμα reset του διαδρόμου είναι ενεργό στο LOW και χρησιμοποιείται για τη λειτουργία reset του συστήματος και του διαδρόμου. Αυτό είναι το μοναδικό σήμα που είναι ενεργό στο LOW.
BPROT[1:0] Protection ctrl	Τα σήματα ελέγχου προστασίας παρέχουν επιπλέον πληροφορίες για μια πρόσβαση στο διάδρομο και χρησιμοποιούνται κυρίως από τον αποκωδικοποιητή του διαδρόμου όταν αυτός λειτουργεί σαν μια βασική μονάδα προστασίας.
BSIZE[1:0] Transfer size	Τα σήματα μεγέθους μεταφοράς δηλώνουν το μέγεθος της μεταφοράς, το οποίο μπορεί να είναι 1 byte, μισή λέξη (2 byte) ή μια λέξη (4byte). Τα σήματα αυτά οδηγούνται από τον ενεργό master του διαδρόμου και έχουν τον ίδιο χρονισμό με το διάδρομο διευθύνσεων.
BTRAN[1:0] Transfer type	Αυτά τα σήματα δηλώνουν τον τύπο της επόμενης μετάβασης, που μπορεί να είναι ADDRESS-ONLY, NONSEQUENTIAL ή SEQUENTIAL. Αυτά τα σήματα οδηγούνται από κάποιο master του διαδρόμου όταν το κατάλληλο σήμα AGNTx έχει ενεργοποιηθεί
BWAIT Wait response	Αυτό το σήμα οδηγείται από τον επιλεγμένο slave του διαδρόμου για να δηλωθεί αν η τρέχουσα μεταφορά μπορεί να ολοκληρωθεί. Αν το BWAIT είναι στο HIGH τότε χρειάζεται ένας ακόμα κύκλος του διαδρόμου, ενώ αν το BWAIT είναι στο LOW τότε η μεταφορά μπορεί να ολοκληρωθεί στον τρέχοντα κύκλο του διαδρόμου. Αν δεν έχει επιλεγεί κανένας slave τότε το σήμα αυτό οδηγείται από τον αποκωδικοποιητή του διαδρόμου.
BWRITE Transfer direction	Όταν το σήμα αυτό είναι στο HIGH τότε δηλώνεται μια μεταφορά εγγραφής ενώ όταν είναι στο LOW δηλώνεται μια μεταφορά ανάγνωσης. Αυτό το σήμα οδηγείται από τον ενεργό master του διαδρόμου και έχει τον ίδιο χρονισμό με το διάδρομο διευθύνσεων.
DSELx Slave select	Ένα σήμα από τον αποκωδικοποιητή του διαδρόμου προς κάποιον slave x του διαδρόμου δηλώνει ότι η συγκεκριμένη συσκευή slave είναι επιλεγμένη και απαιτείται μια μεταφορά δεδομένων. Υπάρχει ένα σήμα DSELx για κάθε slave του διαδρόμου.

4.3.3.2. Σύντομη Περιγραφή του AMBA ASB

Η βασική ροή της λειτουργίας του διαδρόμου είναι:

1. Ο διαιτητής καθορίζει σε ποιο master παραχωρείται η πρόσβαση στο διάδρομο
2. Όταν του παραχωρηθεί ο διάδρομος, ένας master εκκινεί μεταφορές στο διάδρομο
3. Ο αποκωδικοποιητής χρησιμοποιεί τα bits υψηλής τάξης της διεύθυνσης για να επιλέξει ένα "σκλάβο" μέσω του κατάλληλου σήματος επιλογής
4. Ο "σκλάβος" στέλνει μια απάντηση στη μεταφορά πίσω στο master και μεταφέρονται δεδομένα μεταξύ του "σκλάβου" και του master

Υπάρχουν τριών ειδών μεταφορές που μπορούν να εμφανιστούν στο διάδρομο ASB:

NON-SEQUENTIAL : Χρησιμοποιούνται για απλές μεταφορές ή για την πρώτη μεταφορά μιας ριπής.

SEQUENTIAL : Χρησιμοποιούνται για μεταφορές σε μια ριπή. Η διεύθυνση σε μια μεταφορά αυτού του τύπου σχετίζεται πάντα με την προηγούμενη μεταφορά.

ADDRESS ONLY : Χρησιμοποιούνται όταν δεν απαιτείται η μεταφορά δεδομένων. Οι τρεις κύριες χρήσεις για τέτοιου είδους μεταφορές είναι για κύκλους IDLE, για κύκλους HANDOVER του master και για αποκωδικοποίηση διεύθυνσης χωρίς τη συμμετοχή σε μια μεταφορά δεδομένων.

Στις επόμενες ενότητες θα μελετήσουμε αναλυτικά μόνο τις μεταφορές τύπου NON-SEQUENTIAL, αφού είναι και οι μόνες που στην ουσία χρησιμοποιήθηκαν από το συνολικό επαναδιατάξιμο σύστημα που σχεδιάστηκε.

4.3.3.3. Μεταφορές ASB

Όταν ο διάδρομος έχει παραχωρηθεί σε κάποιο master, αυτός μπορεί να υλοποιήσει μία από τα παρακάτω είδη μεταφορών : NON-SEQUENTIAL, SEQUENTIAL ή ADDRESS-ONLY. Μια μεταφορά ορίζεται ότι αρχίζει στην πρώτη αρνητική ακμή του σήματος BCLK μετά την ολοκλήρωση της προηγούμενης μεταφοράς, κάτι το οποίο δηλώνεται με το σήμα BWAIT να βρίσκεται στη στάθμη LOW. Κάθε μεταφορά διαρκεί μέχρι την πρώτη αρνητική ακμή του σήματος BCLK μετά την αποδοχή απάντησης για ολοκλήρωση της μεταφοράς, το οποίο δηλώνεται με το σήμα BWAIT να βρίσκεται στη στάθμη LOW.

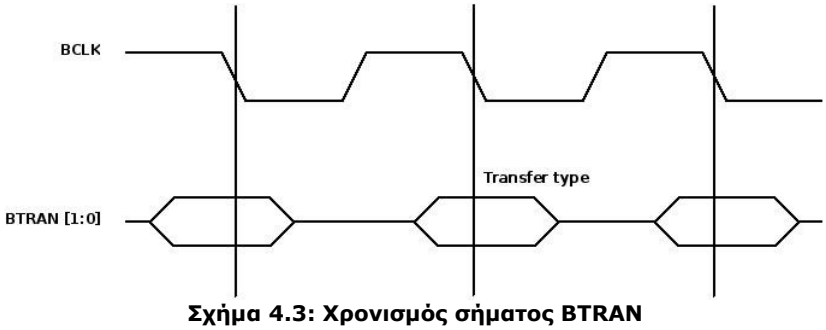
Τύπος μεταφοράς

Ο τύπος της μεταφοράς που ο master θα εκτελέσει μπορεί να καθοριστεί από την τιμή των σημάτων BTRAN στην αρχή της μεταφοράς, όπως δείχνει και ο Πίνακας 4.3. Κατά τη διάρκεια της μεταφοράς τα σήματα BTRAN θα αλλάξουν μόνο όταν πρέπει να δηλώσουν τον τύπο της μεταφοράς που ακολουθεί.

Πίνακας 4.3: Κωδικοποίηση σημάτων BTRAN

BTRAN		**Τύπος μεταφοράς**
[1]	**[0]**	
0	0	ADDRESS-ONLY
0	1	-
1	0	NONSEQUENTIAL
1	1	SEQUENTIAL

Τα σήματα BTRAN οδηγούνται από τον ενεργό master κατά τη διάρκεια της κατάστασης HIGH του σήματος BCLK όταν το σήμα AGNTx είναι στο HIGH (Σχήμα 4.3).



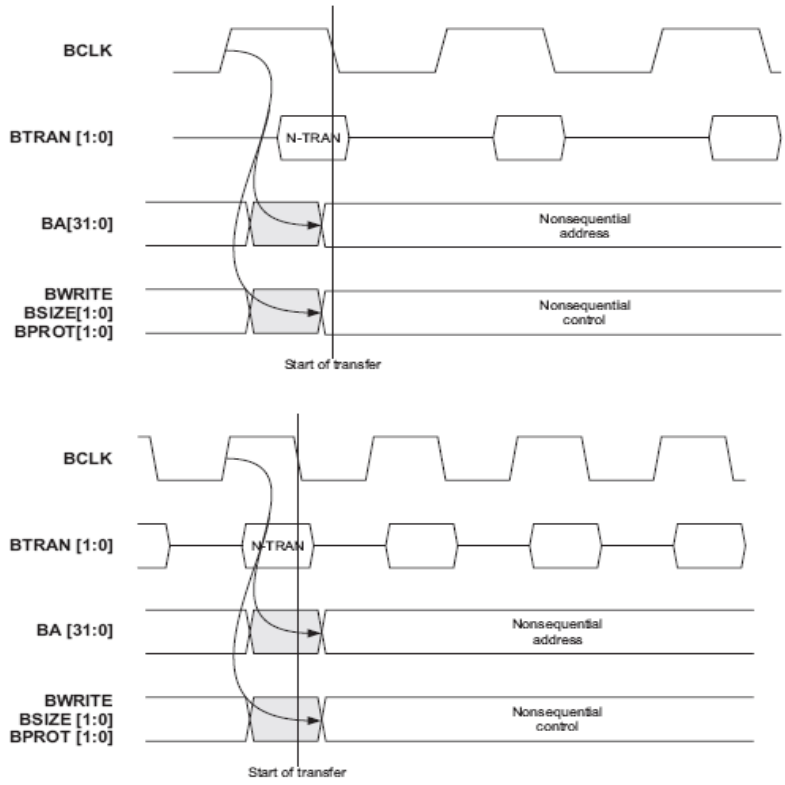
Σχήμα 4.3: Χρονισμός σήματος BTRAN

Πληροφορίες διεύθυνσης και ελέγχου

Τα σήματα διεύθυνσης και ελέγχου του συστήματος είναι :

- *διάδρομος διεύθυνσης (BA[31:0])* – Παρέχει τη διεύθυνση της μεταφοράς
- *κατεύθυνση μεταφοράς (BWRITE)* – Δηλώνει αν πρόκειται για μεταφορά εγγραφής ή ανάγνωσης
- *μέγεθος μεταφοράς (BSIZE[1:0])* – Τα δύο αυτά σήματα κωδικοποιούν το μέγεθος της μεταφοράς, το οποίο μπορεί να είναι ένα εκ των byte (8 bits), halfword (16 bits) και word (32 bits)
- *πληροφορίες προστασίας (BPROT[1:0])* – Τα σήματα αυτά χρησιμοποιούνται όταν ο master του διαδρόμου θέλει να παρέχει επιπλέον πληροφορίες για τη μεταφορά που υλοποιείται. Να σημειωθεί πάντως ότι η πλειοψηφία των "σκλάβων" του διαδρόμου δεν έχουν σχεδιαστεί να υποστηρίζουν τα σήματα αυτά.

Τα παραπάνω σήματα παράγονται από το master του διαδρόμου στη θετική ακμή του BCLK. Παρόλα αυτά ο χρονισμός τους είναι διαφορετικός για μεταφορές NONSEQUENTIAL και SEQUENTIAL, διότι σε κάθε περίπτωση ο master έχει σημαντικές διαφορές στις παραμέτρους του χρονισμού του. Για την περίπτωση των NONSEQUENTIAL μεταφορών ο master του διαδρόμου έχει συχνά πιο αργούς χρόνους έγκυρης εξόδου για τα σήματα διεύθυνσης και ελέγχου, σε σχέση με την περίπτωση των SEQUENTIAL. Στο Σχήμα 4.4 φαίνεται ο χρονισμός στην περίπτωση NONSEQUENTIAL μεταφορών.



Σχήμα 4.4: Χρονισμός διεύθυνσης και σημάτων ελέγχου σε nonsequential μεταφορές, με χαμηλής και υψηλής συχνότητας ρολόγια

Διάδρομος διευθύνσεων

Ο διάδρομος διευθύνσεων έχει πλάτος 32 bits και παρέχει τη διεύθυνση της μεταφοράς. Όλες οι μεταφορές σχετίζονται αποκλειστικά με θέσεις της μνήμης, οπότε όλες οι μνήμες και τα περιφερειακά του συστήματος πρέπει να έχουν μια άμεση απεικόνιση σε ένα εύρος διευθύνσεων από τις οποίες μπορούν να προσπελαστούν. Ο αποκωδικοποιητής χρησιμοποιεί το διάδρομο διευθύνσεων -συνήθως τα υψηλότερης αξίας bits- ώστε να καθορίσει ποιος "σκλάβος" πρέπει να προσπελαστεί.

Κατεύθυνση μεταφοράς

Η κατεύθυνση των μεταφορών ορίζεται από το σήμα BWRITE, όπως δείχνει ο Πίνακας 4.4. Όταν το σήμα BWRITE είναι στο LOW η μεταφορά είναι μια προσπέλαση ανάγνωσης και όταν είναι στο HIGH η μεταφορά είναι μια προσπέλαση εγγραφής.

Πίνακας 4.4: Κωδικοποίηση σήματος BWRITE

BWRITE	Κατεύθυνση μεταφοράς
0	Ανάγνωση
1	Εγγραφή

Απάντηση στη μεταφορά

Για κάθε μεταφορά που αρχικοποιείται από ένα master πρέπει να παραχθεί μια απάντηση και αυτή παρέχεται είτε από τον αποκωδικοποιητή είτε από τον επιλεγμένο "σκλάβο". Αυτή η απάντηση δίνεται χρησιμοποιώντας τον κατάλληλο συνδυασμό τιμών των σημάτων BWAIT, BERROR και BLAST, τα οποία οδηγούνται κατά τη διάρκεια της κατάστασης LOW του ρολογιού και πρέπει να είναι έγκυρα πριν τη θετική ακμή του σήματος BCLK.

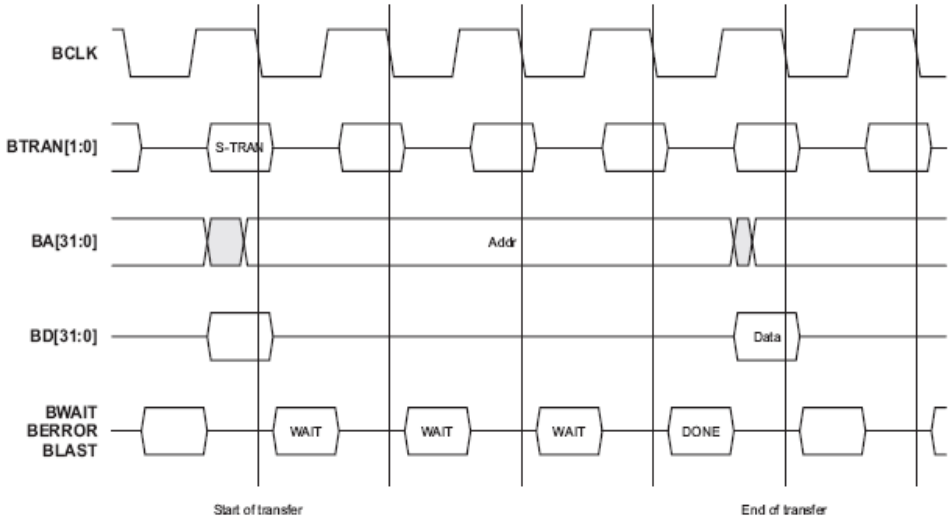
Το Σχήμα 4.5 δίνει ένα παράδειγμα για το πως η απάντηση στη μεταφορά χρησιμοποιείται για την εισαγωγή τριών καταστάσεων αναμονής (WAIT) ώστε να επεκταθεί η διάρκεια μιας μεταφοράς.

Οι δυνατές απαντήσεις σε μια μεταφορά είναι οι ακόλουθες:

- *WAIT* – Η μεταφορά πρέπει να επεκταθεί πριν μπορέσει να ολοκληρωθεί
- *DONE* – Η μεταφορά έχει ολοκληρωθεί με επιτυχία
- *ERROR* – Η μεταφορά έχει ολοκληρωθεί αλλά έχει συμβεί κάποιο σφάλμα. Η κατάσταση σφάλματος θα πρέπει να σηματοδοτηθεί στο master ώστε αυτός να είναι ενήμερος ότι η μεταφορά ήταν μη επιτυχής.
- *LAST* – Η μεταφορά έχει ολοκληρωθεί με επιτυχία αλλά ο "σκλάβος" δεν μπορεί να δεχτεί περαιτέρω μεταφορές ριπής. Αυτή η απάντηση είναι πανομοιότυπη με την απάντηση DONE ως προς το master, αλλά δηλώνει

επιπλέον στον αποκωδικοποιητή ότι πρέπει να εισάγει έναν κύκλο DECODE στην αρχή της επόμενης μεταφοράς.

- *RETRACT* – Η μεταφορά δεν έχει ολοκληρωθεί ακόμα, οπότε ο master θα πρέπει να επαναλάβει τη μεταφορά.



Σχήμα 4.5: Εισαγωγή κύκλων WAIT για επέκταση της διάρκειας μιας μεταφοράς

Στις περισσότερες περιπτώσεις των μεταφορών η απάντηση θα προέλθει από τον επιλεγμένο "σκλάβο". Ωστόσο, ο αποκωδικοποιητής παρέχει απάντηση όταν είτε η μεταφορά είναι τύπου ADDRESS-ONLY είτε η μεταφορά σχετίζεται με μια περιοχή μνήμης όπου δεν έχει απεικονιστεί κανένας "σκλάβος" του διαδρόμου είτε επιχειρείται πρόσβαση σε μια προστατευμένη περιοχή της μνήμης.

Non-sequential μεταφορές

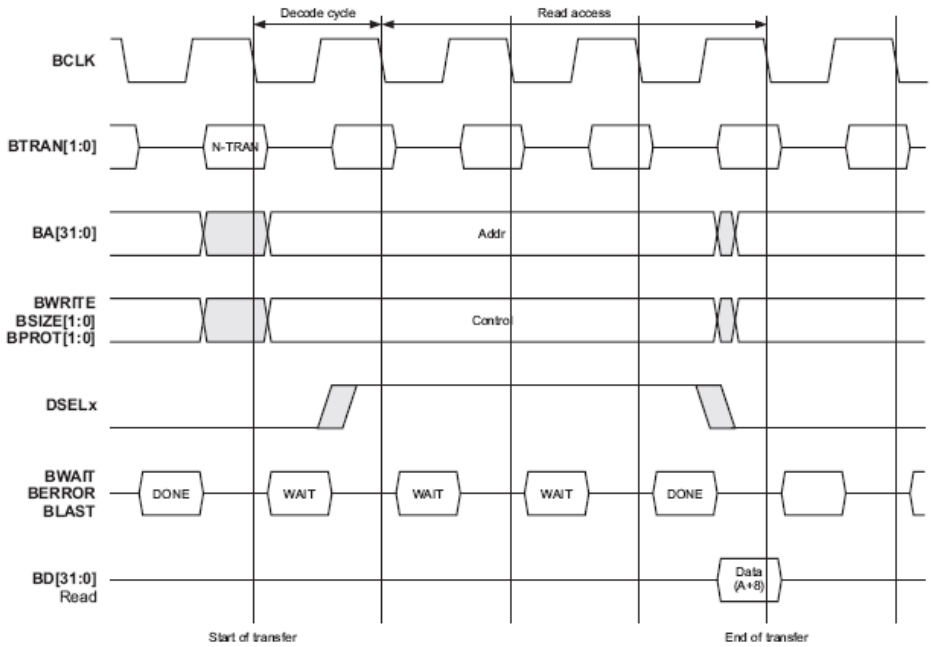
Μια non-sequential μεταφορά εμφανίζεται είτε για μία μοναδική μεταφορά είτε για την πρώτη από μια σειρά συνεχόμενων μεταφορών. Μια τυπική μεταφορά ανάγνωσης τύπου nonsequential, περιλαμβάνοντας και κύκλους WAIT, φαίνεται στο Σχήμα 4.6. Σκοπός του σχήματος αυτού είναι να παρουσιαστούν τα χαρακτηριστικά του συγκεκριμένου τύπου της μεταφοράς. Για περισσότερες λεπτομέρειες σχετικά με τις απαντήσεις από την πλευρά του "σκλάβου" (κύκλοι WAIT κλπ.) καθώς και σχετικά με τα βασικά σημεία των μεταφορών ανάγνωσης και εγγραφής υπάρχουν πληροφορίες σε επόμενες παράγραφους.

Σύμφωνα και με το Σχήμα 4.6, θα πρέπει να τονιστούν τα εξής:

- Τα σήματα διευθύνσεων και ελέγχου αρχίζουν να αλλάζουν κατά τη διάρκεια της περιόδου HIGH του σήματος BCLK ακριβώς πριν την αρχή της μεταφοράς.

- Για μια NONSEQUENTIAL μεταφορά μια έγκυρη διεύθυνση μπορεί να μην είναι έτοιμη παρά αρκετά αργά μέσα στην περίοδο HIGH του σήματος BCLK ή ακόμα και μετά την αρχή της περιόδου LOW του ρολογιού, στην

αρχή της μεταφοράς.



Σχήμα 4.6: Μια nonsequential μεταφορά

- Ο αποκωδικοποιητής, που χρειάζεται μια σταθερή διεύθυνση ώστε να επιλέξει το σωστό slave, θα εισάγει αυτόματα μια κατάσταση αναμονής (WAIT) στον πρώτο κύκλο της μεταφοράς. Αυτός ο κύκλος αναφέρεται ως ένα κύκλος αποκωδικοποίησης (DECODE cycle) και παρέχει επαρκή χρόνο στον αποκωδικοποιητή ώστε να εξετάσει τα υψηλής αξίας bits της διεύθυνσης και να ενεργοποιήσει το κατάλληλο σήμα DSELx κατά τη διάρκεια της περιόδου HIGH του κύκλου DECODE. Να σημειωθεί ότι σε ορισμένα συστήματα, τα οποία συνήθως έχουν σήμα ρολογιού χαμηλής συχνότητας, η διεύθυνση είναι έγκυρη αρκετά νωρίς μέσα στην κατάσταση HIGH του σήματος BCLK πριν την αρχή της μεταφοράς. Με τον τρόπο αυτό επιτρέπεται στον αποκωδικοποιητή να παράγει ένα έγκυρο σήμα DSELx πριν την αρνητική ακμή του σήματος BCLK. Σε τέτοια συστήματα δεν απαιτείται η προσθήκη κύκλου DECODE στην αρχή της nonsequential μεταφοράς. Η λειτουργία ενός τέτοιου συστήματος περιγράφεται με περισσότερη λεπτομέρεια παρακάτω, στην ενότητα "Αποκωδικοποίηση διευθύνσεων". Για να φανεί η διαφορά, παρατίθονται παρακάτω δύο διαγράμματα, το ένα για μεταφορές σε συστήματα χαμηλής συχνότητας και το άλλο για μεταφορές σε συστήματα υψηλής συχνότητας. Στη δεύτερη περίπτωση ο πρώτος κύκλος της μεταφοράς είναι ο κύκλος DECODE που αναφέρθηκε παραπάνω.

- Για τους υπόλοιπους κύκλους της μεταφοράς, ο "σκλάβος" θα παρέχει μια απάντηση στη συγκεκριμένη μεταφορά και τότε θα λάβει χώρα η ανταλλαγή δεδομένων μεταξύ του master και του "σκλάβου".

- Ο διάδρομος δεδομένων, BD[31:0], πρέπει να έχει έγκυρα δεδομένα τη

στιγμή της αρνητικής ακμής του σήματος BCLK στο τέλος της μεταφοράς. Κατά τη διάρκεια ενός κύκλου εγγραφής, ο master του διαδρόμου είναι υπεύθυνος για την οδήγηση του διαδρόμου δεδομένων από την αρχή της κατάστασης HIGH του ρολογιού. Έτσι, ο "σκλάβος" μπορεί να δεχτεί έγκυρα δεδομένα μέχρι την αρνητική ακμή του ρολογιού. Κατά τη διάρκεια ενός κύκλου ανάγνωσης ο κατάλληλος "σκλάβος" πρέπει να διασφαλίσει ότι ο διάδρομος περιέχει έγκυρα δεδομένα μέχρι το τέλος τη κατάστασης HIGH του ρολογιού.

4.3.3.4. Αποκωδικοποίηση Διευθύνσεων

Σε ένα σύστημα βασισμένο σε διάδρομο AMBA ASB, η αποκωδικοποίηση υλοποιείται από έναν κεντρικό αποκωδικοποιητή. Ο αποκωδικοποιητής αυτός χρησιμοποιεί τον τύπο κάθε μεταφοράς για να καθορίσει ποια ακριβώς ενέργεια πρέπει να εφαρμοστεί:

- Για μια μεταφορά τύπου ADDRESS-ONLY ο αποκωδικοποιητής θα απαντήσει με μια απάντηση μεταφοράς τύπου DONE και δε θα επιλεγεί κανένας "σκλάβος".

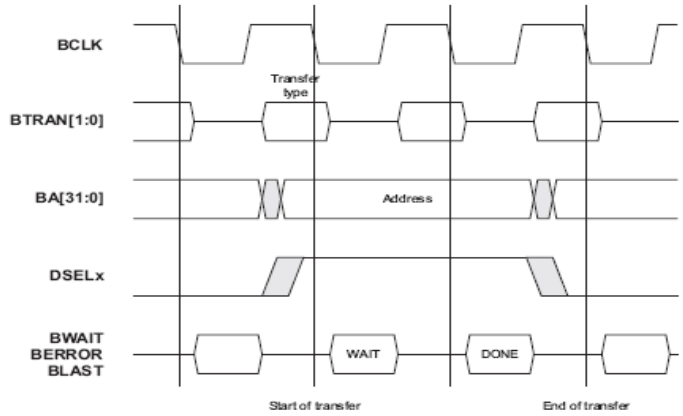
- Για μεταφορές τύπου nonsequential (ή όταν η προηγούμενη μεταφορά είχε τερματιστεί με μια απάντηση με σήμα LAST) ο αποκωδικοποιητής εισάγει ένα μοναδικό κύκλο αναμονής (WAIT) στην αρχή της μεταφοράς ώστε να αφήσει επαρκή χρόνο για την αποκωδικοποίηση της διεύθυνσης, παρόλο που αυτός ο κύκλος μπορεί να μη χρειάζεται σε όλα τα συστήματα. Αυτός ο επιπλέον κύκλος ονομάζεται κύκλος DECODE και κατά τη διάρκειά του δεν ενεργοποιείται κανένα σήμα επιλογής DSELx. Στο δεύτερο κύκλο της μεταφοράς ο αποκωδικοποιητής είτε θα επιλέξει τον κατάλληλο "σκλάβο" είτε θα απαντήσει με ένα σήμα τύπου ERROR. Μια τέτοια απάντηση παρέχεται είτε όταν δεν υπάρχουν "σκλάβοι" που να αντιστοιχούν στη διεύθυνση που παρέχεται από τη μεταφορά, είτε όταν η μεταφορά αναφέρεται σε μια προστατευμένη περιοχή της μνήμης, είτε όταν η κωδικοποίηση της μεταφοράς δεν υποστηρίζεται από τη μνήμη του συστήματος. Στην πιο συνήθη περίπτωση που υπάρχει μια έγκυρη μεταφορά, ο αποκωδικοποιητής θα ενεργοποιήσει το κατάλληλο σήμα επιλογής DSELx και θα επιτρέψει στον επιλεγμένο "σκλάβο" να παρέχει την απάντηση στη μεταφορά στους υπόλοιπους κύκλους της μεταφοράς.

- Τέλος στις μεταφορές τύπου sequential ο αποκωδικοποιητής ενεργοποιεί το κατάλληλο σήμα επιλογής DSELx και ο επιλεγμένος "σκλάβος" παρέχει την κατάλληλη απάντηση στη μεταφορά. Δεν είναι απαραίτητο να αποκωδικοποιηθεί η διεύθυνση αφού θα έχει σίγουρα προηγηθεί μια μεταφορά τύπου nonsequential.

4.3.3.5. Σήματα Επιλογής "σκλάβου"

Όπως προαναφέρθηκε, κάθε συσκευή "σκλάβου" του ASB στο σύστημα έχει ένα σήμα επιλογής DSEL. Αυτό το σήμα δηλώνει στο "σκλάβο" ότι αυτός είναι υπεύθυνος ώστε να παρέχει την απάντηση στη τρέχουσα μεταφορά και ότι απαιτείται μια μεταφορά δεδομένων. Τα σήματα αυτά DSELx (υπάρχει ένα για κάθε "σκλάβο") παράγονται από τον αποκωδικοποιητή, ο οποίος είναι υπεύθυνος ώστε μόνο ένα σήμα DSELx να είναι ενεργό κατά τη διάρκεια μιας μεταφοράς. Υπάρχει βέβαια περίπτωση να μην υπάρχει κανένα σήμα DSELx ενεργό, όπως στις μεταφορές ADDRESS-ONLY.

Τα σήματα DSELx αλλάζουν κατά τη διάρκεια της κατάστασης HIGH του BCLK πριν ακριβώς από την αρχή μιας μεταφοράς και παραμένουν έγκυρα σε όλη τη διάρκεια της μεταφοράς. Αλλάζουν για την επόμενη μεταφορά -αν απαιτείται- στην κατάσταση HIGH του BCLK, αμέσως μετά από μια απάντηση στη μεταφορά όπου το BWAIT είναι στο LOW.



Σχήμα 4.7: Χρονισμοί σημάτων επιλογής

4.3.3.6. Διάδρομος Δεδομένων – Μεταφορές Ανάγνωσης/Εγγραφής

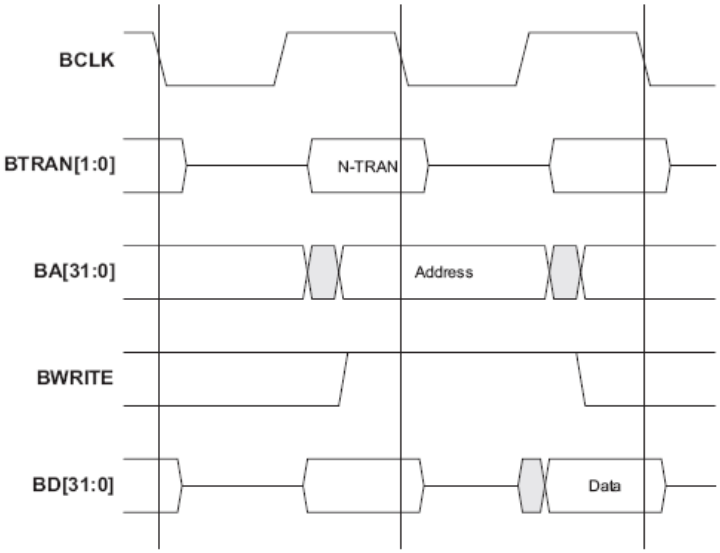
Ο αμφίδρομος διάδρομος δεδομένων με μέγεθος 32 bits, BD[31:0], χρησιμοποιείται για να μεταφέρει δεδομένα μεταξύ των master και των "σκλάβων" του διαδρόμου. Το μέγεθος και η κατεύθυνση της κάθε μεταφοράς καθορίζεται από τα σήματα ελέγχου. Μπορεί να οδηγείται από τον κατάλληλο master ή "σκλάβο" σε οποιαδήποτε στιγμή, εκτός από την πρώτη κατάσταση LOW του σήματος BCLK σε μια μεταφορά nonsequential.

Σε μια μεταφοράς εγγραφής ο master οδηγεί το διάδρομο δεδομένων κατά τη διάρκεια όλων των φάσεων της μεταφοράς, εκτός από την πρώτη κατάσταση LOW του BCLK μιας μεταφοράς nonsequential, ενώ ο "σκλάβος" από τη μεριά του δε συμμετέχει στην οδήγηση του διαδρόμου.

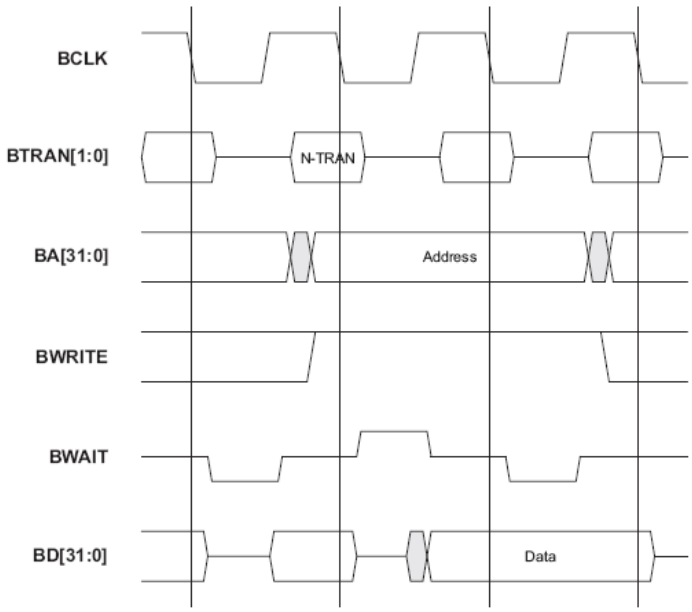
Σε μια μεταφορά ανάγνωσης ο master δε συμμετέχει στην οδήγηση του διαδρόμου, την οποία επιτελεί ο κατάλληλος "σκλάβος" (που έχει επιλεγεί

δηλαδή με το σήμα DSELx), κατά τη διάρκεια όμως της τελευταίας κατάστασης HIGH του σήματος BCLK της μεταφοράς. Για την υπόλοιπη διάρκεια της μεταφοράς ο "σκλάβος" μπορεί να οδηγεί το διάδρομο ή να τον αφήσει σε τρισταθή κατάσταση. Δεν υπάρχει δηλαδή απαίτηση ο διάδρομος να οδηγείται από το "σκλάβο" σε όλη τη διάρκεια της μεταφοράς, αρκεί στην τελευταία κατάσταση HIGH του ρολογιού να υπάρχουν έγκυρα δεδομένα στο διάδρομο.

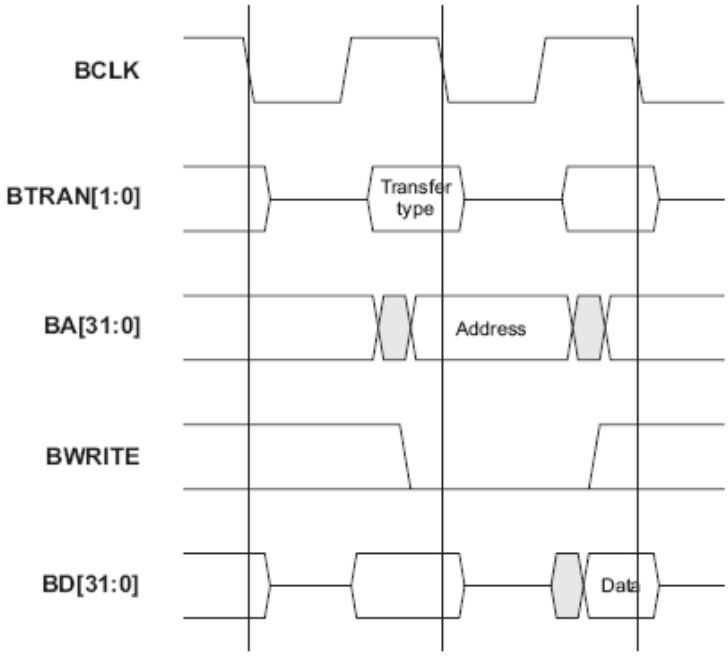
Παρακάτω φαίνονται τρία σχήματα (Σχήμα 4.8, Σχήμα 4.9, Σχήμα 4.10), που αφορούν μεταφορές εγγραφής και ανάγνωσης. Το πρώτο αφορά μια απλή μεταφορά εγγραφής, το δεύτερο μια μεταφορά εγγραφής που έχει επεκταθεί από μια απάντηση WAIT που έχει δοθεί. Στη δεύτερη περίπτωση φαίνεται ότι τα δεδομένα παραμένουν έγκυρα κατά τη διάρκεια των καταστάσεων LOW του BCLK των έξτρα κύκλων που θα απαιτηθούν για να ολοκληρωθεί η μεταφορά.



Σχήμα 4.8: Μια nonsequential μεταφορά εγγραφής



Σχήμα 4.9: Μια nonsequential εκτεταμένη μεταφορά εγγραφής



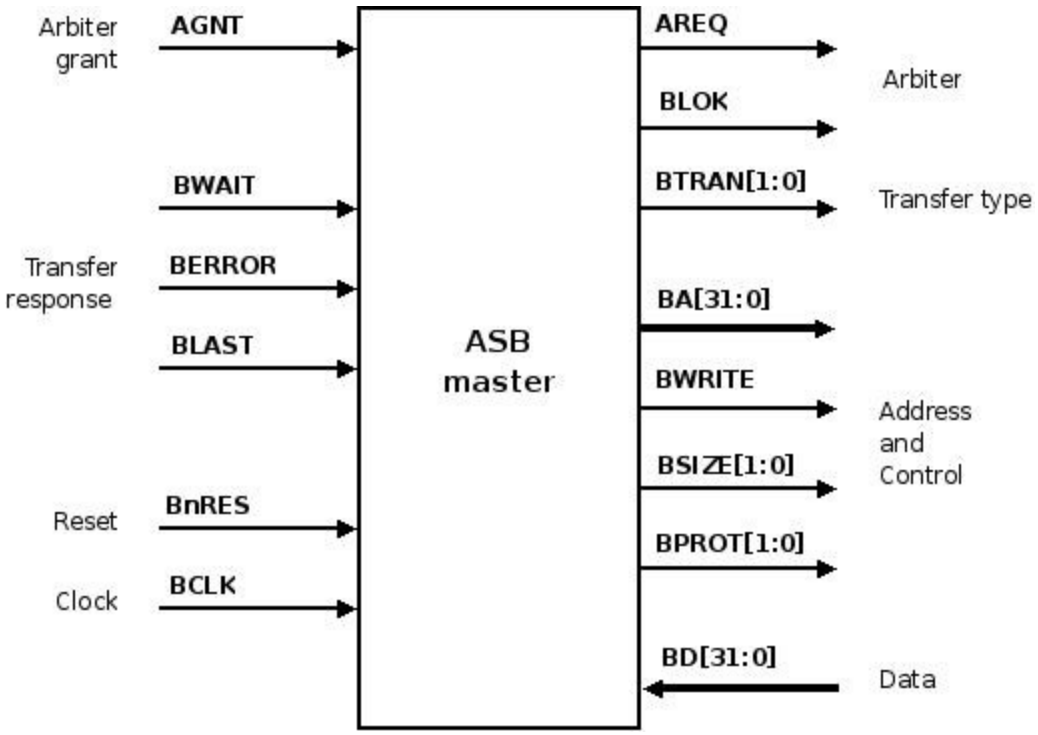
Σχήμα 4.10: Μια nonsequential μεταφορά ανάγνωσης

4.3.3.7. Οι Δομικές Μονάδες του AMBA ASB

Στη παρούσα παράγραφο θα περιγραφούν οι δομικές μονάδες που συναντώνται σε συστήματα με διάδρομο AMBA ASB, δηλαδή ένας master, ένας "σκλάβος", ο αποκωδικοποιητής και ο διαιτητής του διαδρόμου.

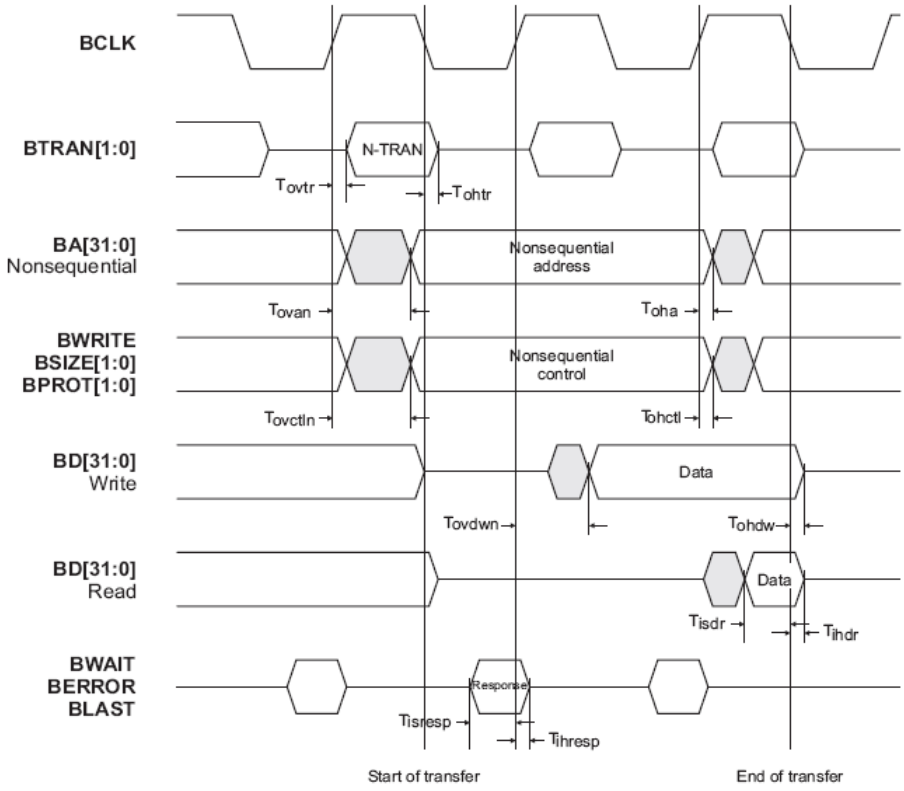
ASB master διαδρόμου

Ένας master του διαδρόμου έχει την πιο πολύπλοκη διεπαφή σε ένα AMBA σύστημα. Συνήθως ο σχεδιαστής ενός συστήματος χρησιμοποιεί προσχεδιασμένους master. Για το λόγο αυτό δε θα αναφερθούν πολλές λεπτομέρειες σχετικά με τη σχεδίασή του. Στο Σχήμα 4.11 φαίνεται το διάγραμμα της διεπαφής ενός master του διαδρόμου όπου φαίνονται οι βασικές ομάδες σημάτων που χρησιμοποιούνται:



Σχήμα 4.11: Διάγραμμα διεπαφής του master του ASB διαδρόμου

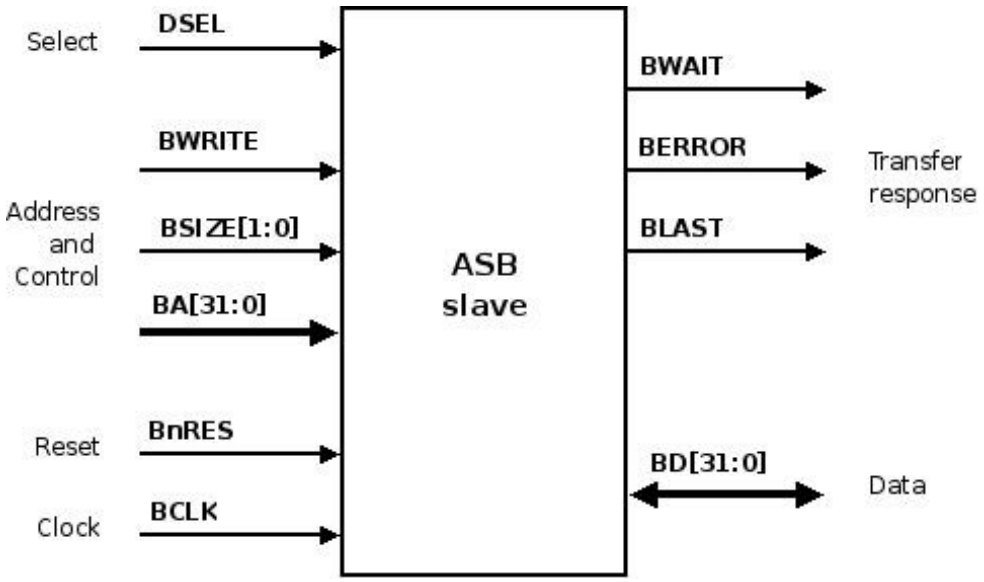
Τέλος παρατίθεται ένα διάγραμμα όπου φαίνονται οι βασικοί χρονισμοί για τον master στην περίπτωση μιας nonsequential μεταφοράς.



Σχήμα 4.12: Χρονισμοί σημάτων για τον ASB master

ASB "σκλάβος" διαδρόμου

Ένας "σκλάβος" του διαδρόμου ASB αποκρίνεται σε μεταφορές που έχουν αρχικοποιηθεί από τους master του διαδρόμου του συστήματος. Χρησιμοποιεί το σήμα επιλογής DSEL από τον αποκωδικοποιητή για να καθορίσει πότε πρέπει να αποκριθεί σε μια μεταφορά. Όλα τα άλλα σήματα που απαιτούνται για τη μεταφορά, όπως η διεύθυνση και οι πληροφορίες ελέγχου θα παραχθούν από τον master του διαδρόμου. Λόγω της παρουσίας του αποκωδικοποιητή η διεπαφή του "σκλάβου" είναι εξαιρετικά απλοποιημένη, καθώς δεν απαιτείται από αυτήν να αντιλαμβάνεται τους διαφορετικούς τύπους των μεταφορών που μπορεί να υπάρξουν. Στο Σχήμα 4.13 φαίνεται η περιγραφή της διεπαφής που πρέπει να έχει ένας "σκλάβος" του διαδρόμου AMBA ASB.



Σχήμα 4.13: Διάγραμμα διεπαφής "σκλάβου" του ASB διαδρόμου

Απόκριση μεταφοράς

Όπως έχει αναφερθεί, ένας "σκλάβος" πρέπει να έχει τη δυνατότητα να παρέχει απόκριση σε μια μεταφορά κατά τη διάρκεια της κατάστασης LOW του σήματος BCLK όταν το σήμα επιλογής DSEL είναι ενεργοποιημένο. Η απόκριση δίνεται χρησιμοποιώντας τα σήματα BWAIT, BERROR και BLAST.

Δεδομένα

Η διεπαφή του "σκλάβου" υλοποιείται επί της ουσίας σαν μια απλή μηχανή δύο καταστάσεων, που αλλάζει καταστάσεις στην αρνητική ακμή του ρολογιού (ανάλογα με το σήμα DSEL) ώστε να καθορίσει πότε μπορεί να συμβεί μεταφορά δεδομένων.

Στις μεταφορές εγγραφής ο "σκλάβος" δειγματοληπτεί τα δεδομένα από το διάδρομο δεδομένων στην αρνητική ακμή του ρολογιού όταν βρίσκεται στην κατάσταση SELECTED (όταν δηλαδή το σήμα επιλογής DSEL είναι

ενεργοποιημένο). Αν απαιτείται, ο "σκλάβος" μπορεί να επεκτείνει τη διάρκεια της μεταφοράς χρησιμοποιώντας τα σήματα απόκρισης μεταφοράς.

Στις μεταφορές ανάγνωσης ο "σκλάβος" πρέπει να οδηγήσει το διάδρομο δεδομένων κατά τη διάρκεια της τελευταίας κατάστασης HIGH του ρολογιού της μεταφοράς.

Αν η μεταφορά επεκταθεί, με την εισαγωγή κύκλων αναμονής, τότε ο "σκλάβος" μπορεί είτε να οδηγήσει το διάδρομο δεδομένων κατά τη διάρκεια των επιπλέων κύκλων της μεταφοράς ή εναλλακτικά μπορεί να τον αφήσει σε τρισταθή κατάσταση μέχρι την τελευταία κατάσταση της μεταφοράς.

Αποκωδικοποιητής διαδρόμου

Ο αποκωδικοποιητής σε ένα σύστημα AMBA ASB χρησιμοποιείται για την υλοποίηση μιας κεντρικής λειτουργίας αποκωδικοποίησης των διευθύνσεων, το οποίο δίνει δύο κύρια πλεονεκτήματα στο σύστημα:

- Αυξάνει τη φορητότητα των περιφερειακών, με το να τα κάνει να μην εξαρτώνται από το πώς έχουν απεικονιστεί στη μνήμη του συστήματος.
- Απλοποιεί τη σχεδίαση και την πολυπλοκότητα μονάδων "σκλάβου" για το διάδρομο, καθώς υπάρχει κεντρική αποκωδικοποίηση των διευθύνσεων και κεντρικός έλεγχος του διαδρόμου.

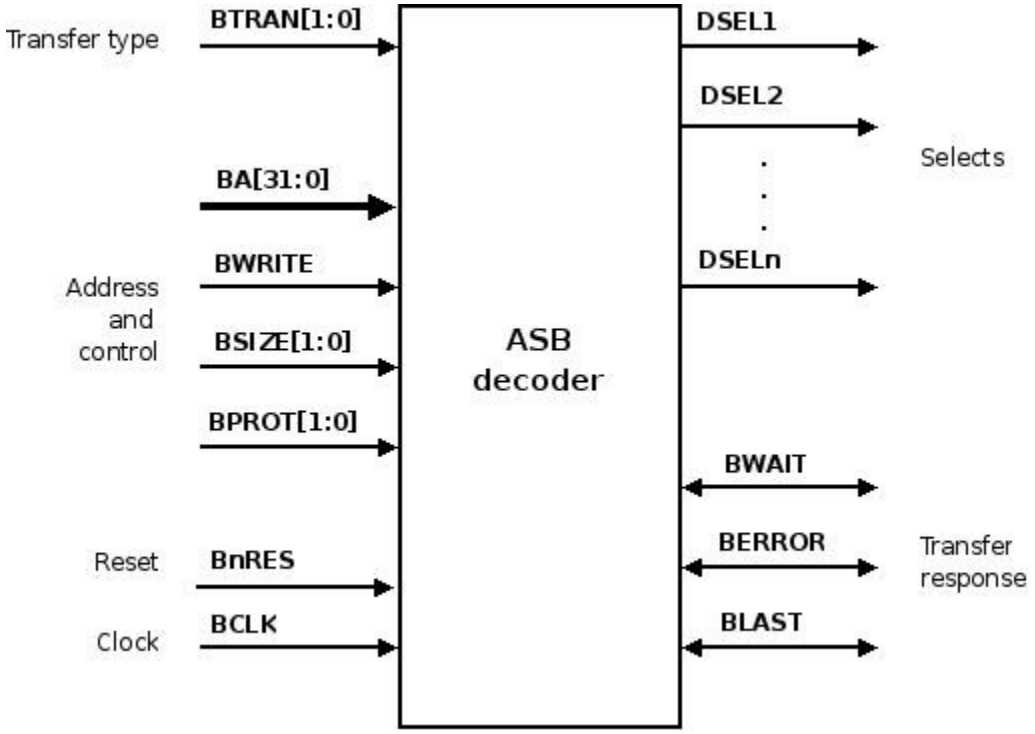
Οι τρεις κύριες λειτουργίες του αποκωδικοποιητή είναι η αποκωδικοποίηση των διευθύνσεων, η προκαθορισμένη απόκριση μεταφοράς (σε περίπτωση που κανένας "σκλάβος" δεν είναι επιλεγμένος) και η δράση του σαν μονάδα προστασίας.

Ένας αποκωδικοποιητής ASB παράγει ένα σήμα επιλογής για κάθε "σκλάβο" στο διάδρομο ASB (για κάθε μεταφορά ένα μόνο είναι ενεργό), αλλά κάτω από προυποθέσεις, δεν επιλέγει κανένα "σκλάβο" και παρέχει την απόκριση μεταφοράς ο ίδιος. Απλοποιεί σημαντικά τη διεπαφή των συσκευών "σκλάβων" και αφαιρεί από το "σκλάβο" την ανάγκη να αντιλαμβάνεται τους διαφορετικούς τύπους της μεταφοράς που μπορεί να συμβούν στο διάδρομο.

Ένα σημαντικό χαρακτηριστικό του αποκωδικοποιητή είναι ότι είναι ικανός να βελτιώσει την απόδοση του συστήματος εισάγοντας τους κύκλους DECODE για την αποκωδικοποίηση των διευθύνσεων. Η προσθήκη ενός τέτοιου κύκλου -όταν απαιτείται- είναι απλή εργασία για τον αποκωδικοποιητή, αφού είναι ικανός να διαχωρίσει αν μια μεταφορά είναι SEQUENTIAL ή NONSEQUENTIAL.

Η βελτίωση της απόδοσης με την εισαγωγή αυτών των κύκλων μπορεί να ερμηνευτεί ως εξής. Σε ένα σύστημα που δεν ακολουθεί το πρωτόκολλο AMBA η κρίσιμη διαδρομή μιας μεταφοράς ανάγνωσης θα περιλάμβανε έξοδο της διεύθυνσης από το master στο διάδρομο διευθύνσεων, αποκωδικοποίηση της διεύθυνσης ώστε να επιλεγεί ο "σκλάβος" και στη συνέχεια έξοδο των δεδομένων στο διάδρομο δεδομένων και απόκριση από το "σκλάβο" πίσω προς το master. Ωστόσο, σε ένα σύστημα AMBA είναι δυνατό να αφαιρεθεί το μεσαίο

στάδιο από τα παραπάνω, όποτε ο master υλοποιεί μια SEQUENTIAL μεταφορά, επειδή σε αυτή την περίπτωση είναι ήδη γνωστό εκ των προτέρων ότι ο "σκλάβος" που θα επιλεγεί θα είναι ο ίδιος με αυτόν της προηγούμενης μεταφοράς. Ο αποκωδικοποιητής μπορεί να χρησιμοποιήσει αυτό το γεγονός για να βελτιώσει την απόδοση του συστήματος εισάγοντας κύκλο WAIT για την αποκωδικοποίηση της διεύθυνσης μόνο όταν αυτό χρειάζεται, δηλαδή στις NONSEQUENTIAL μεταφορές.



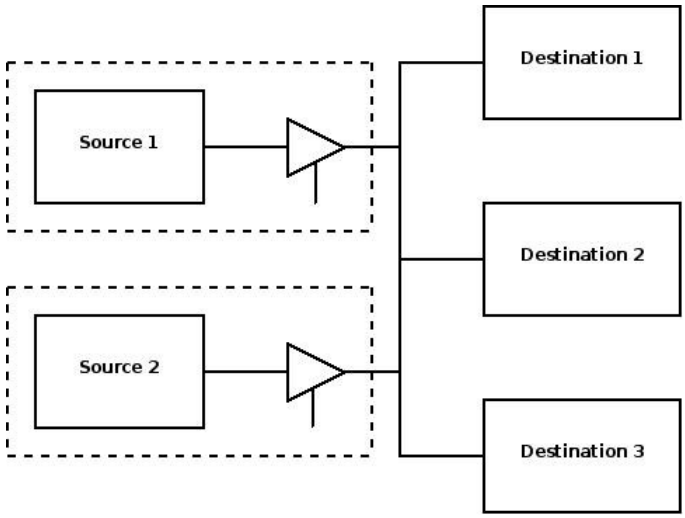
Σχήμα 4.14: Διάγραμμα διεπαφής αποκωδικοποιητή του ASB διαδρόμου

Ο αποκωδικοποιητής χρησιμοποιείται επίσης ώστε να παρέχει μια σειρά λειτουργιών συντήρησης του διαδρόμου. Κατ`αρχήν δρα σαν μια απλή μονάδα προστασίας, που μπορεί να δώσει μια απόκριση ERROR προς ένα master ο οποίος προσπαθεί να προσπελάσει μια απαγορευμένη ή προστατευμένη περιοχή της απεικόνισης της μνήμης. Κατά δεύτερον παρέχει απόκριση μεταφοράς κατά τη διάρκεια ADDRESS-ONLY μεταφορών, όπου κανένας "σκλάβος" δεν επιλέγεται.

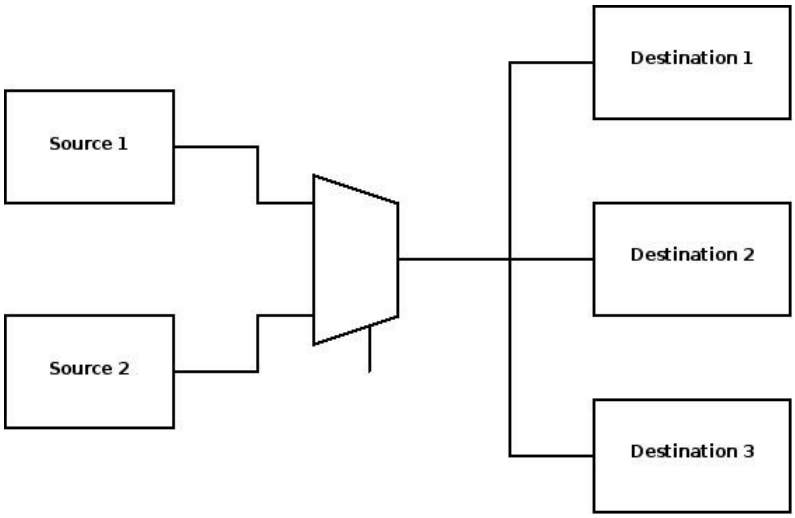
Υπάρχουν δύο δυνατές υλοποιήσεις για τον αποκωδικοποιητή, ανάλογα με τις απαιτήσεις απόδοσης της σχεδίασης του συστήματος. Η συνήθης υλοποίηση ενός αποκωδικοποιητή υποστηρίζει την εισαγωγή των κύκλων DECODE στις NONSEQUENTIAL μεταφορές και τη διάσπαση των μεταφορών ριπής όταν η προσπέλαση στη μνήμη φτάνει στα φυσικά όρια της. Σε κάποια συστήματα ωστόσο, όπου τυπικά η συχνότητα ρολογιού είναι χαμηλή, δεν απαιτείται ο κύκλος DECODE οπότε μπορεί να υλοποιηθεί ένας απλούστερος αποκωδικοποιητής.

4.3.3.8. Τεχνικές Διασυνδέσεων σε Διάδρομο AMBA

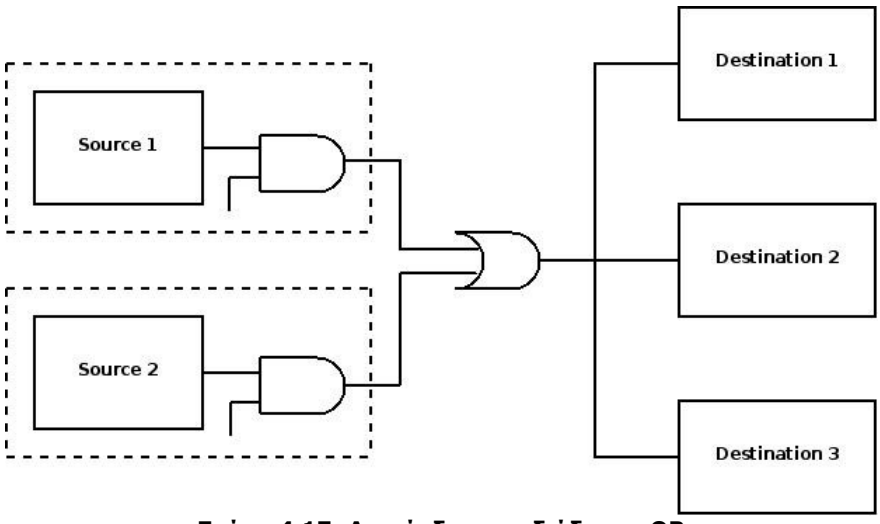
Αρχικά το AMBA είχε σχεδιαστεί για να χρησιμοποιεί τρισταθείς διαδρόμους ως τη κύρια μέθοδο διασύνδεσης μεταξύ των μονάδων πάνω στο ολοκληρωμένο. Ωστόσο,εν τέλει, μπορούν να χρησιμοποιηθούν τρεις διαφορετικές μεθόδοι διασύνδεσης, οι τρισταθείς διάδρομοι, οι διάδρομοι με πολυπλέκτες και οι OR διάδρομοι. Στο , και στο Σχήμα 4.17 φαίνονται απλοικά διαγράμματα των τριών αυτών μεθόδων.



Σχήμα 4.15: Διασύνδεση με τρισταθείς διαδρόμους



Σχήμα 4.16: Διασύνδεση με πολυπλέκτες



Σχήμα 4.17: Διασύνδεση με διάδρομο OR

- Η τρισταθής προσέγγιση χρησιμοποιεί ένα μοναδικό διάδρομο για να συνδέσει μεταξύ τους όλες τις πηγές και τους προορισμούς σήματος. Ωστόσο, μόνο μία πηγή κάθε φορά μπορεί να οδηγεί ενεργά το διάδρομο, ενώ όλες οι υπόλοιπες που είναι συνδεδεμένες στο διάδρομο πρέπει να είναι σε κατάσταση υψηλής αντίστασης (high impedance).

- Η δεύτερη προσέγγιση χρησιμοποιεί έναν πολυπλέκτη για να γίνει η επιλογή μεταξύ των διαφορετικών πηγών του σήματος. Αυτό σημαίνει ότι οδηγείται προς όλους τους προορισμούς η έξοδος από μία μόνο πηγή.

- Η τελευταία προσέγγιση είναι ο διάδρομος OR. Σε αυτή τη μεθοδολογία κάθε μία από τις πηγές πρέπει να έχει την έξοδό της σε κατάσταση LOW, εκτός αν πρόκειται για την ενεργή πηγή του σήματος, οπότε οδηγεί την έξοδό της ανάλογα με την τιμή που απαιτείται. Η έξοδος όλων των πηγών οδηγείται τότε σε μια λογική πύλη OR πριν οδηγηθεί προς όλους τους προορισμούς.

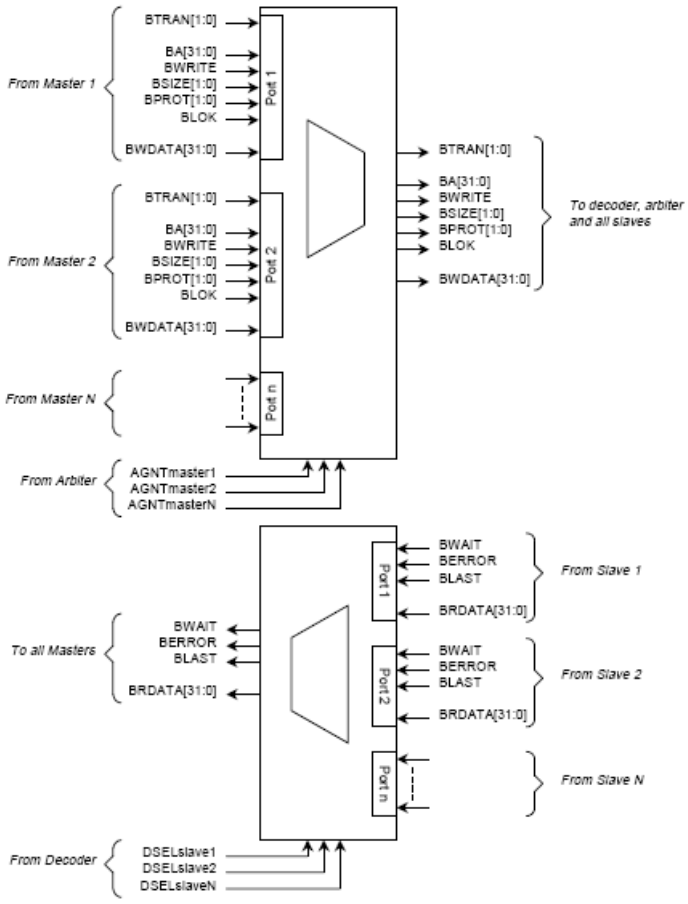
Στο rMicroSoC έχει χρησιμοποιηθεί η δεύτερη προσέγγιση, αυτή με τον πολυπλέκτη. Συνεπώς θα δοθούν περισσότερες λεπτομέρειες για την προσέγγιση αυτή. Στην περίπτωση αυτή δεν απαιτείται επιπλεόν λογική να συμπεριληφθεί γύρω από κάθε μονάδα του συστήματος, παρά ένα κεντρικό συγκρότημα πολύπλεξης.

Συγκεκριμένα, για το διάδρομο ASB χρειάζεται μια ομάδα πολυπλεκτών για την οδήγηση σημάτων από τους πολλαπλούς master προς όλους τους "σκλάβους". Επίσης χρειάζεται μια δεύτερη ομάδα πολυπλεκτών για την οδήγηση των δεδομένων και των αποκρίσεων μεταφοράς από τους διάφορους "σκλάβους" προς τους master. Ένα συνοπτικό διάγραμμα των δύο ομάδων πολυπλεκτών φαίνεται στο Σχήμα 4.18.

Στην πραγματικότητα στο σύστημα rMicroSoC -στην παρούσα μορφή που χρησιμοποιήθηκε- δεν υπάρχουν πολλαπλοί master, οπότε έχει υλοποιηθεί μόνο

η δεύτερη ομάδα πολυπλεκτών.

Πιο αναλυτικά, για την οδήγηση δεδομένων από τους "σκλάβους" προς τους master χρειάζονται δύο πολυπλέκτες, ένας για τις αποκρίσεις μεταφοράς, ελεγχόμενος από το κατάλληλο σήμα DSELx (όπως αυτό έρχεται από τον αποκωδικοποιητή) και ένας για την ανάγνωση δεδομένων, ελεγχόμενος από το κατάλληλο σήμα DSELx, προερχόμενο από μανδαλωτή.



Σχήμα 4.18: Υλοποίηση πολυπλέκτη ASB

4.4. Λεπτομέρειες του MicroSoC

Έχοντας παρουσιάσει τα βασικά σημεία των χαρακτηριστικών του συστήματος επικοινωνίας AMBA, μπορούν να παρουσιαστούν κάποιες λεπτομέρειες της σχεδίασης του MicroSoC που χρησιμοποιήθηκε στην εργασία.

Παρατηρώντας και το Σχήμα 4.1, επισημαίνεται ότι έχει χρησιμοποιηθεί ως κεντρικός διάδρομος του συστήματος ο διάδρομος AMBA ASB, ενώ έχει υλοποιηθεί και διάδρομος APB, για τη σύνδεση περιφερειακών με μικρότερες απαιτήσεις σε ταχύτητα και εύρος ζώνης. Ο διάδρομος APB εμφανίζεται σαν ένα ακόμα περιφερειακό συνδεδεμένο πάνω στο διάδρομο συστήματος ASB, για το οποίο το ρόλο της διεπαφής παίζει η γέφυρα ASB2APB. Η γέφυρα είναι

υπεύθυνη για τη προσαρμογή των χρονισμών και των σημάτων ελέγχου από τον ένα τύπο διαδρόμου στον άλλο.

Στην παρούσα μορφή, το σύστημα δεν έχει σχεδιαστεί να υποστηρίζει την ύπαρξη πολλαπλών master του διαδρόμου και έτσι λειτουργεί με μοναδικό master τον μικροεπεξεργαστή αρχιτεκτονικής ARM7TDMI. Για το λόγο αυτό απουσιάζει και η μονάδα διαιτησίας (arbiter) που θα ήταν απαραίτητη σε ένα σύστημα πολλαπλών master.

Τα περιφερειακά που παίζουν το ρόλο των "σκλάβων" στο διάδρομο ASB είναι τα εξής :

- μια μονάδα γεννήτριας σήματος ρολογιού (Clock Generator)
- μια μονάδα για χαμηλή κατανάλωση ισχύος (Low-Power Device)
- μια μονάδα γεννήτριας σήματος επαναφοράς (Reset Generator)
- ένας καταχωρητής επαναχαρτογράφησης (Remap Register)
- ένα αρχείο καταχωρητών (Register File)
- μια εσωτερική μνήμη RAM
- μια εσωτερική μνήμη ROM

Ολοι οι "σκλάβοι" του συστήματος έχουν απευθείας απεικόνιση στη μνήμη, δηλαδή η επικοινωνία από το master προς αυτούς γίνεται μέσω επικοινωνίας του master με κάποια συμβατική θέση στη μνήμη. Με βάση αυτό το χαρακτηριστικό, σχεδιάζεται ο λεγόμενος χάρτης απεικόνισης στη μνήμη, τον οποίο χρησιμοποιεί ο αποκωδικοποιητής ώστε να ενεργοποιήσει κάθε φορά το κατάλληλο σήμα DSEL προς το περιφερειακό εκείνο που εμπλέκεται με την εκάστοτε μεταφορά. Για την απεικόνιση αυτή χρησιμοποιούνται ορισμένα bits από τα υψηλότερης αξίας bits της διεύθυνσης. Στη συγκεκριμένη περίπτωση χρησιμοποιήθηκαν τα 4 υψηλότερης αξίας bits, οπότε το σύστημα υποστηρίζει συνολικά 16 περιφερειακά-"σκλάβους" για το διάδρομο ASB. Ο Πίνακας 5.1 παρουσιάζει το χάρτη απεικόνισης της αρχικής σχεδίασης του MicroSoC.

Όπως είναι φανερό, υπάρχουν κι άλλα σήματα επιλογής (άρα και θέσεις στο χάρτη απεικόνισης) τα οποία είναι ελεύθερα και μπορούν να αντιστοιχισθούν σε κάποιο νέο "σκλάβο" που τυχόν θα προστεθεί στο σύστημα.

Στο διάδρομο APB υπάρχουν περιφερειακά για τη διασύνδεση με σειριακές θύρες και συγκεκριμένα θύρες SPI, I2C και UART. Υπάρχει ακόμα και μια μονάδα ελέγχου διακοπών, η οποία είναι υπεύθυνη για τη συλλογή όλων των πηγών διακοπής του συστήματος (που προέρχονται είτε από εσωτερικά σήματα ή από το εξωτερικό περιβάλλον) και την αποστολή αίτησης για διακοπή στον ARM.

Υπάρχουν τριών ειδών μνήμες στο σύστημα.

- Μία μνήμη ROM, όπου είναι αποθηκευμένος ο κώδικας προγράμματος που εκτελεί ο μικροεπεξεργαστής ARM και έχει μέγεθος 1K x 32 = 32KB. Περισσότερες πληροφορίες σχετικά με τον προγραμματισμό του MicroSoC και για τον τρόπο προσθήκης προγραμμάτων στη μνήμη ROM δίνονται σε επόμενη ενότητα.

- Μία μνήμη RAM, όπου αποθηκεύονται διάφορα δεδομένα από την εκτέλεση του εκάστοτε προγράμματος, που έχει μέγεθος επίσης 32KB.

- Ένα αρχείο καταχωρητών (register file), με μέγεθος 6 καταχωρητές των 32 bits ο καθένας.

Πίνακας 4.5: Χάρτης απεικόνισης για το σύστημα MicroSoc

bits διεύθυνσης BA[31:28]	**"Σκλάβος"**	**Σήμα DSELx**
0001	ROM	DSEL1
0010	RAM	DSEL2
0011	RegFile	DSEL3
0100	-	DSEL4
0101	Clock Generator	DSEL5
0110	ASB2APB	DSEL6
0111	Reset Generator	DSEL7
1000	LowPower Device	DSEL8
1001	Remap Register	DSEL9
1010	-	DSEL10
1011	-	DSEL11
1100	-	DSEL12
1101	-	DSEL13
1110	-	DSEL14
1111	-	DSEL15

Όσον αφορά, τέλος, στα υπόλοιπα συστατικά του διαδρόμου ASB σημειώνεται ότι απουσιάζει ο διαιτητής διαδρόμου, λόγω της απουσίας παραπάνω του ενός master. Αντίθετα, υπάρχει ο απαραίτητος αποκωδικοποιητής για τη αποκωδικοποίηση της διεύθυνσης σε κάθε μεταφορά και την ενεργοποίηση του κατάλληλου σήματος επιλογής DSELx. Επιπλέον, υπάρχει ένα σύστημα πολυπλεκτών που αναλαμβάνουν τις διασυνδέσεις μεταξύ του master του διαδρόμου -δηλαδή του μικροεπεξεργαστή αρχιτεκτονικής ARM- και των διαφόρων "σκλάβων". Μέσω των πολυπλεκτών αυτών γίνεται η μεταφορά προς τον μικροεπεξεργαστή των δεδομένων από τον κατάλληλο "σκλάβο" που έχει επιλεγεί σε κάθε μεταφορά, η επιλογή της κατάλληλης απόκρισης μεταφοράς από το "σκλάβο" που είναι επιλεγμένος και η μεταφορά δεδομένων από τον μικροεπεξεργαστή προς κάθε "σκλάβο" του συστήματος.

4.5. Προγραμματισμός του MicroSoC (Firmware)

4.5.1. Συγγραφή Λογισμικού

Το λογισμικό το οποίο σχετίζεται με τη λειτουργία του MicroSoc αποτελείται από δύο μέρη, το τμήμα εκκίνησης (boot) και την εφαρμογή χρήστη.

Τμήμα Εκκίνησης
Το τμήμα εκκίνησης αποτελείται κυρίως από τη ρουτίνα χειρισμού του Reset και των υπόλοιπων διακοπών που μπορεί να συμβούν στο σύστημα και είναι γραμμένο σε assembly. Το αρχείο που περιέχει το τμήμα εκκίνησης είναι το αρχείο πηγαίου κώδικα *armcore.s*.

Η ρουτίνα χειρισμού του reset (ResetHandler) εκτελεί την εκκίνηση, δηλαδή αντιγράφει (αν χρειάζεται) κάποιο πρόγραμμα από την εξωτερική flash ή sram και εγκαθιστά τους δείκτες στοίβας. Στη συνέχεια κάνει μεταφορά του ελέγχου στο σημείο έναρξης της εφαρμογής του χρήστη (C_Entry).

Ο τρόπος με τον οποίο θα κάνει εκκίνηση ο ARM εξαρτάται από δύο pins του ολοκληρωμένου (*conf[1:0]*).

Υπάρχουν οι εξής περιπτώσεις:
- *Εκκίνηση από τη ROM (conf=00) :* Το πρόγραμμα εκκίνησης περιέχεται στη ROM και ξεκινάει από τη δ/νση 0x00000000. Για εγγραφή δεδομένων χρησιμοποιείται η RAM που καταλαμβάνει το χώρο διευθύνσεων (0x20000000-0x2fffffff).
- *Εκκίνηση από τη flash & Επαναχαρτογράφηση (remap) (conf=01) :* Στην περίπτωση αυτή το πρόγραμμα εκκίνησης βρίσκεται στη ROM, αλλά η εφαρμογή χρήστη στην εξωτερική flash. Αρχικά η ROM καταλαμβάνει το χώρο δ/νσεων 0x0xxxxxxx και η RAM το χώρο 0x2xxxxxxx όπως και πριν. Ο ARM εκκινείται κανονικά εκτελώντας το ResetHandler που βρίσκεται στη ROM. Βλέποντας ότι το pin *conf* έχει την τιμή 01, αντιγράφει την εφαρμογή χρήστη από τη flash στη RAM και στη συνέχεια εγγράφει τον καταχωρητή *remap*. Με την εγγραφή αυτή η RAM πλέον απεικονίζεται στο χώρο δ/νσεων 0x0xxxxxxx.
- *Εκκίνηση από την SRAM & Επαναχαρτογράφηση (remap) (conf=10)* : Όπως και στην προηγούμενη περίπτωση, μόνο που τώρα χρησιμοποιείται η εξωτερική μνήμη SRAM.

Από τις υπόλοιπες ρουτίνες χειρισμού, αυτές των διακοπών IRQ και FIQ καλούν εξωτερικές συναρτήσεις γραμμένες συνήθως σε C (IrqHandler και FiqHandler αντίστοιχα).

Εφαρμογή χρήστη
Η εφαρμογή χρήστη γράφεται συνήθως σε γλώσσα προγραμματισμού C.

Απαιτείται να περιέχει μία συνάρτηση *C_Entry()* η οποία αποτελεί το σημείο εκκίνησης της εφαρμογής μας, και την οποία καλεί η ρουτίνα χειρισμού reset μετά την εκκίνηση. Σε περίπτωση που επιθυμείται η χρησιμοποίηση διακοπών IRQ και FIQ, πρέπει να γραφτούν αντίστοιχες ρουτίνες χειρισμού, δηλαδή συναρτήσεις IrqHandler και FiqHandler.

Ένα απλό παράδειγμα εφαρμογής παρατίθεται παρακάτω:

```
extern void C_Entry(void);
#pragma arm section rwdata=
"pinakas"
int a[16];
#pragma arm section rwdata

#pragma arm section code
void C_Entry(void)
{
      int i;
      for(i=0;i<16;i++)
            a[i] = i;
}
#pragma arm section code
```

Η χρήση των οδηγιών *pragma* είναι απαραίτητη μόνο σε περιπτώσεις που θέλουμε η εφαρμογή χρήστη να περιέχεται στη ROM (δηλ. να μην έχουμε εκκίνηση από εξωτερική μνήμη). Αυτό γίνεται γιατί τα εγγράψιμα δεδομένα θα πρέπει να αντιστοιχηθούν στη RAM, που εφόσον δε γίνεται επαναχαρτογράφηση, θα βρίσκεται στην περιοχή 0x2xxxxxx.

Το παραπάνω πρόγραμμα εγγράφει έναν πίνακα 16 λέξεων. Σε περίπτωση μη επαναχαρτογράφησης, ο πίνακας θα καταλαμβάνει τις θέσεις μνήμης από 0x20000000 έως 0x2000003f. Σε περίπτωση επαναχαρτογράφησης θα καταλαμβάνει 64bytes στην περιοχή 0x0xxxxxxx.

Η διαδικασία μεταγλώττισης, σύνδεσης (linking) και παραγωγής αρχείων μνημών γίνεται αυτοματοποιημένα με τη χρήση makefiles.

4.5.2. Παραγωγή Αρχείου Αρχικοποίησης ROM

Η περίπτωση αυτή απευθύνεται σε περιπτώσεις που επιθυμείται να αλλάξει το πρόγραμμα εκκίνησης του συστήματος, ή σε περίπτωση που επιθυμείται και η εφαρμογή χρήστη να βρίσκεται στη ROM (δε χρησιμοποιείται εξωτερική μνήμη προγράμματος & επαναχαρτογράφηση).

Στην περίπτωση αυτή γίνεται χρήση των αρχείων armcore.s, centry.c και makefile. Το armcore.s είναι αρχείο assembly ARM, και περιέχει τις ρουτίνες χειρισμού των διακοπών του συστήματος, συμπεριλαμβανομένου και του *ResetHandler* που κάνει την εκκίνηση. Το αρχείο centry.c χρησιμοποιείται για τη διασύνδεση του προγράμματος εκκίνησης με την εφαρμογή χρήστη. Το αρχείο

αυτό έχει τη μορφή:

```
#ifdef TEST_DUMMY
      int C_Entry(void){ return
0;}
#endif
#ifdef TEST_3
#     include "test3.c"
#endif
#ifdef TEST_4
#     include "test4.c"
#endif
```

Αν πρόκειται να υπάρχει εκκίνηση από εξωτερική μνήμη, δεν χρειάζεται να γίνει σύνδεση με κάποιο entry_point. Στην περίπτωση αυτή θα δημιουργηθεί ξεχωριστά το αρχείο αρχικοποίησης της εξωτερικής μνήμης (επόμενη περίπτωση). Αν όμως επιθυμείται να υπάρχει και η εφαρμογή χρήστη στη ROM θα πρέπει να γίνει σύνδεση. Αν, για παράδειγμα, επιθυμείται να τοποθετηθεί στη ROM η εφαρμογή test3.c, που βρίσκεται στον ίδιο φάκελο με τα παραπάνω αρχεία, θα πρέπει να οριστεί η παράμετρος προεπεξεργαστή TEST_3.

Η διαδικασία αυτή παρουσιάζεται μελετώντας το περιεχόμενο του makefile:

```
all: bootrom.vhx
WHICHTEST=TEST_DUMMY
bootrom.vhx: bootrom.elf
      fromelf -vhx -32x1 -o bootrom.vhx bootrom.elf
      fromelf -c bootrom.elf > elf.txt
bootrom.elf: armcore.o centry.o

armlink -ro 0x0 -rw 0x20000000 -first 'armcore.o(asm$$$$code)' -o
bootrom.elf -map -info sizes,totals -list list.txt -errors err_link.txt
armcore.o centry.o

centry.o: centry.c
      armcc -bigend -c -errors err_comp.txt centry.c -D $(WHICHTEST)
armcore.o: armcore.s
      armcc -bigend -c -errors err_compa.txt armcore.s
clean:
      rm *.o err_*.txt list.txt elf.txt bootrom*.*
test_%:
      @touch centry.c
      @touch armcore.s
      make bootrom.vhx WHICHTEST=TEST_$*
```

Εκτελώντας την εντολή *make test_3* θα γίνει παραγωγή αρχείου μνήμης bootrom.vhx με μεταβλητή WHICHTEST=TEST_3. Αυτό σημαίνει ότι στη μεταγλώττιση του centry.c θα περαστεί η παράμετρος προεπεξεργαστή TEST_3. Δηλαδή θα συμπεριληφθεί η εφαρμογή χρήστη *test3.c*.

Επομένως αν επιθυμείται να τοποθετηθεί στη ROM μία νέα εφαρμογή χρήστη *newapplication.c*, τοποθετούνται τα αρχεία της εφαρμογής στον φάκελο που περιέχει το αρχείο *makefile* και armcore.s, έστω bootrom (η main της newapplication θα πρέπει να έχει το όνομα C_Entry (βλ. το παραπάνω

παράδειγμα εφαρμογής χρήστη)). Στη συνέχεια στον ίδιο φάκελο, προστίθονται στο αρχείο *centry.c* οι γραμμές:

```
#ifdef TEST_new
#     include
"newapplication.c
"
#endif
```

Τέλος, στον φάκελο bootrom εκτελούνται *make test_new* και εκτελούνται οι εντολές του makefile. Αν δεν υπάρχει κάποιο λάθος θα παραχθεί το αρχείο bootrom.vhx. Τα διάφορα λάθη και warnings των διαδικασιών μεταγλώττισης και σύνδεσης εγγράφονται στα αρχεία err_compa.txt, err_comp.txt και err_link.txt αντίστοιχα.

Για την παραγωγή του αρχείου αρχικοποίησης της μνήμης, λοιπόν, στην περίπτωση της εφαρμογής test3 το πρόγραμμα εκκίνησης armcore.s αρχικά περνάει από assembler (armcc –bigend –c –errors err_compa.txt armcore.s). Η παράμετρος –c σημαίνει χωρίς link (χρειάζεται γιατί υπάρχουν αναφορές σε entry points που δεν είναι διαθέσιμα), ενώ η παράμετρος –bigend σημαίνει ότι θα εξαχθεί κώδικας με λογική big endian. Τυχόν λάθη εξάγονται στο αρχείο err_compa.txt. Σε περίπτωση που δεν υπάρχει λάθος εξάγεται κώδικας στο αρχείο armcore.o.

Ακολουθεί η μεταγλώττιση της εφαρμογής C (armcc -bigend -c -errors err_comp.txt centry.c -D $(WHICHTEST)). Παράγεται και πάλι κώδικας, στο αρχείο *centry.o*.

Ακολουθεί η διαδικασία της σύνδεσης (linking), οπότε εκτελείται η εντολή

```
armlink -ro 0x0 -rw 0x20000000 -first 'armcore.o(asm$$$$code)' -o bootrom.elf
-map -info sizes,totals -list list.txt -errors err_link.txt armcore.o centry.o
```

Με την εντολή αυτή γίνεται η σύνδεση του αντικειμενικού κώδικα armcore.o με αυτόν του αρχείου centry.o. Ο armcore.o περιέχει εξωτερικές αναφορές σε C_Entry (και σε IrqHandler, FiqHandler προαιρετικά) που υπάρχουν στο centry.o . Έτσι η σύνδεση θα είναι επιτυχής και θα εξαχθεί συνολικό εκτελέσιμο σε μορφή elf (bootrom.elf). Να σημειωθεί ότι το κομμάτι readonly αντιστοιχίζεται στις διευθύνσεις 0x0xxxxxxx και το read/write κομμάτι στις διευθύνσεις 0x2xxxxxxx. Εκτός από το bootrom.elf, εξάγονται επίσης στατιστικά και λάθη στα αρχεία list.txt και err_link.txt αντίστοιχα.

Με το utility *fromelf* του ARM ADS εξάγονται αρχεία αρχικοποίησης μνημών σε περιγραφή Verilog. Για παράδειγμα η εντολή

```
fromelf -vhx -32x1 -o bootrom.vhx bootrom.elf
```

παράγει αρχείο κειμένου με τα περιεχόμενα της ROM μνήμης, με 4 bytes ανά γραμμή σε δεκαεξαδική μορφή. Το αρχείο αυτό κειμένου θα χρησιμοποιηθεί από

εντολή τύπου *$readmemh* στο Verilog μοντέλο της μνήμης ROM, ώστε αυτή να φορτωθεί με τα συγκεκριμένα περιεχόμενα.

Το εργαλείο *fromelf* μπορεί να χρησιμοποιηθεί και για την εξαγωγή των περιεχόμενων του elf αρχείου σε μορφή κειμένου (`fromelf -c bootrom.elf > elf.txt`). Το αρχείο κειμένου αυτό περιέχει διευθύνσεις και εντολές σε δεκαεξαδική και σε συμβολική μορφή, κάτι που σε συνδυασμό με τις κυματομορφές του προγράμματος ModelSim (διάδρομοι διευθύνσεων και δεδομένων) και τα περιεχόμενα μνημών του ModelSim, καθιστούν εφικτή την αποσφαλμάτωση του υλικού και λογισμικού του συστήματος.

Σύνοψη:

- Συγγραφή της εφαρμογής χρήστη *newapplication.c* στο φάκελο bootrom. Η main της εφαρμογής αυτής είναι η συνάρτηση C_Entry().
- Στο αρχείο bootrom/centry.c προστίθεται η εγγραφή
  ```
  #ifdef TEST_new
  #      include "newapplication.c"
  #endif
  ```
- Στον φάκελο bootrom εκτελείται `make test_new`
- Σε περίπτωση μη ύπαρξης λαθών προκύπτουν τα αρχεία *bootrom.vhx*, *elf.txt*, *list.txt*.

4.5.3. Χρήση Περιφερειακών Μονάδων μέσω Λογισμικού

Για την επικοινωνία με τα περιφερειακά χρησιμοποιείται η λογική της απεικόνισης μνήμης (memory map). Περισσότερες λεπτομέρειες δίνονται σε επόμενη ενότητα (*Διεπικοινωνία μέσω software*).

4.5.4. Διακοπές Συστήματος

Οι διακοπές που διαχειρίζεται το σύστημα είναι οι IRQ και FIQ. Η FIQ έχει τη μέγιστη προτεραιότητα. Οι διακοπές μπορεί να προέρχονται από κάποιες εξωτερικες εισόδους-εξόδους γενικού σκοπού (General Purpose I/O, GPIO), ή από εσωτερικά περιφερειακά του συστήματος όπως timers κα.

Ένας ελεγκτής διακοπών, που είναι προσκολλημένος στο APB bus, συνδέει όλες τις διακοπές του συστήματος (που προέρχονται είτε από εσωτερικά σήματα ή από το εξωτερικό περιβάλλον) στα δύο pins IRQ και FIQ του ARM.

Όταν προκληθεί διακοπή από κάποια πηγή (έστω IRQ), το σήμα IRQ του ARM πηγαίνει στη χαμηλή λογική στάθμη, και ο ARM ελέγχοντας τον πίνακά του (vector table) κάνει μεταφορά της εκτέλεσης (branch) στον IrqHandler (armcore.s) και μάλιστα σε κατάσταση λειτουργίας IRQ. Ο IrqHandler του armcore.s καλεί τη ρουτίνα IrqHandler που είναι ένα σημείο εκκίνησης σε C εφαρμογή. Εκεί θα είναι γραμμένη -από το χρήστη- η ρουτίνα εξυπηρέτησης

όλων των IRQ διακοπών.

Ένα παράδειγμα ρουτίνας εξυπηρέτησης IRQ δίνεται παρακάτω:

```
#define INTR_CTRL_BASE 0xf0000000
#define IRQ_STATUS 0
#define IRQ_CLEAR 1
#define IRQ_MASK_ENABLE 2
#define IRQ_MASK_CLEAR 3
void irq0_handler(void);
void irq1_handler(void);


void IrqHandler(void)
{
        int *ptr;
        int icidr, i;
        void (*irq_handler[16])(void);
        irq_handler[0]=&irq0_handler;
        irq_handler[1]=&irq1_handler;

        //identify the peripheral
        //by reading the interrupt controller id register (ICIDR)
        ptr= (int *)INTR_CTRL_BASE;
        icidr= *ptr;
        //ICIDR format: xxxx xxxx xxxx xxxx
        //bit i = 1 means IRQi
        //in case of more than one IRQ
        //give priority to smaller IRQs
        i=0;
        while(i<16)
        {
                if((icidr) & 0x01)
                        break;
                icidr=icidr >>1;
                i++;
        }
        if(i !=16 )
        {
                *(ptr+IRQ_CLEAR)=0x01<<i;
                irq_handler[i]();
        }
}

//IRQ handlers implementations
void irq0_handler(void)
{
        //IRQ0 action
}
void irq1_handler(void)
{
        //IRQ1 action
}
```

Πλαίσιο 4.1: Ρουτίνα εξυπηρέτησης IRQ για το MicroSoc

Ο ελεγκτής διακοπών περιέχει διάφορους καταχωρητές, εκ των οποίων άλλοι διαβάζονται και άλλοι γράφονται και η ανάγνωση/εγγραφή γίνεται μέσω AMBA APB διεπαφής. Η διεύθυνση βάσης είναι η *0xf0000000* για τους καταχωρητές αυτούς. Ο καταχωρητής IRQ_STATUS περιέχει τις διακοπές που εκκρεμούν. Το bit i αντιστοιχεί στο IRQi και συνολικά υπάρχουν 16 IRQs.

Ο καταχωρητής IRQ_CLEAR είναι εγγράψιμος και χρησιμοποιείται για να καθαριστεί μία εκκρεμούσα διακοπή. Η IRQi καθαρίζεται, αν γράψουμε τιμή με 1 στο bit i.

