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

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Επιτάχυνση αλγορίθμων** FIR **φίλτρων με χρήση υλικού σε ενσωματωμένο σύστημα σε προγραμματιζόμενη συσκευή**
Ονοματεπώνυμο Φοιτητή	**ΙΩΑΝΝΗΣ ΜΑΛΤΕΖΟΣ**
Πατρώνυμο	**ΚΡΙΤΩΝ**
Αριθμός Μητρώου	**ΜΠΣΠ**/ 08052
Επιβλέπων	**ΜΙΧΑΛΗΣ ΨΑΡΑΚΗΣ**, **ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ**

Ημερομηνία Παράδοσης **Ιούλιος** 2011

Τριμελής Εξεταστική Επιτροπή

Μιχάλης Ψαράκης Επίκουρος Καθηγητής	Δημήτρης Γκιζόπουλος Αναπληρωτής Καθηγητής	Άγγελος Πικράκης Λέκτορας

Πρόλογος

Αυτή η εργασία εκπονήθηκε στα πλαίσια της μεταπτυχιακής διατριβής μου στο μεταπτυχιακό πρόγραμμα σπουδών «Προηγμένα Συστήματα Πληροφορικής» της κατεύθυνσης «Τεχνολογία Ενσωματωμένων Υπολογιστικών Συστημάτων» του Τμήματος Πληροφορικής του Πανεπιστημίου Πειραιά. Ο τομέας που πραγματεύεται βρίσκεται ανάμεσα στην συ-σχεδίαση Υλικού Λογισμικού, και στους Επεξεργαστές Ειδικού Συνόλου Εντολών, αντικείμενα που εμπεριέχουν θέματα από όλα τα μαθήματα της κατεύθυνσης των μεταπτυχιακών μου σπουδών και ειδικότερα «Σύγχρονοι Επεξεργαστές», «Λειτουργικά Συστήματα Πραγματικού Χρόνου», «Προηγμένη Ψηφιακή Σχεδίαση», «Τεχνικές Ανάπτυξης Ενσωματωμένου Λογισμικού», «Μοντελοποίηση Ενσωματωμένων Υπολογιστικών Συστημάτων», έτσι θα ήθελα να ευχαριστήσω τους διδάσκοντες Δημήτρη Γκιζόπουλο – Αναπληρωτή Καθηγητή και Μιχάλη Ψαράκη – Επίκουρο Καθηγητή, για την ποιότητα της διδασκαλίας τους και το υψηλό επίπεδο των εργασιών που εκπονήθηκαν στα πλαίσια των μαθημάτων, στοιχεία που μου επέτρεψαν να προσεγγίσω ένα θέμα αιχμής, για την έρευνα, στο αντικείμενο των ενσωματωμένων υπολογιστικών συστημάτων.

Μαλτέζος Ιωάννης

Περίληψη

Η παρούσα μεταπτυχιακή διατριβή, έχει ως αντικείμενο την ανάπτυξη ενός συστήματος SoPC, με σκοπό την βελτίωση της απόδοσης, από πλευράς χρόνου, ενός ψηφιακού φίλτρου πεπερασμένης κρουστικής απόκρισης. Αυτό επιτυγχάνεται με δύο τρόπους, με την προσθήκη ενός επιταχυντή υλικού, που συνεργάζεται στην υλοποίηση του αλγορίθμου του φίλτρου με έναν επεξεργαστή, τον NIOS II της ALTERA, και με την προσθήκη ειδικών εντολών στον ίδιο τον επεξεργαστή.

Γίνεται βιβλιογραφική ανασκόπηση στο θέμα των επεξεργαστών ειδικού σκοπού με στόχο ο αναγνώστης να γνωρίζει τις τεχνικές ανάπτυξης του συστήματος που πρόκειται να υλοποιηθεί. Ένας επεξεργαστής ειδικού σκοπού σχεδιάζεται είτε για μία εφαρμογή είτε για σύνολο εφαρμογών με κοινά χαρακτηριστικά. Η επιτυχία στο σχεδιασμό του έγκειται στη βέλτιστη εκμετάλλευση των ιδιαίτερων χαρακτηριστικών των εφαρμογών ώστε να ικανοποιηθούν οι απαιτήσεις σε ταχύτητα, επιφάνεια, και κατανάλωση ισχύος. Για το σκοπό αυτό, η προσπάθεια εστιάζεται στη βελτιστοποίηση των απαιτητικών, επεξεργαστικά, υπολογιστικών διεργασιών, που είναι κρίσιμες για τις συνολικές επιδόσεις του επεξεργαστή.

Το όλο σύστημα στηρίζεται στο επεξεργαστή NIOS II της ALTERA και στους πυρήνες υλικού που μας παρέχονται από αυτήν, έτσι αναλύονται πλήρως τα τεχνικά χαρακτηριστικά και η μέθοδος χρησιμοποίησης των παραπάνω πυρήνων.

Πριν από το στάδιο υλοποποίησης του συστήματος, κρίθηκε χρήσιμο να αναφερθούν οι θεμελιώδεις μαθηματικές αρχές της ψηφιακής επεξεργασίας σήματος, πάνω στις οποίες θα στηριγχούμε για να σχεδιάσουμε το σύστημα του επιταχυντή υλικού και το υλικό πάνω στο οποίο εκτελείτε η ειδική εντολή του ψηφιακού φίλτρου πεπερασμένης κρουστικής απόκρισης. Για λόγους ρεαλιστικής προσέγγισης στην μεθοδολογία ανάπτυξης του λογισμικού του συστήματος, χρησιμοποιήθηκε το λειτουργικό σύστημα πραγματικού χρόνου FreeRTOS, πάνω στο οποίο τρέχουν όλες οι διεργασίες υλοποίησης των αλγορίθμων ψηφιακής επεξεργασίας σήματος, όσο και οι διεργασίες που χρησιμοποιούνται για την απεικόνιση των μετρήσεων σε οθόνη και τον έλεγχο ορθότητας λειτουργίας του συστήματος. Στο στάδιο υλοποίησης του υλικού του συστήματος παρουσιάζεται αναλυτικά η διαδικασία σχεδίασης του επιταχυντή και ο τρόπος εισαγωγής των ειδικών εντολών, καθώς και τα αποτελέσματα βελτίωσης που επιτυγχάνουμε από τις παραπάνω προσεγγίσεις.

Οι στόχοι της μεταπτυχιακής διατριβής οι οποίοι επιτεύχθηκαν είναι: η σημαντική επιτάχυνση εκτέλεσης αλγορίθμου ψηφιακού φίλτρου πεπερασμένης κρουστικής απόκρισης, 40 περίπου φορές με χρήση επιταχυντή υλικού, και 100 φορές με χρήση ειδικής εντολής που ενσωματώνεται στο ρεπερτόριο εντολών του επεξεργαστή NIOS II, η διερεύνηση του τρόπου ανάπτυξης λογισμικού με χρήση λειτουργικού συστήματος πραγματικού χρόνου για τον επεξεργαστή NIOS II και η παρουσίαση μεθοδολογίας για την ανάπτυξη άλλων αντίστοιχων συστημάτων που απαιτούν επιτάχυνση του χρόνου εκτέλεσης. Προκειμένου την πρακτική εφαρμοστικότητα της μεθοδολογίας εισήχθησαν τόσο ηχητικά όσο και αριθμητικά δεδομένα, για να αξιολογηθεί ως προς την απόδοση και την ορθότητα λειτουργίας το σχεδιασθέν σύστημα.

Ιωάννης Μαλτέζος
Πειραιάς - Ιούνιος 2011

Abstract

This postgraduate thesis, has as target the design of a SoPC system, aiming at the optimized output, from side of time, of a digital finite impulse response filter. This is achieved by two ways, with the addition of hardware accelerator that collaborates in the realization of the filter algorithm with a processor, the NIOS II of ALTERA, and with the addition of special instructions in the processor instruction set.

It is done bibliographic examination in the subject of special purpose processors in order the reader to know the design techniques of the system. A special purpose processor is designed for an application or for group of applications with common characteristics. The success of the design lies to the most optimal exploitation of particular characteristics of applications, so the requirements in speed, surface, and consumption of power are satisfied. For this purpose, the effort is focused in the optimization of intensive, computational, calculating tasks that are critical for the total performance of the processor.

The system is based on the processor NIOS II of ALTERA and on hardware cores that are provided, thus we are analyze completely the technical characteristics and the utilization method for the above cores.