Οι καταχωρητές IRQ_MASK_ENABLE και IRQ_MASK_CLEAR χρησιμοποιούνται για το απενεργοποιήση συγκεκριμένων διακοπών. Με γράψιμο της τιμής 1 ή 0

στο bit i παρεμποδίζουμε ή ενεργοποιούμε ξανά την IRQi αντίστοιχα.

Η κύρια εργασία που εκτελεί η ρουτίνα εξυπηρέτησης -που παρουσιάστηκε προηγουμένως- είναι κάθε φορά που προκύψει διακοπή IRQ να ελέγχει τον IRQStatus για εκκρεμούσες διακοπές. Δίνοντας προτεραιότητα σε μικρότερες διακοπές IRQ (IRQ0>IRQ15) αποφασίζει ποια διακοπή θα εξυπηρετήσει και καλεί την αντίστοιχη ρουτίνα εξυπηρέτησης. Παράλληλα καθαρίζει τηνδιακοπή αυτή εγγράφοντας το αντίστοιχο bit στον IRQ_CLEAR. Να σημειωθεί ότι κατά την εκκίνηση του IrqHandler οι διακοπές βρίσκονται απενεργοποιημένες.

Στο σύστημα δικοπών του ARM δεν επιτρέπεται το φώλιασμα διακοπών. Ωστόσο, είναι δυνατό να γραφούν πολυπλοκότερα σχήματα εξυπηρέτησης διακοπών, που να επιτρέπουν διακοπή της εξυπηρέτησης κάποιας διακοπής, όταν φτάσει κάποια με μεγαλύτερη προτεραιότητα.

Παραπάνω παρουσιάστηκε το σύστημα διακοπών του απλού MicroSoc στη μορφή που προυπήρχε. Ωστόσο, στα πλαίσια της παρούσας διπλωματικής εργασίας το σύστημα διακοπών επεκτάθηκε με την προσθήκη ενός νέου ελεγκτή διακοπών συνδεδεμένου πάνω στον κύριο διάδρομο ASB του συστήματος. Περισσότερες λεπτομέρειες για αυτό θα δοθούν σε επόμενη ενότητα, κατά την παρουσίαση του επαναδιατάξιμου MicroSoc, του rMicroSoc.

4.5.5. Προσθήκη Υλικού στο ASB

Προκειμένου να προστεθεί ένα περιφερειακό στο ASB θα πρέπει να υπακούει στις προδιαγραφές του συστήματος AMBA ASB. Προς το παρόν το σύστημα υποστηρίζει μόνο "σκλάβους" (ο μόνος master είναι ο ARM), το οποίο σημαίνει ότι δεν μπορούν ακόμα να υποστηριχθούν DMA λειτουργίες.

4.5.6. Συγγραφή "Σκλάβου"

Αναλυτικές πληροφορίες για τη συγγραφή ενός "σκλάβου" για το διάδρομο AMBA ASB υπάρχουν σε προηγούμενη ενότητα, στη σχετική με τις προδιαγραφές του συγκεκριμένου συστήματος διαδρόμων.

Συνοπτικά, η επικοινωνία master/"σκλάβου" στο διάδρομο ASB είναι η εξής:
Στην αρχή κάποιου κύκλου (ή λίγο πιο μετά) ο ARM φορτώνει στον διάδρομο διευθύνσεων μία διεύθυνση, η οποία αντιστοιχεί σε έναν "σκλάβο". Ο αποκωδικοποιητής του ASB bus κάνει το αντίστοιχο σήμα επιλογής Dsel 1. Ο "σκλάβος" δειγματοληπτεί τη διεύθυνση και το σήμα επιλογής Dsel στην επόμενη αρνητική ακμή ρολογιού. Αν είναι έτοιμος να αποκριθεί, τότε χαμηλώνει το σήμα wait. Ο ARM θα διαβάσει το σήμα αυτό στην επόμενη θετική ακμή. Αν το δει 0 τότε σημαίνει ότι στην επόμενη αρνητική ακμή θα γίνει η

μεταφορά. Σε περίπτωση εγγραφής τα δεδομένα προς εγγραφή θα υπάρχουν σταθερά από τουλάχιστον λίγο πιο πριν από την ακμή και θα λατσαριστούν από τον "σκλάβο" στην αρνητική ακμή. Σε περίπτωση ανάγνωσης, ο "σκλάβος" θα πρέπει να βγάλει στο διάδρομο δεδομένων τα ζητούμενα δεδομένα λίγο πριν την αρνητική ακμή, έτσι ώστε ο ARM να τα διαβάσει στην αρνητική ακμή.

Σε περίπτωση που το περιφερειακό ενεργοποιήσει το σήμα WAIT, ειδοποιώντας τον ARM ότι δε μπορεί να αποκριθεί, τότε αυτόματα η μεταφορά αναβάλλεται για τον μεθεπόμενη αρνητική ακμή, εκτός κι αν προκύψει νέα αναβολή.

Εδώ πρέπει να σημειωθεί ότι τα σήματα addr, Dsel, write από το master μπορεί να διατηρηθούν μόνο στον πρώτο κύκλο που επιλέγεται το περιφερειακό και τους υπόλοιπους να περιέχουν άλλες τιμές (ο διάδρομος είναι pipelined). Γι αυτό πρέπει να μανδαλωθούν από τον "σκλάβο" αμέσως.

Ο "σκλάβος" αντιλαμβάνεται μόνο θετικές και αρνητικές ακμές ρολογιού και σ' αυτές τις χρονικές στιγμές δειγματοληπτεί.

4.5.7. Προσκόλληση στο Διάδρομο ASB

Μετά τη συγγραφή του "σκλάβου" με την ASB διεπαφή, αρκεί να του αποδοθεί διεύθυνση απεικόνισης στον αποκωδικοποιητή του συστήματος, να προστεθεί στο σύστημα μέσω του αρχείου MicrosocChip.v και να συνδεθεί κατάλληλα στον πολυπλέκτη ASB. Μέσα στο παραπάνω αρχείο γίνονται οι κατάλληλες συνδέσεις του στιγμιότυπου του "σκλάβου" με τα σήματα του συστήματος. Για παράδειγμα, αν πρόκειται να προστεθεί ο "σκλάβος" που παρουσιάστηκε προηγουμένως, τότε μέσα στο αρχείο MicrosocChip.v θα πρέπει να γίνει η αρχικοποίηση :

```
SimpleSlave JustInstantiated
(
        .resetn (resetn),
        .clk    (clk),
        .clkn   (clkn),
        .addr   (addr),
        .dsel   (Dsel_11),
        .bwrite (write),
        .bwait  (b_waitS11),
        .bdin   (b_s11in),
        .bdout   (b_s11out)
);
```

Επίσης ο "σκλάβος" μπορεί να πάρει δικό του ρολόι από τη γεννήτρια ρολογιού (clock generator) και δικό του σήμα RESETN από τη γεννήτρια επαναφοράς (reset generator). Ο clock generator είναι προγραμματιζόμενος με ASB διεπαφή (έχει προγραμματιζόμενη διαίρεση συχνότητας & προγραμματιζόμενο clock gater). Ο reset generator έχει κι αυτός ASB διεπαφή, πράγμα που επιτρέπει στον ARM να κάνει reset επιλεκτικά σε περιφερειακά.

4.5.8. Διεπικοινωνία μέσω Λογισμικού

Μετά την προσκόλληση του ASB "σκλάβου" στο διάδρομο ASB, αυτός μπορεί πλέον να χρησιμοποιηθεί μέσω λογισμικού.

Συγκεκριμένα, ελέγχεται αρχικά ο κώδικας Verilog *ASBDec.v* ώστε να βρεθεί σε ποια διεύθυνση αντιστοιχεί το Dsel_i (στο παράδειγμα που αναφέρθηκε προηγουμένως το Dsel_11), στο οποίο έχει αντιστοιχηθεί ο νέος "σκλάβος". Στη συγκεκριμένη περίπτωση, το σήμα αυτό έστω ότι αντιστοιχεί στις διευθύνσεις 0x9xxxxxxx. Επομένως στο λογισμικό, για να γίνει χρήση του νέου "σκλάβου" αρκεί η εγγραφή και ανάγνωση από το συγκεκριμένο χώρο διευθύνσεων. Αυτό πρέπει να γίνεται με γνώση της εσωτερικής αποκωδικοποίησης του "σκλάβου". Στο παράδειγμα που εξετάζεται τα bit [4:2] καθορίζουν 1 από 8 καταχωρητές.

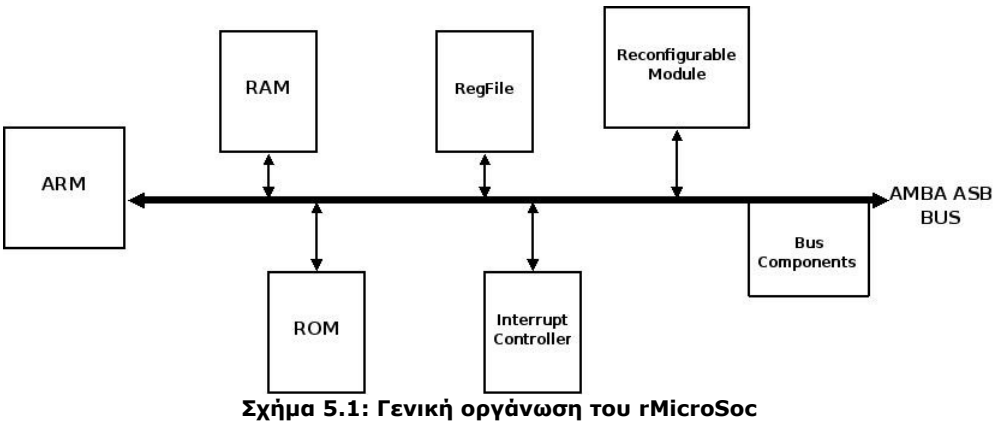
Ακολουθεί ένα παράδειγμα επικοινωνίας με τον παραπάνω "σκλάβο".

```
extern void C_Entry(void);
void C_Entry(void)
{
        int i;
        volatile int * peripheralptr;
        peripheralptr=(int *) 0x90000000;
        for(i=0;i<7;i++)
                *(peripheralptr+i)=i;

*(peripheralptr+1)=*(peripheralptr+2);
}
```

ΚΕΦΑΛΑΙΟ 5
Το Επαναδιατάξιμο Σύστημα MicroSoc

5.1. Μοντέλο του Συστήματος

Το Σχήμα 5.1 παρουσιάζει τη βασική οργάνωση του επαναδιατάξιμου υπολογιστικού συστήματος MicroSoc (του rMicroSoc), στην τελική μορφή που χρησιμοποιήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας. Όπως φαίνεται, αποτελείται από έναν επεξεργαστή ελέγχου, ένα σύστημα κεντρικών μνημών (που απαρτίζεται από μία ROM, μία RAM και ένα RegisterFile), τις απαραίτητες μονάδες για υλοποίηση του πρωτοκόλλου επικοινωνίας AMBA ASB, έναν ελεγκτή διακοπών συνδεδεμένο στο διάδρομο ASB και ένα περιφερειακό επαναδιατάξιμης λογικής. Πρόκειται για ένα SoC με βαθμό επαναδιάταξης επιπέδου λέξης (με χειριστή δεδομένων πλάτους 16 bits), με τον κύριο έλεγχο να έχει δοθεί στον επεξεργαστή που βρίσκεται πάνω στο ολοκληρωμένο. Το επαναδιατάξιμο κομμάτι του αποτελείται από έναν πίνακα διαστάσεων 4x4 με επαναδιατάξιμα κελιά, που προγραμματίζεται μέσω πολλαπλών λέξεων διαμόρφωσης και λειτουργεί σύμφωνα με το μοντέλο υπολογισμού SIMD. Αυτός ο επαναδιατάξιμος πίνακας αποτελεί -ιεραρχικά- ένα αυτόνομο περιφερειακό του κεντρικού συστήματος. Στις ενότητες που ακολουθούν δίνονται λεπτομέρειες για την αρχιτεκτονική και το μοντέλο λειτουργίας του συστήματος.



Σχήμα 5.1: Γενική οργάνωση του rMicroSoc

5.1.1. Επισκόπηση του Συστήματος

Οι τρεις βασικότερες μονάδες του συστήματος είναι ο κεντρικός επεξεργαστής ελέγχου, το επαναδιατάξιμο περιφερειακό και ο διάδρομος επικοινωνίας και

διασύνδεσης AMBA ASB.

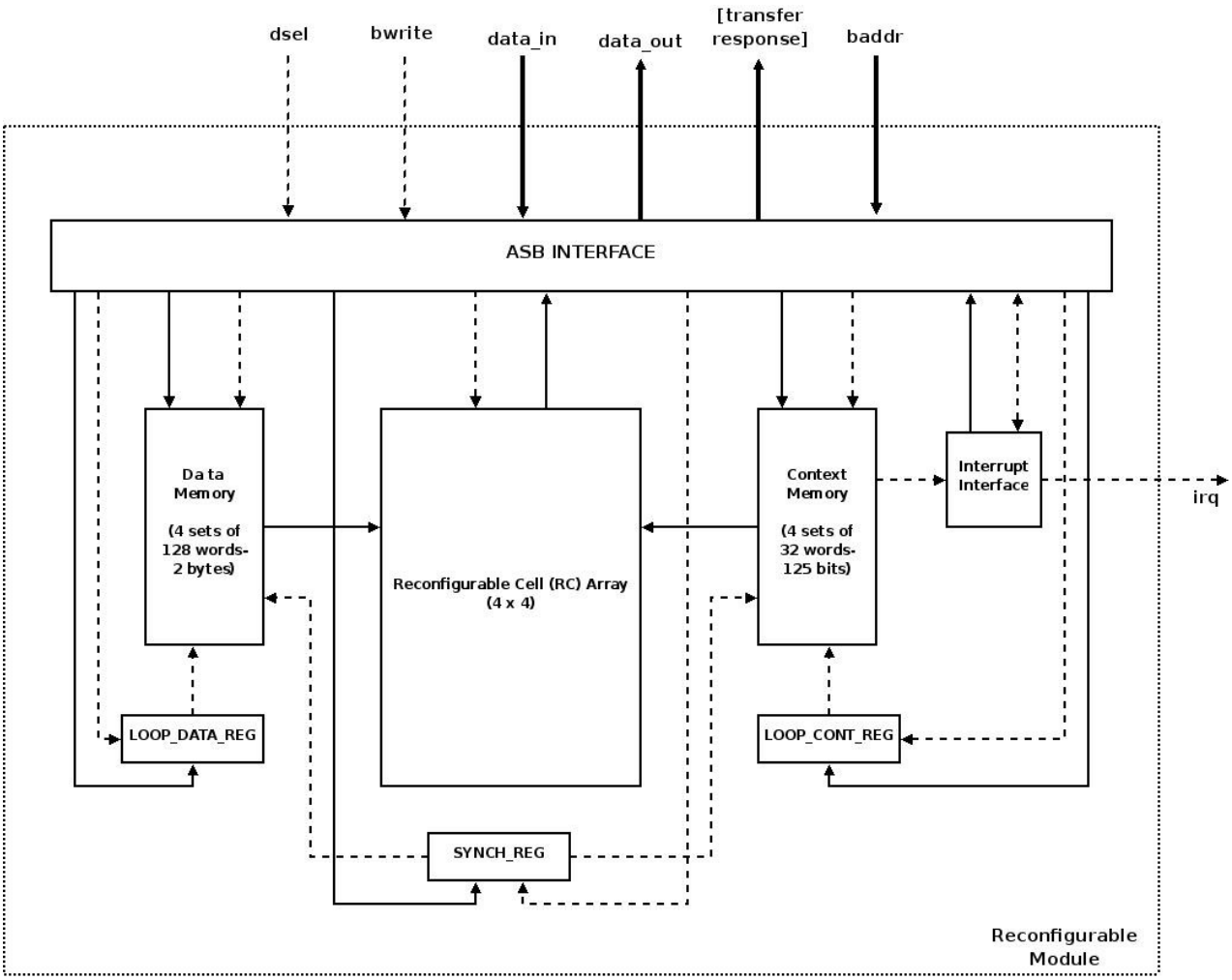
Επεξεργαστής Ελέγχου : Πρόκειται για τη μονάδα ελέγχου ολόκληρου του συστήματος και είναι ένας πυρήνας επεξεργαστή αρχιτεκτονικής ARM (RISC) 32-bit, ίδιος με αυτόν που χρησιμοποιούνταν και στο απλό σύστημα MicroSoc πριν την επέκτασή του. Ο επεξεργαστής αυτός ελέγχει τη λειτουργία του επαναδιατάξιμου περιφερειακού και -γενικά- ολόκληρου του συστήματος, όπως επίσης και τις μεταφορές δεδομένων από και προς το περιφερειακό. Σε αντίθεση με το MorphoSys, δεν έχουν προστεθεί επιπλέον εντολές στο σύνολο εντολών του ARM. Θεωρήθηκε πιο αναγκαίο -σε πρώτη φάση- να επικεντρωθεί η εργασία στη σχεδίαση και στην ανάπτυξη του επαναδιατάξιμου κομματιού του συστήματος. Η επικοινωνία μεταξύ του ARM και του περιφερειακού -στα πλαίσια του προγραμματισμού- γίνεται με χρήση της γλώσσας C.

Διάδρομος AMBA ASB : Το σύστημα διαδρόμου AMBA ASB αποτελεί το στοιχείο επικοινωνίας του κεντρικού επεξεργαστή ARM με τα υπόλοιπα στοιχεία του συστήματος, συνεπώς και με το επαναδιατάξιμο περιφερειακό. Υποστηρίζεται η λειτουργία μοναδικού "κύριου" (master) διαδρόμου, το ρόλο του οποίου έχει αναλάβει ο επεξεργαστής ελέγχου ARM. Το επαναδιατάξιμο περιφερειακό αποτελεί ουσιαστικά μια μονάδα "σκλάβου" του διαδρόμου ASB, η οποία βρίσκεται συνδεδεμένη πάνω στα συστήματα του διαδρόμου. Πληροφορίες για το συγκεκριμένο διάδρομο, όπως και για τους χρονισμούς και τα σήματά του έχουν δοθεί αναλυτικά σε προηγούμενες ενότητες.

Επαναδιατάξιμο Περιφερειακό : Η βασική μονάδα του συστήματος είναι το επαναδιατάξιμο περιφερειακό (Σχήμα 5.2). Η βάση του περιφερειακού είναι ο επαναδιατάξιμος πίνακας (*Reconfigurable Cell Array*), που αποτελείται από 16 όμοιες μονάδες επεξεργασίας, τα επαναδιατάξιμα κελιά, διατεταγμένα ως ένας 4x4 πίνακας. Κάθε κελί έχει εσωτερικά μία μονάδα ALU/πολλαπλασιαστή και ένα αρχείο καταχωρητών (δεδομένα πλάτους 16-bit).

Η λειτουργία και το δίκτυο διασυνδέσεων μεταξύ των κελιών του επαναδιατάξιμου πίνακα καθορίζονται από ειδικές λέξεις διαμόρφωσης, με μέγεθος 31 bits. Οι λέξεις αυτές βρίσκονται αποθηκευμένες σε 4 μπλοκ μιας μνήμης, της μνήμης διαμόρφωσης (*Context Memory*), που αποτελεί μια ακόμη μονάδα του περιφερειακού. Κάθε μπλοκ έχει τη δυνατότητα να αποθηκεύσει συνολικά 128 διαφορετικά περιεχόμενα διαμόρφωσης για τον πίνακα. Τα δεδομένα προς επεξεργασία στους υπολογισμούς που εκτελούνται στα κελιά του πίνακα προέρχονται από τη μνήμη δεδομένων (*Data Memory*), που βρίσκεται ιεραρχικά πάνω στο επαναδιατάξιμο περιφερειακό. Τα δεδομένα της μνήμης αυτής είναι οργανωμένα σε 4 σύνολα (sets). Κάθε σύνολο έχει χωρητικότητα 128 θέσεις μνήμης των 2 bytes και περιέχει δεδομένα που προορίζονται για μία συγκεκριμένη γραμμή (στήλη) του πίνακα. Υπάρχουν ακόμη κάποιοι καταχωρητές για την υλοποίηση εσωτερικών βρόχων επανάληψης, τόσο στη μνήμη διαμόρφωσης (*LOOP_CONT_REG*), όσο και στη μνήμη δεδομένων (*LOOP_DATA_REG*), ένας καταχωρητής για τη συγχρονισμένη έναρξη της

λειτουργίας των δύο μνημών (*SYNCH_REG*) και μια μονάδα υπεύθυνη για τις διακοπές του περιφερειακού προς τον ARM (*Interrupt Interface*). Τέλος, υπάρχουν κυκλώματα διεπαφής που χρησιμοποιούνται για τη σύνδεσή του περιφερειακού με το υπόλοιπο σύστημα (*ASB_INTERFACE*).



Σχήμα 5.2: Εσωτερική δομή του επαναδιατάξιμου περιφερειακού

5.1.2. Ροή Εκτέλεσης

Το επαναδιατάξιμο MicroSoc λειτουργεί ως ακολούθως: Ο κεντρικός επεξεργαστής ARM του συστήματος φορτώνει αρχικά τα δεδομένα διαμόρφωσης από την κεντρική μνήμη του συστήματος στη μνήμη διαμόρφωσης του περιφερειακού. Στη συνέχεια φορτώνεται η μνήμη δεδομένων, επίσης με δεδομένα από την κεντρική μνήμη. Στο σημείο αυτό πλέον και οι δύο μνήμες του περιφερειακού περιέχουν το κατάλληλο περιεχόμενο για τους υπολογισμούς που ακολουθούν.

Έπειτα, θα πρέπει μέσω της εφαρμογής που έχει γραφτεί από τον προγραμματιστή (η οποία είναι σε γλώσσα προγραμματισμού C) να δοθούν

ορισμένες οδηγίες προς το περιφερειακό, απαραίτητες για τη λειτουργία του. Πρώτον, να καθοριστεί ποιο ακριβώς σύνολο της μνήμης διαμόρφωσης πρέπει να εκτελεστεί και αντιστοίχως από ποια διεύθυνση της μνήμης δεδομένων πρέπει να αρχίσουν να μεταφέρονται δεδομένα για επεξεργασία προς τον επαναδιατάξιμο πίνακα. Δεύτερον, πρέπει να αποθηκευτεί εσωτερικά στο περιφερειακού ποιο σύνολο της μνήμης θα εκτελεστεί. Αυτό γίνεται διότι όταν ζητηθεί διακοπή από τον ARM αυτός πρέπει να μπορεί να ενημερωθεί για το σύνολο που εκτελείται εκείνη τη στιγμή. Τρίτον, πρέπει να καθοριστεί ο αριθμός των εσωτερικών βρόχων που θα εκτελεστούν -αν αυτό είναι απαραίτητο- στη μνήμη διαμόρφωσης ή/και στη μνήμη δεδομένων. Τέλος, πρέπει να συγχρονιστούν οι δύο μνήμες ώστε να αρχίσουν τη λειτουργία τους ταυτόχρονα, καθότι μεταξύ των ρυθμίσεων των δύο αρχικών διευθύνσεων ανάγνωσης για τις μνήμες μεσολαβούν κάποιοι κύκλοι ρολογιού που θα δρούσαν διαφορετικά ως αποσυγχρονιστικός παράγοντας.

Αφού γίνουν όλα τα παραπάνω βήματα, το επαναδιατάξιμο περιφερειακό ξεκινάει να εκτελεί τους υπολογισμούς που πρέπει, σύμφωνα με τις λέξεις διαμόρφωσης που διαβάζει από τη μνήμη διαμόρφωσης. Όταν το προκαθορισμένο περιεχόμενο διαμόρφωσης εκτελεστεί, θα πρέπει το περιφερειακό να σταματήσει προσωρινά τη λειτουργία του και να κάνει αίτηση για διακοπή στον επεξεργαστή ελέγχου ARM. Όταν ο επεξεργαστής αποφασίσει να την εξυπηρετήσει, εκτελεί την κατάλληλη ρουτίνα εξυπηρέτησης, η οποία συνήθως είναι διαφορετική, ανάλογα με ποιο σύνολο της μνήμης διαμόρφωσης εκτελέστηκε. Μέσα στη ρουτίνα εξυπηρέτησης γίνεται ανάγνωση από τα τελικά αποτελέσματα των υπολογισμών (τα οποία βρίσκονται αποθηκευμένα στους τοπικούς καταχωρητές των κελιών του επαναδιατάξιμου πίνακα) και στη συνέχεια υπάρχουν δύο επιλογές. Είτε η συνολική εφαρμογή τερματίζεται, είτε ακολουθείται η ίδια διαδικασία από την αρχή, ώστε να εκτελεστεί κάποιο άλλο σύνολο της μνήμης διαμόρφωσης ή το ίδιο αλλά πιθανώς με διαφορετικά δεδομένα προς επεξεργασία.

5.1.3. Χαρακτηριστικά του Συστήματος

Το rMicroSoC που σχεδιάστηκε βασίζεται σε δύο μοντέλα, στο επαναδιατάξιμο σύστημα MorphoSys και στο μοντέλο υπολογισμού SIMD, οπότε πολλά στοιχεία της σχεδίασής του και πολλά από τα χαρακτηριστικά του απορρέουν από τα μοντέλα αυτά.

Η λειτουργία του επαναδιατάξιμου πίνακα και οι υπολογισμοί που θα εκτελεστούν σε αυτόν καθορίζονται από τις λέξεις διαμόρφωσης, οι οποίες μεταφέρονται από τη μνήμη διαμόρφωσης και αποθηκεύονται προσωρινά σε κατάλληλο καταχωρητή στα κελιά του πίνακα. Η μεταφορά γίνεται είτε σε λειτουργία ανά στήλες είτε ανά γραμμές, το οποίο σημαίνει ότι όλα τα κελιά που βρίσκονται στην ίδια γραμμή (στήλη) μοιράζονται την ίδια λέξη διαμόρφωσης, άρα εκτελούν τους ίδιους υπολογισμούς. Ωστόσο, κάθε κελί χρησιμοποιεί

διαφορετικά δεδομένα προς επεξεργασία. Εξάλλου ο τρόπος αυτός λειτουργίας υπαγορεύεται από το μοντέλο υπολογισμών SIMD, σύμφωνα με το οποίο έχει σχεδιαστεί ο επαναδιατάξιμος πίνακας. Πρέπει να σημειωθεί πάντως ότι πολλές φορές τα κελιά σε μια γραμμή ή στήλη χρησιμοποιούν τα ίδια δεδομένα προς επεξεργασία, αφού η μνήμη δεδομένων μεταφέρει στον πίνακα τα περιεχόμενά της ανά γραμμή (ή ανά στήλη). Παρόλα αυτά, εν γένει, κάθε κελί -μπορεί να- επεξεργάζεται διαφορετικά δεδομένα. Ο συγκεκριμένος τρόπος φόρτωσης των λέξεων διαμόρφωσης στον πίνακα αποδεικνύεται αποτελεσματικός και αποδοτικός σε περιπτώσεις εφαρμογών που απαιτούν μεγάλη παραλληλία δεδομένων, όπως για παράδειγμα οι εφαρμογές επεξεργασίας εικόνας και βίντεο.

Παρακάτω παρατίθενται τα σημαντικότερα χαρακτηριστικά του επαναδιατάξιμου MicroSoc.

- *Βαθμός επαναδιάταξης επιπέδου λέξης* : Το rMicroSoc έχει σχεδιαστεί να λειτουργεί με δεδομένα των 16 bits, το οποίο έχει αποδειχθεί αποδοτικό για εφαρμογές επεξεργασίας εικόνας/βίντεο. Επιπλέον, τα περιεχόμενα διαμόρφωσης μοιράζονται στον επαναδιατάξιμο πίνακα υπό μορφή λέξεων διαμόρφωσης των 31 bits η κάθε μία. Μάλιστα, κάθε λέξη διαμόρφωσης προορίζεται να καθορίσει τη λειτουργία μιας ομάδας επαναδιατάξιμων κελιών (είτε σε μια γραμμή είτε σε μια στήλη). Η λέξη αυτή καθορίζει ποια ακριβώς λειτουργία θα υλοποιηθεί στα κελιά και παρέχει bits ελέγχου για τα διάφορα κυκλώματα που υπάρχουν, τόσο εσωτερικά στο κελί όσο και στο υπόλοιπο το σύστημα.

- *Μεγάλο βάθος δυνατότητας προγραμματισμού* : Η μνήμη διαμόρφωσης μπορεί να αποθηκεύσει μέχρι 4 ανεξάρτητες ενότητες διαμόρφωσης, με κάθε ενότητα να αποτελείται -το μέγιστο- από 128 περιεχόμενα διαμόρφωσης. Η μετάδοση μπορεί να γίνει είτε ανά γραμμές είτε ανά στήλες.

5.2. Αρχιτεκτονική Επαναδιατάξιμου Περιφερειακού

5.2.1. Γενικά

Στην ενότητα αυτή αναλύεται η οργάνωση και η εσωτερική αρχιτεκτονική της επαναδιατάξιμης περιφερειακής μονάδας. Αρχικά δίνεται μια συνοπτική περιγραφή των γενικών χαρακτηριστικών της και των σημάτων εισόδου/εξόδου, τα οποία τη συνδέουν με το υπόλοιπο σύστημα. Ακολουθεί η αναλυτική παρουσίαση καθεμιάς από τις αυτόνομες μονάδες που απαρτίζουν το περιφερειακό, δηλαδή του επαναδιατάξιμου πίνακα, του συστήματος μνημών (με τις μνήμες διαμόρφωσης και δεδομένων μαζί με τα απαραίτητα περιφερειακά

κυκλώματά τους), του συστήματος διακοπών και της διεπαφής σύνδεσης του περιφερειακού ως "σκλάβου" πάνω στο διάδρομο συστήματος AMBA ASB.

5.2.2. Σήματα Εισόδου/Εξόδου Περιφερειακού

Στο Σχήμα 5.3 διακρίνονται οι θύρες εισόδου και εξόδου του επαναδιατάξιμου περιφερειακού. Οι θύρες αυτές χρησιμεύουν στην επικοινωνία της μονάδας από και προς το σύστημα διαδρόμων AMBA ASB που αποτελεί τον κεντρικό διάδρομο του συστήματος.

Αναλυτικά, οι θύρες εισόδου για τη μονάδα του περιφερειακού είναι :

- *databus_in* – Η είσοδος αυτή έχει μέγεθος 32 bits και αποτελεί το σήμα δεδομένων εισόδου. Συνδέεται απευθείας με το διάδρομο εγγραφής δεδομένων του συστήματος ASB. Χρησιμοποιείται για μεταφορά δεδομένων από το master προς το επαναδιατάξιμο περιφερειακό.

- *baddr* – Η συγκεκριμένη είσοδος είναι η είσοδος διευθύνσεων και έχει μέγεθος 32 bits. Συνδέεται απευθείας με το διάδρομο διευθύνσεων του συστήματος ASB και ως εκ τούτου χρησιμοποιείται για τη μεταφορά διευθύνσεων ανάγνωσης/εγγραφής από το master προς το περιφερειακό.

- *bwrite* – Σήμα ελέγχου μεγέθους 1 bit που καθορίζει αν μια μεταφορά είναι μεταφορά εγγραφής ή ανάγνωσης, ανάλογα με το αν πάρει την τιμή 1 ή 0 αντίστοιχα. Συνδέεται με το αντίστοιχο σήμα BWRITE του συστήματος ASB και χρησιμοποιείται από το master για να ορίσει στο περιφερειακό το είδος μιας μεταφοράς.

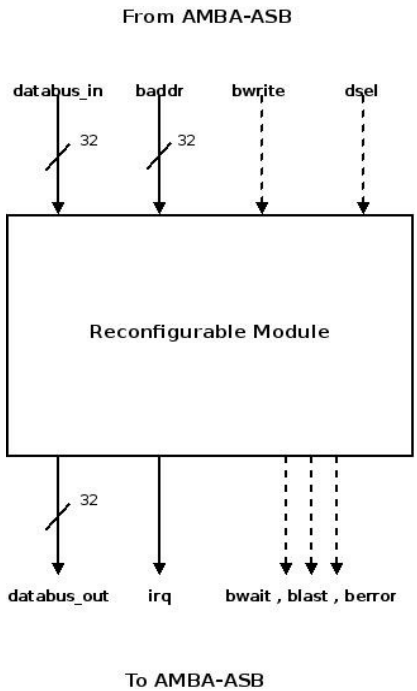
- *dsel* – Το σήμα αυτό επιλογής λειτουργίας του περιφερειακού έχει μέγεθος 1 bit και παρέχεται στο περιφερειακό από τον αποκωδικοποιητή του διαδρόμου ASB. Όταν -κατά τη διάρκεια μιας μεταφοράς- το σήμα αυτό είναι ενεργό, τότε το επαναδιατάξιμο περιφερειακό πρέπει να συμμετάσχει στην εν λόγω μεταφορά. Σε αντίθετη περίπτωση μένει ανενεργό, καθώς η μεταφορά δε σχετίζεται με αυτό.

Οι θύρες εξόδου της μονάδας είναι :

- *databus_out* – Η θύρα αυτή εξόδου έχει μέγεθος 32 bits και αποτελεί το σήμα εξόδου δεδομένων. Συνδέεται απευθείας με το διάδρομο ανάγνωσης δεδομένων του συστήματος ASB και χρησιμοποιείται για μεταφορά δεδομένων από το επαναδιατάξιμο περιφερειακό προς το master, κατά τη διάρκεια μιας μεταφοράς ανάγνωσης.

- *bwait, blast, berror* – Οι έξοδοι αυτές, με μέγεθος 1 bit η κάθε μία, χρησιμοποιούνται συνδυασμένα για να παρέχει το περιφερειακό την απαραίτητη απόκριση στις μεταφορές που εκκινεί ο master προς αυτό.

Υπενθυμίζεται ότι με τον κατάλληλο συνδυασμό των τιμών των σημάτων αυτών μπορεί το περιφερειακό να ζητήσει επιπλέον κύκλους για μία μεταφορά από το master και να δηλώσει ότι μια μεταφορά έχει ολοκληρωθεί με επιτυχία. Εσωτερικά, τα σήματα αυτά οδηγούνται από τη διεπαφή ASB (ASB_Interface) η οποία και είναι υπεύθυνη για το χρονισμό και την πραγματοποίηση της απόκρισης μεταφοράς.



Σχήμα 5.3: Γενικό διάγραμμα του επαναδιατάξιμου περιφερειακού

- *irq* – Η έξοδος αυτή έχει μέγεθος 1 bit και χρησιμοποιείται από το περιφερειακό για να δηλώσει στον ARM ότι ζητάει διακοπή. Ειδικότερα συνδέεται με μια είσοδο πηγής αιτήσεων διακοπής της μονάδας του ελεγκτή διακοπών ASB, που βρίσκεται στο κυρίως σύστημα. Όταν πάρει την τιμή 1, ο ελεγκτής διακοπών αποθηκεύει στον εσωτερικό καταχωρητή κατάστασης ότι το περιφερειακό αυτό έχει μια αναμένουσα αίτηση διακοπής. Ο ελεγκτής με τη σειρά του θα προωθήσει το αίτημα στον ARM.

5.2.3. Επαναδιατάξιμο Κελί

5.2.3.1. Γενικά

Το επαναδιατάξιμο κελί αποτελεί τη πιο σημαντική δομική μονάδα του επαναδιατάξιμου συστήματος και είναι ουσιαστικά ο πυρήνας όλων των υπολογισμών που εκτελούνται. Μπορεί να παρομοιαστεί με έναν μικροεπεξεργαστή με διοχέτευση δύο επιπέδων. Δέχεται δεδομένα προς

επεξεργασία από την εξωτερική μνήμη δεδομένων και πληροφορίες για τον προγραμματισμό του, μέσω της λέξης διαμόρφωσης. Αφού εκτελέσει τους κατάλληλους υπολογισμούς πάνω στα δεδομένα αυτά -σύμφωνα με τα δεδομένα διαμόρφωσης, προκύπτει κάποιο αποτέλεσμα το οποίο προωθείται στην έξοδο του κελιού. Υπάρχει η δυνατότητα να αποθηκευτεί το αποτέλεσμα αυτό στους τοπικούς καταχωρητές του κελιού.

Κάθε κελί δέχεται μία είσοδο δεδομένων από τη μνήμη δεδομένων του συστήματος, εισόδους δεδομένων από τις εξόδους άλλων επαναδιατάξιμων κελιών και μία είσοδο δεδομένων διαμόρφωσης από τη μνήμη διαμόρφωσης (context memory). Η έξοδος του κελιού είναι αποτέλεσμα των υπολογισμών που εκτελούνται εσωτερικά σε αυτό.

Το Σχήμα 5.4 στην επόμενη σελίδα παρουσιάζει ένα γενικό διάγραμμα του επαναδιατάξιμου κελιού με την παρουσίαση των σημάτων εισόδου/εξόδου, χωρίς περισσότερες λεπτομέρειες για την εσωτερική του οργάνωση.

Η εσωτερική αρχιτεκτονική του επαναδιατάξιμου κελιού βασίζεται -σε γενικές γραμμές- στο μοντέλο που περιγράφηκε από την αρχιτεκτονική του MorphoSys. Παρακάτω φαίνονται τα συστατικά που απαρτίζουν εσωτερικά ένα επαναδιατάξιμο κελί του συστήματος:

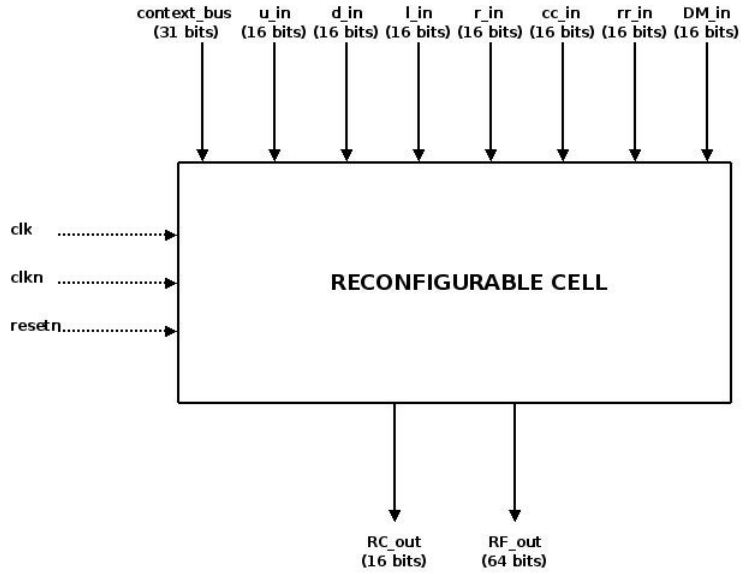
- Μια μονάδα ALU για υπολογισμούς σταθερής υποδιαστολής, με μέγεθος λέξης 16 bit
- Μια μονάδα πολλαπλασιαστή τύπου CSA, με μέγεθος 16x16
- Μια μονάδα ολίσθησης με μήκος λέξης 32bits και δυνατότητα ολίσθησης από 0 μέχρι 32 bits
- Ένας καταχωρητής εξόδου με μήκος 32bits
- Ένα αρχείο καταχωρητών, που αποτελείται από 4 καταχωρητές με μήκος 16bits ο κάθε ένας
- Δύο πολυπλέκτες εισόδου 16-σε-1, που χειρίζονται δεδομένα των 16bits
- Ένας καταχωρητής διαμόρφωσης (Context Register) μεγέθους 31 bits

Τα παραπάνω συστατικά στοιχεία του κελιού έχουν προκύψει από τις ακριβείς λειτουργίες και τα χαρακτηριστικά που επιλέχθηκαν να υποστηρίζονται από ένα κελί. Πρόκειται για κλασσικές λειτουργίες ενός απλού μοντέλου μικροεπεξεργαστή, όπως αριθμητικοί και λογικοί υπολογισμοί. Πιο αναλυτικά αναφέρονται παρακάτω τα βασικά χαρακτηριστικά του επαναδιατάξιμου κελιού.

- Υποστηρίζονται οι κλασσικοί αριθμητικοί υπολογισμοί, δηλαδή η πρόσθεση, η αφαίρεση και ο πολλαπλασιασμός μεταξύ δύο δεδομένων. Ακόμη η αριθμητική ολίσθηση, τόσο προς τα αριστερά όσο και προς τα δεξιά.

- Υποστηρίζεται η λογική πράξη AND και η λογική πράξη OR, αλλά και η λογική σύγκριση μεταξύ δύο δεδομένων.

- Πρέπει να υπάρχει η δυνατότητα ώστε τα δεδομένα εισόδου να διέρχονται μέσα από όλες τις μονάδες υπολογισμού μέχρι την έξοδο του κελιού χωρίς τροποποίηση.



context_bus : Σήμα εισόδου που ενώνει το κελί με τη μνήμη διαμόρφωσης και παρέχει στο κελί τις λέξεις διαμόρφωσης
u_in : Σήμα εισόδου δεδομένων συνδεδεμένο με την έξοδο του από πάνω κελιού
d_in : Σήμα εισόδου δεδομένων συνδεδεμένο με την έξοδο του από κάτω κελιού
l_in : Σήμα εισόδου δεδομένων συνδεδεμένο με την έξοδο του από αριστερά κελιού
r_in : Σήμα εισόδου δεδομένων συνδεδεμένο με την έξοδο του από δεξιά κελιού
cc_in : Σήμα εισόδου δεδομένων συνδεδεμένο με την έξοδο του κελιού που βρίσκεται στην ίδια στήλη αλλά όχι δίπλα
ll_in : Σήμα εισόδου δεδομένων συνδεδεμένο με την έξοδο του κελιού που βρίσκεται στην ίδια γραμμή αλλά όχι δίπλα
DM_in : Σήμα εισόδου δεδομένων συνδεδεμένο με τη μνήμη δεδομένων

Σχήμα 5.4: Γενικό διάγραμμα του επαναδιατάξιμου κελιού

- Πολλές φορές είναι χρήσιμο (ή και αναγκαίο) κατά τον προγραμματισμό με το επαναδιατάξιμο σύστημα το κελί να βρίσκεται σε κατάσταση stall, δηλαδή να μην εκτελείται κανένας υπολογισμός και να μην είναι ενεργό. Έτσι υπάρχει η δυνατότητα να τίθεται το κελί σε ανενεργή κατάσταση και όλες οι λειτουργίες του να αναστέλλονται. Να σημειωθεί ότι όλοι οι υπολογισμοί πραγματοποιούνται όταν το κελί βρίσκεται σε ενεργή κατάσταση.

- Πρέπει να υπάρχει μια τοπική εσωτερική μνήμη μέσα σε κάθε κελί, η οποία και υλοποιείται από ένα αρχείο καταχωρητών. Η μνήμη αυτή χρησιμοποιείται για αποθήκευση των αποτελεσμάτων των υπολογισμών που εκτελούνται στο κελί και αποτελεί τη μόνη μνήμη στην οποία μπορεί το κελί να γράψει.

- Όλες οι παραπάνω λειτουργίες πρέπει να καθορίζονται από την εξωτερική μνήμη περιεχομένου (context memory).

Παρακάτω γίνεται αναλυτική περιγραφή της εσωτερικής οργάνωσης ενός επαναδιατάξιμου κελιού και των συστατικών μονάδων που το απαρτίζουν. Στη συνέχεια, μέσα από την αναλυτική παρουσίαση των μονάδων, προκύπτουν και

τα ακριβή bits της λέξης διαμόρφωσης.

Το Σχήμα 5.5 δείχνει το αναλυτικό διάγραμμα της βασικής οργάνωσης ενός επαναδιατάξιμου κελιού.

5.2.3.2. Πολυπλέκτες Εισόδου

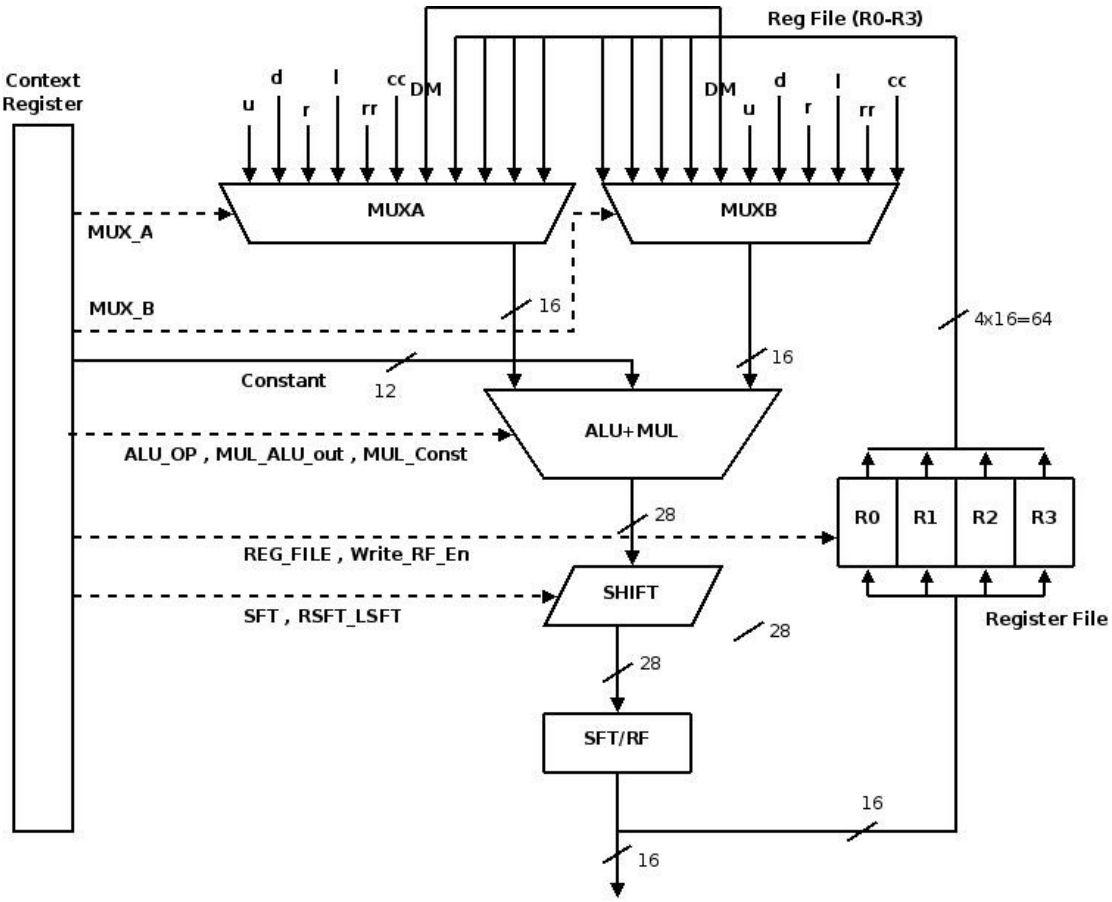
Η αρχική μονάδα στο datapath του επαναδιατάξιμου κελιού είναι ένα σύμπλεγμα από δυο πολυπλέκτες 16-σε-1, που χειρίζονται δεδομένα των 16 bits. Η χρησιμότητά τους στο σύστημα είναι ότι παρέχουν στην κύρια μονάδα υπολογισμών του κελιού (βλ. παρακάτω, μονάδα ALU/MUL) τις δύο εισόδους δεδομένων που θα επεξεργαστεί.

Συγκεκριμένα, κάθε ένας από τους δύο πολυπλέκτες έχει ανατεθεί σε μία είσοδο δεδομένων του επόμενου σταδίου. Οπότε η έξοδος 16 bits του πολυπλέκτη A έχει ενωθεί με τη μία είσοδο δεδομένων της μονάδας ALU/MUL, ενώ η έξοδος του πολυπλέκτη B έχει ενωθεί με την άλλη είσοδο δεδομένων της μονάδας ALU/MUL.

Οι πολυπλέκτες αυτοί έχουν συνολικά 16 εισόδους δεδομένων των 16 bits, και για το λόγο αυτό υπάρχει και είσοδος ελέγχου με μήκος 4 bits. Μέσω της τιμής της εισόδου αυτής γίνεται η επιλογή μεταξύ των 16 εισόδων για προώθηση στην έξοδο του πολυπλέκτη. Η είσοδος ελέγχου παρέχεται από κατάλληλα bits της λέξης διαμόρφωσης που βρίσκεται στον τοπικό καταχωρητή διαμόρφωσης.

Οι δυνατές πηγές δεδομένων εισόδου για τη μονάδα ALU/MUL είναι οι ακόλουθες :

- Η έξοδος από το κελί που βρίσκεται ακριβώς από πάνω στον πίνακα (u_in)
- Η έξοδος από το κελί που βρίσκεται ακριβώς από κάτω στον πίνακα (d_in)
- Η έξοδος από το κελί που βρίσκεται ακριβώς από τα αριστερά στον πίνακα (l_in)
- Η έξοδος από το κελί που βρίσκεται ακριβώς από τα δεξιά στον πίνακα (r_in)
- Η έξοδος από το κελί που βρίσκεται στην ίδια στήλη στον πίνακα αλλά διαφέρει από το παρόν κατά δύο κελιά (cc_in)
- Η έξοδος από το κελί που βρίσκεται στην ίδια σειρά στον πίνακα αλλά διαφέρει από το παρόν κατά δύο κελιά (rr_in)
- Δεδομένα που προέρχονται από τη μνήμη δεδομένων (DM_in). Περισσότερες πληροφορίες σχετικά με την επικοινωνία με τη μνήμη δεδομένων αναφέρονται σε επόμενες ενότητες
- Τα περιεχόμενα των 4 καταχωρητών του τοπικού RF (RFx_in)



Σχήμα 5.5: Η εσωτερική οργάνωση του επαναδιατάξιμου κελιού

Να σημειωθεί ότι το δίκτυο διασυνδέσεων που έχει υλοποιηθεί είναι ένα πλήρες 2D-mesh και κυκλικό δίκτυο, δηλαδή κάθε κελί μπορεί να επικοινωνήσει με κάθε γειτονικό του και επιπλέον με όλα τα κελιά της γραμμής και της στήλης όπου ανήκει. Περισσότερες πληροφορίες δίνονται στην επόμενη ενότητα.

Ο Πίνακας 5.1 δείχνει αναλυτικά το πώς έχουν μοιραστεί οι δυνατές πηγές δεδομένων εισόδου –που περιγράφηκαν παραπάνω- στις εισόδους του κάθε πολυπλέκτη.

Πίνακας 5.1: Είσοδοι δεδομένων των πολυπλεκτών εισόδου

Είσοδος πολυπλέκτη	Σήμα ελέγχου	Σήμα εισόδου
in0	0000	u_in
in1	0001	d_in
in2	0010	l_in
in3	0011	r_in
in4	0100	cc_in
in5	0101	rr_in
in6	0110	DM_in
in7	0111	RF0_in
in8	1000	RF1_in
in9	1001	RF2_in

in10	1010	RF3_in
in11	1011	RF4_in
in12	1100	-
in13	1101	-
in14	1110	-
in15	1111	-

5.2.3.3. Αριθμητική Λογική Μονάδα

Η αριθμητική και λογική μονάδα (ALU) είναι μια από τις σημαντικότερες μονάδες στο datapath του επαναδιατάξιμου κελιού και επιτελεί τις βασικότερες λειτουργίες του. Η συγκεκριμένη ALU που χρησιμοποιήθηκε επεξεργάζεται δεδομένα των 16 bits και -συνεπώς- δέχεται είσοδο ελέγχου μεγέθους 3 bits.

Οι λειτουργίες που έχουν ανατεθεί στην ALU είναι οι κλασσικές λειτουργίες μιας τέτοιας μονάδας σε έναν επεξεργαστή, δηλαδή η πρόσθεση και η αφαίρεση, οι λογικές πράξεις and/or και μία λειτουργία για σύγκριση δεδομένων. Επιπλέον υπάρχει και μια λειτουργία κατά την οποία η ALU δεν επιτελεί κάποιον υπολογισμό, αλλά απλώς μεταφέρει μία από τις εισόδους της στην έξοδο.

Ο Πίνακας 5.2 παρουσιάζει συνοπτικά τις λειτουργίες της ALU μαζί με τις αντίστοιχες τιμές των εισόδων ελέγχου, με βάση τις οποίες γίνεται η επιλογή μεταξύ των διαφορετικών λειτουργιών.

Πίνακας 5.2: Οι λειτουργίες της ALU και οι είσοδοι ελέγχου

Control bits	Function
000	logical and
001	logical or
010	Add
011	no operation / pass data to output
110	Substract
111	set on less than

Η ALU δέχεται δύο εισόδους a και b, με μέγεθος 16 bits και εκτελεί κάποιον υπολογισμό μεταξύ των δύο αυτών εισόδων, ανάλογα με το σήμα *control* των 3 bits. Το αποτέλεσμα του εκάστοτε υπολογισμού βρίσκεται στην έξοδο της ALU, με μέγεθος επίσης 16 bits. Υπάρχουν, επιπλέον, δύο ακόμα έξοδοι, με μήκος 1 bit έκαστη, που ονομάζονται zero και overflow. Η πρώτη από αυτές, η έξοδος zero ενεργοποιείται όταν μετά από μια πράξη αφαίρεσης το αποτέλεσμα είναι 0, δηλαδή οι δύο είσοδοι είναι ίσες. Η άλλη έξοδος, η overflow, ενεργοποιείται όταν σε μια πράξη πρόσθεσης ή αφαίρεσης υπάρχει υπερχείλιση, δηλαδή το αποτέλεσμα δεν μπορεί να παρασταθεί στο μήκος των 16 bits της εξόδου.

Ο Πίνακας 5.3 δείχνει αναλυτικά, σε κάθε περίπτωση λειτουργιών που επιτελεί η

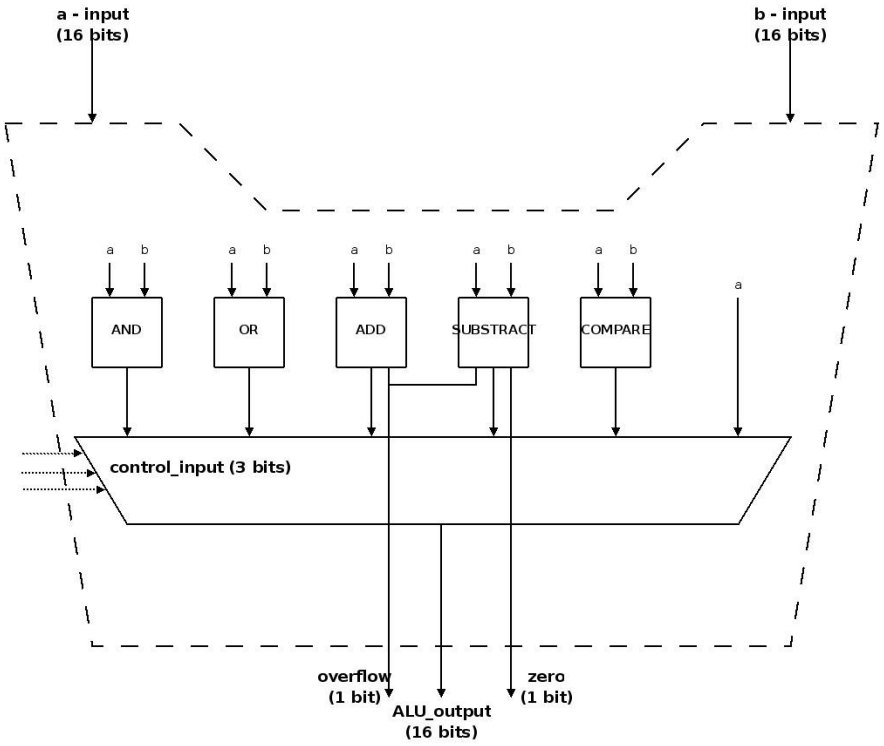
ALU την τιμή που παίρνουν οι τρεις έξοδοι της μονάδας, δηλαδή η έξοδος του αποτελέσματος και οι δύο έξοδοι κατάστασης result και zero.

Πίνακας 5.3: Είσοδοι /έξοδοι για τις λειτουργίες της ALU

Είσοδοι			**Έξοδοι**		
control	**a**	**b**	**result**	**zero**	**overflow**
000	X	X	a&b	0	0
001	X	X	a|b	0	0
010	X	X	a+b	0	1 στην υπερχείλιση και 0 αλλιώς
011	X	X	a	0	0
110	X	a	a-b	1	1 στην υπερχείλιση και 0 αλλιώς
	X	$\neq a$		0	
111	<b	X	1	0	0
	$\geqslant b$	X	0		
XXX	X	X	X	0	0

Υπάρχουν δύο στοιχεία που πρέπει να αναφερθούν και τα οποία σχετίζονται με το πώς ενσωματώνεται η μονάδα της ALU στο επαναδιατάξιμο κελί. Το πρώτο από αυτά είναι ο τρόπος με τον οποίο παρέχονται οι είσοδοι της μονάδας. Οι δύο είσοδοι a και b, που αποτελούν και τα δεδομένα προς επεξεργασία, προέρχονται από τις εξόδους των δύο πολυπλεκτών εισόδου που υπάρχουν σε κάθε κελί. Έτσι η ALU μπορεί να διαμορφωθεί ώστε να επεξεργαστεί δεδομένα από όλες τις δυνατές πηγές του κελιού, ανάλογα με τα bit ελέγχου των δύο πολυπλεκτών. Από την άλλη, οι είσοδοι ελέγχου της ALU, οι οποίες και καθορίζουν τη λειτουργία που θα εκτελέσει αυτή, προέρχονται απευθείας από τη λέξη διαμόρφωσης που έχει αποθηκευτεί στον καταχωρητή διαμόρφωσης του κελιού. Έτσι ο επιθυμητός υπολογισμός που θα εκτελεστεί σε κάθε κύκλο από την ALU μπορεί να προσδιοριστεί με βάση το περιεχόμενο της λέξης διαμόρφωσης και συγκεκριμένα των bits εκείνων που αντιστοιχούν στις εισόδους ελέγχου της ALU (περισσότερες πληροφορίες στην ενότητα για τη λέξη διαμόρφωσης).

Κατά δεύτερον, πρέπει να τονιστεί η σημασία ύπαρξης της λειτουργίας στην περίπτωση που τα bits ελέγχου πάρουν την τιμή 011. Μέσα στις δυνατότητες που πρέπει να υποστηρίζει το επαναδιατάξιμο κελί είναι να μπορούν δεδομένα από την είσοδο να διέρχονται μέχρι την έξοδο χωρίς τροποποίηση. Τα δεδομένα της εισόδου όμως, πρέπει να περάσουν αναγκαστικά και από το επίπεδο όπου γίνονται οι υπολογισμοί (από τη μονάδα ALU-MUL). Για το λόγο αυτό και επιπλέον για να μην καταφεύγει ο προγραμματιστής σε έμμεσες και μη αποδοτικές λύσεις, όπως για παράδειγμα η πρόσθεση του αριθμού 0 στην είσοδο που χρειάζεται χωρίς τροποποίηση να εμφανιστεί στην έξοδο, έχει εισαχθεί η συγκεκριμένη λειτουργία στις λειτουργίες της ALU. Σύμφωνα και με τα όσα έχουν αναφερθεί ως τώρα, στο Σχήμα 5.6 φαίνεται ένα απλοποιημένο διάγραμμα της μονάδας ALU.



Σχήμα 5.6: Γενικό διάγραμμα της ALU σε κάθε επαναδιατάξιμο κελί

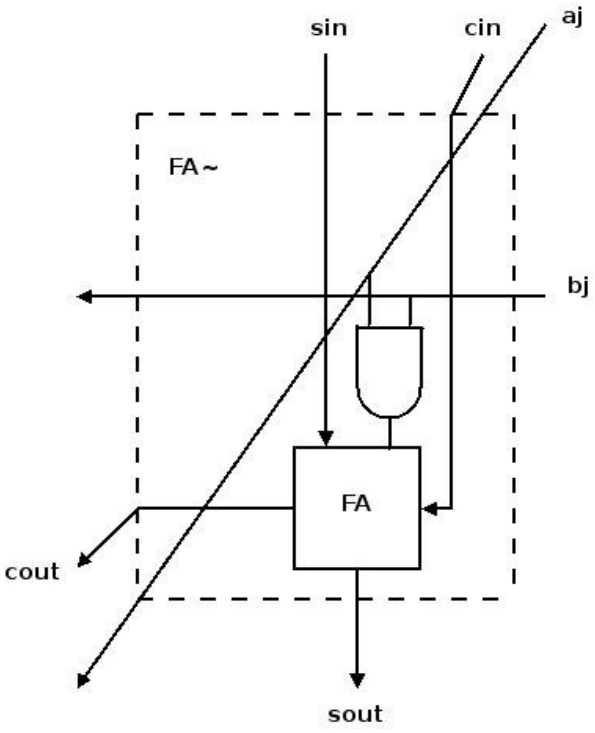
5.2.3.4. Πολλαπλασιαστής

Ο πολλαπλασιαστής που έχει χρησιμοποιηθεί στο επαναδιατάξιμο κελί είναι ένας παράλληλος πολλαπλασιαστής με αθροιστή αποθήκευσης κρατουμένου (Carry Save Adder - CSA) ή αλλιώς ένας πολλαπλασιαστής με σώσιμο κρατουμένου. Το μέγεθος των δύο εισόδων που δέχεται είναι 16 bits και το αποτέλεσμα του πολλαπλασιασμού που προκύπτει είναι 32 bits.

Ο συγκεκριμένος τύπος πολλαπλασιαστή είναι αρκετά διαδεδομένος. Στη γενική του μορφή ακολουθεί την τυπική σχεδίαση των παράλληλων πολλαπλασιαστών, δηλαδή αποτελείται από ένα δίκτυο παραγωγής των μερικών γινομένων $a_i b_j$. Αυτό απαρτίζεται από λογικές πύλες AND και έναν αθροιστή για την πρόσθεση των -τελικών- μερικών γινομένων, στη συγκεκριμένη περίπτωση έναν carry save αθροιστή.

Η βασική δομική μονάδα του πολλαπλασιαστή αυτού φαίνεται στο Σχήμα 5.7 και αποτελείται από έναν πλήρη αθροιστή και μια πύλη AND, η οποία ενσωματώνεται μέσα στο κύτταρο για ομοιομορφία. Η πύλη αυτή χρησιμεύει στη δημιουργία του ενδιάμεσου γινομένου των δύο bits των αριθμών, το οποίο

εισέρχεται στον πλήρη αθροιστή για να προστεθεί με το ενδιάμεσο γινόμενο και το κρατούμενο εξόδου της προηγούμενης βαθμίδας.



Σχήμα 5.7: Δομική μονάδα του πολλαπλασιαστή

Το βασικό στοιχείο που χαρακτηρίζει τον πολλαπλασιαστή είναι το ότι τα κρατούμενα κάθε κυττάρου δε διαδίδονται στο επόμενο κύτταρο του ίδιου επιπέδου (όπως σε άλλους τύπους πολλαπλασιαστών), αλλά οδηγούνται στο επόμενο επίπεδο.

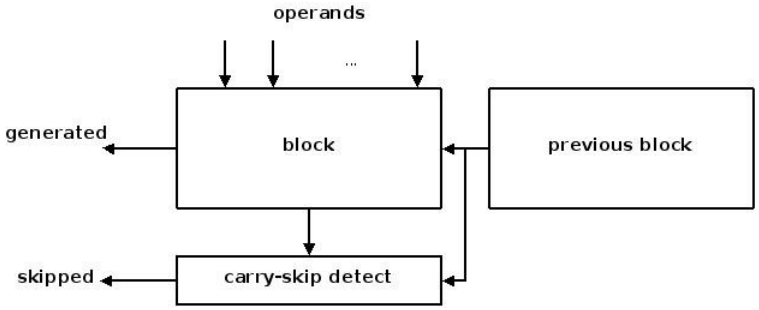
Μετά το τέλος της τελευταίας βαθμίδας του πολλαπλασιαστή, έχουν προκύψει τα πιο σημαντικά bits του αποτελέσματος, όχι όμως στην κοινή δυαδική μορφή. Λόγω της ιδιομορφίας της δικτύωσης, όπου τα κρατούμενα εξόδου διαδίδονται στο επόμενο επίπεδο, το αποτέλεσμα προκύπτει σε μορφή αθροίσματος-κρατουμένου. Για το λόγο αυτό, πρέπει να χρησιμοποιήσουμε έναν απλό παράλληλο αθροιστή, όπως για παράδειγμα έναν αθροιστή διάδοσης κρατουμένου ή έναν αθροιστή πρόβλεψης κρατουμένου.

Ο τύπος του αθροιστή που θα χρησιμοποιηθεί στο τελευταίο στάδιο παίζει σημαντικό ρόλο στον πολλαπλασιαστή, καθώς καθορίζει σε σημαντικό βαθμό το χρόνο καθυστέρησης του όλου κυκλώματος. Για τη συγκεκριμένη υλοποίηση έχει επιλεγεί να χρησιμοποιηθεί ένας αθροιστής τύπου carry skip (παράβλεψης κρατουμένου).

Carry-skip-adder

Σε αντίθεση με τους αθροιστές διάδοσης κρατουμένου, στους οποίους η διάδοση του κρατουμένου και ο υπολογισμός του αθροίσματος υλοποιούνται σε

πανομοιότυπες δομικές μονάδες (τους πλήρεις αθροιστές FA), η κύρια ιδέα (Σχήμα 5.8) στους αθροιστές carry-skip είναι η διάδοση του κρατουμένου να γίνεται ξεχωριστά και ομαδοποιημένα.



Σχήμα 5.8: Η κύρια ιδέα που εφαρμόζεται στους αθροιστές carry-skip

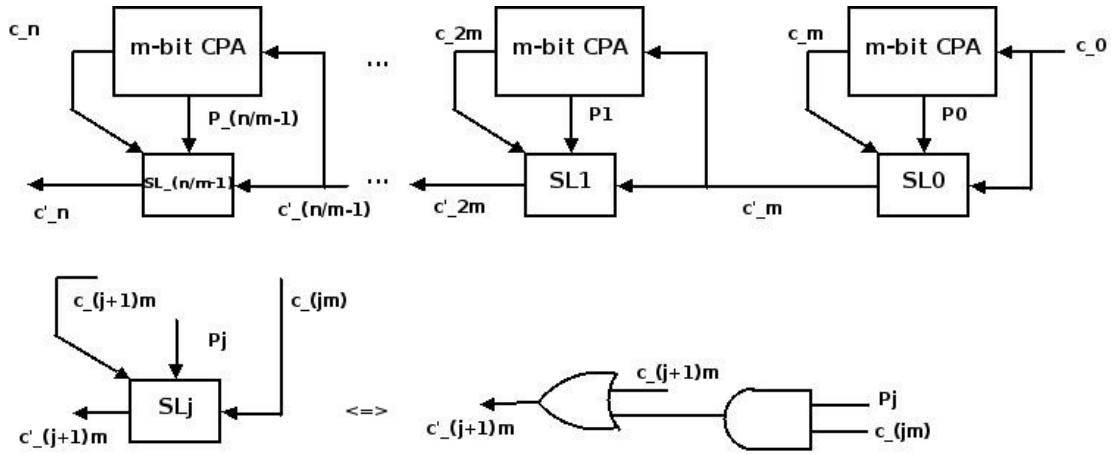
Πιο συγκεκριμένα, τα στάδια της άθροισης χωρίζονται σε ομάδες, έστω των m bits, το μέγεθος των οποίων είναι μεταβλητό αλλά οπωσδήποτε πρέπει να καθοριστεί πριν τη σχεδίαση. Σε κάθε ομάδα η λειτουργία της άθροισης γίνεται κανονικά και επιπλέον προστίθεται λογική για παράβλεψη (skip) του κρατουμένου στην ομάδα αυτή. Η λογική αυτή ανιχνεύει πότε το κρατούμενο εισόδου στην ομάδα (από το προηγούμενο στάδιο) μπορεί να μεταφερθεί κατευθείαν στην επόμενη ομάδα. Για την ακρίβεια αυτό το λογικό κύκλωμα παράγει το κρατούμενο εξόδου της εκάστοτε ομάδας εάν είτε η άθροιση των m-bit των εισόδων παράγει κρατούμενο εξόδου είτε κάποιο κρατούμενο εισέρχεται στη συγκεκριμένη ομάδα και μεταδίδεται διαμέσου του σταδίου αυτού.

Αν για κάθε μονάδα άθροισης οριστεί η μεταφορά κρατουμένου ως $T_i = a_i + b_i$, τότε μπορεί να εντοπιστεί περίπτωση για προσπέραση του κρατουμένου προς την επόμενη ομάδα, όταν $P_{(j+m-1)/m-1} = T_j \cdot T_{j+1} \cdot \ldots \cdot T_{j+m-1} = 1$. Συνολικά, το κρατούμενο εξόδου από μια ομάδα υπολογίζεται από το λογικό κύκλωμα της πρόβλεψης ως

$$c'_{j+m-1} = P_{(j+m-1)/m-1} \cdot c_{j-1} + c_{j+m-1}$$

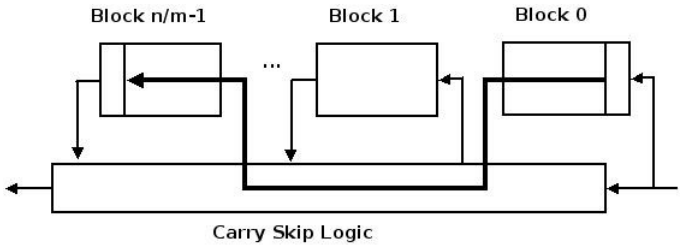
Παρατηρείται ότι η υλοποίηση της λογικής προσπέρασης απαιτεί μόλις δύο πύλες σε διαφορετικό κυκλωματικό επίπεδο (μία πύλη and και μία πύλη or), οπότε εισάγει καθυστέρηση δύο λογικών επιπέδων.

Στο Σχήμα 5.9 φαίνεται το λογικό διάγραμμα ενός αθροιστή carry-skip.



Σχήμα 5.9: Γενικό διάγραμμα ενός carry-skip αθροιστή

Το μέγεθος μιας ομάδας σε έναν carry-skip αθροιστή είναι πολύ σημαντικό, αφού επηρεάζει άμεσα τη χρονική καθυστέρηση που εισάγει ο αθροιστής. Η χειρότερη περίπτωση χρόνου λειτουργίας (Σχήμα 5.10) συμβαίνει όταν παράγεται κρατούμενο στην πρώτη-πρώτη ομάδα του αθροιστή, το κρατούμενο αυτό skip όλα τα ενδιάμεσα στάδια και φτάνει μέχρι το τελευταίο στάδιο, όπου γίνεται κρατούμενο εξόδου.



Σχήμα 5.10: Χειρότερη περίπτωση από πλευράς χρονικής καθυστέρησης

Τότε ο χρόνος άθροισης χειρότερης περίπτωσης είναι ίσος με

$$\frac{2n}{m}+4m-4$$

όπου n το πλάτος του αθροιστή και m το μέγεθος της ομάδας του αθροιστή.

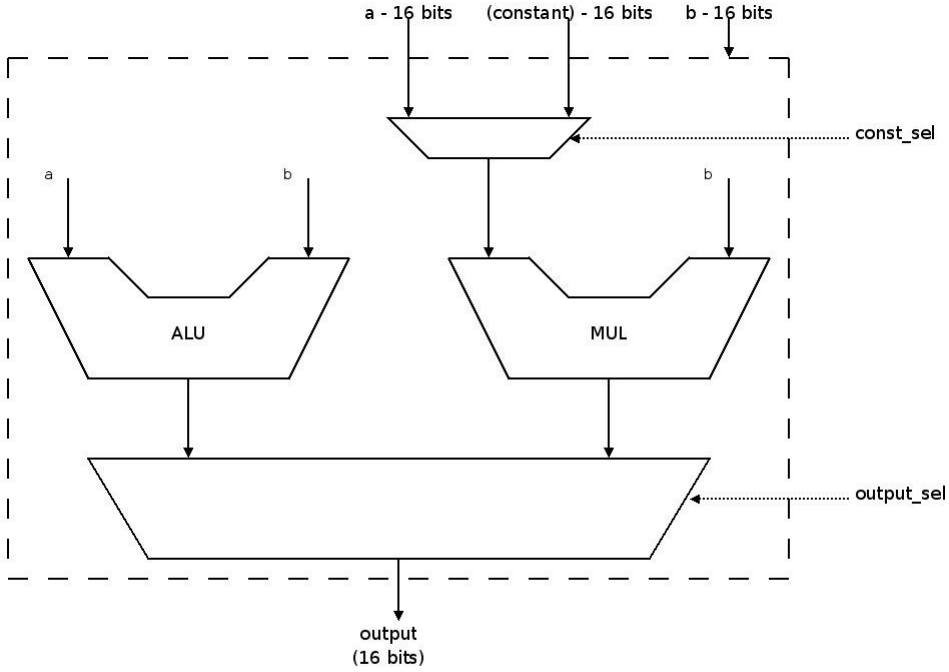
Στην πράξη πολλές φορές τα μεγέθη των ομάδων είναι μη ομοιόμορφα, το οποίο δίνει την καλύτερη απόδοση.

5.2.3.5. Μονάδα ALU-MUL

Στην πραγματικότητα η αριθμητική λογική μονάδα και ο πολλαπλασιαστής αποτελούν μία ενιαία μονάδα υπολογισμών, εννοώντας ότι βρίσκονται στο ίδιο

ακριβώς στάδιο του datapath του επαναδιατάξιμου κελιού. Για το λόγο αυτό πρέπει να υπάρχει μια κοινή έξοδος από την ALU και τον πολλαπλασιαστή. Η μονάδα αυτή δέχεται δύο εισόδους από τους πολυπλέκτες εισόδου, εκτελεί τον κατάλληλο υπολογισμό πάνω στα δεδομένα αυτά -ο οποίος μπορεί να είναι οποιοσδήποτε από το φάσμα λειτουργιών της ALU ή ο πολλαπλασιασμός στον πολλαπλασιαστή- και το αποτέλεσμα της πράξης βγαίνει στη μοναδική έξοδο της μονάδας.

Για να υλοποιηθεί η λειτουργία που περιγράφηκε παραπάνω και να υπάρχει συνεργασία σε μία ενιαία οντότητα της ALU και του πολλαπλασιαστή έχει χρησιμοποιηθεί ένας πολυπλέκτης εξόδου 2-σε-1 με μέγεθος δεδομένων 32 bits. Ο πολυπλέκτης αυτός δέχεται στις δύο εισόδους δεδομένων του την έξοδο από την ALU, έχοντας επεκτείνει τον αριθμό αυτό από τα 16 στα 32 bits με την πρόσθεση μηδενικών στα πιο σημαντικά bits του τελικού αριθμού και τα 32 λιγότερο σημαντικά bits από την έξοδο του πολλαπλασιαστή. Η είσοδος ελέγχου του πολυπλέκτη, με μήκος 1 bit, προσδιορίζει το ποιος αριθμός θα επιλεγεί και θα προωθηθεί προς την έξοδο. Η είσοδος ελέγχου παίρνει την τιμή της από κάποιο bit της τρέχουσας λέξης διαμόρφωσης που ελέγχει το κελί και βρίσκεται στον καταχωρητή διαμόρφωσης. Στο Σχήμα 5.11 φαίνεται ένα δομικό διάγραμμα της μονάδας ALU/MUL, όπου παρουσιάζεται η εσωτερική του οργάνωση και τα σήματα εισόδου/εξόδου.



const_sel (1 bit) : Επιλέγει για την είσοδο 1 του MUL μεταξύ της κανονικής εισόδου δεδομένων και της "σταθεράς"
output_sel (1 bit) : Επιλέγει για την έξοδο της μονάδας ALU-MUL μεταξύ των αποτελεσμάτων της ALU και του MUL

Σχήμα 5.11: Διάγραμμα της μονάδας ALU/MUL του επαναδιατάξιμου κελιού

Ο πολυπλέκτης εξόδου της μονάδας ALU/MUL υπάρχει στο σημείο αυτό για να επιλέγει ουσιαστικά ποια από τις δύο εξόδους, είτε αυτή της ALU είτε αυτή του πολλαπλασιαστή, πρόκειται να χρησιμοποιηθεί από το κελί. Και οι δύο μονάδες λειτουργούν δηλαδή κανονικά, παράγοντας κάποιο αποτέλεσμα με βάση τις

εισόδους που λαμβάνουν και στη συνέχεια, ανάλογα με τον έλεγχο από τη λέξη διαμόρφωσης, γίνεται η επιλογή του εκάστοτε επιθυμητού υπολογισμού.

Να τονιστεί, τέλος, ότι υπάρχει ακόμα ένα πολυπλέκτης 2-σε-1, του οποίου η έξοδος συνδέεται με την είσοδο a του πολλαπλασιαστή. Ο πολυπλέκτης αυτός χρησιμοποιείται για να γίνει η επιλογή μεταξύ της κανονικής εισόδου του πολλαπλασιαστή (η οποία προέρχεται από τον αρχικό πολυπλέκτη εισόδου) και μιας σταθεράς. Η σταθερά αυτή παρέχεται απευθείας από τη τρέχουσα λέξη διαμόρφωσης που είναι αποθηκευμένη στον καταχωρητή διαμόρφωσης.

5.2.3.6. Καταχωρητής Μεταβλητής Ολίσθησης

Η μονάδα της ολίσθησης βρίσκεται αμέσως μετά τη συνδυασμένη μονάδα ALU-MUL και τροφοδοτείται από την έξοδο της τελευταίας. Παίρνει δηλαδή σαν είσοδο τα δεδομένα των 32 bits από τον πολυπλέκτη εξόδου της μονάδας ALU-MUL.

Η ολίσθηση που μπορεί να γίνει στον καταχωρητή αυτό έχει μεταβλητό τόσο το μέγεθος όσο και την κατεύθυνσή της. Οι παράμετροι αυτοί καθορίζονται από τη λέξη διαμόρφωσης που ελέγχει το κελί κάθε φορά και βρίσκεται στον καταχωρητή διαμόρφωσης.

Το πλήθος των θέσεων κατά το οποίο θα γίνει η ολίσθηση κυμαίνεται από 0, οπότε και ο αριθμός διέρχεται από την παρούσα μονάδα αμετάβλητος, έως 32, οπότε και ουσιαστικά προκύπτει ένας μηδενικός αριθμός από τον ολισθητή. Η μονάδα εκτελεί λογική ολίσθηση, οπότε όσες θέσεις του αριθμού αδειάζουν, είτε δεξιά είτε αριστερά, συμπληρώνονται με μηδενικά. Για να είναι δυνατός ο έλεγχος του μεγέθους της ολίσθησης, είναι απαραίτητο να υπάρχει μία είσοδος ελέγχου με πλήθος 5 bits που θα καθορίζει ακριβώς το ποσό αυτό. Αυτή η είσοδος ελέγχου προέρχεται από τη λέξη διαμόρφωσης.

Η κατεύθυνση της ολίσθησης μπορεί να είναι είτε προς τα δεξιά είτε προς τα αριστερά. Καθορίζεται δε από μια είσοδο ελέγχου του 1 bit, η οποία προέρχεται από τη λέξη διαμόρφωσης.

Η μονάδα της ολίσθησης δίνει μεγαλύτερη ευελιξία στο επαναδιατάξιμο κελί και στους υπολογισμούς που επιτελούνται, αφού είναι δυνατόν σε ένα αποτέλεσμα από κάποια πράξη, π.χ. κάποια πρόσθεση ή ακόμα καλύτερα πολλαπλασιασμό, να εφαρμοστεί ολίσθηση στον ίδιο κύκλο λειτουργίας. Αυτό μπορεί να είναι χρήσιμο σε περιπτώσεις όπου το αποτέλεσμα που έχει προκύψει είναι κάποιος δεκαδικός αριθμός και είναι επιθυμητό να διατηρηθούν ορισμένα μόνο δεκαδικά ψηφία, οπότε γίνεται στρογγυλοποίηση του αποτελέσματος. Ακόμη με τον καταχωρητή ολίσθησης είναι δυνατόν να εκτελεστεί στον ίδιο κύκλο τόσο πρόσθεση, στην ALU, όσο και πολλαπλασιασμός ή διαίρεση με παράγοντα κάποια δύναμη του 2, μέσω της ολίσθησης.

5.2.3.7. Καταχωρητής Εξόδου SFT/RF

Ο καταχωρητής αυτός βρίσκεται μεταξύ του καταχωρητή ολίσθησης και του αρχείου καταχωρητών και έχει μέγεθος 32 bits. Η συγκεκριμένη μονάδα λειτουργεί σαν ένας μανδαλωτής που είναι ευαίσθητος στην αρνητική ακμή του παλμού του ρολογιού και παρέχει την έξοδο δεδομένων του κελιού. Η παρουσία του είναι απαραίτητη ώστε να είναι εφικτή η εγγραφή στο αρχείο καταχωρητών, σε κύκλο ρολογιού επόμενο από αυτόν κατά τον οποίο προκύπτει το αποτέλεσμα των υπολογισμών. Λειτουργεί δηλαδή σαν ένα στοιχείο καθυστέρησης στο datapath του επαναδιατάξιμου κελιού.

5.2.3.8. Αρχείο Καταχωρητών (Register File – RF)

Το αρχείο καταχωρητών (Σχήμα 5.12) είναι μια τοπική μονάδα αποθήκευσης μέσα στο επαναδιατάξιμο κελί, όπου μπορούν να αποθηκεύονται δεδομένα από υπολογισμούς που εκτελούνται στο κελί. Μάλιστα είναι η μόνη αποθηκευτική μονάδα στο συνολικό σύστημα στην οποία τα κελιά έχουν πρόσβαση για απευθείας αποθήκευση δεδομένων. Αποτελείται από 4 καταχωρητές, με μέγεθος 16 bits ο κάθε ένας.

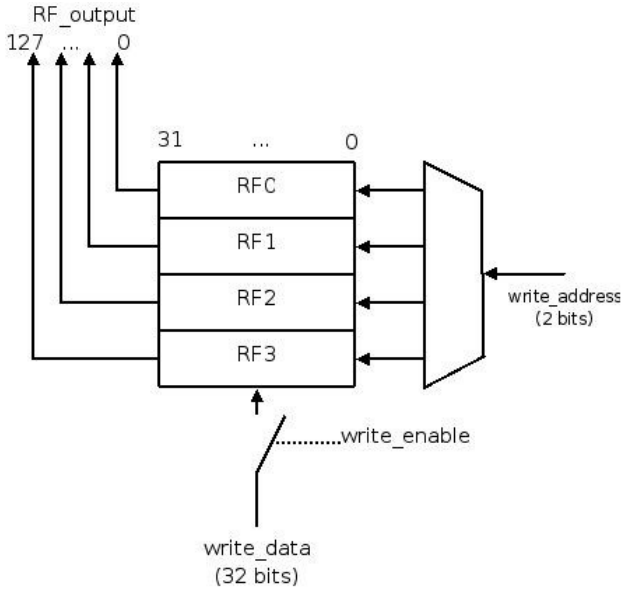
Η έξοδος του RF έχει μέγεθος 4x16 bits, δηλαδή παρέχεται ταυτόχρονα το περιεχόμενο και των τεσσάρων καταχωρητών του αρχείου. Λόγω αυτού του γεγονότος, κατά τη διαδικασία ανάγνωσης από το RF δεν απαιτείται να παρέχεται κάποια διεύθυνση ανάγνωσης προς τη μονάδα αυτή, αφού ουσιαστικά όλα τα δεδομένα είναι διαθέσιμα. Η επιλογή των δεδομένων του κατάλληλου καταχωρητή γίνεται από τη μονάδα που ζήτησε ανάγνωση στο αρχείο. Στη συγκεκριμένη περίπτωση του επαναδιατάξιμου κελιού οι μονάδες που διαβάζουν από το αρχείο καταχωρητών είναι οι δύο πολυπλέκτες εισόδου. Οπότε η έξοδος από το RF μεταφέρεται στους πολυπλέκτες αυτούς, όπου και γίνεται ένα είδος απόπλεξης, αφού τα bits που αντιστοιχούν στα περιεχόμενα των 4 καταχωρητών παρέχονται σε διαφορετικές εισόδους των συγκεκριμένων πολυπλεκτών.

Αντίθετα, υπάρχει μόνο μία είσοδος δεδομένων, μήκους 32 bits. Επομένως, κατά τη διαδικασία εγγραφής στο αρχείο καταχωρητών πρέπει να παρέχεται μία διεύθυνση εγγραφής, η οποία στην ουσία προσδιορίζει άμεσα σε ποιον καταχωρητή πρέπει να μεταφερθούν τα δεδομένα της εισόδου. Επειδή, όπως αναφέρθηκε, υπάρχουν 4 καταχωρητές η συγκεκριμένη διεύθυνση εγγραφής έχει μήκος 2 bits και παρέχεται από τη λέξη διαμόρφωσης που ελέγχει το κελί και είναι αποθηκευμένη στον καταχωρητή διαμόρφωσης. Η προώθηση των δεδομένων εισόδου εσωτερικά στο αρχείο προς τον κατάλληλο καταχωρητή γίνεται με τη χρήση ενός αποπλέκτη 1-σε-4, ο οποίος ελέγχεται από τα bits της

διεύθυνσης εγγραφής.

Η εγγραφή σε κάποιον καταχωρητή του αρχείου είναι μια λειτουργία που γίνεται στις θετικές ακμές του ρολογιού του συστήματος και οποιαδήποτε εγγραφή στο RF θα προέρχεται από τη μονάδα ALU/MUL. Αντίθετα, οι λέξεις διαμόρφωσης φτάνουν στα κελιά από τη μνήμη διαμόρφωσης και εγγράφονται στον καταχωρητή διαμόρφωσης στις αρνητικές ακμές του ρολογιού. Οπότε και τα διάφορα σήματα ελέγχου προς τις μονάδες του RC λαμβάνουν νέες τιμές στις αρνητικές ακμές του ρολογιού. Συνεπώς, οι υπολογισμοί στη μονάδα ALU/MUL αρχίζουν να εκτελούνται από το σημείο αυτό και μετά.

Κάθε κελί υποστηρίζει διοχέτευση μεταξύ της φάσης υπολογισμού (χρήση των δομικών μονάδων ALU, MUL και καταχωρητή ολίσθησης) και της φάσης εγγραφής στο εσωτερικό αρχείο καταχωρητών του κάθε κελιού (εγγραφή στο RF). Στην περίπτωση που οι υπολογισμοί και η εγγραφή των δεδομένων εκτελούνταν εντός ενός κύκλού λειτουργίας, οι μονάδες ALU/MUL θα έπρεπε να παράγουν το επιθυμητό αποτέλεσμα με καθυστέρηση μισού κύκλου λειτουργίας ώστε τα δεδομένα να είναι έτοιμα και να εγγράφονται ορθά στο RF. Το παραπάνω οδηγεί σε μεγάλους κύκλους ρολογιού με αποτέλεσμα να μειώνεται η απόδοση του συνολικού συστήματος. Ο απαραίτητος συγχρονισμός στο datapath του κάθε κελιού επεξεργασίας υλοποιείται μέσω ενός ενδιάμεσου καταχωρητή (καταχωρητής SFT/RF). Με τον τρόπο αυτό μπορούν να υποστηριχθούν και πολυπλοκότερα αρχεία καταχωρητών από αυτά που χρησιμοποιούνται στην παρούσα υλοποίηση.



Σχήμα 5.12: Το διάγραμμα του αρχείου καταχωρητών των κελιών

Σύμφωνα με τα παραπάνω, στο datapath του επαναδιατάξιμου κελιού έχει υλοποιηθεί λογική με διοχέτευση δύο επιπέδων. Τα δύο επίπεδα χωρίζονται μεταξύ τους από τον καταχωρητή SFT/RF. Αρχικά έχουμε τους πολυπλέκτες εισόδου, τη μονάδα ALU/MUL και τη μονάδα της ολίσθησης ενώ από το στάδιο αυτό προκύπτουν η έξοδος του κελιού για την τρέχουσα context word. Και κατά

δεύτερον έχουμε το κομμάτι εγγραφής του αρχείου καταχωρητών.

5.2.3.9. Καταχωρητής Διαμόρφωσης (Context Register)

Ο καταχωρητής αυτός βρίσκεται σε κάθε επαναδιατάξιμο κελί και σκοπός του είναι να αποθηκεύει τη τρέχουσα λέξη διαμόρφωσης που ελέγχει το συγκεκριμένο κελί. Η είσοδός του, επομένως, συνδέεται απευθείας με τη μνήμη διαμόρφωσης, μέσω ενός διαδρόμου δεδομένων, με μήκος όσο και το μήκος της λέξης διαμόρφωσης.

Η έξοδος του καταχωρητή διαμοιράζεται σε ολόκληρο το επαναδιατάξιμο κελί. Υπενθυμίζεται ότι σκοπός της λέξης διαμόρφωσης είναι να διαμορφώσει και να καθορίσει την ακριβή λειτουργία ολόκληρου του κελιού. Συνεπώς, ομάδες bits από τα περιεχόμενα του καταχωρητή διαμόρφωσης πρέπει να μοιραστούν στις εισόδους ελέγχου στις κατάλληλες μονάδες που απαρτίζουν εσωτερικά τα κελιά. Κατά την περιγραφή της κάθε μονάδας του κελιού, αναφέρθηκε αναλυτικά τι ακριβώς καθορίζεται για κάθε μονάδα από τη λέξη διαμόρφωσης και σε ποιες ακριβώς εισόδους ελέγχου πρέπει να φτάσουν δεδομένα από τον καταχωρητή διαμόρφωσης.

Σημασία έχει, τέλος, να τονιστεί ο χρονισμός τους συγκεκριμένου καταχωρητή. Ειδικότερα, οι λέξεις από τη μνήμη διαμόρφωσης εγγράφονται στους καταχωρητές διαμόρφωσης σε αρνητικούς παλμούς του ρολογιού του συστήματος.

5.2.4.Λέξη Διαμόρφωσης (Configuration word)

Η λέξη διαμόρφωσης είναι αυτή που καθορίζει πλήρως τη λειτουργία των επαναδιατάξιμων κελιών και κατ`επέκταση ολόκληρου του συστήματος. Το σύνολο των λέξεων διαμόρφωσης για τον προγραμματισμό του επαναδιατάξιμου περιφερειακού, βρίσκεται αποθηκευμένο στη μνήμη διαμόρφωσης, που είναι μια ακόμα εσωτερική μονάδα του. Σε κάθε θετική ακμή του ρολογιού μεταφέρεται μια θέση από τη μνήμη διαμόρφωσης στο διάδρομο δεδομένων διαμόρφωσης. Σε κάθε αρνητική ακμή του ρολογιού γίνεται η εγγραφή μιας λέξης διαμόρφωσης στον καταχωρητή διαμόρφωσης σε κάθε επαναδιατάξιμο κελί.

Το μήκος της λέξης έχει προκύψει -από τη σχεδίαση- ίσο με 31 bits. Τα bits αυτά αντιστοιχούν -στην πλειοψηφία τους- σε κάποια συγκεκριμένη μονάδα του επαναδιατάξιμου κελιού και συνδέονται στις εκάστοτε εισόδους ελέγχου της.

Υπάρχουν δύο καταστάσεις λειτουργίας του επαναδιατάξιμου κελιού, μία ενεργή και μία σε αναστολή. Όταν το κελί είναι ενεργό, τότε επιτελεί όλες τις κανονικές λειτουργίες του, δηλαδή η ALU, ο πολλαπλασιαστής και οι υπόλοιπες μονάδες

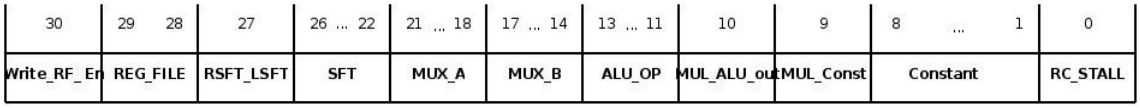
λειτουργούν κανονικά. Στην κατάσταση αναστολής το κελί δεν επιτελεί καμία λειτουργία, δηλαδή οι μονάδες του δεν επιτρέπεται να λειτουργήσουν.

Έτσι έχει χωριστεί και η ίδια η λέξη διαμόρφωσης σε δύο διαφορετικές καταστάσεις λειτουργίας, ανάλογα με την κατάσταση που πρέπει να βρίσκεται το επαναδιατάξιμο κελί. Όταν η λέξη διαμόρφωσης αντιστοιχεί στην ενεργή κατάσταση λειτουργίας, τότε τα bits της λέξης καθορίζουν τις λειτουργίες των διαφόρων μονάδων του κελιού και τη γενικότερη λειτουργία του. Όταν η λέξη διαμόρφωσης αντιστοιχεί στην κατάσταση λειτουργίας σε αναστολή, τότε τα bits της λέξης χρησιμοποιούνται για να καθορίσουν άλλες σημαντικές λειτουργίες που πρέπει να πραγματοποιηθούν. Αυτές σχετίζονται με τη γενικότερη λειτουργία του επαναδιατάξιμου περιφερειακού και τη λειτουργία των μνημών διαμόρφωσης και δεδομένων. (βλ. ενότητα 5.6)

Να σημειωθεί ότι ο διαχωρισμός μεταξύ των δύο καταστάσεων λειτουργίας, όσον αφορά στη λέξη διαμόρφωσης, γίνεται από την τιμή του bit0 (του λιγότερου σημαντικού δηλαδή bit) της λέξης. Έτσι όταν το bit αυτό έχει τιμή 0, τότε η συγκεκριμένη λέξη αντιστοιχεί στην ενεργή κατάσταση λειτουργίας, ενώ όταν έχει την τιμή 1 αντιστοιχεί στη μη ενεργή κατάσταση λειτουργίας.

Παρακάτω θα δοθεί μια αναλυτική περιγραφή της αντιστοίχησης καθενός bit από τη λέξη διαμόρφωσης με τον ακριβή έλεγχο που επιτελεί, μόνο όμως για την ενεργή κατάσταση λειτουργίας. Αναλυτικές πληροφορίες για την αντιστοίχηση στην περίπτωση της μη ενεργής κατάστασης λειτουργίας δίνονται σε επόμενη ενότητα, αφού περιγραφούν και οι αντίστοιχες λειτουργίες που υποστηρίζονται από το σύστημα όταν το κελί βρίσκεται σε αναστολή.

Αρχικά δίνεται ένα γενικό διάγραμμα (Σχήμα 5.13) με την οργάνωση των bits της λέξη διαμόρφωσης, όπου φαίνονται οι διάφορες λογικές ομάδες που δημιουργούνται και στη συνέχεια περιγράφεται αναλυτικά το κάθε πεδίο.



Σχήμα 5.13: Τα πεδία της λέξης διαμόρφωσης στην ενεργή κατάσταση λειτουργίας

- *RC_STALL* (1 bit) – Το πεδίο αυτό καθορίζει την κατάσταση λειτουργίας που αντιστοιχεί η συγκεκριμένη λέξη διαμόρφωσης. Όπως αναφέρθηκε και παραπάνω η τιμή 0 δηλώνει ενεργή κατάσταση λειτουργίας ενώ τη τιμή 1 δηλώνει κατάσταση stall. Το ίδιο αυτό bit τροφοδοτείται εσωτερικά σε όλες τις μονάδες και καθορίζει την κατάσταση λειτουργίας που πρέπει να βρίσκεται το επαναδιατάξιμο κελί.

  Το πώς ακριβώς καθορίζεται εσωτερικά η μη ενεργή κατάσταση, έχει να κάνει με τους πολυπλέκτες εισόδου αλλά και τον τρόπο οργάνωσης του κελιού. Συγκεκριμένα τους πολυπλέκτες αυτούς υπάρχει επιπλέον από τις

άλλες εισόδους που αναφέρθηκαν και μία είσοδος ελέγχου επίτρεψης λειτουργίας, στην οποία συνδέεται απευθείας το bit0 της context word. Όταν επιβάλλεται η κατάσταση ενεργούς λειτουργίας, οι πολυπλέκτες λειτουργούν κανονικά και επιλέγουν την κατάλληλη είσοδο. Όταν όμως επιλέγεται η μη-ενεργή κατάσταση λειτουργίας, τότε οι πολυπλέκτες δεν επιλέγουν καμία είσοδο, παρά βγάζουν μια μηδενική έξοδο.

Στη συνέχεια τα μηδενικά αυτά δεδομένα διέρχονται από τη μονάδα ALU/MUL και τη μονάδα ολίσθησης. Ωστόσο δεν ενδιαφέρει το τι ακριβώς γίνεται εσωτερικά σε αυτές ούτε το αποτέλεσμα που προκύπτει, αφού τα υπόλοιπα bits της λέξης διαμόρφωσης στη μη ενεργή κατάσταση μπορούν να πάρουν οποιαδήποτε τιμή. Υπενθυμίζεται ότι σε αυτή την περίπτωση τα bits της λέξης διαμόρφωσης έχουν κάποια λειτουργικότητα, η οποία όμως δεν αναφέρεται στο κελί αλλά στο συνολικό επαναδιατάξιμο σύστημα.

- *Constant* (8 bits) – Το πεδίο αυτό παρέχει μια σταθερά, η οποία τροφοδοτείται στη μονάδα ALU/MUL και συγκεκριμένα στον πολλαπλασιαστή της, ως ένας πιθανός πολλαπλασιαστικός παράγοντας. Ο αριθμός του πεδίου αυτού είναι δυνατό να αντικαταστήσει το ένα από τα δύο δεδομένα που λαμβάνει ο πολλαπλασιαστής από τις υπόλοιπες πηγές δεδομένων. Παρόλα αυτά δεν είναι σίγουρο πως ο πολλαπλασιαστής θα χρησιμοποιήσει οπωσδήποτε την τιμή αυτή. Έτσι υπάρχει ένα σήμα ελέγχου, που δηλώνει στον πολλαπλασιαστή αν απαιτείται η χρήση της σταθεράς που περιέχεται στο πεδίο αυτό ή αν απλώς πρέπει να χρησιμοποιήσει τις κανονικές εισόδους του. Το σήμα αυτό ελέγχεται από το επόμενο πεδίο της λέξης διαμόρφωσης.

  Η τιμή του πεδίου αποκαλείται "σταθερά" για να γίνει ο διαχωρισμός ότι δεν πρόκειται για κάποιο δεδομένο από τη μνήμη δεδομένων αλλά μια τιμή που προέρχεται απευθείας από τη λέξη διαμόρφωσης.

- *MUL_Const* (1 bit) – Το πεδίο αυτό καθορίζει στη μονάδα ALU/MUL και συγκεκριμένα στον πολλαπλασιαστή της, αν θα πρέπει να γίνει χρήση της σταθεράς που δίνεται μέσω της λέξης διαμόρφωσης στο προηγούμενο πεδίο (*Constant*). Αν έχει την τιμή 0 τότε ο πολλαπλασιαστής θα χρησιμοποιήσει την κανονική είσοδο δεδομένων, ενώ αν έχει την τιμή 1 τότε ο πολλαπλασιαστής θα χρησιμοποιήσει την παραπάνω σταθερά.

  Για να υλοποιηθεί στην πράξη η συγκεκριμένη λειτουργία, όπως αναφέρεται και στην περιγραφή της μονάδας ALU/MUL, υπάρχει ένας πολυπλέκτης εισόδου, η έξοδος του οποίου τροφοδοτεί τη δεύτερη είσοδο δεδομένων του πολλαπλασιαστή. Ο πολυπλέκτης αυτός επιλέγει μεταξύ της σταθεράς από τη λέξη διαμόρφωσης και της κανονικής πηγής εισόδου. Έτσι τα bits του προηγούμενου πεδίου (*Constant*) αποτελούν τη μία είσοδο δεδομένων του συγκεκριμένου πολυπλέκτη, ενώ το bit του τρέχοντος πεδίου τροφοδοτείται στην είσοδο επιλογής του πολυπλέκτη.

- *MUL_ALU_out* (1 bit) – Όπως έχει αναφερθεί, η μονάδα της ALU και ο πολλαπλασιαστής έχουν ενωθεί σε μια ενιαία μονάδα. Λόγω του γεγονότος αυτού και επιπλέον λαμβάνοντας υπόψη ότι τα δύο παραπάνω εργάζονται παράλληλα, θα πρέπει να υπάρχει τρόπος επιλογής μεταξύ της εξόδου από την ALU και της εξόδου του πολλαπλασιαστή. Το πεδίο αυτό χρησιμοποιείται ακριβώς για το σκοπό αυτό. Συγκεκριμένα αν έχει την τιμή 0 τότε επιλέγεται η έξοδος από την ALU, ενώ αν έχει την τιμή 1 τότε επιλέγεται η έξοδος από τον πολλαπλασιαστή. Το bit από το πεδίο αυτό τροφοδοτεί απευθείας την είσοδο επιλογής του πολυπλέκτη εξόδου της μονάδας ALU/MUL, ώστε να γίνει εφικτή η υλοποίηση της παραπάνω λειτουργίας.

- *ALU_OP* (3 bits) – Το πεδίο αυτό ελέγχει τη λειτουργία της μονάδας ALU. Συγκεκριμένα τροφοδοτείται στην είσοδο ελέγχου της ALU. Η τιμή λοιπόν που πρέπει να πάρει το πεδίο *ALU_OP* είναι ακριβώς η τιμή της εισόδου ελέγχου της ALU που αντιστοιχεί στη λειτουργία που είναι επιθυμητό να επιτελεί. Οπότε και η επεξήγηση των bits του συγκεκριμένου πεδίου έχει ήδη παρουσιαστεί στην περιγραφή της μονάδας της ALU.

- *MUX_A* (4 bits) – Το πεδίο αυτό αντιστοιχεί στις εισόδους ελέγχου του ενός από τους δύο πολυπλέκτες εισόδου. Επομένως, η ερμηνεία των bits του πεδίου είναι ακριβώς η ίδια με την ερμηνεία των τιμών που παίρνουν οι είσοδοι ελέγχου του πολυπλέκτη εισόδου. Μέσω του πεδίου αυτού γίνεται η επιλογή της κατάλληλης εισόδου δεδομένων από όλες τις δυνατές που υπάρχουν (βλ. στην αντίστοιχη ενότητα για τους πολυπλέκτες), η οποία θα τροφοδοτηθεί στην πρώτη είσοδο της μονάδας ALU/MUL.

- *MUX_B* (4 bits) – Το πεδίο αυτό αντιστοιχεί στις εισόδους ελέγχου του δεύτερου από τους δύο πολυπλέκτες εισόδου. Είναι το αντίστοιχο πεδίο με το προηγούμενο, αλλά για το δεύτερο πολυπλέκτη.

- *SFT* (5 bits) – Το πεδίο αυτό ελέγχει ένα κομμάτι της λειτουργίας της μονάδας μεταβλητής ολίσθησης που περιλαμβάνεται στο datapath του RC. Συγκεκριμένα τα bits του πεδίου αυτού προσδιορίζουν, σε δυαδική μορφή, τον αριθμό των θέσεων κατά τον οποίο θα γίνει η ολίσθηση. Η άλλη παράμετρος που απαιτείται, η κατεύθυνση, ρυθμίζεται από το επόμενο πεδίο. Το πεδίο αυτό συνδέεται απευθείας με την κατάλληλη είσοδο ελέγχου του καταχωρητή ολίσθησης, εκείνη η οποία καθορίζει τον αριθμό των θέσεων ολίσθησης.

- *RSFT_LSFT* (1 bit) – Το πεδίο αυτό ελέγχει ένα μέρος της λειτουργίας της ολίσθησης που πραγματοποιείται στον καταχωρητή ολίσθησης. Συγκεκριμένα προσδιορίζεται η κατεύθυνση προς την οποία θα γίνει η λειτουργία της ολίσθησης. Αν πάρει την τιμή 0 τότε πραγματοποιείται ολίσθηση προς τα αριστερά, ενώ αν πάρει την τιμή 1 τότε πραγματοποιείται ολίσθηση προς τα δεξιά. Το bit αυτό της λέξης

διαμόρφωσης συνδέεται απευθείας με την είσοδο ελέγχου της μονάδας ολίσθησης η οποία ελέγχει την κατεύθυνση της ολίσθησης.

- *REG_FILE* (2 bits) – Το πεδίο αυτό της λέξης διαμόρφωσης χρησιμοποιείται για τον έλεγχο του αρχείου καταχωρητών. Συγκεκριμένα παρέχει τη διεύθυνση εγγραφής κατά τη διαδικασία εγγραφής δεδομένων στο RF. Στην ουσία εδώ περιέχεται ο αριθμός του καταχωρητή του RF, στον οποίο πρόκειται να γίνει εγγραφή.

- *Write_RF_En* (1 bit) – Το πεδίο αυτό χρησιμοποιείται για τον έλεγχο της διαδικασίας εγγραφής στο αρχείο καταχωρητών. Για την ακρίβεια, η εγγραφή στο RF είναι μια διαδικασία που ελέγχεται από μια είσοδο επίτρεψης, η οποία και οδηγείται από αυτό το συγκεκριμένο bit της λέξης διαμόρφωσης. Όταν παίρνει την τιμή 1, τότε επιτρέπεται εγγραφή στο RF, αλλιώς όταν παίρνει την τιμή 0 τότε η εγγραφή στο RF παρεμποδίζεται.

Ο Πίνακας 5.4 παρουσιάζει συγκεντρωτικά και αναλυτικά τη λειτουργία για κάθε bit της λέξης διαμόρφωσης.

Πίνακας 5.4: Λειτουργίες των πεδίων της λέξης διαμόρφωσης

Πεδίο λέξης διαμόρφωσης	**Τιμή**	**Περιγραφή λειτουργίας**
RC_STALL	0	το κελί σε ενεργή κατάσταση λειτουργίας
	1	το κελί σε μη ενεργή κατάσταση λειτουργίας
Constant	X	σταθερά για τον πολλαπλασιασμό
MUL_Const	0	πολλαπλασιασμός με κανονική είσοδο
	1	χρήση της σταθεράς στον πολλαπλασιασμό
MUL_ALU_out	0	επιλογή αποτελέσματος της ALU
MUL_ALU_out	1	επιλογή αποτελέσματος του MUL
ALU_OP	000	λογικό AND
	001	λογικό OR
	010	πρόσθεση
	011	καμία λειτουργία-μεταφορά εισόδου στην έξοδο
	110	αφαίρεση
	111	σύγκριση
MUX1, MUX2	0000	επιλογή εισόδου από το πάνω κελί
	0001	επιλογή εισόδου από το κάτω κελί
	0010	επιλογή εισόδου από το αριστερά κελί

	0011	επιλογή εισόδου από το δεξιά κελί
	0100	επιλογή εισόδου από το δεύτερο κελί της ίδιας στήλης
	0101	επιλογή εισόδου από το δεύτερο κελί της ίδιας γραμμής
	0110	επιλογή εισόδου από τη μνήμη δεδομένων
	0111	επιλογή εισόδου από τον καταχωρητή reg0 του RF
	1000	επιλογή εισόδου από τον καταχωρητή reg1 του RF
	1001	επιλογή εισόδου από τον καταχωρητή reg2 του RF
	1010	επιλογή εισόδου από τον καταχωρητή reg3 του RF
	1011-1111	δε χρησιμοποιούνται
SFT	00000	ολίσθηση κατά 0 bit
	00001	ολίσθηση κατά 1 bit
	00010	ολίσθηση κατά 2 bit
	...	...
	11111	ολίσθηση κατά 31 bit
RSFT_LSFT	0	ολίσθηση αριστερά
	1	ολίσθηση δεξιά
REG_FILE	00	εγγραφή στον καταχωρητή reg0 του RF
	01	εγγραφή στον καταχωρητή reg1 του RF
	10	εγγραφή στον καταχωρητή reg2 του RF
	11	εγγραφή στον καταχωρητή reg3 του RF
Write_RF_En	0	εγγραφή στο RF μη ενεργή
	1	εγγραφή στο RF ενεργή

5.2.5. Δίκτυο Διασυνδέσεων – Επαναδιατάξιμος Πίνακας

Τα συνολικά 16 επαναδιατάξιμα κελιά που αποτελούν τον επαναδιατάξιμο πίνακα είναι οργανωμένα δομικά σε έναν πίνακα 4x4, με την έννοια του πίνακα να διαμορφώνεται από τις διασυνδέσεις μεταξύ τους.

Με τον όρο "σύνδεση" μεταξύ των κελιών εννοείται ότι οι έξοδοι άλλων κελιών, δηλαδή τα αποτελέσματα από τους υπολογισμούς που εκτελούνται σε άλλες μονάδες επεξεργασίας του συστήματος, έχουν τη δυνατότητα να προσπελαστούν από ένα κελί. Αυτή ακριβώς η διασύνδεση μεταξύ των κελιών επεξεργασίας είναι αναγκαία να υπάρχει, καθώς δίνει στο σύστημα μεγάλη

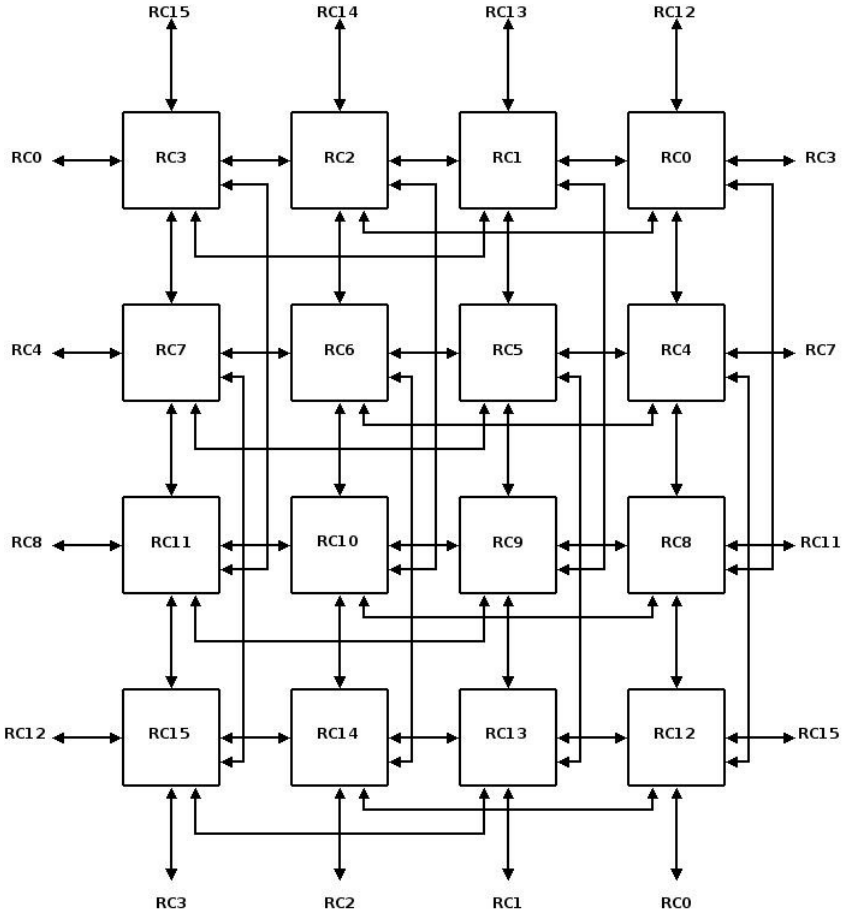
ευελιξία. Έτσι, είναι δυνατόν να ανταλλάσσονται τοπικά διάφορα δεδομένα μεταξύ των κελιών, χωρίς την παρεμβολή της μνήμης του συστήματος, κάτι το οποίο σαφώς προσδίδει μεγάλο χρονικό κέρδος και είναι πολύ αποδοτικό. Μπορεί λοιπόν να σχεδιαστεί μια αλυσίδα υπολογισμών, οι οποίοι να χρειάζεται κάποια στιγμή να χρησιμοποιήσουν ο ένας τα αποτελέσματα του άλλου.

Το δίκτυο διασυνδέσεων που έχει χρησιμοποιηθεί μοιάζει πολύ με το αντίστοιχο δίκτυο που προτάθηκε στην αρχιτεκτονική MorphoSys. Συγκεκριμένα υπάρχει ένα πλήρες δίκτυο 2D mesh, δηλαδή κάθε κελί συνδέεται -σε πρώτη φάση- με όλα τα γειτονικά του. Μάλιστα υπάρχει, κατά κάποιο τρόπο, μια τρισδιάστατη οργάνωση των διασυνδέσεων. Έτσι τα ακριανά κελιά του πίνακα συνδέονται με τα κελιά της άλλης άκρης του πίνακα. Για παράδειγμα, όλα τα κελιά της πρώτης αριστερά στήλης συνδέονται με τα αντίστοιχα κελιά των ίδιων γραμμών της τελευταίας δεξιά στήλης κ.ο.κ Επιπλέον όμως αυτού του πλήρους mesh network, υπάρχουν διασυνδέσεις από ένα κελί προς τα κελιά του πίνακα που βρίσκονται στην ίδια γραμμή/στήλη αλλά διαφέρουν κατά ένα κελί. Στην πραγματικότητα, με το δίκτυο αυτό διασυνδέσεων, κάθε κελί συνδέεται απευθείας με όλα τα κελιά που βρίσκονται στην ίδια γραμμή και στην ίδια στήλη του πίνακα. Επίσης, σε δύο χρονικά βήματα μπορεί να προσπελάσει οποιοδήποτε άλλο κελί του πίνακα, με την έννοια ότι η έξοδος ενός κελιού μεταφέρεται προς κάποιο κελί που βρίσκεται σε διαφορετική γραμμή ή στήλη, με το ενδιάμεσο πέρασμά του από ένα άλλο κελί.

Στο Σχήμα 5.14 φαίνεται ένα απλοποιημένο διάγραμμα του πίνακα, μαζί με τις διασυνδέσεις του.

Παρατηρείται ότι ο πυρήνας του επαναδιατάξιμου συστήματος που σχεδιάστηκε είναι ουσιαστικά μια τετράδα του συνολικού συστήματος MorphoSys. Αυτό σημαίνει ότι ο παρών πίνακας μπορεί πολύ εύκολα να επεκταθεί σε μέγεθος, διατηρώντας ανά τετράδες την ίδια ακριβώς δομή με την τωρινή και με επιπλέον γενικές διασυνδέσεις μεταξύ των τετράδων μέσω διαδρόμων.

Στον πίνακα αυτό μπορεί να θεωρηθεί ότι τα κελιά που βρίσκονται στην ίδια στήλη/γραμμή βρίσκονται στην ίδια ομάδα επεξεργασίας. Αυτό οφείλεται στο ότι όλα τα κελιά σε μια γραμμή/στήλη δέχονται τα ίδια δεδομένα από τη μνήμη δεδομένων, αλλά και την ίδια λέξη διαμόρφωσης από τη context memory. Αυτό σημαίνει ότι τα κελιά σε μια γραμμή/στήλη εκτελούν τους ίδιους υπολογισμούς, αφού έχουν την ίδια ακριβώς διαμόρφωση. Ωστόσο το ότι δέχονται και τα ίδια δεδομένα από τη μνήμη δεδομένων, δε σημαίνει ότι τα κελιά σε μια στήλη/γραμμή θα χρησιμοποιήσουν στους υπολογισμούς τους τα ίδια ακριβώς δεδομένα. Για παράδειγμα, μπορεί κάποια γραμμή του πίνακα να έχει ρυθμιστεί να εκτελέσει πρόσθεση μεταξύ κάποιου αριθμού που προέρχεται από τη μνήμη δεδομένων και του περιεχομένου ενός καταχωρητή από το αρχείο καταχωρητών, το οποίο δεν είναι απαραίτητο να είναι ίδιο.



Σχήμα 5.14: Οι διασυνδέσεις των κελιών του επαναδιατάξιμου πίνακα

Έτσι διαπιστώνεται ότι, ο πίνακας έχει εμφανή χαρακτηριστικά του μοντέλου επεξεργασίας SIMD, σύμφωνα με το οποίο μια ομάδα μονάδων επεξεργασίας εκτελεί του ίδιους υπολογισμούς, αλλά χρησιμοποιώντας διαφορετικά δεδομένα. Περισσότερες πληροφορίες πάνω στις μνήμες και τη σύνδεσή τους με τον επαναδιατάξιμο πίνακα δίνονται σε επόμενες ενότητες του παρόντος.

Η συγκεκριμένη οργάνωση των κελιών επεξεργασίας σε πίνακα παρέχει πολλά οφέλη. Καταρχήν η δομή του πίνακα δίνει τη δυνατότητα να συνδεθούν τα επαναδιατάξιμα κελιά μεταξύ τους με αποδοτικό τρόπο, ώστε να μη χρειάζεται μεγάλο κυκλωματικό κομμάτι για τις διασυνδέσεις. Κατά δεύτερον η οργάνωση των κελιών σε πίνακα διευκολύνει την ομαδοποιημένη χρησιμοποίηση των δεδομένων διαμόρφωσης (context words). Τέλος με αυτή την αρχιτεκτονική υπάρχουν μεγάλες δυνατότητες επέκτασης του μεγέθους του συστήματος, όπως ακριβώς συμβαίνει και με στο MorphoSys.

Θα πρέπει να σημειωθεί ότι αυτή η κατά κάποιο τρόπο τρισδιάστατη αναδίπλωση του πίνακα δεν είναι υποχρεωτικό να χρησιμοποιηθεί για όλες τις εφαρμογές που πρόκειται να υλοποιηθούν. Σε άλλες περιπτώσεις δηλαδή μπορεί να χρειάζεται, για παράδειγμα, τα κελιά που βρίσκονται στην τελευταία γραμμή σε κάθε στήλη να συνδέονται με τα κελιά που βρίσκονται στην πρώτη γραμμή της επόμενης στήλης. Στις εφαρμογές που υλοποιήθηκαν για τον έλεγχο της απόδοσης του συστήματος στα πλαίσια της παρούσας εργασίας, πραγματικά

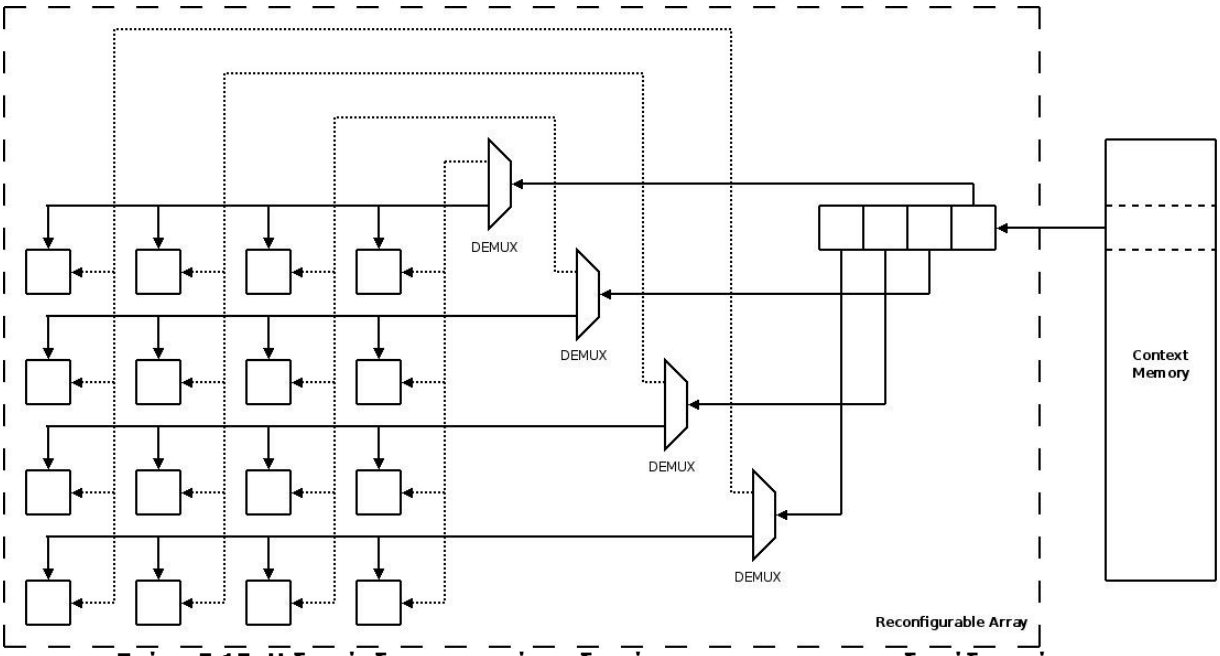
χρειάστηκαν κάποιες μικρές αλλαγές στον τρόπο διασύνδεσης στα όρια του πίνακα. Φυσικά, όταν βρισκόμαστε στο στάδιο της κυκλωματικής περιγραφής του συστήματος με τη γλώσσα Verilog (ή κάποια άλλη γλώσσα περιγραφής υλικού) τέτοιες αλλαγές είναι πολύ εύκολο να γίνουν. Όταν όμως το σύστημα υλοποιηθεί στην πράξη πάνω σε ένα κομμάτι πυριτίου, το χαρακτηριστικό αυτό μπορεί να υλοποιηθεί με τη χρήση ορισμένων αποπλεκτών (demuxers). Η έξοδος δηλαδή των κελιών των ακριανών θέσεων του πίνακα δε συνδέεται απευθείας σε κάποια άλλα κελιά αλλά τροφοδοτείται στην είσοδο ενός demuxer, ο οποίος στη συνέχεια επιλέγει σε ποια έξοδό του θα οδηγήσει το σήμα αυτό. Η επιλογή θα είναι μία ακόμα παράμετρος για διαμόρφωση του συστήματος.

5.3. Σύστημα Μνημών

Το επαναδιατάξιμο περιφερειακό που σχεδιάστηκε έχει δύο εσωτερικές μονάδες μνήμης, μία μνήμη για την αποθήκευση δεδομένων επεξεργασίας και μία μνήμη για αποθήκευση δεδομένων διαμόρφωσης, συγκεκριμένα λέξεων διαμόρφωσης. Οι δύο αυτές μνήμες παρέχουν τη δυνατότητα εγγραφής τους από το master του συστήματος (στη συγκεκριμένη περίπτωση από τον πυρήνα επεξεργαστή ARM). Ωστόσο δεν υπάρχει δυνατότητα εγγραφής από τα επαναδιατάξιμα κελιά. Επίσης, η δυνατότητα ανάγνωσης παρέχεται μόνο από την πλευρά των κελιών και όχι από αυτή του επεξεργαστή. Υπάρχουν ακόμα τα αρχεία καταχωρητών (Register File - RF), που βρίσκονται μέσα σε κάθε επαναδιατάξιμο κελί του πίνακα. Τα RFs αυτά λειτουργούν σαν τοπικές μνήμες κάθε κελιού. Παρακάτω θα δοθούν αναλυτικές πληροφορίες για τις δύο μνήμες του περιφερειακού και για το πως αυτές συνεργάζονται με τον επανδιατάξιμο πίνακα και το master του συστήματος.

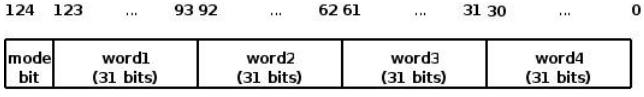
5.3.1.Μνήμη Διαμόρφωσης

Στη μνήμη αυτή αποθηκεύονται οι λέξεις διαμόρφωσης, οι οποίες αν εφαρμοστούν στα επαναδιατάξιμα κελιά του πίνακα πραγματοποιούν υπολογισμούς για την υλοποίηση ενός συγκεκριμένου αλγορίθμου. Αποτελείται από 129 θέσεις μνήμης με μέγεθος 125 bits η κάθε μία. Το συγκεκριμένο μέγεθος της κάθε θέσης μνήμης είναι τόσο, όσο χρειάζεται για να αποθηκευτούν 4 λέξεις διαμόρφωσης (των 31 bits η κάθε μία) σε κάθε θέση και επιπλέον ένα bit ελέγχου. Στη συνέχεια ερμηνεύεται το παραπάνω χαρακτηριστικό της μνήμης.



Σχήμα 5.15: Η διασύνδεση της μνήμης διαμόρφωσης με τον επαναδιατάξιμο πίνακα

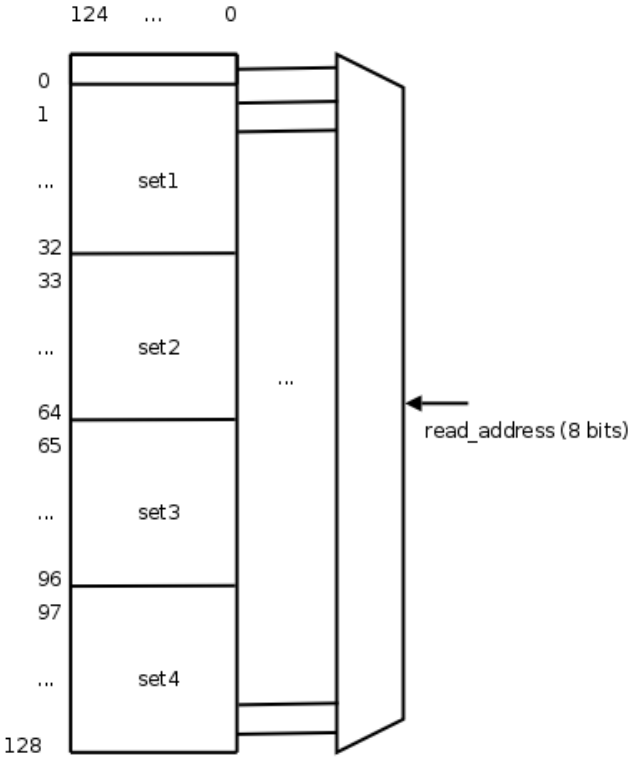
Όπως έχει ήδη αναφερθεί, οι λέξεις διαμόρφωσης μεταφέρονται στον επαναδιατάξιμο πίνακα είτε ανά στήλες είτε ανά σειρές, δηλαδή η ίδια λέξη διαμορφώνει όλα τα κελιά μιας στήλης ή μιας σειράς του πίνακα. Υπάρχουν δηλαδή δύο είδη λειτουργιών του επαναδιατάξιμου πίνακα, μία ανά γραμμές και μία ανά στήλες. Αυτά τα δύο είδη μπορεί να συνυπάρξουν κατά τη διάρκεια υλοποίησης ενός αλγορίθμου, ωστόσο σε κάθε κύκλο διαμόρφωσης πρέπει να χρησιμοποιείται μόνο το ένα από τα δύο. Από τα παραπάνω συμπεραίνεται ότι σε κάθε κύκλο διαμόρφωσης, δηλαδή σε κάθε κύκλο που μια καινούρια λέξη διαμόρφωσης μεταφέρεται στον πίνακα, πρέπει να υπάρχουν 4 διαφορετικές λέξεις διαμόρφωσης. Ο αριθμός αυτός προκύπτει από το μέγεθος του πίνακα των επαναδιατάξιμων κελιών, αφού υπάρχουν 4 διαφορετικές στήλες/γραμμές σε αυτόν. Για το λόγο αυτό κάθε θέση μνήμης πρέπει να περιέχει 4 διαφορετικές λέξεις διαμόρφωσης, μία για κάθε στήλη/γραμμή του πίνακα. Δηλαδή σε κάθε διαφορετικό κύκλο διαμόρφωσης μεταφέρεται στον επαναδιατάξιμο πίνακα μια ολόκληρη θέση από τη μνήμη διαμόρφωσης. Τέλος, η επιλογή μεταξύ των δύο ειδών λειτουργίας για κάθε κύκλο διαμόρφωσης γίνεται από το πιο σημαντικό bit της κάθε θέσης μνήμης. Αν αυτό είναι ίσο με 0 τότε έχουμε λειτουργία ανά γραμμές -οπότε οι λέξεις μοιράζονται στις τέσσερις γραμμές του πίνακα- ενώ αν είναι ίσο με 1 τότε έχουμε λειτουργία ανά στήλες -οπότε οι λέξεις μοιράζονται στις τέσσερις στήλες του πίνακα. Το γεγονός αυτό επιτρέπει στο σχεδιαστή να καθορίζει ανά κύκλο ποιο είδος λειτουργίας θέλει να χρησιμοποιήσει. Στο Σχήμα 5.16 φαίνεται σχηματικά μια θέση μνήμης της μνήμης διαμόρφωσης.



Σχήμα 5.16: Μια θέση της μνήμης διαμόρφωσης

Σε κάθε κύκλο διαμόρφωσης μεταφέρεται στον επαναδιατάξιμο πίνακα ένα σύνολο από 4 λέξεις διαμόρφωσης σαν ένα ενιαίο δεδομένο 125 bits συνολικά (μαζί με το bit επιλογής λειτουργίας), μέσω ενός τοπικού διαδρόμου (context bus). Αυτή η μεταφορά γίνεται στις θετικές ακμές του ρολογιού του συστήματος. Στην έξοδο του διαδρόμου δεδομένων στην πλευρά του επαναδιατάξιμου πίνακα τα δεδομένα διαμοιράζονται ώστε να σχηματιστούν σωστά οι 4 λέξεις διαμόρφωσης. Στη συνέχεια γίνεται η επιλογή για το πού θα σταλεί η κάθε λέξη. Στην πραγματικότητα οι 4 λέξεις διαμόρφωσης τροφοδοτούνται σε 4 αποπλέκτες, οι οποίοι χρησιμοποιώντας το bit επιλογής λειτουργίας σαν την είσοδο επιλογής τους, μεταφέρουν τις λέξεις ανά στήλες ή ανά γραμμές (Σχήμα 5.15). Για παράδειγμα αν το bit επιλογής είναι 0, τότε ο πρώτος αποπλέκτης οδηγεί τη λέξη που υπάρχει στην είσοδό του στην πρώτη γραμμή του πίνακα (σε όλα τα κελιά της), ενώ αν το bit επιλογής είναι 1, τότε η λέξη διαμόρφωσης οδηγείται στην πρώτη στήλη του πίνακα (σε όλα τα κελιά της).

Η μνήμη διαμόρφωσης δεν είναι ενιαία από πλευράς περιεχομένου. Για την ακρίβεια έχει χωριστεί σε 4 διαφορετικά σύνολα των 32 θέσεων μνήμης το κάθε ένα (4x32=128 θέσεις μνήμης συνολικά), τα οποία έχουν συνεχόμενη διευθυνσιοδότηση. Η μία επιπλέον θέση μνήμης που περισσεύει από τις συνολικά 129 που υπάρχουν στη μνήμη χρησιμοποιείται για έναν ειδικό σκοπό. Περισσότερες πληροφορίες για τη θέση αυτή της μνήμης δίνονται αργότερα στην παρούσα ενότητα. Στο Σχήμα 5.17 φαίνεται ο διαχωρισμός αυτός της μνήμης διαμόρφωσης.



Σχήμα 5.17: Τα σύνολα της μνήμης διαμόρφωσης

Κάθε σύνολο της μνήμης έχει αποθηκευμένο και ένα διαφορετικό περιεχόμενο διαμόρφωσης. Δηλαδή κάθε σύνολο έχει πληροφορίες για την πραγματοποίηση διαφορετικών κομματιών ενός ίδιου αλγορίθμου ή για την υλοποίηση εντελώς ανεξάρτητων μεταξύ τους αλγορίθμων. Κάθε χρονική στιγμή μπορεί να χρησιμοποιείται μόνο ένα σύνολο από τη μνήμη διαμόρφωσης, ωστόσο το χαρακτηριστικό αυτό δίνει δύο πλεονεκτήματα στο επαναδιατάξιμο σύστημα:

- Η διάσπαση ενός αλγορίθμου σε τμήματα ανεξάρτητων υπολογισμών, δίνει τη δυνατότητα καλύτερης συνεργασίας μεταξύ του επαναδιατάξιμου περιφερειακού και του επεξεργαστή του συστήματος και ευελιξίας στο σχεδιασμό. Στο τέλος της εκτέλεσης του περιεχομένου κάθε συνόλου της μνήμης, δίνεται η δυνατότητα στον επεξεργαστή να προσπελάσει κάποια αποτελέσματα του υπολογισμού που επιτελέστηκε, είτε απευθείας από τις εξόδους κάποιων κελιών είτε από τα εσωτερικά αρχεία καταχωρητών των κελιών. Τα αποτελέσματα αυτά ίσως είναι απαραίτητο να χρησιμοποιηθούν περαιτέρω από τους υπολογισμούς που πιθανώς εκτελεί ο επεξεργαστής, γι`αυτό και το συγκεκριμένο χαρακτηριστικό της μνήμης διαμόρφωσης είναι πολύ χρήσιμο σε αυτό τον τομέα. Μπορεί δηλαδή ο σχεδιαστής να χωρίσει τον αλγόριθμο που εκτελείται στο επαναδιατάξιμο περιφερειακό σε κομμάτια, ανάλογα με το πότε απαιτείται συνεργασία με τον επεξεργαστή. Εξάλλου το επαναδιατάξιμο περιφερειακό δεν έχει τη δυνατότητα να γράψει απευθείας στην εξωτερική μνήμη του συστήματος, ακριβώς επειδή είναι ένας "σκλάβος" και πρέπει να ζητηθεί να αναλάβει ο επεξεργαστής (ο master) τη διαδικασία της ανάγνωσης από αυτό. Όταν τελειώσει η εκτέλεση των εντολών σε κάποιο σύνολο της μνήμης, το περιφερειακό ζητά διακοπή στο master, ώστε να διαβάσει αν τυχόν θέλει αποτελέσματα από τους υπολογισμούς και να καθορίσει σε ποιο σύνολο πρέπει να περάσει ο έλεγχος του προγράμματος.

- Δίνεται η δυνατότητα αποθήκευσης στη μνήμη διαμόρφωσης πολλαπλών διαφορετικών ομάδων διαμόρφωσης, που μπορεί να ανήκουν σε εντελώς ανεξάρτητους αλγορίθμους. Έτσι υπάρχει δυνητικά πολλαπλό υλικό σε ένα μόνο κομμάτι υλικού, το οποίο μπορεί να εναλλάσσεται ανάλογα με τις οδηγίες του master.

Επιλογή διεύθυνσης ανάγνωσης

Η διεύθυνση ανάγνωσης από τη μνήμη διαμόρφωσης, δηλαδή η θέση μνήμης η οποία είναι η επόμενη που θα μεταφερθεί από τη μνήμη προς τον επαναδιατάξιμο πίνακα, προσδιορίζεται από ένα σύνθετο κομμάτι κυκλώματος.

Υπάρχουν τέσσερις πιθανές διευθύνσεις που θα αποτελέσουν την επόμενη διεύθυνση ανάγνωσης:

- Μια διεύθυνση που παρέχεται απευθείας από το master (επεξεργαστής

ARM). Πρέπει να υπάρχει η δυνατότητα ο master να μπορεί να προσδιορίσει τη θέση μνήμης στην οποία πρέπει να μεταφερθεί η εκτέλεση. Ωστόσο δε χρησιμοποιείται για να επεμβαίνει ο master για να καθορίσει διευθύνσεις εσωτερικά σε ένα σύνολο, αλλά για να καθορίσει ποιο σύνολο πρέπει να εκτελεστεί στη συνέχεια. Παρέχει δηλαδή την αρχική διεύθυνση του συνόλου, όπου περιέχεται το κομμάτι του προγράμματος που πρέπει να εκτελεστεί στη συνέχεια. Η δυνατότητα αυτή χρησιμοποιείται κυρίως στην αρχή της λειτουργίας του συστήματος, για να προσδιοριστεί το πρώτο προς εκτέλεση σύνολο, αλλά και μετά το τέλος κάθε σύνολο, όταν και πρέπει να προσδιοριστεί ποιο σύνολο θα εκτελεστεί στη συνέχεια.

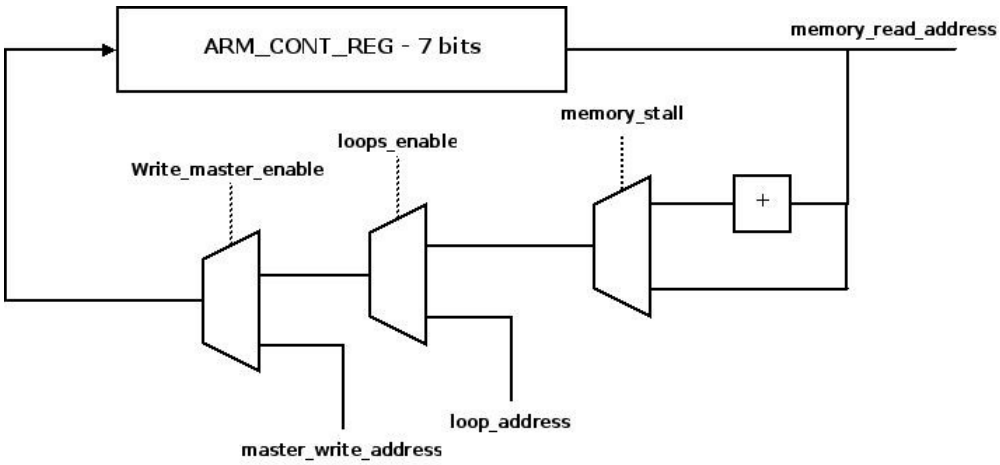
- Η επόμενη διεύθυνση στη μνήμη, σε σχέση με την τρέχουσα θέση μνήμης που διαμορφώνει τον πίνακα. Αυτή είναι και η κανονική περίπτωση λειτουργίας κατά τη διάρκεια εκτέλεσης ενός συνόλου της μνήμης. Συνήθως η επόμενη προς εκτέλεση ομάδα λέξεων διαμόρφωσης βρίσκεται στην επόμενη θέση της μνήμης.

- Η διεύθυνση στη μνήμη, που αντιστοιχεί στην τρέχουσα θέση μνήμης που διαμορφώνει τον πίνακα. Αυτή η περίπτωση χρησιμοποιείται αν για οποιοδήποτε λόγο πρέπει το σύστημα να τεθεί σε κατάσταση αναστολής, δηλαδή η εκτέλεση των λέξεων διαμόρφωσης να σταματήσει σε μια συγκεκριμένη θέση της μνήμης διαμόρφωσης. Συνήθως, κατά τη σχεδίαση, στη θέση αυτή όλα τα κελιά του επαναδιατάξιμου πίνακα βρίσκονται σε κατάσταση stall, ώστε να μην πραγματοποιούνται άσκοποι υπολογισμοί.

- Μια διεύθυνση που παρέχεται εσωτερικά από το ίδιο το επαναδιατάξιμο σύστημα, η οποία χρησιμοποιείται για την εσωτερική υλοποίηση μέσα στο επαναδιατάξιμο περιφερειακό βρόχων επανάληψης. Οι βρόχοι αυτοί γίνονται χωρίς την παρέμβαση του master. Ουσιαστικά η διεύθυνση αυτή είναι η διεύθυνση στην οποία επιστρέφει η εκτέλεση μέσα σε ένα σύνολο της μνήμης, στα πλαίσια του βρόχου που υλοποιείται. Παρέχεται μέσω της λέξης διαμόρφωσης. Περισσότερες πληροφορίες στην επόμενη υποενότητα, περί του μηχανισμού εκτέλεσης βρόχων επανάληψης.

Επομένως το κύκλωμα που προσδιορίζει την επόμενη διεύθυνση ανάγνωσης στη μνήμη διαμόρφωσης πρέπει να λαμβάνει υπόψη του και τις τρεις παραπάνω περιπτώσεις (φαίνεται σχηματικά στο Σχήμα 5.18).

Στο σχήμα αυτό φαίνεται πως το κύριο συστατικό είναι ένας καταχωρητής, ο οποίος και περιέχει την τρέχουσα διεύθυνση της μνήμης διαμόρφωσης που χρησιμοποιείται. Ο καταχωρητής αυτός συνδέεται απευθείας με τη μνήμη διαμόρφωσης και την τροφοδοτεί με τη διεύθυνση της θέσης μνήμης που πρέπει να μεταφερθεί στον επαναδιατάξιμο πίνακα. Ο καταχωρητής εγγράφεται στις αρνητικές ακμές του ρολογιού, με τη νέα τιμή να προκύπτει από το υπόλοιπο κομμάτι του κυκλώματος επιλογής.



Σχήμα 5.18: Κύκλωμα επιλογής της επόμενης διεύθυνσης ανάγνωσης

Συγκεκριμένα υπάρχει ένας πολυπλέκτης, ο οποίος επιλέγει αν η νέα διεύθυνση ανάγνωσης παρέχεται από το master ή όχι. Αν ναι, τότε η νέα τιμή για εγγραφή στον καταχωρητή θα δίνεται από την είσοδο του πολυπλέκτη που αντιστοιχεί σε δεδομένα από το κυρίως σύστημα και το master. Η είσοδος επιλογής του συγκεκριμένου πολυπλέκτη ελέγχεται από τον ίδιο το master, οποίος στέλνει προς το περιφερειακό σήμα ότι θέλει να πραγματοποιήσει εγγραφή προς το συγκεκριμένο καταχωρητή για την επόμενη διεύθυνση της μνήμης διαμόρφωσης.

Αν αντίθετα η διεύθυνση δεν παρέχεται από το master, δηλαδή η είσοδος ελέγχου από την πλευρά του είναι απενεργοποιημένη (τιμή 0), τότε η νέα τιμή για εγγραφή στον καταχωρητή προέρχεται από την έξοδο ενός άλλου πολυπλέκτη. Ο τελευταίος επιλέγει αν η νέα διεύθυνση θα είναι διεύθυνση άλματος για την υλοποίηση βρόχου επανάληψης ή όχι. Ειδικότερα, αν πρέπει να γίνει άλμα στα πλαίσια ενός βρόχου επανάληψης τότε η επόμενη διεύθυνση για τη μνήμη διαμόρφωσης θα είναι η διεύθυνση άλματος. Στην περίπτωση αυτή η είσοδος ελέγχου του δεύτερου πολυπλέκτη είναι 1 και η διεύθυνση άλματος παρέχεται από τη λέξη διαμόρφωσης.

Αν όμως η εκτέλεση των βρόχων επανάληψης έχει ολοκληρωθεί και η συγκεκριμένη λειτουργία είναι απενεργοποιημένη, τότε η επόμενη διεύθυνση θα είναι είτε η επόμενη της τρέχουσας διεύθυνσης είτε ίδια με την τρέχουσα διεύθυνση, κάτι το οποίο επιλέγεται επίσης από έναν άλλο πολυπλέκτη. Το σήμα ελέγχου για το συγκεκριμένο πολυπλέκτη παρέχεται από ένα bit της λέξης διαμόρφωσης, όταν αυτή βρίσκεται σε κατάσταση λειτουργίας stall. Για τον προσδιορισμό της επόμενης από την τρέχουσα διεύθυνση χρησιμοποιείται ένας αθροιστής, ο οποίος προσθέτει στο περιεχόμενο του καταχωρητή τον αριθμό ένα.

Θα πρέπει να τονιστεί, τέλος, ότι η μοναδική έξοδος του καταχωρητή δεν είναι απευθείας συνδεδεμένη με το περιεχόμενό του. Σε ορισμένες περιπτώσεις, οι οποίες συναντώνται κατά την έναρξη λειτουργίας του πίνακα από περίοδο stall, απαιτείται συγχρονισμός μεταξύ των δύο μνημών του συστήματος. Συγκεκριμένα, υπάρχει μια επιπλέον είσοδος στον καταχωρητή, η είσοδος

συγχρονισμού. Όταν η είσοδος αυτή είναι 0 τότε η έξοδος του καταχωρητή αποσυνδέεται από το περιεχόμενό του. Για την ακρίβεια η έξοδος παραμένει στην τιμή που είχε πριν η είσοδος συγχρονισμού γίνει 0 και δεν αλλάζει, ανεξάρτητα από τις αλλαγές που γίνονται στο περιεχόμενο του καταχωρητή. Όταν όμως η είσοδος συγχρονισμού είναι 1 τότε η έξοδος του καταχωρητή είναι απευθείας συνδεδεμένη με το περιεχόμενό του, ακολουθώντας πλήρως όσες αλλαγές γίνονται σε αυτό (πάντα βέβαια η νέα τιμή μεταφέρεται στην έξοδο σε κάθε αρνητική ακμή του ρολογιού).

Υλοποίηση εσωτερικών βρόχων επανάληψης

Η σχεδίαση του επαναδιατάξιμου συστήματος παρέχει τη δυνατότητα στο επαναδιατάξιμο περιφερειακό να εκτελεί βρόχους επανάληψης στο περιεχόμενο διαμόρφωσης, χωρίς την παρέμβαση του επεξεργαστή. Το χαρακτηριστικό αυτό είναι ιδιαίτερα αποδοτικό από πλευράς ταχύτητας και ταυτόχρονα δίνει ισχυρές επιλογές στη σχεδίαση και στην υλοποίηση αλγορίθμων. Από την πλευρά του επεξεργαστή αρκεί και μόνο ο προσδιορισμός του πλήθους των επαναλήψεων που πρέπει να γίνουν και η διεύθυνση στην οποία θα επιστρέφει -προς τα πίσω- η εκτέλεση του προγράμματος.

Υπάρχουν δύο σημεία όπου εντοπίζεται η εμφάνιση του χαρακτηριστικού αυτού. Κατά πρώτον υπάρχει ένας καταχωρητής μέσα στο επαναδιατάξιμο περιφερειακό στον οποίο υπάρχει το πλήθος των εναπομείναντων επαναλήψεων του βρόχου, ο καταχωρητής επαναλήψεων. Αυτός εγγράφεται αρχικά από το master, στην αρχή εκτέλεσης του κάθε συνόλου της μνήμης διαμόρφωσης, ώστε να καθοριστεί το συνολικό πλήθος των επαναλήψεων του βρόχου που απαιτούνται. Στη συνέχεια, με το τέλος της εκτέλεσης κάθε επανάληψης το περιεχόμενο του καταχωρητή αυτού μειώνεται κατά ένα. Όσο ο καταχωρητής περιέχει αριθμό διάφορο του μηδενός, τότε οι επαναλήψεις επιτρέπονται. Όταν όμως μηδενιστεί, η εκτέλεση των επαναλήψεων απενεργοποιείται και η εκτέλεση του προγράμματος συνεχίζεται με θέσεις μνήμης εκτός του βρόχου.

Τα εξωτερικά σήματα εισόδου και εξόδου του συγκεκριμένου καταχωρητή είναι τα εξής:

- *data_in* (9 bits) : Πρόκειται για το σήμα δεδομένων εισόδου. Η συγκεκριμένη είσοδος ελέγχεται από το master και χρησιμοποιείται όταν ο τελευταίος επιθυμεί να εγγράψει στον καταχωρητή το πλήθος των επαναλήψεων που πρέπει να εκτελεστούν.
- *write_en* (1 bit) : Το σήμα αυτό είναι σήμα εισόδου ελέγχου και καθορίζει πότε θα γίνει εγγραφή στον καταχωρητή. Ειδικότερα, όταν έχει τιμή '1' τότε δηλώνεται ότι ο master θέλει να εγγράψει τον καταχωρητή αλμάτων. Όταν έχει τιμή '0', όμως, τότε δεν υπάρχει λειτουργία εγγραφής.
- *control_bits* (2 bits) : Το συγκεκριμένο σήμα εισόδου καθορίζει πότε πρέπει να μειωθεί η τιμή του περιεχομένου του καταχωρητή και ταυτόχρονα δίνει σήμα στο σύστημα ότι πρέπει να εκτελεστεί άλμα επανάληψης. Συγκεκριμένα, όταν και τα δύο bits της εισόδου αυτής έχουν

τιμή '1', τότε το περιεχόμενο του καταχωρητή -δηλαδή το εναπομείνων πλήθος βρόχων επανάληψης- μειώνεται κατά ένα και η *έξοδος loop*_en παίρνει την τιμή '1', οπότε γίνεται ακόμα μία επανάληψη. Τα δύο αυτά bits προέρχονται απευθείας από ορισμένα bits της λέξης διαμόρφωσης και συγκεκριμένα από τα bits 0 και 3.

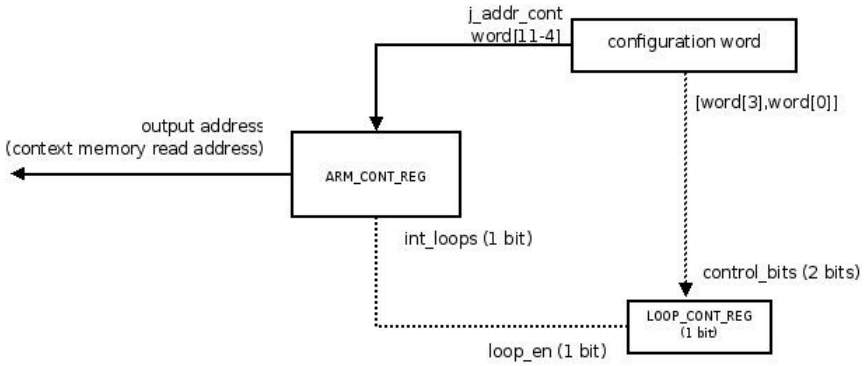
- *loop_en* (1 bit) : Το συγκεκριμένο σήμα είναι σήμα εξόδου και όταν πάρει την τιμή 1, τότε εκτελείται ένα άλμα επανάληψης με το να οριστεί η επόμενη διεύθυνση ανάγνωσης της μνήμης διαμόρφωσης (στον καταχωρητή επόμενης διεύθυνσης ARM_CONT_REG) στην τιμή της διεύθυνσης άλματος επανάληψης, που προέρχεται από τη λέξηδιαμόρφωσης.

Το δεύτερο σημείο στο επαναδιατάξιμο περιφερειακό, το οποίο σχετίζεται με την υλοποίηση των εσωτερικών βρόχων επανάληψης είναι ο καταχωρητής επόμενης διεύθυνσης ανάγνωσης για τη μνήμη διαμόρφωσης (ο ARM_CONT_REG). Όπως περιγράφηκε και στη προηγούμενη υποενότητα "Επιλογή διεύθυνσης ανάγνωσης", ανάμεσα στις πιθανές διευθύνσεις που θα αποτελέσουν την επόμενη διεύθυνση ανάγνωσης υπάρχει και μία διεύθυνση η οποία παρέχεται από το περιεχόμενο της λέξης διαμόρφωσης. Η διεύθυνση αυτή αποτελεί τη διεύθυνση στην οποία θα επιστρέψει η εκτέλεση του προγράμματος, ώστε να υλοποιηθεί ο βρόχος επανάληψης.

Συνοπτικά, οι μονάδες που σχετίζονται με τα άλματα των βρόχων επανάληψης είναι ο καταχωρητής επανάληψης, η λέξη διαμόρφωσης και ο καταχωρητής επόμενης διεύθυνσης ανάγνωσης της μνήμης.

- Ο καταχωρητής επανάληψης περιέχει το πλήθος των βρόχων που πρέπει -ακόμα- να γίνουν και με δικό του σήμα ελέγχου, το οποίο συνδέεται με κατάλληλη είσοδο του καταχωρητή επόμενης διεύθυνσης, η διεύθυνση ανάγνωσης από τη μνήμη γίνεται ίδια με την πρώτη διεύθυνση του βρόχου επανάληψης.
- Ο καταχωρητής επόμενης διεύθυνσης είναι η μονάδα στην οποία υλοποιούνται επί της ουσίας οι βρόχοι επανάληψης, καθώς εδώ τίθεται η επόμενη διεύθυνση ανάγνωσης ίση με τη διεύθυνση άλματος, μετά από κατάλληλο έλεγχο από τον καταχωρητή επανάληψης.
- Τέλος, η λέξη διαμόρφωσης παρέχει τη διεύθυνση στη μνήμη στην οποία πρέπει να μεταβεί η εκτέλεση για να υλοποιηθεί η επανάληψη και καθορίζει το πότε πρέπει να γίνει αυτή η μετάβαση.

Στο Σχήμα 5.19 απεικονίζεται το υλικό που σχετίζεται με τους εσωτερικούς βρόχους.

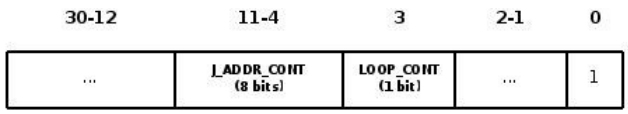


Σχήμα 5.19: Υλοποίηση εσωτερικών βρόχων για τη μνήμη διαμόρφωσης

Ο μηχανισμός των εσωτερικών αλμάτων επανάληψης στη μνήμη διαμόρφωσης έχει ως εξής. Μετά την τελευταία εντολή του βρόχου επανάληψης τοποθετείται στην επόμενη θέση της μνήμη διαμόρφωσης ένα ειδικό περιεχόμενο. Στη λέξη διαμόρφωσης, όταν αυτή βρίσκεται στην κατάσταση λειτουργίας stall, έχουν καθοριστεί κάποια bits να καθορίζουν αν πρέπει να γίνει άλμα στα πλαίσια ενός βρόχου επανάληψης και τη διεύθυνση στην οποία πρέπει να γίνει το άλμα αυτό. Έτσι στο τέλος του βρόχου επανάληψης τοποθετούνται λέξεις επανάληψης, σε κατάσταση λειτουργίας stall, οι οποίες έχουν στο κατάλληλο πεδίο την ένδειξη ότι στον αμέσως επόμενο κύκλο πρέπει να γίνει άλμα. Συγκεκριμένα, η είσοδος ελέγχου *control_*bits του καταχωρητή επαναλήψεων συνδέεται με τα bits 0 και 3 της λέξης διαμόρφωσης που αντιστοιχεί στο κελί. Το bit3 της λέξης διαμόρφωσης, όταν γίνει '1', καθορίζει ότι πρέπει να γίνει άλμα εσωτερικού βρόχου επανάληψης, με την προυπόθεση ότι ταυτόχρονα το bit0 έχει την τιμή '1', δηλαδή η λέξη είναι σε κατάσταση stall.

Στη συνέχεια γίνεται έλεγχος για το αν ο καταχωρητής επαναλήψεων έχει μηδενιστεί ή όχι, λαμβάνοντας υπόψη και την επανάληψη που μόλις τελείωσε. Αν αυτός περιέχει το μηδέν, τότε η επόμενη θέση μνήμης για μεταφορά προς τον επαναδιατάξιμο πίνακα είναι η θέση μνήμης που ακολουθεί την τρέχουσα. Ωστόσο αν ο καταχωρητής επαναλήψεων δεν είναι μηδέν, τότε γίνεται και πάλι επανάληψη. Δηλαδή δίνεται σήμα στο κύκλωμα επιλογής της επόμενης διεύθυνσης ότι η επόμενη διεύθυνση που πρέπει να σταλεί προς τη μνήμη διαμόρφωσης είναι η διεύθυνση άλματος, η οποία παρέχεται από τη λέξη διαμόρφωσης.

Στο Σχήμα 5.20 φαίνονται τα πεδία της λέξης διαμόρφωσης, τα οποία σχετίζονται -και χρησιμοποιούνται- με το μηχανισμό των εσωτερικών επαναλήψεων.



LOOP_CONT (1 bit) : Η τιμή 1 σηματοδοτεί άλμα για βρόχο στη μνήμη διαμόρφωσης, αν ταυτόχρονα [LOOP_REG] > 0
Η τιμή 0 σημαίνει ότι δεν πρέπει να γίνει άλμα

I_ADDR_CONT (8 bits) : Παρέχει τη διεύθυνση άλματος για βρόχους στη μνήμη διαμόρφωσης

Σχήμα 5.20: Πεδία στη λέξη διαμόρφωσης που σχετίζονται με τα εσωτερικά άλματα στη μνήμη διαμόρφωσης

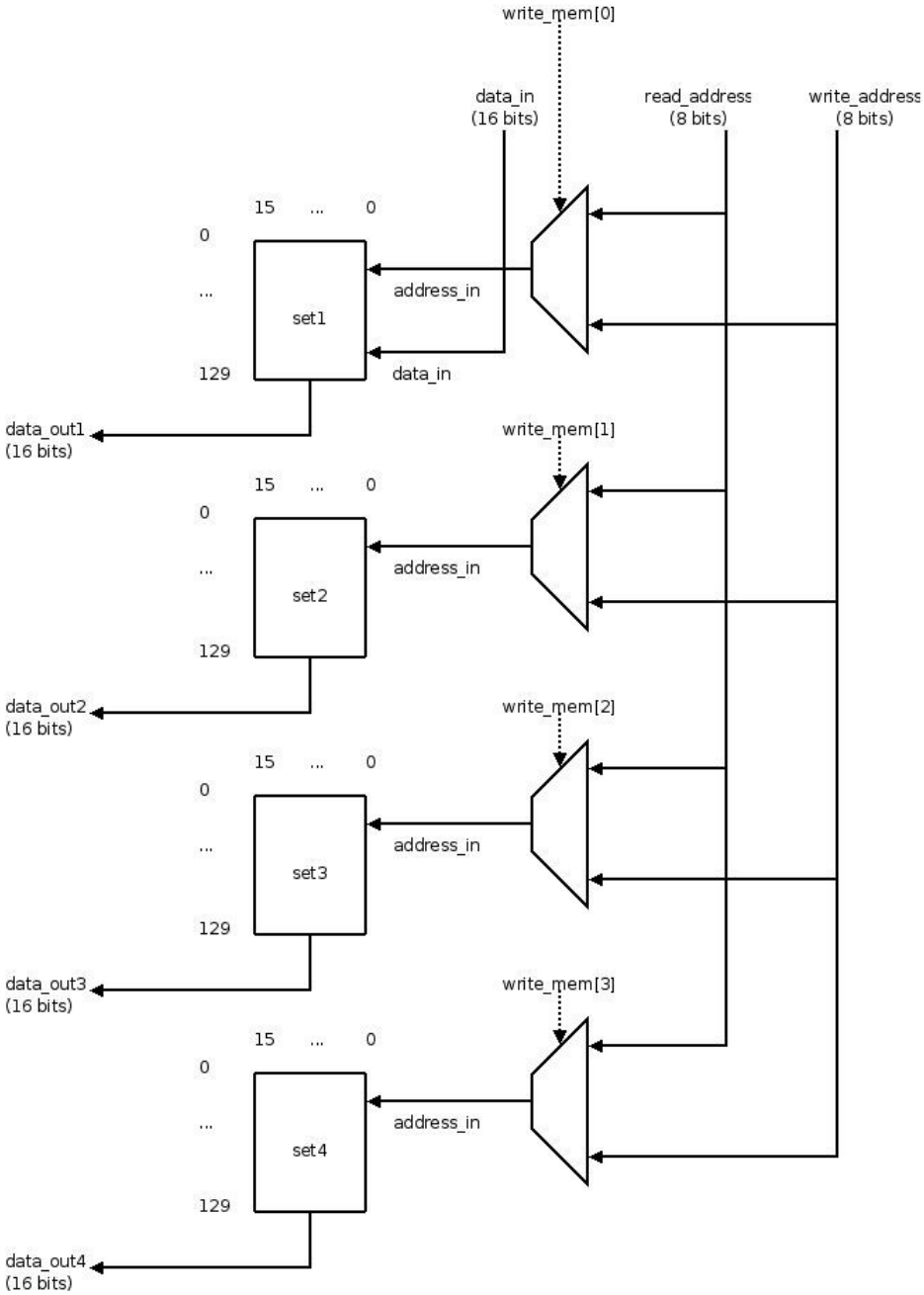
5.3.2.Μνήμη Δεδομένων

Στη μνήμη αυτή αποθηκεύονται δεδομένα που πρόκειται να χρησιμοποιηθούν για επεξεργασία από τα επαναδιατάξιμα κελιά. Υπάρχουν συνολικά 4 σύνολα μνήμης, με το μέγεθος του καθενός να ανέρχεται στις 130 θέσεις μνήμης των 16 bits η κάθε μία. Η μνήμη είναι αποκλειστικά μνήμη ανάγνωσης και όχι εγγραφής για την πλευρά του επαναδιατάξιμου πίνακα. Αντίθετα ο master του επαναδιατάξιμου συστήματος, δηλάδη στην προκειμένη περίπτωση ο επεξεργαστικός πυρήνας ARM, έχει τη δυνατότητα εγγραφής στη μνήμη αλλά όχι ανάγνωσης.

Ο διαχωρισμός της μνήμης σε 4 σύνολα, τα οποία δεν έχουν συνεχόμενη διευθυνσιοδότηση, αντικατοπτρίζει το γεγονός ότι υπάρχουν συνολικά 4 ομάδες επεξεργασίας από επαναδιατάξιμα κελιά, είτε ο πίνακας λειτουργεί ανά στήλες είτε ανά γραμμές. Έτσι το πρώτο σύνολο αντιστοιχεί στην πρώτη γραμμή (αλλά και στην πρώτη στήλη) του πίνακα, το δεύτερο στη δεύτερη κ.ό.κ. Για την επικοινωνία μεταξύ της μνήμης δεδομένων και του επαναδιατάξιμου πίνακα υπάρχει ένας τοπικός διάδρομος δεδομένων με πλάτος 16 bits, ο οποίος μεταφέρει δεδομένα μόνο από τη μνήμη προς τον πίνακα.

Ανάγνωση/Εγγραφή της μνήμης

Ο τρόπος ανάγνωσης/εγγραφής της μνήμης έχει σχεδιαστεί να υποστηρίζει ισχυρές -σχεδιαστικά- δυνατότητες και είναι αρκετά πολύπλοκος. Το Σχήμα 5.21 δίνει μια διαγραμματική παρουσίαση του μηχανισμού για την ανάγνωση και εγγραφή.



Σχήμα 5.21: Μηχανισμός ανάγνωσης / εγγραφής της μνήμης δεδομένων

Όπως φαίνεται, στη μνήμη παρέχονται δύο διευθύνσεις, μία διεύθυνση ανάγνωσης και μία διεύθυνση εγγραφής, με μέγεθος 7 bits η κάθε μία. Ακόμα υπάρχει μια είσοδος δεδομένων εγγραφής μήκους 16 bits, 4 είσοδοι ελέγχου εγγραφής/ανάγνωσης των μνημών με μέγεθος 1 bits η κάθε μία και τέλος τέσσερις έξοδοι δεδομένων των 16 bits.

Οι είσοδοι ελέγχου ανάγνωσης/εγγραφής επιλέγουν -ανεξάρτητα για κάθε μνήμη- μεταξύ των λειτουργιών ανάγνωσης και εγγραφής. Αν σε μια μνήμη η είσοδος αυτή είναι 0 τότε γίνεται ανάγνωση από τη μνήμη αυτή, δηλαδή στην έξοδο δεδομένων μεταφέρεται η θέση μνήμης που αντιστοιχεί στη διεύθυνση ανάγνωσης. Αν αντίθετα η είσοδος ελέγχου είναι 1, τότε γίνεται εγγραφή στη

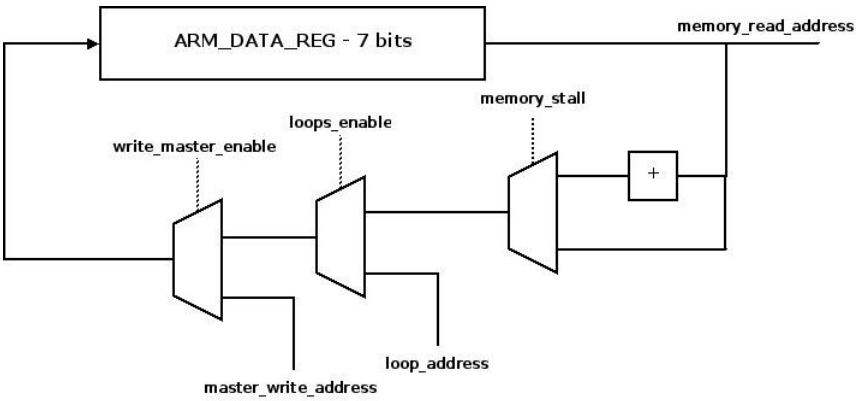
μνήμη αυτή, δηλαδή τα δεδομένα στην είσοδο δεδομένων εγγραφής εγγράφονται στη θέση μνήμης που αντιστοιχεί στη διεύθυνση εγγραφής.

Να σημειωθεί ότι οι διευθύνσεις ανάγνωσης και εγγραφής είναι κοινές και για τα 4 σύνολα της μνήμης και επιπλέον κοινή είναι και η είσοδος δεδομένων. Αυτό σημαίνει ότι και στις τέσσερις ομάδες επεξεργασίας του επαναδιατάξιμου πίνακα μεταφέρονται οι ίδιες θέσεις μνήμης από κάθε σύνολο. Από την άλλη όταν ο επεξεργαστής θέλει να γράψει στις μνήμες, η εγγραφή γίνεται χρησιμοποιώντας τα ίδια δεδομένα και την ίδια διεύθυνση εγγραφής. Η διαφοροποίηση στη διαδικασία εγγραφής έγκειται στις διαφορετικές εισόδους ελέγχου για κάθε σύνολο, με τη βοήθεια των οποίων ο επεξεργαστής μπορεί να γράψει μόνο στα σύνολα που έχει επιλέξει να γράψει.

Ο χρονισμός των διαδικασιών ανάγνωσης και εγγραφής είναι σχετικά απλός. Συγκεκριμένα και οι δύο διαδικασίες γίνονται στις θετικές ακμές του ρολογιού του συστήματος. Έτσι, σε κάθε θετική ακμή του ρολογιού τα κυκλώματα της μνήμης ελέγχουν την είσοδο ελέγχου. Αν αυτή είναι 0 τότε μεταφέρονται στο διάδρομο δεδομένων εξόδου προς τον επαναδιατάξιμο πίνακα τα δεδομένα της θέσης μνήμης που υποδεικνύεται από τη διεύθυνση ανάγνωσης. Αν η είσοδος ελέγχου είναι 1 τότε γίνεται εγγραφή των δεδομένων που βρίσκονται στην είσοδο δεδομένων στη θέση μνήμης που υποδεικνύεται από τη διεύθυνση εγγραφής.

Επιλογή διεύθυνσης ανάγνωσης

Η διεύθυνση ανάγνωσης από τη μνήμη δεδομένων, δηλαδή η θέση μνήμης η οποία είναι η επόμενη που θα μεταφερθεί από τη μνήμη προς τον επαναδιατάξιμο πίνακα, προσδιορίζεται από ένα σύνθετο κομμάτι κυκλώματος, το οποίο είναι πανομοιότυπο με εκείνο που προσδιορίζει τη διεύθυνση ανάγνωσης από τη μνήμη διαμόρφωσης. Το συγκεκριμένο κύκλωμα απεικονίζεται στο Σχήμα 5.22.



Σχήμα 5.22: Κύκλωμα για την επιλογή της διεύθυνσης ανάγνωσης για τη μνήμη δεδομένων

Οι λειτουργίες που υποστηρίζονται στη μνήμη δεδομένων, όσον αφορά στην

επόμενη διεύθυνση ανάγνωσης είναι ακριβώς οι ίδιες με εκείνες της μνήμης διαμόρφωσης. Συνεπώς όλα τα κυκλώματα που χρησιμοποιούνται για τον προσδιορισμό της επόμενης διεύθυνσης ανάγνωσης είναι ίδια με τα αντίστοιχα εκείνα της μνήμης διαμόρφωσης.

Υπάρχουν, λοιπόν, τέσσερις πιθανές διευθύνσεις που μπορεί να αποτελέσουν την επόμενη διεύθυνση ανάγνωσης:

- Μια διεύθυνση που παρέχεται απευθείας από το master (επεξεργαστής ARM). Η περίπτωση αυτή υποστηρίζει τη δυνατότητα να μπορεί ο master να προσδιορίσει επακριβώς τη θέση μνήμης από την οποία θα συνεχιστεί η ανάγνωση δεδομένων. Στην πραγματικότητα αυτή η δυνατότητα είναι αντίστοιχη με την περίπτωση που ο master δίνει τη διεύθυνση από την οποία πρέπει να συνεχιστεί η μεταφορά διευθύνσεων διαμόρφωσης. Όπως έχει αναφερθεί η περίπτωση αυτή συναντάται όταν τελειώσουν οι λέξεις διαμόρφωσης που υπάρχουν σε ένα σύνολο της μνήμης διαμόρφωσης, όταν και απαιτείται να καθοριστεί ποιο σύνολο θα εκτελεστεί στη συνέχεια, αλλά και στην αρχή της λειτουργίας του συστήματος, για να προσδιοριστεί το πρώτο σύνολο προς εκτέλεση. Αντιστοίχως, θα πρέπει ο master να μπορεί να προσδιορίσει σε ποιο σημείο της μνήμης δεδομένων υπάρχουν αποθηκευμένα τα δεδομένα που αντιστοιχούν στο σύνολο της μνήμης διαμόρφωσης που πρόκειται να εκτελεστεί στη συνέχεια.

- Η επόμενη διεύθυνση στη μνήμη σε σχέση με την τρέχουσα θέση μνήμης, της οποίας τα περιεχόμενα έχουν μεταφερθεί στον επαναδιατάξιμο πίνακα. Αυτή η περίπτωση χρησιμοποιείται όταν χρειάζεται να μεταφερθεί μια ομάδα δεδομένων επεξεργασίας στο πίνακα, η οποία βρίσκεται σε συνεχόμενες θέσεις στη μνήμη δεδομένων.

- Η διεύθυνση στη μνήμη που αντιστοιχεί στην τρέχουσα θέση μνήμης που διαμορφώνει τον πίνακα. Αυτή η περίπτωση χρησιμοποιείται αν για οποιοδήποτε λόγο πρέπει να σταματήσει η μεταφορά δεδομένων από τη μνήμη στον επαναδιατάξιμο πίνακα. Όπως διαφάνηκε από την παρουσίαση της διαδικασίας ανάγνωσης/εγγραφής της μνήμης δεδομένων προηγουμένως, δεν υπάρχει κάποια είσοδος ελέγχου η οποία να απενεργοποιεί τη λειτουργία της μνήμης προσωρινά. Έτσι, αν δεν υπήρχε η συγκεκριμένη δυνατότητα, δηλαδή η επόμενη διεύθυνση να είναι ίδια με την τρέχουσα διεύθυνση, σε κάθε κύκλο ρολογιού θα μεταφέρονταν θέσεις μνήμης προς τον επαναδιατάξιμο πίνακα, χωρίς πολλές από αυτές να χρειάζονται στην πραγματικότητα για επεξεργασία από τον πίνακα. Κάποια στιγμή κατά τη διάρκεια της εκτέλεσης του προγράμματος είναι πολύ πιθανό οι θέσεις της μνήμης δεδομένων να εξαντλούνταν και να απαιτούνταν η επέμβαση του επεξεργαστή, ο οποίος θα έπρεπε να δώσει μια νέα διεύθυνση για ανάγνωση. Με τη συγκεκριμένη επιλογή για την επόμενη διεύθυνση ανάγνωσης λοιπόν, λύνεται αποτελεσματικά το παραπάνω πρόβλημα. Επιπλέον παρέχεται η δυνατότητα στο σχεδιαστή να επιλέγει (με τη χρήση κατάλληλων λέξεων διαμόρφωσης) σε ποια σημεία

του προγράμματος θα ενεργοποιείται η ανάγνωση από τη μνήμη δεδομένων.

- Μια διεύθυνση που παρέχεται εσωτερικά από το ίδιο το επαναδιατάξιμο σύστημα, η οποία χρησιμοποιείται για την εσωτερική υλοποίηση εσωτερικά μέσα στο επαναδιατάξιμο περιφερειακό βρόχων επανάληψης. Οι βρόχοι αυτοί γίνονται χωρίς την παρέμβαση του master και έχουν τη ίδια ακριβώς λειτουργία με τους βρόχους που υλοποιούνται πάνω στα περιεχόμενα της μνήμης διαμόρφωσης. Η διεύθυνση αυτή είναι η διεύθυνση άλματος που απαιτείται για την υλοποίηση του βρόχου επανάληψης και παρέχεται από τη λέξη διαμόρφωσης. Περισσότερες πληροφορίες στην επόμενη υποενότητα, περί του μηχανισμού εκτέλεσης βρόχων επανάληψης.

Να σημειωθεί ότι το bit της λέξης διαμόρφωσης που ελέγχει την επιλογή μεταξύ της τρέχουσας διεύθυνσης ή της επόμενης της τρέχουσας είναι διαφορετικό από το bit που ελέγχει την αντίστοιχη λειτουργία για τη μνήμη διαμόρφωσης. Υπάρχει όμως και μία σημαντική διαφορά στο χρονισμό της συγκεκριμένης λειτουργίας ελέγχου, σε σχέση πάντα με την αντίστοιχη λειτουργία για τη μνήμη διαμόρφωσης. Στη μνήμη διαμόρφωσης η επιλογή μεταξύ της ίδιας ή της επόμενης διεύθυνσης είναι ευαίσθητη στο bit ελέγχου. Αν το bit αυτό είναι 1 τότε η επόμενη διεύθυνση παραμένει ίδια με την παρούσα, ενώ όταν γίνει 0 τότε η επόμενη διεύθυνση αλλάζει. Στη μνήμη δεδομένων, αντίθετα, έχει υλοποιηθεί ένας εσωτερικός καταχωρητής. Αυτός εγγράφεται με την τιμή του bit ελέγχου, όταν το bit0 της λέξης διαμόρφωσης είναι στο 1. Η εγγραφή διατηρείται μέχρι την επόμενη φορά που η λέξη διαμόρφωσης έχει στο bit0 την τιμή 1. Αν το περιεχόμενο του καταχωρητή αυτού είναι 1, τότε η μεταφορά από τη μνήμη γίνεται από την ίδια θέση. Αν το περιεχόμενο του καταχωρητή είναι 0, τότε η μεταφορά συνεχίζεται με την επόμενη θέση μνήμης.

Υλοποίηση εσωτερικών βρόχων επανάληψης

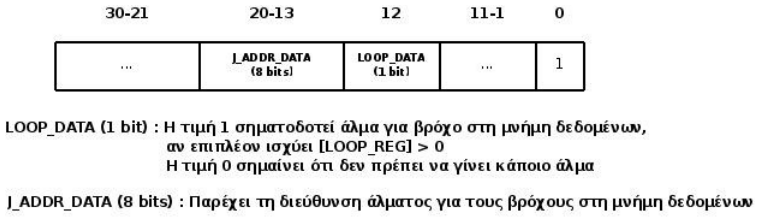
Η σχεδίαση του επαναδιατάξιμου συστήματος παρέχει τη δυνατότητα στο επαναδιατάξιμο περιφερειακό να εκτελεί βρόχους επανάληψης στο περιεχόμενο της μνήμης δεδομένων, εκτός από την ίδια δυνατότητα για τη μνήμη διαμόρφωσης, χωρίς την παρέμβαση του επεξεργαστή. Η λειτουργία αυτή είναι ακριβώς ίδια με αυτήν για την περίπτωση της μνήμης διαμόρφωσης. Από την πλευρά του επεξεργαστή αρκεί και μόνο ο προσδιορισμός του πλήθους των επαναλήψεων που πρέπει να γίνουν και η διεύθυνση στην οποία θα επιστρέφει προς τα πίσω η ανάγνωση δεδομένων.

Όπως και στην περίπτωση της μνήμης διαμόρφωσης υπάρχει ένας καταχωρητής μέσα στο επαναδιατάξιμο περιφερειακό στον οποίο υπάρχει το πλήθος των εναπομείναντων επαναλήψεων του βρόχου, ο καταχωρητής επαναλήψεων για τη μνήμη δεδομένων. Αυτός εγγράφεται αρχικά από το master, ώστε να καθοριστεί το συνολικό πλήθος των επαναλήψεων του βρόχου που απαιτούνται. Η λειτουργία του και η χρησιμότητά του είναι ακριβώς η ίδια με αυτή του

καταχωρητή επαναλήψεων για τη μνήμη διαμόρφωσης, επιτρέπει δηλαδή ή αντίθετα εμποδίζει την εκτέλεση των βρόχων επανάληψης στη μνήμη δεδομένων.

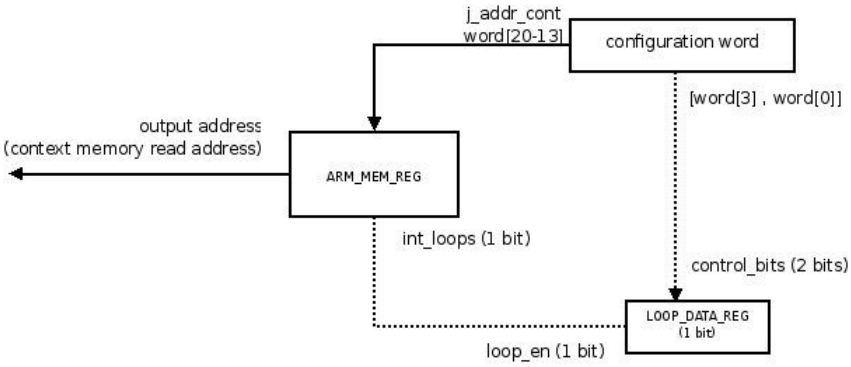
Αφετέρου, όπως περιγράφηκε και στη προηγούμενη υποενότητα "Επιλογή διεύθυνσης ανάγνωσης", εκτός από τις συνηθισμένες πιθανές διευθύνσεις που θα αποτελέσουν την επόμενη διεύθυνση ανάγνωσης, υπάρχει και μία διεύθυνση η οποία παρέχεται από το ίδιο το περιφερειακό, μέσω της λέξης διαμόρφωσης. Η διεύθυνση αυτή αποτελεί τη διεύθυνση στην οποία θα επιστρέψει η εκτέλεση του προγραμματος για την υλοποίηση του βρόχου επανάληψης.

Ο μηχανισμός λοιπόν των εσωτερικών αλμάτων επανάληψης στη μνήμη δεδομένων -καθώς και το υλικό που σχετίζεται με το μηχανισμό αυτό- είναι ίδιος με αυτόν για τη μνήμη διαμόρφωσης. Η διαφορά έγκειται στο ότι τα πεδία στη λέξη διαμόρφωσης που προσδιορίζουν τη διεύθυνση άλματος για το βρόχο επανάληψης καθώς και το bit που καθορίζει πότε πρέπει να γίνει το άλμα για το βρόχο επανάληψης είναι διαφορετικά από τα αντίστοιχα για τη μνήμη διαμόρφωσης. Στο Σχήμα 5.23 παρουσιάζονται τα πεδία αυτά μέσα σε μια λέξη διαμόρφωσης.



Σχήμα 5.23: Τα πεδία στη λέξη διαμόρφωσης που σχετίζονται με το σύστημα εσωτερικών βρόχων στη μνήμη δεδομένων

Ακόμη, το Σχήμα 5.24 απεικονίζει πλήρως τα κυκλώματα που εμπλέκονται στο μηχανισμό εσωτερικών αλμάτων στη μνήμη διαμόρφωσης.



Σχήμα 5.24: Κύκλωμα που σχετίζεται με την υλοποίηση εσωτερικών βρόχων στη μνήμη δεδομένων

5.3.3.Συγχρονισμός Μεταξύ των Μνημών

Ο έλεγχος του επαναδιατάξιμου υλικού στα πλαίσια του συστήματος MicroSoc γίνεται μέσω λογισμικού και συγκεκριμένα μέσω κώδικα σε γλώσσα C, ο οποίος φορτώνεται στη RAM του συστήματος και εκτελείται από τον ARM. Στους κώδικες αυτούς υπάρχουν συνήθως, πριν την έναρξη λειτουργίας του επαναδιατάξιμου περφερειακού, διάφοροι κύκλοι αρχικοποίησης του ARM ή πρέπει να εκτελεστούν αρχικά κάποιες εντολές του προγράμματος αποκλειστικά στον ARM. Για το λόγο αυτό πρέπει αρχικά τόσο η μνήμη διαμόρφωσης όσο και η μνήμη δεδομένων να επιλέγουν σε κάθε κύκλο για την επόμενη διεύθυνση ανάγνωσης τη διεύθυνση της πρώτης θέσης μνήμης. Οι μνήμες δηλαδή θα βρίσκονται σε μια κατάσταση stall. Σε μεταγενέστερη χρονική στιγμή πρέπει να δοθεί σήμα στο επαναδιατάξιμο περιφερειακό ότι πρέπει να ξεκινήσει η εκτέλεση των υπολογισμών, οι λέξεις διαμόρφωσης των οποίων βρίσκονται εκ των προτέρων στη μνήμη διαμόρφωσης.