Before the stage of system realization, it was useful to report the fundamental mathematics of digital signal processing, on which we based in order to design the hardware accelerator system and the hardware on what we execute the special instruction of digital finite impulse response filter. For reasons of realistic approach in the design methodology for the software system, was used the Real Time Operating System FreeRTOS, on what runs all realization processes for digital signal processing algorithms, as much as the processes that are used to display the measurements on screen and the operational validation of the system. In the stage of hardware realization, are presented analytically the design process of the accelerator and the way to import the special instruction, as well as the results of optimization that we achieve from the above approaches.

The postgraduate thesis objectives which were achieved are: the important acceleration for implementing a digital finite impulse response filter algorithm, about 40 times with use of hardware accelerator, and 100 times with use of special instruction that is incorporated in the instruction set of processor NIOS II, the investigation of a way to design software with use of Real Time Operational System for processor NIOS II and the presentation of a methodology for design other corresponding systems that requires acceleration of execution time. In order of the methodology practical appliance, were imported, sound, and numerical data, in order to be evaluated for the output and the validation of the designed system.

Ioannis Maltezos
Piraeus – June 2011

Πίνακας Περιεχομένων

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

1.1. Βασικές Έννοιες - Εισαγωγή

Στα ενσωματωμένα συστήματα, όπου απαιτείται υψηλή υπολογιστική ικανότητα παράλληλα με ευελιξία στην διαμόρφωση του υλικού και του λογισμικού, με σκοπό την βελτιωμένη απόδοσή τους σε εμπορικά, ιατρικά, στρατιωτικά συστήματα επεξεργασίας ροής ηχητικών και οπτικών σημάτων, εκτός από την χρησιμοποίηση εξειδικευμένων πολύ-επεξεργαστών ψηφιακής επεξεργασίας σήματος μία άλλη λύση είναι η ανάπτυξη ενός ειδικά σχεδιασμένου για κάθε εφαρμογή ενσωματωμένου συστήματος υλικού-λογισμικού. Αυτό επιτυγχάνετε ακολουθώντας μία θεώρηση συ-σχεδίασης του υλικού και του λογισμικού με στόχο την βέλτιστη λύση λαμβάνοντας υπόψη παράγοντες όπως χρόνος ανάπτυξης, κατανάλωση ισχύος, επιφάνεια υλικού και κυρίως ταχύτητα επεξεργασίας απαραίτητη για ένα τόσο απαιτητικό πεδίο εφαρμογών.

Η προσπάθεια αυτή για την βέλτιστη λύση, έχει ως στόχο την καλύτερη διαμοίραση των τμημάτων των αλγορίθμων σε επεξεργαστικά στοιχεία που υλοποιούνται σε υλικό και σε συναρτήσεις κώδικα που υλοποιούνται σε λογισμικό. Αυτή η διαδικασία αρχικά γινόταν εμπειρικά από τον σχεδιαστή του συστήματος, σήμερα όμως έχοντας αναπτύξει την σχεδίαση στο επίπεδο συστήματος (System Level Design) υπάρχουν εμπορικά και ακαδημαϊκά εργαλεία που μας επιτρέπουν την εισαγωγή της σχεδίασης σε γλώσσα υψηλού επιπέδου όπως C, την αυτοματοποιημένη διαμοίραση της εφαρμογής σε υλικό και λογισμικό και την εξαγωγή RTL κώδικα για το υλικό και C κώδικα για τον επεξεργαστή ή τους επεξεργαστές που έχουμε επιλέξει να έχει το σύστημα μας.

Τα επεξεργαστικά στοιχεία του υλικού μπορούν να είναι εκτός από κοινούς επεξεργαστές, επιταχυντές υλικού όπου σε συνεργασία με τον επεξεργαστή αναλαμβάνουν την υλοποίηση των απαιτητικών υπολογιστικά αλγορίθμων. Μία άλλη τεχνική, όπου αυτό επιτρέπεται, είναι η εισαγωγή ειδικών εντολών στον επεξεργαστή όπου ακόμα και με μία εντολή μπορούμε να υλοποιήσουμε ολόκληρη συνάρτηση του λογισμικού. Η τεχνολογία των συστημάτων σε προγραμματιζόμενες ψηφίδες (System on Programmable Chip) μας παρέχει όλες τις παραπάνω δυνατότητες, έτσι επιλέγεται για την ανάπτυξη ενσωματωμένων συστημάτων σε FPGAs ακολουθώντας την προαναφερθείσα μεθοδολογία.

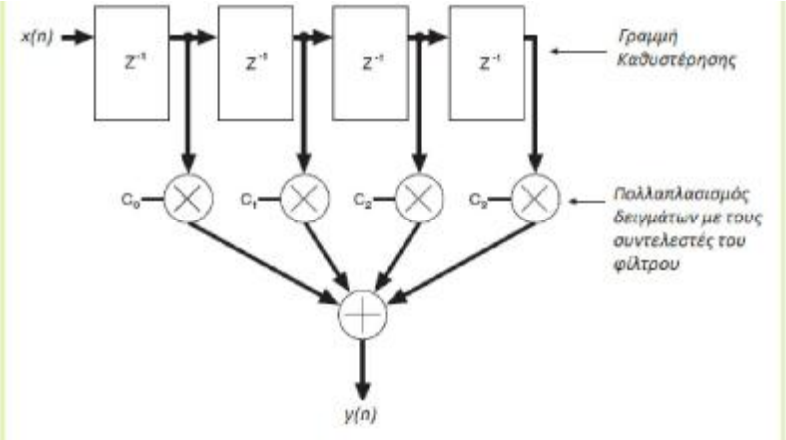
Στην διαδικασία ανάπτυξης του λογισμικού, λόγο της φύσης των εφαρμογών (ροές δεδομένων), η χρήση ενός λειτουργικού συστήματος πραγματικού χρόνου μας επιτρέπει την ακριβή πρόβλεψη της λειτουργίας του συστήματος από την φάση της σχεδίασης του. Αυτό γίνεται στο τελικό στάδιο ανάπτυξης αφού έχει γίνει η επιλογή των τμημάτων του κώδικα που θα υλοποιηθεί σε υλικό. Παράλληλα ένα λειτουργικό σύστημα πολλές φορές περιέχει έτοιμες συναρτήσεις γραφικών και γενικά επιτρέπει στον προγραμματιστή μια ταχύτερη, μεθοδικότερη και ακριβής διαδικασία ανάπτυξης του λογισμικού της εφαρμογής.

1.2 Πρόβλημα

Στην παρούσα μεταπτυχιακή διατριβή επιλέγηκε η υλοποίηση ενός ψηφιακού φίλτρου πεπερασμένης κρουστικής απόκρισης FIR ακολουθώντας την παραπάνω μεθοδολογία. Σε ένα FIR φίλτρο η σχέση μεταξύ ενός ψηφιακού σήματος εισόδου x(n) και του ψηφιακού σήματος εξόδου y(n) έχει την μορφή:

$$(y\,n\,) = \sum_{i=0}^{N-1} b_i\, x\,(n- i)$$

όπου b_i είναι οι συντελεστές του ψηφιακού φίλτρου και N ο αριθμός τους. Οι συντελεστές αυτοί παρέχουν τα χαρακτηριστικά του φίλτρου μέσα από το οποίο περνάει η ακολουθία x(n) έτσι ώστε να προκύψει η επιθυμητή έξοδος y(n). Σχηματικά ένα FIR φίλτρο παρουσιάζεται στην εικόνα 1.1



Εικόνα 1.1 – Σχηματικό διάγραμμα FIR φίλτρου

Η υλοποίηση ενός FIR φίλτρου μπορεί να επιτευχθεί με διάφορες δομές – μεθόδους. Η πιο διαδεδομένη δομή είναι η ευθεία υλοποίηση (tapped delayline ή transversal).

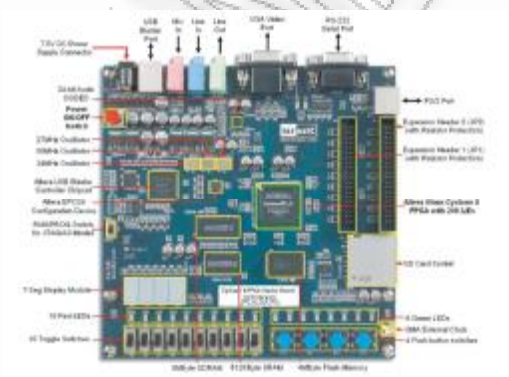
Για την υλοποίηση ενός FIR φίλτρου σε μόνο-επεξεργαστή λόγο του περιορισμού της ακολουθιακής μορφής που θα έχει ο κώδικας καταλήγουμε σε ένα ιδιαίτερα υπολογιστικά απαιτητικό λογισμικό, που μόνο σε ειδικούς επεξεργαστές ψηφιακής επεξεργασίας σήματος έχει πρακτική εφαρμογή.