Ειδικότερα θα πρέπει το πρόγραμμα -μέσω του ARM- να πραγματοποιήσει εγγραφή στον καταχωρητή επόμενης διεύθυνσης της μνήμης διαμόρφωσης και να μεταφέρει εκεί τη διεύθυνση εκείνου του συνόλου της μνήμης, στο οποίο περιέχονται οι λέξεις διαμόρφωσης που θα εκτελεστούν. Για τη διαδικασία αυτή ο ARM χρειάζεται δύο κύκλους ρολογιού -τουλάχιστον- (περισσότερες πληροφορίες αργότερα) μετά από τους οποίους οι πρώτες λέξεις του συνόλου που επιλέχτηκε αρχίζουν να μεταφέρονται στον πίνακα. Ωστόσο είναι πιθανό οι λέξεις αυτές να απαιτούν κάποια δεδομένα από τη μνήμη δεδομένων για να τα επεξεργαστούν. Αυτή η περίπτωση είναι προβληματική, καθώς τα κελιά δεν μπορούν να λάβουν τα σωστά δεδομένα από τη μνήμη δεδομένων, αφού η τελευταία βρίσκεται ακόμη σε κατάσταση stall. Θα πρέπει να δοθεί λοιπόν σήμα και στη μνήμη δεδομένων, ώστε να αρχίσει η μεταφορά δεδομένων από τη μνήμη στον επαναδιατάξιμο πίνακα. Θα πρέπει δηλαδή, κατ`αντιστοιχία με τη μνήμη διαμόρφωσης, να μεταφερθεί στον καταχωρητή επόμενης διεύθυνσης της μνήμης δεδομένων η διεύθυνση της θέσης μνήμης όπου βρίσκονται τα δεδομένα που πρέπει να μεταφερθούν στον πίνακα.

Η μεταφορά αυτή για εγγραφή από τον ARM προς το επαναδιατάξιμο περιφερειακό απαιτεί επίσης τουλάχιστον 2 κύκλους ρολογιού, μέσα στους οποίους έχουν ήδη μεταφερθεί προς τον επαναδιατάξιμο πίνακα δύο -τουλάχιστον- θέσεις από τη μνήμη διαμόρφωσης. Λόγω των παραπάνω δεν υπάρχει συγχρονισμός μεταξύ της αρχικής λειτουργίας των δύο μνημών. Το πρόβλημα αυτό έχει λυθεί με έναν ειδικό μηχανισμό συγχρονισμού.

Ο μηχανισμός αυτός αποτελείται από ένα καταχωρητή με μέγεθος 1 bit, τον καταχωρητή συγχρονισμού. Ο συγκεκριμένος καταχωρητής έχει μια είσοδο δεδομένων, απ`όπου έρχονται τα δεδομένα για εγγραφή και μία είσοδο ελέγχου εγγραφής, η οποία τίθεται στο 1 όταν χρειάζεται να γίνει εγγραφή στον καταχωρητή. Η λειτουργία του είναι απλή, καθώς σε κάθε θετική ακμή του ρολογιού του συστήματος γίνεται δειγματοληψία της εισόδου ελέγχου εγγραφής. Στη συνέχεια γίνεται έλεγχος του δείγματος που λήφθηκε και αν

αυτό είναι ίσο με 1, τότε το περιεχόμενο του καταχωρητή γίνεται ίσο με τα δεδομένα από την είσοδο δεδομένων. Η έξοδος του καταχωρητή είναι απευθείας συνδεδεμένη με το περιεχόμενο του καταχωρητή.

Ο καταχωρητής συγχρονισμού παρέχει την έξοδό του στον καταχωρητή επόμενης διεύθυνσης τόσο της μνήμης διαμόρφωσης όσο και της μνήμης δεδομένων και συγκεκριμένα στην είσοδο συγχρονισμού τους. Οπως φάνηκε και κατά την περιγραφή των καταχωρητών αυτών παραπάνω, η είσοδος αυτή, ανάλογα με την τιμή της, επιτρέπει -ή όχι- στον καταχωρητή επόμενης διεύθυνσης να μεταφέρει στην έξοδό του την καινούρια τιμή της διεύθυνσης (που έχει αποθηκευτεί στο περιεχόμενό του).

Ο τρόπος χρήσης του συγκεκριμένου καταχωρητή είναι ο εξής. Αρχικά, κατά την έναρξη του κώδικα που είναι αποθηκευμένος στην RAM του συστήματος και τον οποίο εκτελεί ο επεξεργαστής, ο καταχωρητής συγχρονισμού τίθεται στο 0. Κάποια στιγμή μέσα στο πρόγραμμα γίνεται η μεταφορά στους καταχωρητές επόμενης διεύθυνσης των δύο μνημών του επανδιατάξιμου περιφερειακού των κατάλληλων διευθύνσεων αρχής, έστω πρώτα για τη μνήμη διαμόρφωσης και έπειτα για τη μνήμη δεδομένων. Ωστόσο, η μεταφορά των θέσεων των μνημών προς τον πίνακα επαναδιατάξιμων κελιών δεν αρχίζει, διότι ο καταχωρητής συγχρονισμού έχει ακόμα την τιμή 0. Όταν ολοκληρωθεί η μεταφορά των διευθύνσεων στους δύο καταχωρητές, τότε το πρόγραμμα πρέπει να μεταφέρει στον καταχωρητή συγχρονισμού την τιμή 1. Έτσι, μετά από την τελευταία μεταφορά οι δύο μνήμες αρχίζουν να μεταφέρουν θέσεις μνήμης προς τον επαναδιατάξιμο πίνακα ταυτόχρονα, ώστε τα δεδομένα που μεταφέρονται προς επεξεργασία να ταιριάζουν με τα δεδομένα διαμόρφωσης.

5.4. Σύστημα Διακοπών

5.4.1.Χειρισμός Εξαιρέσεων και ARM

Η αρχιτεκτονική ARM υποστηρίζει ένα μεγάλο εύρος εξαιρέσεων, οι οποίες χρησιμοποιούνται για το χειρισμό αναπάντεχων γεγονότων που ανακύπτουν κατά τη διάρκεια εκτέλεσης του προγράμματος, όπως οι διακοπές ή τα σφάλματα μνήμης. Ο όρος 'εξαίρεση' χρησιμοποιείται ακόμα για να καλύψει διακοπές λογισμικού (software) αλλά και μη ορισμένες 'παγίδες' στο πρόγραμμα εκτέλεσης και τη συνάρτηση reset του συστήματος, το οποίο λογικά ανακύπτει πριν και όχι κατά τη διάρκεια εκτέλεσης ενός προγράμματος. Όλα αυτά τα γεγονότα ομαδοποιούνται κάτω από τη γενική κατηγορία των εξαιρέσεων και η εξυπηρέτησή τους μέσα στον επεξεργαστή γίνεται με τον ίδιο βασικό μηχανισμό.

Όταν εμφανίζεται μια οποιαδήποτε εξαίρεση, τότε ο ARM ολοκληρώνει την εντολή που εκτελείται εκείνη τη στιγμή όσο καλύτερα μπορεί (εκτός από τις

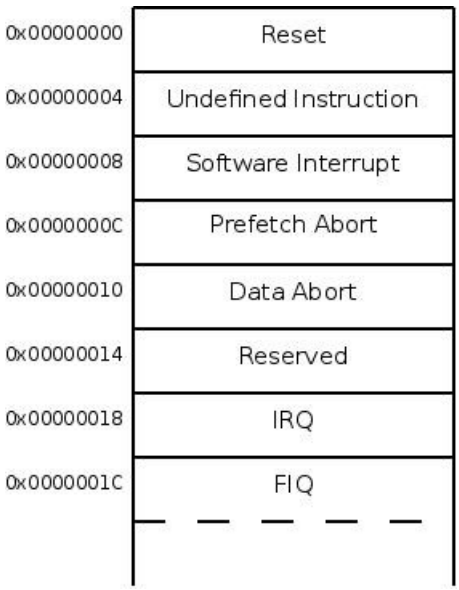
εξαιρέσεις επαναφοράς στις οποίες η τρέχουσα εντολή τερματίζεται άμεσα) και στη συνέχεια φεύγει από την τρέχουσα ακολουθία εντολών για να εξυπηρετήσει την εξαίρεση. Ο επεξεργαστής εφαρμόζει τελικά την παρακάτω ακολουθία ενεργειών :

- Αλλάζει την κατάσταση λειτουργίας σε αυτήν που αντιστοιχεί στη συγκεκριμένη εξαίρεση που εμφανίστηκε. Η κανονική κατάσταση λειτουργίας είναι η *user_mode* και ανάλογα με το είδος της εξαίρεσης υπάρχουν για παράδειγμα οι καταστάσεις λειτουργίας *irq_mode* και *fiq_mode*.

- Αποθηκεύεται η διεύθυνση της εντολής που θα ακολουθούσε στην κανονική ακολουθία εντολών μετά το σημείο εμφάνισης της εξαίρεσης με την αντιγραφή του περιεχομένου του καταχωρητή PC στον *r14_<exc>* και του περιεχομένου του CPSR στον *SPSR_<exc>,* όπου το *<exc>* αντιστοιχεί στον τύπο της εξαίρεσης. Δηλαδή για παράδειγμα υπάρχει ξεχωριστός καταχωρητής *r14* για τις διακοπές FIQ (ο *r14_fiq*) και άλλος για τις διακοπές IRQ (ο *r14_irq*).

- Οι διακοπές απενεργοποιούνται προσωρινά, για όσο διάστημα διαρκεί η εξυπηρέτηση, με το να τίθεται το bit7 του καταχωρητή CPSR στην τιμή '1' και -αν η εξαίρεση ανήκει στην κατηγορία της γρήγορης διακοπής- απενεργοποιούνται επίσης οι γρήγορες διακοπές, με το να τίθεται το bit6 του CPSR στην τιμή '1'.

- Το περιεχόμενο του PC τίθεται σε μια τιμή μεταξύ $(00)_{16}$ και $(1C)_{16}$, με την τιμή αυτή να εξαρτάται από το είδος της εξαίρεσης. Η εντολή που περιέχεται στη θέση μνήμης στην οποία τέθηκε να δείχνει ο καταχωρητής PC, που αποτελεί τη διεύθυνση του διανύσματος εξαίρεσης, περιέχει συνήθως ένα άλμα στο χειριστή της συγκεκριμένης εξαίρεσης (exception handler). Ο τελευταίος θα χρησιμοποιήσει τον καταχωρητή *r13_<exc>*, ο οποίος συνήθως αρχικοποιείται ώστε να δείχνει σε μια στοίβα δεσμευμένη στη μνήμη, για να αποθηκεύσει το περιεχόμενο μερικών καταχωρητών που χρησιμοποιούνται στην κατάσταση λειτουργίας *user mode* από το κυρίως πρόγραμμα. Συνήθως η διεύθυνση του διάνυσματος για τις εξαιρέσεις τύπου IRQ είναι η $(18)_{16}$ και για τις εξαιρέσεις τύπου FIQ είναι η $(1C)_{16}$. Στο Σχήμα 5.25 παρέχεται μια τυπική εικόνα της περιοχής μνήμης $(00)_{16}$ - $(1C)_{16}$, όπου φαίνεται η αντιστοίχιση της κάθε θέσης στην περιοχή αυτή με ένα συγκεκριμένο είδος εξαίρεσης.



Σχήμα 5.25: Η περιοχή μνήμης 00-1C με τα διανύσματα εξαιρέσεων

Μόλις λήξει η εξυπηρέτηση της εξαίρεσης τότε το σύνηθες είναι να γίνει επαναφορά της εφαρμογής του χρήστη που βρισκόταν σε εκτέλεση όταν εμφανίστηκε η εξαίρεση. Αυτό απαιτεί ο κώδικας χειρισμού της εξαίρεσης να αποκαταστήσει το *user mode* ακριβώς όπως ήταν πριν :

- Όλοι οι τροποποιημένοι καταχωρητές του *user mode* πρέπει να αποκατασταθούν από τη στοίβα του χειριστή.
- Ο καταχωρητής CPSR πρέπει να αποκατασταθεί με βάση ατον καταχωρητή *SPSR_<exc>*
- Ο καταταχωρητής PC πρέπει να αλλαχθεί πάλι και να λάβει την κατάλληλη διεύθυνση εντολής στην ακολουθία εντολών χρήστη.

Να σημειωθεί ότι τα δύο τελευταία βήματα δεν μπορούν να πραγματοποιηθούν ανεξάρτητα. Αν αποκατασταθεί πρώτα ο CPSR, τότε ο *r14_<exc>* που διατηρεί τη διεύθυνση επιστροφής από τη διακοπή δε θα είναι προσβάσιμος. Από την άλλη αν αποκατασταθεί πρώτα ο PC, τότε ο χειριστής εξαίρεσης χάνει τον έλεγχο και δεν μπορεί να πραγματοποιήσει την αποκατάσταση της τιμής του CPSR. Για το λόγο αυτό ο ARM παρέχει δύο μηχανισμούς οι οποίοι προκαλούν και τα δύο παραπάνω βήματα -για τα οποία γίνεται συζήτηση- να πραγματοποιηθούν ατομικά σαν κομμάτι μιας μοναδικής εντολής. Ο πρώτος -και ο οποίος θα αναλυθεί περισσότερο- αφορά στην περίπτωση που η διεύθυνση επιστροφής διατηρείται στον *r14_<exc>* και ο άλλος όταν η διεύθυνση αυτή διατηρείται σε στοίβα.

Για την επαναφορά, λοιπόν, από διακοπές τύπου IRQ / FIQ χρησιμοποιείται η εντολή:

```
SUBS        pc, r14, #4
```

Το 's' μετά από την κανονική εντολή SUB σηματοδοτεί την ειδική μορφή της εντολής αυτής και χρησιμοποιείται όταν ο καταχωρητής προορισμού είναι ο PC. Με την ειδική αυτή μορφή γίνεται παράλληλα και μια προσαρμογή στη διεύθυνση επιστροφής -όπου χρειάζεται. Για παράδειγμα στις περιπτώσεις των IRQ / FIQ η επιστροφή πρέπει να γίνει μία εντολή πριν από αυτή που έχει αποθηκευτεί στον r14_<exc> διότι κατά τη διαδικασία της διακοπής του προγράμματος για την εξυπηρέτηση της εξαίρεσης 'χάθηκε' στην πορεία μια εντολή.

Από πλευράς υλικού, ο επεξεργαστής ARM έχει δύο ειδικές εισόδους για τις εξωτερικές διακοπές, δηλαδή τις IRQ και FIQ. Υπάρχει μία είσοδος για υψηλής προτεραιότητας διακοπές (είσοδος FIQ) και μία για τις υπόλοιπες εξωτερικές διακοπές (είσοδος IRQ) από τα περιφερειακά. Αυτές οι είσοδοι είναι και οι δύο ενεργές χαμηλά, το οποίο σημαίνει πως όταν στις εισόδους αυτές υπάρξει μετάβαση από υψηλό σε χαμηλό σήμα ο επεξεργαστής καταλαβαίνει πως υπάρχει κάποια αίτηση για διακοπή, οπότε και ενεργοποιείται η διαδικασία που περιγράφηκε προηγουμένως για την εξυπηρέτησή της.

Πίνακας 5.5: Οι εξαιρέσεις του ARM

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined Instruction	UND	0x00000004
Software Interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

5.4.2.Διακοπές και rMicroSoc

Μετά την ολοκλήρωση της μεταφοράς όλων των λέξεων διαμόρφωσης που υπάρχουν σε κάποιο σύνολο της μνήμης διαμόρφωσης η μεταφορά λέξεων προς τον επαναδιατάξιμο πίνακα πρέπει να σταματήσει. Η επόμενη διεύθυνση ανάγνωσης για τη μνήμη διαμόρφωσης πρέπει να είναι ίδια με την προηγούμενη, ώστε στην ουσία να μεταφέρονται διαρκώς οι ίδιες λέξεις, οι οποίες μπορούν απλά να θέτουν όλα τα κελιά σε κατάσταση stall. Υπάρχουν δύο λόγοι για την ενέργεια αυτή.

Κατ᾽αρχήν υπάρχει μεγάλη πιθανότητα ο master να χρειάζεται κάποια αποτελέσματα από τους υπολογισμούς που εκτελέστηκαν στο επαναδιατάξιμο περιφερειακό. Άλλωστε αυτό είναι το νόημα ύπαρξης του περιφερειακού επαναδιατάξιμης λογικής, να πραγματοποιεί δηλαδή διάφορους υπολογισμούς για λογαριασμό του master. Οποιαδήποτε δεδομένα είναι απαραίτητο να διαβαστούν από το master πρέπει να υπάρχουν αποθηκευμένα στα αρχεία καταχωρητών των διάφορων κελιών που απαρτίζουν τον επαναδιατάξιμο πίνακα. Υπό αυτή την έννοια η εκτέλεση του προγράμματος διακόπτεται ώστε να μεταφερθούν στο κυρίως σύστημα κάποια προσωρινά -ή και τελικά- δεδομένα. Υπενθυμίζεται εξάλλου ότι το MicroSoc στην παρούσα του μορφή που χρησιμοποιήθηκε στην εργασία δεν υποστηρίζει την ύπαρξη πολλαπλών master του διαδρόμου ASB. Έτσι οποιαδήποτε μεταφορά προς τις μνήμες του συστήματος (για την ακρίβεια οποιαδήποτε χρήση του διαδρόμου) από κάποιο περιφερειακό θα πρέπει να γίνει μέσω του master. Το περιφερειακό δηλαδή δεν μπορεί να ενεργήσει αυτόνομα και να μεταφέρει τα αποτελέσματα που απαιτούνται προς τις μνήμες του συστήματος, γιατί δεν μπορεί να ενεργήσει σα master του διαδρόμου.

Ένας δεύτερος λόγος ύπαρξης του χαρακτηριστικού αυτού είναι ότι δεν -μπορεί να- είναι εκ των προτέρων γνωστό ποιο σύνολο της μνήμης πρέπει να ακολουθήσει. Εξάλλου υπάρχουν πολλές επιλογές· μπορεί το κομμάτι του αλγορίθμου που πρέπει να εκτελέσει το επαναδιατάξιμο περιφερειακό να έχει ολοκληρωθεί, μπορεί να επαναληφθεί η εκτέλεση του ίδιου συνόλου -πιθανώς με χρήση άλλων δεδομένων, αλλά μπορεί και όχι. Επίσης μπορεί να ακολουθήσει οποιοδήποτε από τα άλλα 3 σύνολα, όχι απαραίτητα με την ίδια σειρά που περιλαμβάνονται στη μνήμη. Επιπλέον είναι πιθανό το επόμενο σύνολο να εξαρτάται από κάποια αποτελέσματα που θα προκύψουν από τους υπολογισμούς στο τρέχον σύνολο.

Για όλα τα παραπάνω είναι απαραίτητο η εκτέλεση του προγράμματος να σταματάει στο τέλος κάθε συνόλου και να αναμένεται η αντίδραση του master προς το γεγονός αυτό. Λαμβάνοντας υπόψη το χαρακτηριστικό της παραλληλίας των υπολογισμών, δηλαδή ότι οι υπολογισμοί που πραγματοποιούνται στο επαναδιατάξιμο περιφερειακό γίνονται παράλληλα με άλλους υπολογισμούς που εκτελεί ο ίδιος ο master, διαφαίνεται κάποιο πρόβλημα. Το πρόβλημα είναι το πώς θα αντιληφθεί ο master το γεγονός αυτό, ώστε να πράξει ανάλογα. Το παραπάνω λύνεται με τη χρησιμοποίηση του συστήματος διακοπών.

Στο τέλος κάθε συνόλου θα πρέπει το επαναδιατάξιμο περιφερειακό να κάνει αίτηση για διακοπή προς το master, μένοντας στη συνέχεια σε κατάσταση αδράνειας περιμένοντας να εξυπηρετηθεί. Όταν ο master εξυπηρετήσει τελικά τη συγκεκριμένη αίτηση για διακοπή θα πραγματοποιηθούν όλες οι απαιτούμενες ενέργειες, οι οποίες χωρίζονται σε δύο βασικές κατηγορίες. Αφενός θα γίνει η μεταφορά ορισμένων δεδομένων από τα αρχεία καταχωρητών των κελιών του επαναδιατάξιμου πίνακα προς την κεντρική μνήμη του συστήματος. Αφετέρου θα καθοριστεί με ποιο σύνολο χρειάζεται -και αν- να συνεχιστεί η εκτέλεση των υπολογισμών. Ταυτόχρονα δίνεται και η δυνατότητα

στο master να γράψει νέα δεδομένα στη μνήμη δεδομένων και/ή να ορίσει το πλήθος κάποιων βρόχων επανάληψης που πρέπει να γίνουν στις μνήμες στη συνέχεια της εκτέλεσης.

Για την υποστήριξη της λειτουργίας των διακοπών πρέπει να προστεθούν δύο ακόμα στοιχεία στο σύστημα. Στην πλευρά του επαναδιατάξιμου περιφερειακού προστίθεται μια διεπαφή για τις διακοπές και στην πλευρά του κυρίως συστήματος, στο διάδρομο ASB, προστίθεται ένας ελεγκτής για τις διακοπές. Παρακάτω θα γίνει μια αναλυτική αναφορά στις δύο αυτές μονάδες.

5.4.3.Διεπαφή Διακοπών στο Περιφερειακό

Το συγκεκριμένο στοιχείο χρησιμοποιείται για να υλοποιεί στην πράξη την αίτηση για διακοπή προς το master του κυρίως συστήματος και να παρέχει στον τελευταίο -κατά την εξυπηρέτηση της διακοπής- ορισμένες απαραίτητες πληροφορίες.

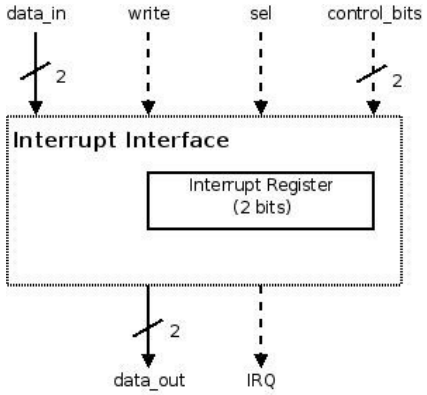
Η μονάδα περιέχει έναν καταχωρητή 2 bit, τον *interrupt register*, το περιεχόμενο του οποίου είναι ένας αριθμός σε δυαδική μορφή που αντιστοιχεί στο τρέχον σύνολο της μνήμης διαμόρφωσης που εκτελείται. Ο καταχωρητής αυτός έχει μια είσοδο επιλογής λειτουργίας (*sel* - 1 bit), μια είσοδο επιλογής μεταξύ της λειτουργίας εγγραφής και ανάγνωσης (*write* - 1 bit), μια είσοδο δεδομένων εγγραφής (*data_in* - 2 bits) και τέλος μια έξοδο δεδομένων (*data_out* - 2 bits). Τα παραπάνω σήματα σχετίζονται με την επικοινωνία της διεπαφής με το διάδρομο AMBA-ASB. Όταν επιλεγεί για εγγραφή, δηλαδή όταν το σήμα επιλογής λειτουργίας και το σήμα επιλογής εγγραφής/ανάγνωσης τεθούν στο 1, τότε θέτει το περιεχόμενό του με τα δεδομένα από την είσοδο δεδομένων εγγραφής. Όταν επιλεγεί για ανάγνωση, δηλαδή όταν η είσοδος επιλογής λειτουργίας τεθεί 1 και η είσοδος επιλογής εγγραφής/ανάγνωσης τεθεί στο 0, τότε τα περιεχόμενα του καταχωρητή μεταφέρονται στην έξοδο δεδομένων.

Ο καταχωρητής αυτός χρησιμεύει για να μπορεί ο master να ενημερώνεται, κατά τη διάρκεια εξυπηρέτησης της διακοπής, για το ποιο σύνολο έχει ολοκληρώσει την εκτέλεσή του, ώστε να πράξει ανάλογα. Αυτό είναι απαραίτητο διότι είναι πολύ πιθανό οι ενέργειες που θα ακολουθηθούν από το master, μετά το τέλος κάποιου συνόλου, να εξαρτώνται άμεσα από το ποιο σύνολο έχει τελειώσει. Συγκεκριμένα κατά την έναρξη εκτέλεσης ενός συγκεκριμένου συνόλου της μνήμης, ο master γράφει στον καταχωρητή αυτό το νούμερο του συνόλου που πρόκειται να ξεκινήσει να εκτελείται. Όταν γίνει τελικά η εξυπηρέτηση της διακοπής από το master, διαβάζει το περιεχόμενο του καταχωρητή και μπορεί έτσι να καταλάβει ποιο σύνολο έχει τελειώσει.

Μια άλλη βασική λειτουργία που επιτελεί η συγκεκριμένη διεπαφή είναι η αίτηση

διακοπής προς τον ARM. Δηλαδή έχει μια έξοδο του 1 bit, την έξοδο *IRQ*, της οποίας η τιμή δηλώνει αν το περιφερειακό θέλει διακοπή (τιμή 1) ή όχι (τιμή 0). Η έξοδος αυτή συνδέεται με μια κατάλληλη είσοδο του ελεγκτή διακοπών που βρίσκεται στο κυρίως σύστημα. Από τη στιγμή που δηλωθεί στον ελεγκτή ότι το περιφερειακό θέλει διακοπή, η αίτηση αυτή αποθηκεύεται εσωτερικά στον ελεγκτή και η περαιτέρω επεξεργασία της αίτησης (επιβολή απαραίτητων συνθηκών προτεραιτότητας, προώθηση της αίτησης προς το master κλπ.) επαφίεται στον ελεγκτή.

Το περιφερειακό ζητάει αίτηση από το master κατά το τέλος της εκτέλεσης κάθε συνόλου της μνήμης διαμόρφωσης. Ωστόσο για να μπορέσει το περιφερειακό να αντιληφθεί το τέλος κάποιου συνόλου και να κάνει την αίτηση για διακοπή θα πρέπει να υπάρχει κάποιος τρόπος ελέγχου. Ο τρόπος αυτός είναι μέσω της λέξης διαμόρφωσης. Υπάρχει λοιπόν ένα bit στη λέξη διαμόρφωσης, συγκεκριμένα το bit 1, το οποίο όταν η λέξη είναι στην κατάσταση stall δηλώνει ότι το περιφερειακό αιτείται διακοπή από το σύστημα. Η αίτηση αυτή οδηγείται αρχικά μέσω της εξόδου αίτησης διακοπής του περιφερειακού στον ελεγκτή διακοπών του ASB, ο οποίος με τη σειρά του αποφασίζει πότε πρέπει να προωθήσει την αίτηση αυτή προς το master. Για να επιτελεί αυτή τη λειτουργία, η διεπαφή έχει επιπλέον μία είσοδο ελέγχου των 2 bits, την είσοδο *control_bits.* Το ένα από αυτά συνδέεται με το bit0 της λέξης διαμόρφωσης, το οποίο δηλώνει ότι η λέξη είναι σε κατάσταση stall. Το άλλο bit της εισόδου ελέγχου συνδέεται με το bit εκείνο της λέξης διαμόρφωσης που δηλώνει αν το περιφερειακό πρέπει να ζητήσει διακοπή ή όχι. Όταν και τα δύο bits της εισόδου *control_bits* είναι στο '1', τότε η μονάδα της διεπαφής των διακοπών θέτει την έξοδο *IRQ* στην τιμή '1' και έτσι μεταφέρει στον ελεγκτή διακοπών του συστήματος ASB την αίτηση αυτή.



Σχήμα 5.26: Η διεπαφή διακοπών του επαναδιατάξιμου περιφερειακού

5.4.4.Ελεγκτής Διακοπών ASB

Ο ελεγκτής αυτός υπάρχει στο κεντρικό κομμάτι του συστήματος MicroSoc και είναι ένα ακόμη περιφερειακό του διαδρόμου ASB, πάνω στον οποίο βρίσκεται συνδεδεμένος. Επιλέγεται για λειτουργία από τον αποκωδικοποιητή ASB όταν τα bits 31-28 του διαδρόμου διευθύνσεων πάρουν την τιμή 0100 (4). Σκοπός της συγκεκριμένης μονάδας είναι να συγκεντρώνει από όλα τα δυνατά περιφερειακά τις αιτήσεις διακοπής που έχουν προς το master και στη συνέχεια να δηλώνει στο master ότι υπάρχουν διαθέσιμες αιτήσεις προς εξυπηρέτησης. Η περαιτέρω επεξεργασία των αιτήσεων από τα διάφορα περιφερειακά, όπως το ποιες, το πώς και με ποια σειρά θα εξυπηρετηθούν, είναι ευθύνη του software. Συγκεκριμένα γράφεται μια συνάρτηση εξυπηρέτησης διακοπών, η οποία καλείται από το κυρίως πρόγραμμα που εκτελείται στο master όταν θέλει να κάνει εξυπηρέτηση των διακοπών που υπάρχουν.

Τα σήματα εισόδου του ελεγκτή είναι τα εξής :

- *dsel* (1 bit) – Σήμα επιλογής λειτουργίας του ελεγκτή. Το σήμα αυτό συνδέεται απευθείας με μια έξοδο του αποκωδικοποιητή ASB και παίρνει την τιμή 1 όταν ο αποκωδικοποιητής επιλέγει τον ελεγκτή ώστε να συμμετάσχει σε κάποια μεταφορά του συστήματος. Η επιλογή του ελεγκτή από τον αποκωδικοποιητή γίνεται όταν τα bits 31-28 πάρουν την τιμή 4, σε δυαδική μορφή.

- *baddr* (32 bits) – Είσοδος διεύθυνσης που συνδέεται απευθείας με το διάδρομο διευθύνσεων του ASB.

- *bdata_in* (32 bits) – Είσοδος δεδομένων εισόδου που συνδέεται απευθείας με το διάδρομο εγγραφής δεδομένων του ASB.

- *bwrite* (1 bit) – Σήμα επιλογής εγγραφής/ανάγνωσης που συνδέεται απευθείας με το αντίστοιχο σήμα επιλογής εγγραφής/ανάγνωσης BWRITE του συστήματος ASB. Καθορίζει, κατά τα γνωστά, το αν μια μεταφορά θα είναι μεταφορά εγγραφής (παίρνει την τιμή 1) ή μεταφορά ανάγνωσης (παίρνει την τιμή 0).

- *IRQsource* (16 bits) – Η είσοδος αυτή αντιστοιχεί στις πηγές αιτήσεων διακοπής από τα περιφερειακά του ASB. Δηλαδή τα περιφερειακά που δυνητικά μπορούν να ζητήσουν διακοπή από τον ARM έχουν συνδεδεμένες τις εξόδους αίτησης διακοπής με ένα bit της εισόδου αυτής. Όταν κάποιο περιφερειακό θέσει την έξοδό του στην τιμή 1, γεγονός το οποίο αντικατοπτρίζεται σε κάποιο bit της εισόδου IRQsource, τότε ο ελεγκτής αντιλαμβάνεται ότι το περιφερειακό αυτό ζητάει διακοπή και πρέπει να δράσει ανάλογα. Να σημειωθεί ότι σε ένα bit της εισόδου αυτής είναι συνδεδεμένη και η έξοδος αίτησης διακοπής *IRQ* του επαναδιατάξιμου περιφερειακού, η οποία με τη σειρά της συνδέεται απευθείας με την αντίστοιχη έξοδο *irq* της διεπαφής διακοπών του περιφερειακού (η οποία περιγράφηκε στην προηγούμενη υποενότητα).

Τα σήματα εξόδου του ελεγκτή είναι τα εξής:

- *nIRQ* (1 bit) – Έξοδος αίτησης διακοπής προς τον ARM η οποία συνδέεται απευθείας με την είσοδο διακοπής του επεξεργαστή. Όταν έστω και μία πηγή αιτήσεων διακοπής (τα περιφερειακά δηλαδή) έχει κάνει αίτηση διακοπής, τότε ο ελεγκτής θέτει τη συγκεκριμένη έξοδο στο 0 κάνοντας γνωστό στον ARM ότι υπάρχουν αιτήσεις διακοπής προς εξυπηρέτηση.

- *bdata_out* (32 bits) – Σήμα δεδομένων εξόδου που χρησιμοποιείται για την έξοδο δεδομένων από τον ελεγκτή προς το σύστημα και συνδέεται απευθείας με το διάδρομο ανάγνωσης ASB.

Ο συγκεκριμένος ελεγκτής υποστηρίζει την ύπαρξη συνολικά 16 περιφερειακών, τα οποία είναι πιθανό να αιτηθούν διακοπής στο master. Παρόλα αυτά στην πράξη χρησιμοποιήθηκε μόνο το επαναδιατάξιμο περιφερειακό, καθώς δεν υπήρχε κάποιο άλλο με τέτοιες απαιτήσεις. Στηρίζει τη λειτουργία του σε 4 εσωτερικούς καταχωρητές που περιέχει, όλοι με μέγεθος 16 bits, με κάθε bit να αντιστοιχεί και σε ένα περιφερειακό που σχετίζεται με διακοπές στο διάδρομο ASB. Οι καταχωρητές αυτοί φαίνονται απευθείας στο master μέσω της απεικόνισης στη μνήμη και για την επιλογή τους χρησιμοποιούνται τα bits 27-26 του διαδρόμου διευθύνσεων, καθώς τα bits 31-28 χρησιμοποιούνται από τον αποκωδικοποιητή του διαδρόμου ASB για την επιλογή του κατάλληλου περιφερειακού του ASB. Οι καταχωρητές αυτοί είναι οι εξής:

- *IRQ_STATUS* – Χρησιμοποιείται για να δηλώνεται η κατάσταση αίτησης διακοπής για κάθε περιφερειακό. Συγκεκριμένα, κάθε bit του καταχωρητή αντιστοιχεί σε ένα διαφορετικό περιφερειακό και η τιμή του δηλώνει αν το συγκεκριμένο περιφερειακό έχει ενεργή αίτηση για διακοπή. Αν ένα bit του IRQ_STATUS έχει την τιμή 1, τότε το αντίστοιχο περιφερειακό έχει μια αίτηση για διακοπή η οποία δεν έχει ακόμα -και πρέπει να- εξυπηρετηθεί. Αντίθετα αν ένα bit έχει την τιμή 0, τότε το αντίστοιχο περιφερειακό δεν έχει κάποια άιτηση που δεν έχει εξυπηρετηθεί. Ο καταχωρητής αυτός επιλέγεται όταν έχει επιλεγεί το περιφερειακό του ελεγκτή από τον αποκωδικοποιητή ASB και τα bits 27-26 του διαδρόμου διευθύνσεων έχουν την τιμή 00. Η ανάγνωσή του γίνεται από το master, όταν επιλεγεί και το σήμα BWRITE του διαδρόμου ASB έχει την τιμή 0. Κατά την ανάγνωση μεταφέρονται τα περιεχόμενά του στην έξοδο δεδομένων του ελεγκτή διακοπών, στη θετική ακμή του ρολογιού του συστήματος. Η εγγραφή του καταχωρητή δεν ελέγχεται από το master, αλλά γίνεται για κάθε bit χωριστά και σχετίζεται με τις εισόδους από τις πηγές αιτήσεων διακοπής και τον καταχωρητή IRQ_MASK_ENABLE. Για την εγγραφή κάθε bit του IRQ_STATUS, ελέγχεται το αντίστοιχο bit του καταχωρητή IRQ_MASK_ENABLE, κάθε φορά που το αντίστοιχο bit της εισόδου αιτήσεων διακοπής IRQsource μεταβεί από το 0 στο 1. Να σημειωθεί ότι μια τέτοια μετάβαση σημαίνει ότι το περιφερειακό που αντιστοιχεί στο bit αυτό κάνει αίτηση για διακοπή. Αν το αντίστοιχο bit του IRQ_MASK_ENABLE έχει την τιμή 1, τότε και το bit του IRQ_STATUS

τίθεται στο 1, διαφορετικά παίρνει την τιμή 0. Η έξοδος αίτησης διακοπής προς τον ARM τίθεται όταν έστω και ένα bit του IRQ_STATUS έχει την τιμή 1.

- *IRQ_CLEAR* – Χρησιμοποιείται για τον επιλεγμένο 'καθαρισμό' ορισμένων bit του καταχωρητή IRQ_STATUS. Συγκεκριμένα όταν κάποιο bit πάρει την τιμή 1, τότε το αντίστοιχο bit του IRQ_STATUS γίνεται 0. Ο καταχωρητής αυτός είναι διαθέσιμος μόνο για εγγραφή από το master και η επιλογή του γίνεται όταν επιλεγεί ο ελεγκτής διακοπών από τον αποκωδικοποιητή ASB και τα bits 27-26 του διαδρόμου διευθύνσεων πάρουν την τιμή 1. Να τονιστεί ότι τα δεδομένα εγγραφής διατηρούνται μόνο για έναν κύκλο μετά την εγγραφή τους και στη συνέχεια διαγράφονται από αυτόν.

- *IRQ_MASK_ENABLE* – Ο καταχωρητής αυτός χρησιμοποιείται για να καθορίσει ο προγραμματιστής ποια περιφερειακά επιτρέπεται να αιτηθούν για διακοπή και ποια όχι. Συγκεκριμένα κάθε bit του αντιστοιχεί σε ένα διαφορετικό περιφερειακό και αν έχει την τιμή 1 τότε οι αιτήσεις από το περιφερειακό αυτό επιτρέπονται, διαφορετικά οι αιτήσεις του συγκεκριμένου περιφερειακού εμποδίζονται. Ο καταχωρητής IRQ_MASK_ENABLE επιτρέπεται μόνο να εγγραφεί από το master και επιλέγεται όταν έχει επιλεγεί ο ελεγκτής διακοπών και τα bits 27-26 έχουν την τιμή 2 (σε δυαδική μορφή). Υπάρχει και η δυνατότητα 'καθαρισμού' ορισμένων bits του καταχωρητή, μέσω της λειτουργίας του καταχωρητή IRQ_MASK_CLEAR.

- *IRQ_MASK_CLEAR* – Χρησιμοποιείται για τον επιλεγμένο 'καθαρισμό' ορισμένων bit του καταχωρητή IRQ_MASK_ENABLE. Συγκεκριμένα όταν κάποιο bit πάρει την τιμή 1, τότε το αντίστοιχο bit του IRQ_MASK_ENABLE τίθεται στο 0. Ο IRQ_MASK_CLEAR είναι διαθέσιμος μονάχα για εγγραφή από το master και επιλέγεται όταν επιλεγεί ο ελεγκτής διακοπών ASB και τα bits 27-26 του διαδρόμου διευθύνσεων πάρουν την τιμή 3 (σε δυαδική μορφή). Να σημειωθεί ότι τα δεδομένα εγγραφής διατηρούνται μονάχα για έναν κύκλο, μετά την εγγραφή τους στον καταχωρητή και στη συνέχεια διαγράφονται από αυτόν.

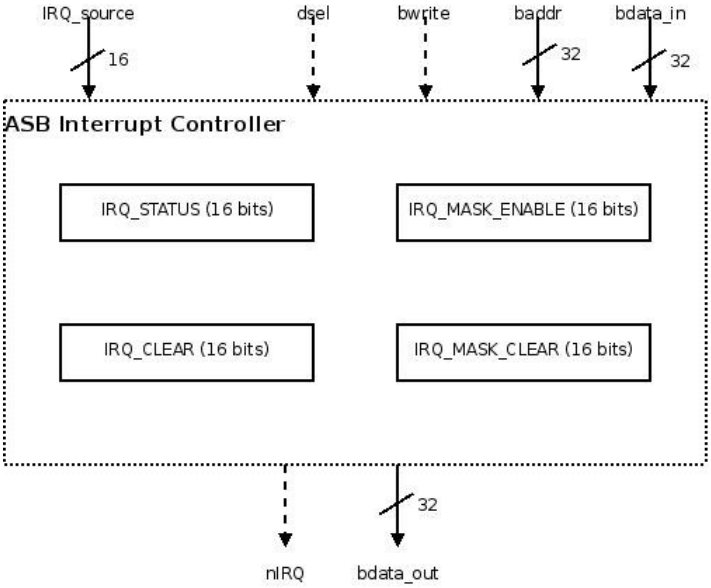
Ο Πίνακας 5.6 παρουσιάζει συγκεντρωτικά το πώς αντιστοιχίζονται οι παραπάνω τέσσερις καταχωρητές με τα bits 27,26 του διαδρόμου διευθύνσεων.

Πίνακας 5.6: Επιλογή των καταχωρητών του ελεγκτή διακοπών ASB

BA[27-26]	Register
00	IRQ_STATUS
01	IRQ_CLEAR
10	IRQ_MASK_ENABLE
11	IRQ_MASK_CLEAR

Το Σχήμα 5.27 δείχνει, τέλος, ένα σχηματικό διάγραμμα του ελεγκτή, όπου

φαίνονται οι είσοδοι και οι έξοδοί του, καθώς και οι εσωτερικοί του καταχωρητές.

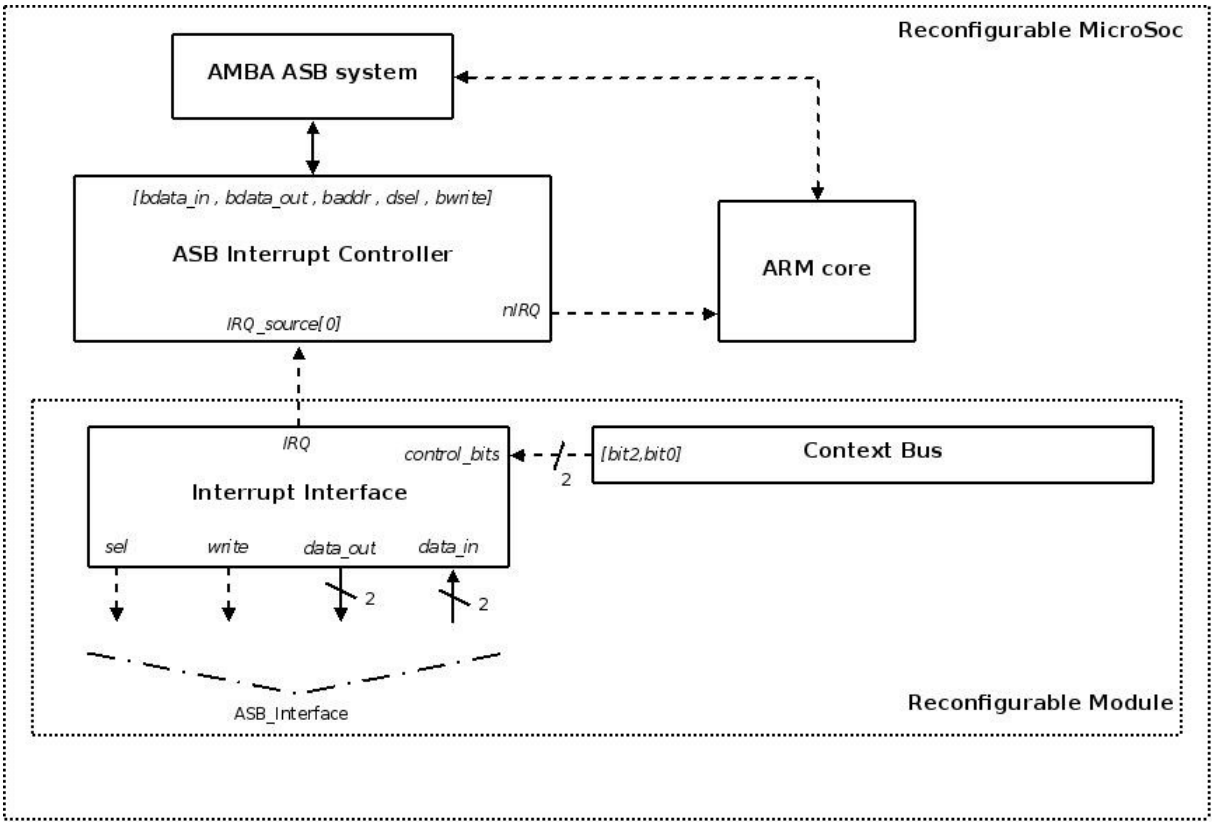


Σχήμα 5.27: Ο ελεγκτής διακοπών του rMicroSoc

Συνοψίζοντας τα όσα έχουν αναφερθεί παραπάνω, το συνολικό υλικό του συστήματος που σχετίζεται με το σύστημα διακοπών που υποστηρίζει το σύστημα φαίνεται συγκεντρωμένο στο Σχήμα 5.28 της επόμενης σελίδας.

Ο συνολικός έλεγχος του συστήματος διακοπών, όσον αφορά στο επαναδιατάξιμο περιφερειακό, επαφίεται στη μνήμη διαμόρφωσης και στις λέξεις διαμόρφωσης. Πιο συγκεκριμένα, το περιφερειακό ζητάει διακοπή από το σύστημα όταν και μόνο τα bit0 και bit2 του εσωτερικού διαδρόμου διαμόρφωσης έχουν την τιμή '1'. Η αίτηση υλοποιείται μέσω της διεπαφής διακοπών του περιφερειακού, η οποία όταν λάβει την επιβεβαίωση από τα παραπάνω bits του διαδρόμου διαμόρφωσης τότε θέτει την τιμή της εξόδου της *IRQ* στην τιμή '1'. Η έξοδος αυτή συνδέεται με ένα συγκεκριμένο bit (εδώ έχει επιλεγεί το bit0) της εισόδου *IRQ_sources* του ελεγκτή διακοπών ASB. Η τελευταία μονάδα αποτελεί ξεχωριστό περιφερειακό του MicroSoc και είναι συνδεδεμένη πάνω στον κυρίως διάδρομο AMBA-ASB του συστήματος. Ο συγκεκριμένος ελεγκτής διακοπών αποθηκεύει την κατάσταση αιτήσεων των διαφόρων περιφερειακών, για τα οποία είναι υπεύθυνος, σε έναν εσωτερικό καταχωρητή (τον *IRQ_STATUS*). Αυτό το γεγονός πρακτικά σημαίνει ότι το περιφερειακό δεν είναι αναγκαίο να έχει συνεχώς την έξοδο αίτησης διακοπής στο '1', αλλά αρκεί ένας μόνο κύκλος για να γίνει ο ελεγκτής ενήμερος. Όταν υπάρχει έστω και μία αίτηση από περιφερειακά, δηλαδή όταν έστω και ένα bit του καταχωρητή *IRQ_STATUS* έχει την τιμή '1', τότε ο ελεγκτής διακοπών κάνει -με τη σειρά του- αίτηση για διακοπή αντιπροσωπεύοντας τα περιφερειακά του ASB διαδρόμου απευθείας στον ARM. Στην ουσία ο ελεγκτής παρεμβάλλεται μεταξύ των διαφόρων περιφερειακών που δυνητικά μπορούν να ζητήσουν διακοπή από το σύστημα και του ίδιου του ARM, έτσι ώστε να επιτευχθεί ο χρονισμός που να συμβαδίζει με τις απαιτήσεις του τελευταίου, αλλά και να ρυθμιστούν θέματα παρεμπόδισης

της δυνατότητας διακοπής σε ορισμένα περιφερειακά. Μετά την αίτηση ο ARM θα προσπαθήσει να εξυπηρετήσει τη διακοπή ξεκινώντας πρώτα με επικοινωνία με τον ελεγκτή για να αναγνωρίσει από τα περιεχόμενα του *IRQ_STATUS* ποια περιφερειακά ζήτησαν διακοπή. Περισσότερες πληροφορίες σχετικά με τον τρόπο που ο ARM εξυπηρετεί τις αιτήσεις για διακοπές που δέχεται από τα περιφερειακά παρέχονται στην επόμενη υποενότητα.



Σχήμα 5.28: Το υλικό για την υποστήριξη του συστήματος διακοπών του rMicroSoc

Υπάρχει ακόμα ένα σημείο που χρειάζεται ανάλυση και είναι το πώς καθορίζεται από το περιεχόμενο της μνήμης διαμόρφωσης σε ποια σημεία πρέπει το περιφερειακό να ζητήσει διακοπή. Ο ελεγκτής διακοπών αποθηκεύει την κατάσταση αίτησης διακοπής του περιφερειακού σε έναν καταχωρητή και ως εκ τούτου δεν απαιτείται από το περιφερειακό να έχει διαρκώς την έξοδο αίτησης διακοπής στο 1, όταν χρειάζεται όντως διακοπή. Για την ακρίβεια αρκεί μόνο να γίνει αυτό σε έναν κύκλο ρολογιού. Οπότε η συνήθης τεχνική που ακολουθείται κατά τον προγραμματισμό, είναι στο τέλος του επιθυμητού συνόλου λέξεων που τοποθετούνται στη μνήμη διαμόρφωσης να τοποθετούνται επιπλέον δύο ειδικές λέξεις διαμόρφωσης και οι δύο σε κατάσταση stall. Η πρώτη από αυτές έχει το bit1 (και το bit0) στην τιμή '1' και όλα τα άλλα στην τιμή '0', οπότε με αυτή τη λέξη ζητείται διακοπή από το master. Η δεύτερη από αυτές έχει το bit2 (και το bit0) στην τιμή '1' και όλα τα άλλα στην τιμή 0, οπότε με αυτή τη λέξη υπάρχει παύση στη λειτουργία της μνήμης διαμόρφωσης, δηλαδή η μνήμη περιμένει στο ίδιο σημείο μέχρι να γίνει εξυπηρέτηση της διακοπής από το master και να καθοριστεί ποιο σύνολο πρέπει να εκτελεστεί στη συνέχεια.

5.4.5.Ρουτίνες Εξυπηρέτησης Διακοπών

Στην υποενότητα αυτή θα περιγραφεί ο τρόπος με τον οποίο γράφεται μια συνάρτηση για την εξυπηρέτηση διακοπών από περιφερειακά στο διάδρομο ASB στο rMicroSoc με βάση το σύστημα διακοπών που περιγράφηκε στην προηγούμενη παράγραφο και ειδικά για τις διακοπές από το επαναδιατάξιμο περιφερειακό.

Η μόνη παρέμβαση που απαιτείται από την πλευρά του χρήστη στο σύστημα διακοπών του ARM είναι η συγγραφή ρουτίνων εξυπηρέτησης των τύπων εξαιρέσεων που επιθυμείται να υποστηρίζεται από το σύστημα και η σωστή σύνταξη των περιεχομένων της περιοχής μνήμης $(00)_{16}$ - $(1C)_{16}$, με τα διανύσματα διακοπών, ώστε σε κάθε εξαίρεση ο έλεγχος του προγράμματος να πηγαίνει στην αντίστοιχη -σωστή- ρουτίνα εξυπηρέτησης. Στο σύστημα rMicroSoc η διαδικασία έχει απλοποιηθεί και αρκεί η συγγραφή των ρουτίνων εξυπηρέτησης. Ουσιαστικά η αντιστοίχηση μιας ρουτίνας με την αντίστοιχη διακοπή της (IRQ/FIQ) -οπότε κατ`επέκταση η σύνταξη του πίνακα διακοπών- γίνεται αυτόματα από το πρόγραμμα μεταγλώττισης και ανάπτυξης των εφαρμογών για τον ARM.

Για τις διακοπές τύπου IRQ η αντίστοιχη ρουτίνα εξυπηρέτησης πρέπει να είναι μια ρουτίνα με όνομα *IrqHandler*, ενώ για τις διακοπές τύπου FIQ η αντίστοιχη ρουτίνα είναι η *FiqHandle*r. Αυτές οι ρουτίνες πρέπει να υπάρχουν σαν ξεχωριστές συναρτήσεις μέσα στο αρχείο κώδικα που περιέχει και τη συνάρτηση C_Entry με το κυρίως πρόγραμμα που εκτελεί ο ARM. Όταν υπάρξει αίτηση για κάποια διακοπή, τότε καλείται αυτόματα η αντίστοιχη ρουτίνα για την εξυπηρέτησής της.

Από την πλευρά της λειτουργίας της και του περιεχομένου της, η ρουτίνα εξυπηρέτησης διακοπών θα πρέπει αρχικώς να αλληλεπιδράσει με τον ελεγκτή διακοπών ASB και να χρησιμοποιήσει τους εσωτερικούς καταχωρητές που αυτός περιέχει για να λάβει πληροφορίες σχετικά με τις αιτήσεις που έχουν γίνει. Επειδή θεωρητικά μπορεί να υπάρχουν πολλές πιθανές πηγές για αιτήσεις διακοπής, θα πρέπει κανονικά η ρουτίνα να εξετάσει όλες τις πηγές και στη συνέχεια σε περίπτωση που όντως υπάρχουν παραπάνω της μία αιτήσεις να καθορίσει -ανάλογα με τις σχεδιαστικές επιλογές που έχουν γίνει- ποιες και με ποια σειρά θα εξυπηρετηθούν. Ωστόσο για την απλοποίηση της υλοποίησης και εξαιτίας του γεγονότος ότι στο σύστημα υπάρχει μόνο μία πηγή διακοπών, το επαναδιατάξιμο περιφερειακό, η ρουτίνα εξυπηρέτησης που περιγράφεται εδώ και που τελικά χρησιμοποιήθηκε στα παραδείγματα εφαρμογών που απεικονίστηκαν στο σύστημα σχεδιάστηκε για μία μόνο πηγή διακοπής.

Σχετικά με τη γενική περίπτωση των πολλαπλών πηγών διακοπής, θα πρέπει αρχικά μέσω της κύριας ρουτίνας εξυπηρέτησης όλων των διακοπών του συστήματος να γίνει ανάγνωση και αποθήκευση σε τοπική μεταβλητή του

περιεχομένου του καταχωρητή IRQ_STATUS του ελεγκτή διακοπών, ώστε να εντοπιστεί ποια περιφερειακά έχουν ζητήσει διακοπή. Υπενθυμίζεται ότι κάθε bit του καταχωρητή έχει αντιστοιχιστεί νοητά με ένα διαφορετικό περιφερειακό και αν κάποιο bit έχει την τιμή 1 τότε το αντίστοιχο περιφερειακό έχει ζητήσει διακοπή. Οπότε με την ανάγνωση του περιεχομένου του IRQ_STATUS είναι δυνατή η αναγνώριση των πηγών που έχουν διακοπές προς εξυπηρέτηση. Η ανάγνωση μπορεί να γίνει στην πράξη με χρήση μιας μεταβλητής-δείκτη σε int, έστω *ptr*. Επειδή ο καταχωρητής IRQ_STATUS στον χάρτη απεικόνισης του συστήματος αντιστοιχεί σε διευθύνσεις τύπου 40xxxxxx, με τις παρακάτω δύο γραμμές μεταφέρονται τα περιεχόμενα του καταχωρητή σε μια τοπική μεταβλητή *istr*:

```
ptr = (int *)40000000;
istr = *ptr
```

Αμέσως μετά την ανάγνωση, πρέπει να διαγραφούν τα περιεχόμενα του IRQ_STATUS, αφού αυτά υπάρχουν τώρα στην τοπική μεταβλητή *istr*. Στη συνέχεια ακολουθεί αυτή καθ`εαυτή η εξυπηρέτηση των διακοπών ανάλογα με τις προτεραιότητες που έχουν δοθεί. Για την ευκολότερη και πιο ιεραρχημένη εξυπηρέτηση η τεχνική που ακολουθείται είναι να γράφεται μια διαφορετική συνάρτηση για κάθε πιθανή πηγή διακοπής και να καλείται αυτή -μέσα από τον κώδικα της κύριας ρουτίνας εξυπηρέτησης διακοπών- όταν πρέπει να εξυπηρετηθεί αίτηση από το συγκεκριμένο περιφερειακό. Στην περίπτωση του επαναδιατάξιμου MicroSoc, στη μορφή που χρησιμοποιήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας, έχει γραφτεί μόνο μία, αφού η ρουτίνα εξυπηρέτησης αφορά ουσιαστικά μόνο μία πηγή διακοπής.

```
void IrqHandler(void)
{
        volatile unsigned int *ptr;
        int istr, i;

        void (*irq_handler[16])(void);
        irq_handler[0] = &irq0_handler;

        // Identify the peripheral
        // by reading the interrupt status register (ISTR)
        // Store it to istr
        ptr = (unsigned int *)IRQ_STATUS;
        istr = *ptr;

        // ISTR format: xxxx xxxx xxxx xxxx
        // bit j = 1 means peripheral j has demanded a request
        // so run irqj_handler
        // In case of more than one IRQ
        // give priority to smaller IRQs
      if(istr & 0x0001)
            irq_handler[0]();

        // clear the ISTR
        ptr = (unsigned int *)IRQ_CLEAR;
        *ptr = 0x0001;
}
```

Κώδικας 5-5.1: Η κεντρική ρουτίνα εξυπηρέτησης για όλες τις διακοπές του συστήματος MicroSoc

Θα περιγραφεί ακολούθως η συνάρτηση αυτή που αντιστοιχεί στην εξυπηρέτηση

των διακοπών από το επαναδιατάξιμο περιφερειακό. Υπενθυμίζεται ότι το περιφερειακό αυτό είναι σχεδιασμένο να ζητάει διακοπές μόνο όταν ολοκληρωθεί η εκτέλεση των λέξεων διαμόρφωσης που υπάρχουν σε ένα set της μνήμης διαμόρφωσης. Κύριος σκοπός των διακοπών αυτών είναι η πιθανή ανάγνωση αποτελεσμάτων των υπολογισμών που υλοποιήθηκαν από το set που ολοκληρώθηκε και έχουν αποθηκευτεί στα τοπικά αρχεία καταχωρητών των επαναδιατάξιμων κελιών και ο καθορισμός του επόμενου set της μνήμης διαμόρφωσης που πρέπει να εκτελεστεί, αν αυτό είναι φυσικά απαραίτητο. Από τα παραπάνω γίνεται αντιληπτό ότι μια βασική ενέργεια που πρέπει αρχικά να γίνει είναι η αναγνώριση του set που έχει ολοκληρωθεί, διότι μπορεί να υπάρχει διαφορετική αντιμετώπιση για κάθε set. Η αναγνώριση αυτή γίνεται με τη βοήθεια ενός καταχωρητή που υπάρχει μέσα στη διεπαφή διακοπών στο επαναδιατάξιμο περιφερειακό. Ο συγκεκριμένος καταχωρητής περιέχει το νούμερο που αντιστοιχεί στο set που εκτελείται στη μνήμη διαμόρφωσης. Οπότε η συνάρτηση εξυπηρέτησης διαβάζει τον καταχωρητή αυτό, που αντιστοιχεί σε διευθύνσεις της μορφής 54xxxxxx και ανάλογα με το περιεχόμενό του καταλαβαίνει ποιο set ολοκληρώθηκε.

Για την πιο κομψή εμφάνιση του κώδικα οι ενέργεις που πρέπει να γίνουν για κάθε περίπτωση, ανάλογα με το set που τελείωσε, υπάρχουν σε ξεχωριστές συναρτήσεις. Εν συνεχεία θα εξεταστεί το γενικό πλαίσιο των ενεργειών που πρέπει να γίνουν στις συναρτήσεις αυτές. Αρχικά πρέπει να γίνει ανάγνωση -αν αυτό απαιτείται- από τους τοπικούς καταχωρητές των επαναδιατάξιμων κελιών και αποθήκευση των περιεχομένων στη μνήμη του συστήματος. Η ανάγνωση γίνεται με απευθείας χρήση των διευθύνσεων των καταχωρητών των κελιών, όπως αυτές εμφανίζονται στο χάρτη απεικόνισης (περισσότερες πληροφορίες στην επόμενη ενότητα). Έπειτα, πρέπει να ακολουθήσουν οι ενέργειες εκείνες, οι οποίες θα καθορίσουν τόσο στη μνήμη δεδομένων όσο και στη μνήμη διαμόρφωσης ποιο set θα ακολουθήσει -αν υπάρχει τέτοιο. Πριν γίνουν οι απαραίτητες ρυθμίσεις στις μνήμες, τίθεται το περιεχόμενο του καταχωρητή συγχρονισμού στο 0 ώστε να επιτευχθεί, αργότερα, ο απαραίτητος συγχρονισμός με το να τεθεί στην τιμή 1.

Στη συνέχεια πρέπει να ακολουθήσει η εγγραφή του εσωτερικού καταχωρητή της διεπαφής διακοπών του περιφερειακού με την τιμή του αριθμού του επόμενου set της μνήμη διαμόρφωσης που θα εκτελεστεί, εκτός κι αν δεν υπάρχει οπότε το βήμα αυτό παραλείπεται. Το επόμενο βήμα ακολουθείται αν υπάρχει πρόβλεψη να γίνουν βρόχοι επανάληψης στο περιεχόμενο των μνημών διαμόρφωσης ή/και δεδομένων και περιλαμβάνει την εγγραφή των καταχωρητών επαναλήψεων της μνήμης διαμόρφωσης ή/και δεδομένων με την κατάλληλη τιμή του αριθμού των συνολικών επαναλήψεων των βρόχων. Ακολουθεί η εγγραφή των καταχωρητών επόμενης διεύθυνσης των μνημών διαμόρφωσης και δεδομένων με την πρώτη διεύθυνση του επόμενου προς εκτέλεση set της μνήμης διαμόρφωσης και την πρώτη διεύθυνση στη μνήμη δεδομένων των δεδομένων που αντιστοιχούν στο set αυτό. Τέλος τίθεται ο καταχωρητής συγχρονισμού στην τιμή 1, ώστε να υπάρχει ο απαραίτητος συγχρονισμός στη λειτουργία των δύο μνημών. Από τη στιγμή που τελειώσει η

εγγραφή στον καταχωρητή αυτό αρχίζει η εκτέλεση των λέξεων διαμόρφωσης του νέου set της μνήμης διαμόρφωσης.

Παρακάτω ακολουθεί η συνάρτηση *irq0_handler()*, η οποία υλοποιεί την ειδική ρουτίνα εξυπηρέτησης των διακοπών που προέρχονται από το επαναδιατάξιμο περιφερειακό.

```
void irq0_handler(void) {
      int prog_number;
      volatile unsigned int *ptr;

      // Read the interrupt register to find out
      // which context program has ended
      ptr = (unsigned int *)INT_REG;
      prog_number = *ptr;

      switch(prog_number = *ptr) {
            case CONTEXT_PROG0 :
            {
                  prog0_end_handler();
                  break;
            }

            case CONTEXT_PROG1 :
            {
                  prog1_end_handler();
                  break;
            }

            case CONTEXT_PROG2 :
            {
                  prog2_end_handler();
                  break;
            }

            case CONTEXT_PROG3 :
            {
                  prog3_end_handler();
                  break;
            }
      }
}
```

Κώδικας 5-5.2: Η ρουτίνα εξυπηρέτησης διακοπών από το επαναδιατάξιμο περιφερειακό

Τέλος παρατίθεται ένα τυπικό παράδειγμα για τη συνάρτηση που υλοποιεί τις ενέργειες που πρέπει να γίνουν όταν ένα συγκεκριμένο set της μνήμης διαμόρφωσης του συστήματος ολοκληρωθεί και ζητήσει διακοπή. Όπως αναφέρθηκε και παραπάνω, η σχεδίαση περιλαμβάνει τη συγγραφή τεσσάρων τέτοιων παρόμοιων συναρτήσεων, μία για κάθε set της μνήμης. Δεν είναι απαραίτητο να περιληφθούν για όλα τα set όλες οι ενέργειες.

```
void prog0_end_handler(void)
{
	volatile unsigned int *ptr;

	// read any output from prog0 here
	// start prog1 or stop
	// -------PROG1------- //
	ptr = (unsigned int *)MEM_SYNCH_REG;
	*ptr = 0x0;
	ptr = (unsigned int *)INT_REG;
	*ptr = CONTEXT_PROG1;

	// ----------- LOOP_REG ---------- //
	// --- This is used to set the --- //
	// --- loops_register, in case --- //
	// ------ a loop is required ------//
	ptr = (unsigned int *)LOOP_CONT_REG;
	*ptr = 0x2; // give here the number of loops
	ptr = (unsigned int *)LOOP_DATA_REG;
	*ptr = 0x2; // give here the number of loops
	// ------------- END --------------//

	ptr = (unsigned int *)ARM_MEM_REG;
	*ptr = PROG1_ADDR;
	ptr = (unsigned int *)ARM_CONT_REG;
	*ptr = PROG1_ADDR;
	ptr = (unsigned int *)MEM_SYNCH_REG;
	*ptr = 0x1;

	// -------STOP------- //
	run = 0;
}
```

Πλαίσιο 5.3: Εξυπηρέτηση διακοπών από συγκεκριμένο set της μνήμης

5.5. Διεπαφή για το Διάδρομο ASB

Ένα σημαντικό συστατικό στοιχείο του επαναδιατάξιμου περιφερειακού είναι το ASB slave interface, δηλαδή το στοιχείο εκείνο που αποτελεί τη διεπαφή προς το διάδρομο AMBA ASB του κύριου συστήματος. Η διεπαφή αυτή καθιστά το επαναδιατάξιμο περιφερειακό ένα "σκλάβο" του συστήματος MicroSoC και υλοποιεί όλη την επικοινωνία μεταξύ του περιφερειακού και master, δηλαδή του επεξεργαστή ARM. Έτσι είναι υπεύθυνη για την υλοποίηση και το σωστό χρονισμό για όλες τις μεταφορές ανάγνωσης και εγγραφής προς τον επεξεργαστή και τις απαντήσεις μεταφοράς που είναι υποχρεωμένο να στείλει το περιφερειακό προς τον επεξεργαστή σε κάθε μεταφορά. Επίσης και για το σωστό χρονισμό και αντιμετώπιση των μεταφορών εγγραφής και ανάγνωσης από τον επεξεργαστή προς το περιφερειακό. Στην πραγματικότητα αποτελεί το συνδετικό κρίκο μεταξύ του περιφερειακού και του υπόλοιπου συστήματος και είναι αυτό που καθιστά εφικτή τη συνεργασία των δύο αυτών μερών.

Όπως είχε αναφερθεί και κατά τη γενική παρουσίαση της αρχιτεκτονικής των

διαδρόμων AMBA, υπάρχουν κάποιες βασικές απαιτήσεις που πρέπει να ικανοποιεί ένα slave του διαδρόμου AMBA ASB. Οι απαιτήσεις αυτές στην ουσία πρέπει να ικανοποιούνται από το interface του περιφερειακού για να υπάρχει σωστή σύνδεση και επικοινωνία με το διάδρομο.

5.5.1.Απόκριση στη Μεταφορά

Η μία βασική λειτουργία που επιτελεί η διεπαφή του περιφερειακού είναι να απαντάει προς το master σε κάθε μεταφορά, με την κατάλληλη απόκριση. Όπως περιγράφηκε στην παρουσίαση των χαρακτηριστικών του AMBA ASB διαδρόμου υπάρχουν διάφορες δυνατές αποκρίσεις. Η επιλογή μεταξύ αυτών έγκειται στη διεπαφή του εκάστοτε "σκλάβου", ανάλογα με την περίσταση.

Υπάρχει η απόκριση WAIT, με την οποία ο "σκλάβος" ζητάει η μεταφορά να διαρκέσει περισσότερους κύκλους πριν ολοκληρωθεί, η απόκριση DONE, με την οποία ο "σκλάβος" δηλώνει ότι η μεταφορά έχει ολοκληρωθεί και με επιτυχία και η απόκριση ERROR, με την οποία ο "σκλάβος" δηλώνει ότι η μεταφορά έχει ολοκληρωθεί αλλά έχει διαγνωστεί κάποιο σφάλμα. Ακόμα υπάρχει η απόκριση LAST, με την οποία δηλώνεται ότι η μεταφορά έχει ολοκληρωθεί με επιτυχία και επιπλέον, ταυτόχρονα, ότι η μεταφορά ριπής πρέπει να ολοκληρωθεί και τέλος η απόκριση RETRACT, με την οποία ο "σκλάβος" δηλώνει ότι ο master πρέπει να επαναλάβει τη μεταφορά. Η επιθυμητή απόκριση δηλώνεται προς τον master με τον κατάλληλο συνδυασμό τριών ειδικών σημάτων απόκρισης. Η απόκριση από πλευράς ενός "σκλάβου" δίνεται στις αρνητικές ακμές των παλμών του ρολογιού.

Η υλοποίηση ενός "σκλάβου" -και κατ`επέκταση μιας διεπαφής για το "σκλάβο"- οποίος θα υποστηρίζει πλήρως όλες τις δυνατές αποκρίσεις σε όλες τις περιπτώσεις είναι αρκετά πολύπλοκη. Για το λόγο αυτό, η διεπαφή του επανδιατάξιμου περιφερειακού σχεδιάστηκε αρκετά πιο απλοποιημένη σε σχέση με το πλήρες μοντέλο.

Συγκεκριμένα, το περιφερειακό παρέχει μόνο δύο δυνατές αποκρίσεις, είτε WAIT είτε DONE. Με δεδομένο μάλιστα ότι δεν υπάρχει κάποιος τρόπος ώστε η διεπαφή να ενημερώνεται από τους διάφορους καταχωρητές ή μνήμες του περιφερειακού για τη διάρκεια μιας εγγραφής ή ανάγνωσης, έχει επιλεγεί σε κάθε πιθανή μεταφορά το περιφερειακό να αποκρίνεται με έναν κύκλο WAIT και στη συνέχεια με κύκλο DONE. Δηλαδή το περιφερειακό ζητάει από τον master να επεκτείνει την κάθε μεταφορά που υλοποιεί κατά έναν κύκλο υποχρεωτικά, πριν δηλώσει ότι η μεταφορά μπορεί να ολοκληρωθεί. Αυτό γίνεται ώστε οι εγγραφές ή αναγνώσεις στις μνήμες -κυρίως- να προλάβουν να ολοκληρωθούν πριν τερματιστεί η μεταφορά.

Για την παραπάνω συμπεριφορά έχει υλοποιηθεί στην ουσία μια μηχανή καταστάσεων, με δύο πιθανές καταστάσεις. Στην μία κατάσταση η διεπαφή υλοποιεί απόκριση τύπου WAIT και στην άλλη η διεπαφή υλοποιεί απόκριση

τύπου DONE. Η διεπαφή αντιλαμβάνεται τις μεταφορές παρατηρώντας το σήμα dsel που αντιστοιχεί στο επαναδιατάξιμο περιφερειακό στις αρνητικές ακμές του ρολογιού του συστήματος. Όταν το συγκεκριμένο σήμα επιλογής είναι 1, τότε η μηχανή καταστάσεων αλλάζει κατάσταση, οπότε αν είναι σε κατάσταση WAIT μεταβαίνει σε κατάσταση DONE και αντίστροφα. Φυσικά σε κάθε κατάσταση τίθονται και αναλόγως τα σήματα απόκρισης των μεταφορών, ώστε να δηλωθεί στο master η κατάλληλη απόκριση.

5.5.2.Αντιμετώπιση των Μεταφορών

Η άλλη λειτουργία που επιτελεί η διεπαφή του περιφερειακού είναι η σωστή αντιμετώπιση των διαφόρων μεταφορών που αφορούν το περιφερειακό, είτε πρόκειται για μεταφορές ανάγνωσης είτε εγγραφής, από το master προς το slave.

Η ανάγκη ύπαρξης της διεπαφής ως ο ενδιάμεσος μεταξύ του περιφερειακού και του εξωτερικού συστήματος εμφανίζεται διότι υπάρχουν πολλές πιθανές υπομονάδες μέσα στο περιφερειακό με τις οποίες είναι δυνατό να χρειάζεται ο master να επικοινωνήσει. Έτσι η διεπαφή θα πρέπει να μπορεί να διαχωρίζει πότε πρόκειται για μεταφορά εγγραφής και πότε για μεταφορά ανάγνωσης και να επιλέγει τον κατάλληλο εσωτερικό καταχωρητή. Οι διάδρομοι διεύθυνσης και δεδομένων και το σήμα επιλογής εγγραφής του διαδρόμου AMBA ASB δε διοχετεύονται απευθείας σε κάθε καταχωρητή του περιφερειακού, αλλά πηγαίνουν στην διεπαφή και από εκεί ξεκινούν ανάλογα σήματα προς τους καταχωρητές.

Ο διαχωρισμός μεταξύ των δύο ειδών μεταφοράς γίνεται με βάση το σήμα BWRITE του διαδρόμου AMBA ASB, το οποίο δηλώνει πότε γίνεται εγγραφή και πότε ανάγνωση. Συγκεκριμένα αν το σήμα πάρει την τιμή 0 τότε η μεταφορά που πραγματοποιείται είναι μεταφορά ανάγνωσης, ενώ αν πάρει την τιμή 1 τότε η μεταφορά είναι μεταφορά εγγραφής. Η διεπαφή λοιπόν ελέγχει το συγκεκριμένο σήμα και ενημερώνει ανάλογα τον καταχωρητή με τον οποίον γίνεται η επικοινωνία.

Η επιλογή του κατάλληλου καταχωρητή που σχετίζεται με την εκάστοτε μεταφορά είναι μια πολύ σημαντική λειτουργία της διεπαφής. Βασίζεται στο γεγονός ότι στα συστήματα με διαδρόμους AMBA -όπως το επαναδιατάξιμο σύστημα που υλοποιήθηκε- όλοι οι καταχωρητές του συστήματος, τους οποίους μπορεί να προσπελάσει ο master, έχουν απευθείας απεικόνιση στη μνήμη. Αντιστοιχίζονται δηλαδή με συγκεκριμένες θέσεις στη μνήμη και αντιμετωπίζονται σαν κοινές θέσεις της κεντρικής μνήμης του συστήματος. Για την απεικόνιση των καταχωρητών του επαναδιατάξιμου περιφερειακού υπεύθυνη είναι η διεπαφή.

Συνολικά υπάρχουν 7 θέσεις μνήμης μέσα στο περιφερειακό, με τις οποίες

επιτρέπεται ο master να επικοινωνήσει. Λόγω του πλήθους των μονάδων αυτών χρησιμοποιούνται 4 bits του διαδρόμου διευθύνσεων για την επιλογή τους και συγκεκριμένα τα bits 31 έως και 28. Ο τρόπος επιλογής των καταχωρητών είναι ο εξής. Κάθε μια από τις μονάδες αυτές έχει μια είσοδο επιλογής, η οποία τίθεται στο 1 όταν ταυτόχρονα η διεύθυνση του διαδρόμου διευθύνσεων αντιστοιχεί σε μια συγκεκριμένη μονάδα και το σήμα BWRITE έχει κατάλληλη τιμή (ανάλογα με το αν η μονάδα αυτή είναι για ανάγνωση ή εγγραφή). Η μονάδα της διεπαφής παράγει σήματα επιλογής για κάθε μία από τις μονάδες αυτές. Τα σήματα αυτά ενεργοποιούνται ανάλογα με την τιμή που λαμβάνουν τα bit 31-28 του διαδρόμου διευθύνσεων και επιλέγουν κατάλληλα ποια μονάδα πρέπει να συμμετάσχει στην τρέχουσα επικοινωνία με το master. Σε κάθε μεταφορά μόνο ένα από τα σήματα αυτά είναι ενεργό. Η διεπαφή υλοποιεί κατ` αυτό τον τρόπο έναν αποκωδικοποιητή. Ανάλογα με την περίπτωση, η διεπαφή πρέπει να αποκωδικοποιήσει τις πληροφορίες που στέλνονται από το master -μέσω του διαδρόμου διευθύνσεων- και να ενεργοποιήσει τα κατάλληλα σήματα ελέγχου στις μονάδες. Παρακάτω δίνεται μια σύντομη παρουσίαση των μονάδων του περιφερειακού προς ανάγνωση/εγγραφή από τον master.

- *Καταχωρητής επόμενης διεύθυνσης της μνήμης διαμόρφωσης*. Ο συγκεκριμένος καταχωρητής έχει μελετηθεί στην ενότητα για το σύστημα μνημών του περιφερειακού. Όπως είχε τότε αναφερθεί, ο συγκεκριμένος καταχωρητής περιέχει την επόμενη διεύθυνση ανάγνωσης από τη μνήμη διαμόρφωσης. Οι μεταφορές που αφορούν τον καταχωρητή αυτό είναι εκείνες οι μεταφορές με τις οποίες ο master εγγράφει μια νέα διεύθυνση ανάγνωσης, η οποία με τον τρόπο που είναι σχεδιασμένη η μνήμη διαμόρφωσης είναι -συνήθως- η αρχική διεύθυνση ενός από τα 4 set της μνήμης. Ο καταχωρητής αυτός έχει μια είσοδο επιλογής εγγραφής από τον master, η οποία τίθεται στο 1 από τη διεπαφή όταν ο διάδρομος διευθύνσεων περιέχει την τιμή που αντιστοιχεί στον καταχωρητή και το σήμα BWRITE έχει την τιμή 1. Ταυτόχρονα, σε αυτήν την περίπτωση, τα δεδομένα από το διάδρομο διευθύνσεων BDATA μεταφέρονται στην είσοδο δεδομένων εγγραφής του συγκεκριμένου καταχωρητή.

- *Καταχωρητής επόμενης διεύθυνσης της μνήμης δεδομένων.* Η αντιμετώπιση του συγκεκριμένου καταχωρητή είναι ίδια ακριβώς με αυτήν του προηγούμενου. Άλλωστε η ίδια η λειτουργία των δύο αυτών καταχωρητών είναι πανομοιότυπη. Η μόνη διαφορά έγκειται στη διαφορετική τιμή του διαδρόμου διευθύνσεων για την οποία επιλέγεται ο καταχωρητής που αντιστοιχεί στη μνήμη δεδομένων.

- *Καταχωρητής για εσωτερικά άλματα στη μνήμη διαμόρφωσης.* Ο καταχωρητής αυτός έχει μελετηθεί στην ενότητα για τα συστήματα μνήμης, στην ειδική υποενότητα για τη δυνατότητα της μνήμης διαμόρφωσης να υποστηρίζει εσωτερικά λογικά άλματα στην εκτέλεση των λέξεων διαμόρφωσης. Ειδικότερα ο καταχωρητής αυτός περιέχει το πλήθος των επαναλήψεων που απομένουν ακόμα προς εκτέλεση. Πρέπει να υπάρχει η δυνατότητα ο καταχωρητής να εγγράφεται από τον master

με μια αρχική τιμή για το συνολικό πλήθος των αλμάτων που θα εκτελεστούν. Για το λόγο αυτό υπάρχει μια είσοδος για επιλογή εγγραφής από τον master. Η είσοδος αυτή ελέγχεται απευθείας από τη διεπαφή, η οποία την θέτει στην τιμή 1 όταν το σήμα BWRITE πάρει την τιμή 1 και τα 4 πιο σημαντικά bits του διαδρόμου διευθύνσεων πάρουν την τιμή που αντιστοιχεί στον καταχωρητή αυτόν. Τότε επιτρέπεται η εγγραφή νέων δεδομένων στον καταχωρητή, οπότε η διεπαφή μεταφέρει τα δεδομένα από το διάδρομο δεδομένων προς την είσοδο δεδομένων εγγραφής του καταχωρητή.

- *Καταχωρητής για εσωτερικά άλματα στη μνήμη δεδομένων.* Ο συγκεκριμένος καταχωρητής αντιμετωπίζεται από τη διεπαφή του περιφερειακού ακριβώς όπως και ο προηγούμενος. Εξάλλου η λειτουργία τους είναι πανομοιότυπη, όπως έχει αναφερθεί και σε προηγούμενη ενότητα. Η μόνη διαφορά εντοπίζεται στην διαφορετική τιμή των 4 πιο σημαντικών bits του διαδρόμου διευθύνσεων που επιλέγει τον καταχωρητή για τη μνήμη δεδομένων, σε σχέση με αυτόν της μνήμης διαμόρφωσης.

- *Καταχωρητής συγχρονισμού των δύο μνημών.* Η ανάγκη ύπαρξης του καταχωρητή αυτού και η λειτουργία του έχουν περιγραφεί αναλυτικά σε προηγούμενη ενότητα. Από όσα έχουν αναφερθεί εκεί ο συγκεκριμένος καταχωρητής πρέπει να εγγράφεται από τον master. Η λογική ελέγχου του από την διεπαφή είναι ακριβώς η ίδια με όλους τους προηγούμενους καταχωρητές που περιγράφηκαν παραπάνω. Δηλαδή υπάρχει μια είσοδος ελέγχου εγγραφής από τον master, η οποία ελέγχεται από τη διεπαφή. Η είσοδος αυτή τίθεται στην τιμή 1 όταν η διεύθυνση του διαδρόμου διευθύνσεων πάρει τιμή που αντιστοιχεί στον καταχωρητή αυτό και ταυτόχρονα το σήμα BWRITE γίνει 1, κάτι που δηλώνει μεταφορά εγγραφής. Σε αυτή την περίπτωση γίνεται επίσης μεταφορά μέσω της διεπαφής των δεδομένων από το διάδρομο δεδομένων προς την είσοδο δεδομένων εγγραφής του καταχωρητή. Βέβαια ο καταχωρητής έχει μέγεθος 1 bit μόνο, οπότε μεταφέρεται μόνο το bit0 του διαδρόμου δεδομένων.

- *Μνήμη δεδομένων.* Ο master έχει μόνο τη δυνατότητα εγγραφής της μονάδας αυτής. Υπενθυμίζεται ότι κάθε σύνολο της μνήμης δεδομένων μπορεί να εγγράφεται ανεξάρτητα από τα υπόλοιπα, ένα όμως σε κάθε εγγραφή. Συνεπώς, σε κάθε μεταφορά εγγραφής προς τη μνήμη πρέπει να καθοριστεί σε ποιο σύνολο γίνεται η εγγραφή και σε ποια διεύθυνση στο σύνολο αυτό, εκτός από τα δεδομένα εγγραφής. Για την επιλογή του συνόλου, η μνήμη διαθέτει μια είσοδο ελέγχου με μέγεθος 4 bits και κάθε bit αντιστοιχεί σε ένα από τα 4 σύνολα. Θέτοντας 1 το αντίστοιχο bit της εισόδου αυτής, η διεπαφή δηλώνει σε ποιο σύνολο πρέπει να γραφούν τα δεδομένα που προωθεί στη μνήμη. Ακόμα, η διεπαφή θέτει κατάλληλα την είσοδο διεύθυνσης εγγραφής της μνήμης δεδομένων, με τα δεδομένα που λαμβάνει από το master. Τα παραπάνω συμβαίνουν όταν το σήμα BWRITE

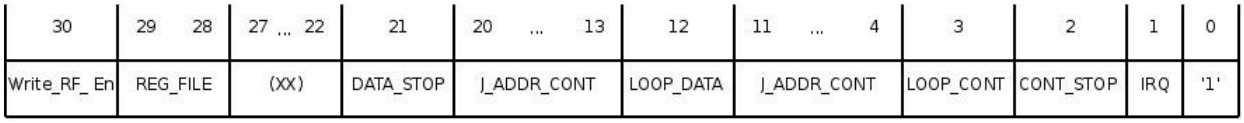
πάρει την τιμή 1 και το σήμα επιλογής της μονάδας της μνήμης δεδομένων έχει ενεργοποιηθεί.

- *Επαναδιατάξιμος πίνακας.* Σύμφωνα με τη σχεδίαση του συστήματος, ο master έχει πρόσβαση ανάγνωσης στις εξόδους και στους τοπικούς καταχωρητές όλων των κελιών του επαναδιατάξιμου πίνακα. Συνεπώς όταν ο master επικοινωνεί με τον επαναδιατάξιμο πίνακα, πρέπει να καθοριστεί ο αριθμός του κελιού και ο καταχωρητής από το κελί αυτό που πρέπει να προσπελαστεί. Ο πίνακας διαθέτει μια είσοδο επιλογής ανάγνωσης κελιού, μεγέθους 4 bits (συνολικά υπάρχουν 16 κελιά), και μία είσοδο επιλογής ανάγνωσης καταχωρητή, μεγέθους 3 bits (συνολικά υπάρχουν 4 καταχωρητές σε κάθε κελί). Η διεπαφή πρέπει να θέσει τις δύο αυτές εισόδους του πίνακα στις κατάλληλες τιμές, κάθε φορά που το σήμα BWRITE έχει την τιμή 0 και ο επαναδιατάξιμος πίνακας επιλέγεται για ανάγνωση.

- *Ελεγκτής διακοπών.* Ο ελεγκτής διακοπών έχει τη δυνατότητα είτε να εγγραφεί είτε να διαβαστεί από το master. Για το λόγο αυτό έχει ένα σήμα ελέγχου (1 bit) που η τιμή του καθορίζει εγγραφή ή ανάγνωση -ακολουθώντας την τιμή του σήματος BWRITE. Ακόμα, ο ελεγκτής διαθέτει ένα σήμα επιλογής λειτουργίας, που ενεργοποιείται και στη μία και στην άλλη περίπτωση. Συνεπώς όταν ο ελεγκτής επιλεγεί από το master για μεταφορά, η διεπαφή πρέπει να ενεργοποιήσει το σήμα επιλογής λειτουργίας του ελεκγτή και να διακρίνει δύο περιπτώσεις, ανάλογα με την τιμή του σήματος BWRITE. Αν αυτό έχει την τιμή 1 (εγγραφή), τότε θέτει κατάλληλα το σήμα ελέγχου εγγραφής/ανάγνωσης του ελεγκτή και του μεταφέρει τα δεδομένα προς εγγραφή. Αν πρόκειται για μεταφορά ανάγνωσης, η διεπαφή απλώς θέτει το σήμα ελέγχου εγγραφής/ανάγνωσης του ελεγκτή στο 0.

5.6. Λέξη Διαμόρφωσης σε Κατάσταση Stall

Με βάση όλες τις παραπάνω ενότητες μπορούν να περιγραφούν συγκεντρωτικά όλα τα πεδία της λέξης διαμόρφωσης, όταν αυτή βρίσκεται σε κατάσταση stall, δηλαδή το λιγότερο σημαντικό bit έχει την τιμή 1. Στο παρακάτω σχεδιάγραμμα φαίνονται συγκεντρωτικά τα πεδία αυτά.



Σχήμα 5.29: Τα πεδία της λέξης διαμόρφωσης σε κατάσταση stall

Με βάση και τα όσα φαίνονται στο Σχήμα 5.29, τα πεδία που υπάρχουν είναι τα εξής:

- *IRQ* – Το πεδίο αυτό έχει μέγεθος 1 bit και καθορίζει αν το περιφερειακό

κάνει αίτηση για διακοπή προς το κεντρικό σύστημα και τον ARM. Για την ακρίβεια όταν το πεδίο πάρει την τιμή 1, τότε δίνεται σήμα προς τον ελεγκτή διακοπών ASB στο κυρίως σύστημα ότι το επαναδιατάξιμο περιφερειακό ζητάει διακοπή από τον ARM. Όπως έχει αναφερθεί, η αίτηση αυτή αποθηκεύεται στον καταχωρητή IRQ_STATUS εσωτερικά μέσα στον ελεγκτή διακοπών. Αντίθετα αν το πεδίο έχει την τιμή 0, τότε δε ζητείται διακοπή.

- *CONT_STOP* – Έχει μέγεθος επίσης 1 bit και καθορίζει αν θα συνεχίζεται κανονικά η εκτέλεση των λέξεων διαμόρφωσης ή όχι. Συγκεκριμένα αν το πεδίο πάρει την τιμή 1 τότε η επόμενη διεύθυνση ανάγνωσης από τη μνήμη διαμόρφωσης, όπως αυτή καθορίζεται από το κύκλωμα γύρω από τον καταχωρητή επόμενης διεύθυνσης, παραμένει η ίδια με την προηγούμενη διεύθυνση. Δηλαδή η διεύθυνση ανάγνωσης εποδίζεται από τον να πάρει καινούρια τιμή. Με αυτό τον τρόπο η εκτέλεση από τη μνήμη διαμόρφωσης 'κολλάει' σε μία συγκεκριμένη θέση μνήμης (ουσιαστικά σε αυτήν που έχει το πεδίο OP_STOP στην τιμή 1) και μεταφέρεται διαρκώς η ίδια αυτή λέξη διαμόρφωσης προς τον επαναδιατάξιμο πίνακα, θέτοντας ταυτόχρονα και όλα τα κελιά στην κατάσταση stall. Αν αντίθετα το πεδίο έχει την τιμή 0 τότε η επόμενη διεύθυνση ανάγνωσης από τη μνήμη διαμόρφωσης μπορεί να πάρει οποιαδήποτε από τις διαθέσιμες περιπτώσεις, ανάλογα με άλλα σήματα ελέγχου, όπως έχει αναφερθεί στην υποενότητα για την εύρεση της επόμενης διεύθυνσης ανάγνωσης για τη μνήμη διαμόρφωσης.

- *LOOP_CONT* – Το πεδίο LOOP_CONT έχει μέγεθος 1 bit και καθορίζει αν πρέπει να γίνει άλμα στη μνήμη διαμόρφωσης στα πλαίσια κάποιου εσωτερικού βρόχου επανάληψης. Το bit αυτό τροφοδοτείται στην κατάλληλη είσοδο του καταχωρητή επαναλήψεων της μνήμης διαμόρφωσης και όταν έχει την τιμή 1 ο καταχωρητής εκείνος μειώνει κατά ένα το πλήθος των εναπομείναντων επαναλήψεων βρόχου. Αν το πλήθος έχει τιμή μη μηδενική μετά τη μείωση, τότε ο καταχωρητής επαναλήψεων με κατάλληλο σήμα δηλώνει στον καταχωρητή επόμενης διεύθυνσης μνήμης διαμόρφωσης ότι πρέπει να κάνει άλμα (στα πλαίσια βρόχου επανάληψης) σε κάποια θέση μνήμης, θέτοντας την επόμενη διεύθυνση ανάγνωσης στη τιμή που λαμβάνει από το πεδίο J_ADDR_CONT (περιγράφεται ακολούθως). Να σημειωθεί ότι όλες αυτές οι ενέργεις γίνονται στον ίδιο κύκλο ρολογιού, οπότε αν επιθυμείται να γίνει ένα άλμα για βρόχο επανάληψης στη μνήμη διαμόρφωσης, θα πρέπει στην ίδια λέξη διαμόρφωσης να τεθεί το πεδίο LOOP_CONT στην τιμή 1 και στο πεδίο J_ADDR_CONT να τοποθετηθεί η διεύθυνση στην οποία θα γίνει το άλμα.