1.3 **Κίνητρο**

Στην περίπτωση που θέλουμε να αναπτύξουμε ένα σύστημα σε προγραμματιζόμενη ψηφίδα χρησιμοποιώντας κοινούς soft-core επεξεργαστές που μας παρέχουν οι εταιρείες όπως ο NIOS II της ALTERA, έχουμε την δυνατότητα να βελτιώσουμε το σύστημά μας από πλευράς επιτάχυνσης της εκτέλεσης του FIR αλγόριθμου, επιλέγοντας την υλοποίησή του με επιταχυντή υλικού ή ακόμα προσθέτοντας ειδικές εντολές στον ίδιο τον επεξεργαστή.
Έτσι αναπτύχθηκε ένα σύστημα με σκοπό την ψηφιακή επεξεργασία ηχητικών σημάτων και την αξιολόγηση ως προς την ταχύτητα επεξεργασίας του αλγορίθμου FIR μεταξύ διαφορετικών υλοποιήσεων (Software NIOS, Software NIOS + Custom Instructions NIOS, FIR Hardware Accelerator).

1.4 **Περιγραφή του συστήματος**

Το σύστημα στηρίζεται στον NIOS II επεξεργαστή της ALTERA, έχει φορτωθεί το λειτουργικό σύστημα **FreeRtos** και υλοποιεί αλγόριθμους ψηφιακής επεξεργασίας ηχητικού σήματος. Οι επιλογές (mode) λειτουργίας επιλέγονται από ένα πληκτρολόγιο μέσω PS/2 και απεικονίζετε ανάλογο GUI σε VGA οθόνη. Το ηχητικό σήμα εισέρχεται και εξέρχεται ανάλογα επεξεργασμένο από τις σχετικές θύρες του αναπτυξιακού DE1 της ALTERA [14], [15] .
Στην οθόνη απεικονίζονται και στοιχεία για τον χρόνο (ταχύτητα) επεξεργασίας σχετικών αλγορίθμων επεξεργασίας ηχητικού σήματος.



Εικόνα 1.2 – Η πλακέτα ανάπτυξης DE1

Οι επιλογές λειτουργίας είναι οι ακόλουθες:

PLAY Mode: Το σύστημα εισέρχεται σε κατάσταση αναπαραγωγής ήχου. Επομένως, τα ηχητικά δεδομένα που εισέρχονται στο DE1 εξέρχονται από αυτό χωρίς καμία επεξεργασία.

NIOS_FIR Mode: Το σύστημα υλοποιεί την όλη διαδικασία του FIR φιλτραρίσματος μέσα από C πρόγραμμα – εφαρμογή που εκτελείται. Το φίλτρο που χρησιμοποιείται στο σύστημα είναι ένα βαθυπερατό φίλτρο, ενώ ο τύπος, οι ρυθμίσεις και οι συντελεστές του μπορούν να μεταβληθούν για τις εκάστοτε ανάγκες του συστήματος. Ο NIOS II επιβαρύνεται με ένα μεγάλο αριθμό πολλαπλασιασμών και προσθέσεων κάτι που καθιστά την εκτέλεση του προγράμματος ιδιαίτερα υπολογιστικά απαιτητική.

NIOS_FIR_Custom_Instruction Mode: Στην κατάσταση αυτή το σύστημα υλοποιεί την όλη διαδικασία του FIR φιλτραρίσματος μέσα από C πρόγραμμα που εκτελείται περιλαμβάνοντας σε αυτό custom instructions που αναπτύχθηκαν ειδικά για την εφαρμογή. Έτσι ο NIOS II δεν επιβαρύνεται με ένα μεγάλο αριθμό πολλαπλασιασμών και προσθέσεων κάτι που καθιστά την έννοια εκτέλεσης του προγράμματος σε πραγματικό χρόνο πιο προσιτή.

NIOS_FIR_Hardware_Accelerator Mode: Στην κατάσταση αυτή τα ηχητικά δεδομένα φιλτράρονται από ένα εξωτερικό Hardware Accelerator FIR φίλτρο. Ο NIOS II στέλνει τα ηχητικά δεδομένα στο FIR φίλτρο που υλοποιείται εξωτερικά του συστήματος, το φίλτρο στέλνει τα δεδομένα πίσω στο σύστημα από όπου ο NIOS II τα διαβάζει και τα αναπαράγει.

StandBy Mode: Η κύρια λειτουργία του συστήματος, όπως είδη αναφέρθηκε είναι η αναπαραγωγή ηχητικών επεξεργασμένων ή μη δεδομένων. Άρα είναι απαραίτητη και η ύπαρξη μίας κατάστασης στην οποία το σύστημα δεν παίζει καθόλου ήχο, δηλαδή τα ηχητικά δεδομένα μηδενίζονται.

1.5 **Αποτελέσματα**

Οι στόχοι της μεταπτυχιακής διατριβής οι οποίοι επιτεύχθηκαν είναι: η σημαντική επιτάχυνση εκτέλεσης αλγορίθμου ψηφιακού φίλτρου πεπερασμένης κρουστικής απόκρισης, 40 περίπου φορές με χρήση επιταχυντή υλικού, και 100 φορές με χρήση ειδικής εντολής που ενσωματώνεται στο ρεπερτόριο εντολών του επεξεργαστή NIOS II, η διερεύνηση του τρόπου ανάπτυξης λογισμικού με χρήση λειτουργικού συστήματος πραγματικού χρόνου για τον επεξεργαστή NIOS II και η παρουσίαση μεθοδολογίας για την ανάπτυξη άλλων αντίστοιχων συστημάτων που απαιτούν επιτάχυνση του χρόνου εκτέλεσης.

1.6 **Οργάνωση της μεταπτυχιακής διατριβής**

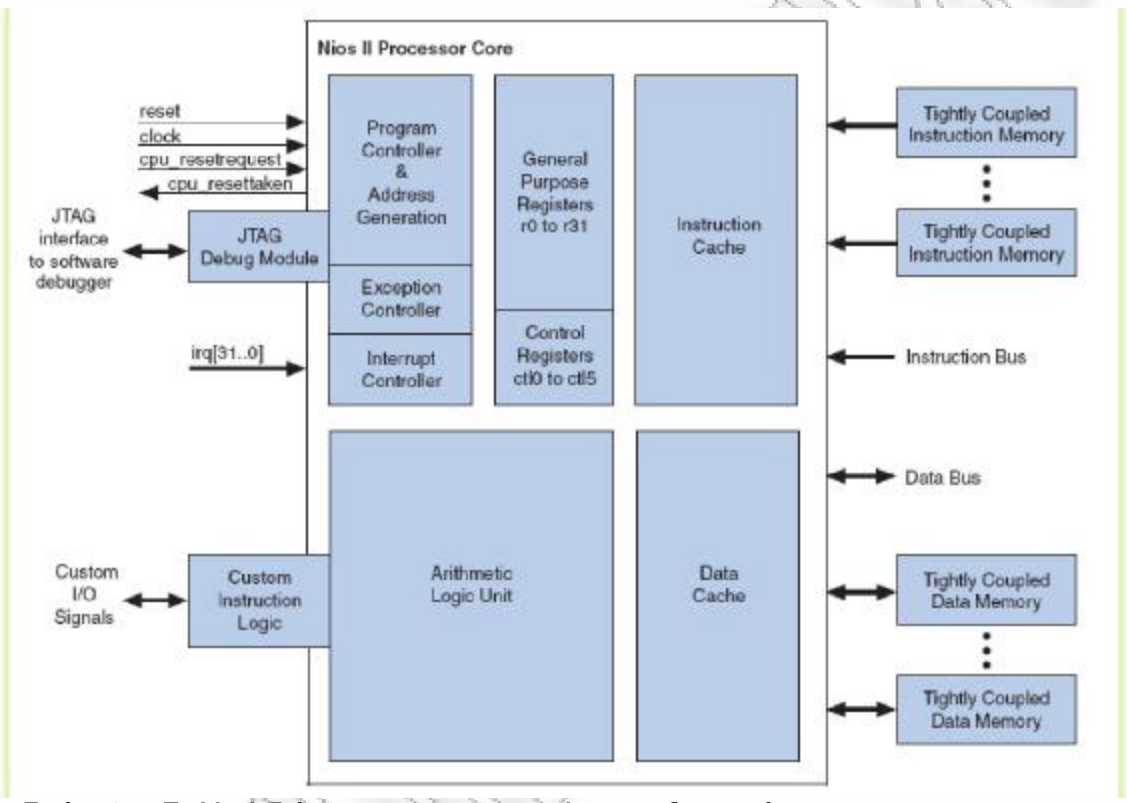
Στο δεύτερο κεφάλαιο αναπτύσσεται λεπτομερώς η αρχιτεκτονική του επεξεργαστή NIOS II, που θα χρησιμοποιηθεί για την υλοποίηση του συστήματος. Στο τρίτο κεφάλαιο γίνεται μία εισαγωγή στους επεξεργαστές ειδικού σκοπού. Στο τέταρτο κεφάλαιο περιγράφονται όλες οι μονάδες του υλικού και οι ρυθμίσεις που χρησιμοποιούνται για την υλοποίηση του SoPC της εργασίας. Στο πέμπτο κεφάλαιο γίνεται πρώτα μια αναφορά στην θεωρία των FIR φίλτρων και ακολουθούν οι τρεις διαφορετικές προσεγγίσεις (με λογισμικό, με επιταχυντή υλικού, με ειδικές εντολές) και παρατείθονται οι σχετικές μετρήσεις για τον χρόνο εκτέλεσης του αλγόριθμου FIR. Τέλος αναλύεται το λογισμικό του συστήματος και το λειτουργικό σύστημα πραγματικού χρόνου που έχει φορτωθεί στον NIOS II.

Κεφάλαιο 2

Ο Επεξεργαστής Nios II

Ο Επεξεργαστής Nios II

Η αρχιτεκτονική Nios II [1],[2] περιγράφει ένα σετ εντολών (Instruction Set Architecture - ISA). Η ISA υπονοεί την ύπαρξη λειτουργικών μονάδων στις οποίες υλοποιούνται οι εντολές. Στην εικόνα 2.1 παρουσιάζεται το block διάγραμμα του core του Nios II επεξεργαστή.



Εικόνα 2.1 - Το block διάγραμμα του core του Nios II επεξεργαστή

Οι λειτουργικές μονάδες του Nios II, αποτελούν τη βάση του σετ εντολών του Nios II, ωστόσο αυτό δεν υποδεικνύει ότι όλες οι μονάδες αυτές έχουν υλοποιηθεί σε hardware. Όπως αναφέρθηκε προηγουμένως, η αρχιτεκτονική του Nios II περιγράφει ένα σετ εντολών και όχι κάποια συγκεκριμένη υλοποίηση hardware. Μία λειτουργική μονάδα μπορεί είτε να υλοποιηθεί σε hardware, είτε να εξομοιωθεί σε software ή και να παραβλεφθεί εντελώς. Η υλοποίηση του Nios II πραγματοποιείται μέσω ενός συνόλου επιλογών αρχιτεκτονικής (διαφορετικά Nios II cores) που εξυπηρετούν συγκεκριμένες προδιαγραφές, όπως μικρό μέγεθος του core ή υψηλή απόδοση, κάτι το οποίο επιτρέπει στον Nios II να προσαρμόζεται στις διαφορετικές ανάγκες διαφορετικών εφαρμογών. Οι λειτουργικές μονάδες του Nios II είναι:

- Αρχείο καταχωρητών (register file)
- Αριθμητική και λογική μονάδα (arithmetic logic unit)
- Δίαυλος εντολών (instruction bus)
- Δίαυλος δεδομένων (data bus)
- Ελεγκτής εξαιρέσεων (exception controller)
- Ελεγκτής διακοπών (interrupt controller)
- Κρυφές μνήμες δεδομένων και εντολών (instruction-data cache memories)
- Διασύνδεση με λογική προσδιοριζόμενη από το χρήστη (custom instruction logic)
- Μονάδα αποσφαλμάτωσης JTAG (JTAG debug module)

- Επιπλέον στενά-συνδεδεμένη με το σύστημα μνήμη για εντολές και δεδομένα (tightly-coupled memory)

Οι μεταβλητές υλοποίησης, σε γενικές γραμμές, ανήκουν σε μία εκ των τριών κατηγοριών:

- *Αύξηση ή ελάττωση ενός χαρακτηριστικού (more or less of a feature)* – πχ, για βελτίωση της απόδοσης, το μέγεθος της κρυφής μνήμης εντολών μπορεί να αυξηθεί ή να μειωθεί. Μία κρυφή μνήμη μεγάλου μεγέθους, αυξάνει την ταχύτητα εκτέλεσης μεγάλων προγραμμάτων, ενώ μία αντίστοιχη μικρού μεγέθους διατηρεί τους πόρους της on-chip μνήμης.
- *Συμπερίληψη ή αποκλεισμός ενός χαρακτηριστικού (inclusion or exclusion of a feature)* – πχ, για τη μείωση του κόστους, η μονάδα αποσφαλμάτωσης JTAG μπορεί να παραλειφθεί. Η απόφαση αυτή δεν "σπαταλάει" on-chip λογική και μνήμη, εξαλείφει όμως τη δυνατότητα χρήσης ενός software debugger για αποσφαλμάτωση εφαρμογών.
- *Hardware υλοποίηση ή software* υλοποίηση – πχ, σε εφαρμογές ελέγχου όπου σπάνια το σύστημα εκτελεί πολύπλοκες αριθμητικές πράξεις, η εντολή της διαίρεσης μπορεί να επιλεγεί να εξομοιωθεί σε software. Η αφαίρεση του hardware διατηρεί μεν on-chip πόρους, αυξάνει όμως τον χρόνο εκτέλεσης διαιρέσεων.
- Nios II/f – Το Nios II/f "fast" core είναι σχεδιασμένο για μέγιστη απόδοση. Ως αποτέλεσμα, το core παρουσιάζει τις περισσότερες ρυθμίσεις παραμέτρων για το σκοπό αυτό.
- Nios II/s – Το Nios II/s "standard" core έχει σχεδιαστεί για μικρό μέγεθος, διατηρώντας παράλληλα την απόδοση σε ικανοποιητικά επίπεδα.
- Nios II/e – Το Nios II/e "economy" core έχει σχεδιαστεί ώστε να έχει το μικρότερο δυνατό μέγεθος, με αποτέλεσμα, το core αυτό να έχει περιορισμένο σετ χαρακτηριστικών, ενώ πολλές ρυθμίσεις δεν είναι διαθέσιμες.

Πρέπει να σημειωθεί ότι το core του Nios II δεν περιλαμβάνει τη λογική σύνδεσης του επεξεργαστή με τις όποιες περιφερειακές εξωτερικές συσκευές.

2.1 **Αρχείο καταχωρητών** (register file)

Η Nios II αρχιτεκτονική υποστηρίζει ένα αρχείο καταχωρητών που αποτελείται από 32 γενικού σκοπού καταχωρητές των 32-bit και μέχρι 32 καταχωρητές ελέγχου των 32-bit .

2.1.1 **Καταχωρητές γενικού σκοπού**

Οι καταχωρητές γενικού σκοπού είναι ακέραιοι καταχωρητές των 32-bit (r0 έως r31) όπως φαίνεται στον πίνακα 2.1.

Ο καταχωρητής *zero* (r0) πάντα επιστρέφει την τιμή μηδέν ενώ η εγγραφή ενός μηδενικού σε αυτόν τον καταχωρητή δεν έχει καμία επίδραση. Ο καταχωρητής *ra* (r31) διατηρεί τη διεύθυνση επιστροφής μετά από κλήση κάποιας υπό-ρουτίνας και είναι προσβάσιμος αποκλειστικά μέσα από *call* και *ret* εντολές.

Καταχωρητής	Όνομα	Λειτουργία	Καταχωρητής	Όνομα	Λειτουργία
r0	zero	0x00000000	r16		
r1	at	Assembler Temporary	r17		
r2		Return Value	r18		
r3		Return Value	r19		
r4		Register Arguments	r20		
r5		Register Arguments	r21		
r6		Register Arguments	r22		
r7		Register Arguments	r23		
r8		Caller-Saved Register	r24	et	Exception Temporary
r9		Caller-Saved Register	r25	bt	Breakpoint Temporary

r10		Caller-Saved Register	r26	gp	Global Pointer
r11		Caller-Saved Register	r27	sp	Stack Pointer
r12		Caller-Saved Register	r28	fp	Frame Pointer
r13		Caller-Saved Register	r29	ea	Exception Return Address
r14		Caller-Saved Register	r30	ba	Breakpoint Return Address *(1)*
r15		Caller-Saved Register	r31	ra	Return Address

Πίνακας 2.1 - Οι καταχωρητές γενικού σκοπού

2.1.2 **Καταχωρητές ελέγχου**

Οι καταχωρητές ελέγχου, που παρουσιάζονται στον πίνακα 2.2, αναφέρουν την κατάσταση του επεξεργαστή και μπορούν να αλλάξουν τη συμπεριφορά του.