- *J_ADDR_CONT* – Το πεδίο αυτό έχει μέγεθος 8 bits και παρέχει τη διεύθυνση άλματος για τους εσωτερικούς βρόχους που πρέπει να γίνουν στη μνήμη διαμόρφωσης. Τα bits του J_ADDR_CONT παρέχονται στον καταχωρητή επόμενης διεύθυνσης της μνήμης διαμόρφωσης, στην αντίστοιχη είσοδο για διεύθυνση άλματος βρόχου. Ο τελευταίος, με τη σειρά του, καθορίζει αν πρέπει να γίνει άλμα ή όχι, λαμβάνοντας ένα σήμα

από τον καταχωρητή αλμάτων μνήμης διαμόρφωσης για το αν έχει ολοκληρωθεί το πλήθος των επαναλήψεων ή όχι.

- *LOOP_DATA* – Το πεδίο αυτό είναι πανομοιότυπο ως προς την λειτουργικότητά του με το πεδίο LOOP_CONT που περιγράφηκε παραπάνω, με τη διαφορά ότι αναφέρεται σε άλματα βρόχων επανάληψης που αφορούν τη μνήμη δεδομένων. Το bit αυτό τροφοδοτείται στον καταχωρητή επαναλήψεων της μνήμης δεδομένων, ο οποίος ενημερώνει τον καταχωρητή επόμενης διεύθυνσης μνήμης δεδομένων για το αν πρέπει να γίνει άλμα ή όχι, ανάλογα με τον αριθμό των εναπομείναντων επαναλήψεων βρόχου. Αν πρέπει να γίνει άλμα ο καταχωρητής επόμενης διεύθυνσης λαμβάνει τη διεύθυνση άλματος από την ίδια λέξη διαμόρφωσης που έχει το πεδίο LOOP_CONT με τιμή 1 από το πεδίο J_ADDR_DATA, το οποίο περιγράφεται ακολούθως.

- *J_ADDR_DATA* – Το πεδίο αυτό έχει μέγεθος 8 bits και παρέχει τη διεύθυνση άλματος για βρόχο επανάληψης στη μνήμη δεδομένων. Τροφοδοτείται απευθείας στην ανάλογη είσοδο του συστήματος του καταχωρητή επόμενης διεύθυνσης της μνήμης δεδομένων και ο τελευταίος αποφασίζει αν θα ορίσει την επόμενη διεύθυνση ανάγνωσης για τη μνήμη ίση με τη διεύθυνση άλματος.

- *DATA_STOP* – Έχει μέγεθος 1 bit και όταν έχει την τιμή 1 τότε η διεύθυνση ανάγνωσης από τη μνήμη δεδομένων μπλοκάρεται από το να πάρει νέα τιμή. Η μνήμη δεδομένων παραμένει δηλαδή διαρκώς σε μία ίδια διεύθυνση ανάγνωσης, μεταφέροντας τα ίδια δεδομένα προς τον επαναδιατάξιμο πίνακα. Η λειτουργία του bit είναι πανομοιότυπη με το πεδίο CONT_STOP, μόνο που αναφέρεται στην παύση λειτουργίας της μνήμης δεδομένων.

5.7. Πλήρης Χάρτης Απεικόνισης

Λαμβάνοντας υπόψη όλα όσα έχουν αναφερθεί έως τώρα είναι δυνατόν να παρουσιαστεί μια πλήρη εικόνα του χάρτη απεικόνισης των διαφόρων στοιχείων του επαναδιατάξιμου συστήματος. Σε αυτόν πρέπει να συμπεριληφθούν όλα τα περιφερειακά του ASB, αλλά επίσης και όλοι οι εσωτερικοί καταχωρητές του επαναδιατάξιμου περιφερειακού και του ελεγκτή διακοπών, των οποίων τα εσωτερικά χαρακτηριστικά είναι γνωστά με λεπτομέρειες. Ο Πίνακας 5.7 παρουσιάζει το χάρτη απεικόνισης του συστήματος.

Επιπλέον, ο Πίνακας 5.8 παρέχει περισσότερες λεπτομέρειες για την ανάγνωση από τον επαναδιατάξιμο πίνακα.

Πίνακας 5.7: Χάρτης απεικόνισης του συστήματος rMicroSoc

<table>
<tr><th colspan="3">BA
(in hex format)</th><th>Μονάδα</th><th>Καταχωρητής</th></tr>
<tr><td colspan="3">0xxxxxxx</td><td>ROM Memory</td><td>-</td></tr>
<tr><td colspan="3">2xxxxxxx</td><td>RAM Memory</td><td>-</td></tr>
<tr><td colspan="3">3xxxxxxx</td><td>GPIO Register File</td><td>-</td></tr>
<tr><td rowspan="4">4xxxxxxx</td><td colspan="2">40xxxxxx</td><td rowspan="4">ASB Interrupt Controller</td><td>IRQ_STATUS</td></tr>
<tr><td colspan="2">44xxxxxx</td><td>IRQ_CLEAR</td></tr>
<tr><td colspan="2">48xxxxxx</td><td>IRQ_MASK_ENABLE</td></tr>
<tr><td colspan="2">4Cxxxxxx</td><td>IRQ_MASK_CLEAR</td></tr>
<tr><td rowspan="13">5xxxxxxx</td><td colspan="2">50xxxxxx</td><td rowspan="13">Reconfigurable Module</td><td>Καταχωρητής επόμενης διεύθυνσης μνήμης διαμόρφωσης</td></tr>
<tr><td colspan="2">51xxxxxx</td><td>Καταχωρητής επόμενης διεύθυνσης μνήμης δεδομένων</td></tr>
<tr><td rowspan="4">52xxxxxx</td><td>520xxxxx</td><td rowspan="4">Μνήμη δεδομένων</td><td>set1</td></tr>
<tr><td>524xxxxx</td><td>set2</td></tr>
<tr><td>528xxxxx</td><td>set3</td></tr>
<tr><td>52Cxxxxx</td><td>set4</td></tr>
<tr><td colspan="2">53xxxxxx</td><td>Επαναδιατάξιμος πίνακας</td></tr>
<tr><td colspan="2">54xxxxxx</td><td>Διεπαφή διακοπών</td></tr>
<tr><td colspan="2">55xxxxxx</td><td>Καταχωρητής επαναλήψεων μνήμης διαμόρφωσης</td></tr>
<tr><td colspan="2">56xxxxxx</td><td>Καταχωρητής συγχρονισμού μνημών</td></tr>
<tr><td colspan="2">57xxxxxx</td><td>Καταχωρητής επαναλήψεων μνήμης δεδομένων</td></tr>
</table>

Πίνακας 5.8: Χάρτης απεικόνισης για ανάγνωση από τον επαναδιατάξιμο πίνακα

BA (in hex format)	Σημείο ανάγνωσης από τον επαναδιατάξιμο πίνακα
5300 xxxx	Output from RC1
5301 xxxx	Register1 from RC1
5302 xxxx	Register2 from RC1
5303 xxxx	Register3 from RC1
5304 xxxx	Register4 from RC1
5310 xxxx	Output from RC2
5311 xxxx	Register1 from RC2
5312 xxxx	Register2 from RC2
5313 xxxx	Register3 from RC2
5314 xxxx	Register4 from RC2
(αντιστοιχία όπως πριν)	(αντιστοιχία όπως πριν)
5320 xxxx – 5324 xxxx	Output and Register File from RC3
5330 xxxx – 5334 xxxx	Output and Register File from RC4

5340 xxxx – 5344 xxxx	Output and Register File from RC5
5350 xxxx – 5354 xxxx	Output and Register File from RC6
5360 xxxx – 5364 xxxx	Output and Register File from RC7
5370 xxxx – 5374 xxxx	Output and Register File from RC8
5380 xxxx – 5384 xxxx	Output and Register File from RC9
5390 xxxx – 5394 xxxx	Output and Register File from RC10
53A0 xxxx – 53A4 xxxx	Output and Register File from RC11
53B0 xxxx – 53B4 xxxx	Output and Register File from RC12
53C0 xxxx – 53C4 xxxx	Output and Register File from RC13
53D0 xxxx – 53D4 xxxx	Output and Register File from RC14
53E0 xxxx – 53E4 xxxx	Output and Register File from RC15

ΚΕΦΑΛΑΙΟ 6
Απεικόνιση Εφαρμογών στο rMicroSoc

6.1.Γενικά

Στα πλαίσια της παρούσας διπλωματικής εργασίας, για να επιτευχθεί ο έλεγχος της ορθότητας και της αποδοτικότητας της σχεδίασης του rMicroSoc, υλοποιήθηκαν ορισμένες εφαρμογές. Συγκεκριμένα, σχεδιάστηκε και πραγματοποιήθηκε η απεικόνιση ενός φίλτρου FIR 4-tap, ενός φίλτρου FIR 16-tap και ενός τμήματος του μετασχηματισμού DCT. Για τη σύγκριση των επιδόσεων του rMicroSoc σε σχέση με το MicroSoc χωρίς το επαναδιατάξιμο περιφερειακό, υλοποιήθηκαν επιπλέον οι παραπάνω εφαρμογές αποκλειστικά στο αρχικό σύστημα χωρίς το επιπλέον επαναδιατάξιμο περιφερειακό. Τα αποτελέσματα που προέκυψαν από τις δύο υλοποιήσεις των τριών εφαρμογών οδηγούν σε συμπεράσματα, σχετικά με την επιτάχυνση και την αύξηση της απόδοσης που προκαλεί η χρήση επαναδιατάξιμου υλικού. Στις επόμενες ενότητες, θα παρουσιαστούν ο τρόπος σχεδίασης και υλοποίησης της απεικόνισης κάθε εφαρμογής, καθώς και τα αποτελέσματα που προέκυψαν από τις συγκριτικές μελέτες που προαναφέρθηκαν.

6.2.FIR Φίλτρα

6.2.1.Εισαγωγή

Γενικά, τα φίλτρα χρησιμοποιούνται για την τροποποίηση σημάτων. Λειτουργούν λαμβάνοντας ένα σήμα εισόδου, αποκόπτοντας ορισμένες προκαθορισμένες συχνότητες και περνώντας στην έξοδο του συστήματος το αρχικό σήμα μείον τις συχνότητες εκείνες που αφαιρέθηκαν.

Τα ψηφιακά φίλτρα λαμβάνουν στην είσοδο ένα ψηφιακό σήμα, δίνουν στην έξοδο επίσης ψηφιακό σήμα και αποτελούνται από ψηφιακά δομικά στοιχεία. Σε τυπικές εφαρμογές ψηφιακής επεξεργασίας, ειδικό λογισμικό που εκτελείται σε έναν επεξεργαστή ψηφιακών σημάτων (DSP) διαβάζει δείγματα εισόδου από έναν A/D μετατροπέα, εκτελεί τις μαθηματικές τροποποιήσεις που υποδεικνύονται από τη θεωρία του υλοποιούμενου τύπου του φίλτρου και βγάζει το αποτέλεσμα μέσω ενός D/A μετατροπέα.

Ένα αναλογικό φίλτρο, αντίθετα, λειτουργεί απευθείας στις αναλογικές εισόδους

και δομείται αποκλειστικά με αναλογικά δομικά στοιχεία, όπως αντιστάσεις, πυκνωτές κλπ.

Υπάρχουν πολλά είδη φίλτρων, όσον αφορά την απόκριση συχνότητας, όπως τα βαθυπερατά, υψιπερατά κλπ, τα οποία χρησιμοποιούνται σε διαφορετικές περιστάσεις και διαφορετικές εφαρμογές.

Τα Finite Impulse Response (FIR) φίλτρα είναι μια δομή φίλτρων που μπορούν να χρησιμοποιηθούν για την υλοποίηση σχεδόν οποιουδήποτε είδους απόκρισης συχνότητας ψηφιακά. Συνήθως υλοποιείται στο υλικό χρησιμοποιώντας μια σειρά από καθυστερήσεις, πολλαπλασιαστές και αθροιστές για να δημιουργηθεί η έξοδος του φίλτρου.

Τα FIR χαρακτηρίζονται ως πεπερασμένης απόκρισης (‘finite’) διότι δεν υπάρχει ανάδραση μέσα στο φίλτρο. Για την πληρότητα της παρουσίασης αναφέρουμε ότι τα φίλτρα IIR, ο άλλος βασικός τύπος φίλτρων, δεν είναι πεπερασμένα και ενέχουν εσωτερική ανάδραση, που σημαίνει ότι θεωρητικά δύναται να συνεχίσουν να αποκρίνονται επ` άπειρο.

Τα περισσότερα FIR φίλτρα είναι φίλτρα γραμμικής φάσης (χωρίς να είναι απαραίτητο να ισχύει για όλα τα φίλτρα). Αυτό το χαρακτηριστικό των φίλτρων FIR σημαίνει ότι η απόκριση φάσης είναι μια γραμμική συνάρτηση της συχνότητας. Αυτό έχει ως αποτέλεσμα η καθυστέρηση που εισάγει το φίλτρο να είναι η ίδια σε όλες τις συχνότητες. Συνεπώς, το φίλτρο δεν προκαλεί διαστρέβλωση φάσης και αυτό το χαρακτηριστικό είναι ένα κρίσιμο πλεονέκτημα των FIR φίλτρων έναντι των IIR και των αναλογικών φίλτρων σε ορισμένα συστήματα, όπως για παράδειγμα στα συστήματα Διαμορφωτή-Αποδιαμορφωτή (MoDem). Η συνθήκη για να είναι ένα FIR φίλτρο γραμμικής φάσης είναι οι συντελεστές του να είναι συμμετρικοί γύρω από τον κεντρικό συντελεστή.

Ασφαλώς υπάρχουν και FIR φίλτρα τα οποία δεν είναι γραμμικής φάσης, με το πιο δημοφιλή να είναι τα λεγόμενα ‘ελάχιστης φάσης’ ή καλύτερα ‘ελάχιστης καθυστέρησης’. Αυτά τα φίλτρα έχουν λιγότερη καθυστέρηση από τα φίλτρα γραμμικής φάσης διατηρώντας την ίδια απόκριση πλάτους, με το κόστος των χαρακτηριστικών μη-γραμμικής φάσης (όπως η παραμόρφωση φάσης). Συνήθως τα ελάχιστης-φάσης φίλτρα έχουν το συντελεστή μεγαλύτερου πλάτους πιο κοντά στην αρχή, ενώ στα γραμμικής-φάσης φίλτρα ο συντελεστής μεγαλύτερου πλάτους είναι ο κεντρικός.

6.2.1.1. Σχεδίαση

Δομικώς, τα FIR φίλτρα αποτελούνται από δύο στοιχεία, μια γραμμή καθυστέρησης και μια ομάδα από συντελεστές. Για την υλοποίηση λοιπόν του φίλτρου:

- Τροφοδοτούνται τα δείγματα εισόδου στη γραμμή καθυστέρησης

- Πολλαπλασιάζεται κάθε δείγμα στη γραμμή καθυστέρησης με τον αντίστοιχο συντελεστή και συσσωρεύουμε το αποτέλεσμα
- Ολισθαίνεται η γραμμή καθυστέρησης κατά ένα δείγμα για να κάνουμε χώρο για το επόμενο δείγμα εισόδου

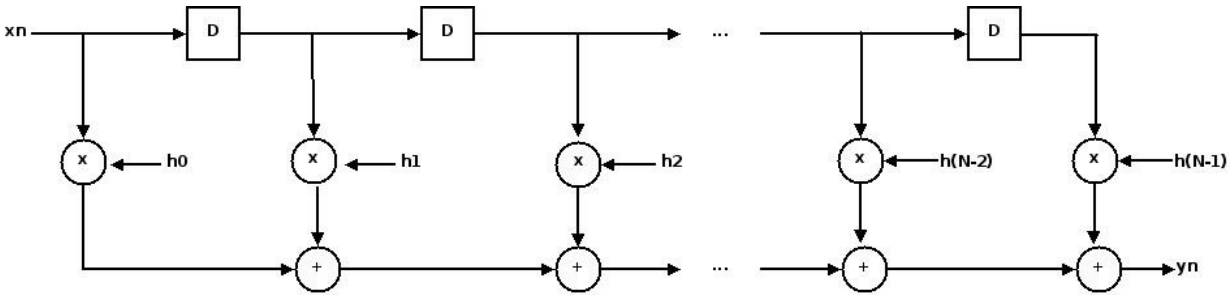
Τα παραπάνω παριστάνονται συμβολικά με τον ακόλουθο μαθηματικό τύπο που περιγράφει τη λειτουργία του φίλτρου :

$$y_k = \sum_{j=0}^{N-1} x_{k-j} h_j \text{ , ή αλλιώς}$$

$$y_k = x_k h_0 + x_{k-1} h_1 + x_{k-2} h_2 + \ldots + x_{k-N-1} h_{N-1}$$

όπου h_k οι συντελεστές βαρύτητας, x_k τα δείγματα εισόδου και N το μέγεθος του φίλτρου.

Το Σχήμα 6.1 δείχνει το βασικό μπλοκ διάγραμμα για ένα φίλτρο FIR μήκους N. Οι καθυστερήσεις που εισάγονται έχουν ως αποτέλεσμα την επεξεργασία περασμένων δειγμάτων εισόδου. Οι τιμές h_k είναι οι συντελεστές που χρησιμοποιούνται για τους πολλαπλασιασμούς, έτσι ώστε η έξοδος τη χρονική στιγμή n να είναι η άθροιση όλων των καθυστερημένων δειγμάτων πολλαπλασιασμένων με τους κατάλληλους συντελεστές.



Σχήμα 6.1: Το μπλοκ διάγραμμα ενός φίλτρου FIR με μήκος N

Η διαδικασία επιλογής του μήκους του φίλτρου και των συντελεστών ονομάζεται σχεδίαση φίλτρου. Ο στόχος είναι να τεθούν αυτοί οι παράμετροι έτσι ώστε κατά τη λειτουργία του φίλτρου να προκύψουν συγκεκριμένες επιθυμητές ιδιότητες.

Συνοπτικά ένα φίλτρο FIR παράγει απλά έναν σταθμισμένο άθροισμα των N πιο πρόσφατων δειγμάτων εισόδου. Οι συντελεστές καθορίζουν την έξοδο για μια δοσμένη ακολουθία δειγμάτων εισόδου. Όσο περισσότεροι συντελεστές υπάρχουν στο φίλτρο, δηλαδή όσο μακρύτερη είναι η γραμμή καθυστέρησής του και τόσο καλύτερη είναι η προσαρμογή της εξόδου στα επιθυμητά χαρακτηριστικά.

6.2.1.2. Υλοποίηση

Έχοντας πλέον προσδιορίσει το μήκος, N, και τους συντελεστές h_k μέσα από τη διαδικασία της σχεδίασης, η υλοποίηση του FIR φίλτρου σε μορφή κώδικα για να εκτελεστεί από έναν DSP είναι σχετικά απλή. Παρακάτω φαίνεται μια απλή υλοποίηση σε γλώσσα προγραμματισμού C.

```
/*
 * Sample the input signal (perhaps via A/D).
 */
sample = input();

/*
 * Insert the newest sample into an N-sample circular
buffer.
 * The oldest sample in the circular buffer is
overwritten.
 */
x[oldest] = sample;

/*
 * Multiply the last N inputs by the appropriate
coefficients.
 * Their sum is the current output.
 */
y = 0;
for (k = 0; k < N; k++)
{
    y += h[k] * x[(oldest + k) % N];
}

oldest = (oldest + 1) % N;

/*
 * Output the result.
 */
output(y);
```

6.2.1.3. Απεικόνιση Φίλτρων FIR σε Υλικό

Στην ενότητα αυτή θα περιγραφούν κάποιες τεχνικές που εφαρμόζονται για την απεικόνιση φίλτρων FIR απευθείας στο MicroSoc. Στη συνέχεια θα παρουσιαστεί ειδικότερα ο τρόπος απεικόνισης που χρησιμοποιήσαμε για να υλοποιήσουμε δύο φίλτρα FIR (ένα 4-tap και ένα 16-tap) στο επαναδιατάξιμο MicroSoc.

Καταρχήν πρέπει να γίνουν κάποιες βασικές υποθέσεις, οι οποίες θα διευκολύνουν την υλοποίηση. Ας θεωρηθεί αρχικά ένα φίλτρο 3 μονάδων καθυστέρησης –από εδώ και πέρα θα χρησιμοποιείται η ειδική ορολογία των

ψηφιακών φίλτρων, σύμφωνα με την οποία ο αριθμός δειγμάτων λήψης ενός φίλτρου εκφράζεται με τον όρο *tap*, οπότε το συγκεκριμένο φίλτρο 3-tap φίλτρο-, πάνω στο οποίο θα παρουσιαστούν οι υποθέσεις που πρέπει να γίνουν. Ωστόσο τα συμπεράσματα γενικεύονται πολύ εύκολα για φίλτρο N-tap.

Για να επιτευχθεί παραλληλία στον υπολογισμό των δειγμάτων εξόδου, γεγονός που εκμεταλλεύεται η διαδικασία υλοποίησης του φίλτρου στο επαναδιατάξιμο σύστημα, γίνεται η υπόθεση ότι η μαθηματική περιγραφή ενός φίλτρου 3-tap είναι ως εξής:

$$y_0 = x_0 h_0 + x_{-1} h_1 + x_{-2} h_2$$
$$y_1 = x_1 h_0 + x_0 h_1 + x_{-1} h_2$$

με $x_{-2} = x_{-1} = 0$

Η ιδέα της παραπάνω μορφής του μαθηματικού τύπου του φίλτρου είναι ότι ο αλγόριθμος υπολογισμού του φίλτρου έχει πρόσβαση σε N-1 τιμές εισόδου από $x_{-(N-1)}$ έως x_{-1} , οι οποίες είναι όλες μηδενικές.

Για τον ίδιο σκοπό με πριν γίνεται επίσης μια άλλη ομάδα υποθέσεων, η οποία έχει να κάνει με τα τελευταία στοιχεία στο διάνυσμα εξόδου. Ο αριθμός των σχετικών μη μηδενικών στοιχείων εξόδου μπορεί να βρεθεί από τον αριθμό των στοιχείων εισόδου και της τάξης του φίλτρου. Συγκεκριμένα η διαφορά μεταξύ των δύο αριθμών των στοιχείων εισόδου και εξόδου είναι ίση με (N-1) και μάλιστα υπάρχουν (N-1) στοιχεία εξόδου περισσότερα από στοιχεία εισόδου. Για το λόγο αυτό, αν για παράδειγμα το τελευταίο δείγμα εισόδου είναι το x_k , τότε το τελευταίο σχετικό μη μηδενικό δείγμα εξόδου είναι το y_{k+N-1} . Έτσι πρέπει να τοποθετηθούν συνολικά (N-1) μηδενικά στο τέλος του διανύσματος εισόδου. Για να γίνει πιο κατανοητό το παραπάνω, οι υποθέσεις που πρέπει να γίνουν για το φίλτρο 3-tap είναι οι ακόλουθες:

$$y_{k+N-2} = x_{k+N-2} h_0 + x_{k+N-3} h_1 + x_{k+N-4} h_2$$
$$y_{k+N-1} = x_{k+N-1} h_0 + x_{k+N-2} h_1 + x_{k+N-3} h_2$$
$$x_{k+N-2} = 0 \text{ , } x_{k+N-1} = 0$$

Πρέπει να σημειωθεί ότι όλες αυτές οι υποθέσεις δεν επηρεάζουν την ορθότητα της συνάρτησης υλοποίησης του φίλτρου FIR. Χρειάζονται απλώς για την ομοιομορφία της υλοποίησης, δηλαδή για να είναι δυνατή η εφαρμογή του ίδιου αλγορίθμου για τον υπολογισμό όλων των στοιχείων του διανύσματος εξόδου.

6.2.2.Φίλτρο FIR 4-tap στο MicroSoc

6.2.2.1. Τεχνική Απεικόνισης

Η πρώτη εφαρμογή που υλοποιήθηκε στο επαναδιατάξιμο Microsoc είναι ένα φίλτρο FIR 4-tap, δηλαδή ένα φίλτρο 4 μονάδων καθυστέρησης. Παρακάτω, θα περιγραφεί αναλυτικά η μέθοδος αυτής της υλοποίησης και κάποια συγκριτικά συμπεράσματα που προέκυψαν.

Έχει σημασία να τονιστεί ότι η τάξη του συγκεκριμένου φίλτρου είναι ίδια με το μέγεθος του επαναδιατάξιμου πίνακα. Έτσι ο συγκεκριμένος τρόπος υλοποίησης που προτείνεται εφαρμόζεται μόνο σε αυτή την περίπτωση[27]. Η απεικόνιση ενός φίλτρου τάξης μεγαλύτερης του μεγέθους του επαναδιατάξιμου πίνακα προτείνεται στην επόμενη παράγραφο, όταν και περιγράφεται η υλοποίηση ενός φίλτρου FIR 16-tap.

Ο μαθηματικός τύπος που περιγράφει το φίλτρο αυτό είναι ο ακόλουθος:

$$y_k = \sum_{j=0}^{3} x_{k-j} h_j \quad \text{, ή αλλιώς}$$

$$y_k = x_k h_0 + x_{k-1} h_1 + x_{k-2} h_2 + x_{k-3} h_3$$

Η προτεινόμενη απεικόνιση υποθέτει ότι ο επαναδιατάξιμος πίνακας χρησιμοποιείται σε κατάσταση λειτουργίας ανά στήλες, στην οποία όλα τα κελιά στην ίδια στήλη λαμβάνουν την ίδια λέξη διαμόρφωσης και εκτελούν την ίδια λειτουργία. Ωστόσο τα δεδομένα εισόδου, δηλαδή τα δείγματα του διανύσματος εισόδου, τροφοδοτούνται ανά γραμμή, δηλαδή όλα τα κελιά σε μια γραμμή λαμβάνουν τα ίδια δεδομένα από τη μνήμη δεδομένων. Αρχικά θα παρουσιαστούν συνοπτικά οι τιμές με τις οποίες τροφοδοτούνται οι είσοδοι των κελιών του πίνακα, καθώς και οι τιμές που λαμβάνονται στην έξοδο κάθε κελιού στο τέλος κάθε κύκλου λειτουργίας. Έπειτα θα εξηγηθεί το πως γίνεται ο προγραμματισμός του επαναδιατάξιμου περιφερειακού για να επιτευχθεί αυτή η διαμόρφωση.

Θα θεωρηθεί ότι το διάνυσμα δειγμάτων εισόδου έχει μέγεθος 10 στοιχεία. Στο διάνυσμα εισόδου, το οποίο πρέπει να τροφοδοτηθεί σαν είσοδος στο φίλτρο θα πρέπει να υπάρχουν ακόμα 3 μηδενικά στοιχεία στην αρχή του.

Τα μη μηδενικά δείγματα εξόδου θα εκτείνονται μέχρι το δείγμα y_{12} , σύμφωνα με τα όσα αναφέρθηκαν στην προηγούμενη παράγραφο. Παρακάτω παρουσιάζονται οι τύποι υπολογισμού όλων των μη μηδενικών στοιχείων που αναμένονται στην έξοδο του φίλτρου:

$$y_0 = x_0 h_0 + x_{-1} h_1 + x_{-2} h_2 + x_{-3} h_3$$
$$y_1 = x_1 h_0 + x_0 h_1 + x_{-1} h_2 + x_{-2} h_3$$
$$y_2 = x_2 h_0 + x_1 h_1 + x_0 h_2 + x_{-1} h_3$$
$$y_3 = x_3 h_0 + x_2 h_1 + x_1 h_2 + x_0 h_3$$

$$y_4 = x_4 h_0 + x_3 h_1 + x_2 h_2 + x_1 h_3$$
$$y_5 = x_5 h_0 + x_4 h_1 + x_3 h_2 + x_2 h_3$$
$$y_6 = x_6 h_0 + x_5 h_1 + x_4 h_2 + x_3 h_3$$
$$y_7 = x_7 h_0 + x_6 h_1 + x_5 h_2 + x_4 h_3$$
$$y_8 = x_8 h_0 + x_7 h_1 + x_6 h_2 + x_5 h_3$$
$$y_9 = x_9 h_0 + x_8 h_1 + x_7 h_2 + x_6 h_3$$
$$y_{10} = x_{10} h_0 + x_9 h_1 + x_8 h_2 + x_7 h_3$$
$$y_{11} = x_{11} h_0 + x_{10} h_1 + x_9 h_2 + x_8 h_3$$
$$y_{12} = x_{12} h_0 + x_{11} h_1 + x_{10} h_2 + x_9 h_3$$

Ο Πίνακας 6.1 παρουσιάζει συνοπτικά όλες τις πληροφορίες για τις εισόδους και εξόδους των κελιών του επαναδιατάξιμου πίνακα (παρουσιάζονται τόσοι κύκλοι όσοι απαιτούνται μέχρι να προκύψει και το δείγμα εξόδου y_7).

Πίνακας 6.1: Λειτουργία του πίνακα ανά κύκλο και ανά στήλη

Cycl.	Data	Column 0	Column 1	Column 2	Column 3
C0	-	RC(0,0)=0	RC(0,1)=0	RC(0,2)=0	RC(0,3)=0
	-	RC(1,0)=0	RC(1,1)=0	RC(1,2)=0	RC(1,3)=0
	-	RC(2,0)=0	RC(2,1)=0	RC(2,2)=0	RC(2,3)=0
	-	RC(3,0)=0	RC(3,1)=0	RC(3,2)=0	RC(3,2)=0
C1	x_{-3}	$x_{-3}h_3$	$x_{-3}h_2$	$x_{-3}h_1$	$x_{-2}h_0$
	x_{-2}	$x_{-2}h_3$	$x_{-2}h_2$	$x_{-2}h_1$	$x_{-2}h_0$
	x_{-1}	$x_{-1}h_3$	$x_{-1}h_2$	$x_{-1}h_1$	$x_{-1}h_0$
	x_0	x_0h_3	x_0h_2	x_0h_1	x_0h_0
C2	x_{-2}	$x_{-2}h_3$	$x_{-2}h_2+x_{-3}h_3$	$x_{-2}h_1+x_{-3}h_2$	$x_{-2}h_0+x_{-3}h_1$
	x_{-1}	$x_{-1}h_3$	$x_{-1}h_2+x_{-2}h_3$	$x_{-1}h_1+x_{-2}h_2$	$x_{-1}h_0+x_{-2}h_1$
	x_0	x_0h_3	$x_0h_2+x_{-1}h_3$	$x_0h_1+x_{-1}h_2$	$x_0h_0+x_{-1}h_1$
	x_1	x_1h_3	$x_1h_2+x_0h_3$	$x_1h_1+x_0h_2$	$x_1h_0+x_0h_1$
C3	x_{-1}	$x_{-1}h_3$	$x_{-1}h_2+x_{-2}h_3$	$x_{-1}h_1+x_{-2}h_2+x_{-3}h_3$	$x_{-1}h_0+x_{-2}h_1+x_{-3}h_2$

	x_0	x_0h_3	$x_0h_2+x_{-1}h_3$	$x_0h_1+x_{-1}h_2+x_{-2}h_3$	$x_0h_0+x_{-1}h_1+x_{-2}h_2$
	x_1	x_1h_3	$x_1h_2+x_0h_3$	$x_1h_1+x_0h_2+x_{-1}h_3$	$x_1h_0+x_0h_1+x_{-1}h_2$
	x_2	x_2h_3	$x_2h_2+x_1h_3$	$x_2h_1+x_1h_2+x_0h_3$	$x_2h_0+x_1h_1+x_0h_2$
C4	x_0	x_0h_3	$x_0h_2+x_{-1}h_3$	$x_0h_1+x_{-1}h_2+x_{-2}h_3$	$x_0h_0+x_{-1}h_1+x_{-2}h_2+x_{-3}h_3$
	x_1	x_1h_3	$x_1h_2+x_0h_3$	$x_1h_1+x_0h_2+x_{-1}h_3$	$x_1h_0+x_0h_1+x_{-1}h_2+x_{-2}h_3$
	x_2	x_2h_3	$x_2h_2+x_1h_3$	$x_2h_1+x_1h_2+x_0h_3$	$x_2h_0+x_1h_1+x_0h_2+x_{-1}h_3$
	x_3	x_3h_3	$x_3h_2+x_2h_3$	$x_3h_1+x_2h_2+x_1h_3$	$x_3h_0+x_2h_1+x_1h_2+x_0h_3$
C5	x_1	x_1h_3	$x_1h_2+x_0h_3$	$x_1h_1+x_0h_2+x_{-1}h_3$	$x_1h_0+x_0h_1+x_{-1}h_2+x_{-2}h_3$
	x_2	x_2h_3	$x_2h_2+x_1h_3$	$x_2h_1+x_1h_2+x_0h_3$	$x_2h_0+x_1h_1+x_0h_2+x_{-1}h_3$
	x_3	x_3h_3	$x_3h_2+x_2h_3$	$x_3h_1+x_2h_2+x_1h_3$	$x_3h_0+x_2h_1+x_1h_2+x_0h_3$
	x_4	x_4h_3	$x_4h_2+x_3h_3$	$x_4h_1+x_3h_2+x_2h_3$	$x_4h_0+x_3h_1+x_2h_2+x_1h_3$
C6	x_2	x_2h_3	$x_2h_2+x_1h_3$	$x_2h_1+x_1h_2+x_0h_3$	$x_2h_0+x_1h_1+x_0h_2+x_{-1}h_3$
	x_3	x_3h_3	$x_3h_2+x_2h_3$	$x_3h_1+x_2h_2+x_1h_3$	$x_3h_0+x_2h_1+x_1h_2+x_0h_3$
	x_4	x_4h_3	$x_4h_2+x_3h_3$	$x_4h_1+x_3h_2+x_2h_3$	$x_4h_0+x_3h_1+x_2h_2+x_1h_3$
	x_5	x_5h_3	$x_5h_2+x_4h_3$	$x_5h_1+x_4h_2+x_3h_3$	$x_5h_0+x_4h_1+x_3h_2+x_2h_3$
C7	x_3	x_3h_3	$x_3h_2+x_2h_3$	$x_3h_1+x_2h_2+x_1h_3$	$x_3h_0+x_2h_1+x_1h_2+x_0h_3$
	x_4	x_4h_3	$x_4h_2+x_3h_3$	$x_4h_1+x_3h_2+x_2h_3$	$x_4h_0+x_3h_1+x_2h_2+x_1h_3$
	x_5	x_5h_3	$x_5h_2+x_4h_3$	$x_5h_1+x_4h_2+x_3h_3$	$x_5h_0+x_4h_1+x_3h_2+x_2h_3$
	x_6	x_6h_3	$x_6h_2+x_5h_3$	$x_6h_1+x_5h_2+x_4h_3$	$x_6h_0+x_5h_1+x_4h_2+x_3h_3$
C8	x_4	x_4h_3	$x_4h_2+x_3h_3$	$x_4h_1+x_3h_2+x_2h_3$	$x_4h_0+x_3h_1+x_2h_2+x_1h_3$
	x_5	x_5h_3	$x_5h_2+x_4h_3$	$x_5h_1+x_4h_2+x_3h_3$	$x_5h_0+x_4h_1+x_3h_2+x_2h_3$
	x_6	x_6h_3	$x_6h_2+x_5h_3$	$x_6h_1+x_5h_2+x_4h_3$	$x_6h_0+x_5h_1+x_4h_2+x_3h_3$
	x_7	x_7h_3	$x_7h_2+x_6h_3$	$x_7h_1+x_6h_2+x_5h_3$	$x_7h_0+x_6h_1+x_5h_2+x_4h_3$

Στον πίνακα αυτό κάθε στήλη ‘Column X’ αντιστοιχεί σε μια στήλη του επαναδιατάξιμου πίνακα και επιπλέον κάθε 4 γραμμές αντιστοιχούν σε έναν κύκλο, καθώς ο επαναδιατάξιμος πίνακας έχει μέγεθος 4x4. Επίσης κάθε γραμμή μιας τετράδας αντιστοιχεί σε μια γραμμή του επαναδιατάξιμου πίνακα. Για παράδειγμα στον κύκλο 1, η πρώτη γραμμή αντιστοιχεί στη γραμμή 1 του επαναδιατάξιμου πίνακα, η δεύτερη γραμμή στη γραμμή 2 κ.ο.κ. Συνεπώς σε κάθε κύκλο λειτουργίας οι 4 γραμμές και οι 4 στήλες σχηματίζουν εικονικά τον επαναδιατάξιμο πίνακα και κάθε κελί αντιστοιχεί σε ένα κελί του.

Από τους κύκλους που φαίνονται ο πρώτος είναι ένας φανταστικός κύκλος, ο οποίος εμφανίζεται εκεί ώστε να παρουσιαστεί η αρχική κατάσταση κάθε κελιού. Τέλος η στήλη δεδομένων δείχνει τα δεδομένα εισόδου που έρχονται από τη μνήμη και τροφοδοτούνται σε ολόκληρη τη σειρά του πίνακα.

Παρατηρώντας τον πίνακα αυτό, φαίνεται κατ`αρχήν ότι τα αποτελέσματα στην έξοδο των 4 κελιών της στήλης *Column 3* μετά την ολοκλήρωση του κύκλου C4 αντιστοιχούν στα δείγματα εξόδου $y_0 - y_3$. Το ίδιο συμβαίνει, αντίστοιχα, με τα κελιά της ίδιας στήλης μετά την ολοκλήρωση και του κύκλου C8. Συνεπώς τα δείγματα εξόδου του φίλτρου λαμβάνονται στις εξόδους των 4 κελιών της τελευταίας στήλης του πίνακα, ανά τετράδες και ανά τέσσερις κύκλους.

Η λογική λειτουργίας των κελιών του επαναδιατάξιμου πίνακα είναι ότι πολλαπλασιάζουν την είσοδο δεδομένων από τη μνήμη δεδομένων με έναν συντελεστή του φίλτρου -σταθερό για κάθε κελί- και προσθέτουν το γινόμενο στο αποτέλεσμα εξόδου του προηγούμενου κύκλου του κελιού της ίδιας γραμμής αλλά προηγούμενης στήλης. Παρατηρείται επίσης ότι τα κελιά κάθε στήλης πολλαπλασιάζουν το δείγμα εισόδου με τον ίδιο συντελεστή. Ο συγκεκριμένος τρόπος λειτουργίας προκύπτει μελετώντας τη μαθηματική περιγραφή του φίλτρου. Με τον τρόπο αυτό σε κάθε γραμμή του επαναδιατάξιμου πίνακα γίνεται προσομοίωση της γραμμής καθυστέρησης του φίλτρου, από την οποία όμως εκτός των τελικών δειγμάτων εξόδου προκύπτουν και κάποιες τιμές ‘dummy’, χωρίς χρησιμότητα δηλαδή.

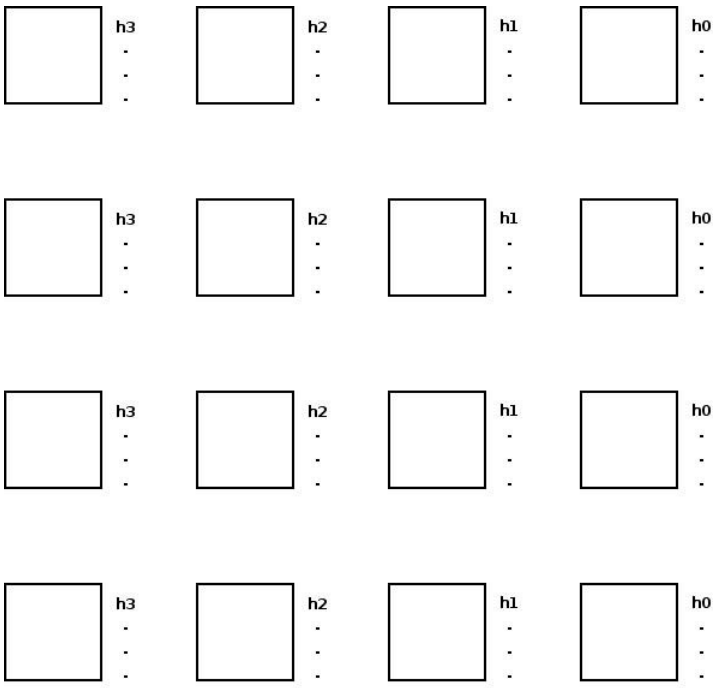
6.2.2.2. Περίοδος Λειτουργίας

Ακολουθεί στη συνέχεια μια αναλυτική περιγραφή του τρόπου με τον οποίο γίνεται η διαμόρφωση του πίνακα ώστε να λειτουργεί σύμφωνα με τον παραπάνω τρόπο.

Όπως έχει ήδη αναφερθεί, πρέπει τα κελιά μιας συγκεκριμένης στήλης του πίνακα να πολλαπλασιάζουν το δείγμα εισόδου με το οποίο τροφοδοτούνται με τον ίδιο συντελεστή. Για να επιτευχθεί αυτό, πρέπει πριν την έναρξη της κανονικής λειτουργίας του πίνακα να υπάρξουν κάποιοι κύκλοι αρχικοποίησης, κατά τους οποίους φορτώνονται ανά στήλη στα κελιά δεδομένα από τη μνήμη δεδομένων (τα οποία δεν είναι άλλα από τους συντελεστές βαρύτητας του

φίλτρου) και αποθηκεύονται σε κάποιον καταχωρητή από το τοπικό αρχείο καταχωρητών των κελιών (ας υποθέσουμε στον καταχωρητή RF0). Κατά τη διάρκεια εκτέλεσης, ωστόσο, των υπολογισμών η μετάδοση των λέξεων γίνεται ανά γραμμή. Οι συντελεστές τοποθετούνται στον πίνακα ανάλογα με το δείκτη τους, δηλαδή ο συντελεστής h_3 αποθηκεύεται στην πρώτη στήλη από τα αριστερά (τη στήλη #3 δηλαδή), ο h_2 στη δεύτερη από τα αριστερά (τη στήλη #2 δηλαδή) κ.ο.κ.

Το Σχήμα 6.2 παρουσιάζει την τελική εικόνα του επαναδιατάξιμου πίνακα, μετά τη φάση της φόρτωσης των συντελεστών βαρύτητας του φίλτρου στα κελιά.



Σχήμα 6.2: Η εικόνα των εσωτερικών καταχωρητών των κελιών του επαναδιατάξιμου πίνακα, μετά τη φάση φόρτωσης των συντελεστών βαρύτητας

Στο σχήμα αυτό από τα δεξιά κάθε κελιού απεικονίζεται το περιεχόμενο του τοπικού αρχείου καταχωρητών, όπου στον καταχωρητή RF0 φαίνεται αποθηκευμένος ο συντελεστής βαρύτητας που αντιστοιχεί σε κάθε κελί. Να σημειωθεί ότι αυτή η φάση φόρτωσης των συντελεστών στα κελιά διαρκεί συνολικά δύο κύκλους διαμόρφωσης, εκ των οποίων στον πρώτο γίνεται η φόρτωση των δεδομένων από τη μνήμη δεδομένων και στο δεύτερο η αποθήκευση των δεδομένων αυτών στους τοπικούς καταχωρητές.

Στη συνέχεια αρχίζει η κανονική λειτουργία των κελιών, η οποία είναι περιοδική με περίοδο που διαρκεί τέσσερις κύκλους λειτουργίας. Σε κάθε περίοδο το κελί εκτελεί τα εξής, κατά τη διάρκεια των τεσσάρων κύκλων:

- Φορτώνει το επόμενο δεδομένο από τη μνήμη δεδομένων και το πολλαπλασιάζει με το περιεχόμενο του καταχωρητή RF0 του τοπικού

αρχείου καταχωρητών του.

- Αποθηκεύει το αποτέλεσμα του πολλαπλασιασμού του προηγούμενου κύκλου στον καταχωρητή RF1 του αρχείου καταχωρητών ενώ παράλληλα μεταφέρει το περιεχόμενο του καταχωρητή RF2 στην έξοδο. Στον τελευταίο καταχωρητή γίνεται η υπόθεση ότι αποθηκεύεται το τελικό αποτέλεσμα εξόδου του προηγούμενου κύκλου του συγκεκριμένου κελιού. Την πρώτη φορά πρέπει, συνεπώς, να τεθεί το περιεχόμενό του στο 0.

- Προσθέτει το περιεχόμενο του καταχωρητή RF1 (το αποτέλεσμα του γινομένου του πρώτου κύκλου) με την έξοδο του κελιού στην ίδια γραμμή με το τρέχον κελί αλλά στην προηγούμενη στήλη. Για την ομοιομορφία της υλοποίησης γίνεται η υπόθεση ότι αριστερά από τα κελιά που βρίσκονται στην πρώτη από τα αριστερά στήλη υπάρχει άλλη μια –φανταστική- στήλη κελιών, των οποίων η έξοδος δεδομένων είναι συνεχώς ίση με 0.

- Το αποτέλεσμα της πρόσθεσης που εκτελέστηκε στον προηγούμενο κύκλο αποθηκεύεται στον καταχωρητή RF2, του τοπικού αρχείου καταχωρητών.

6.2.2.3. Ψευδοκώδικας – Περιεχόμενο Διαμόρφωσης

Συμβολικά, οι λειτουργίες κάθε κύκλου μιας περιόδου περιγράφονται με τον παρακάτω ψευδοκώδικα:

Κύκλος1 : $RF_0 \cdot MEM[\,] = ALU_{out1}$
Κύκλος2 : $RF_1 \Leftarrow ALU_{out1}$ / $OUT \Leftarrow RF_2$
Κύκλος3 : $RF_1 + RC_{LEFTout} = ALU_{out2}$
Κύκλος4 : $RF_2 \Leftarrow ALU_{out2}$

Ακολούθως θα παρουσιαστούν οι λέξεις διαμόρφωσης που πρέπει να σταλούν στον επαναδιατάξιμο πίνακα για να επιτευχθεί η παραπάνω επιθυμητή λειτουργία. Να τονιστεί ότι οι λέξεις διαμόρφωσης είναι ακριβώς ίδιες για όλα τα κελιά και ότι μεταδίδονται ανά γραμμές, εκτός από τις λέξεις που αφορούν τη φόρτωση των συντελεστών βαρύτητας του φίλτρου από τη μνήμη, οι οποίες μεταδίδονται ανά στήλες. Για την περαιτέρω επεξήγηση των λέξεων διαμόρφωσης που παρουσιάζονται παρακάτω, ο αναγνώστης παραπέμπεται στις ενότητες 5.2.4. και 5.6. Τα σημάδια '-' ανάμεσα στα bits διαχωρίζουν τα πεδία της λέξης διαμόρφωσης.

- *0-00-0-00000-0110-0000-011-0-0-00000000-0* : Η λέξη αυτή ορίζει στα κελιά ότι πρέπει να γίνει φόρτωση από τη μνήμη δεδομένων. Όπως παρατηρείται η λέξη αφορά κανονική κατάσταση λειτουργίας. Ορίζει ότι θα λειτουργήσει η

ALU και καθορίζει τη λειτουργία που θα εκτελέσει, η οποία είναι η λειτουργία κατά την οποία τα δεδομένα εισόδου στην είσοδο a της μονάδας μεταφέρονται αυτούσια -χωρίς τροποποιήσεις- στην έξοδο της μονάδας. Ακόμη η λέξη ορίζει ότι στην είσοδο δεδομένων a της μονάδας ALU-MUL επιλέγονται από τον MUX εισόδου δεδομένα από τη μνήμη δεδομένων. Συνεπώς τα δεδομένα από τη μνήμη δεδομένων εγγράφονται στον καταχωρητή εξόδου.

- *1-00-0-00000-0000-0000-000-0-0-00000000-0* : Με τη λέξη αυτή καθορίζεται η εγγραφή των αποτελεσμάτων της μονάδας ALU-MUL του προηγούμενου κύκλου ρολογιού στο αρχείο καταχωρητών. Αυτά αποτελούν τα δεδομένα όπως έρχονται απευθείας από τη μνήμη δεδομένων. Μάλιστα ορίζεται και σε ποιο καταχωρητή γίνεται η εγγραφή και συγκεκριμένα για το σκοπό αυτό ορίζεται ο καταχωρητής RF_0 .

- *0-00-0-00000-0110-0111-000-1-0-00000000-0* : Με αυτή τη λέξη διαμόρφωσης ορίζεται να λειτουργήσει ο πολλαπλασιαστής από τη μονάδα ALU-MUL, ο οποίος δέχεται σαν εισόδους το περιεχόμενο του καταχωρητή RF_0 και κάποιο δεδομένο από τη μνήμη δεδομένων, το οποίο αποτελεί κάποιο δείγμα εισόδου.

- *1-01-000000-1-00000000-0-00000000-0-0-0-1* : Με τη λέξη αυτή γίνονται τρεις ενέργειες ταυτόχρονα. α)Τίθονται όλα τα κελιά σε κατάσταση stall, β)η μνήμη δεδομένων τίθεται επίσης σε κατάσταση stall και γ)γίνεται η εγγραφή του γινομένου του πολλαπλασιασμού του προηγούμενου κύκλου διαμόρφωσης στον καταχωρητή RF_1 . Το τελευταίο γίνεται με ορισμό εγγραφής στο αρχείο καταχωρητών και επιλογή του συγκεκριμένου καταχωρητή RF_1 .

- *0-00-0-00000-1001-0000-011-0-0-00000000-0* : Η συγκεκριμένη λέξη διαμόρφωσης μεταφέρει το περιεχόμενο του καταχωρητή RF_2 στον καταχωρητή εξόδου και στην έξοδο του κελιού. Αυτό επιτυγχάνεται με το να τίθεται η λειτουργία της ALU στη λειτουργία περάσματος δεδομένων από την είσοδο στην έξοδο και με το να τροφοδοτείται η είσοδος δεδομένων a της μονάδας ALU-MUL με τα περιεχόμενα του καταχωρητή RF_0 . Το τελευταίο γίνεται με επιλογή της κατάλληλης εισόδου δεδομένων από τον MUX εισόδου. Σκοπός του κύκλου αυτού ρολογιού είναι να γίνουν διαθέσιμα τα αποτελέσματα του προηγούμενου κύκλου λειτουργίας στον επόμενο κύκλο διαμόρφωσης, ώστε να χρησιμοποιηθούν από τα κελιά στα δεξιά σαν όρισμα στην πράξη πρόσθεσης που πρέπει να γίνει.

- *0-00-0-00000-0010-1000-010-0-0-00000000-0*: Αυτή η λέξη διαμόρφωσης επιτελεί την πρόσθεση μεταξύ του γινομένου του συντελεστή βαρύτητας του κελιού με το νέο δείγμα εισόδου και του τελικού αποτελέσματος του προηγούμενου κύκλου του κελιού αριστερά από το τρέχον. Για να επιτευχθεί αυτό ρυθμίζεται να λειτουργήσει η ALU από τη μονάδα

ALU/MUL και τροφοδοτείται με εισόδους τον καταχωρητή RF_1 (όπου είχε αποθηκευτεί νωρίτερα το αποτέλεσμα του πολλαπλασιασμού) και την έξοδο του κελιού αριστερά του τρέχοντος (αφού με την προηγούμενη λέξη διαμόρφωσης όλα τα κελιά έχουν μεταφέρει στην έξοδό τους το περιεχόμενο του καταχωρητή RF_2, ο οποίος περιέχει τα αποτελέσματα των υπολογισμών του προηγούμενου κύκλου λειτουργίας).

- *1-10-0-00000-0000-0000-000-0-0-00000000-0* : Τέλος με τη λέξη αυτή γίνεται η εγγραφή του αθροίσματος που προέκυψε από την πρόσθεση στον προηγούμενο κύκλο διαμόρφωσης στον καταχωρητή RF_2.

Από τις παραπάνω 7 λέξεις διαμόρφωσης, αν εξαιρεθούν οι δύο πρώτες που αφορούν στη φόρτωση στα κελιά των σταθερών συντελεστών βαρύτητας του φίλτρου, οι υπόλοιπες 5 αποτελούν μια περίοδο λειτουργίας και πρέπει να επαναληφθούν όσες φορές είναι αναγκαίο. Συνεπώς πρέπει να εκτελεστούν συνολικά 8 φορές, για να προκύψουν τα δείγματα εξόδου μέχρι το y7. Για το σκοπό αυτό χρησιμοποιείται η ειδική λειτουργία των δύο μνημών, διαμόρφωσης και δεδομένων, των εσωτερικών βρόχων επαναλήψεων. Έτσι τίθεται από την αρχή το περιεχόμενο του καταχωρητή επαναλήψεων στην τιμή 8 και μετά το τέλος των λέξεων διαμόρφωσης που αποτελούν μια περίοδο λειτουργίας τοποθετείται στη μνήμη διαμόρφωσης μια ειδική θέση μνήμης. Αυτή περιέχει λέξεις διαμόρφωσης που είναι της μορφής

0-00-000000-0-00000000-0-00000100-1-0-0-1

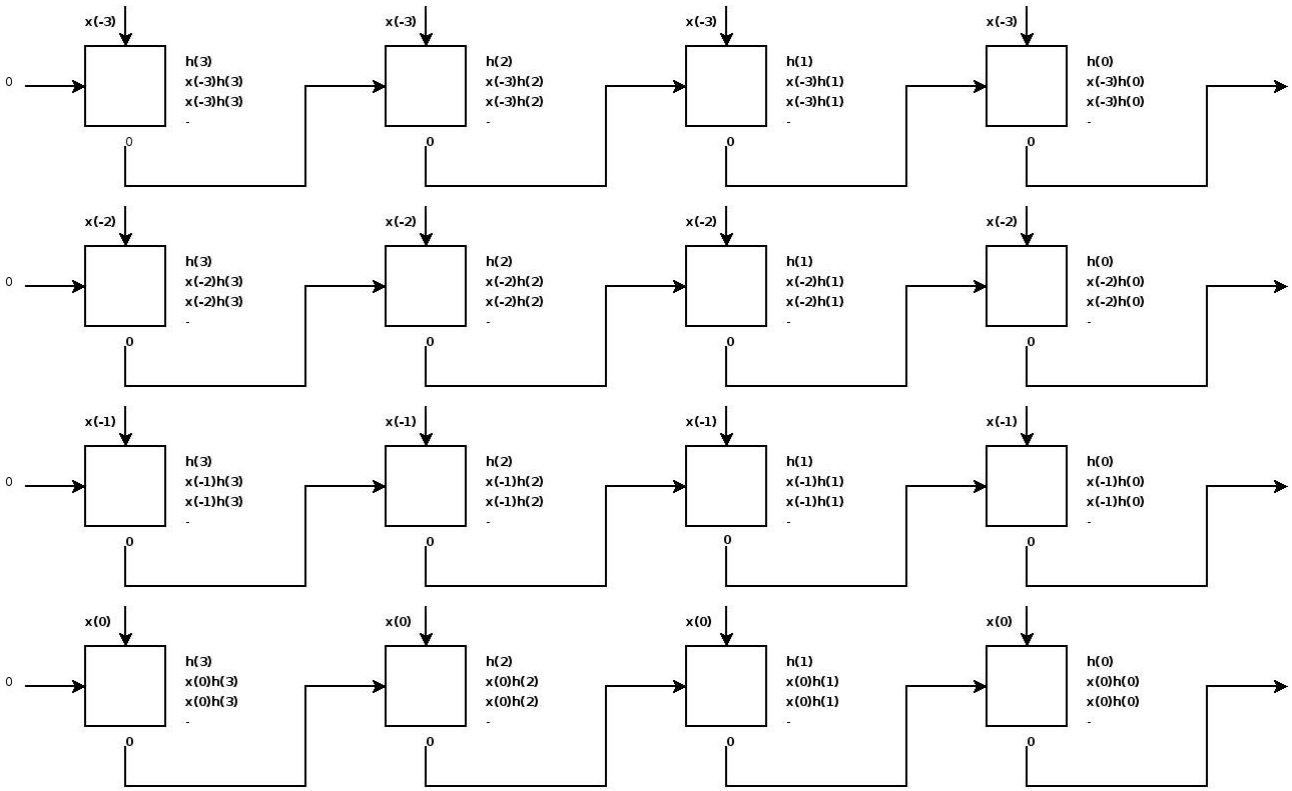
η οποία καθορίζει ότι πρέπει να γίνει άλμα επανάληψης, εκτός κι αν ο καταχωρητής επαναλήψεων είναι μηδενισμένος. Επιπλέον καθορίζει τη διεύθυνση του άλματος, η οποία είναι η διεύθυνση στη μνήμη διαμόρφωσης με αριθμό 4. Σε αυτή τη διεύθυνση έχει τοποθετηθεί η πρώτη από τις λέξεις που αποτελούν μια περίοδο λειτουργίας.

Με την επεξήγηση και των λέξεων διαμόρφωσης που χρησιμοποιήθηκαν για τη λειτουργία των κελιών του επαναδιατάξιμου πίνακα σύμφωνα με το επιθυμητό, ολοκληρώθηκε και η παρουσίαση του τρόπου απεικόνισης του φίλτρου FIR 4-tap στο επαναδιατάξιμο MicroSoc.

6.2.2.4. Παρουσίαση Λειτουργίας του Πίνακα ανά Περίοδο

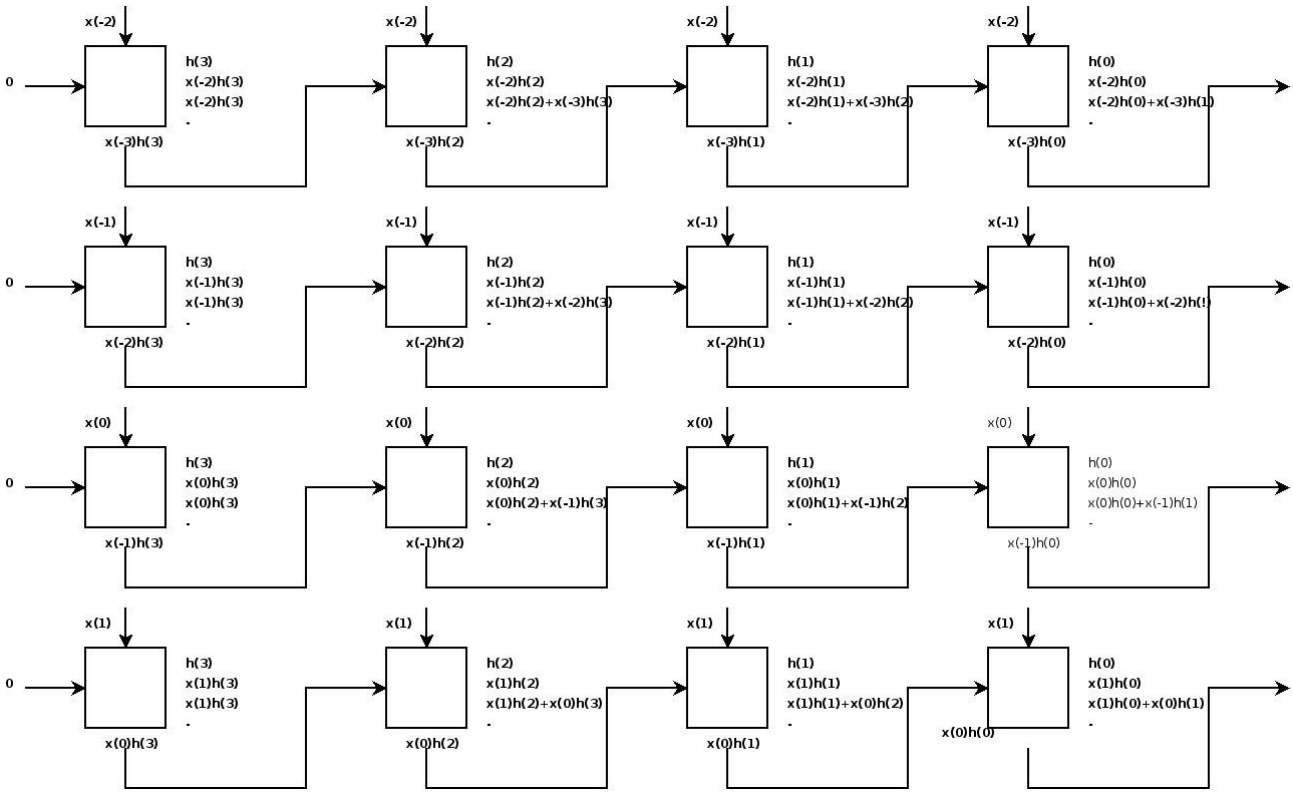
Παρακάτω, παρουσιάζεται σχηματικά (Σχήμα 6.3-Σχήμα 6.10) η λειτουργία του επαναδιατάξιμου πίνακα, κατά την εκτέλεση των υπολογισμών για το 4-tap FIR φίλτρο. Κάθε σχήμα αντιστοιχεί σε μια διαφορετική περίοδο λειτουργίας, μέχρι την παραγωγή και του δείγματος εξόδου y_7. Αριστερά από κάθε κελί απεικονίζονται τα περιεχόμενα του αρχείου καταχωρητών. Κάτω από κάθε κελί απεικονίζονται τα δεδομένα εξόδου του εκάστοτε κελιού.

Περίοδος Λειτουργίας 1



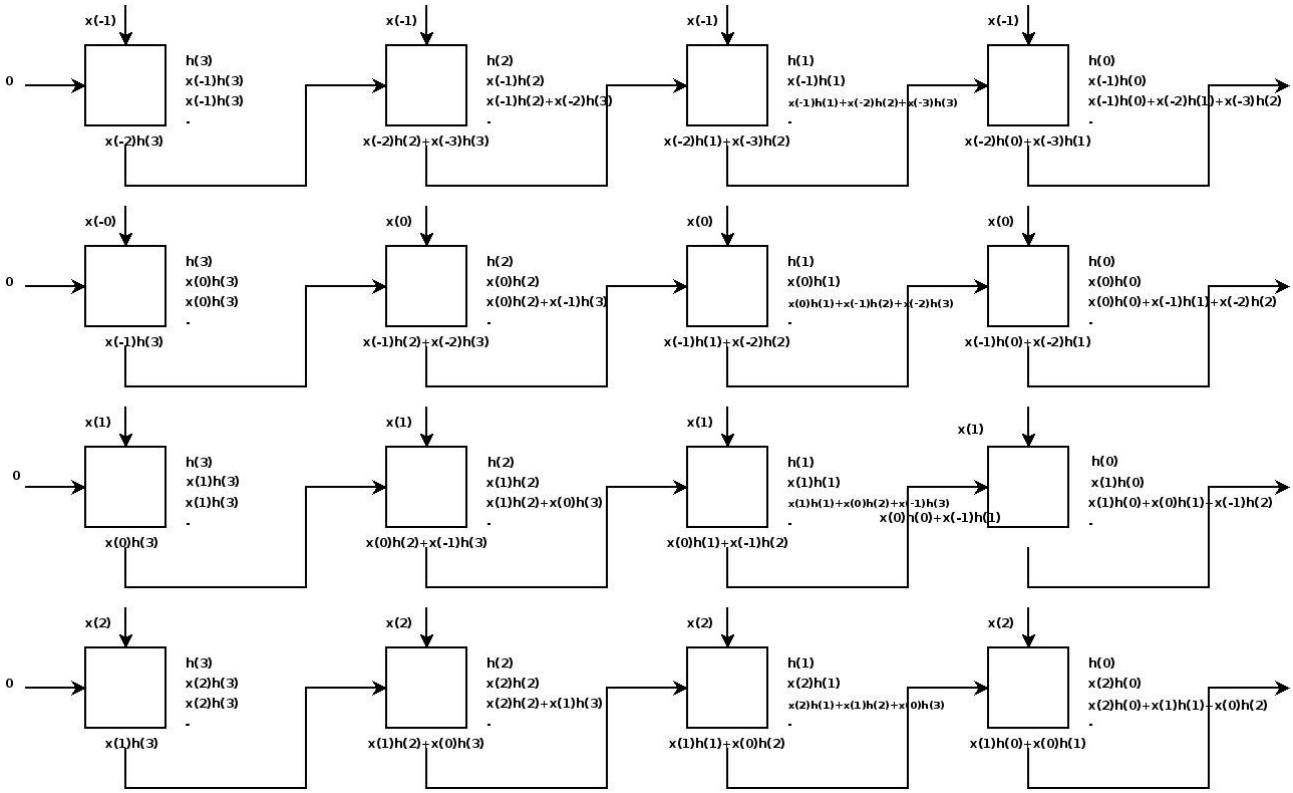
Σχήμα 6.3: Περιεχόμενα του αρχείου καταχωρητών και της εξόδου των κελιών μετά την πρώτη περίοδο λειτουργίας

Περίοδος Λειτουργίας 2



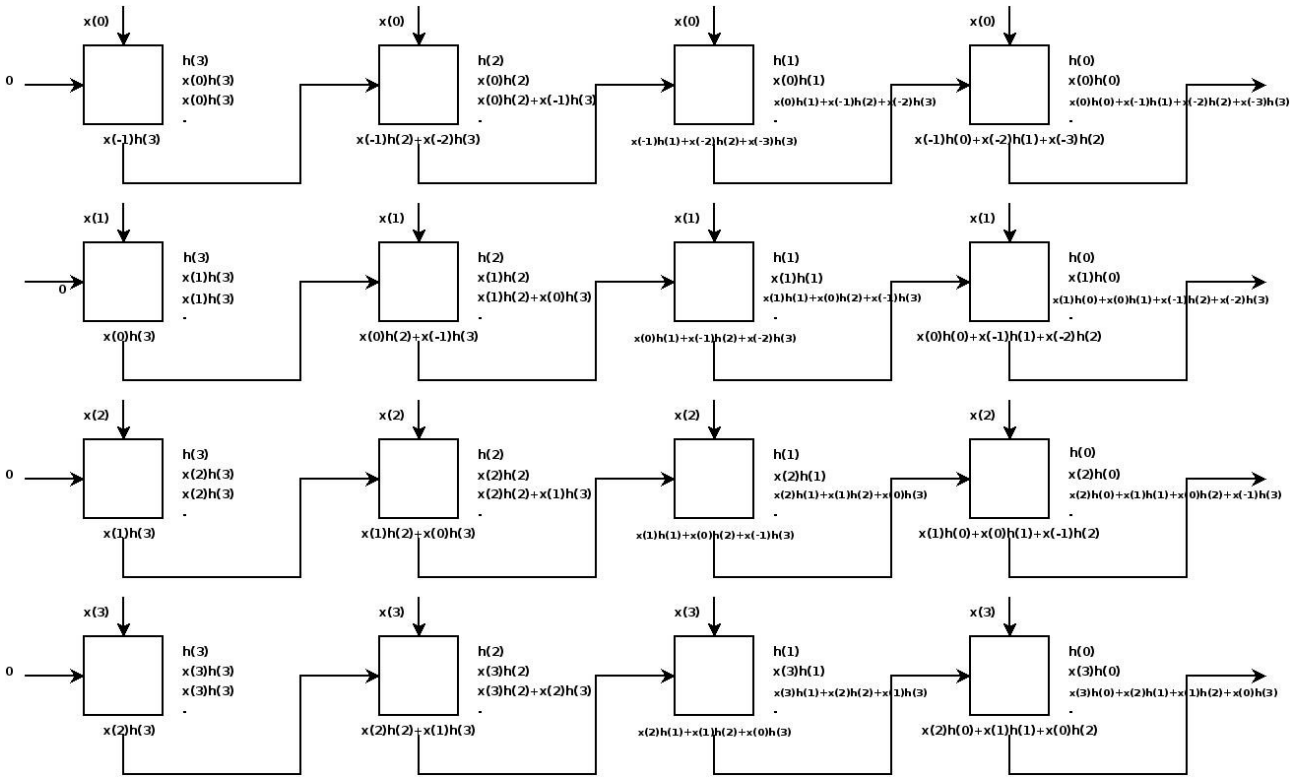
Σχήμα 6.4: Περιεχόμενα του αρχείου καταχωρητών και της εξόδου των κελιών μετά τη δεύτερη περίοδο λειτουργίας

Περίοδος Λειτουργίας 3



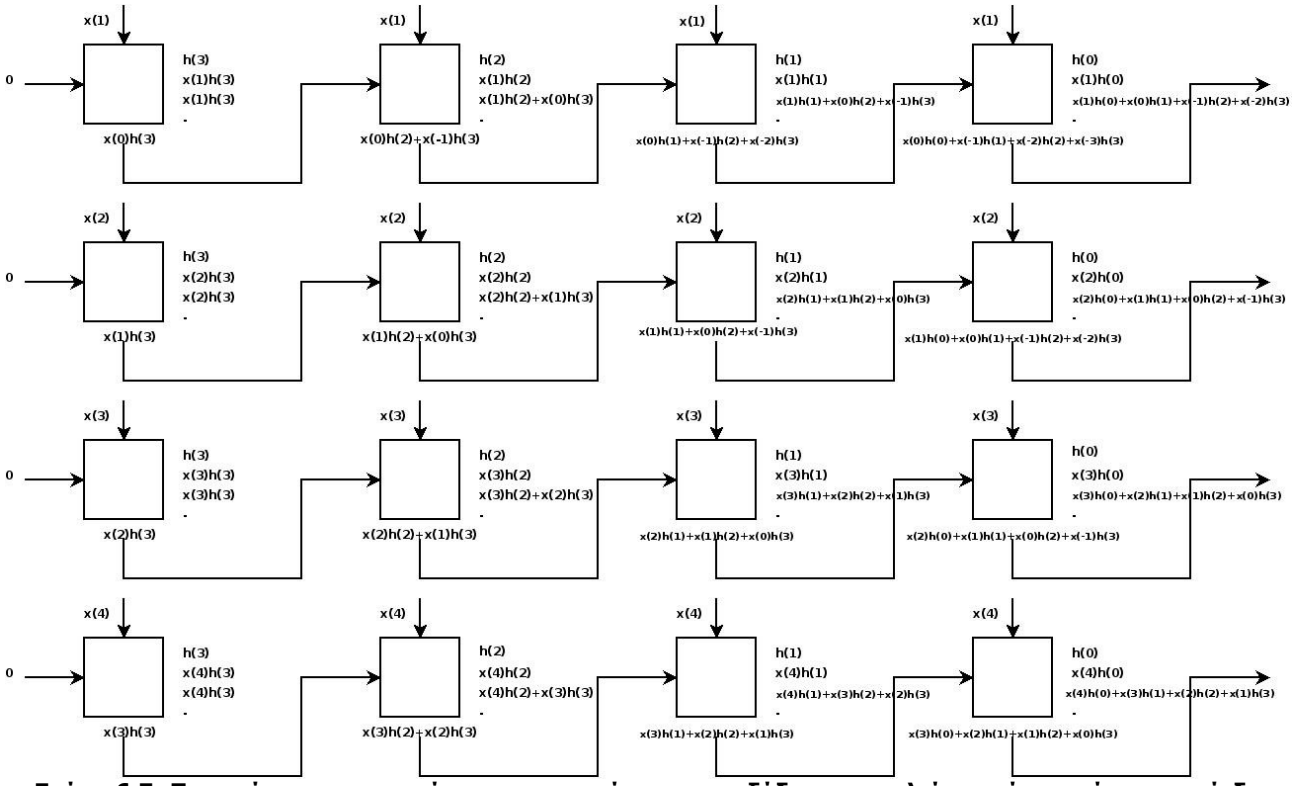
Σχήμα 6.5: Περιεχόμενα του αρχείου καταχωρητών και της εξόδου των κελιών μετά την τρίτη περίοδο λειτουργίας

Περίοδος Λειτουργίας 4



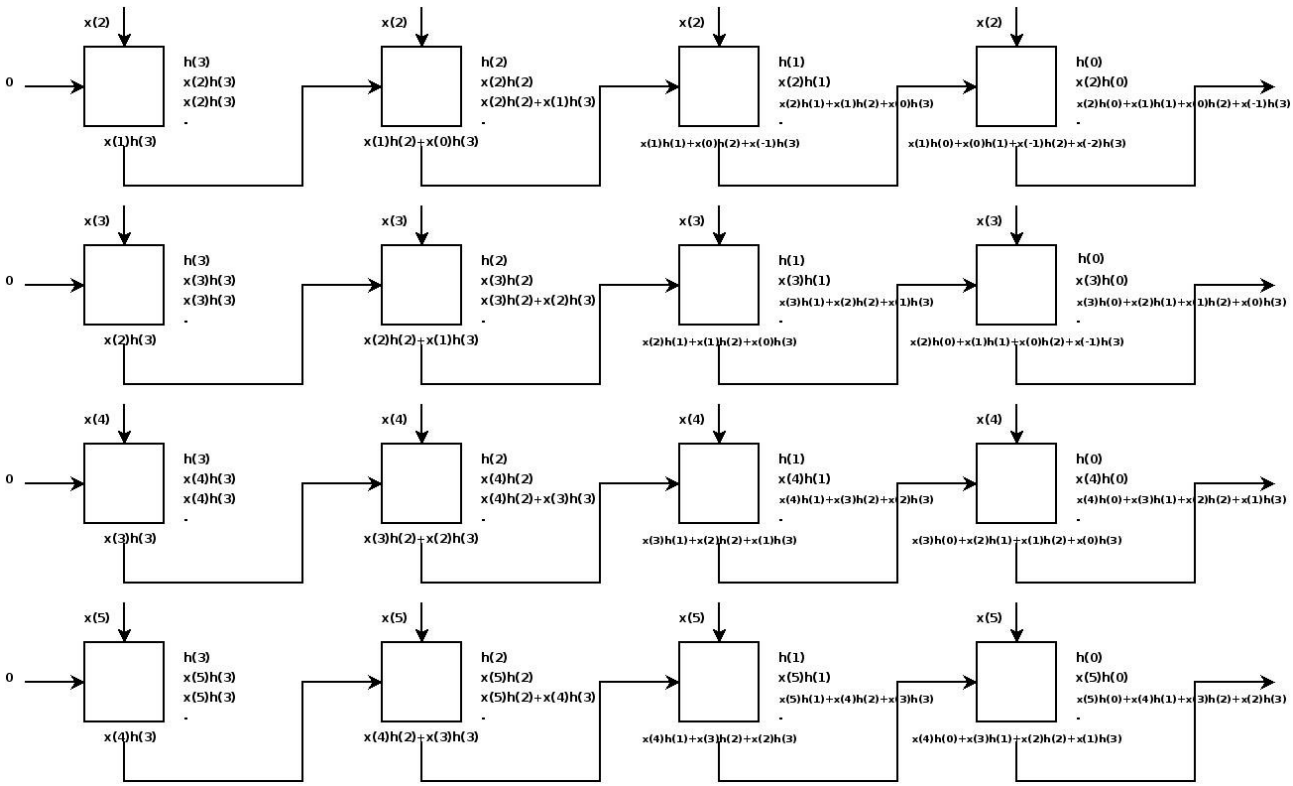
Σχήμα 6.6: Περιεχόμενα του αρχείου καταχωρητών και της εξόδου των κελιών μετά την τέταρτη περίοδο λειτουργίας. Παραγωγή πρώτης ομάδας αποτελεσμάτων.

Περίοδος Λειτουργίας 5



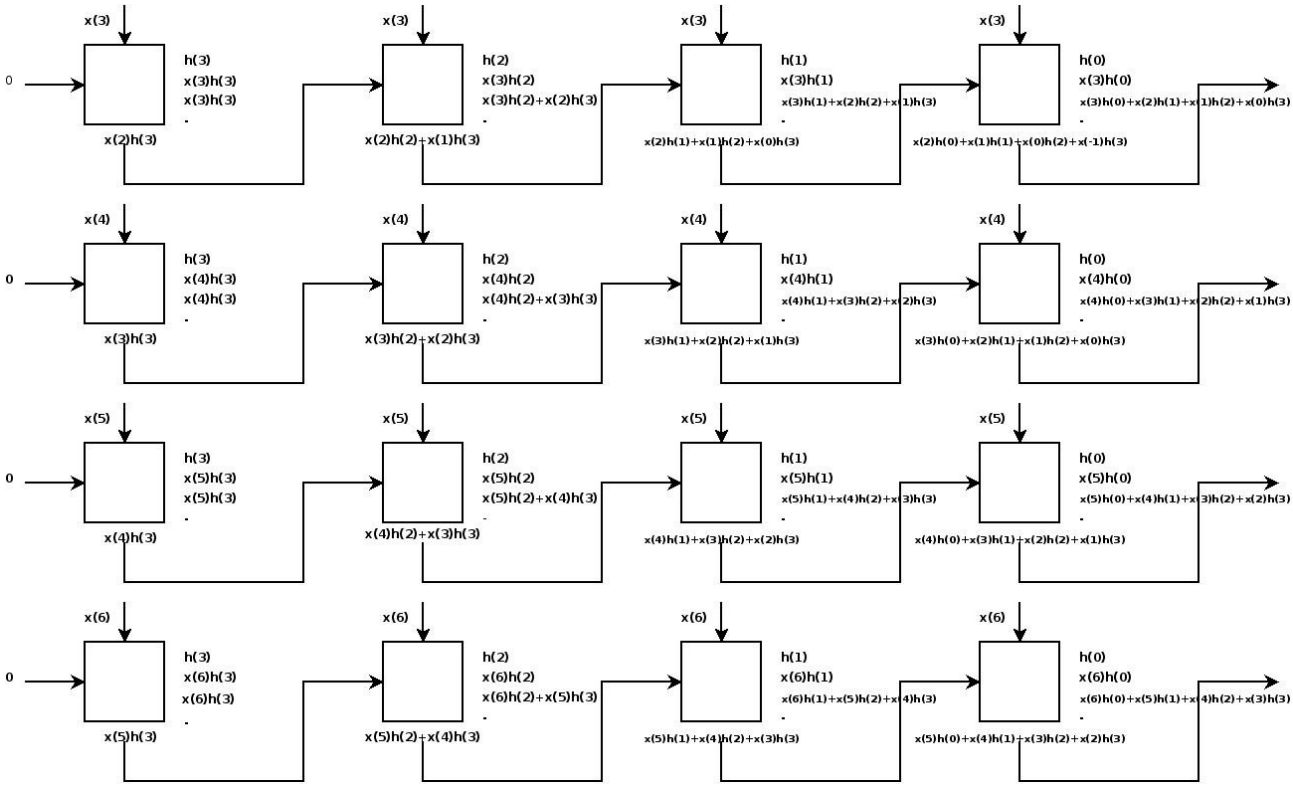
Σχήμα 6.7: Περιεχόμενα του αρχείου καταχωρητών και της εξόδου των κελιών μετά την πέμπτη περίοδο λειτουργίας

Περίοδος Λειτουργίας 6



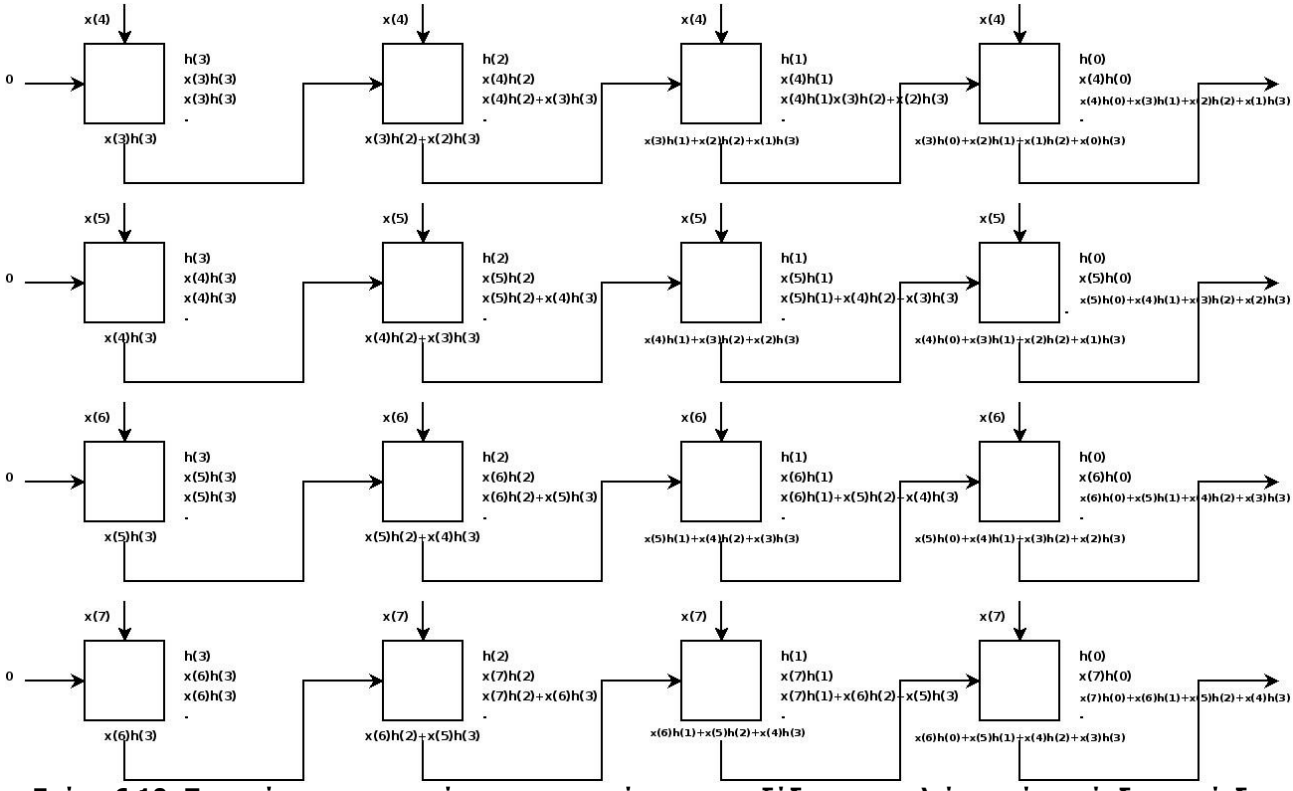
Σχήμα 6.8: Περιεχόμενα του αρχείου καταχωρητών και της εξόδου των κελιών μετά την έκτη περίοδο λειτουργίας

Περίοδος Λειτουργίας 7



Σχήμα 6.9: Περιεχόμενα του αρχείου καταχωρητών και της εξόδου των κελιών μετά την έβδομη περίοδο λειτουργίας

Περίοδος Λειτουργίας 8



Σχήμα 6.10: Περιεχόμενα του αρχείου καταχωρητών και της εξόδου των κελιών μετά την όγδοη περίοδο λειτουργίας. Παραγωγή δεύτερου συνόλου τελικών αποτελεσμάτων.

6.2.2.5. Έλεγχος Επιδόσεων

Για τον έλεγχο των επιδόσεων της απεικόνισης έγινε συγκριτική λειτουργία μεταξύ αυτής και ενός άλλου τρόπου απεικόνισης, με πλήρη χρήση το software και μηδενική χρήση hardware. Για το δεύτερο τρόπο απλώς αναπτύχθηκε ένας κώδικας σε γλώσσα C, ο οποίος υλοποιεί πλήρως ένα φίλτρο FIR 4-tap και στη συνέχεια τοποθετήθηκε στη μνήμη RAM του συστήματος MicroSoc για να εκτελεστεί από τον επεξεργαστή ARM.

Πριν παρουσιαστεί ο κώδικας C που χρησιμοποιήθηκε θα γίνει σύντομη αναφορά στα δείγματα εισόδου με τα οποία τροφοδοτήθηκαν τα φίλτρα που υλοποιήθηκαν. Παραπάνω έγινε η υπόθεση πως χρησιμοποιείται διάνυσμα εισόδου με μήκος 10 στοιχεία, χωρίς να υπολογίζονται τα μηδενικά στοιχεία που πρέπει να τοποθετηθούν λόγω των υποθέσεων που έγιναν. Έγινε η υπόθεση πως δειγματοληπτήθηκε η γραφική παράσταση ενός σήματος που ακολουθεί την κανονική κατανομή με μέγιστο στο 16 και τα δείγματα εισόδου προκύπτουν από αυτή τη δειγματοληψία. Τα δείγματα που λαμβάνονται είναι τέτοια, ώστε δύο συνεχόμενα δείγματα να είναι το ένα διπλάσιο του άλλου. Έτσι τα δείγματα εισόδου που χρησιμοποιήθηκαν είναι

$$[0\ ,\ 2\ ,\ 4\ ,\ 8\ ,\ 16\ ,\ 16\ ,\ 8\ ,\ 4\ ,\ 2\ ,\ 0]$$

Ακόμα χρησιμοποιήθηκαν ενδεικτικά ως συντελεστές βαρύτητας του φίλτρου τους αριθμούς 1, 2, 3 και 4.

Το Πλαίσιο 6.1 παρουσιάζει τον κώδικα σε C, ο οποίος υλοποιεί ένα φίλτρο FIR 4-tap.

Ο κώδικας που παρουσιάζεται στο Πλαίσιο 6.1 είναι σε μορφή έτοιμη προς εκτέλεση στο σύστημα Microsoc, καθώς έχουν προστεθεί στα κατάλληλα σημεία οι οδηγίες μεταγλώττισης *#pragma*. Υπάρχουν δύο μορφές των οδηγιών αυτών. Αφενός υπάρχει το κομμάτι κώδικα μεταξύ των οδηγιών #pragma arm section code, οι οποίες καθορίζουν ποιο είναι το κύριο κομμάτι κώδικα, το τμήμα *C_Entry*. Αφετέρου υπάρχει το κομμάτι κώδικα μεταξύ των οδηγιών *#pragma arm section rwdata="pinakas"*. Το κομμάτι αυτό καθορίζει ορισμένα τμήματα μέσα στη μνήμη δεδομένων ROM, τα οποία θα αποτελούν μεταβλητές για εγγραφή και ανάγνωση από το πρόγραμμα. Για παράδειγμα στον παραπάνω κώδικα, έχει δεσμευτεί ένα κομμάτι της μνήμης για χρήση του ως ένας πίνακας 8 στοιχείων, στον οποίο πρόκειται να αποθηκευτούν κατά τη διάρκεια εκτέλεσης του προγράμματος οι τελικές τιμές εξόδου του φίλτρου.

Η λειτουργία του κώδικα είναι απλή. Αρχικά, γίνεται μηδενισμός των στοιχείων του πίνακα εξόδου και στην συνέχεια ακολουθεί ο βασικός βρόχος υλοποίησης του φίλτρου. Αυτός επαναλαμβάνεται τόσες φορές, όσες είναι το πλήθος των δειγμάτων εξόδου που επιθυμούμε. Σε κάθε του επανάληψη ορίζεται μια τοπική

μεταβλητή για συσσώρευση της τελικής τιμής της τρέχουσας εξόδου. Σε κάθε επανάληψη ενός δεύτερου εσωτερικού βρόχου, που επαναλαμβάνεται τόσες φορές όση είναι και η τάξη του φίλτρου, γίνεται πολλαπλασιασμός του συντελεστή βαρύτητας που αντιστοιχεί στον αριθμό του βρόχου επανάληψης με το κατάλληλο δείγμα εισόδου, σύμφωνα πάντα με το μαθηματικό τύπο περιγραφής του φίλτρου. Τα γινόμενα αυτά συσσωρεύονται στην τοπική μεταβλητή συσσώρευσης που αναφέρθηκε προηγουμένως και στο τέλος της εκτέλεσης του εσωτερικού βρόχου το περιεχόμενο της μεταβλητής αυτής αποθηκεύεται στην κατάλληλη θέση στον πίνακα εξόδου στη μνήμη ROM.

```
#define NTAPS 4
#define INP_SIZE (3 * NTAPS - 1)

static const int w[NTAPS] = { 1, 2, 3, 4 };
static const int x[INP_SIZE] = { 0,0,0,2,4,8,16,16,8,4,2 };

extern void C_Entry(void);

#pragma arm section rwdata= "pinakas"
int y[8];
#pragma arm section rwdata

#pragma arm section code

void C_Entry(void)
{
    int i,j;

    for(i=0;i<8;i++) {
          y[i] = 0;
    }

    for(j = 0; j < 8; j++) {
          int accum = 0;

          for(i = 0; i < NTAPS; i++) {
                accum += w[i] * x[j-i+3];
          }

          y[j] = accum;
    }
}

#pragma arm section code
```

Πλαίσιο 6.1: Κώδικας C για την υλοποίηση ενός φίλτρου 4-tap στο MicroSoc

6.2.2.6. Συγκριτικά Αποτελέσματα

Αρχικά φορτώθηκε ο κώδικας για την υλοποίηση του φίλτρου χωρίς το επαναδιατάξιμο περιφερειακό στη μνήμη ROM του συστήματος. Μετά την εκτέλεση της εφαρμογής τα τελικά αποτελέσματα αποθηκεύτηκαν από τον κώδικα σε θέσεις της μνήμης RAM.

Η εικόνα της μνήμης μετά το τέλος της εκτέλεσης φαίνεται παρακάτω (Πλαίσιο 6.2).

0:	*2*	*8*	*22*	*52*
4:	*88*	*120*	*132*	*98*
8:	*x*	*x*	*x*	*x*
c:	*x*	*x*	*x*	*x*
10:	*x*	*x*	*x*	*x*
14:	*x*	*x*	*x*	*x*
...	*...*	*...*	*...*	*...*

Πλαίσιο 6.2: Περιεχόμενο της μνήμης RAM μετά την εκτέλεση της εφαρμογής στον ARM

Επομένως, το διάνυσμα με τα δείγματα εξόδου είναι το εξής :

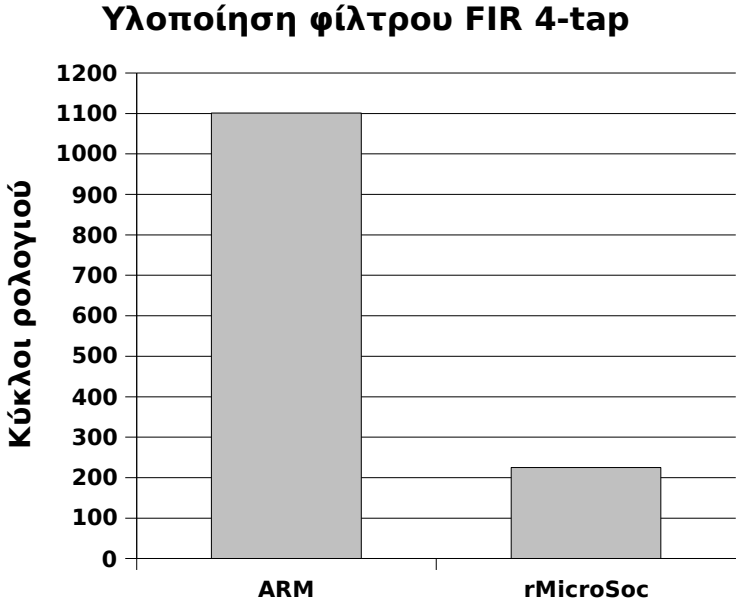
$$\{0,2,8,22,52,88,120,132,98\}$$

Οι κύκλοι που χρειάστηκαν για να υπολογιστούν τα δείγματα του διανύσματος εξόδου του φίλτρου ήταν συνολικά 1180 με το βρόχο for -που εκτελεί τους υπολογισμούς- να διαρκεί 1101.