Καταχωρητής	Όνομα	31...0	0
0	status	Reserved	PIE
1	estatus	Reserved	EPIE
2	bstatus	Reserved	EPIE
3	ipending	Interrupt-enable Bits	
4	ienable	Pending-interrupt bits	
5	cpuid	Unique processor identifier	
6-31	Reserved	Reserved	

Πίνακας 2.2 - Οι καταχωρητές ελέγχου

Οι καταχωρητές ελέγχου προσπελαύνονται με ειδικές εντολές ανάγνωσης και εγγραφής

status register

Η τιμή στον καταχωρητή *status* ελέγχει την κατάσταση του Nios II επεξεργαστή. Όλα τα bits του καταχωρητή μηδενίζονται κατά την επανεκκίνηση του επεξεργαστή. Μερικά bits χρησιμοποιούνται αποκλειστικά και είναι διαθέσιμα σε συγκεκριμένα χαρακτηριστικά του επεξεργαστή.

Bit	Περιγραφή	Πρόσβαση	Reset	Διαθεσιμότητα
PIE	Το *PIE* είναι το bit ενεργοποίησης διακοπών του επεξεργαστή. Όταν το *PIE* είναι στο λογικό "0", οι όποιες διακοπές αγνοούνται. Όταν το *PIE* είναι στο λογικό "1", οι διακοπές εξυπηρετούνται ανάλογα με την τιμή του καταχωρητή *ienable*	Ανάγνωση / Εγγραφή	0	Πάντα

estatus register

Ο καταχωρητής estatus διατηρεί ένα αντίγραφο του καταχωρητή status κατά τη διάρκεια επεξεργασίας μιας εξαίρεσης (όχι break εξαίρεσης). Ο χειριστής εξαιρέσεων (exception handler) εξετάζει τον καταχωρητή estatus για να προσδιορίσει την κατάσταση του επεξεργαστή πριν την εμφάνιση της εξαίρεσης. Όταν γίνεται η επιστροφή από μία εξαίρεση, η εντολή eret αντιγράφει το περιεχόμενο του estatus πίσω στον status καταχωρητή, επαναφέροντας έτσι την αρχική τιμή του.

bstatus register
Ο bstatus καταχωρητής διατηρεί ένα αντίγραφο του καταχωρητή status κατά τη διάρκεια επεξεργασίας μιας break εξαίρεσης. Όταν δημιουργείται ένα break, η τιμή του status καταχωρητή αντιγράφεται στον bstatus. Ο debugger, κάνοντας χρήση του bstatus μπορεί να επαναφέρει τον καταχωρητή status στην τιμή που είχε πριν την εμφάνιση του break. Η bret εντολή, αντιγράφει την τιμή του bstatus στον status καταχωρητή.

ipending register
Η τιμή στον ipending καταχωρητή υποδεικνύει τη διακοπή που εισάγεται στον επεξεργαστή. Η τιμή "1" στο bit n σημαίνει ότι η αντίστοιχη IRQn διακοπή εκκρεμεί, ενώ η εγγραφή κάποιας τιμής στον ipending δεν έχει καμιά επίδραση.

ienable register
Ο ienable καταχωρητής χειρίζεται τις διακοπές από εξωτερικές συσκευές. Κάθε bit του ienable καταχωρητή, αντιστοιχεί σε μία από τις εισόδους διακοπών, IRQ0 έως IRQ31. Η τιμή "1" στο bit n σημαίνει ότι η αντίστοιχη IRQn διακοπή έχει ενεργοποιηθεί, ενώ η τιμή "0" σημαίνει ότι η αντίστοιχη διακοπή έχει απενεργοποιηθεί.

cpuid register
Ο cupid καταχωρητής διατηρεί μία συνεχή τιμή η οποία υποδεικνύει με μοναδικό τρόπο κάθε επεξεργαστή σε ένα σύστημα πολλών επεξεργαστών. Η τιμή του καθορίζεται κατά την παραγωγή του συστήματος ενώ η εγγραφή κάποιας τιμής στον cupid δεν έχει καμιά επίδραση.

2.2 **Αριθμητική και λογική μονάδα** (arithmetic logic unit)
Η αριθμητική και λογική μονάδα (ALU) του Nios II λειτουργεί με δεδομένα αποθηκευμένα στους καταχωρητές γενικού σκοπού. Οι διεργασίες που εκτελούνται στην ALU λαμβάνουν ως ορίσματα τις τιμές ενός ή δύο καταχωρητών και αποθηκεύουν το αποτέλεσμα σε ένα τρίτο. Η ALU υποστηρίζει: i) αριθμητικές πράξεις όπως πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση προσημασμένων και μη προσημασμένων τελεστών, ii)σχεσιακές πράξεις όπως ίσο, άνισο, μεγαλύτερο ή ίσο από, μικρότερο από (==, !=, >=, <) μεταξύ προσημασμένων και μη προσημασμένων τελεστών, iii) λογικές πράξεις όπως AND, NOR, OR, XOR και iv) πράξεις ολίσθησης και περιστροφής δεδομένων από 0 έως 31 θέσεις bit ανά εντολή. Η ALU υποστηρίζει αριθμητικές ολισθήσεις προς τη δεξιά κατεύθυνση και λογικές ολισθήσεις προς τη δεξιά/αριστερή. Για την υλοποίηση οποιασδήποτε άλλης λειτουργίας εκτελείται ένας συνδυασμός των παραπάνω βασικών διεργασιών.

2.2.1 **Μη Υλοποιημένες Εντολές**
Μερικές υλοποιήσεις του Nios II επεξεργαστή δεν παρέχουν το απαραίτητο hardware για την εκτέλεση πράξεων πολλαπλασιασμού ή διαίρεσης. Οι εντολές mul, muli, mulxxs, mulxsu, mulxuu, div, divu δεν εμφανίζονται σε μερικές υλοποιήσεις του Nios II core και είναι γνωστές ως μη υλοποιημένες εντολές. Όλες οι υπόλοιπες εντολές υλοποιούνται σε hardware. Ο επεξεργαστής παράγει μία εξαίρεση όταν συναντά μία μη υλοποιημένη εντολή και ο χειριστής εξαιρέσεων καλεί την κατάλληλη υπό-ρουτίνα εξομοίωσης της εντολής. Συνεπώς, οι μη υλοποιημένες εντολές δεν επηρεάζουν την όποια προγραμματιστική τεχνική. Η αρχιτεκτονική του Nios II υποστηρίζει εντολές προσδιοριζόμενες από το χρήστη. Η ALU του Nios II συνδέεται απευθείας στην αντίστοιχη λογική υλοποιώντας σε hardware λειτουργίες που χρησιμοποιούνται ακριβώς όπως οι μητρικές εντολές.

2.2.2 **Εντολές Κινητής Υποδιαστολής**
Η Nios II αρχιτεκτονική υποστηρίζει επίσης εντολές κινητής υποδιαστολής (floating point) απλής ακρίβειας, [8], [62] όπως προσδιορίζονται στο IEEE Std 754-1985. Αυτές οι floating point εντολές υλοποιούνται με τη λογική των προσδιοριζόμενων από το χρήστη ειδικών εντολών και παρουσιάζονται στον πίνακα 2.3.

Χαρακτηριστικό		Υλοποίηση
Πράξεις[1]	Πρόσθεση	Υλοποιημένη
	Αφαίρεση	Υλοποιημένη
	Πολλαπλασιασμός	Υλοποιημένη
	Διαίρεση	Προαιρετική
Ακρίβεια	Απλή	Υλοποιημένη
	Διπλή	Μη-υλοποιημένη. Οι πράξεις διπλής ακριβείας υλοποιούνται σε software
Καταστάσεις Εξαίρεσης	Μη-έγκυρη πράξη	Το αποτέλεσμα δεν είναι αριθμός (NaN)
	Διαίρεση με το μηδέν	Το αποτέλεσμα είναι ± ∞
	Υπερχείληση	Το αποτέλεσμα είναι ± ∞
	Inexact	Το αποτέλεσμα είναι κανονικός αριθμός
	Υποχείληση	Το αποτέλεσμα είναι ± 0
Τύποι Στρογγυλοποίησης	Στρογγυλοποίηση στον κοντινότερο	Υλοποιημένη
	Στρογγυλοποίηση προς το μηδέν	Μη-υλοποιημένη
	Στρογγυλοποίηση προς το +∞	Μη-υλοποιημένη
	Στρογγυλοποίηση προς το -∞	Μη-υλοποιημένη
NaN	Quiet	Υλοποιημένη
	Signaling	Μη-υλοποιημένη
Αριθμοί εκτός ορίων		Οι αριθμοί κάτω του ορίου μεταχειρίζονται ως μηδέν
Software Εξαιρέσεις		Μη-υλοποιημένες
Σημαία Κατάστασης		Μη-υλοποιημένη

Πίνακας 2.3 - Εντολές Κινητής Υποδιαστολής

2.3 **Σήματα επαναφοράς** (reset signals)

Ο Nios II επεξεργαστής υποστηρίζει δύο σήματα επαναφοράς: i)*reset* – είναι το καθολικό σήμα hardware επαναφοράς που υποχρεώνει τον επεξεργαστή σε άμεση επανεκκίνηση και ii)*cpu_reset request* – αυτό είναι ένα τοπικό σήμα επαναφοράς που προκαλεί την επανεκκίνηση του επεξεργαστή χωρίς όμως να επηρεάζει τα άλλα components του Nios II συστήματος. Ο επεξεργαστής ολοκληρώνει την εκτέλεση οποιασδήποτε εντολής είναι μέσα στο pipeline και μετά εισέρχεται σε κατάσταση επαναφοράς. Η διαδικασία αυτή μπορεί να καταναλώσει αρκετούς κύκλους ρολογιού. Όταν ολοκληρωθεί η επαναφορά του επεξεργαστή τότε αυτός θέτει το σήμα *cpu_resettaken* για έναν κύκλο ρολογιού και έπειτα ανά περιοδικά χρονικά διατήματα, εάν το σήμα *cpu_resetrequest* παραμένει στο λογικό "1". Ενώ ο επεξεργαστής βρίσκεται σε κατάσταση επαναφοράς, διαβάζει περιοδικά τη διεύθυνση επαναφοράς, απορρίπτει το αποτέλεσμα της ανάγνωσης και παραμένει στην ίδια κατάσταση. Ο επεξεργαστής δεν ανταποκρίνεται στο *cpu_resetrequest* όταν είναι κάτω από τον έλεγχο της μονάδας αποσφαλμάτωσης JTAG, δηλαδή όταν είναι σε κατάσταση παύσης.

2.4 **Ελεγκτής εξαιρέσεων και διακοπών**

Μία εξαίρεση είναι οποιαδήποτε κατάσταση ή σήμα (hardware ή software) διακόπτει την κανονική ροή εκτέλεσης του κυρίως προγράμματος-εφαρμογής. Έτσι, η εξυπηρέτηση μια εξαίρεσης είναι ουσιαστικά η ανταπόκριση του επεξεργαστή στη συγκεκριμένη εξαίρεση και η επαναφορά έπειτα, στην αρχική κατάσταση εκτέλεσης προγράμματος. Ο ελεγκτής / διαχειριστής εξαιρέσεων είναι ένα ολοκληρωμένο σύστημα από software ρουτίνες οι οποίες εξυπηρετούν οποιαδήποτε εξαίρεση και

δίνουν τον έλεγχο σε συγκεκριμένες ρουτίνες εξυπηρέτησης διακοπών (Interrupt Service Routines - ISRs) όπου είναι αναγκαίο [3], [11].
Κάθε εξαίρεση σε ένα Nios II σύστημα ανήκει σε μία εκ των παρακάτω κατηγοριών:

- Reset Εξαιρέσεις
- Break Εξαιρέσεις
- Hardware Διακοπές (hardware interrupts)
- Εξαιρέσεις Σχετικές με Εντολές

Η εξυπηρέτηση εξαιρέσεων στον Nios II υλοποιείται με την κλασική RISC λογική, δηλαδή όλοι οι τύποι εξαιρέσεων εξυπηρετούνται από ένα γενικό διαχειριστή εξαιρέσεων. Έτσι οι εξαιρέσεις αυτές (hardware και software) διαχειρίζονται από software κώδικα που βρίσκεται σε μία συγκεκριμένη διεύθυνση μνήμης γνωστή ως "διεύθυνση εξαίρεσης" (exception address).

2.4.1 Reset **Εξαιρέσεις**

Όταν δημιουργείται ένα σήμα επαναφοράς (*reset*) ο Nios II εκτελεί συγκεκριμένες εργασίες όπως ο μηδενισμός του *status* καταχωρητή που σκοπό έχει την απενεργοποίηση των hardware διακοπών. Εκτός αυτού όμως, τα περιεχόμενα της cache μνήμης μετά το reset είναι σε μία απροσδιόριστη κατάσταση. Για την εξασφάλιση της συνοχής των δεδομένων στην cache μνήμη, ο διαχειριστής reset (reset handler) πρέπει να αρχικοποιήσει άμεσα την cache μνήμη μετά το reset. Η κατάσταση επαναφοράς δεν επηρεάζει τα υπόλοιπα components του συστήματος συμπεριλαμβανομένων των καταχωρητών γενικού σκοπού (εκτός των *zero* καταχωρητή που είναι μόνιμα μηδενισμένος), των καταχωρητών ελέγχου (εκτός του *status* καταχωρητή που αρχικοποιείται στο "0x00"), μνημών δεδομένων και εντολών, περιφεριακών συσκευών και άλλων.

2.4.2 Break **Εξαιρέσεις**

Ένα "*break*" είναι μια μεταφορά ελέγχου εκτός της κανονικής ροής εκτέλεσης του προγράμματος / εφαρμογής με σκοπό την αποσφαλμάτωση του. Το software αποσφαλμάτωσης (Nios II IDE) αποκτά τον έλεγχο του επεξεργαστή μέσω του JTAG και υλοποιεί τη διαδικασία αποσφαλμάτωσης (με *breakpoints* και *watchpoints*). Ο μηχανισμός των break εξαιρέσεων είναι ανεξάρτητος από το γενικό μηχανισμό εξυπηρέτησης εξαιρέσεων. Ο επεξεργαστής εισάγεται σε κατάσταση εξυπηρέτησης break εξαίρεσης είτε όταν εκτελεστεί η εντολή *break (software break)*, είτε όταν η JTAG μονάδα δημιουργήσει ένα *hardware break*. Και στις δύο αυτές περιπτώσεις ο επεξεργαστής:

i. αποθηκεύει τα περιεχόμενα του *status* καταχωρητή στον *bstatus*,
ii. μηδενίζει το *PIE* bit από τον *status* καταχωρητή απενεργοποιώντας έτσι τις hardware διακοπές,
iii. γράφει τη διεύθυνση της εντολής που υπάρχει μετά το break στον *ba (r30)* καταχωρητή
iv. μεταφέρει την εκτέλεση στο διαχειριστή break εξαιρέσεων, που είναι αποθηκευμένος στο *break vector(διεύθυνση ορισμένη κατά την παραγωγή τουσυστήματος)*.

Μετά την εξυπηρέτηση της break εξαίρεσης, ο διαχειριστής break εξαιρέσεων απελευθερώνει τον έλεγχο του επεξεργαστή εκτελώντας μία *bret* εντολή η οποία επαναφέρει το περιεχόμενο του *status* καταχωρητή από τον *bstatus* καταχωρητή και επιστέφει την εκτέλεση του προγράμματος στη θέση μνήμης που υπάρχει στον *ba (r30)*καταχωρητή.