Στη συνέχεια προετοιμάστηκε το rMicroSoc για την απεικόνιση σε αυτό του φίλτρου. Φορτώθηκαν τα περιεχόμενα διαμόρφωσης και τα δεδομένα εισόδου στις μνήμες διαμόρφωσης και δεδομένων αντίστοιχα. Επίσης γράφτηκε η κατάλληλη ρουτίνα εξυπηρέτησης της διακοπής και ο υπόλοιπος κώδικας υλοποίησης. Μετά την εκτέλεση της απεικόνισης, μελετήθηκαν οι κυματομορφές εξόδου, όπως προκύπτουν από το πρόγραμμα προσομοίωσης ModelSim[28]. Αφενός επιβεβαιώθηκε η ορθότητα της απεικόνισης και αφετέρου διαπιστώθηκε ότι μέχρι τα τελικά αποτελέσματα απαιτήθηκαν συνολικά 225 κύκλοι.

Στο Διάγραμμα 6.1 φαίνεται η σύγκριση στην απόδοση των δύο περιπτώσεων υλοποίησης.

Από τον αριθμό των κύκλων που χρειάστηκαν για να εξαχθούν τα αποτελέσματα, διαπιστώνεται ότι η υλοποίηση του φίλτρου FIR με τη χρήση του επαναδιατάξιμου περιφερειακού επιτάχυνε και βελτίωσε τους υπολογισμούς κατά περίπου 80%, σε σύγκριση με την υλοποίηση σε software μέσω του ARM.



Διάγραμμα 6.1: Σύγκριση της υλοποίησης φίλτρου FIR 4-tap στο rMicroSoc και στον ARM με βάση τον αριθμό των κύκλων ρολογιού

6.2.2.7. Πλεονεκτήματα-Μειονεκτήματα Απεικόνισης στο Υλικό

Έχοντας μια γενική εικόνα για την προτεινόμενη λύση απεικόνισης του φίλτρου στο επαναδιατάξιμο Microsoc, θα παρουσιαστούν συνοπτικά τα πλεονεκτήματα κα μειονεκτήματα της υλοποίησης αυτής.

Ένα πλεονέκτημα που διακρίνεται είναι η βελτίωση στη καθυστέρηση της εκτέλεσης των υπολογισμών. Όπως φάνηκε και στην προηγούμενη υποενότητα, υπάρχει μεγάλη βελτίωση στο χρόνο που απαιτείται για να εκτελεστούν οι υπολογισμοί για την υλοποίηση του φίλτρου FIR στο επαναδιατάξιμο MircroSoc, σε σύγκριση με την αντίστοιχη υλοποίηση αποκλειστικά μέσω software. Η αποδοτικότητα λοιπόν της απεικόνισης στο επαναδιατάξιμο υλικό είναι μεγάλη και αυτό ευνοεί την επιλογή της συγκεκριμένης λύσης.

Ένα άλλο πλεονέκτημα που μπορεί επίσης να αναφερθεί είναι η παραλληλία της εκτέλεσης των υπολογισμών, όταν χρησιμοποιηθεί η λύση της απεικόνισης στο υλικό. Αυτό σημαίνει πρακτικά ότι όσο χρόνο διαρκεί η εκτέλεση των υπολογισμών που υλοποιούν το φίλτρο FIR και επεξεργάζονται τις εισόδους του για να προκύψουν οι έξοδοι, είναι εφικτό να εκτελούνται παράλληλα άλλοι υπολογισμοί στον κυρίως σύστημα, στις κεντρικές μνήμες του ARM. Αυτό εξάλλου είναι και ένα σημαντικό πλεονέκτημα που προκρίνει τη λύση χρησιμοποίησης επαναδιατάξιμου υλικού για DSP επεξεργασία.

Υπάρχουν ωστόσο και κάποια μειονεκτήματα. Το σημαντικότερο από αυτά είναι η περιορισμένη ευελιξία της συγκεκριμένης λύσης, υπό την έννοια ότι

εφαρμόζεται μονάχα όταν η τάξη του φίλτρου FIR που θέλουμε να υλοποιήσουμε είναι ίση (ή και μικρότερη) από το μέγεθος του επαναδιατάξιμου πίνακα. Ειδικότερα στο επαναδιατάξιμο MicroSoc που σχεδιάστηκε, στο οποίο ο πίνακας έχει μέγεθος 4, μπορούν να απεικονιστούν μέσω της τεχνικής που αναλύθηκε στην παράγραφο αυτή μόνο φίλτρα FIR με τάξη 4 ή μικρότερη. Δεν υπάρχει λοιπόν μεγάλη ευελιξία της τεχνικής για δεδομένο μέγεθος επαναδιατάξιμου πίνακα.

6.2.3.Φίλτρο FIR 16-tap στο MicroSoc

6.2.3.1. Κεντρική Ιδέα της Τεχνικής Απεικόνισης

Ένα δεύτερο είδος φίλτρου που υλοποιήθηκε και απεικονίστηκε στο rMicroSoc είναι ένα φίλτρο FIR 16-tap. Το φίλτρο αυτό έχει τάξη μεγαλύτερη από το μέγεθος του επαναδιατάξιμου πίνακα, οπότε η μέθοδος απεικόνισης που περιγράφηκε στην προηγούμενη ενότητα δεν μπορεί να εφαρμοστεί στη συγκεκριμένη περίπτωση, δεδομένου ότι η μέθοδος απεικόνισης της προηγούμενης ενότητας εφαρμόζεται μόνο σε φίλτρα με τάξη μικρότερη ή ίση με το μέγεθος του επαναδιατάξιμου πίνακα του συστήματος. Για το λόγο αυτό προτείνεται μια άλλη τεχνική, η οποία και περιγράφεται αναλυτικά στην παρούσα ενότητα.

Το φίλτρο FIR 16-tap περιγράφεται με τον παρακάτω μαθηματικό τύπο:

$$y_k = \sum_{j=0}^{15} x_{k-j} w_j \quad \text{ή αλλιώς}$$

$$y_k = x_k h_0 + x_{k-1} h_1 + x_{k-2} h_2 + x_{k-3} h_3 + \ldots + x_{k-14} h_{14} + x_{k-15} h_{15}$$

Στο φίλτρο που υλοποιήθηκε στα πλαίσια της παρούσας εργασίας χρησιμοποιήθηκαν τελικά οι εξής έξοδοι :

$$\begin{aligned}
y_0 &= x_0 h_0 + x_{-1} h_1 + x_{-2} h_2 + x_{-3} h_3 + \ldots + x_{-14} h_{14} + x_{-15} h_{15} \\
y_1 &= x_1 h_0 + x_0 h_1 + x_{-1} h_2 + x_{-2} h_3 + \ldots + x_{-13} h_{14} + x_{-14} h_{15} \\
y_2 &= x_2 h_0 + x_1 h_1 + x_0 h_2 + x_{-1} h_3 + \ldots + x_{-12} h_{14} + x_{-13} h_{15} \\
&\vdots \\
y_{15} &= x_{15} h_0 + x_{14} h_1 + x_{13} h_2 + x_{12} h_3 + \ldots + x_1 h_{14} + x_0 h_{15}
\end{aligned}$$

Η απεικόνιση που παρουσιάζεται στην παρούσα ενότητα βασίζεται σε μια κεντρική ιδέα, γνωστή ως της *Κοινής Χρήσης Εύρους Ζώνης της Μνήμης* (*Memory Bandwidth Sharing, MBS)[29].* Παρακάτω θα γίνει μια περιληπτική παρουσίασή της, καθώς αποτελεί την ουσία της απεικόνισης που

χρησιμοποιήθηκε για την υλοποίηση του φίλτρου 16-tap.

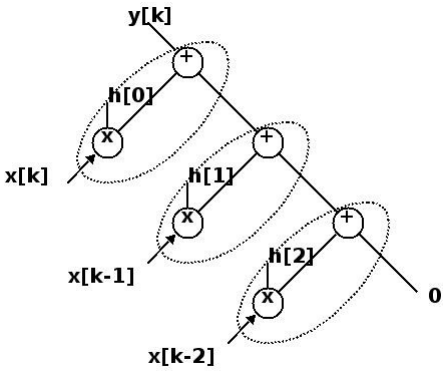
Η τεχνική του *MBS* βρίσκει εφαρμογή στις απεικονίσεις DSP εφαρμογών σε επαναδιατάξιμα συστήματα, στις οποίες οι βασικές λειτουργίες εκτελούνται σε επαναλαμβανόμενους βρόχους εκτέλεσης. Εξάλλου, αυτό συναντάται και στα φίλτρα FIR, όπου ο βασικός μαθηματικός τύπος περιγραφής τους -πάνω στον οποίο στηρίζεται η υλοποίησή τους- περιλαμβάνει βρόχους επανάληψης για τον υπολογισμό κάθε δείγματος εξόδου του φίλτρου.

Πολλές φορές, οι βρόχοι αυτοί, που υλοποιούνται στο επαναδιατάξιμο σύστημα, είναι άμεσα συνδεδεμένοι με τη λειτουργία της μνήμης δεδομένων, υπό την έννοια ότι σε κάποιο/α σημείο/α τους απαιτούν ανάγνωση δεδομένων από τη μνήμη για την εκτέλεση των υπολογισμών. Το γεγονός αυτό μειώνει τη συνολική αποδοτικότητα του συστήματος και είναι επιθυμητό να αντιμετωπιστεί. Ένας τρόπος είναι να μειωθεί ο ενεργός αριθμός των υπολογισμών που σχετίζονται με τη μνήμη, κάτι το οποίο επιτυγχάνεται με τη συγκεκριμένη τεχνική που περιγράφεται. Αυτή η μείωση μπορεί να πραγματοποιηθεί, πρακτικά, με την επαναχρησιμοποίηση δεδομένων της μνήμης στους βρόχους των DSP αλγορίθμων. Θα χρησιμοποιηθεί σαν παράδειγμα ένας αλγόριθμος FIR φίλτρου 2-tap, ο οποίος ορίζεται ως

$$y[k] = \sum_{j=0}^{2} h_j \times x[k-j]$$

με τους γνωστούς συμβολισμούς (h για τους σταθερούς συντελεστές του φίλτρου, x και y τα δείγματα εισόδου και εξόδου αντίστοιχα).

Το Σχήμα 6.11 παρουσιάζει σε μορφή δέντρου τους υπολογισμούς που πρέπει να γίνουν για να προκύψει το τελικό δείγμα εξόδου. Στο διάγραμμα αυτό έχουν σημειωθεί με κύκλο, πόσα διαφορετικά κελιά (μονάδες επεξεργασίας) πρέπει να χρησιμοποιηθούν για να πραγματοποιηθεί ο υπολογισμός και τι λειτουργία επιτελεί το καθένα από αυτά.



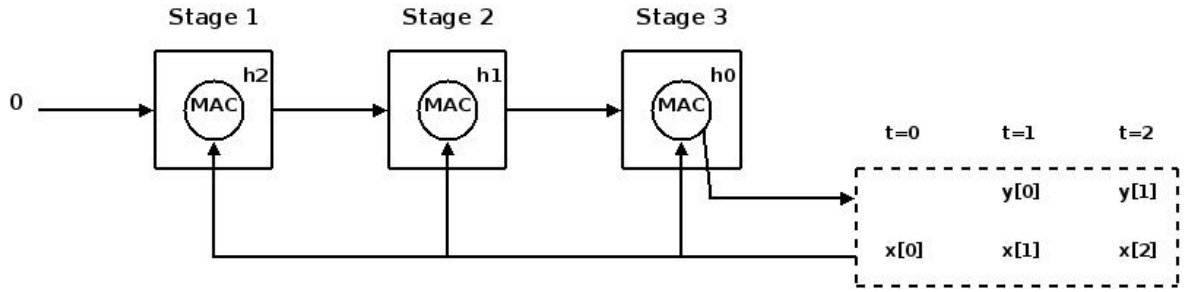
Σχήμα 6.11: Δέντρο απεικόνισης του φίλτρου FIR 2-tap

Παρατηρείται ότι συνολικά πρέπει να χρησιμοποιηθούν 3 διαφορετικές μονάδες επεξεργασίας, κάθε μια από τις οποίες εκτελεί την πράξη του πολλαπλασιασμού

και στη συνέχεια άθροισης του γινομένου με άλλα δεδομένα (λειτουργία συσσώρευσης). Γίνεται η υπόθεση, για χάρη της γενικότητας, ότι είτε η σύνθετη αυτή λειτουργία επιτελείται με μία μόνο διαμόρφωση είτε επιτελείται σε πολλαπλούς κύκλους διαμόρφωσης, χωρίς να παίζει ουσιαστικό ρόλο η διαφοροποίηση μεταξύ των δύο περιπτώσεων.

Από τα παραπάνω προκύπτει το συμπέρασμα ότι τα 3 διαφορετικά κελιά που θα χρησιμοποιηθούν είναι δυνατό να μοιραστούν τα ίδια δεδομένα ανάγνωσης από τη μνήμη. Αυτό είναι εφικτό διότι κάθε κελί αντιπροσωπεύει και διαφορετικό στάδιο pipeline. Για παράδειγμα, όταν το κελί με το συντελεστή h0 βρίσκεται στην επανάληψη i=0, τα κελιά με συντελεστές h1 και h2 βρίσκονται στις επαναλήψεις i=1 και i=2 αντίστοιχα. Επιπλέον, λόγω του γεγονότος ότι τα τρία αυτά κελιά προσπελαύνουν συνεχόμενα δεδομένα του διανύσματος εισόδου και συγκεκριμένα τα $x[i]$, $x[i-1]$ και $x[i-2]$, τελικά προσπελαύνουν το ίδιο ακριβώς δείγμα, το $x[0]$. Συνεπώς, οι τρεις υπολογισμοί που σχετίζονται με τη μνήμη είναι ίδιοι και τα κελιά μπορούν να μοιράζονται τα ίδια δεδομένα από τη μνήμη.

Μια γραφική αναπαράσταση της διαπίστωσης για την παράλληλη λειτουργία των τριών κελιών δίνει το Σχήμα 6.12.



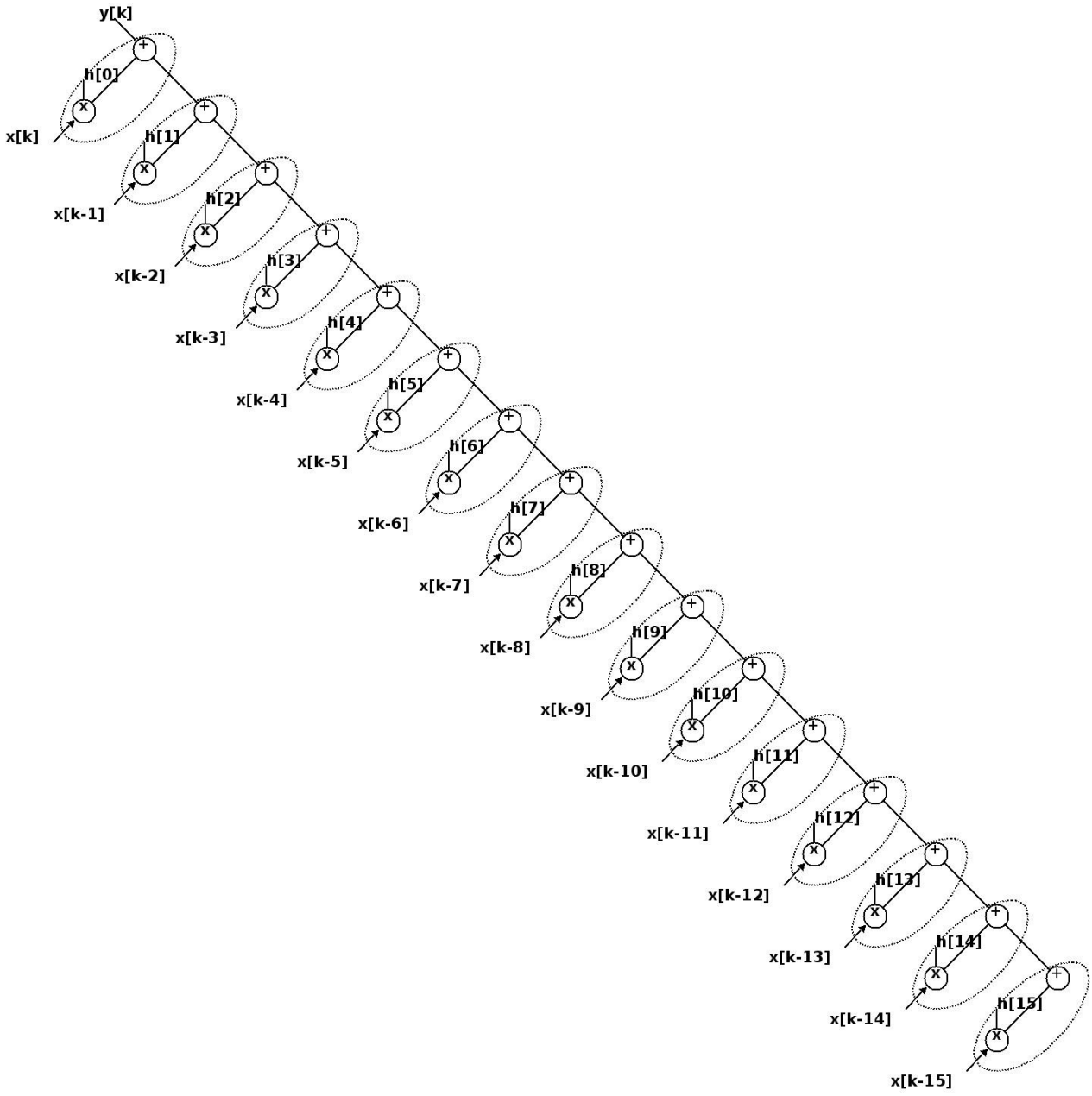
Σχήμα 6.12: Παράλληλη λειτουργία των μονάδων επεξεργασίας σε σχέση με τα δεδομένα από τη μνήμη (στα δεξιά του σχήματος)

Η τεχνική αυτή δεν απαιτεί τη χρήση επιπλέον υλικού, το οποίο είναι ένα ισχυρό θετικό στοιχείο της. Επιπλέον, στην περίπτωση εφαρμογής του στο rMicroSoc, το σύστημα διαθέτει ως δεδομένο χαρακτηριστικό του την κοινή χρήση δεδομένων στα κελιά. Συνεπώς, η εφαρμογή της συγκεκριμένης μεθόδου για την απεικόνιση φίλτρου 16-tap στο rMicroSoc είναι εφικτή. Στην ουσία η τεχνική του *MBS* εκμεταλλεύεται μια καλύτερη τοποθέτηση των στοιχείων του πίνακα, που συνδράμει στην επαναχρησιμοποίηση των δεδομένων της μνήμης και την εκτέλεση σε pipeline.

6.2.3.2. Εφαρμογή της Τεχνικής για Φίλτρο 16-tap

Εφαρμόζοντας τώρα τη συγκεκριμένη τεχνική στην περίπτωση του φίλτρου 16-tap, κατασκευάζεται αρχικά ένα παρόμοιο δέντρο που παρουσιάζει την όλη

διαδικασία των υπολογισμών και το οποίο απεικονίζει το Σχήμα 6.13.



Σχήμα 6.13: Γράφος απεικόνισης του FIR φίλτρου 16-tap

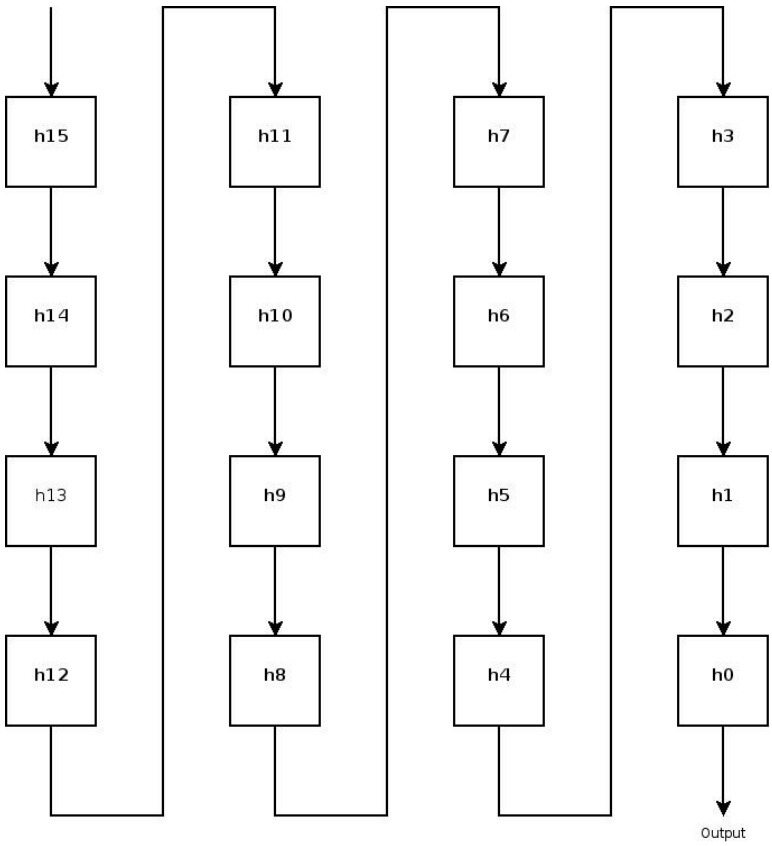
Διακρίνεται ότι πρέπει να χρησιμοποιηθούν συνολικά 16 διαφορετικά επαναδιατάξιμα κελιά. Κάθε ένα από αυτά αντιστοιχεί σε έναν διαφορετικό αριθμό επανάληψης -όσον αφορά στο βρόχο υλοποίησης του φίλτρου- και αντιπροσωπεύει ένα στάδιο pipeline. Με τη συγκεκριμένη απεικόνιση, δηλαδή, δημιουργείται συνολικά ένα pipeline 16 σταδίων, με τα δεδομένα που υπολογίζονται σε κάποιο στάδιο να χρησιμοποιούνται σε όλα τα επόμενα. Όλα τα κελιά θα τροφοδοτούνται σε κάθε κύκλο λειτουργίας με το ίδιο δείγμα εισόδου από τη μνήμη δεδομένων και θα εκτελούν την ίδια ακριβώς λειτουργία, ωστόσο το κάθε ένα θα πρέπει να χρησιμοποιεί στους υπολογισμούς του ένα διαφορετικό συντελεστή βαρύτητας του φίλτρου.

Σε κάθε κύκλο λειτουργίας του επαναδιατάξιμου πίνακα υπολογίζονται ταυτόχρονα πολλά διαφορετικά γινόμενα στα κελιά. Τα γινόμενα αυτά θα αθροιστούν με τα γινόμενα που υπολογίζονται σε μεταγενέστερους κύκλους και σε διαφορετικά κελιά, έχοντας ως τελικό αποτέλεσμα τον υπολογισμό των δειγμάτων εξόδου του φίλτρου. Υπό την έννοια αυτή εφαρμόζεται ένα είδος αναδίπλωσης των βρόχων υπολογισμού των δειγμάτων εξόδου του φίλτρου.

Το pipeline που υλοποιείται θα χρειαστεί ένα χρονικό διάστημα αρχικοποίησης μέχρι να "γεμίσει" και να αρχίσει να παράγει δείγματα εξόδου. Το διάστημα αυτό ισούται με N κύκλους λειτουργίας, όπου N=16 η τάξη του φίλτρου. Μετά από 16, λοιπόν, κύκλους θα προκύψει το πρώτο δείγμα εξόδου, το y_0. Από το σημείο αυτό και μετά τα επόμενα δείγματα εξόδου θα εξάγονται συνεχώς, ένα διαφορετικό σε κάθε κύκλο λειτουργίας, δημιουργώντας έτσι ένα υψηλό throughput της τάξης του ενός δείγματος ανά κύκλο. Επίσης, από το στάδιο της αρχικοποίησης και έπειτα κάθε τοπικό γινόμενο που προκύπτει στα κελιά του πίνακα θα συμμετάσχει σε κάποιο μεταγενέστερο δείγμα εξόδου -κάτι το οποίο δε γινόταν στο διάστημα αρχικοποίησης, όπου παράγονται πολλές αδιάφορες έξοδοι.

Συμπερασματικά, η μέθοδος αυτή απεικόνισης του φίλτρου 16-tap κάνει χρήση όλων των κελιών του πίνακα με διαφορετικό συντελεστή βαρύτητας αποθηκευμένο στο καθένα. Εξάλλου, υπενθυμίζεται ότι το φίλτρο έχει ακριβώς 16 συντελεστές βαρύτητας, όσα είναι και τα κελιά του επαναδιατάξιμου πίνακα. Τη βασική ιδέα, η οποία πρέπει να εφαρμοστεί, παρουσιάζει το Σχήμα 6.14.

Το διάγραμμα αυτό αποτελεί τη μεταφορά του δέντρου υπολογισμού που παρουσιάζει το Σχήμα 6.13 πάνω στον επαναδιατάξιμο πίνακα και δείχνει ότι όλα τα κελιά του πίνακα έχουν τοποθετηθεί νοητά σε μια ευθεία γραμμή, με κάθε κελί του πίνακα να πολλαπλασιάζει το δείγμα εισόδου που λαμβάνει με έναν διαφορετικό συντελεστή. Πρακτικά λοιπόν προσπαθούμε να υλοποιήσουμε τη γραμμή καθυστέρησης που χρησιμοποιεί το φίλτρο. Τα δείγματα εξόδου του φίλτρου εξάγονται από το τελευταίο κελί αυτής της νοητής γραμμής. Επίσης οι γραμμές που φαίνονται στο παραπάνω διάγραμμα να συνδέουν τα κελιά μεταξύ τους δείχνουν για κάθε κελί, σε ποιο άλλο κελί του πίνακα στέλνει σε κάθε κύκλο την έξοδο του προηγούμενου κύκλου λειτουργίας. Οι γραμμές δείχνουν δηλαδή, κατά κάποιο τρόπο, τη διαδρομή που ακολουθεί η συσσώρευση γινομένων, από την οποία τελικά θα προκύψει το τελικό αποτέλεσμα.



Σχήμα 6.14: Συσσώρευση των ενδιάμεσων γινομένων για την παραγωγή των δειγμάτων εξόδου του φίλτρου

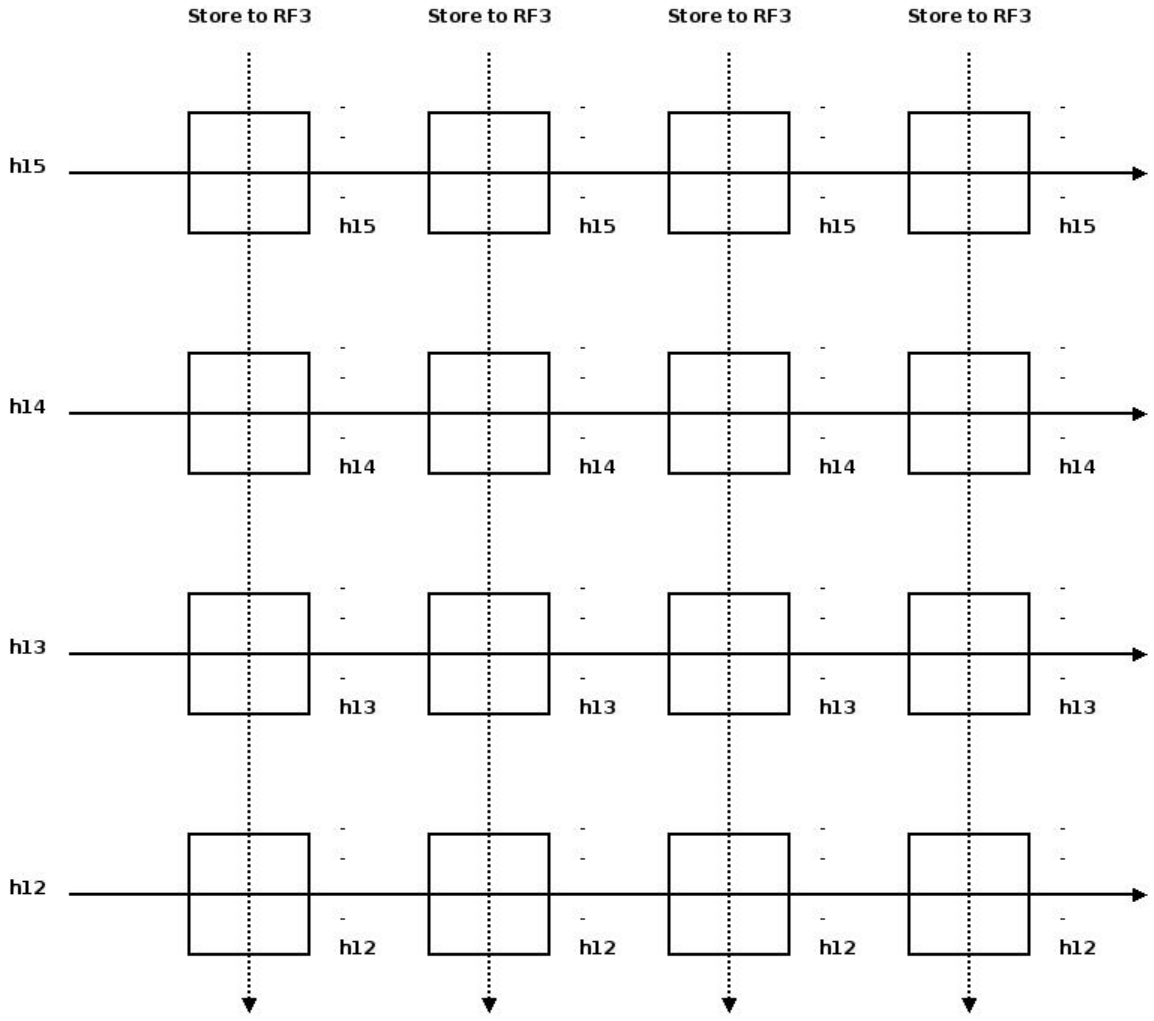
6.2.3.3. Φόρτωση Συντελεστών Βαρύτητας

Στο σημείο αυτό εμφανίζεται η πρακτική δυσκολία του τρόπου φόρτωσης των διαφορετικών συντελεστών βαρύτητας σε κάθε επαναδιατάξιμο κελί του πίνακα, υπό την έννοια ότι η τροφοδότηση δεδομένων από τη μνήμη δεδομένων στον επαναδιατάξιμο πίνακα γίνεται -εκ των πραγμάτων- είτε κατά στήλες είτε κατά γραμμές. Για να λυθεί το ζήτημα αυτό, επιλέχθηκε ο καταχωρητής του τοπικού αρχείου καταχωρητών στον οποίο αποθηκεύεται ο συντελεστής βαρύτητας να είναι κοινός ανά στήλη, αλλά όχι ίδιος για όλα τα κελιά. Υπενθυμίζεται ότι στην προηγούμενη μέθοδο απεικόνισης φίλτρου 4-tap οι συντελεστές βαρύτητας φορτώνονται από τη μνήμη δεδομένων ανά στήλη και αποθηκεύονται στον καταχωρητή RF0, στον ίδιο για όλα τα κελιά του πίνακα. Υπήρχε δηλαδή μια γενικότητα στη σχεδίαση στο σημείο αυτό. Ωστόσο, για το φίλτρο 16-tap αυτό υποχρεωτικά δεν μπορεί να ακολουθηθεί, για το λόγο που αναφέρθηκε.

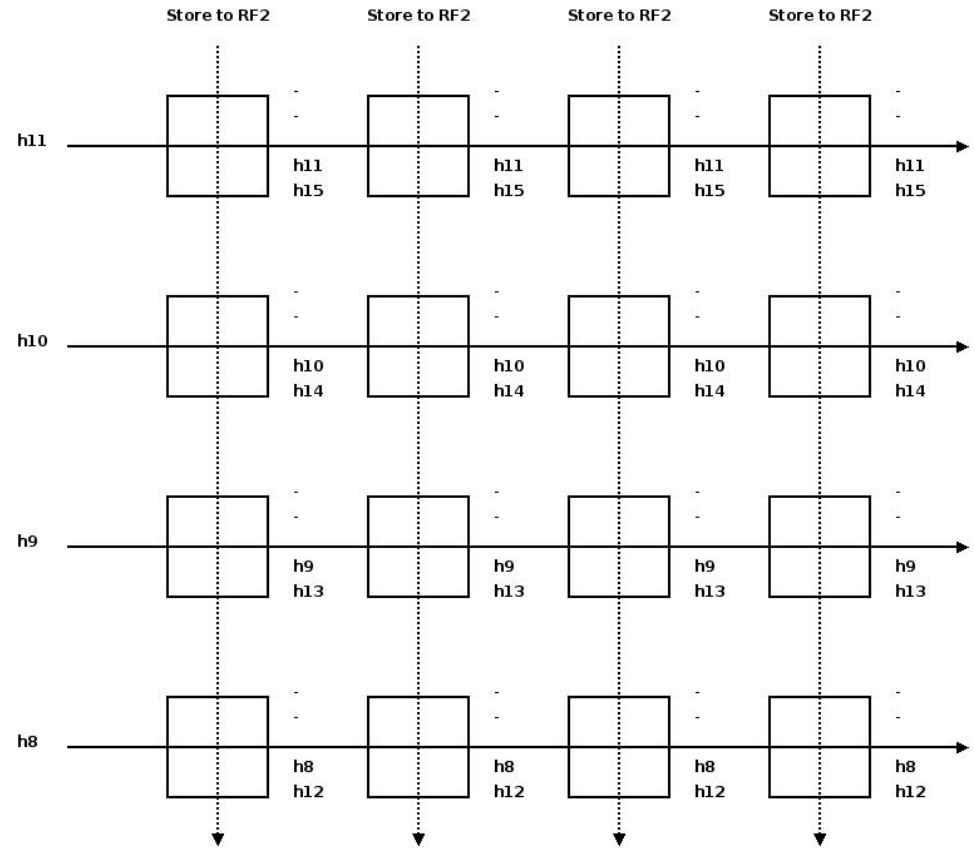
Η φόρτωση των συντελεστών γίνεται σε τέσσερα στάδια, στιγμιότυπα των οποίων φαίνονται σε επόμενα σχήματα (Σχήμα 6.15-Σχήμα 6.18). Από αυτά φαίνεται πως τα δεδομένα από τη μνήμη δεδομένων (οι συντελεστές βαρύτητας) φτάνουν στον πίνακα ανά γραμμές, με την κατάλληλη σειρά ώστε να προκύψει η τελική επιθυμητή διάταξη στα κελιά του πίνακα. Η αποθήκευση γίνεται επίσης ανά γραμμές και σε κάθε κύκλο οι συντελεστές αποθηκεύονται στον ίδιο

καταχωρητή του αρχείου καταχωρητών για όλες τις γραμμές. Βέβαια, ο καταχωρητής αυτός είναι διαφορετικός για κάθε σύνολο συντελεστών που φτάνουν στον πίνακα, έτσι ώστε όταν ολοκληρωθούν οι τέσσερις φάσεις στην πρώτη στήλη οι σωστοί συντελεστές να βρίσκονται στον RF3, στη δεύτερη στον RF2, στην τρίτη στον RF1 και στην τέταρτη στον RF0. Στο τέλος όλοι οι υπόλοιποι καταχωρητές των αρχείων είναι γεμάτοι με δεδομένα, τα οποία όμως είναι αδιάφορα.

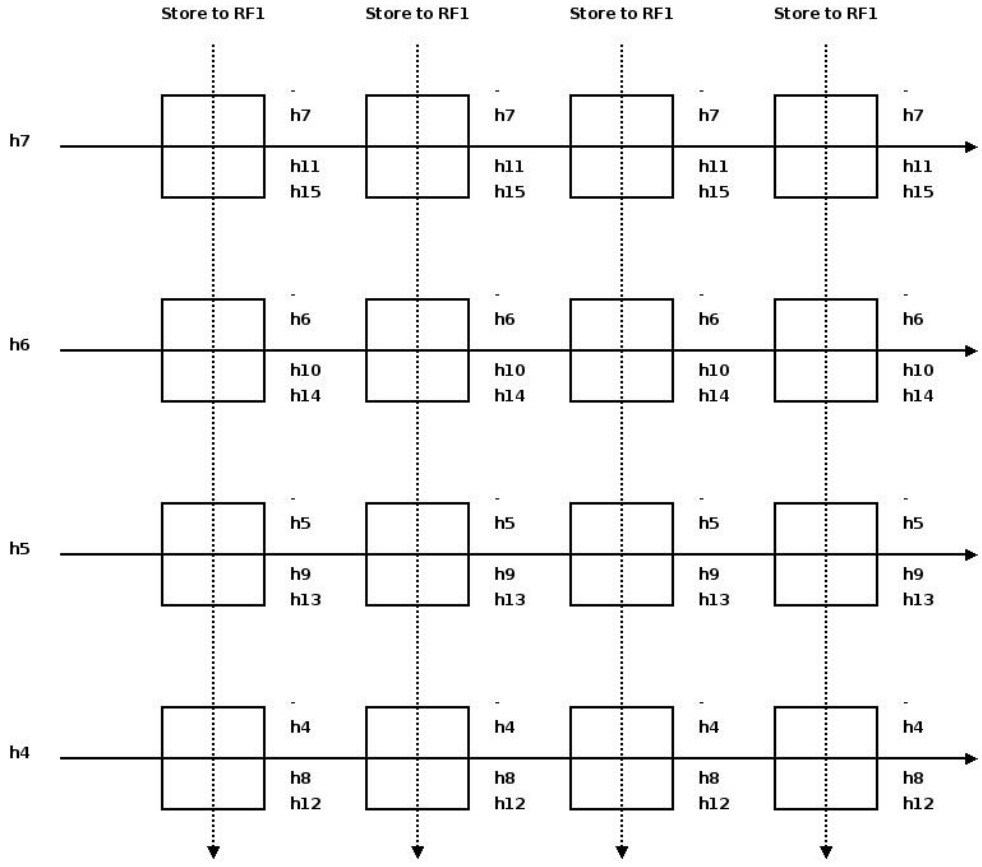
Στα επόμενα σχήματα (Σχήμα 6.15-Σχήμα 6.18) απεικονίζονται τα στάδια που ακολουθούνται για την αποθήκευση των συντελεστών στα κελιά.



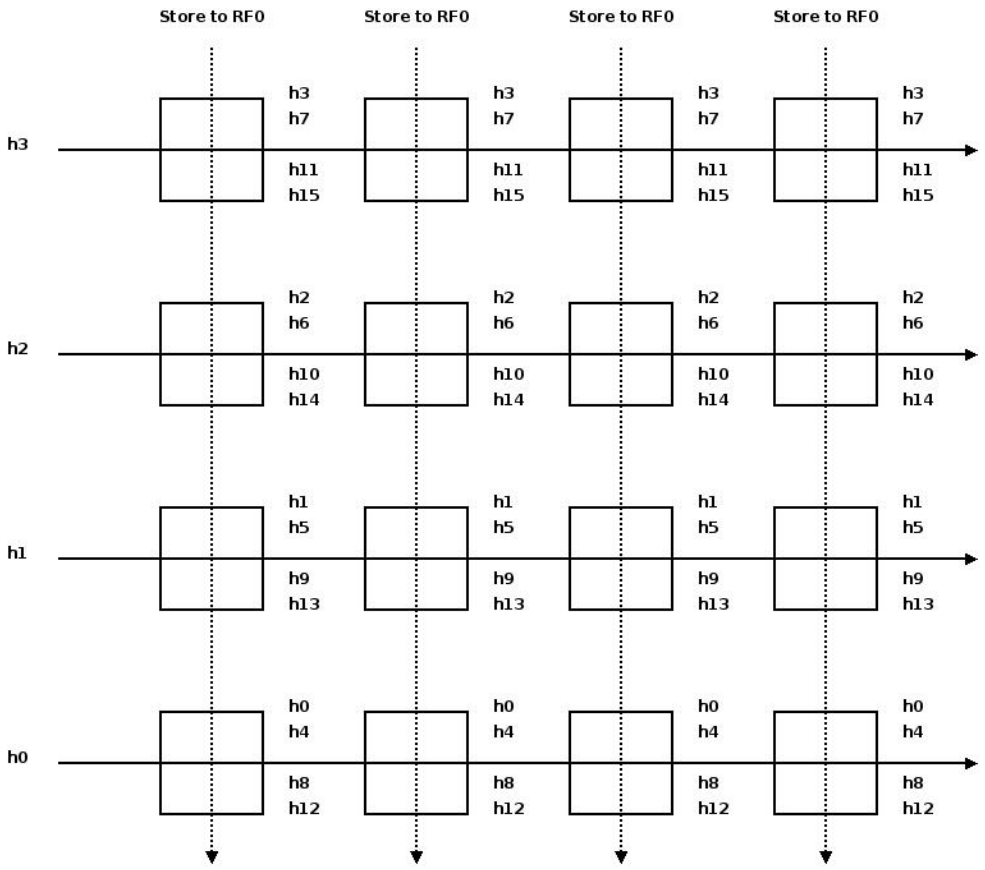
Σχήμα 6.15: Φόρτωση των συντελεστών h12-h15 του φίλτρου στον πίνακα



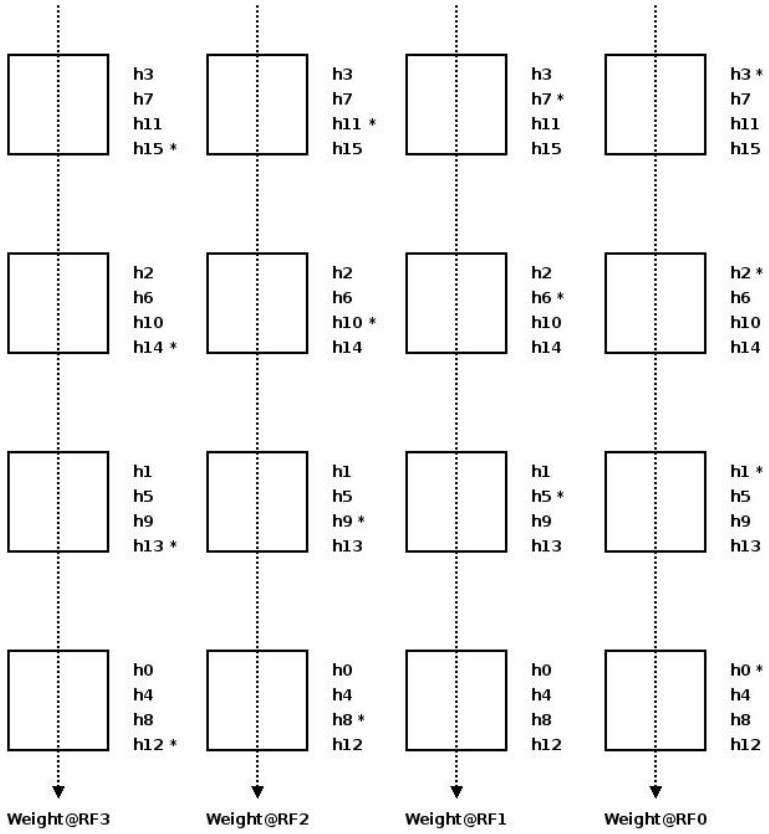
Σχήμα 6.16: Φόρτωση των συντελεστών h8-h11 του φίλτρου στον πίνακα



Σχήμα 6.17: Φόρτωση των συντελεστών h4-h7 του φίλτρου στον πίνακα



Σχήμα 6.18: Φόρτωση των συντελεστών h1-h3 του φίλτρου στον πίνακα



Σχήμα 6.19: Το περιεχόμενο των RF των κελιών, μετά τη φόρτωση

Η τελική εικόνα των περιεχομένων των αρχείων καταχωρητών του πίνακα, μετά τη φάση φόρτωσης, φαίνεται στο Σχήμα 6.19, όπου με ένα * είναι σημειωμένοι οι συντελεστές που ενδιαφέρουν από κάθε κελί.

Στη συνέχεια θα παρουσιαστούν οι λέξεις διαμόρφωσης που απαιτούνται για να φορτωθούν σωστά οι συντελεστές βαρύτητας στους εσωτερικούς καταχωρητές των κελιών. Τονίζεται ότι οι λέξεις είναι ίδιες για όλα τα κελιά του πίνακα, αλλά τα δεδομένα από τη μνήμη δεδομένων πρέπει να σταλούν ανά στήλη, ώστε να γίνει σωστή κατανομή των συντελεστών στα κελιά.

- *0-00-0-00000-0110-0000-011-0-0-00000000-0*
  Με τη λέξη αυτή γίνεται η φόρτωση του πρώτου set συντελεστών βαρύτητας στα κελιά. Πρόκειται για τους συντελεστές h15, h14, h13 και h12. Στην ουσία η λέξη ορίζει ότι ο καταχωρητής εισόδου που βρίσκεται στην είσοδο 1 της ALU επιλέγει την είσοδο από τη μνήμη δεδομένων. Επιπλέον η λειτουργία της ALU τίθεται στην κατάσταση στην οποία προωθεί απλώς τα δεδομένα στην είσοδο 1 μέχρι την έξοδο, χωρίς τροποποίηση. Εν τέλει, τα δεδομένα από τη μνήμη διέρχονται από όλες τις μονάδες του κελιού και φτάνουν μέχρι την έξοδο, όπου και εγγράφονται στον ενδιάμεσο καταχωρητή (εξόδου).

- *1-11-0-00000-0000-0000-000-0-0-00000000-0*
  Η συγκεκριμένη λέξη διαμόρφωσης ορίζει ότι τα δεδομένα στον καταχωρητή εξόδου πρέπει να γραφούν στο τοπικό αρχείο καταχωρητών και μάλιστα στον καταχωρητή RF3.

- *0-00-0-00000-0110-0000-011-0-0-00000000-0*
  Ακολουθεί και πάλι κύκλος για φόρτωση δεδομένων από τη μνήμη δεδομένων μέχρι τον καταχωρητή εξόδου. Πρόκειται για την ίδια λέξη όπως στην περίπτωση 1.

- *1-11-0-00000-0000-0000-000-0-0-00000000-0*
  Με τη λέξη αυτή γίνεται η αποθήκευση των δεδομένων που έχουν αποθηκευτεί στον καταχωρητή εξόδου κατά τον προηγούμενο κύκλο στο αρχείο καταχωρητών των κελιών. Ο καταχωρητής που ορίζεται να εγγραφεί είναι ο RF2.

- *0-00-0-00000-0110-0000-011-0-0-00000000-0*
  Ακολουθεί και πάλι κύκλος για φόρτωση δεδομένων από τη μνήμη δεδομένων μέχρι τον καταχωρητή εξόδου. Πρόκειται για την ίδια λέξη όπως στις περιπτώσεις 1 και 3.

- *1-01-0-00000-0000-0000-000-0-0-00000000-0*
  Σε αυτό τον κύκλο διαμόρφωσης γίνεται η αποθήκευση των συντελεστών που φορτώθηκαν στον καταχωρητή εξόδου κατά τον προηγούμενο κύκλο, στον καταχωρητή RF1 του τοπικού αρχείου καταχωρητών των κελιών.

- *0-00-0-00000-0110-0000-011-0-0-00000000-0*

Με τη λέξη αυτή γίνεται πάλι φόρτωση δεδομένων από τη μνήμη δεδομένων (δηλαδή νέου συνόλου συντελεστών βαρύτητας) στον καταχωρητή εξόδου των κελιών.

- *1-00-0-00000-0000-0000-000-0-0-00000000-0*
  Η λέξη αυτή καθορίζει να γίνει η αποθήκευση του τελευταίου συνόλου συντελεστών που φορτώθηκε, των δεδομένων δηλαδή που βρίσκονται στους καταχωρητές εξόδου των κελιών, στον καταχωρητή RF0 του τοπικού αρχείου καταχωρητών.

6.2.3.4. Περίοδος Λειτουργίας

Όπως και στην περίπτωση της απεικόνισης του φίλτρου 4-tap, έτσι και στην παρούσα απεικόνιση η λειτουργία των κελιών του πίνακα οργανώνεται σε κύκλους λειτουργίας, που αποτελούνται από πολλαπλούς κύκλους ρολογιού (κύκλους διαμόρφωσης). Διαμορφώνεται με τον τρόπο αυτό μία περίοδο λειτουργίας για την εξαγωγή ενός δείγματος εξόδου, που επαναλαμβάνεται μέχρι να προκύψει ο αριθμός δειγμάτων εξόδου που επιθυμείται. Οι λειτουργίες που πρέπει να εκτελεστούν κατά τη διάρκεια μιας περιόδου λειτουργίας σε κάθε κελί είναι συγκεκριμένες και είναι παρόμοιες με αυτές που αντιστοιχούν στην απεικόνιση του φίλτρου 4-tap.

Ειδικότερα, κάθε κελί πρέπει να:

- Εκτελέσει τον πολλαπλασιασμό μεταξύ ενός δείγματος εισόδου και του συντελεστή βαρύτητας που έχει αποθηκευμένο στο τοπικό αρχείο καταχωρητών
- Να αποθηκεύσει το παραπάνω γινόμενο σε έναν καταχωρητή
- Να μεταφέρει στην έξοδό του το τελικό αποτέλεσμα που είχε προκύψει κατά τον προηγούμενο κύκλο λειτουργίας
- Να αθροίσει το μερικό γινόμενο του τρέχοντος κύκλου με το τελικό αποτέλεσμα που προέκυψε στο προηγούμενο κελί στη νοητή σειρά λειτουργίας, μετά τον προηγούμενο κύκλο λειτουργίας (αυτό παρέχεται στην έξοδο του προηγούμενου κελιού, μέσω της προηγούμενης λειτουργίας των κελιών) και
- Τέλος να αποθηκεύσει το παραπάνω άθροισμα (το τελικό αποτέλεσμα του τρέχοντος κύκλου) σε κάποιον τοπικό καταχωρητή.

Πριν παρουσιαστούν οι παραπάνω λειτουργίες σε συμβολική γλώσσα, ανά κύκλο διαμόρφωσης του πίνακα, πρέπει να σημειωθεί μια ιδιαιτερότητα της απεικόνισης και ο τρόπος που αντιμετωπίστηκε. Το γεγονός ότι ανά στήλη, οι συντελεστές βαρύτητας του φίλτρου βρίσκονται αποθηκευμένοι σε διαφορετικό καταχωρητή, αποτρέπει την εύρεση μιας γενικής λύσης. Κατά τη διάρκεια μιας περιόδου λειτουργίας πρέπει να γίνουν ορισμένες εγγραφές σε καταχωρητές του αρχείου καταχωρητών, οι οποίες δεν πρέπει να επηρεάσουν τους

αποθηκευμένους συντελεστές βαρύτητας. Για να αντιμετωπιστεί η συγκεκριμένη κατάσταση επιλέχθηκε ο πίνακας να λειτουργεί ανά στήλες, δηλαδή οι λέξεις διαμόρφωσης να μεταδίδονται ανά στήλες στον πίνακα. Λαμβάνοντας, επιπλέον, υπόψη το γεγονός ότι με τον τρόπο που έχουν φορτωθεί οι συντελεστές, αυτοί βρίσκονται στον ίδιο καταχωρητή σε κελιά της ίδιας στήλης, η λειτουργία των καταχωρητών του αρχείου καταχωρητών μπορεί να ομαδοποιηθεί ανά στήλες. Δηλαδή, τα κελιά που βρίσκονται σε διαφορετικές στήλες έχουν διαφορετική λειτουργία, όχι όμως ως προς τη βασική ιδέα που είναι η ίδια, αλλά ως προς το ποιοι καταχωρητές χρησιμοποιούνται.

Ο Πίνακας 6.2 δείχνει το πως χρησιμοποιούνται οι τέσσερις καταχωρητές του αρχείου καταχωρητών των κελιών, ανάλογα με τη στήλη.

Πίνακας 6.2: Ομαδοποιημένη χρήση των καταχωρητών των κελιών ανά στήλη του πίνακα, κατά την απεικόνιση του FIR φίλτρου 16-tap

Στήλες	Καταχωρητής	Περιεχόμενο
Στήλη 1	RF0	Είσοδος x Συντ. βαρύτητας
	RF1	Αποτέλεσμα κύκλου
	RF2	-
	RF3	Συντελεστής βαρύτητας
Στήλη 2	RF0	Αποτέλεσμα κύκλου
	RF1	-
	RF2	Συντελεστής βαρύτητας
	RF3	Είσοδος x Συντ. βαρύτητας
Στήλη 3	RF0	-
	RF1	Συντελεστής βαρύτητας
	RF2	Είσοδος x Συντ. βαρύτητας
	RF3	Αποτέλεσμα κύκλου
Στήλη 4	RF0	Συντελεστής βαρύτητας
	RF1	Είσοδος x Συντ. βαρύτητας
	RF2	Αποτέλεσμα κύκλου
	RF3	-

6.2.3.5. Ψευδοκώδικας – Περιεχόμενο Διαμόρφωσης

Στη συνέχεια παρατίθενται οι τέσσερις διαφορετικές παραλλαγές του ψευδοκώδικα που περιγράφουν τη λειτουργία των κελιών, μία για καθεμιά από τις στήλης του επαναδιατάξιμου πίνακα.

Στήλη 1 :

- MEM_DATA x RF3 = $ALU_{out\ 1}$
- RF0 <- $ALU_{out\ 1}$ / OUTPUT <- RF1
- RF0 + $OUT_{up\ cell}$ = $ALU_{out\ 2}$
- RF1 <- $ALU_{out\ 2}$

Στήλη 2 :

- MEM_DATA x RF3 = $ALU_{out\ 1}$
- RF3 <- $ALU_{out\ 1}$ / OUTPUT <- RF0
- RF3 + $OUT_{up\ cell}$ = $ALU_{out\ 2}$
- RF0 <- $ALU_{out\ 2}$

Στήλη 3 :

- MEM_DATA x RF1 = $ALU_{out\ 1}$
- RF2 <- $ALU_{out\ 1}$ / OUTPUT <- RF3
- RF2 + $OUT_{up\ cell}$ = $ALU_{out\ 2}$
- RF3 <- $ALU_{out\ 2}$

Στήλη 4 :

- MEM_DATA x RF0 = $ALU_{out\ 1}$
- RF1 <- $ALU_{out\ 1}$ / OUTPUT <- RF2
- RF1 + $OUT_{up\ cell}$ = $ALU_{out\ 2}$
- RF2 <- $ALU_{out\ 2}$

Όπως αναφέρθηκε, οι παραπάνω περίοδοι λειτουργίας αντιστοιχούν σε παραπάνω του ενός κύκλους διαμόρφωσης, συνολικά 4 για κάθε περίοδο. Ακολούθως θα παρουσιαστούν οι λέξεις διαμόρφωσης που στέλνονται στα κελιά, σε κάθε κύκλο διαμόρφωσης, κατά τη διάρκεια μιας περιόδου λειτουργίας. Δύο είναι τα σημεία που πρέπει να τονιστούν. Αφενός οι λέξεις διαμόρφωσης μεταφέρονται στον πίνακα ανά στήλες (κελιά στην ίδια στήλη διαμορφώνονται από την ίδια λέξη), καθώς αυτό υπαγορεύει η διάταξη των συντελεστών βαρύτητας στους εσωτερικούς καταχωρητές των κελιών. Αφετέρου οι λέξεις διαφέρουν για κάθε στήλη, καθώς είναι διαφορετικοί οι καταχωρητές που πρέπει να χρησιμοποιηθούν ανά περίπτωση, σύμφωνα με όσα έχουν αναφερθεί προηγουμένως. Οι λέξεις που παρουσιάζονται παρακάτω αντιστοιχούν -με τη σειρά που εμφανίζονται- στις 4 στήλες του πίνακα, από τα αριστερά προς τα δεξιά.

- *0-00-0-00000-1010-0110-000-1-0-00000000-0*
  0-00-0-00000-1001-0110-000-1-0-00000000-0
  0-00-0-00000-1000-0110-000-1-0-00000000-0
  0-00-0-00000-0111-0110-000-1-0-00000000-0
  Οι λέξεις αυτές διαμορφώνουν τα κελιά ώστε να πραγματοποιηθεί ο

πολλαπλασιασμός του νέου δείγματος εισόδου που φορτώνεται και του αντίστοιχου συντελεστή που είναι αποθηκευμένος στο κελί. Συγκεκριμένα ενεργοποιείται να λειτουργήσει το κομμάτι του πολλαπλασιαστή από τη μονάδα ALU-MUL και να δεχτεί σαν εισόδους του, μέσω κατάλληλης επιλογής στους πολυπλέκτες εισόδου, ένα δεδομένο από τη μνήμη δεδομένων και το περιεχόμενο του κατάλληλου καταχωρητή που περιέχει το συντελεστή του κελιού. Το ποιος καταχωρητής επιλέγεται καθορίζεται από τη στήλη για την οποία προορίζεται η λέξη, σύμφωνα με όσα έχουν αναφερθεί παραπάνω σχετικά με το θέμα αυτό

- *1-00-000000-1-00000000-0-00000000-0-0-0-1*
  1-11-000000-1-00000000-0-00000000-0-0-0-1
  1-10-000000-1-00000000-0-00000000-0-0-0-1
  1-01-000000-1-00000000-0-00000000-0-0-0-1
  Με τις λέξεις αυτές ρυθμίζεται η προσωρινή παύση λειτουργίας της μνήμης δεδομένων, όλα τα κελιά τίθενται προσωρινά σε κατάσταση stall και τέλος αποθηκεύεται το αποτέλεσμα του πολλαπλασιασμού του προηγούμενου κύκλου διαμόρφωσης στον κατάλληλο καταχωρητή του αρχείου καταχωρητών. Ο καταχωρητής αυτός εξαρτάται άμεσα από τη στήλη στην οποία αναφέρεται η λέξη διαμόρφωσης, οπότε η τελευταία καθορίζει διαφορετικό καταχωρητή για κάθε στήλη.

- *0-00-0-00000-1000-0000-011-0-0-00000000-0*
  0-00-0-00000-0111-0000-011-0-0-00000000-0
  0-00-0-00000-1010-0000-011-0-0-00000000-0
  0-00-0-00000-1001-0000-011-0-0-00000000-0
  Η λέξη αυτή καθορίζει ότι πρέπει να φορτωθεί ο καταχωρητής εξόδου κάθε κελιού με το αποτέλεσμα της προηγούμενης περιόδου λειτουργίας κάθε κελιού. Το αποτέλεσμα αυτό είναι αποθηκευμένο στον κατάλληλο καταχωρητή του αρχείου καταχωρητών, ο οποίος διαφέρει από στήλη σε στήλη. Στην ουσία η λέξη καθορίζει ότι ο πολυπλέκτης στην είσοδο a της ALU επιλέγει την είσοδο δεδομένων που προέρχεται από το συγκεκριμένο καταχωρητή και επίσης ότι η μονάδα ALU/MUL προωθεί τα δεδομένα αυτά προς την έξοδο χωρίς τροποποίηση.

- *0-00-0-00000-0111-0000-010-0-0-00000000-0*
  0-00-0-00000-1010-0000-010-0-0-00000000-0
  0-00-0-00000-1001-0000-010-0-0-00000000-0
  0-00-0-00000-1000-0000-010-0-0-00000000-0
  Με τη λέξη αυτή γίνεται η άθροιση του γινομένου που υπολογίστηκε στην αρχή της τρέχουσας περιόδου λειτουργίας (και βρίσκεται αποθηκευμένο σε κάποιον τοπικό καταχωρητή του κελιού, ανάλογα με τη στήλη) με το αποτέλεσμα της προηγούμενης περιόδου λειτουργίας του προηγούμενου κελιού στη γραμμή λειτουργίας του πίνακα. Η λέξη λοιπόν καθορίζει ότι από τη μονάδα ALU/MUL θα επιλεγεί η έξοδος από την ALU, η οποία ρυθμίζεται να εκτελέσει την πράξη της πρόσθεσης. Επιπλέον ρυθμίζονται ποιες είσοδοι δεδομένων θα επιλέξουν οι πολυπλέκτες εισόδου στις μονάδες ALU/MUL των κελιών και ειδικότερα αυτές θα είναι ένας τοπικός καταχωρητής (ανάλογα με τη στήλη) και η έξοδος του προηγούμενου κελιού.

- *1-01-0-00000-0000-0000-000-0-0-00000000-0*
  1-00-0-00000-0000-0000-000-0-0-00000000-0
  1-11-0-00000-0000-0000-000-0-0-00000000-0
  1-10-0-00000-0000-0000-000-0-0-00000000-0
  Τέλος αυτή η λέξη καθορίζει την αποθήκευση των γινομένων του προηγούμενου κύκλου διαμόρφωσης στον κατάλληλο καταχωρητή που πρέπει να χρησιμοποιηθεί για το σκοπό αυτό (ανάλογα πάντα με τη στήλη).

Οι παραπάνω λέξεις διαμόρφωσης αποτελούν μια περίοδο λειτουργίας για τα κελιά και όπως είναι κατανοητό πρέπει να επαναλαμβάνονται οι ίδιες συνέχεια, μέχρι να προκύψει το πλήθος των δειγμάτων εξόδου που είναι επιθυμητό. Αυτό επιτυγχάνεται με τη χρήση της δυνατότητας των εσωτερικών βρόχων στη μνήμη διαμόρφωσης, κάτι που χρησιμοποιήθηκε και στην περίπτωση της απεικόνισης του φίλτρου FIR 4-tap. Δηλαδή, αφού τοποθετηθούν οι παραπάνω 5 λέξεις στη μνήμη διαμόρφωσης, στη συνέχεια τοποθετείται επίσης στην επόμενη θέση μνήμης η παρακάτω ειδική λέξη

0-00-000000-0-00000000-0-00001100-1-0-0-1

Αυτή, όπως και στην περίπτωση του φίλτρου 4-tap ρυθμίζει τις εσωτερικές επαναλήψεις που πρέπει να γίνουν στη μνήμη διαμόρφωσης. Συγκεκριμένα, καθορίζει ότι όταν η εκτέλεση φτάσει σε αυτή τη θέση μνήμης, τότε πρέπει να γίνει άλμα επανάληψης και η επόμενη θέση μνήμης που θα εκτελεστεί θα είναι και πάλι αυτή που περιέχει το πρώτο σύνολο λέξεων διαμόρφωσης στην περίοδο λειτουργίας του κελιού. Σύμφωνα λοιπόν με τη λέξη αυτή, τίθεται η κατάσταση λειτουργίας του συστήματος στην κατάσταση stall, το bit 3 τίθεται στο '1' για να δηλωθεί ότι πρέπει να γίνει άλμα και επιπλέον παρέχεται και η διεύθυνση για το άλμα, που είναι η θέση μνήμης #12. Σύμφωνα και με όσα έχουν αναφερθεί κατά την περιγραφή του συστήματος των επαναλήψεων, πριν γίνει το άλμα θα πρέπει πρώτα να γίνει έλεγχος αν ο καταχωρητής επαναλήψεων έχει μηδενιστεί ή όχι.

Παρακάτω παρατίθεται το περιεχόμενο της μνήμης διαμόρφωσης που αντιστοιχεί στην απεικόνιση του φίλτρου 16-tap που περιγράφεται στην παρούσα ενότητα. Η δομή της μνήμης δεν είναι ακριβώς όπως φαίνεται παρακάτω, απλώς παρουσιάζονται οι λέξεις διαμόρφωσης που θα τοποθετηθούν.

6.2.3.6. Έλεγχος Επιδόσεων

Για τον έλεγχο των επιδόσεων και της αποδοτικότητας της συγκεκριμένης τεχνικής απεικόνισης που προτείνεται, πραγματοποιήθηκε συγκριτική λειτουργία μεταξύ αυτής και μιας απεικόνισης στο σύστημα MicroSoc που βασίζεται αποκλειστικά στον ARM και δεν περιλαμβάνει τη χρησιμοποίηση της επαναδιατάξιμης μονάδας. Όπως και στην περίπτωση του φίλτρου 4-tap δημιουργήθηκε ένας κώδικας σε γλώσσα C, ο οποίος περιγράφει πλήρως την

υλοποίηση ενός φίλτρου FIR 16-tap. Ο κώδικας αυτός φορτώθηκε στη συνέχεια στη μνήμη RAM του MicroSoc και εκτελέστηκε από τον ARM, με αποτέλεσμα να παραχθούν στη μνήμη ROM τα αναμενόμενα αποτελέσματα, δηλαδή οι τιμές των επιθυμητών δειγμάτων εισόδου. Αυτό που ενδιαφέρει είναι να επιβεβαιωθεί η ορθότητα της απεικόνισης στην επαναδιατάξιμη λογική αλλά και να συγκριθούν οι χρόνοι που απαιτούνται στις δύο περιπτώσεις, από την έναρξη λειτουργίας μέχρι να παραχθούν τα τελικά αποτελέσματα.

Πριν αναλυθεί ο τρόπος υλοποίησης που βασίζεται αποκλειστικά στο software, θα γίνει αναφορά στα δείγματα είσοδο που χρησιμοποιήθηκαν και για τις δύο περιπτώσεις των μεθόδων υλοποίησης. Τα δείγματα αυτά είναι παρόμοια με τα δείγματα που χρησιμοποιήθηκαν και στην περίπτωση του φίλτρου 4-tap. Ειδικότερα προέρχονται και αυτά από την ίδια γραφική παράσταση, από τη γραφική παράσταση δηλαδή μιας κανονικής κατανομής με μέγιστο την τιμή 16. Ωστόσο, τα δείγματα που ελήφθησαν είναι πιο πυκνά, καθώς έπρεπε να χρησιμοποιηθούν συνολικά 32 σε αριθμό. Οπότε, οι τιμές με τις οποίες έπρεπε να τροφοδοτηθεί το φίλτρο είναι οι εξής:

$$[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]$$

Σύμφωνα όμως με τη θεωρία που παρουσιάσαμε στην προηγούμενη ενότητα, σχετικά με την απεικόνιση FIR φίλτρων, εκτός από τις παραπάνω εισόδους θα πρέπει στην αρχή του διανύσματος εισόδου του φίλτρου να τοποθετηθούν επιπλέον μηδενικά δείγματα. Τα μηδενικά αυτά δείγματα πρέπει να είναι συνολικά 15, δηλαδή ίσα με (N-1), όπου N=16 η τάξη του φίλτρου. Συνολικά λοιπόν το διάνυσμα εισόδου με το οποίο υποθέτουμε ότι τροφοδοτείται το φίλτρο που υλοποιούμε είναι το ακόλουθο:

$$[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1]$$

Να σημειωθεί το παραπάνω διάνυσμα εισόδου χρησιμοποιήθηκε στην υλοποίηση του φίλτρου χωρίς το επαναδιατάξιμο περιφερειακό. Για την απεικόνιση της μεθόδου στο υλικό, χρησιμοποιήθηκαν απευθείας οι τιμές των δειγμάτων εισόδου, χωρίς την προσθήκη των επιπλέον μηδενικών στην αρχή του διανύσματος. Ο λόγος εντοπίζεται στην παραλληλία της υλοποίησης και στη μέθοδο της κοινής χρήσης των δεδομένων από τη μνήμη, τεχνικές που εφαρμόστηκαν στην απεικόνιση.

Ακόμα πρέπει να αναφερθεί ότι ως συντελεστές βαρύτητας για το φίλτρο χρησιμοποιήθηκαν ενδεικτικά οι τιμές 1,2,3,...,14,15,16.

Παρακάτω (Πλαίσιο 6.3) φαίνεται ο κώδικας σε γλώσσα C, ο οποίος υλοποιεί ένα φίλτρο 16-tap και ο οποίος φορτώθηκε στο MicroSoc για να εκτελεστεί από τον ARM επεξεργαστή του.

```
#define NTAPS 16
#define INP_SIZE (3 * NTAPS - 1)

static const int w[NTAPS] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };

static const int x[INP_SIZE] =
 { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,
16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1 };

extern void C_Entry(void);

#pragma arm section rwdata= "pinakas"
int y[32];
#pragma arm section rwdata

#pragma arm section code

void C_Entry(void)

{
    int i,j;

    for(i=0; i<32; i++) {
          y[i] = 0;
    }

    for(j=0; j<32; j++) {
          int accum = 0;

          for(i = 0; i < NTAPS; i++) {
                accum += w[i] * x[j-i+15];
          }

          y[j] = accum;
    }
}

#pragma arm section code
```

Πλαίσιο 6.3: Κώδικας σε γλώσσα C που υλοποιεί φίλτρο 16-tap

Η μορφή του παραπάνω κώδικα είναι ακριβώς ίδια με τον κώδικα που χρησιμοποιήθηκε στην περίπτωση του φίλτρου 4-tap. Για το λόγο αυτό γίνεται παραπομπή στην αντίστοιχη ενότητα 6.2.2.5. για περαιτέρω ανάλυση. Η μόνη διαφοροποίηση που υπάρχει προέρχεται από το μέγεθος του φίλτρου και από το μέγεθος του τελικού διανύσματος δειγμάτων εξόδου που επιθυμείται. Εξαιτίας αυτών των στοιχείων, ο χώρος μνήμης που δεσμεύεται στη μνήμη RAM για αποθήκευση των τιμών των δειγμάτων εξόδου είναι συνολικά 32 στοιχεία μνήμης. Επιπλέον, οι βρόχοι επανάληψης, κάθε ένας από τους οποίους υπολογίζει και ένα διαφορετικό δείγμα εξόδου, είναι περισσότεροι και συγκεκριμένα ίσοι με 32. Τέλος, και ο εσωτερικός βρόχος, ο οποίος είναι φωλιασμένος μέσα στον κύριο βρόχο του προγράμματος και στον οποίο γίνεται ο πολλαπλασιασμός των εισόδων με τους συντελεστές βαρύτητας, έχει συνολικά 16 επαναλήψεις καθώς τόση είναι και η τάξη του φίλτρου.

6.2.3.7. Συγκριτικά Αποτελέσματα

Αρχικά έγινε η υλοποίηση του φίλτρου χωρίς το επαναδιατάξιμο περιφερειακό, οπότε φορτώθηκε ο αντίστοιχος κώδικας στη μνήμη ROM του συστήματος. Μετά την εκτέλεση της εφαρμογής, τα τελικά αποτελέσματα -δηλαδή το διάνυσμα εξόδου από το φίλτρο- είναι αποθηκευμένα στη μνήμη RAM.

Η εικόνα της μνήμης μετά το τέλος της εκτέλεσης φαίνεται παρακάτω (Πλαίσιο 6.4).

0:	*1*	*4*	*10*	*20*	*35*	*56*
6:	*84*	*120*	*165*	*220*	*286*	*364*
c:	*455*	*560*	*680*	*816*	*951*	*1083*
12:	*1210*	*1330*	*1441*	*1541*	*1628*	*1700*
18:	*1755*	*1791*	*1806*	*1798*	*1765*	*1705*
1e:	*1616*	*1496*	*x*	*x*	*x*	*x*
...	*...*	*...*	*...*	*...*	*...*	*...*

Πλαίσιο 6.4: Περιεχόμενο μνήμης RAM μετά τους υπολογισμούς για το 16-tap φίλτρο

Το διάνυσμα εξόδου προέκυψε, συνεπώς, ίσο με :

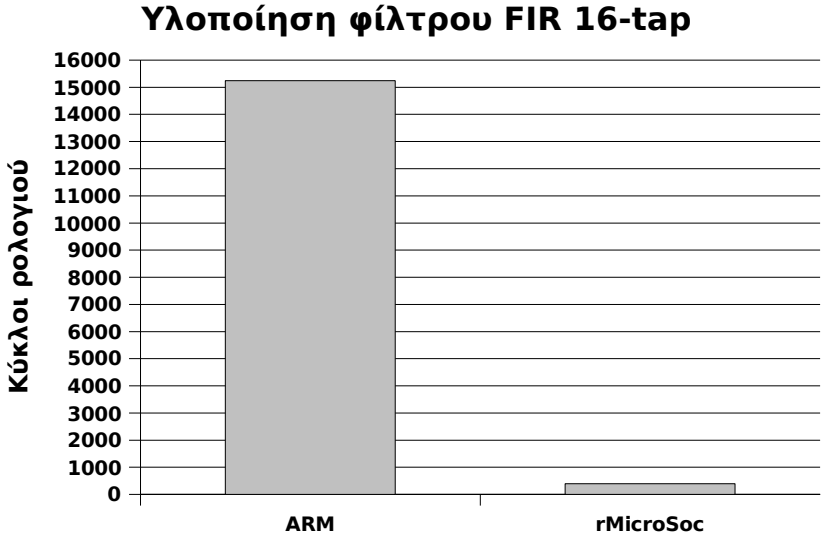
$$\begin{Bmatrix} 1,4,10,20,35,56,84,120,165,220,286,364,455,560,680,816,951,1083,1210, \\ 1330,1441,1541,1628,1700,1755,1791,1806,1798,1765,1705,1616,1496 \end{Bmatrix}$$

Οι κύκλοι που απαιτήθηκαν για να υπολογιστούν τα παραπάνω αποτελέσματα και να αποθηκευτούν στη μνήμη ήταν συνολικά 15244.

Έπειτα προετοιμάστηκε το rMicroSoc κατάλληλα για να υλοποιήσει το φίλτρο. Αφού φορτώθηκαν τα δεδομένα προς επεξεργασία και τα δεδομένα διαμόρφωσης στο επαναδιατάξιμο περιφερειακό, εκτελέστηκε η εφαρμογή. Για την εξαγωγή των αντίστοιχων δειγμάτων εξόδου με την περίπτωση της απεικόνισης χωρίς το επαναδιατάξιμο περιφερειακό και την ολοκλήρωση του κώδικα εκτέλεσης, απαιτήθηκαν κύκλοι ίσοι με 388.

Στο Διάγραμμα 6.2 απεικονίζεται η σύγκριση της απόδοσης των δύο υλοποιήσεων, με βάση το τον αριθμό των απαιτούμενων κύκλων ρολογιού.

Από τους χρόνους αυτούς που χρειάστηκαν για να εξαχθούν τα αποτελέσματα, διαπιστώνεται ότι η υλοποίηση του φίλτρου FIR με τη χρήση του επαναδιατάξιμου περιφερειακού επιτάχυνε τους υπολογισμούς κατά περίπου 97%.



Διάγραμμα 6.2: Σύγκριση της υλοποίησης φίλτρου FIR 16-tap στο rMicroSoc και στον ARM με βάση τον αριθμό των κύκλων ρολογιού

Συγκρίνοντας το ποσοστό επιτάχυνσης που επιτυγχάνεται από το επαναδιατάξιμο περιφερειακό κατά την υλοποίηση του φίλτρου 16-tap με το αντίστοιχο ποσοστό στην υλοποίηση του φίλτρου 4-tap (αναλυτικά στην προηγούμενη ενότητα) προκύπτει ένα συμπέρασμα. Η επιτάχυνση και η βελτίωση της απόδοσης φαίνεται να είναι μεγαλύτερη με τη χρήση της τεχνικής που εφαρμόστηκε στο φίλτρο 16-tap. Ωστόσο, μια ασφαλέστερη -ίσως- σύγκριση θα μπορούσε να γίνει αν γινόταν η υλοποίηση του φίλτρου 4-tap με βάση την τεχνική που χρησιμοποιήθηκε για το φίλτρο 16-tap. Στο Διάγραμμα 6.3 φαίνεται παραστατικά η διαφορά στην αποδοτικότητα των δύο μεθόδων υλοποίησης φίλτρων FIR.

Διάγραμμα 6.3: Σύγκριση των δύο τεχνικών απεικόνισης φίλτρων FIR

6.2.3.8. Πλεονεκτήματα-Μειονεκτήματα Απεικόνισης στο Υλικό

Ανακεφαλαιώνοντας, θα αναφερθούν κάποια βασικά στοιχεία της απεικόνισης που προτείνεται στην παρούσα ενότητα και θα τονιστούν πλεονεκτήματα και μειονεκτήματα που διακρίνονται.

Στην ενότητα αυτή προτείνεται μια μέθοδος απεικόνισης για φίλτρα FIR με εξειδίκευση στο επαναδιατάξιμο σύστημα MicroSoc. Κάποια βασικά σημεία της συγκεκριμένης απεικόνισης είναι τα εξής:

- Μπορεί να εφαρμοστεί για την υλοποίηση φίλτρων με τάξη μεγέθους ίση, μεγαλύτερη ή και μικρότερη από το μέγεθος του πίνακα.

- Τα κελιά του επαναδιατάξιμου πίνακα ρυθμίζονται ώστε η ανταλλαγή δεδομένων μεταξύ τους να τα τοποθετεί σε μια νοητή ευθεία γραμμή. Το μέγεθος της γραμμής αυτής ισούται με την τάξη του φίλτρου που απεικονίζεται.

- Η κύρια λειτουργία που επιτελεί κάθε κελί είναι τριπλή : α)να πολλαπλασιάζει το δείγμα εισόδου που δέχεται από τη μνήμη δεδομένων με το συντελεστή βαρύτητας που έχει αντιστοιχιστεί με το κάθε κελί, β)να προσθέτει το παραπάνω γινόμενο με δεδομένα που προέρχονται από το προηγούμενο κελί στη νοητή γραμμή που δημιουργείται στον επαναδιατάξιμο πίνακα και γ)να παρέχει -στην κατάλληλη χρονική στιγμή- στο επόμενο -νοητά- κελί του πίνακα το τελικό αποτέλεσμα -άθροισμα- που υπολογίζει σε κάθε κύκλο. Πρέπει να σημειωθεί ότι σε κάθε κελί πρέπει να έχει αντιστοιχηθεί και διαφορετικός συντελεστής βαρύτητας.

- Τα αποτελέσματα εξόδου λαμβάνονται από το τελευταίο κελί στη νοητή γραμμή στην οποία οργανώνονται τα επαναδιατάξιμα κελιά του πίνακα, με throughput ένα δείγμα ανά περίοδο λειτουργίας, εκτός από ένα διάστημα αρχικοποίησης κατά τη διάρκεια του οποίου δεν παράγονται δεδομένα.

- Η βασικότερη ιδέα που εφαρμόζεται είναι αυτή της *Memory Operation (Bandwidth) Sharing*. Σύμφωνα με αυτήν, σε περιπτώσεις απεικόνισης σε υλικό αλγορίθμων που περιλαμβάνουν βρόχους επανάληψης, η κοινή χρήση στις μονάδες επεξεργασίας και ανά επανάληψη δεδομένων από τη μνήμη βελτιώνει την ταχύτητα και την αποτελεσματικότητα της υλοποίησης.

- Ακολουθώντας την παραπάνω ιδέα, τα δείγματα εισόδου του φίλτρου -τα οποία έρχονται από τη μνήμη δεδομένων του συστήματος- παρέχονται τα ίδια και ταυτόχρονα σε όλα τα κελιά του πίνακα. Η τροφοδότηση γίνεται σταδιακά, ξεκινώντας από το πρώτο δείγμα του διανύσματος εισόδου (το οποίο κατά τα γνωστά περιέχει και ορισμένα μηδενικά δείγματα στην αρχή του για λόγους ομοιομορφίας και παραλληλίας).

Τα πλεονεκτήματα της βελτίωσης της αποδοτικότητας (όσον αφορά στην χρονική διάρκεια εκτέλεσης) και της παραλληλίας που παρατηρήθηκαν και παρουσιάστηκαν και στην περίπτωση της απεικόνισης του φίλτρου 4-tap, ισχύουν και για την παρούσα απεικόνιση. Αυτά αφορούν στη σύγκριση μεταξύ υλοποιήσεων με χρήση και μη χρήση του επαναδιατάξιμου υλικού.

Ωστόσο, υπάρχουν -επιπλέον- σημαντικά πλεονεκτήματα της παρούσας απεικόνισης σε σχέση με την προηγούμενη τεχνική απεικόνισης που παρουσιάστηκε. Συγκεκριμένα, το κύριο πλεονέκτημα που εμφανίζεται είναι ότι με την παρούσα απεικόνιση γίνεται δυνατή η υλοποίησης φίλτρων με τάξη ακόμα και μεγαλύτερη από το μέγεθος του πίνακα. Για το λόγο αυτό η συγκεκριμένη απεικόνιση είναι περισσότερο ευέλικτη από την άλλη, καθώς επιτρέπει την υλοποίηση φίλτρων μεγαλύτερης τάξης. Υπάρχει, όμως, ο περιορισμός ότι το φίλτρο με τη μέγιστη τάξη που μπορεί να απεικονιστεί είναι αυτό με τάξη ίση με το συνολικό πλήθος των κελιών του επαναδιατάξιμου πίνακα. Με την παρούσα μορφή του, το επαναδιατάξιμο MicroSoc μπορεί να απεικονίσει μέχρι και φίλτρα 16-tap, όσο είναι δηλαδή και το φίλτρο το οποίο υλοποιήθηκε σαν παράδειγμα εφαρμογής της μεθόδου απεικόνισης που προτείνεται.

Τέλος, διακρίνεται μια υπεροχή της μεθόδου που χρησιμοποιήθηκε στην παρούσα απεικόνιση στον τομέα της αποδοτικότητας. Αυτό συμπεραίνεται διότι η επιτάχυνση του χρόνου εκτέλεσης για το φίλτρο 16-tap (σε σχέση με την υλοποίηση χωρίς επαναδιάταξη) είναι μεγαλύτερη από την αντίστοιχη που επιτυγχάνεται με την τεχνική που εφαρμόστηκε για το φίλτρο 4-tap.

6.3.Ψηφιακός Συνημιτονικός Μετασχηματισμός

6.3.1.Εισαγωγή

Η κωδικοποίηση μετασχηματισμών αποτελεί ένα συστατικό και σημαντικό κομμάτι των σύγχρονων εφαρμογών επεξεργασίας εικόνας/βίντεο. Οι μετασχηματισμοί, γενικά, βασίζονται στην ιδέα ότι τα pixels σε μια στατική εικόνα εμπεριέχουν έναν ορισμένο βαθμό συσχέτισης με τα γειτονικά pixels. Παρομοίως σε ένα σύστημα μετάδοσης βίντεο, τα γειτονικά pixels σε δύο συνεχόμενα πλαίσια εμφανίζουν πολύ μεγάλη συσχέτιση. Συνεπώς, ένα σύστημα κωδικοποίησης μπορεί να εκμεταλλευτεί αυτούς τους συσχετισμούς για να προβλέψει την τιμή ενός pixel από τους αντίστοιχους γείτονες. Ένας μετασχηματισμός έχει σχεδιαστεί ώστε να απεικονίζει αυτά τα χωρικά συσχετισμένα δεδομένα σε μετασχηματισμένους, μη συσχετισμένους συντελεστές. Σαφώς, ο μετασχηματισμός θα πρέπει να εκμεταλλεύεται το γεγονός ότι το περιεχόμενο πληροφορίας ενός μεμονωμένου pixel είναι σχετικά μικρό, για παράδειγμα η οπτική συνεισφορά ενός pixel μπορεί σε ένα μεγάλο βαθμό να προβλεφτεί χρησιμοποιώντας τους γείτονές του.

Το Σχήμα 6.20 περιγράφει ένα τυπικό σύστημα μετάδοσης εικόνας/βίντεο. Ο σκοπός του κωδικοποιητή πηγής είναι να εκμεταλλευτεί τις εξαρτήσεις που υπάρχουν στα δεδομένα της εικόνας για να παρέχει συμπίεση. Με άλλα λόγια, ο κωδικοποιητής πηγής μειώνει την εντροπία, το οποίο στη συγκεκριμένη περίπτωση σημαίνει μείωση του μέσου αριθμού bits που απαιτούνται για την αναπαράσταση της εικόνας. Αντίθετα, ο κωδικοποιητής καναλιού προσθέτει εξαρτήσεις στην έξοδο του κωδικοποιητή πηγής έτσι ώστε να βελτιώσει την αξιοπιστία της μετάδοσης. Η περαιτέρω συζήτηση θα επικεντρωθεί στη μονάδα μετασχηματισμού στον κωδικοποιητή πηγής.

Όπως αναφέρθηκε προηγουμένως, κάθε υπομονάδα μέσα στον κωδικοποιητή πηγής εκμεταλλεύεται κάποια εξάρτηση στα δεδομένα τις εικόνας ώστε να επιτευχθεί καλύτερη συμπίεση. Η υπομονάδα μετασχηματισμού αποσυσχετίζει τα δεδομένα της εικόνας, μειώνοντας -και σε μερικές περιπτώσεις εξαφανίζοντας- το συσχετισμό μεταξύ των pixel.

Η υπομονάδα κβαντοποίηση χρησιμοποιεί το γεγονός ότι το ανθρώπινο μάτι δεν είναι ικανό να αντιληφθεί ορισμένες οπτικές πληροφορίες σε μια εικόνα. Τέτοιες πληροφορίες κρίνονται ως περιττές και μπορούν να απορριφθούν χωρίς να εισάγονται αντιλήψιμα οπτικά σφάλματα. Η ιδέα αυτή μπορεί να επεκταθεί σε αποδέκτες χαμηλού ρυθμού bit, στους οποίους λόγω των χαμηλών απαιτήσεων σε εύρος ζώνης μπορεί να θυσιαστεί η οπτική ποιότητα ώστε να επιτευχθεί αποδοτικότητα εύρους ζώνης. Η βασική ιδέα πίσω από αυτό είναι η βάση για τη θεωρία παραμόρφωσης ρυθμού bit, σύμφωνα με την οποία οι παραλήπτες είναι πιθανό να ανέχονται μια κάποια οπτική παραμόρφωση με αντάλλαγμα τη διατήρηση σταθερού εύρους ζώνης.

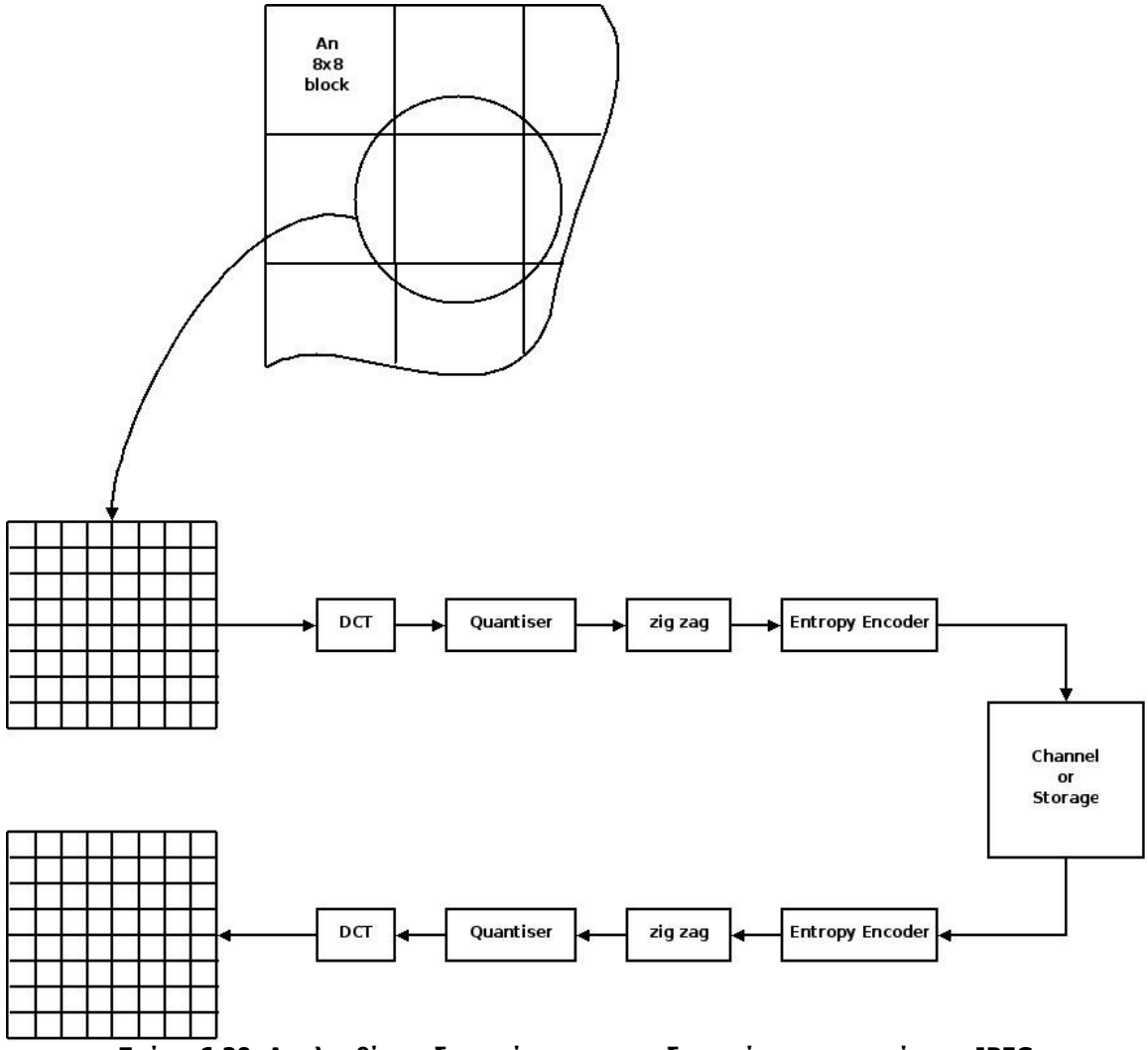
Τέλος, ο κωδικοποιητής εντροπίας εφαρμόζει τη γνώση των διαδικασιών μετασχηματισμού και κβαντοποίησης για να μειώσει τον αριθμό των bits που απαιτούνται για να απεικονίσουν κάθε σύμβολο στην έξοδο του κβαντιστή.