2.4.3 **H**ardware **Διακοπές**

Μία περιφερειακή συσκευή μπορεί να προκαλέσει ένα αίτημα hardware διακοπής (Interrupt Requests - IRQs), θέτοντας μία από τις 32 εισόδους διακοπών (*IRQ0* έως *IRQ31*) του Nios II. Μία hardware διακοπή συμβαίνει μόνο όταν :

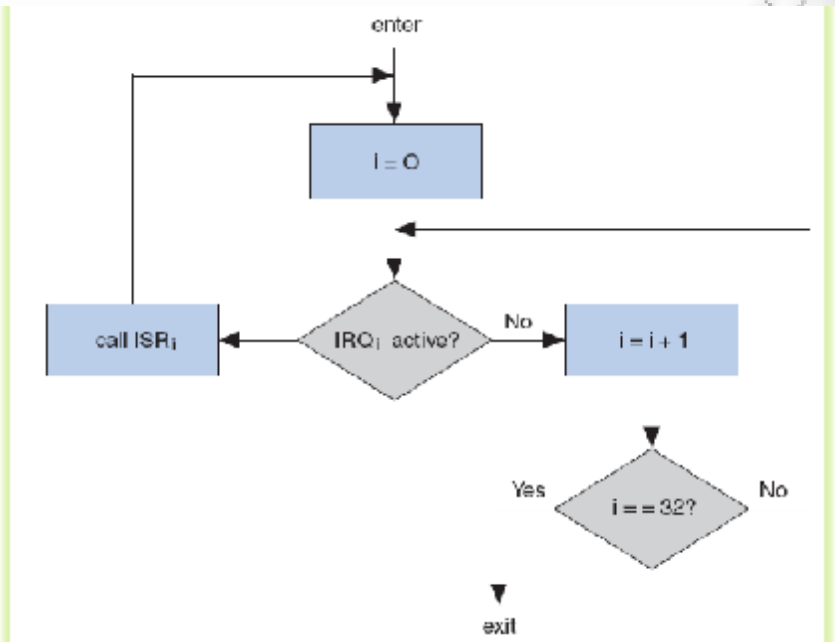
- Το *PIE* bit του *status* καταχωρητή είναι στο λογικό "1"
- Έχει δημιουργηθεί η κατάλληλη αίτηση διακοπής IRQ*n*
- Το αντίστοιχο *n* bit του *inable* καταχωρητή είναι και αυτό στο λογικό "1"

Όταν συμβεί μια διακοπή ο επεξεργαστής εκτελεί μια σειρά εργασιών παρόμοιες με αυτές της break εξαίρεσης :

i. αντιγράφει τα περιεχόμενα του *status* καταχωρητή στον *estatus*, αποθηκεύοντας την προ-διακοπής κατάσταση του.
ii. μηδενίζει το *PIE* bit από τον *status* καταχωρητή απενεργοποιώντας έτσι τις hardware διακοπές,

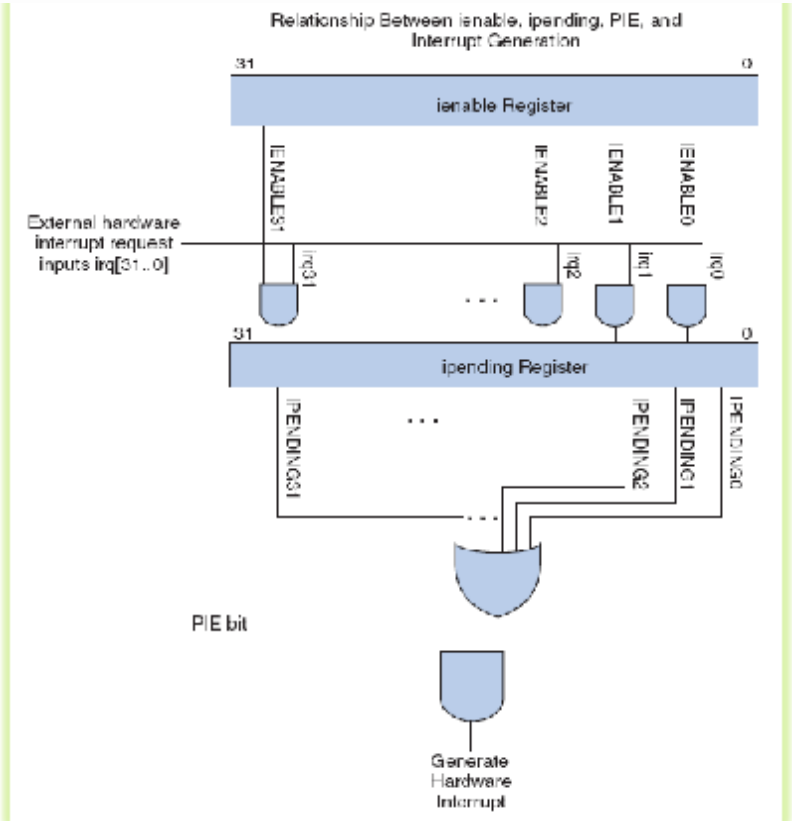
iii. γράφει τη διεύθυνση της εντολής που υπάρχει μετά την διακοπή στον *ea (r29)* καταχωρητή

iv. μεταφέρει την εκτέλεση στο διαχειριστή διακοπών, ο οποίος εξετάζει όλες τις καταχωρημένες (από τον χρήστη) IRQs σε εκκρεμότητα και βρίσκει την πρώτη ενεργοποιημένη IRQ με την υψηλότερη προτεραιότητα. Έπειτα καλεί την αντίστοιχη καταχωρημένη ρουτίνα εξυπηρέτησης διακοπής (Interrupt Service Routine–ISR). Μετά την εκτέλεση της συγκεκριμένης ISR ο χειριστής εξαιρέσεων σαρώνει ξανά τις IRQs αρχίζοντας από την IRQ0. Με τον τρόπο αυτό οι διακοπές υψηλής προτεραιότητας εξυπηρετούνται πάντα πριν από τις χαμηλής προτεραιότητας διακοπές. Η εικόνα 2.2 παρουσιάζει την παραπάνω διαδικασία σε μορφή διαγράμματος ροής.



Εικόνα 2.2 – Διαδικασία εξυπηρέτησης διακοπών

Η τιμή του *ipending* καταχωρητή υποδεικνύει ποιες αιτήσεις διακοπής εκκρεμούν. Μία αίτηση διακοπής εκκρεμεί έως ότου ο επεξεργαστής ανταποκριθεί στην αντίστοιχη περιφερειακή συσκευή. Η εικόνα 2.3 δείχνει τη σχέση μεταξύ των *ipending*, *inable* και *PIE* – bit με τη δημιουργία μιας διακοπής.



Εικόνα 2.3 – Οι καταχωρητές των διακοπών

Πρέπει να σημειωθεί ότι οι διακοπές μπορούν να επα-ενεργοποιηθούν εγγράφοντας το λογικό “1” στο *PIE* bit. Συνήθως, το *ienable* bit προσαρμόζεται (από τη ρουτίνα εξαίρεσης) ώστε οι IRQs ίδιας ή χαμηλότερης προτεραιότητας να απενεργοποιούνται πριν την επα-ενεργοποίηση των διακοπών.

Όπως έχει υπονοηθεί, το software αλληλεπιδρά/επικοινωνεί με τις εξωτερικές περιφερειακές συσκευές μέσω των διακοπών. Όταν μία περιφερειακή συσκευή προκαλεί μία IRQ, η αντίστοιχη ISR πρέπει να εξυπηρετήσει τη διακοπή αυτή και να επαναφέρει στο τέλος τον επεξεργαστή στην κατάσταση προ-διακοπής. Η HAL βιβλιοθήκη συστήματος παρέχει ένα σύνολο εντολών διασύνδεσης (Application Program Interface - API) μέσω του οποίου διευκολύνεται η δημιουργία και η “συντήρηση” ρουτινών εξυπηρέτησης διακοπών (ISRs). Το HAL API καθορίζει τις παρακάτω συναρτήσεις για τη διαχείριση hardware διακοπών:

- alt_irq_register()
- alt_irq_disable()
- alt_irq_enable()
- alt_irq_disable_all()
- alt_irq_enable_all()
- alt_irq_interruptible()
- alt_irq_non_interruptible()
- alt_irq_enabled()

Πρέπει να σημειωθεί ότι το API αυτό εφαρμόζεται και σε προγράμματα που βασίζονται σε λειτουργικά συστήματα πραγματικού χρόνου (RTOS). Η χρήση του HAL API για την υλοποίηση των ISRs έγκειται σε δύο βασικά βήματα που θα περιγραφούν παρακάτω: i) Τη συγγραφή αυτής καθαυτής της ISR που θα χειρίζεται τις διακοπές από μια συγκεκριμένη περιφερειακή συσκευή και ii) την καταχώρηση της ISR από το κυρίως πρόγραμμα-εφαρμογή, καλώντας τη συνάρτηση alt_irq_register(). Η συνάρτηση αυτή ενεργοποιεί τις διακοπές καλώντας με τη σειρά της τη συνάρτηση alt_irq_disable_all().