Λόγω των αυξημένων αναγκών της συμπίεσης βίντεο για αποθήκευση και μετάδοση τα τελευταία χρόνια, εμφανίστηκαν πολλά διεθνή πρότυπα και μάλιστα υπάρχουν διαφορετικά πρότυπα για διαφορετικές εφαρμογές. Για παράδειγμα, το πρότυπο JPEG έχει σχεδιαστεί για την κωδικοποίηση στατικών εικόνων, ενώ πρότυπα όπως τα MPEG1, MPEG2, MPEG4 σχεδιάστηκαν για την υποστήριξη πιο εξελιγμένων εφαρμογών επικοινωνίας βίντεο.

Το πρότυπο JPEG σχεδιάστηκε για πρώτη φορά το 1992 για την συμπίεση και επαναφορά στατικών φωτογραφικών εικόνων. Παρέχει ένα σχετικά καλό λόγο συμπίεσης και απαιτεί λιγότερους υπολογισμούς σε σύγκριση με το πρότυπο MPEG, το οποίο με τη σειρά του σχεδιάστηκε για μετάδοση κινούμενων εικόνων. Το βασικό συστατικό του προτύπου JPEG είναι ο Διακριτός Συνημιτονικός Μετασχηματισμός (*Discrete Cosine Transform – DCT*).

Γενικά, ο συγκεκριμένος μετασχηματισμός εμφανίζεται ως ο πλέον

χρησιμοποιούμενος μετασχηματισμός εικόνας/βίντεο στα περισσότερα συστήματα κωδικοποίησης εικόνας/βίντεο. Η μεγάλη προτίμηση για το DCT οφείλεται στο ότι παρέχει τον καλύτερο συμβιβασμό μεταξύ συμπίεσης ενέργειας και υπολογιστικής πολυπλοκότητας. Ο DCT, ως γενική ιδέα, προσπαθεί να αποσυσχετίσει τα δεδομένα της εικόνας, ώστε μετά την αποσυσχέτιση κάθε συντελεστής του μετασχηματισμού να μπορεί να κωδικοποιηθεί ανεξάρτητα χωρίς την απώλεια της αποδοτικότητας της συμπίεσης. Παρακάτω θα περιγραφεί ο DCT, η συνεισφορά του κατά τη διαδικασία εφαρμογής της συμπίεσης JPEG και ορισμένα σημαντικά χαρακτηριστικά του.



Σχήμα 6.20: Ακολουθία κωδικοποίησης-αποκωδικοποίησης στο πρότυπο JPEG

Υπάρχουν διάφορες επιλογές (με απώλειες, χωρίς απώλειες κτλ) και τρόποι λειτουργίας (ιεραρχικός κτλ) που μπορούν να εφαρμοστούν στο πρότυπο JPEG, ωστόσο η βασική λειτουργία της κωδικοποίησης JPEG είναι αρχικά η υποδιαίρεση μιας εικόνας σε ομάδες των 8x8 pixels και στη συνέχεια η εφαρμογή του DCT σε κάθε ομάδα από αυτές. Ο DCT παράγει 64 συντελεστές μετά την εφαρμογή του, οι οποίοι κβαντίζονται στη μονάδα κβαντισμού και αναδιατάσσονται σε ένα μονοδιάστατο πίνακα με τρόπο "ζιγκ-ζαγκ" πριν κωδικοποιηθούν περαιτέρω στη μονάδα κωδικοποίησης εντροπίας.

Για την αποκωδικοποίηση JPEG εφαρμόζεται η ανάστροφη διαδικασία της κωδικοποίησης, δηλαδή τα κωδικοποιημένα δεδομένα διέρχονται μέσα από έναν αποκωδικοποιητή εντροπίας και μέσω ενός πίνακα "ζιγκ-ζαγκ" για να παραχθεί ένας δισδιάστατος πίνακας. Ο πίνακας αυτός περνάει από μια αντίστροφη διαδικασία κβαντοποίησης, χρησιμοποιώντας τον παράγοντα κβαντοποίησης και στη συνέχεια εφαρμόζεται σε αυτόν ο αντίστροφος DCT (Inverse DCT – IDCT).

Υπάρχουν πολλές διαφορετικές εκδόσεις του μετασχηματισμού, οι οποίες έχουν διαφορετικά χαρακτηριστικά και διαφορετικό μαθηματικό τύπο περιγραφής. Ωστόσο οι κωδικοποιητές που βασίζονται στο DCT χρησιμοποιούν μια δισδιάστατη έκδοση του μετασχηματισμού. Ο δισδιάστατος DCT και ο αντίστροφός του (IDCT) για ένα block NxN από pixels περιγράφονται στις επόμενες δύο εξισώσεις. Να σημειωθεί ότι ο DCT είναι παρόμοιος, ως ένα βαθμό, με τον Discrete Fourier Transform (DFT) αφού στην ουσία αποσυνθέτει ένα σήμα σε μια σειρά αρμονικών συνημιτονικών συναρτήσεων.

2-D DCT:

$$F(u,v) = \frac{2}{N} C(u) C(v) \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} f(x,y) \cos\left[\frac{2x+1}{2N} u\pi\right] \cos\left[\frac{2y+1}{2N} v\pi\right] \quad (1)$$

2-D IDCT:

$$f(x,y) = \frac{2}{N} \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} C(u) C(v) F(u,v) \cos\left[\frac{2x+1}{2N} u\pi\right] \cos\left[\frac{2y+1}{2N} v\pi\right] \quad (2)$$

όπου

$$C(a) = \begin{cases} \frac{1}{\sqrt{2}} , & a=0 \\ 1 , & \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases}$$

f(x,y) είναι η ένταση του pixel σε διαστάσεις x και y, και
F(u,v) είναι ο αντίστοιχος DCT συντελεστής

Από την εξίσωση (1) προκύπτει το συμπέρασμα ότι χρειάζονται συνολικά N^4 πολλαπλασιασμοί για να εφαρμοστεί το DCT σε ένα μπλοκ εικόνας NxN. Έτσι, για να υπολογιστεί ο DCT μιας εικόνας 256x256 χρειάζονται $256^4 = 4.294.967.296$ πολλαπλασιασμοί. Ωστόσο, αν η εικόνα διασπαστεί σε 1024 μπλοκ των 8x8 pixels, τότε θα χρεάζονται μόνο $1024 \times 8^4 = 4.194.304$ πολλαπλασιασμοί. Δηλαδή παρατηρείται μεγάλη βελτίωση στον αριθμό των υπολογισμών που απαιτούνται σε περίπτωση που διασπαστεί η εικόνα σε μπλοκ. Το γεγονός αυτό έχει περαιτέρω πλεονεκτήματα, που σχετίζονται με το βασικό στόχο του μετασχηματισμού, ο οποίος είναι να διαμοιράσει την ενέργεια της εικόνας μεταξύ όσο το δυνατόν λιγότερους συντελεστές. Λαμβάνοντας όμως υπόψη ότι μια εικόνα περιλαμβάνει συνήθως πολλές διαφορετικές συχνότητες, η

προσπάθεια για συμπύκνωση της ενέργειας είναι υπολογιστικά απαιτητική. Αν διασπαστεί η εικόνα σε μπλοκ, τότε κάθε μπλοκ θα περιέχει ένα μικρό μόνο υποσύνολο από παρόμοιες συχνότητες και η συμπύκνωση της ενέργειας γίνεται ευκολότερη και πιο αποδοτική. Επομένως είναι σύνηθες να διασπάται μια εικόνα σε περισσότερα κομμάτια και να εφαρμόζεται σε κάθε ένα από αυτά χωριστά ο DCT.

Μια άλλη πολύ χρήσιμη μορφή του μετασχηματισμού είναι ο μονοδιάστατος DCT και IDCT, ο οποίος υλοποιείται όπως φαίνεται από τις παρακάτω εξισώσεις (3) και (4).

1-D DCT :

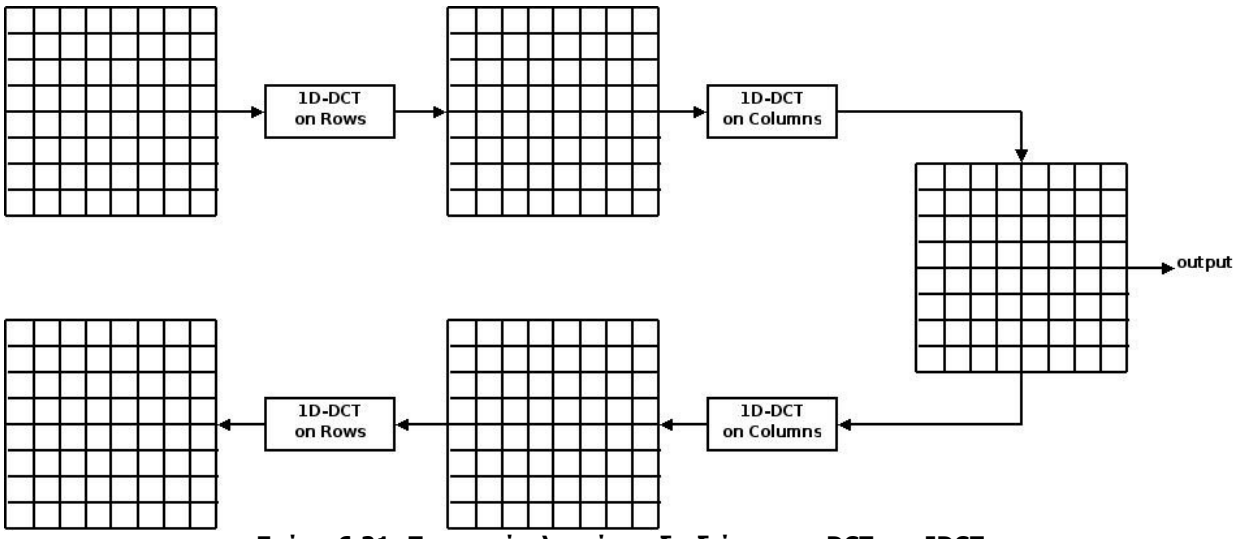
$$X(k) = \sqrt{\frac{2}{N}} C(k) \sum_{i=0}^{N-1} x(i) \cos\left[\frac{2i+1}{2N} k\pi\right] \quad (3)$$

1-D IDCT :

$$X(k) = \sqrt{\frac{2}{N}} \sum_{i=0}^{N-1} C(u) X(u) \cos\left[\frac{(2u+1) n\pi}{2N}\right] \quad (4)$$

Όπως φαίνεται από τις εξισώσεις (1), (2), (3) και (4) οι συντελεστές των μετασχηματισμών DCT και IDCT είναι σταθεροί αριθμοί και συνεπώς μπορούν να προ-υπολογιστούν και να χρησιμοποιηθούν πριν το μετασχηματισμό ώστε να υπάρχει κέρδος στο χρόνο επεξεργασίας. Πρέπει, επίσης, να σημειωθεί ότι οι συντελεστές χρησιμοποιούνται όχι στη δεκαδική αλλά στη Q12 μορφή τους. Αργότερα θα περιγραφεί ποια ακριβώς είναι αυτή η μορφή αλλά και ο λόγος που χρησιμοποιείται.

Η τεχνική που συνήθως εφαρμόζεται για να βελτιωθεί η πολυπλοκότητα των υπολογισμών ενός 2-D μετασχηματισμού DCT είναι αυτός να διασπάται στον υπολογισμό ενός ζευγαριού 1-D μετασχηματισμών. Αυτό είναι εφικτό διότι ο 2-D DCT έχει την ιδιότητα να είναι διασπάσιμος σε αυτό το ζευγάρι των δύο 1-D DCT. Ειδικότερα, για να ληφθεί ο 2-D DCT ενός μπλοκ εικόνας, εφαρμόζεται αρχικά ένας 1-D DCT στις σειρές του μπλοκ και στη συνέχεια εφαρμόζεται ένας 1-D DCT στις στήλες του μπλοκ που προέκυψε σαν αποτέλεσμα από τον προηγούμενο 1-D DCT. Το ίδιο εφαρμόζεται και για τον IDCT. Αυτή τη διαδικασία απεικονίζει το Σχήμα 6.21.



Σχήμα 6.21: Πρακτική υλοποίηση δισδιάστατων DCT και IDCT

Η επόμενη εικόνα δείχνει δύο πίνακες, οι οποίοι περιέχουν ένα παράδειγμα συνόλου συντελεστών στη δεκαδική και στη Q12 μορφή τους. Να σημειωθεί ότι οι τιμές στη δεκαδική μορφή δεν περιλαμβάνουν τον παράγοντα $\sqrt{\frac{2}{N}} = \sqrt{\frac{2}{8}} = \frac{1}{2}$. Ο λόγος για το γεγονός αυτό είναι ότι διαιρώντας τους συντελεστές με το 2 πριν τη μετατροπή τους σε μορφή Q12 μπορεί να έχει ως αποτέλεσμα κάποιες απώλειες σε ακρίβεια. Αντιθέτως, μπορεί να επιτευχθεί μεγαλύτερη ακρίβεια αν αγνοηθεί η διαίρεση και στη συνέχεια γίνει πολλαπλασιασμός με το 2^{11} (αντί για 2^{12}) κατά την μετατροπή σε Q12. Φυσικά είναι δυνατόν να χρησιμοποιηθεί η μορφή Q15 αντί για Q12, ωστόσο δεν προτιμείται για τον εξής λόγο. Οι τιμές των συντελεστών, πριν τη μετατροπή τους, είναι κοντά στο '1'. Συνεπώς η άθροιση των N όρων που λαμβάνει χώρα κατά τον υπολογισμό του μετασχηματισμού DCT μπορεί να προκαλέσει υπερχείλιση. Για να αποφευχθεί αυτό, κάθε όρος πρέπει να μειωθεί κατά 1/N. Για N=8, αυτό μπορεί να γίνει δουλεύοντας με τη μορφή Q12 αντί για τη Q15.

Στη συνέχεια θα δοθεί η μαθηματική ανάλυση της εφαρμογής του μονοδιάστατου μετασχηματισμού DCT πάνω σε ένα block εικόνας με μέγεθος 8x8. Αρχικά γίνεται η υπόθεση πως τα pixels στο block όπου εφαρμόζεται ο DCT έχουν τις τιμές που φαίνονται στον παρακάτω πίνακα (Πίνακας 6.3).

Πίνακας 6.3: Τα pixels του 8x8 block της εικόνας

x[0][0]	x[0][1]	x[0][2]	x[0][3]	x[0][4]	x[0][5]	x[0][6]	x[0][7]
x[1][0]	x[1][1]	x[1][2]	x[1][3]	x[1][4]	x[1][5]	x[1][6]	x[0][7]
x[2][0]	x[2][1]	x[2][2]	x[2][3]	x[2][4]	x[2][5]	x[2][6]	x[0][7]
x[3][0]	x[3][1]	x[3][2]	x[3][3]	x[3][4]	x[3][5]	x[3][6]	x[0][7]
x[4][0]	x[4][1]	x[4][2]	x[4][3]	x[4][4]	x[4][5]	x[4][6]	x[0][7]
x[5][0]	x[5][1]	x[5][2]	x[5][3]	x[5][4]	x[5][5]	x[5][6]	x[0][7]
x[6][0]	x[6][1]	x[6][2]	x[6][3]	x[6][4]	x[6][5]	x[6][6]	x[0][7]

x[7][0]	x[7][1]	x[7][2]	x[7][3]	x[7][4]	x[7][5]	x[7][6]	x[0][7]

Επιπλέον οι τιμές των συντελεστών βαρύτητας του μετασχηματισμού δίνονται στον πίνακα που ακολουθεί (Πίνακας 6.4).

Πίνακας 6.4: Οι τιμές των συντελεστών βαρύτητας του μετασχηματισμού

c[0][0]	c[0][1]	c[0][2]	c[0][3]	c[0][4]	c[0][5]	c[0][6]	c[0][7]
c[1][0]	c[1][1]	c[1][2]	c[1][3]	c[1][4]	c[1][5]	c[1][6]	c[0][7]
c[2][0]	c[2][1]	c[2][2]	c[2][3]	c[2][4]	c[2][5]	c[2][6]	c[0][7]
c[3][0]	c[3][1]	c[3][2]	c[3][3]	c[3][4]	c[3][5]	c[3][6]	c[0][7]
c[4][0]	c[4][1]	c[4][2]	c[4][3]	c[4][4]	c[4][5]	c[4][6]	c[0][7]
c[5][0]	c[5][1]	c[5][2]	c[5][3]	c[5][4]	c[5][5]	c[5][6]	c[0][7]
c[6][0]	c[6][1]	c[6][2]	c[6][3]	c[6][4]	c[6][5]	c[6][6]	c[0][7]
c[7][0]	c[7][1]	c[7][2]	c[7][3]	c[7][4]	c[7][5]	c[7][6]	c[0][7]

Αν το τελικό block 8x8 των τιμών των μετασχηματισμένων pixels συμβολιστεί με **V**, τότε παρακάτω φαίνονται οι ακριβείς μαθηματικοί τύποι των υπολογισμών για το κάθε pixel.

$$V_{00} = x_{00}c_{00}+x_{01}c_{01}+x_{02}c_{02}+x_{03}c_{03}+x_{04}c_{04}+x_{05}c_{05}+x_{06}c_{06}+x_{07}c_{07}$$
$$V_{01} = x_{00}c_{10}+x_{01}c_{11}+x_{02}c_{12}+x_{03}c_{13}+x_{04}c_{14}+x_{05}c_{15}+x_{06}c_{16}+x_{07}c_{17}$$
$$V_{02} = x_{00}c_{20}+x_{01}c_{21}+x_{02}c_{22}+x_{03}c_{23}+x_{04}c_{24}+x_{05}c_{25}+x_{06}c_{26}+x_{07}c_{27}$$
$$V_{03} = x_{00}c_{30}+x_{01}c_{31}+x_{02}c_{32}+x_{03}c_{33}+x_{04}c_{34}+x_{05}c_{35}+x_{06}c_{36}+x_{07}c_{37}$$
$$V_{04} = x_{00}c_{40}+x_{01}c_{41}+x_{02}c_{42}+x_{03}c_{43}+x_{04}c_{44}+x_{05}c_{45}+x_{06}c_{46}+x_{07}c_{47}$$
$$V_{05} = x_{00}c_{50}+x_{01}c_{51}+x_{02}c_{52}+x_{03}c_{53}+x_{04}c_{54}+x_{05}c_{55}+x_{06}c_{56}+x_{07}c_{57}$$
$$V_{06} = x_{00}c_{60}+x_{01}c_{61}+x_{02}c_{62}+x_{03}c_{63}+x_{04}c_{64}+x_{05}c_{65}+x_{06}c_{66}+x_{07}c_{67}$$
$$V_{07} = x_{00}c_{70}+x_{01}c_{71}+x_{02}c_{72}+x_{03}c_{73}+x_{04}c_{74}+x_{05}c_{75}+x_{06}c_{76}+x_{07}c_{77}$$
$$\vdots$$
$$V_{10}=x_{10}c_{00}+x_{11}c_{01}+x_{12}c_{02}+x_{13}c_{03}+x_{14}c_{04}+x_{15}c_{05}+x_{16}c_{06}+x_{17}c_{07}$$
$$V_{11}=x_{10}c_{10}+x_{11}c_{11}+x_{12}c_{12}+x_{13}c_{13}+x_{14}c_{14}+x_{15}c_{15}+x_{16}c_{16}+x_{17}c_{17}$$
$$\vdots$$
$$V_{51}=x_{50}c_{10}+x_{51}c_{11}+x_{52}c_{12}+x_{53}c_{13}+x_{54}c_{14}+x_{55}c_{15}+x_{56}c_{16}+x_{57}c_{17}$$

6.3.2.Υλοποίηση σε Γλώσσα C

Η υλοποίηση του DCT σε γλώσσα προγραμματισμού C βασίζεται στην τεχνική της διάσπασης μιας εικόνας σε πολλαπλά blocks και η εφαρμογή του DCT σε κάθε ένα από αυτά[26]. Ακόμη εφαρμόζεται η διάσπαση του 2-D μετασχηματισμού σε δύο ανεξάρτητους 1-D, σύμφωνα με όσα αναφέρθηκαν

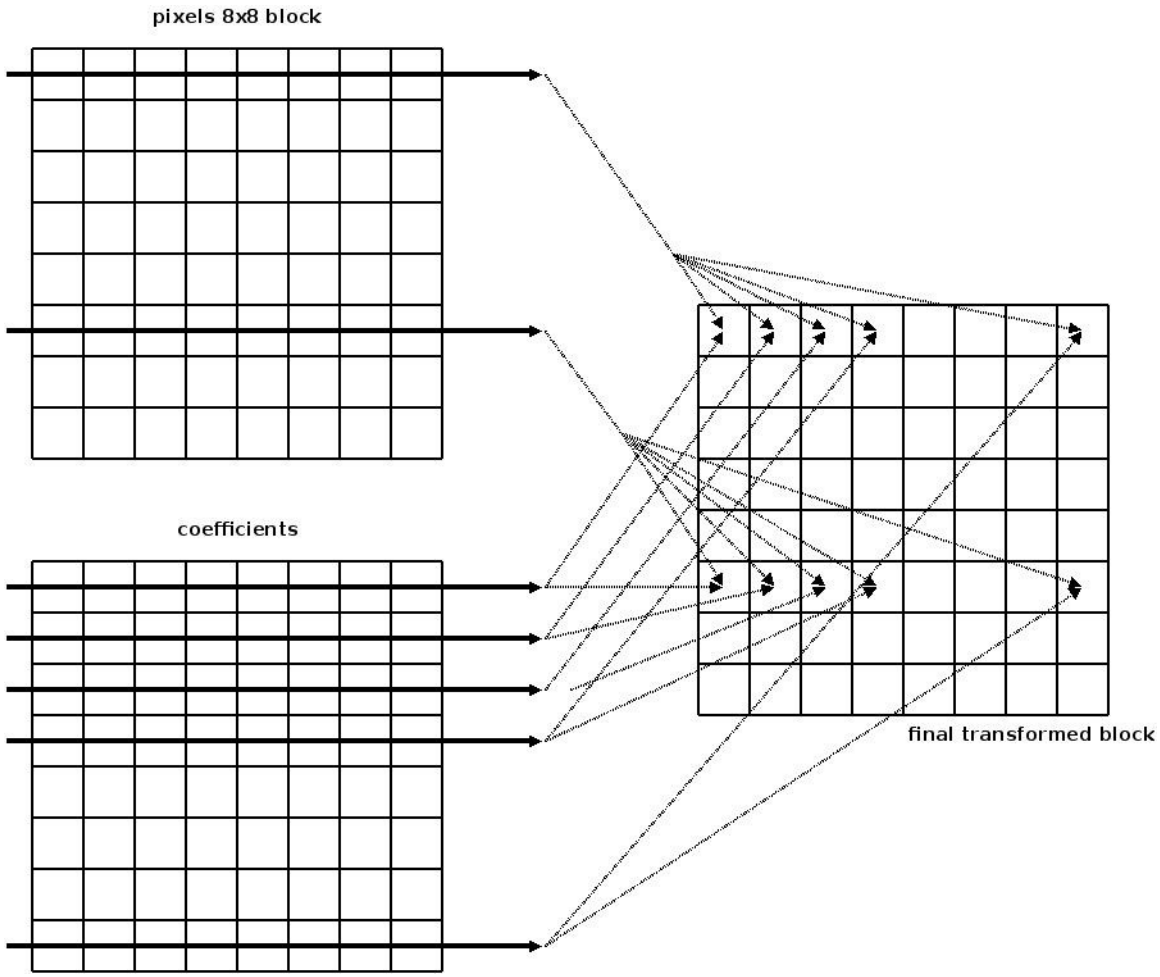
στην παρούσα ενότητα. Παρακάτω (Πλαίσιο 6.5) φαίνεται η συνάρτηση σε γλώσσα προγραμματισμού C, η οποία υλοποιεί έναν 2-D μετασχηματισμό, σύμφωνα με την παραπάνω τεχνική. Η συνάρτηση αυτή βρίσκει εφαρμογή σε blocks εικόνας με μέγεθος 8x8 το καθένα.

```
void dct (void) {
      int i,j,x,y;
      int value[8];

      /*perform 1D DCT on the rows*/
      for(i=0;i<64;i+=8) {
            for(y=0;y<8;++y) {
                  value[y] = 0;

                  for(x=0;x<8;++x)
                        value[y]+=coe[y][x]*block[i+x];
            }
      for(y=0;y<8;++y)
            block[i+y] = (value[y]>>12);
      }

      /*perform 1D DCT on the columns*/
      for(j=0;j<64;j++) {
            for(y=0;y<8;++y) {
                  value[y] = 0;

                  for(x=0;x<8;++x)

      value[y]+=coe[y][x]*block[j+(x*8)];
            }
      for(y=0;y<8;++y)
            block[j+(y*8)] = (value[y]>>12);
      }
}
```

Πλαίσιο 6.5: Κώδικας C για την υλοποίηση μετασχηματισμού DCT σε block εικόνας 8x8

Η υλοποίηση που εφαρμόζει ο παραπάνω κώδικας βασίζεται άμεσα στους τύπους (3) και (4) της παρούσας ενότητας. Γίνεται υπόθεση ότι υπάρχουν διαθέσιμοι στο πρόγραμμα δύο πίνακες, ένας coe[8][8] που περιλαμβάνει τους συντελεστές του μετασχηματισμού σε μορφή Q12 και ένας block[64], ο οποίος περιλαμβάνει σε συμπτυγμένη μορφή τις τιμές της έντασης των pixels για το συγκεκριμένο block πάνω στο οποίο εφαρμόζεται ο μετασχηματισμός. Αξίζει να σημειωθεί ότι αρχικά ο 1-D μετασχηματισμός εφαρμόζεται στις γραμμές του αρχικού μπλοκ και στη συνέχεια εφαρμόζεται ο ίδιος μετασχηματισμός στις στήλες του ενδιάμεσου τροποποιημένου block. Ωστόσο η σειρά -είτε πρώτα ανά στήλες και έπειτα ανά γραμμές, είτε ανάποδα- δεν έχει σημασία. Παρατηρείται, ακόμη, ότι κάθε pixel του block συσχετίζεται -κατά το μετασχηματισμό- με τα υπόλοιπα pixels της γραμμής (στήλης) στην οποία ανήκει. Μάλιστα, ο συσχετισμός αυτός είναι διαφορετικός για κάθε pixel, καθώς χρησιμοποιείται για κάθε ένα από αυτά και κάποιος διαφορετικός συντελεστής του μετασχηματισμού. Το τελικό block μετά τον πλήρη μετασχηματισμό 2-D DCT αποθηκεύεται και πάλι στον αρχικό πίνακα block[64].

Το Σχήμα 6.22 παρουσιάζει παραστατικά τον τρόπο λειτουργίας του μετασχηματισμού DCT. Το σημαντικό στοιχείο που πρέπει να τονιστεί είναι ότι για τον υπολογισμό της μετασχηματισμένης τιμής ενός pixel πρέπει να γίνει πολλαπλασιασμός μιας γραμμής του πίνακα των pixel με μια γραμμή του πίνακα των συντελεστών του μετασχηματισμού. Για pixel που ανήκουν στην ίδια γραμμή του block, ο μετασχηματισμός τους προκύπτει από τον πολλαπλασιασμό των τιμών των pixel της ίδιας γραμμής στην οποία ανήκουν, με διαφορετικές όμως γραμμές του πίνακα συντελεστών. Οι γραμμές αυτές είναι οι αντίστοιχες της στήλης που ανήκουν τα pixel. Συμπερασματικά, για τον υπολογισμό του μετασχηματισμού ενός pixel της εικόνας πρέπει να πολλαπλασιαστεί η γραμμή του block της εικόνας στην οποία ανήκει το συγκεκριμένο pixel με τη γραμμή του πίνακα συντελεστών που έχει το ίδιο νούμερο με τη στήλη που ανήκει το εν λόγω pixel στο block της εικόνας.



Σχήμα 6.22: Τρόπος υπολογισμού των μετασχηματισμών DCT ενός block εικόνας

6.3.3.Υλοποίηση στο Επαναδιατάξιμο MicroSoc

Για να αναδειχθούν ορισμένα χαρακτηριστικά του επαναδιατάξιμου MicroSoc και η ικανότητά του να απεικονίζει με αποδοτικότητα το μετασχηματισμό DCT (ή τον

IDCT) επιλέχθηκε η υλοποίηση ενός απλού 1-D DCT και η εφαρμογή του σε ένα block εικόνας με μέγεθος 8x8, που περιέχει πραγματικές τιμές έντασης για τα pixel. Ο συγκεκριμένος μετασχηματισμός αποτελεί άλλωστε ένα τμήμα από την εφαρμογή του πλήρη 2-D DCT, οπότε στην ουσία υλοποιείται ένα κομμάτι του 2-D DCT.

Οι συντελεστές του μετασχηματισμού ελήφθησαν από πραγματικά δεδομένα και φαίνονται στον επόμενο πίνακα (Πίνακας 6.5), απευθείας στη μορφή Q12 στην οποία χρησιμοποιούνται από το μετασχηματισμό.

Πίνακας 6.5: Οι συντελεστές μετασχηματισμού που χρησιμοποιήθηκαν στην εφαρμογή του DCT

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Επίσης, παρακάτω (Πίνακας 6.6) δίνεται το block της εικόνας πάνω στο οποίο εφαρμόστηκε ο μετασχηματισμός 1-D κατά γραμμές, όπου οι τιμές αντιστοιχούν στη ένταση pixels από υπαρκτή εικόνα.

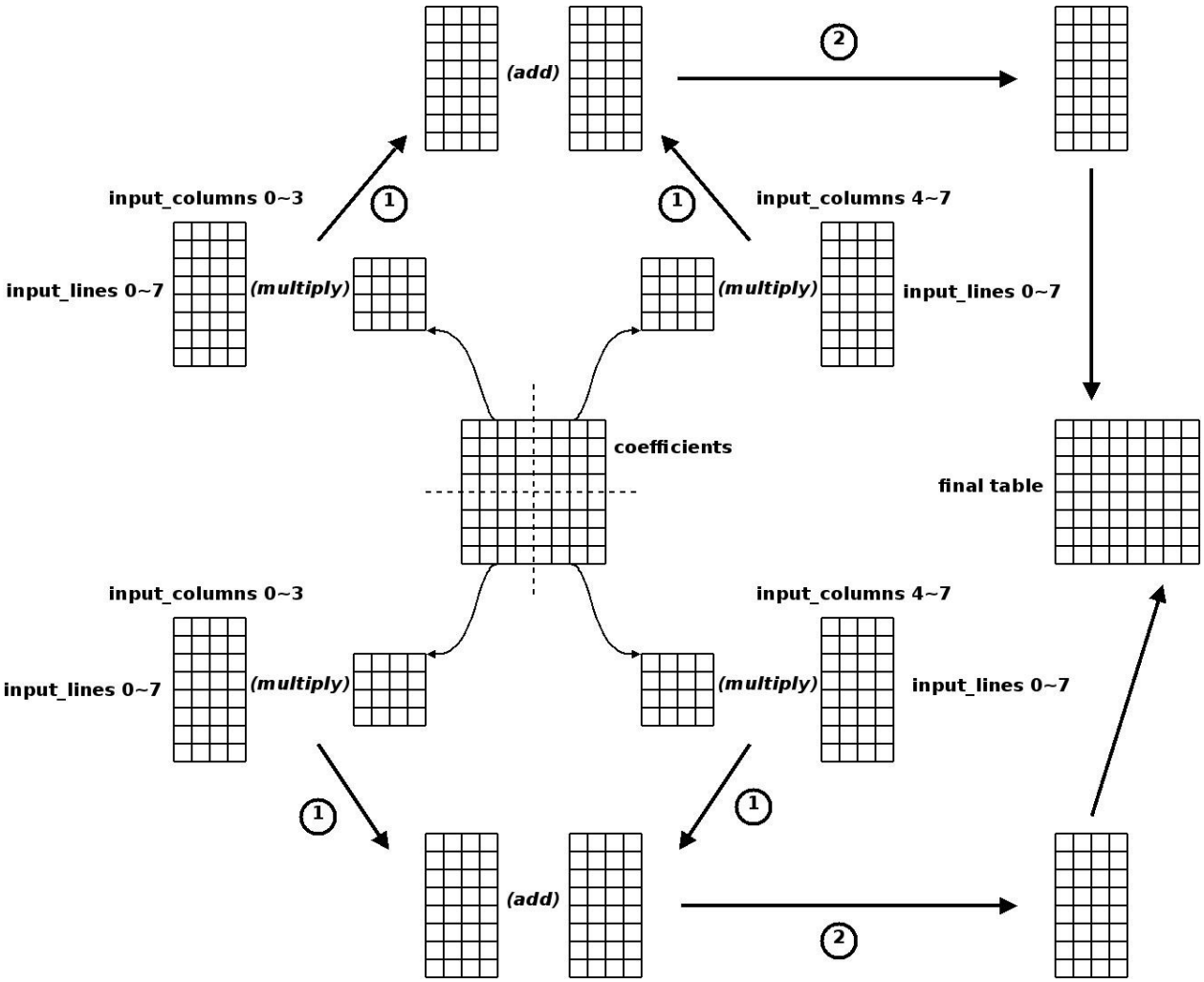
Πίνακας 6.6: Το block της εικόνας που χρησιμοποιήθηκε στην εφαρμογή του DCT

128	127	130	128	134	130	128	128
128	127	130	128	134	130	128	128
125	126	130	127	130	131	128	127
124	128	127	126	129	130	127	127
131	132	130	129	132	128	129	131
129	131	134	131	133	129	131	131
129	134	134	136	136	137	134	132
133	135	136	138	137	140	135	132

Η τεχνική (Σχήμα 6.23) που χρησιμοποιήθηκε για την απεικόνιση του 1-D DCT σε ένα block από pixels μεγέθους 8x8 βασίζεται στην ιδέα της διάσπασης του block σε μικρότερα και ο συνδυασμός των αποτελεσμάτων για να προκύψουν οι τελικοί συντελεστές του μετασχηματισμού. Ο πίνακας των συντελεστών μετασχηματισμού χωρίζεται σε τέσσερις υποπίνακες διαστάσεων 4x4 και ότι ο πίνακας με τις τιμές των εντάσεων των pixels χωρίζεται σε δύο υποπίνακες, διαστάσεων 8x4 ο κάθε ένας. Τα βήματα που ακολουθούνται στη συνέχεια είναι τα εξής :

- Κάθε υποπίνακας με τις τιμές των συντελεστών μετασχηματισμού πολλαπλασιάζεται με τον υποπίνακα των pixels που ανήκει στην ίδια πλευρά του αρχικού πίνακα (δεξιά ή αριστερά). Επομένως οι υποπίνακες

των συντελεστών της πάνω και της κάτω δεξιά γωνίας πολλαπλασιάζονται με το δεξιά υποπίνακα των pixels (στήλες 4 έως 7). Αντίστοιχα, οι υποπίνακες των συντελστών της πάνω και κάτω αριστερά γωνίας πολλαπλασιάζονται με τον αριστερά υποπίνακα των pixels (στήλες 0 έως 3). Αποτέλεσμα αυτών των πολλαπλασιασμών είναι η παραγωγή τεσσάρων πινάκων 8x4. Το βήμα αυτό πραγματοποιείται αποκλειστικά στο επαναδιατάξιμο περιφερειακό.



Σχήμα 6.23: Τεχνική απεικόνισης του 1-D DCT που χρησιμοποιήθηκε στο rMicroSoc

- Στη συνέχεια πρέπει να γίνει πρόσθεση μεταξύ των ενδιάμεσων πινάκων που προέκυψαν από το προηγούμενο βήμα. Όπως δείχνει το Σχήμα 6.23, πρόσθεση γίνεται μεταξύ των πινάκων που προέκυψαν από τους πολλαπλασιασμούς των υποπινάκων συντελεστών που ανήκουν στην ίδια πλευρά του αρχικού πίνακα (κάτω ή πάνω). Οι δύο πίνακες 8x4 που προκύπτουν απαρτίζουν τη δεξιά και την αριστερή πλευρά του τελικού πίνακα με τις μετασχηματισμένες τιμές των pixels. Το βήμα αυτό υλοποιείται στον πυρήνα επεξεργαστή ARM. Ο ARM μεταφέρει τα αποτελέσματα του επαναδιατάξιμου περιφερειακού στην κεντρική μνήμη του συστήματος και από εκεί αντλεί τα αποτελέσματα για να εκτελέσει τους υπολογισμούς που απαιτούνται.

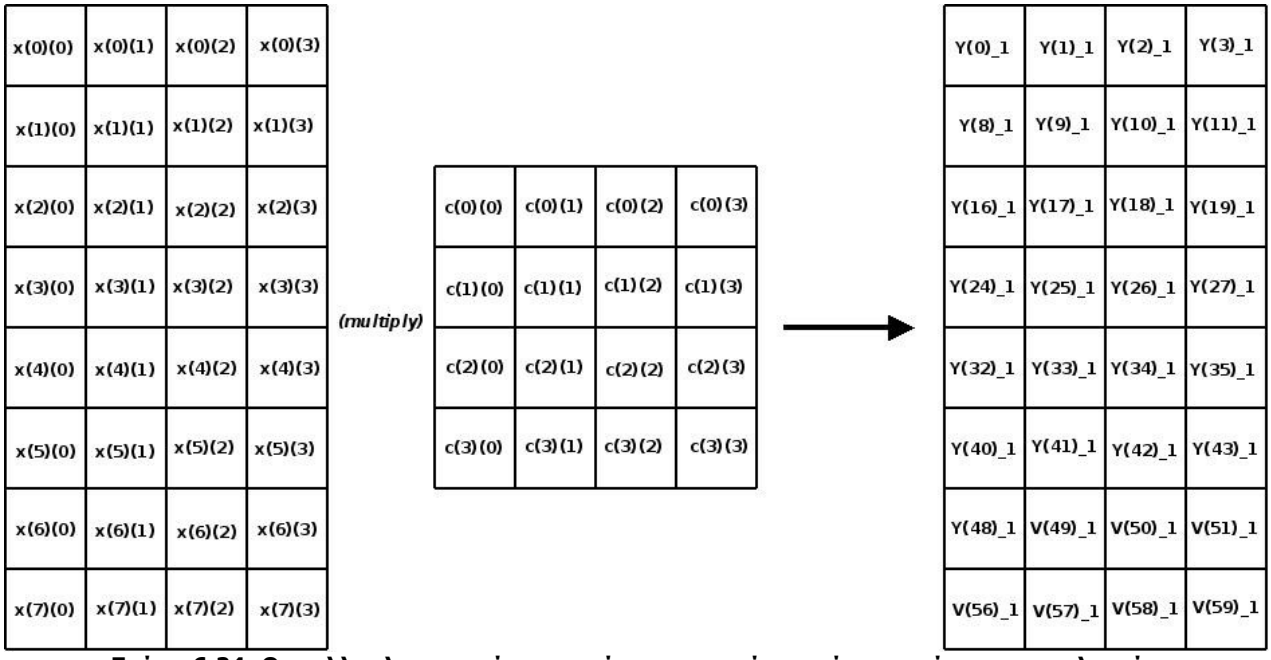
Ουσιαστικά, οι πίνακες που προκύπτουν στο πρώτο βήμα περιέχουν το μισό μέρος από την τελική τιμή των μετασχηματισμένων τιμών των pixels. Το άλλο μισό μέρος προκύπτει από τους πολλαπλασιασμούς που θα γίνουν με το τεταρτημόριο του πίνακα συντελεστών που βρίσκεται στην ίδια πλευρά (πάνω ή κάτω) του αρχικού πίνακα με το τεταρτημόριο του παρόντος μέρους.

Για παράδειγμα ας θεωρηθεί ο μετασχηματισμός που αφορά στο pixel στη θέση (1,3) -γραμμή #1, στήλη #3- του αρχικού block της εικόνας. Η αντίστοιχη μετασχηματισμένη τιμή του είναι το Y(11) (βλ. Σχήμα 6.24, Σχήμα 6.25). Η τιμή του Y(11) θα υπολογίζεται ως:

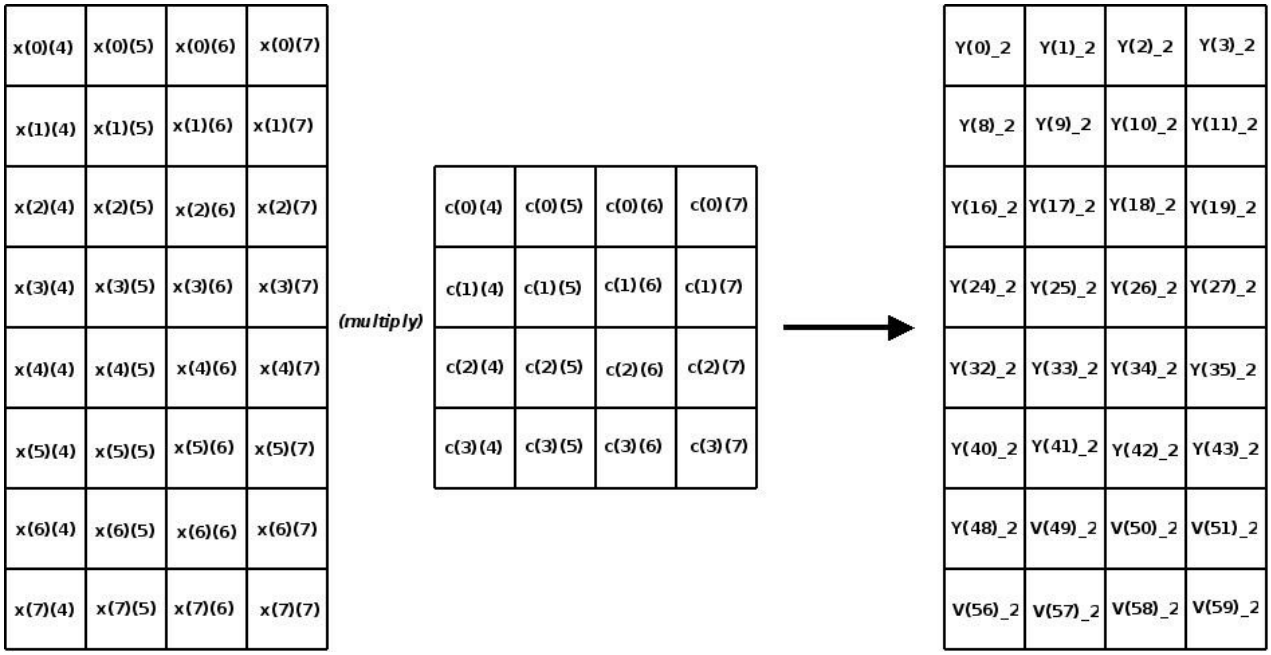
$$Y(11) = x_{10}c_{30}+x_{11}c_{31}+x_{12}c_{32}+x_{13}c_{33}+x_{14}c_{34}+x_{15}c_{35}+x_{16}c_{36}+x_{17}c_{37}$$

Φαίνεται ότι για τον υπολογισμό του πρέπει να πολλαπλασιαστεί η τρίτη γραμμή του πίνακα συντελεστών με όλη τη δεύτερη γραμμή του πίνακα των pixels. Με την τεχνική που εφαρμόζεται, τα δύο μέρη του συντελεστή θα υπολογιστούν χωριστά στο πρώτο βήμα και στο δεύτερο βήμα θα προστεθούν για να προκύψει η τελική τιμή. Συγκεκριμένα, θα χρησιμοποιηθούν τα δύο τεταρτημόρια της πάνω πλευράς του αρχικού πίνακα συντελεστών. Από τους πολλαπλασιασμούς με το αριστερά τεταρτημόριο θα προκύψει το κομμάτι $Y(11)_1 = x_{10}c_{30}+x_{11}c_{31}+x_{12}c_{32}+x_{13}c_{33}$, ενώ με τους πολλαπλασιασμούς με το δεξιά τεταρτημόριο θα προκύψει το κομμάτι $Y(11)_2 = x_{14}c_{34}+x_{15}c_{35}+x_{16}c_{36}+x_{17}c_{37}$. Στη δεύτερη φάση τα δύο αυτά μέρη θα προστεθούν για να προκύψει το τελικό αποτέλεσμα.

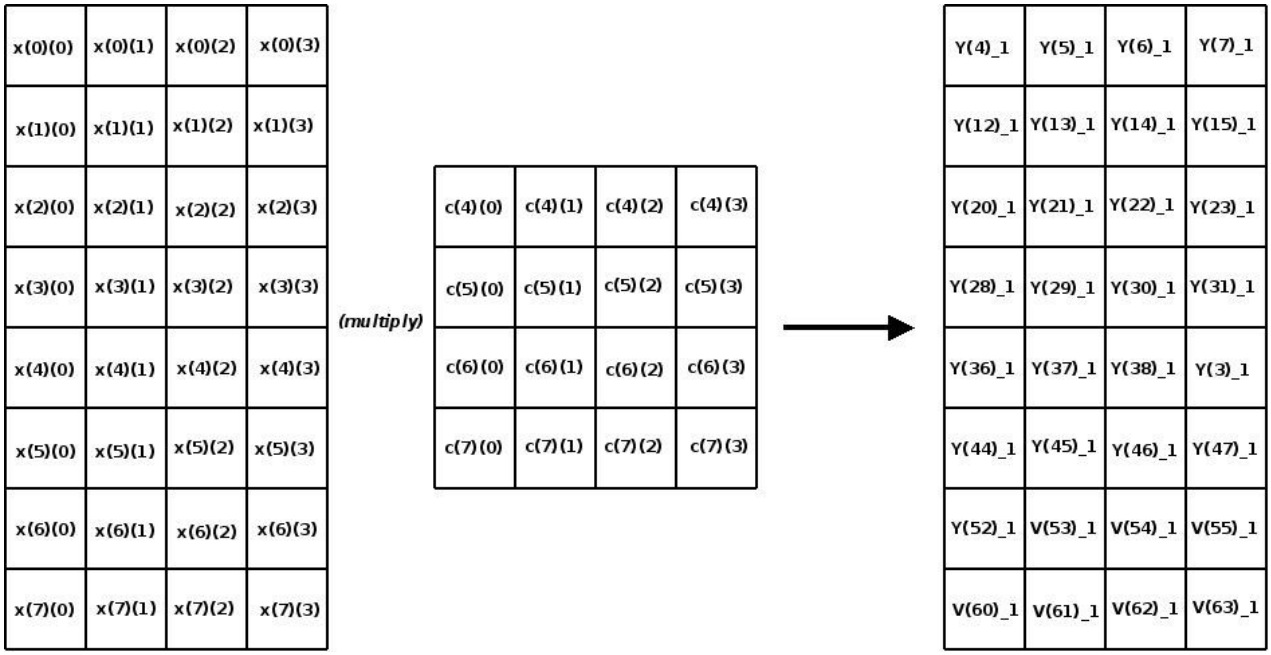
Στα επόμενα σχήματα (Σχήμα 6.24-Σχήμα 6.27) φαίνονται οι τέσσερις πολλαπλασιασμοί που πρέπει να πραγματοποιηθούν στο πρώτο βήμα της υλοποίησης.



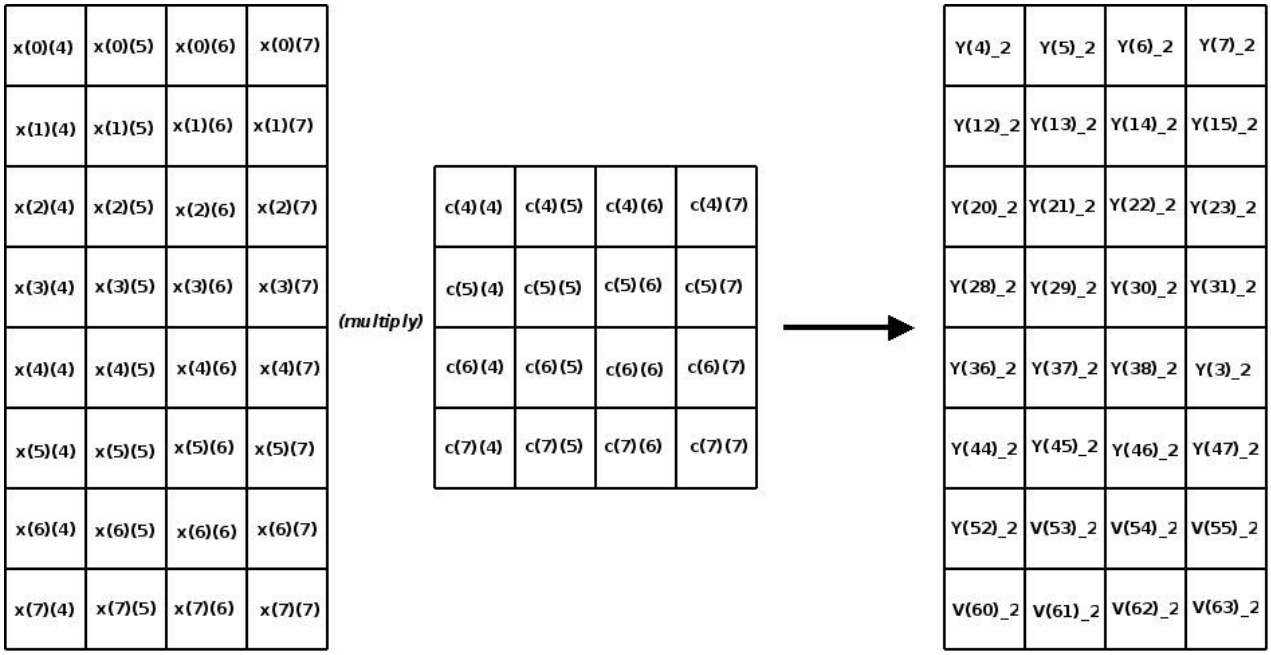
Σχήμα 6.24: Οι πολλαπλασιασμοί για το πάνω αριστερά κομμάτι του πίνακα συντελεστών



Σχήμα 6.25: Οι πολλαπλασιασμοί για το πάνω δεξιά κομμάτι του πίνακα συντελεστών



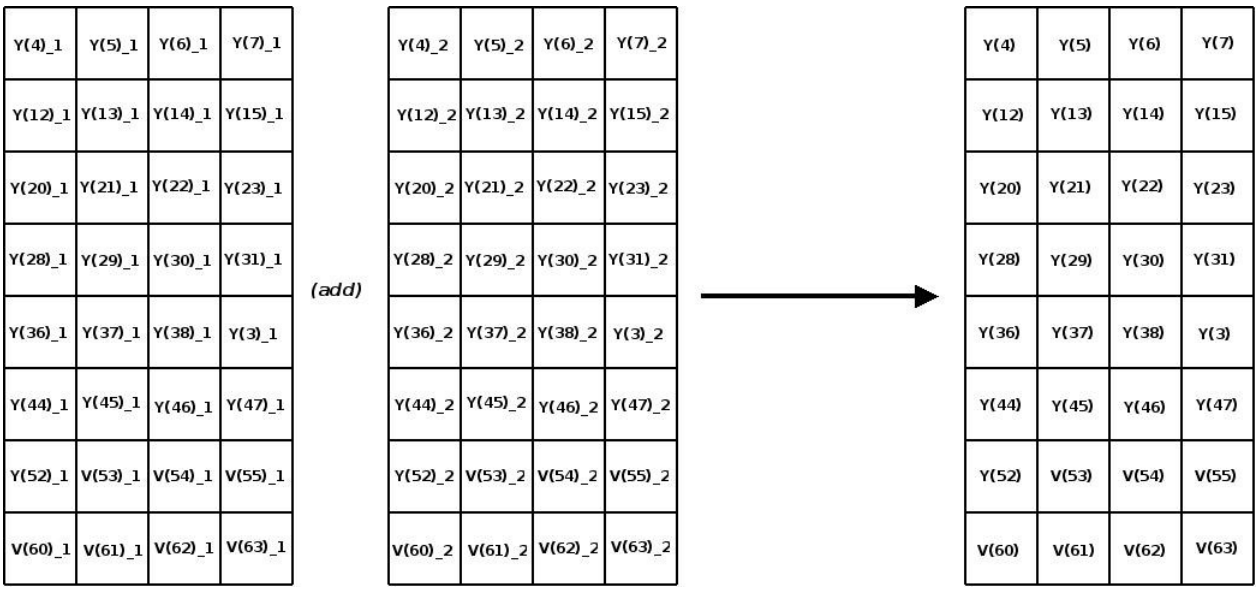
Σχήμα 6.26: Οι πολλαπλασιασμοί για το κάτω αριστερά κομμάτι του πίνακα συντελεστών



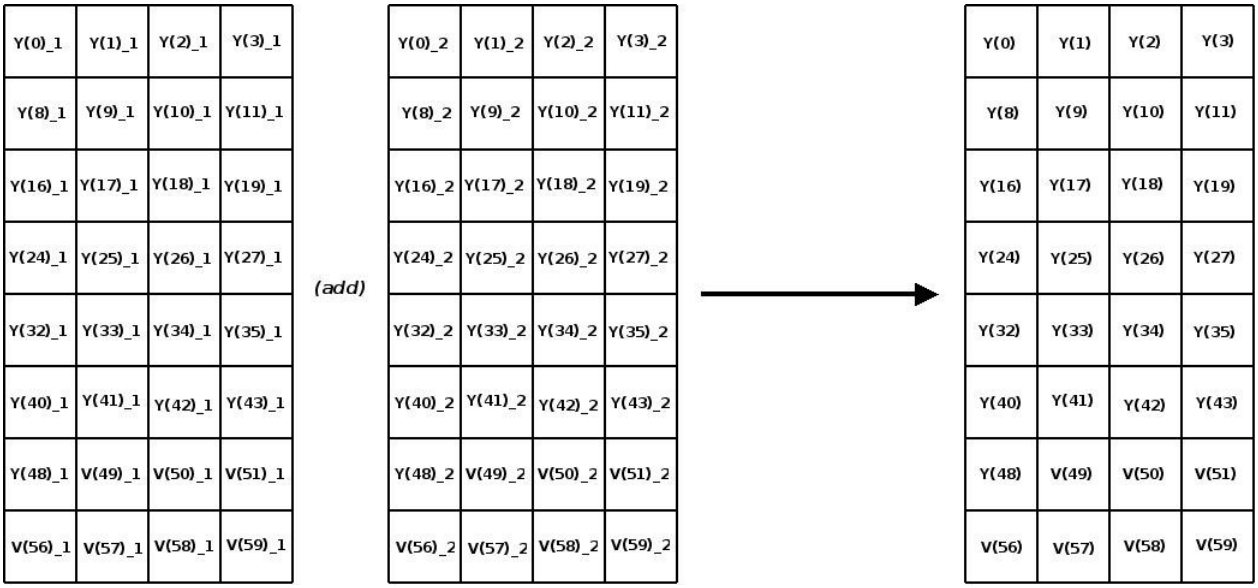
Σχήμα 6.27: Οι πολλαπλασιασμοί για το κάτω δεξιά κομμάτι του πίνακα συντελεστών

Οι πίνακες που προκύπτουν στο τέλος των παραπάνω πολλαπλασιασμών πρέπει να μεταφερθούν στη κεντρική μνήμη RAM του συστήματος, για να είναι αργότερα προσβάσιμες από τον ARM.

Στη συνέχεια, στα επόμενα σχήματα (Σχήμα 6.28, Σχήμα 6.29), παρουσιάζονται οι δύο προσθέσεις που απαιτούνται για να προκύψουν τα τελικά αποτελέσματα.



Σχήμα 6.28: Η πρόσθεση για να προκύψει το αριστερά κομμάτι του τελικού πίνακα



Σχήμα 6.29: Η πρόσθεση για να προκύψει το δεξιά κομμάτι του τελικού πίνακα

Όπως φαίνεται, οι πίνακες που προκύπτουν από τις προσθέσεις αυτές αποτελούν τμήματα του πίνακα με τα τελικά αποτελέσματα.

6.3.4.Περίοδος Λειτουργίας

Στην ενότητα αυτή περιγράφονται τα βασικά βήματα που αποτελούν μια περίοδο λειτουργίας του συστήματος. Κατά τη διάρκεια μιας περιόδου φορτώνονται στον πίνακα οι συντελεστές που ανήκουν σε ένα από τα τεταρτημόρια στα οποία χωρίστηκε ο αρχικός πίνακας συντελεστών και γίνονται όλοι οι πολλαπλασιασμοί που αφορούν το τεταρτημόριο αυτό. Αποτέλεσμα των υπολογισμών της περιόδου είναι ένας από τους 8x4 πίνακες, για τους οποίους έγινε λόγος παραπάνω. Μια περίοδος αποτελείται από τη φάση αρχικοποίησης και τη φάση εκτέλεσης, δηλαδή έναν εσωτερικό βρόχο επανάληψης που επαναλαμβάνεται συνολικά 8 φορές σε μια περίοδο. Κατά τη διάρκεια μιας επανάληψης του βρόχου αυτού υπολογίζονται τα γινόμενα που αφορούν μια γραμμή του τελικού πίνακα της περιόδου και αυτός είναι ο λόγος που πρέπει να γίνουν συνολικά 8 επαναλήψεις (μία για κάθε μία από τις 8 γραμμές του τελικού πίνακα). Οι συντελεστές πολλαπλασιασμού του μετασχηματισμού φορτώνονται στη φάση αρχικοποίησης της περιόδου και διατηρούνται σταθεροί σε όλη τη διάρκειά της. Αντίθετα, σε κάθε επανάληψη των βρόχων εκτέλεσης φορτώνεται στον πίνακα και μια διαφορετική γραμμή από το κομμάτι του πίνακα των pixels που συμμετέχει στους πολλαπλασιασμούς με το τρέχον τεταρτημόριο (βλ. την ανάλυση της τεχνικής απεικόνισης παραπάνω). Μελετώντας την τεχνική που εφαρμόζεται, όπως αυτή περιγράφεται παραπάνω στην παρούσα ενότητα, συμπεραίνεται ότι απαιτούνται 4 περίοδοι λειτουργίας για να ολοκληρωθεί η υλοποίηση του αλγορίθμου. Η μετάβαση από τη μια περίοδο λειτουργίας την επόμενη, γίνεται με την παρέμβαση του επεξεργαστή ARM.

Με την ολοκλήρωση όλων των περιόδων λειτουργίας που πρέπει να εκτελεστούν, στη μνήμη RAM υπάρχουν αποθηκευμένα τα αποτελέσματα των πολλαπλασιασμών που πραγματοποιήθηκαν στον επαναδιατάξιμο πίνακα. Αναφορικά και με το Σχήμα 6.23 στη μνήμη υπάρχουν οι τέσσερις πίνακες 8x4 που προκύπτουν από τον πολλαπλασιασμό των 4 τμημάτων του πίνακα συντελεστών με τα αντίστοιχα του block της εικόνας. Το βήμα που απομένει είναι να γίνουν οι κατάλληλες προσθέσεις μεταξύ αυτών των πινάκων, ώστε να προκύψει ο τελικός μετασχηματισμένος πίνακας. Οι προσθέσεις αυτές πραγματοποιούνται στον επεξεργαστή ARM, δηλαδή το επαναδιατάξιμο περιφερειακό στο σημείο αυτό έχει ολοκληρώσει την λειτουργία του.

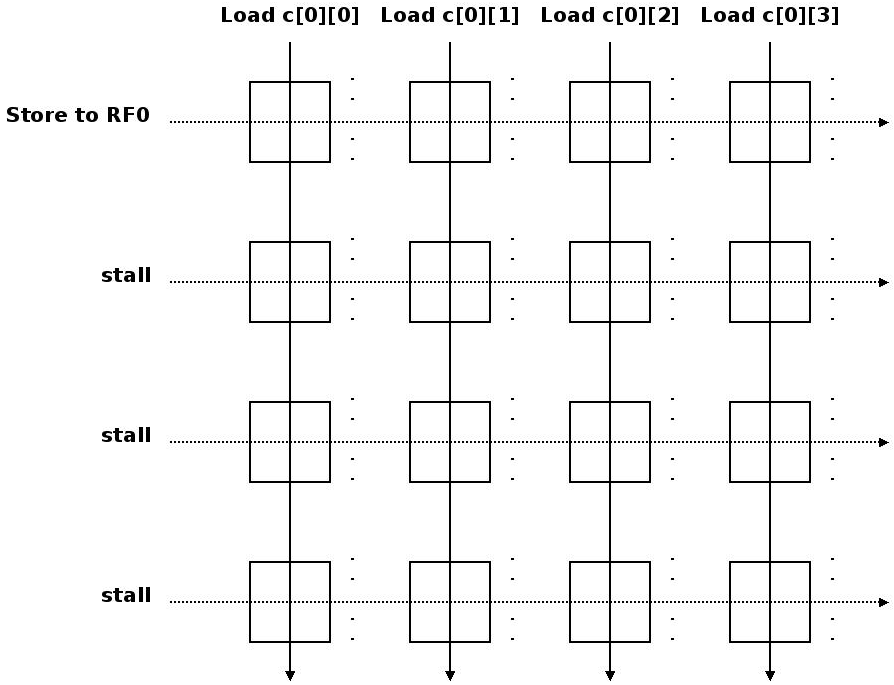
Παρακάτω περιγράφονται αναλυτικά η φάση αρχικοποίησης και η φάση της εκτέλεσης των βρόχων επανάληψης.

6.3.4.1. Φάση Αρχικοποίησης

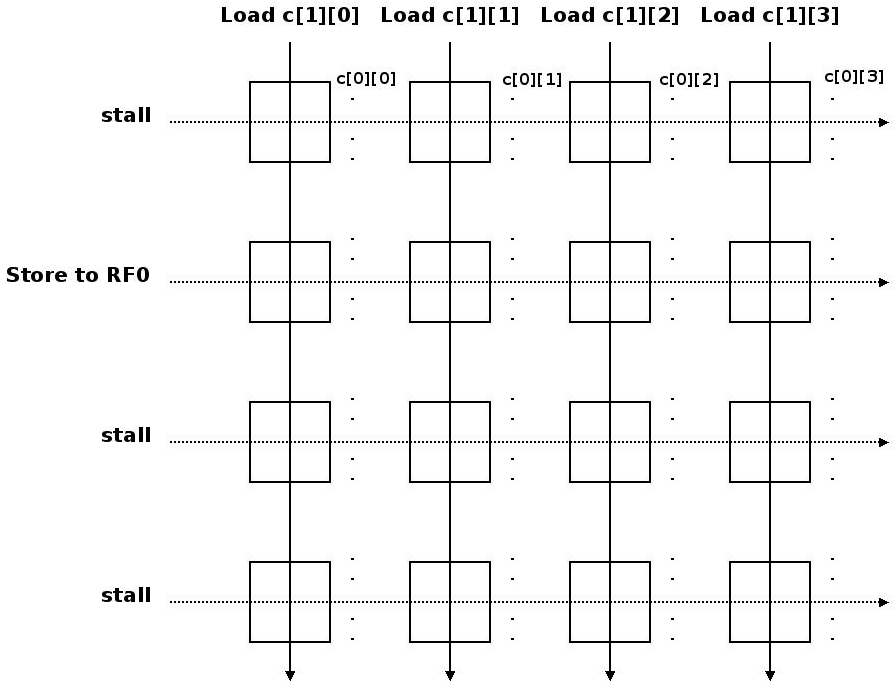
Κατά τη διάρκεια της φάσης αρχικοποίησης κάθε περιόδου λειτουργίας πρέπει να γίνει η φόρτωση των κελιών του επαναδιατάξιμου πίνακα με τις τιμές του κατάλληλου τεταρτημορίου με τους πολλαπλασιαστικούς συντελεστές του μετασχηματισμού. Η φόρτωση γίνεται με τέτοιο τρόπο, ώστε ο πίνακας των συντελεστών του μετασχηματισμού να απεικονιστεί νοητά στα κελιά του επαναδιατάξιμου πίνακα. Δηλαδή σε κάθε κελί θα αποθηκευτεί και ένας διαφορετικός συντελεστής, έτσι ώστε στο πρώτο κελί της πρώτης γραμμής να υπάρχει ο συντελεστής coeff[0][0], στο δεύτερο κελί της πρώτης γραμμής να υπάρχει ο coeff[0][1], στο πρώτο κελί της δεύτερης γραμμής να υπάρχει ο coeff[1][0] κ.ο.κ.

Η διαδικασία αυτή ολοκληρώνεται συνολικά σε 8 κύκλους διαμόρφωσης του πίνακα, εκ των οποίων οι τέσσερις αντιστοιχούν στη μεταφορά των συντελεστών από τη μνήμη δεδομένων στον πίνακα και οι άλλοι τέσσερις στην αποθήκευση των συντελεστών στους τοπικούς καταχωρητές των κελιών (με τους κύκλους να εναλλάσσονται ο ένας του άλλου).

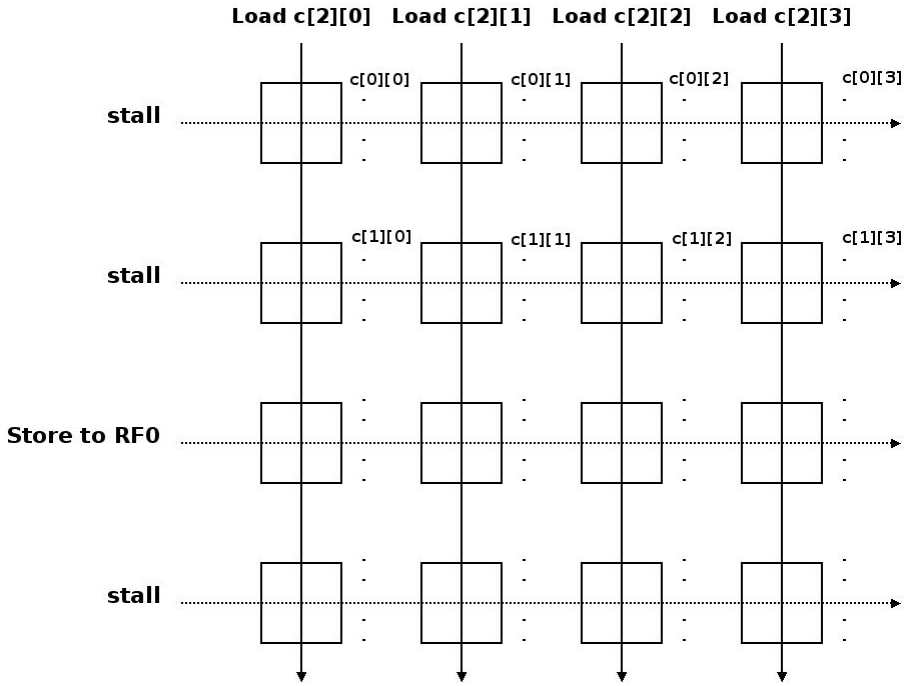
Τα επόμενα σχήματα (Σχήμα 6.30-Σχήμα 6.33) δείχνουν παραστατικά το πώς φορτώνονται οι συντελεστές στα κελιά. Δεξιά από κάθε κελί φαίνονται τα περιεχόμενα του αρχείου καταχωρητών του κελιού αυτού. Να σημειωθεί ότι τα σχήματα περιγράφουν τη φόρτωση των συντελεστών του πρώτου τεταρτημορίου του πίνακα. Η διαδικασία, ωστόσο, είναι ίδια και για τα υπόλοιπα τεταρτημόρια.



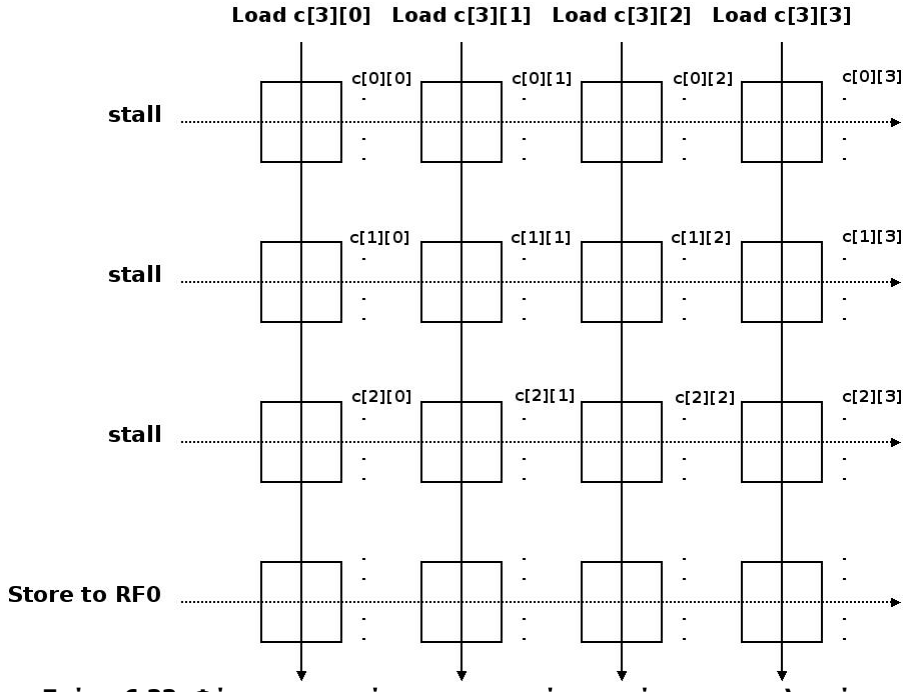
Σχήμα 6.30: Φόρτωση της πρώτης γραμμής του πίνακα συντελεστών



Σχήμα 6.31: Φόρτωση της δεύτερης γραμμής του πίνακα συντελεστών



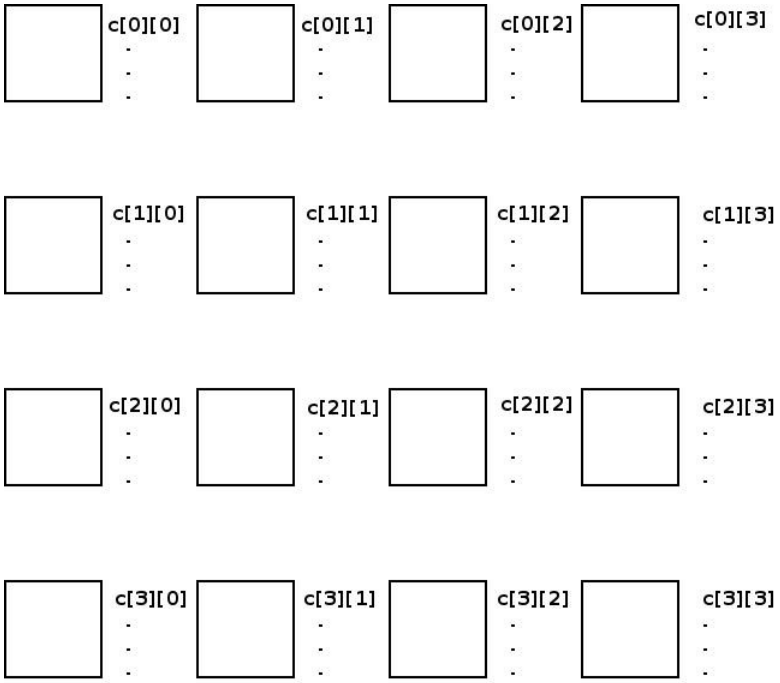
Σχήμα 6.32: Φόρτωση της τρίτης γραμμής του πίνακα συντελεστών



Σχήμα 6.33: Φόρτωση της τέταρτης γραμμής του πίνακα συντελεστών

Το Σχήμα 6.34 δείχνει την τελική εικόνα των περιεχομένων των καταχωρητών όλων των κελιών. Διακρίνεται ότι σε κάθε κελί ο συντελεστής που θα χρησιμοποιηθεί στους πολλαπλασιασμούς βρίσκεται αποθηκευμένος στον

καταχωρητή RF0 του αρχείου καταχωρητών.



Σχήμα 6.34: Τα περιεχόμενα των καταχωρητών των κελιών μετά τη φάση της φόρτωσης των συντελεστών

Παρατηρείται ότι η φόρτωση των συντελεστών του μετασχηματισμού γίνεται σε λειτουργία ανά στήλες και ακολούθως η αποθήκευση σε λειτουργία ανά γραμμές. Επιπλέον, σε κάθε κύκλο διαμόρφωσης γίνεται αποθήκευση σε μία μόνο γραμμή του επαναδιατάξιμου πίνακα, ανάλογα με το ποια γραμμή από τον πίνακα συντελεστών πολλαπλασιασμού έχει φορτωθεί. Η αποθήκευση γίνεται στον καταχωρητή RF0 του τοπικού αρχείου καταχωρητών των κελιών της κατάλληλης γραμμής, ενώ οι υπόλοιπες γραμμές παραμένουν σε κατάσταση stall.

6.3.4.2. Φάση Υπολογισμών

Στο τέλος της φάσης φόρτωσης όλα τα κελιά του πίνακα έχουν φορτωθεί με τους κατάλληλους συντελεστές πολλαπλασιασμού του μετασχηματισμού. Επομένως, θα πρέπει στη συνέχεια να ακολουθήσει η φάση των πολλαπλασιασμών των συντελεστών του μετασχηματισμού με τις τιμές των pixels που προβλέπονται από την τεχνική υλοποίησης. Στη φάση αυτή υλοποιείται, όπως έχει αναφερθεί παραπάνω, ένας βρόχος 8 επαναλήψεων και σε κάθε επανάληψη υπολογίζεται ένα μέρος από την τελική μετασχηματισμένη τιμή από pixels που ανήκουν σε μια ίδια γραμμή του αρχικού block εικόνας. Στο τέλος των υπολογισμών προκύπτουν, ταυτόχρονα, κομμάτια από 4 τελικούς

συντελεστές, ένας συντελεστής ανά γραμμή του επαναδιατάξιμου πίνακα.

Τα βασικά βήματα που ακολουθούνται σε κάθε μία από αυτές τις επαναλήψεις είναι:

- Αρχικά φορτώνονται οι τιμές έντασης των pixels, τα οποία βρίσκονται στη γραμμή του block που αντιστοιχεί στο νούμερο της τρέχουσας επανάληψης. Οι τιμές αυτές εισόδου πρέπει να φορτωθούν σε λειτουργία ανά στήλες.

- Παράλληλα με τη φόρτωση των τιμών εισόδου, σε κάθε κελί γίνεται ο πολλαπλασιασμός των εισόδων με τον συντελεστή πολλαπλασιασμού που έχει αποθηκευτεί στο εκάστοτε κελί. Με τον τρόπο αυτό γίνεται η βασική πράξη του μετασχηματισμού, δηλαδή ο πολλαπλασιασμός τιμών έντασης των pixels με τον κατάλληλο συντελεστή πολλαπλασιασμού. Ακολούθως, το τελικό αποτέλεσμα που προκύπτει από τους πολλαπλασιασμούς αυτούς στα κελιά του πίνακα πρέπει να αποθηκευτεί στο τοπικό αρχείο καταχωρητών. Η αποθήκευση γίνεται στον καταχωρητή RF1 του εκάστοτε κελιού.

- Στο επόμενο βήμα, ακολουθεί η πράξη της συσσώρευσης των κατάλληλων γινομένων, με σκοπό να προκύψουν οι τελικές τιμές των κομματιών των συντελεστών του μετασχηματισμού. Οι τιμές αυτές εμφανίζονται, τελικά, ταυτόχρονα στην έξοδο των κελιών της δεξιάς στήλης του επαναδιατάξιμου πίνακα και επιπλέον αποθηκεύονται και στον καταχωρητή RF2 των συγκεκριμένων κελιών.

- Το τελευταίο βήμα της κάθε επανάληψης περιλαμβάνει την ανάγνωση από τον κεντρικό επεξεργαστή ARM των τιμών των συντελεστών που υπολογίστηκαν και η αποθήκευσή τους στη μνήμη RAM του συστήματος. Να σημειωθεί ωστόσο ότι, σε αντίθεση με τα προηγούμενα, αυτό το βήμα υλοποιείται εξ` ολοκλήρου στον κώδικα C που έχει γραφτεί και που εκτελείται στον επεξεργαστή ARM.

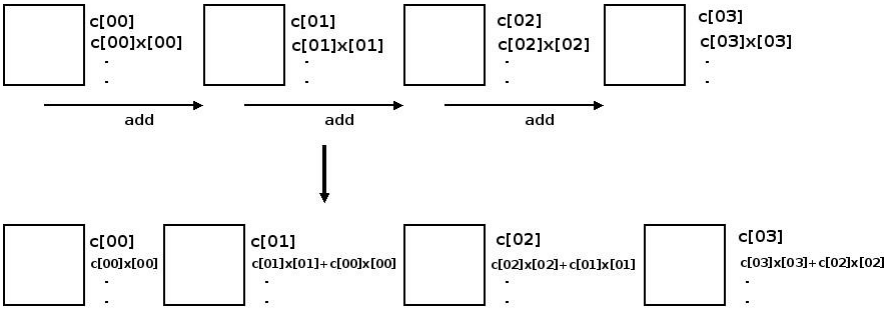
6.3.5.Ψευδοκώδικας – Περιεχόμενο Διαμόρφωσης

Τα βήματα της φάσης υπολογισμού, υλοποιούνται με το ψευδο-κώδικα που ακολουθεί. Παρατηρείται ότι η φάση υπολογισμών διαρκεί 8 κύκλους διαμόρφωσης.

C1 : $MEM_{DATA}\ x\ RF0 = ALU_{out1}$
C2 : $RF1 <- ALU_{out1}$
C3 : $OUT <- RF1$
C4 : $RF1 + OUT_{leftcell} = ALU_{out2}$

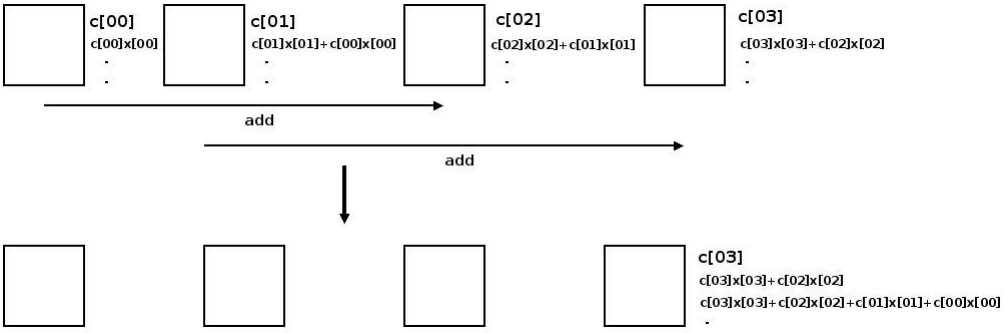
C5 : $RF1 <- ALU_{out2}$
C6 : $OUT <- RF1$
C7 : $RF1 + OUT_{twoleftcell} = ALU_{out3}$
C8 : $RF2 <- ALU_{out3}$
C9 : $\langle Request\ Interrupt \rangle$
C10 : $\langle stall \rangle$

Για τη συσσώρευση των γινομένων των κελιών, υλοποιούνται δύο πράξεις πρόσθεσης. Αρχικά μια πρόσθεση μεταξύ του γινομένου κάθε κελιού με το αντίστοιχο γινόμενο του κελιού στα αριστερά, όπως φαίνεται στο επόμενο σχήμα (Σχήμα 6.35) για μία σειρά του πίνακα.



Σχήμα 6.35: Άθροιση των ενδιάμεσων γινομένων των γειτονικών κελιών, από αριστερά προς τα δεξιά

Στη συνέχεια, άλλη μία πρόσθεση μεταξύ των προηγούμενων προσωρινών αθροισμάτων κάθε κελιού με τα αντίστοιχα αθροίσματα του δεύτερου κελιού από τα αριστερά. Μετά τα δύο αυτά βήματα προκύπτουν στα κελιά της δεξιάς πλευράς τα τελικά αποτελέσματα της τρέχουσας επανάληψης, όπως φαίνεται στο επόμενο σχήμα (Σχήμα 6.36).



Σχήμα 6.36: Άθροιση ενδιάμεσων γινομένων των μη διπλανών κελιών, από αριστερά προς τα δεξιά

Μετά από 8 κύκλους διαμόρφωσης, το περιφερειακό ζητάει διακοπή από τον ARM και περνάει σε κατάσταση stall. Όταν ο επεξεργαστής εξυπηρετήσει τη διακοπή, διαβάζει τους καταχωρητές RF1 των κελιών που βρίσκονται στη δεξιά πλευρά του πίνακα και αποθηκεύει τα περιεχόμενά τους στη μνήμη RAM. Στη

συνέχεια συνεχίζει την εκτέλεση του περιεχομένου διαμόρφωσης από τον πρώτο κύκλο της φάσης εκτέλεσης (C1). Τα περιεχόμενα που φορτώνονται προς επεξεργασία από τη μνήμη δεδομένων είναι διαφορετικά σε σχέση με την προηγούμενη επανάληψη. Ωστόσο, αν έχουν ολοκληρωθεί 8 επαναλήψεις, σε αυτή την περίπτωση ο ARM καθορίζει ότι η εκτέλεση πρέπει να συνεχιστεί μετά τον κύκλο C10. Το περιεχόμενο διαμόρφωσης που ακολουθεί δηλώνει ότι πρέπει να γίνει άλμα μέσα στη μνήμη διαμόρφωσης και η νέα θέση εκτέλεσης από τη μνήμη θα είναι η αρχή της φάσης φόρτωσης. Ο λόγος είναι ότι μετά από 8 επαναλήψεις η τρέχουσα περίοδος λειτουργίας πρέπει να ολοκληρωθεί και να ξεκινήσει η επόμενη. Στη νέα περίοδο θα φορτωθούν από την αρχή νέοι συντελεστές πολλαπλασιασμού (φάση φόρτωσης) και θα ακολουθήσουν οι 8 επαναλήψεις της φάσης εκτέλεσης. Η διαδικασία αυτή επαναλαμβάνεται συνολικά 4 φορές, σύμφωνα με την τεχνική απεικόνισης.

Ακολουθώντας στη συνέχεια τον παραπάνω κώδικα, κατασκευάζεται το αντίστοιχο περιεχόμενο διαμόρφωσης που πρέπει να φορτωθεί στη μνήμη διαμόρφωσης ώστε να επιτευχθεί η απεικόνιση της εφαρμογής στο επαναδιατάξιμο υλικό.

6.3.6. Έλεγχος Επιδόσεων

Για τον έλεγχο των επιδόσεων και της αποδοτικότητας της απεικόνισης, πραγματοποιήθηκε σύγκριση μεταξύ αυτής και μιας απεικόνισης στο σύστημα MicroSoc χωρίς τη χρησιμοποίηση της επαναδιατάξιμης μονάδας. Για τη δεύτερη αυτή απεικόνιση έγινε χρήση ενός κώδικα σε γλώσσα C, ο οποίος περιγράφει πλήρως την υλοποίηση ενός μετασχηματισμού 1D DCT. Ο κώδικας αυτός φορτώθηκε στη μνήμη RAM του MicroSoc και εκτελέστηκε από τον ARM, με αποτέλεσμα να παραχθούν στη μνήμη ROM τα αναμενόμενα αποτελέσματα, δηλαδή οι μετασχηματισμένες τιμές των pixels της οθόνης. Αυτό που ενδιαφέρει είναι να επιβεβαιωθεί η ορθότητα της απεικόνισης στην επαναδιατάξιμη λογική αλλά και να συγκριθούν οι χρόνοι που απαιτούνται στις δύο περιπτώσεις, από την έναρξη λειτουργίας μέχρι να παραχθούν τα τελικά αποτελέσματα.

Το Πλαίσιο 6.6 δείχνει τον κώδικα, ο οποίος υλοποιεί το μετασχηματισμό και ο οποίος φορτώθηκε στο MicroSoc για να εκτελεστεί από τον ARM επεξεργαστή.

Ο παρακάτω κώδικας υλοποιεί το 1D DCT πάνω σε ένα block εικόνας 8x8. Πρόκειται για τη προσαρμογή του κώδικα υλοποίησης ενός DCT, ο οποίος παρουσιάστηκε σε προηγούμενη ενότητα, και συγκεκριμένα του τμήματος που αφορά την εφαρμογή σε γραμμές. Στο τμήμα αυτό, λοιπόν, έχουν προστεθεί οι οδηγίες *#pragma*, ώστε να οριστεί στον ARM ποιο είναι το κυρίως κομμάτι του κώδικα και πόσος χώρος πρέπει να δεσμευτεί στη RAM (64 θέσεις για τον πίνακα y). Στην αρχή του κώδικα εισάγονται στον κώδικα οι πίνακες συντελεστών (*coe[]*) και pixel εικόνας (*image_in*).

```
static const int coe[8][8] =
{
16,11,10,16,24,40,51,61,
12,12,14,19,26,58,60,55,
14,13,16,24,40,57,69,56,
14,17,22,29,51,87,80,62,
18,22,37,56,68,109,103,77,
24,35,55,64,81,104,113,92,
49,64,78,87,103,121,120,101,
72,92,95,98,112,100,103,99
};

static const int image_in[64] =
{
128,127,130,128,134,130,128,128,
128,127,130,128,134,130,128,128,
125,126,130,127,130,131,128,127,
124,128,127,126,129,130,127,127,
131,132,130,129,132,128,129,131,
129,131,134,131,133,129,131,131,
129,134,134,126,126,137,134,132,
133,135,136,138,137,140,135,132
};

extern void C_Entry(void);

#pragma arm section rwdata= "pinakas"
int y[64];
#pragma arm section rwdata

#pragma arm section code
void C_Entry(void)
{
    int i,y,x;
    int value[8];
    for(i=0; i<64; i+=8) {
          for(y=0; y<8; y++) {
            value[y]=0;
            for(x=0;x<8;x++)
                  value[y] += coe[y][x]*image_in[i+x];
          }
          for(y=0;y<8;y++)
            y[i+y] = value[y]>>12;
    }
}
#pragma arm section code
```

Πλαίσιο 6.6: Κώδικας υλοποίησης του 1-D DCT μετασχηματισμού στον ARM

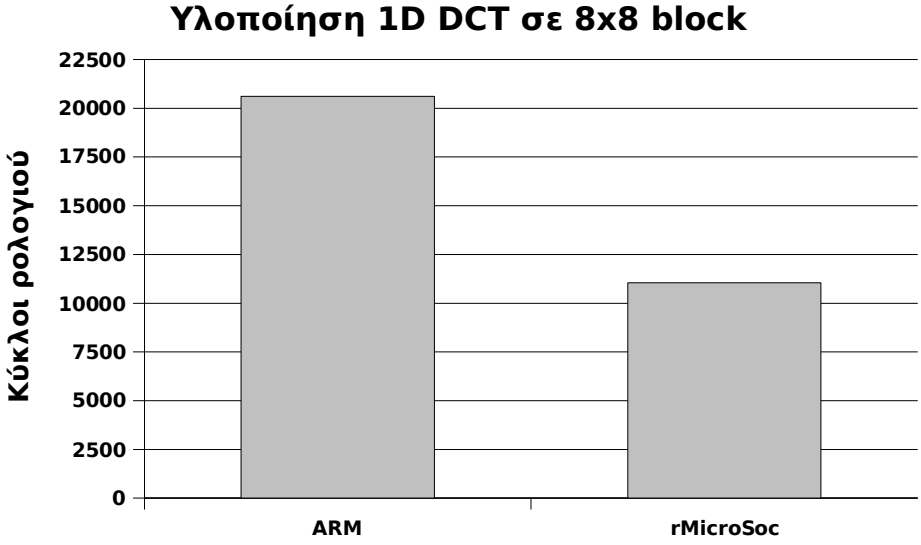
6.3.7.Συγκριτικά Αποτελέσματα

Σε πρώτο στάδιο πραγματοποιήθηκε η υλοποίηση του 1D μετασχηματισμού στο σύστημα χωρίς την επαναδιατάξιμη λογική. Φορτώθηκε ο κώδικας που παρουσιάστηκε παραπάνω στη μνήμη ROM και μετά την εκτέλεσή τού, τα τελικά αποτελέσματα του μετασχηματισμού βρίσκονται αποθηκευμένα στη μνήμη RAM.

Ο συνολικός αριθμός των κύκλων που απαιτήθηκαν για να εκτελεστεί η παραπάνω υλοποίηση και να αποθηκευτούν τα δεδομένα στη μνήμη RAM ήταν 20614.

Στη συνέχεια ακολούθησε η υλοποίηση στο rMicroSoc. Φορτώθηκαν τα περιεχόμενα της μνήμης διαμόρφωσης και δεδομένων, προετοιμάστηκαν οι ρουτίνες εξυπηρέτησης των διακοπών και εκτελέστηκε το περιεχόμενο διαμόρφωσης. Τα τελικά αποτελέσματα αποθηκεύτηκαν στη μνήμη RAM και ήταν ίδια με αυτά που προέκυψαν από την υλοποίηση στο σύστημα χωρίς το επαναδιατάξιμο περιφερειακό. Για την εξαγωγή των αποτελεσμάτων με αυτή τη μέθοδο και την αποθήκευσή τους στη μνήμη απαιτήθηκαν συνολικά 11055 κύκλοι.

Το Σχήμα 6.37 απεικονίζει η σύγκριση της απόδοσης των δύο υλοποιήσεων, σύμφωνα με τους κύκλους που απαιτήθηκαν.



Σχήμα 6.37: Σύγκριση της υλοποίησης ενός 1-D DCT στο rMicroSoc και στον ARM, σύμφωνα με τον αριθμό των κύκλων ρολογιού που απαιτούνται

Από τον αριθμό των κύκλων που απαιτήθηκαν για τις δύο περιπτώσεις των υλοποιήσεων διαπιστώνεται ότι η χρήση της επαναδιατάξιμης λογικής επιταχύνει σημαντικά τη διάρκεια εκτέλεσης των υπολογισμών. Μάλιστα, η επιτάχυνση αυτή είναι ίση με 46%.

ΚΕΦΑΛΑΙΟ 7
Συμπεράσματα και Μελλοντική Εργασία

7.1.Συμπεράσματα

Σε αυτή τη διπλωματική εργασία παρουσιάστηκε ένα ενσωματωμένο και ολοκληρωμένο (SoC) επαναδιατάξιμο σύστημα επεξεργασίας, το rMicroSoc, με επαναδιάταξη επιπέδου λέξης. Η υλοποίηση του rMicroSoc στηρίχθηκε στα συστήματα MorphoSys[25],[7] και MicroSoc[1],[2], ακολουθώντας το μοντέλο επαναδιατάξιμης αρχιτεκτονικής του πρώτου και επεκτείνοντας το υλικό του δεύτερου με την προσθήκη ενός επαναδιατάξιμου περιφερειακού. Το συγκεκριμένο περιφερειακό αποτελεί μια slave μονάδα του συστήματος επικοινωνίας AMBA-ASB, το οποίο έχει υλοποιηθεί ως η βάση ανταλλαγής δεδομένων μεταξύ του επεξεργαστή ARM και των υπολοίπων περιφερειακών στο MicroSoc. Η κύρια μονάδα του επαναδιατάξιμου περιφερειακού είναι ένας 4x4 πίνακας επαναδιατάξιμων μονάδων επεξεργασίας, ο οποίος ακολουθεί το μοντέλο επεξεργασίας SIMD[5],[6],[24]. Η λειτουργία και οι εσωτερικές διασυνδέσεις του πίνακα καθορίζονται από λέξεις διαμόρφωσης που παρέχονται από τη μνήμη διαμόρφωσης του περιφερειακού.

Με την υλοποίηση του συστήματος rMicroSoc επιτεύχθηκε η επιτάχυνση των υπολογισμών για φίλτρα FIR και για το μετασχηματισμό DCT. Συγκεκριμένα, απεικονίστηκαν στο rMicroSoc ένα φίλτρο FIR 4-tap (με επέκταση της τεχνικής απεικόνισης που παρουσιάστηκε στο [27]), ένα φίλτρο FIR 16-tap (με τεχνική απεικόνισης που βασίστηκε στην ιδέα της κοινής χρήσης της μνήμης, όπως παρουσιάστηκε στο [29]) και ο 1-D DCT μετασχηματισμός σε ένα block εικόνας 8x8 (με τεχνική απεικόνισης βασισμένη στο [26]). Στις περιπτώσεις της υλοποίησης των συγκεκριμένων εφαρμογών παρατηρήθηκε μείωση του αριθμού των κύκλων που απαιτήθηκαν για την ολοκλήρωση των υπολογισμών, λόγω της χρήσης του επαναδιατάξιμου υλικού, σε σύγκριση με την αντίστοιχη υλοποίηση σε software –μέσω του ARM- , κατά ποσοστό 80%, 97% και 46% αντίστοιχα. Τα δύο φίλτρα FIR που επιλέχθηκαν, απεικονίστηκαν με βάση διαφορετικές τεχνικές υλοποίησης και παρατηρήθηκε στο τέλος ότι η τεχνική που εφαρμόστηκε στο 16-tap φίλτρο υπερέχει σε απόδοση αυτής που εφαρμόστηκε στο 4-tap φίλτρο.

7.2.Μελλοντική Εργασία

Η μελλοντική εργασία που αφορά το σύστημα που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας μπορεί να χωριστεί σε δύο κατηγορίες, το ερευνητικό και το πρακτικό κομμάτι.

Στο πρώτο κομμάτι, το κύριο ερευνητικό ενδιαφέρον αφορά στην αλλαγή του μοντέλου επεξεργασίας από το μοντέλο SIMD σε ένα μικτό μοντέλο SIMD/MIMD, με πιθανή προσθήκη μιας ξεχωριστής μνήμης διαμόρφωσης μέσα σε κάθε επαναδιατάξιμο κελί, και η μελέτη στο πεδίο της ελάχιστης κατανάλωσης. Άλλη πιθανή ερευνητική εργασία είναι η ανάπτυξη ενός εργαλείου αυτοματοποιημένης απεικόνισης ψηφιακών εφαρμογών στο υπάρχον σύστημα. Ένα τέτοιο εργαλείο θα δέχεται σαν είσοδο τον κώδικα εκτέλεσης της εφαρμογής σε κάποια γλώσσα προγραμματισμού υψηλού επιπέδου και θα παράγει τα περιεχόμενα διαμόρφωσης και τα περιεχόμενα της μνήμης δεδομένων, τα οποία θα απεικονίζουν τη συγκεκριμένη εφαρμογή στο υλικό του συστήματος. Επίσης, ως πιθανή μελλοντική εργασία, προτείνεται και η πλήρης σύνθεση του κώδικα περιγραφής του συστήματος με κατάλληλα εργαλεία λογισμικού, ώστε να γίνει αξιολόγηση των χαρακτηριστικών του συστήματος και εξερεύνηση του χώρου σχεδίασης που προκύπτει από αρχιτεκτονικές τύπου MorphoSys.

Στο πρακτικό κομμάτι, τέλος, η αρχιτεκτονική μπορεί να τροποποιηθεί ώστε το μέγεθος του επαναδιατάξιμου πίνακα να είναι μεγαλύτερο του 4x4 που χρησιμοποιείται τώρα με δυνατότητες επιλογής διαφορετικών δομικών μονάδων ανάλογα με τις εφαρμογές που πρόκειται να απεικονιστούν . Ακόμα, η προσθήκη -στο κυρίως σύστημα- της δυνατότητας λειτουργίας πολλαπλών master στο διάδρομο AMBA-ASB και η χρησιμοποίηση μονάδας DMA.

ΚΕΦΑΛΑΙΟ 8
Παράρτημα - Πηγαίοι Κώδικες Verilog

8.1.Κώδικας RF_module

```
module RF_module(
	clk,		// Clock signal
	clkn,		// Reverse clock signal
	resetn,		// Reset signal
	databus_in,	// Data in from the ASB
	databus_out,	// Data out to the ASB
	baddr,		// Address in from the ASB
	bwrite,		// The write signal from the ASB
	dsel,		// Select signal for the module (from the ASB)
	bwait,		// WAIT signal - response of the module
	blast,		// LAST signal - response of the module
	berror,		// ERROR signal - response of the module
	irq		// Request interrupt from the ARM
);


//-----------------------------------------Input ports ---------------------------------------//
	input wire [31:0] databus_in;
	input wire [31:0] baddr;
	input wire bwrite;
	input wire dsel;
	input wire clk;
	input wire clkn;
	input wire resetn;

// ---------------------------------------- Output ports ---------------------------------------//
	output wire [31:0] databus_out;
	output wire bwait;
	output wire blast;
	output wire berror;
	output wire irq;

// ------------------------------------ Internal variables -------------------------------------//

	wire [124:0] context_bus;
	wire [7:0] contmem_addr;

	wire [7:0] contreg_data;
	wire contreg_sel;

	wire [15:0] datamem_bus_in;
	wire [63:0] datamem_bus_out;		// 4x16 bit - for every line/column
	wire [7:0] datamem_read_addr;
	wire [7:0] datamem_write_addr;

	wire [3:0] datamem_sel;

	wire [7:0] memreg_data;
	wire memreg_sel;

	wire [8:0] loop_cont_reg_data;
	wire loop_cont_reg_sel;
	wire cont_loop_en;

	wire synch_reg_data;
	wire synch_reg_sel;
	wire syn_bit;

	wire [8:0] loop_data_reg_data;
	wire loop_data_reg_sel;
	wire data_loop_en;

	wire [15:0] array_out;
```

```
        wire [3:0] array_cell_sel;
        wire [2:0] array_word_sel;

        wire intr_ctrl_sel;
        wire intr_ctrl_wr;
        wire [1:0] intr_ctrl_data_wr;        // Data sent to the interrupt interface
        wire [1:0] intr_ctrl_data_r;         // Data read from the interrupt interface


// ------------------------------------ Code starts here... ---------------------------------------//
        ReconfigArray array(
                        .context_mem_in              (context_bus[123:0]),
                        .data_mem_in                 (datamem_bus_out),
                        .func_mode                   (context_bus[124]),
                                        // 0 for horizontal function mode, 1 for vertical function mode
                        .data_out                    (array_out),
                        .output_cell_sel             (array_cell_sel),
                        .output_word_sel             (array_word_sel),
                        .clk                         (clk),
                        .clkn                        (clkn),
                        .resetn                      (resetn)
        );

        loop_reg l_cont_reg(
                        .clk                         (clk),
                        .resetn                      (resetn),
                        .data_in                     (loop_cont_reg_data),
                        .write_en                    (loop_cont_reg_sel),
                        .control_bits                ({context_bus[3] , context_bus[0]}),
                        .loop_en                     (cont_loop_en)
        );

        mem_synch_reg syn_reg(
                        .clk                         (clk),
                        .resetn                      (resetn),
                        .data_in                     (synch_reg_data),
                        .write_en                    (synch_reg_sel),
                        .syn_bit                     (syn_bit)
        );

        ARM_CONT_REG cont_reg(
                        .clk                         (clk),
                        .clkn                        (clkn),
                        .resetn                      (resetn),
                        .master_wr_en                (contreg_sel),
                        .int_loops                   (cont_loop_en),
                        .function_mode               (context_bus[0] && context_bus[2]),
                        .syn_bit                     (syn_bit),
                        .data_in                     (contreg_data),
                        .jump_address                (context_bus[11:4]),
                        .cont_addr                   (contmem_addr)
        );

        cont_mem c_memory(
                        .clk                         (clk),
                        .resetn                      (resetn),
                        .address                     (contmem_addr),
                        .context_bus                 (context_bus)
        );

        loop_reg l_data_reg(
                        .clk                         (clk),
                        .resetn                      (resetn),
                        .data_in                     (loop_data_reg_data),
                        .write_en                    (loop_data_reg_sel),
                        .control_bits                ({context_bus[12] , context_bus[0]}),
                        .loop_en                     (data_loop_en)
        );

        ARM_MEM_REG mem_reg(
                        .clk                         (clk),
                        .clkn                        (clkn),
                        .resetn                      (resetn),
                        .master_wr_en                (memreg_sel),
                        .int_loops                   (data_loop_en),
                        .function_mode               ({context_bus[21] , context_bus[0]} ),
                        .syn_bit                     (syn_bit),
```

```
                .data_in                    (memreg_data),
                .jump_address               (context_bus[20:13]),
                .data_mem_addr              (datamem_read_addr)
        );

        mem_interface d_memory(
                .write_mem                  (datamem_sel),
                .read_addr                  (datamem_read_addr),
                .write_addr                 (datamem_write_addr),
                .data_in                    (datamem_bus_in),
                .data_out1                  (datamem_bus_out[15:0]),
                .data_out2                  (datamem_bus_out[31:16]),
                .data_out3                  (datamem_bus_out[47:32]),
                .data_out4                  (datamem_bus_out[63:48]),
                .clk                        (clk),
                .resetn                     (resetn)
        );

        interrupt_interface interface1(
                .clk                        (clk),
                .resetn                     (resetn),
                .data_in                    (intr_ctrl_data_wr),
                .data_out                   (intr_ctrl_data_r),
                .sel                        (intr_ctrl_sel),
                .write                      (intr_ctrl_wr),
                .control_bits               (context_bus[1:0]),
                .IRQ                        (irq)
        );

        ASB_INTERFACE interface2(
                .dsel                       (dsel),
                .baddr                      (baddr),
                .bdata_in                   (databus_in),
                .bdata_out                  (databus_out),
                .bwrite                     (bwrite),
                .bwait                      (bwait),
                .blast                      (blast),
                .berror                     (berror),
                .r_array_data               (array_out),
                .cont_reg_sel               (contreg_sel),
                .cont_reg_data              (contreg_data),
                .mem_reg_sel                (memreg_sel),
                .mem_reg_data               (memreg_data),
                .mem_wr                     (datamem_sel),
                .mem_data                   (datamem_bus_in),
                .mem_wr_addr                (datamem_write_addr),
                .cell_sel                   (array_cell_sel),
                .cell_output_sel            (array_word_sel),
                .intr_ctrl_sel              (intr_ctrl_sel),
                .intr_ctrl_wr               (intr_ctrl_wr),
                .intr_ctrl_data_out         (intr_ctrl_data_wr),
                .intr_ctrl_data_in          (intr_ctrl_data_r),
                .loop_cont_reg_sel          (loop_cont_reg_sel),
                .loop_cont_reg_data         (loop_cont_reg_data),
                .synch_reg_sel              (synch_reg_sel),
                .synch_reg_data             (synch_reg_data),
                .loop_data_reg_sel          (loop_data_reg_sel),
                .loop_data_reg_data         (loop_data_reg_data),
                .clk                        (clk),
                .clkn                       (clkn),
                .resetn                     (resetn)
        );

endmodule
```

8.2.Κώδικας ReconfigArray

```
module ReconfigArray(
        context_mem_in,         // Data from the context memory (4x31 bits)
        data_mem_in,            // Data from the internal memory (4x16 bits)
        func_mode,              // 0 for horizontal function mode, 1 for vertical function mode
        data_out,               // Data to be read from the master
```

```
                        // (the output or a word from the RF of a cell)

        output_cell_sel,        // Select the cell, from which the master wants to read
        output_word_sel,        // Select which output of the cell, the master wants to read
        clk,                    // Input clock signal
        clkn,                   // Reverse input clock signal
        resetn                  // Input reset signal
);

//-----------------------------------------Input ports ---------------------------------------//
        input wire [123:0] context_mem_in;
        input wire [63:0] data_mem_in;
        input wire func_mode;
        input wire [3:0] output_cell_sel;    // 0 for cell0, 1 for cell1, 2 for cell2...
        input wire [2:0] output_word_sel;    // 0 for the cell output, 1-4 for the 4 words of the RF
        input wire clk;
        input wire clkn;
        input wire resetn;

// ---------------------------------- Output ports ---------------------------------------//
        output wire [15:0] data_out;

// ---------------------------------- Internal variables ---------------------------------//
        wire [30:0] context_word1;
        wire [30:0] context_word2;
        wire [30:0] context_word3;
        wire [30:0] context_word4;
        wire [15:0] data_mem_set1;
        wire [15:0] data_mem_set2;
        wire [15:0] data_mem_set3;
        wire [15:0] data_mem_set4;
        wire [15:0] RC_outputs [15:0];
        wire [1023:0] cells_RF;
        wire [79:0] sel_cell_out;           // All 5x16 outputs of the selected cell
        wire [15:0] dec_cell_sel;           // Decoded output cell selection
        wire [7:0] dec_word_sel;            // Decoded output word selection

// ---------------------------------- Code starts here... --------------------------------//
        assign context_word1 = context_mem_in[30:0];
        assign context_word2 = context_mem_in[61:31];
        assign context_word3 = context_mem_in[92:62];
        assign context_word4 = context_mem_in[123:93];

        assign data_mem_set1 = data_mem_in[15:0];
        assign data_mem_set2 = data_mem_in[31:16];
        assign data_mem_set3 = data_mem_in[47:32];
        assign data_mem_set4 = data_mem_in[63:48];

        // First row
        ReconfigCell RC0(
                .context_bus                (func_mode ? context_word1 : context_word1),
                .u_in                       (RC_outputs[12]),
                .d_in                       (RC_outputs[4]),
                .l_in                       (RC_outputs[1]),
                .r_in                       (16'b0/*RC_outputs[3]*/),
                .cc_in                      (RC_outputs[8]),
                .rr_in                      (RC_outputs[2]),
                .DM_in                      (func_mode ? data_mem_set1 : data_mem_set1),
                .RC_out                     (RC_outputs[0]),
                .RF_out                     (cells_RF[63:0]),
                .clk                        (clk),
                .clkn                       (clkn),
                .resetn                     (resetn)
        );

        ReconfigCell RC1(
                .context_bus                (func_mode ? context_word2 : context_word1),
                .u_in                       (RC_outputs[13]),
                .d_in                       (RC_outputs[5]),
                .l_in                       (RC_outputs[2]),
                .r_in                       (RC_outputs[0]),
                .cc_in                      (RC_outputs[9]),
                .rr_in                      (RC_outputs[3]),
                .DM_in                      (func_mode ? data_mem_set2 : data_mem_set1),
                .RC_out                     (RC_outputs[1]),
                .RF_out                     (cells_RF[127:64]),
                .clk                        (clk),
```

```
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC2(
        .context_bus                (func_mode ? context_word3 : context_word1),
        .u_in                       (RC_outputs[14]),
        .d_in                       (RC_outputs[6]),
        .l_in                       (RC_outputs[3]),
        .r_in                       (RC_outputs[1]),
        .cc_in                      (RC_outputs[10]),
        .rr_in                      (RC_outputs[0]),
        .DM_in                      (func_mode ? data_mem_set3 : data_mem_set1),
        .RC_out                     (RC_outputs[2]),
        .RF_out                     (cells_RF[191:128]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC3(
        .context_bus                (func_mode ? context_word4 : context_word1),
        .u_in                       (RC_outputs[15]),
        .d_in                       (RC_outputs[7]),
        .l_in                       (16'b0/*RC_outputs[0]*/),
        .r_in                       (RC_outputs[2]),
        .cc_in                      (RC_outputs[11]),
        .rr_in                      (RC_outputs[1]),
        .DM_in                      (func_mode ? data_mem_set4 : data_mem_set1),
        .RC_out                     (RC_outputs[3]),
        .RF_out                     (cells_RF[255:192]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

// Second row
ReconfigCell RC4(
        .context_bus                (func_mode ? context_word1 : context_word2),
        .u_in                       (RC_outputs[0]),
        .d_in                       (RC_outputs[8]),
        .l_in                       (RC_outputs[5]),
        .r_in                       (16'b0/*RC_outputs[7]*/),
        .cc_in                      (RC_outputs[12]),
        .rr_in                      (RC_outputs[6]),
        .DM_in                      (func_mode ? data_mem_set1 : data_mem_set2),
        .RC_out                     (RC_outputs[4]),
        .RF_out                     (cells_RF[319:256]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC5(
        .context_bus                (func_mode ? context_word2 : context_word2),
        .u_in                       (RC_outputs[1]),
        .d_in                       (RC_outputs[9]),
        .l_in                       (RC_outputs[6]),
        .r_in                       (RC_outputs[4]),
        .cc_in                      (RC_outputs[13]),
        .rr_in                      (RC_outputs[7]),
        .DM_in                      (func_mode ? data_mem_set2 : data_mem_set2),
        .RC_out                     (RC_outputs[5]),
        .RF_out                     (cells_RF[383:320]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC6(
        .context_bus                (func_mode ? context_word3 : context_word2),
        .u_in                       (RC_outputs[2]),
        .d_in                       (RC_outputs[10]),
        .l_in                       (RC_outputs[7]),
        .r_in                       (RC_outputs[5]),
        .cc_in                      (RC_outputs[14]),
        .rr_in                      (RC_outputs[4]),
```

```
        .DM_in                      (func_mode ? data_mem_set3 : data_mem_set2),
        .RC_out                     (RC_outputs[6]),
        .RF_out                     (cells_RF[447:384]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC7(
        .context_bus                (func_mode ? context_word4 : context_word2),
        .u_in                       (RC_outputs[3]),
        .d_in                       (RC_outputs[11]),
        .l_in                       (16'b0/*RC_outputs[4]*/),
        .r_in                       (RC_outputs[6]),
        .cc_in                      (RC_outputs[15]),
        .rr_in                      (RC_outputs[5]),
        .DM_in                      (func_mode ? data_mem_set4 : data_mem_set2),
        .RC_out                     (RC_outputs[7]),
        .RF_out                     (cells_RF[511:448]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

// Third row
ReconfigCell RC8(
        .context_bus                (func_mode ? context_word1 : context_word3),
        .u_in                       (RC_outputs[4]),
        .d_in                       (RC_outputs[12]),
        .l_in                       (RC_outputs[9]),
        .r_in                       (16'b0/*RC_outputs[11]*/),
        .cc_in                      (RC_outputs[0]),
        .rr_in                      (RC_outputs[10]),
        .DM_in                      (func_mode ? data_mem_set1 : data_mem_set3),
        .RC_out                     (RC_outputs[8]),
        .RF_out                     (cells_RF[575:512]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC9(
        .context_bus                (func_mode ? context_word2 : context_word3),
        .u_in                       (RC_outputs[5]),
        .d_in                       (RC_outputs[13]),
        .l_in                       (RC_outputs[10]),
        .r_in                       (RC_outputs[8]),
        .cc_in                      (RC_outputs[1]),
        .rr_in                      (RC_outputs[11]),
        .DM_in                      (func_mode ? data_mem_set2 : data_mem_set3),
        .RC_out                     (RC_outputs[9]),
        .RF_out                     (cells_RF[639:576]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC10(
        .context_bus                (func_mode ? context_word3 : context_word3),
        .u_in                       (RC_outputs[6]),
        .d_in                       (RC_outputs[14]),
        .l_in                       (RC_outputs[11]),
        .r_in                       (RC_outputs[9]),
        .cc_in                      (RC_outputs[2]),
        .rr_in                      (RC_outputs[8]),
        .DM_in                      (func_mode ? data_mem_set3 : data_mem_set3),
        .RC_out                     (RC_outputs[10]),
        .RF_out                     (cells_RF[703:640]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC11(
        .context_bus                (func_mode ? context_word4 : context_word3),
        .u_in                       (RC_outputs[7]),
        .d_in                       (RC_outputs[15]),
```

```
        .l_in                       (16'b0/*RC_outputs[8]*/),
        .r_in                       (RC_outputs[10]),
        .cc_in                      (RC_outputs[3]),
        .rr_in                      (RC_outputs[9]),
        .DM_in                      (func_mode ? data_mem_set4 : data_mem_set3),
        .RC_out                     (RC_outputs[11]),
        .RF_out                     (cells_RF[767:704]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

// Fourth row
ReconfigCell RC12(
        .context_bus                (func_mode ? context_word1 : context_word4),
        .u_in                       (RC_outputs[8]),
        .d_in                       (RC_outputs[0]),
        .l_in                       (RC_outputs[13]),
        .r_in                       (16'b0/*RC_outputs[15]*/),
        .cc_in                      (RC_outputs[4]),
        .rr_in                      (RC_outputs[14]),
        .DM_in                      (func_mode ? data_mem_set1 : data_mem_set4),
        .RC_out                     (RC_outputs[12]),
        .RF_out                     (cells_RF[831:768]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC13(
        .context_bus                (func_mode ? context_word2 : context_word4),
        .u_in                       (RC_outputs[9]),
        .d_in                       (RC_outputs[1]),
        .l_in                       (RC_outputs[14]),
        .r_in                       (RC_outputs[12]),
        .cc_in                      (RC_outputs[5]),
        .rr_in                      (RC_outputs[15]),
        .DM_in                      (func_mode ? data_mem_set2 : data_mem_set4),
        .RC_out                     (RC_outputs[13]),
        .RF_out                     (cells_RF[895:832]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC14(
        .context_bus                (func_mode ? context_word3 : context_word4),
        .u_in                       (RC_outputs[10]),
        .d_in                       (RC_outputs[2]),
        .l_in                       (RC_outputs[15]),
        .r_in                       (RC_outputs[13]),
        .cc_in                      (RC_outputs[6]),
        .rr_in                      (RC_outputs[12]),
        .DM_in                      (func_mode ? data_mem_set3 : data_mem_set4),
        .RC_out                     (RC_outputs[14]),
        .RF_out                     (cells_RF[959:896]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

ReconfigCell RC15(
        .context_bus                (func_mode ? context_word4 : context_word4),
        .u_in                       (RC_outputs[11]),
        .d_in                       (RC_outputs[3]),
        .l_in                       (16'b0/*RC_outputs[12]*/),
        .r_in                       (RC_outputs[14]),
        .cc_in                      (RC_outputs[7]),
        .rr_in                      (RC_outputs[13]),
        .DM_in                      (func_mode ? data_mem_set4 : data_mem_set4),
        .RC_out                     (RC_outputs[15]),
        .RF_out                     (cells_RF[1023:960]),
        .clk                        (clk),
        .clkn                       (clkn),
        .resetn                     (resetn)
);
```

```
        // Decode the select inputs
        assign dec_cell_sel = (1 << output_cell_sel);
        assign dec_word_sel = (1 << output_word_sel);

        assign sel_cell_out =
                (dec_cell_sel[0])   ? {cells_RF[63:0],RC_outputs[0]}      :
                ((dec_cell_sel[1])  ? {cells_RF[127:64],RC_outputs[1]}    :
                ((dec_cell_sel[2])  ? {cells_RF[191:128],RC_outputs[2]}   :
                ((dec_cell_sel[3])  ? {cells_RF[255:192],RC_outputs[3]}   :
                ((dec_cell_sel[4])  ? {cells_RF[319:256],RC_outputs[4]}   :
                ((dec_cell_sel[5])  ? {cells_RF[383:320],RC_outputs[5]}   :
                ((dec_cell_sel[6])  ? {cells_RF[447:384],RC_outputs[6]}   :
                ((dec_cell_sel[7])  ? {cells_RF[511:448],RC_outputs[7]}   :
                ((dec_cell_sel[8])  ? {cells_RF[575:512],RC_outputs[8]}   :
                ((dec_cell_sel[9])  ? {cells_RF[639:576],RC_outputs[9]}   :
                ((dec_cell_sel[10]) ? {cells_RF[703:640],RC_outputs[10]}  :
                ((dec_cell_sel[11]) ? {cells_RF[767:704],RC_outputs[11]}  :
                ((dec_cell_sel[12]) ? {cells_RF[831:768],RC_outputs[12]}  :
                ((dec_cell_sel[13]) ? {cells_RF[895:832],RC_outputs[13]}  :
                ((dec_cell_sel[14]) ? {cells_RF[959:896],RC_outputs[14]}  :
                ((dec_cell_sel[15]) ? {cells_RF[1023:960],RC_outputs[15]} :
                16'b0)))))))))))))));

        assign data_out =
                (dec_word_sel[0])  ? {sel_cell_out[15:0]}   :
                ((dec_word_sel[1]) ? {sel_cell_out[31:16]}  :
                ((dec_word_sel[2]) ? {sel_cell_out[47:32]}  :
                ((dec_word_sel[3]) ? {sel_cell_out[63:48]}  :
                ((dec_word_sel[4]) ? {sel_cell_out[79:64]}  :
                16'b0))));

endmodule
```

8.3.Κώδικας loop_reg

```
module loop_reg(
        clk,                    // Clock signal
        resetn,                 // Reset signal
        data_in                 // Data input from the ASB - sets a new value to the counter
        write_en,               // 1 to select for writing
        control_bits,           // Context input bus
        loop_en                 // Indicates if a jump must be taken (non-zero value) or not (zero)
);

//-----------------------------------------Input ports ---------------------------------------//
        input wire clk;
        input wire resetn;
        input wire [8:0] data_in;
        input wire write_en;
        input wire [1:0] control_bits;

// ---------------------------------------- Output ports ---------------------------------------//
        output wire loop_en;

// ------------------------------------ Internal variables -------------------------------------//
        reg [8:0] loop_register;
        wire value_dec;              // Indicates if the value of the reg must be decreased
        wire [8:0] new_reg_value;    // The next address to be loaded to the register
        wire [8:0] mux1_out;
        reg write_en_latch;

// ------------------------------------ Code starts here... ------------------------------------//
        always @ (negedge resetn , posedge clk) begin
                if (!resetn)
                        write_en_latch <= 1'b0;
                else
                        write_en_latch <= write_en;
        end

        // Assignment : value_dec <- bit0 && bit3 of the context bus
        assign value_dec = control_bits[0] && control_bits[1];
```

```
	// MUX1 :  selects either the previous value of the reg decreased by 1 (new loop taken)
	// or the previous value (no new loop)
	assign mux1_out = (value_dec) ? (loop_register-1) : (loop_register);

// MUX2 : selects the mux1_out (normal mode) or the address coming from the ARM (ARM writes new
data)
	assign new_reg_value = (write_en_latch) ? (data_in) : (mux1_out);

//-------------------- Assignment : loop_en <- 0 if loop_register=0 else 1 -----------------------//
	assign loop_en = (value_dec) ? ((loop_register == 9'b0) ? (1'b0) : (1'b1)) : (1'b0);

	// Positive edge-triggered register
	always @ (negedge resetn , posedge clk) begin
		if (!resetn)
			loop_register <= 9'b000000000;
		else
			loop_register <= new_reg_value;
	end

endmodule
```

8.4.Κώδικας mem_synch_reg

```
module mem_synch_reg(
	clk,			// Clock signal
	resetn,			// Reset signal
	data_in,		// Data input from the ASB - sets a new value to the register
	write_en,		// 1 to select for writing
	syn_bit
);

//-----------------------------------------Input ports ---------------------------------------//
	input wire clk;
	input wire resetn;
	input wire data_in;
	input wire write_en;

// ---------------------------------------- Output ports ---------------------------------------//
	output wire syn_bit;

// ------------------------------------ Internal variables -------------------------------------//
	reg mem_synch_reg;
	reg write_en_latch;

// ------------------------------------ Code starts here... ------------------------------------//
	assign syn_bit = mem_synch_reg;

	always @ (negedge resetn , posedge clk) begin
		if (!resetn)
			write_en_latch <= 1'b0;
		else
			write_en_latch <= write_en;
	end

	always @ ( * ) begin
		if (!resetn)
			mem_synch_reg <= 1'b0;
		else if (write_en_latch)
			mem_synch_reg <= data_in;
	end

endmodule
```

8.5.Κώδικας ARM_CONT_REG

```
module ARM_CONT_REG(
	clk,			// Input clock signal
```

```
        clkn,                   // Reverse clk signal
        resetn,                 // Input reset signal
        master_wr_en,           // 1 to enable the master to write the reg
                                // 0 to disable it and enable the normal mode
        int_loops,              // 1 to load a new address to jump to, in order to implement a loop
                                // 0 for normal mode (new address comes from the internal adder)
        function_mode,          // control signal -> (bit0 && bit2) of the context word
                                // indicates whether we are in stall or in normal mode
        syn_bit,                // Synchronization bit for data and context memory
                                // 1 to enable output, 0 to block and wait for synchronization
        data_in,                // Data from the external data bus of the system - the
                                // new offset to be written from the master
        jump_address,           // New address to jump to - implements internal loops in the module
                                // comes from the internal reg of the module
        cont_addr               // The next address in the context memory
);

//-----------------------------------------Input ports ---------------------------------------//
        input wire clk;
        input wire clkn;
        input wire resetn;
        input wire master_wr_en;
        input wire int_loops;
        input wire function_mode;
        input wire syn_bit;
        input wire [7:0] data_in;
        input wire [7:0] jump_address;

// ---------------------------------------- Output ports ---------------------------------------//
        output reg [7:0] cont_addr;

// ------------------------------------- Internal variables ------------------------------------//
        reg [7:0] new_reg_value;      // The new value to be written to the reg
        wire [7:0] adder_out;         // Previous value increased by 1 if function_mode==0
                                      // or same as the previous value if function_mode==1
        wire [7:0] mux1_out;  // Selects between the jump_address or the new value from the adder
        reg [7:0] arm_cont_reg;
        reg master_wr_en_latch;

        reg write_done;
        reg write_done_latch;
        reg write_done_latch2;

// ------------------------------------ Code starts here... ------------------------------------//
        always @ (negedge resetn , posedge clk) begin
                if (!resetn)
                        master_wr_en_latch <= 1'b0;
                else
                        master_wr_en_latch <= master_wr_en;
        end

        always @ (negedge resetn , posedge clk) begin
                if (!resetn)
                        write_done_latch <= 1'b0;
                else
                        write_done_latch <= write_done;
        end

        always @ (negedge resetn , posedge clk) begin
                if (!resetn)
                        write_done_latch2 <= 1'b0;
                else
                        write_done_latch2 <= write_done_latch;
        end

        // The adder - either increases the register or keeps it the same
        assign adder_out = (!resetn) ? 8'b0 : ((function_mode) ? (arm_cont_reg) : (arm_cont_reg +
1));

        // The MUX1 - selects the output of the adder or the external new jump address
        assign mux1_out = (!resetn) ? 8'b0 : ((int_loops) ? (jump_address) : (adder_out));

        // The new value to load to the register - either the MUX output or the data from the master
        always @ ( * ) begin
                if (!resetn) begin
                        new_reg_value <= 8'b0;
                        write_done <= 1'b0;
```

```
                end
                else if (master_wr_en_latch) begin
                        new_reg_value <= data_in;
                        write_done <= 1'b1;
                end
                else if (syn_bit && !write_done_latch2)
                        new_reg_value <= mux1_out;
                else if (syn_bit && write_done_latch2)
                        write_done <= 1'b0;
        end

        // Send out the context address on every negative edge of the clock
        always @ (negedge resetn , negedge clk) begin
                if (!resetn)
                        arm_cont_reg <= 8'b0;
                else if (syn_bit)
                        arm_cont_reg <= new_reg_value;
        end

        // The context address is the content of the register
        always @ ( * ) begin
                if (!resetn)
                        cont_addr <= 8'b0;
                else
                        cont_addr <= arm_cont_reg;
        end

endmodule
```

8.6.Κώδικας cont_mem

```
`define DATAFILE "D:/AMBA_BASED_SoC/ReconfigMicroSoc/Memories data/context_mem_data_DCT.conf"

module cont_mem(
                clk,                    // Clock signal
                resetn,                 // Reset signal
                address,                // Address from which to read - coming from the ARM_CONT_REG
                context_bus             // Context words - output bus
);

//-----------------------------------------Input ports ---------------------------------------//
        input wire [7:0] address;
        input wire clk;
        input wire resetn;

// ---------------------------------------- Output ports ---------------------------------------//
        output reg [124:0] context_bus;

// ------------------------------------- Internal variables -------------------------------------//
        reg [124:0] context_memory[0:128];

// ------------------------------------ Code starts here... ------------------------------------//
        always @ (posedge clk or negedge resetn) begin
                if (!resetn)
                        context_bus <= 125'b0;
                else
                        context_bus <= context_memory[address];
        end

        // Initialization of memory - read from the filesystem
        initial begin
                $readmemb(`DATAFILE, context_memory);
        end
endmodule
```

8.7.Κώδικας ARM_MEM_REG

```
module ARM_MEM_REG(
```

```
        clk,                    // Input clock signal
        clkn,                   // Reverse input clock signal
        resetn,                 // Input reset signal
        master_wr_en,           // 1 to enable the master to write the reg
                                // 0 to disable it and enable the normal mode
        int_loops,              // 1 to load a new address to jump to, in order to implement a loop
                                // 0 for normal mode (new address comes from the internal adder)
        function_mode,          // Control signal -> () of the context word,
                                // indicates whether we are in stall or in normal mode
        syn_bit,                // Synchronization bit for data and context memory
                                // 1 to enable output, 0 to block and wait for synchronization
        data_in,                // Data from the external data buts of the system -
                                // the new address to be written from the master
        jump_address,           // New address to jump to - implements internal loops in the module
                                // comes from the internal reg of the module
        data_mem_addr           // The current address in the data memory
);

//-----------------------------------------Input ports ----------------------------------------//
        input wire clk;
        input wire clkn;
        input wire resetn;
        input wire master_wr_en;
        input wire int_loops;
        input wire [1:0] function_mode;
        input wire syn_bit;
        input wire [7:0] data_in;
        input wire [7:0] jump_address;

// ---------------------------------------- Output ports ----------------------------------------//
        output reg [7:0] data_mem_addr;

// ---------------------------------------- Internal variables ----------------------------------//
        reg [7:0] new_reg_value;      // The new value to be written to the reg
        wire [7:0] adder_out;         // Previous value increased by 1 if function_mode==0
                                      // or same as previous value if function_mode==1
        wire [7:0] mux1_out;

        reg [7:0] arm_mem_reg;
        reg master_wr_en_latch;

        reg write_done;
        reg write_done_latch;
        reg write_done_latch2;
        reg dff_function_mode;

// ---------------------------------- Code starts here... ---------------------------------//
        always @ ( * ) begin
                if (!resetn)
                        dff_function_mode <= 1'b0;
                else if (function_mode[0])
                        dff_function_mode <= function_mode[1];
        end

        always @ (negedge resetn , posedge clk) begin
                if (!resetn)
                        master_wr_en_latch <= 1'b0;
                else
                        master_wr_en_latch <= master_wr_en;
        end

        always @ (negedge resetn , posedge clk) begin
                if (!resetn)
                        write_done_latch <= 1'b0;
                else
                        write_done_latch <= write_done;
        end

        always @ (negedge resetn , posedge clk) begin
                if (!resetn)
                        write_done_latch2 <= 1'b0;
                else
                        write_done_latch2 <= write_done_latch;
        end

        // The adder - either increases the register or keeps it the same
        assign adder_out = (!resetn) ? 8'b0 : ((dff_function_mode) ? (arm_mem_reg) : (arm_mem_reg +
```

```
1));

	// The MUX1 - selects the output of the adder or the external new jump address
	assign mux1_out = (!resetn) ? 8'b0 : ((int_loops) ? (jump_address) : (adder_out));

	// The new value to load to the register - either the MUX output or the data from the master
	always @ ( * ) begin
		if (!resetn) begin
			new_reg_value <= 8'b0;
			write_done <= 1'b0;
		end
		else if (master_wr_en_latch) begin
			new_reg_value <= data_in;
			write_done <= 1'b1;
		end
		else if (syn_bit && !write_done_latch2)
			new_reg_value <= mux1_out;
		else if (syn_bit && write_done_latch2)
			write_done <= 1'b0;
	end

	// Send out the data memory address on every positive edge of the clock
	always @ (negedge resetn , negedge clk) begin
		if (!resetn)
			arm_mem_reg <= 8'b0;
		else if (syn_bit)
			arm_mem_reg <= new_reg_value;
	end

	// The current address in the data memory is the content of the register
	always @ ( * ) begin
		if (!resetn)
			data_mem_addr <= 8'b0;
		else
			data_mem_addr <= arm_mem_reg;
	end

endmodule
```

8.8.Κώδικας mem_interface

```
`define DATAFILE1 "D:/AMBA_BASED_SoC/ReconfigMicroSoc/Memories data/memory1_data_DCT.conf"
`define DATAFILE2 "D:/AMBA_BASED_SoC/ReconfigMicroSoc/Memories data/memory2_data_DCT.conf"
`define DATAFILE3 "D:/AMBA_BASED_SoC/ReconfigMicroSoc/Memories data/memory3_data_DCT.conf"
`define DATAFILE4 "D:/AMBA_BASED_SoC/ReconfigMicroSoc/Memories data/memory4_data_DCT.conf"

module mem_interface(
	write_mem,		// Signals which of the four memories to write
				// with an 1. If all set to 0, only reading
	read_addr,		// Address from the ARM_MEM_REG for reading
	write_addr,		// Address from the ARM directly for writing
	data_in,		// Data for writing to the memory from the ARM
	data_out1,		// Data to line/column 1
	data_out2,		// Data to line/column 2
	data_out3,		// Data to line/column 3
	data_out4,		// Data to line/column 4
	clk,
	resetn
);

//-----------------------------------------Input ports ---------------------------------------//
	input wire [3:0] write_mem;		// 0000 -> all read
						// 0001 -> write MEM1, read all others
						// 0010 -> write MEM2, read all others
						// 0100 -> write MEM3, read all others
						// 1000 -> write MEM4, read all others
	input wire [7:0] read_addr;
	input wire [7:0] write_addr;
	input wire [15:0] data_in;
	input wire clk;
	input wire resetn;

// ------------------------------------ Output ports ----------------------------------------//
```

```
        output wire [15:0] data_out1;
        output wire [15:0] data_out2;
        output wire [15:0] data_out3;
        output wire [15:0] data_out4;

// ---------------------------------- Code starts here... ------------------------------------//
        DataMem MEM1(
                .data_in                    (data_in),
                .data_out                   (data_out1),
                .address                    ( write_mem[0] ? write_addr : read_addr ),
                .read_write                 (write_mem[0]),
                .clk                        (clk),
                .resetn                     (resetn)
        );

        DataMem MEM2(
                .data_in                    (data_in),
                .data_out                   (data_out2),
                .address                    ( write_mem[1] ? write_addr : read_addr ),
                .read_write                 (write_mem[1]),
                .clk                        (clk),
                .resetn                     (resetn)
        );

        DataMem MEM3(
                .data_in                    (data_in),
                .data_out                   (data_out3),
                .address                    ( write_mem[2] ? write_addr : read_addr ),
                .read_write                 (write_mem[2]),
                .clk                        (clk),
                .resetn                     (resetn)
        );

        DataMem MEM4(
                .data_in                    (data_in),
                .data_out                   (data_out4),
                .address                    ( write_mem[3] ? write_addr : read_addr ),
                .read_write                 (write_mem[3]),
                .clk                        (clk),
                .resetn                     (resetn)
        );

        //initialization of memories
        initial begin
                $readmemh(`DATAFILE1, MEM1.data_mem);
                $readmemh(`DATAFILE2, MEM2.data_mem);
                $readmemh(`DATAFILE3, MEM3.data_mem);
                $readmemh(`DATAFILE4, MEM4.data_mem);
        end

endmodule
```

8.9.Κώδικας DataMem

```
module DataMem(
        data_in,                // input data for writing to the memory
        data_out,               // output data read from the memory
        address,                // the address for reading from or writing to
        read_write,             // 0 to read from memory, 1 to write to memory
        clk,                    // the clock input
        resetn                  // the reset input
);

//-----------------------------------------Input ports ---------------------------------------//
        input wire [15:0] data_in;
        input wire [7:0] address;
        input wire read_write;
        input wire clk;
        input wire resetn;

// ------------------------------------ Output ports ----------------------------------------//
        output reg [15:0] data_out;
```

```
// ------------------------------------ Internal variables ------------------------------------//
	reg[15:0] data_mem[0:129];
	integer i=0;

// ------------------------------------ Code starts here... ------------------------------------//
	always @ (posedge clk or negedge resetn) begin
		if (read_write == 0)
			data_out <= data_mem[address];
		else if (read_write == 1)
			data_mem[address] <= data_in;
	end
endmodule
```

8.10.Κώδικας interrupt_interface

```
module interrupt_interface(
	clk,
	resetn,
	data_in,		// Data in bus
	data_out,		// Data out bus
	sel,			// Select signal
	write,			// 1 for writing, 0 for reading
	control_bits,		// Bits from the Context  bus
	IRQ			// Request interrupt from the ARM (positive logic)
);

//-----------------------------------------Input ports ------------------------------------//
	input wire clk;
	input wire resetn;
	input wire [1:0] data_in;
	input wire write;
	input wire sel;
	input wire [1:0] control_bits;

// ------------------------------------ Output ports ------------------------------------//
	output reg [1:0] data_out;
	output wire IRQ;

// ------------------------------------ Internal variables ------------------------------------//
	reg [1:0] interrupt_reg;
	reg write_latch;
	reg sel_latch;

// ------------------------------------ Code starts here... ------------------------------------//

	always @ (negedge resetn , posedge clk) begin
		if (!resetn)
			write_latch <= 1'b0;
		else
			write_latch <= write;
	end

	always @ (negedge resetn , posedge clk) begin
		if (!resetn)
			sel_latch <= 1'b0;
		else
			sel_latch <= sel;
	end

	always @ ( * /*posedge clk , negedge resetn*/) begin
		if (!resetn) begin
			interrupt_reg <= 2'b0;
			data_out <= 2'b0;
		end
		else if (write_latch && sel_latch)
			interrupt_reg <= data_in[1:0];
		else if (!write && sel)
			data_out <= interrupt_reg;
	end

	assign IRQ = control_bits[1] & control_bits[0];
```

```
endmodule
```

8.11.Κώδικας ASB_INTERFACE

```
module ASB_INTERFACE(
        dsel,                           // ASB Select signal
        baddr,                          // ASB Address signal
        bdata_in,                       // ASB Data input signal
        bdata_out,                      // ASB Data output singal
        bwrite,                         // ASB Write signal
        bwait,                          // ASB Wait signal
        blast,                          // ASB Last signal
        berror,                         // ASB Error signal
        r_array_data,                   // Data from the reconfigurable array
        cont_reg_sel,                   // ARM_CONT_REG select
        cont_reg_data,                  // Data for the ARM_CONT_REG
        mem_reg_sel,                    // ARM_MEM_REG select
        mem_reg_data,                   // Data for the ARM_MEM_REG
        mem_wr,                         // Selects data memory to write
        mem_data,                       // Data to write to the data memory
        mem_wr_addr,                    // Address of the data memory to write to
        cell_sel,                       // Select cell from the reconfig array
        cell_output_sel,                // Select output from the cell
        intr_ctrl_sel,                  // Select the interrupt controller
        intr_ctrl_wr,                   // 1 to write to the interrupt controller, 0 for reading
        intr_ctrl_data_out,             // Data to write to the interrupt controller
        intr_ctrl_data_in,              // Data from the interrupt controller
        loop_cont_reg_sel,              // Select the context loop register for writing
        loop_cont_reg_data,             // Data to write to the context loop register
        synch_reg_sel,                  // Select the synchronization register for writing
        synch_reg_data,                 // Data to write to the synchronization register
        loop_data_reg_sel,              // Select the data loop register for writing
        loop_data_reg_data,             // Data to write to the data loop register
        clk,
        clkn,
        resetn
);

//-----------------------------------------Input ports ---------------------------------------//
        input wire dsel;
        input wire [31:0] baddr;
        input wire [31:0] bdata_in;
        input wire bwrite;
        input wire [15:0] r_array_data;
        input wire clk;
        input wire clkn;
        input wire resetn;
        input wire [1:0] intr_ctrl_data_in;

// ------------------------------------ Output ports ---------------------------------------//
        output wire [31:0] bdata_out;
        output reg bwait;
        output reg blast;
        output reg berror;
        output reg cont_reg_sel;
        output reg [7:0] cont_reg_data;
        output reg mem_reg_sel;
        output reg [7:0] mem_reg_data;
        output reg [3:0] mem_wr;        // Selects data memory to write
                                        // 0001 -> memory 1
                                        // 0010 -> memory 2
                                        // 0100 -> memory 3
                                        // 1000 -> memory 4
        output reg [15:0] mem_data;
        output reg [7:0] mem_wr_addr;
        output reg [3:0] cell_sel;
        output reg [2:0] cell_output_sel;
        output reg intr_ctrl_sel;
        output reg intr_ctrl_wr;
        output reg [1:0] intr_ctrl_data_out;
        output reg loop_cont_reg_sel;
        output reg [8:0] loop_cont_reg_data;
```

```
        output reg synch_reg_sel;
        output reg synch_reg_data;
        output reg loop_data_reg_sel;
        output reg [8:0] loop_data_reg_data;

// ------------------------------------ Internal variables ----------------------------------------//
        parameter ARM_CONT_REG = 4'b0000;
        parameter ARM_MEM_REG = 4'b0001;
        parameter DATA_MEM = 4'b0010;
        parameter R_ARRAY = 4'b0011;
        parameter INTR_CTRL = 4'b0100;
        parameter LOOP_CONT_REG = 4'b0101;
        parameter MEM_SYNCH_REG = 4'b0110;
        parameter LOOP_DATA_REG = 4'b0111;

        parameter STATE1 = 1'b0;
        parameter STATE2 = 1'b1;

        reg dsel_latch;
        reg [31:0] baddr_latch;
        reg [31:0] bdata_in_latch;
        reg bwrite_latch;
        reg [15:0] r_array_data_latch;
        reg [1:0] intr_ctrl_data_latch;

        reg dsel_latch2;
        reg dsel_latch3;
        reg bwrite_latch2;

        reg state;

        wire [3:0] slave_comp_sel;    // which slave component to select
                                      // 0000 -> ARM_CONT_REG (hex for baddr -> 0)
                                      // 0001 -> ARM_MEM_REG (hex for baddr -> 1)
                                      // 0010 -> DATA MEMORY (hex for baddr -> 2)
                                      // 0011 -> RECONFIGURABLE ARRAY (hex for baddr -> 3)
                                      // 0100 -> INTERRUPT CONTROLLER (hex for baddr -> 4)
                                      // 0101 -> LOOP CONTEXT REGISTER (hex for baddr -> 5)
                                      // 0110 -> MEM_SYNCH_REG (hex for baddr -> 6)
                                      // 0111 -> LOOP DATA REGISTER (hex for baddr -> 7)

        wire [1:0] data_mem_sel;      // which data memory to select
                                      // 00 -> MEMORY 1
                                      // 01 -> MEMORY 2
                                      // 10 -> MEMORY 3
                                      // 11 -> MEMORY 4

// ------------------------------------ Code starts here... ---------------------------------------//
        assign slave_comp_sel = (baddr_latch[31:28]==4'b0101) ? (baddr_latch[27:24]) : (4'b1111);
        assign data_mem_sel = baddr_latch[23:22];
        assign bdata_out =
        (dsel_latch3 && !bwrite_latch && (slave_comp_sel == R_ARRAY)) ? ({16'b0,r_array_data}) :
        ((dsel && !bwrite_latch && (slave_comp_sel == INTR_CTRL)) ? ({30'b0,intr_ctrl_data_latch}) :
        (32'b0));

        // latch signals
        always @ (dsel) begin
                dsel_latch <= dsel;
        end

        always @ (posedge clkn or negedge resetn) begin
                if (!resetn) begin
                        baddr_latch <= 32'b0;
                        bdata_in_latch <= 32'b0;
                        bwrite_latch <= 1'b0;
                        r_array_data_latch <= 16'b0;
                        intr_ctrl_data_latch <= 2'b0;
                        dsel_latch2 <= 1'b0;
                        dsel_latch3 <= 1'b0;
                        bwrite_latch2 <= 1'b0;
                end
                else begin
                        baddr_latch <= baddr;
                        bdata_in_latch <= bdata_in;
                        bwrite_latch <= bwrite;
                        r_array_data_latch <= r_array_data;
                        intr_ctrl_data_latch <= intr_ctrl_data_in;
```

```
                dsel_latch2 <= dsel_latch;
                dsel_latch3 <= dsel_latch2;
                bwrite_latch2 <= bwrite_latch;
        end
end

// function depends on the component selected
always @ ( * ) begin
        if (!resetn) begin
                cont_reg_sel <= 1'b0;
                cont_reg_data <= 8'b0;
                mem_reg_sel <= 1'b0;
                mem_reg_data <= 8'b0;
                mem_wr <= 4'b0;
                mem_data <= 16'b0;
                mem_wr_addr <= 8'b0;
                intr_ctrl_sel <= 1'b0;
                intr_ctrl_wr <= 1'b0;
                intr_ctrl_data_out <= 2'b0;
                loop_cont_reg_sel <= 1'b0;
                loop_cont_reg_data <= 9'b0;
                synch_reg_sel <= 1'b0;
                synch_reg_data <= 1'b0;
                loop_data_reg_sel <= 1'b0;
                loop_data_reg_data <= 9'b0;
        end
        else begin
                case (slave_comp_sel)
                ARM_CONT_REG :
                begin
                        mem_reg_sel <= 1'b0;
                        mem_wr <= 4'b0000;
                        intr_ctrl_sel <= 1'b0;
                        loop_cont_reg_sel <= 1'b0;
                        synch_reg_sel <= 1'b0;
                        loop_data_reg_sel <= 1'b0;

                        if (bwrite_latch2 && dsel_latch3) begin
                                cont_reg_sel <= 1'b1;
                                cont_reg_data <= bdata_in_latch[7:0];
                        end
                        else
                                cont_reg_sel <= 1'b0;
                end

                ARM_MEM_REG :
                begin
                        cont_reg_sel <= 1'b0;
                        mem_wr <= 4'b0000;
                        intr_ctrl_sel <= 1'b0;
                        loop_cont_reg_sel <= 1'b0;
                        synch_reg_sel <= 1'b0;
                        loop_data_reg_sel <= 1'b0;

                        if (bwrite_latch2 && dsel_latch3) begin
                                mem_reg_sel <= 1'b1;
                                mem_reg_data <= bdata_in_latch[7:0];
                        end
                        else
                                mem_reg_sel <= 1'b0;
                end

                DATA_MEM :
                begin
                        cont_reg_sel <= 1'b0;
                        mem_reg_sel <= 1'b0;
                        intr_ctrl_sel <= 1'b0;
                        loop_cont_reg_sel <= 1'b0;
                        synch_reg_sel <= 1'b0;
                        loop_data_reg_sel <= 1'b0;

                        if (bwrite_latch2 && dsel_latch3) begin
                                mem_data <= bdata_in_latch[15:0];
                                mem_wr_addr <= baddr_latch[21:15];

                                case (data_mem_sel)
                                        2'b00 : mem_wr <= 4'b0001;
```

```
                        2'b01 : mem_wr <= 4'b0010;
                        2'b10 : mem_wr <= 4'b0100;
                        2'b11 : mem_wr <= 4'b1000;
                endcase
        end
        else
                mem_wr <= 4'b0000;
end

R_ARRAY :
begin
        cont_reg_sel <= 1'b0;
        mem_reg_sel <= 1'b0;
        mem_wr <= 4'b0000;
        intr_ctrl_sel <= 1'b0;
        loop_cont_reg_sel <= 1'b0;
        synch_reg_sel <= 1'b0;
        loop_data_reg_sel <= 1'b0;

        if (!bwrite_latch2 && dsel_latch3) begin
                cell_sel <=    baddr_latch[23:20];
                cell_output_sel <= baddr_latch[19:17];
        end
end

INTR_CTRL :
begin
        cont_reg_sel <= 1'b0;
        mem_reg_sel <= 1'b0;
        mem_wr <= 4'b0000;
        loop_cont_reg_sel <= 1'b0;
        synch_reg_sel <= 1'b0;
        loop_data_reg_sel <= 1'b0;

        if (bwrite_latch2 && dsel_latch3) begin
                intr_ctrl_sel <= 1'b1;
                intr_ctrl_wr <= 1'b1;
                intr_ctrl_data_out <= bdata_in_latch[1:0];
        end
        else if (!bwrite_latch2 && dsel_latch3) begin
                intr_ctrl_sel <= 1'b1;
                intr_ctrl_wr <= 1'b0;
        end
        else intr_ctrl_sel <= 1'b0;
end

LOOP_CONT_REG :
begin
        cont_reg_sel <= 1'b0;
        mem_reg_sel <= 1'b0;
        mem_wr <= 4'b0000;
        intr_ctrl_sel <= 1'b0;
        synch_reg_sel <= 1'b0;
        loop_data_reg_sel <= 1'b0;

        if (bwrite_latch2 && dsel_latch3) begin
                loop_cont_reg_sel <= 1'b1;
                loop_cont_reg_data <= bdata_in_latch[8:0];
        end
        else
                loop_cont_reg_sel <= 1'b0;
end

MEM_SYNCH_REG :
begin
        cont_reg_sel <= 1'b0;
        mem_reg_sel <= 1'b0;
        mem_wr <= 4'b0000;
        intr_ctrl_sel <= 1'b0;
        loop_cont_reg_sel <= 1'b0;
        loop_data_reg_sel <= 1'b0;

        if (bwrite_latch2 && dsel_latch3) begin
                synch_reg_sel <= 1'b1;
                synch_reg_data <= bdata_in_latch[0];
        end
        else
```

```
                                synch_reg_sel <= 1'b0;
                        end

                        LOOP_DATA_REG :
                        begin
                                cont_reg_sel <= 1'b0;
                                mem_reg_sel <= 1'b0;
                                mem_wr <= 4'b0000;
                                intr_ctrl_sel <= 1'b0;
                                synch_reg_sel <= 1'b0;
                                loop_cont_reg_sel <= 1'b0;

                                if (bwrite_latch2 && dsel_latch3) begin
                                        loop_data_reg_sel <= 1'b1;
                                        loop_data_reg_data <= bdata_in_latch[8:0];
                                end
                                else
                                        loop_data_reg_sel <= 1'b0;
                        end

                        default :
                        begin
                                cont_reg_sel <= 1'b0;
                                mem_reg_sel <= 1'b0;
                                mem_wr <= 4'b0000;
                                intr_ctrl_sel <= 1'b0;
                                loop_cont_reg_sel <= 1'b0;
                                synch_reg_sel <= 1'b0;
                                loop_data_reg_sel <= 1'b0;
                        end
                        endcase
                end
        end

        // Transfer response of the slave
        always @ (posedge clkn , negedge resetn) begin
                if (!resetn) begin
                        {bwait,blast,berror} <= 3'b000;
                        state <= STATE1;
                end
                else begin
                        if (dsel_latch) begin
                                if (state == STATE1) begin                  // STATE1
                                        {bwait,blast,berror} <= 3'b100;     // WAIT
                                        state <= STATE2;                    // change state
                                end
                                else begin                                  // STATE2
                                        {bwait,blast,berror} <= 3'b000;     // DONE
                                        state <= STATE1;                    // change state
                                end
                        end
                end
        end

endmodule
```

8.12.Κώδικας ReconfigCell

```
`define CONTEXT_WORD 31
`define DATA_WORD 16
`define OUT_WORD 32

module ReconfigCell(
        context_bus,          // Data from the context bus (memory)
        u_in,                 // Data from the cell up in the same column
        d_in,                 // Data from the cell down in the same column
        l_in,                 // Data from the cell left in the same row
        r_in,                 // Data from the cell right in the same row
        cc_in,                // Data from the cell of the quadrant
                              // two places below or above in the same column
        rr_in,                // Data from the cell of the quadrant
                              // two places left or right in the same row
        DM_in,                // Data from the internal data memory (bus)
```

```
        RC_out,                 // The output of the cell
        RF_out,                 // The RF content, available for the master to read
        clk,                    // Input clock signal
        clkn,                   // Reverse input clock signal
        resetn                  // Input reset signal
);

//-----------------------------------------Input ports ---------------------------------------//
        input wire [(`CONTEXT_WORD)-1:0] context_bus;
        input wire [(`DATA_WORD)-1:0] u_in;
        input wire [(`DATA_WORD)-1:0] d_in;
        input wire [(`DATA_WORD)-1:0] l_in;
        input wire [(`DATA_WORD)-1:0] r_in;
        input wire [(`DATA_WORD)-1:0] cc_in;
        input wire [(`DATA_WORD)-1:0] rr_in;
        input wire [(`DATA_WORD)-1:0] DM_in;
        input wire clk;
        input wire clkn;
        input wire resetn;

// ---------------------------------------- Output ports ---------------------------------------//
        output wire [(`DATA_WORD)-1:0] RC_out;
        output wire [63:0] RF_out;

// ------------------------------------- Internal variables ------------------------------------//
        wire [(`CONTEXT_WORD)-1:0] context_word;    // latched context_word

        // latched inputs
        reg [(`DATA_WORD)-1:0] lat_u_in;
        reg [(`DATA_WORD)-1:0]lat_d_in;
        reg [(`DATA_WORD)-1:0]lat_l_in;
        reg [(`DATA_WORD)-1:0]lat_r_in;
        reg [(`DATA_WORD)-1:0]lat_cc_in;
        reg [(`DATA_WORD)-1:0]lat_rr_in;
        reg [(`DATA_WORD)-1:0] lat_DM_in;

        // wires to connect the input MUX to the ALU_MUL module
        wire [(`DATA_WORD)-1:0]       ALU_MUL_in_a;
        wire [(`DATA_WORD)-1:0] ALU_MUL_in_b;

        // output of the ALU_MUL module - input to the shifter
        wire [(`OUT_WORD)-1:0] ALU_MUL_out;

        // output of the shifter - input to the SFT_RF register
        wire [(`OUT_WORD)-1:0] SFT_out;

        // output of the SFT_RF register - input to the register file
        wire [(`OUT_WORD)-1:0] RF_data_in;

        // wire to the output of the register file - connect with the input MUX
        // and the output of the cell
        wire [63:0] RF_data_out;

// ------------------------------------ Code starts here... ------------------------------------//
        always @ (posedge clkn or negedge resetn) begin
                if (!resetn) begin
                        lat_u_in <= 'b0;
                end
                else begin
                        lat_u_in <= u_in;
                end
        end

                always @ (posedge clkn or negedge resetn) begin
                if (!resetn) begin
                        lat_d_in <= 'b0;
                end
                else begin
                        lat_d_in <= d_in;
                end
        end

        always @ (posedge clkn or negedge resetn) begin
                if (!resetn) begin
                        lat_l_in <= 'b0;
                end
                else begin
```

```
                lat_l_in <= l_in;
        end
end

always @ (posedge clkn or negedge resetn) begin
        if (!resetn) begin
                lat_r_in <= 'b0;
        end
        else begin
                lat_r_in <= r_in;
        end
end

always @ (posedge clkn or negedge resetn) begin
        if (!resetn) begin
                lat_cc_in <= 'b0;
        end
        else begin
                lat_cc_in <= cc_in;
        end
end

always @ (posedge clkn or negedge resetn) begin
        if (!resetn) begin
                lat_rr_in <= 'b0;
        end
        else begin
                lat_rr_in <= rr_in;
        end
end

always @ (posedge clkn or negedge resetn) begin
        if (!resetn) begin
                lat_DM_in <= 'b0;
        end
        else begin
                lat_DM_in <= DM_in;
        end
end

ContextReg CONT_REG(
        .context_bus                (context_bus),
        .context_word               (context_word),
        .clkn                       (clkn),
        .resetn                     (resetn)
);

InputMUX MUX_A(
        .din0                       (lat_u_in),
        .din1                       (lat_d_in),
        .din2                       (lat_l_in),
        .din3                       (lat_r_in),
        .din4                       (lat_cc_in),
        .din5                       (lat_rr_in),
        .din6                       (lat_DM_in),
        .din7                       (RF_data_out[15:0]),
        .din8                       (RF_data_out[31:16]),
        .din9                       (RF_data_out[47:32]),
        .din10                      (RF_data_out[63:48]),
        .din11                      (),
        .din12                      (),
        .din13                      (),
        .din14                      (),
        .din15                      (),
        .dout                       (ALU_MUL_in_a),
        .sel                        (context_word[21:18]),
        .enable_n                   (context_word[0])
);

InputMUX MUX_B(
        .din0                       (lat_u_in),
        .din1                       (lat_d_in),
        .din2                       (lat_l_in),
        .din3                       (lat_r_in),
        .din4                       (lat_cc_in),
        .din5                       (lat_rr_in),
        .din6                       (lat_DM_in),
```

```
                .din7                           (RF_data_out[15:0]),
                .din8                           (RF_data_out[31:16]),
                .din9                           (RF_data_out[47:32]),
                .din10                          (RF_data_out[63:48]),
                .din11                          (),
                .din12                          (),
                .din13                          (),
                .din14                          (),
                .din15                          (),
                .dout                           (ALU_MUL_in_b),
                .sel                            (context_word[17:14]),
                .enable_n                       (context_word[0])
        );

        ALU_MUL AluMul(
                .a                              (ALU_MUL_in_a),
                .b                              (ALU_MUL_in_b),
                .const                          ({8'b0,context_word[8:1]}),
                .control                        (context_word[13:11]),
                .sel_out                        (context_word[10]),
                .sel_MUXin                      (context_word[9]),
                .result                         (ALU_MUL_out),
                .zero                           (),
                .overflow                       ()
        );

        VarShifter SFT(
                .shift_control                  (context_word[27]),
                .shift_count                    (context_word[26:22]),
                .din_data                       (ALU_MUL_out),
                .sft_out                        (SFT_out)
        );

        SftRf_Reg SRF_REG(
                .SFT_out                        (SFT_out),
                .RF_in                          (RF_data_in),
                .clkn                           (clkn),
                .resetn                         (resetn)
        );

        RegFile REG_FILE(
                .port_out                       (RF_data_out),
                .wr_addr                        (context_word[29:28]),
                .wr_data                        (RF_data_in[15:0]),
                .wr_en                          (context_word[30]),
                .clk                            (clk),
                .resetn                         (resetn)
        );

        // Assign the output signals
        assign RC_out = SFT_out[15:0];
        assign RF_out = RF_data_out;

endmodule
```

8.13.Κώδικας ContextReg

```
`define N 31                    // Length of the context word

module ContextReg(
        context_bus,            // Input from the context bus
        context_word,           // The latched context word,output of the register
        clkn,                   // negative clock pulse
        resetn                  // reset signal
);

//-----------------------------------------Input ports ---------------------------------------//
input wire [(`N)-1:0] context_bus;
input wire clkn;
input wire resetn;

// ---------------------------------- Output ports ----------------------------------------//
output reg [(`N)-1:0] context_word;
```

```
// ---------------------------------- Code starts here... ------------------------------------//
always @ (posedge clkn or negedge resetn)   begin
        if (!resetn) begin
                context_word <= 'b0;
        end
        else begin
                context_word <= context_bus;
        end
end
endmodule
```

8.14.Κώδικας InputMUX

```
module InputMUX(
        din0,                   // MUX input 0
        din1,                   // MUX input 1
        din2,                   // MUX input 2
        din3,                   // MUX input 3
        din4,                   // MUX input 4
        din5,                   // MUX input 5
        din6,                   // MUX input 6
        din7,                   // MUX input 7
        din8,                   // MUX input 8
        din9,                   // MUX input 9
        din10,                  // MUX input 10
        din11,                  // MUX input 11
        din12,                  // MUX input 12
        din13,                  // MUX input 13
        din14,                  // MUX input 14
        din15,                  // MUX input 15
        dout,                   // MUX output 16-bit length
        sel,                    // MUX select input
        enable_n                // MUX enable input / set to 0 to enable
);

//-----------------------------------------Input ports ---------------------------------------//
        input wire [15:0] din0;
        input wire [15:0] din1;
        input wire [15:0] din2;
        input wire [15:0] din3;
        input wire [15:0] din4;
        input wire [15:0] din5;
        input wire [15:0] din6;
        input wire [15:0] din7;
        input wire [15:0] din8;
        input wire [15:0] din9;
        input wire [15:0] din10;
        input wire [15:0] din11;
        input wire [15:0] din12;
        input wire [15:0] din13;
        input wire [15:0] din14;
        input wire [15:0] din15;
        input wire [3:0] sel;
        input wire enable_n;

// ---------------------------------------- Output ports ---------------------------------------//
        output wire [15:0] dout;

// ------------------------------------ Internal variables -------------------------------------//
        wire [15:0] dec_sel;          // decoded select input

// ---------------------------------- Code starts here... ------------------------------------//
        assign dec_sel = (1 << sel);

        assign dout =  (enable_n) ? 16'b0 :
                       ((dec_sel[0])  ? din0 :
                       ((dec_sel[1])  ? din1 :
                       ((dec_sel[2])  ? din2 :
                       ((dec_sel[3])  ? din3 :
                       ((dec_sel[4])  ? din4 :
                       ((dec_sel[5])  ? din5 :
                       ((dec_sel[6])  ? din6 :
```

```
                        ((dec_sel[7])  ? din7 :
                        ((dec_sel[8])  ? din8 :
                        ((dec_sel[9])  ? din9 :
                        ((dec_sel[10]) ? din10 :
                        ((dec_sel[11]) ? din11 :
                        ((dec_sel[12]) ? din12 :
                        ((dec_sel[13]) ? din13 :
                        ((dec_sel[14]) ? din14 :
                        ((dec_sel[15]) ? din15 :
                        16'b0)))))))))))))));

endmodule
```

8.15.Κώδικας VarShifter

```
module VarShifter(
        shift_control,          // direction of shift
                                // 0 for shift left, 1 for shift right
        shift_count,
        din_data,               //      data in
        sft_out                 //      data out
);


//-----------------------------------------Input ports ---------------------------------------//
input wire shift_control;
input wire [4:0] shift_count;
input wire [31:0] din_data;

// ---------------------------------------- Output ports ---------------------------------------//
output reg [31:0] sft_out;

// ------------------------------------ Code starts here... ------------------------------------//
always @ (shift_control or shift_count or din_data) begin : shift_func
        case ({shift_control , shift_count})

        //shift left
        6'b000000 : sft_out = din_data;
        6'b000001 : sft_out = {din_data[30:0] , 1'b0};
        6'b000010 : sft_out = {din_data[29:0] , 2'b0};
        6'b000011 : sft_out = {din_data[28:0] , 3'b0};
        6'b000100 : sft_out = {din_data[27:0] , 4'b0};
        6'b000101 : sft_out = {din_data[26:0] , 5'b0};
        6'b000110 : sft_out = {din_data[25:0] , 6'b0};
        6'b000111 : sft_out = {din_data[24:0] , 7'b0};
        6'b001000 : sft_out = {din_data[23:0] , 8'b0};
        6'b001001 : sft_out = {din_data[22:0] , 9'b0};
        6'b001010 : sft_out = {din_data[21:0] , 10'b0};
        6'b001011 : sft_out = {din_data[20:0] , 11'b0};
        6'b001100 : sft_out = {din_data[19:0] , 12'b0};
        6'b001101 : sft_out = {din_data[18:0] , 13'b0};
        6'b001110 : sft_out = {din_data[17:0] , 14'b0};
        6'b001111 : sft_out = {din_data[16:0] , 15'b0};
        6'b010000 : sft_out = {din_data[15:0] , 16'b0};
        6'b010001 : sft_out = {din_data[14:0] , 17'b0};
        6'b010010 : sft_out = {din_data[13:0] , 18'b0};
        6'b010011 : sft_out = {din_data[12:0] , 19'b0};
        6'b010100 : sft_out = {din_data[11:0] , 20'b0};
        6'b010101 : sft_out = {din_data[10:0] , 21'b0};
        6'b010110 : sft_out = {din_data[9:0] , 22'b0};
        6'b010111 : sft_out = {din_data[8:0] , 23'b0};
        6'b011000 : sft_out = {din_data[7:0] , 24'b0};
        6'b011001 : sft_out = {din_data[6:0] , 25'b0};
        6'b011010 : sft_out = {din_data[5:0] , 26'b0};
        6'b011011 : sft_out = {din_data[4:0] , 27'b0};
        6'b011100 : sft_out = {din_data[3:0] , 28'b0};
        6'b011101 : sft_out = {din_data[2:0] , 29'b0};
        6'b011110 : sft_out = {din_data[1:0] , 30'b0};
        6'b011111 : sft_out = {din_data[0] , 31'b0};

        //shift right
        6'b100000 : sft_out = din_data;
        6'b100001 : sft_out = {1'b0 , din_data[31:1]};
```

```
        6'b100010 : sft_out = {2'b0 , din_data[31:2]};
        6'b100011 : sft_out = {3'b0 , din_data[31:3]};
        6'b100100 : sft_out = {4'b0 , din_data[31:4]};
        6'b100101 : sft_out = {5'b0 , din_data[31:5]};
        6'b100110 : sft_out = {6'b0 , din_data[31:6]};
        6'b100111 : sft_out = {7'b0 , din_data[31:7]};
        6'b101000 : sft_out = {8'b0 , din_data[31:8]};
        6'b101001 : sft_out = {9'b0 , din_data[31:9]};
        6'b101010 : sft_out = {10'b0 , din_data[31:10]};
        6'b101011 : sft_out = {11'b0 , din_data[31:11]};
        6'b101100 : sft_out = {12'b0 , din_data[31:12]};
        6'b101101 : sft_out = {13'b0 , din_data[31:13]};
        6'b101110 : sft_out = {14'b0 , din_data[31:14]};
        6'b101111 : sft_out = {15'b0 , din_data[31:15]};
        6'b110000 : sft_out = {16'b0 , din_data[31:16]};
        6'b110001 : sft_out = {17'b0 , din_data[31:17]};
        6'b110010 : sft_out = {18'b0 , din_data[31:18]};
        6'b110011 : sft_out = {19'b0 , din_data[31:19]};
        6'b110100 : sft_out = {20'b0 , din_data[31:20]};
        6'b110101 : sft_out = {21'b0 , din_data[31:21]};
        6'b110110 : sft_out = {22'b0 , din_data[31:22]};
        6'b110111 : sft_out = {23'b0 , din_data[31:23]};
        6'b111000 : sft_out = {24'b0 , din_data[31:24]};
        6'b111001 : sft_out = {25'b0 , din_data[31:25]};
        6'b111010 : sft_out = {26'b0 , din_data[31:26]};
        6'b111011 : sft_out = {27'b0 , din_data[31:27]};
        6'b111100 : sft_out = {28'b0 , din_data[31:28]};
        6'b111101 : sft_out = {29'b0 , din_data[31:29]};
        6'b111110 : sft_out = {30'b0 , din_data[31:30]};
        6'b111111 : sft_out = {31'b0 , din_data[31]};

        default : sft_out = 32'bx;
        endcase
end

endmodule
```

8.16.Κώδικας SftRf_Reg

```
`define NF 32                   // Length of the register

module SftRf_Reg(
        SFT_out,                // Output of the shifter-input for the reg
        RF_in,                  // Input for the register file-output of the reg
        clkn,                   // The reverse clock pulse
        resetn                  // The reset signal
);

//-----------------------------------------Input ports ---------------------------------------//
input wire [(`NF)-1:0] SFT_out;
input wire clkn;
input wire resetn;

// ---------------------------------------- Output ports ---------------------------------------//
output reg [(`NF)-1:0] RF_in;

// ---------------------------------- Code starts here... ---------------------------------------//
always @ (posedge clkn or negedge resetn) begin
        if(!resetn) begin
                RF_in <= 'b0;
        end
        else begin
                RF_in <= SFT_out;
        end
end
endmodule
```

8.17.Κώδικας RegFile

```
module RegFile(
	port_out,		// the output read port
	wr_addr,		// the write address
	wr_data,		// the write data
	wr_en,			// the write enable input, 1 to enable
	clk,			// the clock input
	resetn			// the reset input
);

//---------------------------------------- Parameters ----------------------------------------//
	parameter N = 16;	// length of a word
	parameter L = 4;	// number of the registers
	parameter M = 2;	// length of an address (2^M = L)

//----------------------------------------Input ports ----------------------------------------//
	input wire [N-1:0] wr_data;
	input wire [M-1:0] wr_addr;
	input wire wr_en;
	input wire clk;
	input wire resetn;

// ---------------------------------------- Output ports ----------------------------------------//
	output wire [N*L-1:0] port_out;		// all registers can be read from outside
						// an external MUX is needed to choose one only

// ---------------------------------- Internal variables ----------------------------------//
	reg [N-1:0] reg_file [0:L-1];		//definition of the reg file
	integer i;
	genvar j;

// ---------------------------------- Code starts here... ----------------------------------//
	generate
		for(j=0;j<L;j=j+1) begin : loop1
			assign port_out[((j*N)+N-1):(j*N)]  = reg_file[j];
		end
	endgenerate

	always @ (posedge clk or negedge resetn) begin
		if(!resetn) begin		// in case a reset is needed
			for(i=0;i<L;i=i+1) begin : loop2
				reg_file[i] <= 'b0;
			end
		end
		else if(wr_en) begin		// write only if wr_en=1
			reg_file[wr_addr] <=  wr_data;
		end
	end
endmodule
```

8.18.Κώδικας ALU_MUL

```
`define NA 16			// Length of first input
`define NB 16			// Length of second input
`define M  32			// Length of output

module ALU_MUL(
	a,			// First input of module
	b,			// Second input of module
	const,			// Constant direct from context word / potential input for MUL
	control,		// Control signal for the ALU
	sel_out,		// Select signal between ALU and MUL for the output
	sel_MUXin,		// Select signal between constant and normal input for the MUX
	result,			// Result output of the module
	zero,			// Zero output of the ALU
	overflow		// Overflow output of the ALU
);


//----------------------------------------Input ports ----------------------------------------//
	input wire [(`NA)-1:0] a;
	input wire [(`NB)-1:0] b;
	input wire [(`NA)-1:0] const;
	input wire [2:0] control;
```

```
        input wire sel_out;
        input wire sel_MUXin;

// ------------------------------------- Output ports ----------------------------------------//
        output wire [(`M)-1:0] result;
        output wire zero;
        output wire overflow;

// ------------------------------------- Internal variables ----------------------------------//
        wire [(`M)-1:0] alu_out;
        assign alu_out[(`M)-1:16] = 'b0;
        wire [(`M):0] mul_out;
        wire [(`NA)-1:0] a_sel;

// ------------------------------------- Code starts here... ---------------------------------//
        ALU16bit ALU(
                .result                 (alu_out[15:0]),
                .zero                   (zero),
                .overflow               (overflow),
                .a                      (a),
                .b                      (b),
                .control                (control)
        );

        InputMUX2to1 MUX1(
                .din0                   (a),
                .din1                   (const),
                .sel                    (sel_MUXin),
                .mux_out                (a_sel)
        );

        mult MUL(
                .a                      (a_sel),
                .b                      (b),
                .product                (mul_out)
        );

        MUX2to1 MUX2(
                .din0                   (alu_out),
                .din1                   (mul_out[31:0]),
                .sel                    (sel_out),
                .mux_out                (result)
        );
endmodule
```

8.19.Κώδικας ALU16bit

```
module ALU16bit(
        result,                 // Result output of ALU
        zero,                   // Set when the 2 inputs are equal
        overflow,               // Set when an overflow occurs
        a,                      // First input of ALU
        b,                      // Second input of ALU
        control                 // Controls the function of the ALU
);


//-----------------------------------------Input ports ----------------------------------------//
        input  wire [15:0] a;
        input wire [15:0] b;
        input wire [2:0] control;

// ------------------------------------- Output ports ----------------------------------------//
        output reg [15:0] result;
        output reg zero;
        output reg overflow;

// ------------------------------------- Internal variables ----------------------------------//
        reg [15:0] b_in;
        reg [15:0] result_in;
        reg c_in;
        reg c_out;
```

```
// ---------------------------------- Code starts here... ----------------------------------//
	always @(a or b or control)
	begin
		zero=1'b0;
		overflow=1'b0;

		case(control)
		3'b010,3'b110,3'b111:
		begin
		b_in = control[2] ? ~b : b;
		c_in = control[2] ? 1 : 0;
		{c_out,result_in} = a + b_in + c_in;
				if (control[0])
				result = (a[15] ^ b[15]) ? a[15] : result_in[15];
				else
					begin
					result = result_in;
					zero = control[2] ? (~|result) : 1'b0;
						if (a[15]==b_in[15])
						overflow = c_out ^ result_in[15];
		end
			end
		3'b000: result = a & b;
		3'b001: result = a | b;
			3'b011: result = a;
	default: result = 16'bx;
		endcase
	end

endmodule
```

8.20.Κώδικας mult

```
module mult(
	a,				// First operand of multiplier
	b,				// Second operand of multiplier
	product				// Product output
);

//---------------------------------- Parameters for generate ----------------------------------//
	parameter NA = 16;		// Length of first operand
	parameter NB = 16;		// Length of second operand
	parameter NP = 32;		// NA+NB
	parameter NL = 256;		// NA*NB

//-----------------------------------------Input ports ----------------------------------------//
	input wire [NA-1:0] a;
	input wire [NB-1:0] b;

// ---------------------------------- Output ports ----------------------------------------//
	output wire [NP:0] product;

// ---------------------------------- Internal variables ----------------------------------//

// Creates arrays, every cell of which corresponds with the
// port of the mult_cell respectively
	wire [NL-1:0] ain;
	wire [NL-1:0] bin;
	wire [NL-1:0] sin;
	wire [NL-1:0] cin;
	wire [NL-1:0] aout;
	wire [NL-1:0] bout;
	wire [NL-1:0] sout;
	wire [NL-1:0] cout;
	wire [NP-1:NB] cs_product;

	genvar i,j;

// ---------------------------------- Code starts here... ----------------------------------//
	//Generate all the cells
	generate
		for(i=0;i<NB;i=i+1) begin	: c_loop_ex
			for(j=0;j<NA;j=j+1) begin : c_loop_in
```

```
                            mul_cell
m_cell(ain[i*NA+j],bin[i*NA+j],sin[i*NA+j],cin[i*NA+j],aout[i*NA+j],bout[i*NA+j],sout[i*NA+j],cout[i
*NA+j]);
                        end //c_loop_in
                end //c_loop_ex
        endgenerate

        //Generate the intermediate wires
        generate
                for(i=1;i<NB;i=i+1) begin     : net_loop1
                        assign ain[((i*NA)+NA-1):(i*NA)] = aout[(((i-1)*NA)+NA-1):((i-1)*NA)];
                        assign     sin[((i*NA)+NA-1):(i*NA)]     =     {1'b0,sout[(((i-1)*NA)+NA-1):((i-
1)*NA+1)]};
                        assign cin[((i*NA)+NA-1):(i*NA)] = cout[(((i-1)*NA)+NA-1):((i-1)*NA)];
                end //net_loop1
        endgenerate

        generate
                for(i=0;i<NB;i=i+1) begin     : net_loop2
                        for(j=1;j<NA;j=j+1) begin : net_loop2_int
                                assign bin[i*NA+j] = bout[i*NA+j-1];
                        end  //net_loop2_int
                end //net_loop2
        endgenerate

        //Generate input connections
        assign ain[NA-1:0] = a;

        generate
                for(j=0;j<NA;j=j+1) begin : input_loop1
                        assign sin[j] = 1'b0;
                        assign cin[j] = 1'b0;
                end //input_loop1
        endgenerate

        generate
                for(i=0;i<NB;i=i+1) begin     : input_loop2
                        assign bin[i*NA] = b[i];
                end //input_loop2
        endgenerate

        //Generate output connections
        assign cs_product = {1'b0,sout[NA*NB-1:(NA*(NB-1))+1]};
        skip_adder SA(cs_product,cout[NA*NB-1:(NA*(NB-1))],1'b0,product[NP-1:NB],product[NP]);

        generate
                for(i=0;i<NB;i=i+1) begin : output_loop
                        assign product[i] = sout[i*NA];
                end //output_loop
        endgenerate
endmodule
```

8.21.Κώδικας skip_adder

```
module skip_adder(
        a,                      // First input of adder
        b,                      // Second input of adder
        cin,                    // Carry in input
        s,                      // Sum output
        cout                    // Carry out output
);

//------------------------------------ Parameters for generic ----------------------------------//
        parameter N = 16;      //total bit size
        parameter M = 4;       //group size

//-------------------------------------------Input ports ----------------------------------------//
        input wire [N-1:0] a;
        input wire [N-1:0] b;
        input wire cin;

// ---------------------------------------- Output ports ----------------------------------------//
        output wire [N-1:0] s;
```

```
        output wire cout;

// ------------------------------------ Internal variables ------------------------------------//
        wire [N-1:0] sum;
        wire [N/M:0] carry;

// ------------------------------------ Code starts here... ------------------------------------//
        assign carry[0] = cin;
        assign cout = carry[N/M];
        assign s = sum;

        genvar i,j;

        generate
                for(i=0;i<N/M;i=i+1)
                        begin : generate_out
                                //declare wires for inside the group
                                wire [M:0] int_carry; //internal carries of the full adders
                                wire [M:0] w; //help wires for creating a multi-input 'and' gate
                        wire [M-1:0] prop; //propagation signal for each full adder of the group
                                wire g_prop; //propagation signal  && carry in of the group

                                //assign wires
                                assign int_carry[0] = carry[i];
                                assign w[0] = int_carry[0];
                                assign g_prop = w[M];

                                //generate full adders of the group
                                for(j=0;j<M;j=j+1)
                                        begin : generate_in
                                                full_adder
FA(a[i*M+j],b[i*M+j],int_carry[j],sum[i*M+j],int_carry[j+1]);
                                                xor (prop[j],a[i*M+j],b[i*M+j]);
                                                and (w[j+1],prop[j],w[j]);
                                        end //for j

                                //carry out of the group
                                or      (carry[i+1],g_prop,int_carry[M]);
                        end //for i
        endgenerate
endmodule
```

8.22.Κώδικας mul_cell

```
module mul_cell(
        ain,                    // First input of cell
        bin,                    // Second input of cell
        sin,                    // Sum in input
        cin,                    // Carry in input
        aout,                   // First output of cell
        bout,                   // Second output of cell
        sout,                   // Sum out output
        cout                    // Carry out output
);

//-----------------------------------------Input ports ------------------------------------------//
        input wire ain;
        input wire bin;
        input wire sin;
        input wire cin;

// ------------------------------------ Output ports ------------------------------------------//
        output wire aout;
        output wire bout;
        output wire sout;
        output wire cout;

// ------------------------------------ Internal variables ------------------------------------//
        wire w1;                //hold the output of the and gate

// ------------------------------------ Code starts here... ------------------------------------//
        and (w1,ain,bin);                   //and gate of the cell
        full_adder FA(sin,w1,cin,sout,cout); //the FA of the cell
```

```
        assign aout = ain;
        assign bout = bin;
endmodule
```

ΚΕΦΑΛΑΙΟ 9
ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] : "Οδηγίες Χρήσης MicroSoc", Ι.Σιδέρης, Οκτώβριος 2006
[2] : "Προδιαγραφές του Συστήματος MicroSoc", Ν. Μοσχόπουλος, Νοέμβριος 2005
[3] : "ARM System-On-Chip Architecture", S.Furber, March 2000
[4] : "AMBA Specification (Rev 2.0)", ARM Limited, 1999
[5] : "Importance of SIMD Computation Reconsidered", W.C.Meilander, J.W.Baker and M.Jin, Department of Computer Science, Kent State University, Kent, Ohio
[6] : "Exploiting SIMD Computers for General Purpose Computation, P.A.Wilsey and D.A.Hensgen, IEEE, 6th International Parallel Processing Symp. IPPS, 1992
[7] : "MorphoSys: An Integrated Reconfigurable System", H.Singh, Guangming Lu, N.Bagherzadeh, IEEE, IEEE Transactions on Computers, 49, May 2000
[8] : "Reconfigurable Computing : A Survey of Systems and Software", Katherine Compton, Scott Hauck, ACM, ACM Computing Surveys, Vol.34, pp.171-210, 2002
[9] : "Reconf. Multi-Proc. IC for Rapid Prot. of Algorithmic-Specific Datapaths", D.Chen and J.Rabaey, IEEE, IEEE J.Solid-State Circuits, vol.27, Dec. 1992
[10] : "A Quantitative Analysis of Reconf. Coproc. for Multimedia Applications", T.Miyamori and K.Olukotun, Proc. IEEE Symp., FPGAs for Custom Computing Machines, Apr. 1998
[11] : "The RAW Benchmark Suite:Comput Structures for General-Purpose Computing", J.Babb,M.Frank,V.Lee et al., Proc.IEEE Symp., Field-Programmable Custom Computing Machines, pp. 134-143, Apr. 1997
[12] : "A 2.4 GOPS Data-Driven Reconfigurable Multiprocessor IC for DSP", A.K.Yeung and J.M.Rabaey, Proc.IEEE Solid-State Circuits Conf., pp. 108-109, Feb. 1995
[13] : "Configurable Computing: The Catalyst for High-Performance Architectures", C.Ebeling, D.Cronquist and P.Franklin, IEEE, Proc.IEEE Int`l Conf. Application, Specific Systems,Architectures and Processors, pp. 364-372, July 1997
[14] : "Building and Using a Highly Parallel Programmable Logic Array", M.Gokhale,W.Holmes,A.Kopser et al, Computer, pp.81-89, Jan. 1991
[15] : "Introduction to Programmable Active Memories", P.Bertin, D.Roncin and J.Vuillemin, Prentice Hall, Systolic Array Processors, pp. 300-309, 1989
[16] : "The Chimaera Reconfigurable Functional Unit", S.Hauck,T.W.Fry,M.M.Hosler and J.P.Kao, IEEE, Proc.IEEE Symp., Field-Programmable Custom Computing Machines, pp. 87-96, Apr. 1997
[17] : "A Dynamic Instruction Set Computer", M.J.Wirthlin and B.L.Hutchings, IEEE, Proc. IEEE Symp., FPGAs for Custom Computing Machines, pp. 99-107, Apr. 1995
[18] : "MATRIX: A Reconf. Computing Archit. with Configurable Instr.Distribution", E.Mirsky and A.DeHon, IEEE, Proc.IEEE Symp., FPGAs for Custom-Computing Machines, pp. 157-166, Apr. 1996
[19] : "A First Generation DPGA Implementation", E.Tau,D.Chen,I.Esxlick,J.Brown and A.DeHon, Proc. Canadian Workshop Field-Programmable Devices, May 1995
[20] : "Garp: A MIPS Processor with a Reconfigurable Co-Processor", J.R.Hauser and J.Wawrzynek, IEEE, Proc. IEEE Symp., Field-Programmable Custom Computing Machines, , April 1997
[21] : "OneChip: An FPGA Processor with Reconfigurable Logic", R.D.Wittig and P.Chow, IEEE, Proc. IEEE Symp., FPGAs for Custom Computing Machines, pp. 126-135, Apr. 1996
[22] : "PipeRench:A Coprocessor for Streaming Multimedia Acceleration", S.C.Goldstein,H.Schmit et al., Proc.Int`l Symp. Computer Architecture, pp. 28-39, May 1999
[23] : "Reconf. Architectures for Multimedia and DataParallel Application Domains", H.Singh, PhD thesis, Univ. of California,Irvine, 2000
[24] : "Some computer organizations and their effectiveness", M.Flynn, IEEE Transactions on Computers, pp. 948-960, Sep. 1972
[25] : "MorphoSys:An Integrated Re-configurable Architecture", H. Singh, Ming-Hau Lee,

Guangming Lu et al
[26] : "Digital Signal Processing Iplementation Using the TMS320C6000 DSP Platform", Naim Dahnoun
[27] : "FIR Filter Mapping and Performance Analysis on MorphoSys", H.Diab, E.Abdennour and F.Kurdahi, IEEE, pp.99-102, 2000
[28] : "Ιστοσελίδα www.model.com"
[29] : "Compilation Approcah for Coarse-Grained Reconfigurable Architectures", Jong-eun Lee, Kiyoung Choi, Nikil D. Dutt, IEEE, IEEE Design & Test of Computers, pp.26-33, 2003