Συγγραφή της ISR

Το πρωτότυπο της ISR πρέπει έχει την εξής μορφή:

void isr (void* context, alt_u32 id)

ώστε να ταιριάζει με το πρωτότυπο που περιμένει η συνάρτηση alt_irq_register(). Οι ορισμοί των όρων context και id είναι πανομοιότυποι με αυτούς της συνάρτησης alt_irq_register(). Είναι αναγκαίο να τονιστεί ότι για το HAL σύστημα διαχείρισης εξαιρέσεων, η πιο σημαντική εργασία μιας ISR είναι ο "καθαρισμός" (clearing) της κατάστασης διακοπής του αντιστοίχου περιφερειακού. Η διαδικασία αυτή είναι συγκεκριμένη για κάθε είδους περιφερειακή συσκευή. Όταν η ISR έχει πλέον ολοκληρώσει την εξυπηρέτηση της διακοπής πρέπει να επιστρέψει στον HAL διαχειριστή εξαιρέσεων.

Καταχώρηση της ISR

Για να είναι σε θέση το software να εκτελέσει την ISR αυτή πρέπει πρώτα να έχει καταχωρηθεί καλώντας τη συνάρτηση alt_irq_register(), το πρωτότυπο της οποίας είναι :

int alt_irq_register (alt_u32 id,void* context,void (*isr)(void*, alt_u32));

Οι παράμετροι του πρωτοτύπου της συνάρτησης αυτής ορίζονται ως εξής:

- id είναι ο αριθμός της hardware διακοπής για τη συγκεκριμένη συσκευή όπως αυτός ορίζεται στον SOPC Builder φαίνεται στο αρχείο *system.h.* Η προτεραιότητα των διακοπών είναι αντίστροφη των αριθμών τους, υποδηλώνοντας έτσι ότι η IRQ0 έχει την υψηλότερη προτεραιότητα ενώ η IRQ31 την χαμηλότερη.
- context είναι ο δείκτης που χρησιμοποιείται για να περαστεί οποιαδήποτε σχετική πληροφορία στην ISR. Η μεταβλητή context είναι αδιαφανής στο HAL.
- isr είναι ο δείκτης στη συνάρτηση που καλείται και αντιστοιχεί στον αριθμό id της αντίστοιχης IRQ. Τα δύο ορίσματα που εισάγονται στη συνάρτηση αυτή είναι ο δείκτης context και ο αριθμός id. Η καταχώρηση ενός μηδενικού δείκτη στην isr έχει ως αποτέλεσμα την απενεργοποίηση της διακοπής. Η HAL καταχωρεί την ISR αποθηκεύοντας το δείκτη isr της συνάρτησης, σε έναν πίνακα αναφοράς (look-up table). Η συνάρτηση alt_irq_register() επιστρέφει την τιμή μηδέν αν η διαδικασία έχει ολοκληρωθεί επιτυχώς, σε αντίθετη περίπτωση επιστρέφει ένα μη μηδενικό αριθμό. Αν η ISR καταχωρηθεί επιτυχώς η αντίστοιχη Nios II διακοπή (όπως ορίζεται από το id) ενεργοποιείται κατά την επιστροφή από την alt_irq_register().

Ενεργοποίηση και Απενεργοποίηση ISRs

Η HAL προσφέρει τις συναρτήσεις, alt_irq_disable(), alt_irq_enable(), alt_irq_disable_all(), alt_irq_enable_all(), alt_irq_enabled() με τις οποίες είναι δυνατή η απενεργοποίηση των διακοπών για συγκεκριμένες περιοχές - κομμάτια κώδικα και η επα-ενεργοποίηση τους αργότερα. Οι συναρτήσεις alt_irq_disable() και alt_irq_enable() επιτρέπουν την ενεργοποίηση και απενεργοποίηση συγκεκριμένων (μεμονωμένων) διακοπών. Η συνάρτηση alt_irq_disable_all() απενεργοποιεί όλες τις διακοπές και επιστρέφει την τιμή context, ενώ η συνάρτηση alt_irq_enable_all() ενεργοποιεί ξανά όλες τις διακοπές περνώντας την παράμετρο context. Με αυτόν τον τρόπο αυτό οι διακοπές επιστρέφουν στην αρχική τους κατάσταση, δηλαδή πριν την κλήση στην alt_irq_disable_all(). Επίσης η alt_irq_enabled() επιστρέφει μη μηδενική τιμή αν οι διακοπές ενεργοποιηθούν επιτυχώς επιτρέποντας έτσι στο κυρίως πρόγραμμα να ελέγχει την κατάστασή τους.

2.4.4 **Εξαιρέσεις Σχετικές με Εντολές**

Οι εξαιρέσεις σχετικές με εντολές συμβαίνουν κατά τη διάρκεια εκτέλεσης συγκεκριμένων εντολών του Nios II, ενώ οι εργασίες που εκτελούνται είναι ίδιες με αυτές των hardware διακοπών. Ο επεξεργαστής Nios II παράγει τις εξής εξαιρέσεις σχετικές με εντολές :

- Trap εντολή
- Break εντολή
- Μη-Υλοποιημένη εντολή

Trap **Εντολή**

Όταν στο κυρίως πρόγραμμα-εφαρμογή συναντάται η εντολή *trap* παράγεται μία software trap εξαίρεση. Ένα πρόγραμμα συναντά συνήθως μία ***software trap*** όταν ζητά εξυπηρέτηση από ένα λειτουργικό σύστημα. Ο γενικός διαχειριστής διακοπών του λειτουργικού συστήματος θα καθορίσει το λόγο του *trap* και θα ανταποκριθεί καταλλήλως.

Break **Εντολή**

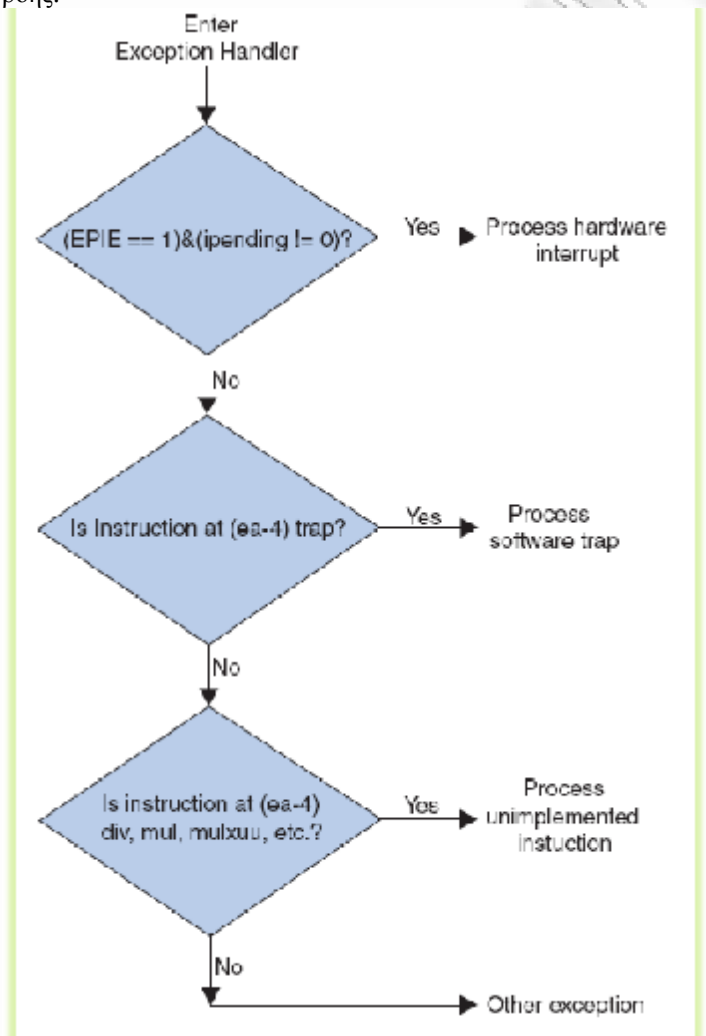
Η εντολή break χειρίζεται όπως η break εξαίρεση που έχει ήδη αναλυθεί .

Μη-Υλοποιημένη Εντολή

Όταν ο επεξεργαστής συναντήσει μία έγκυρη εντολή που δεν είναι υλοποιημένη σε hardware, τότε παράγεται μία εξαίρεση μη-υλοποιημένης εντολής. Ο γενικός διαχειριστής διακοπών καθορίζει ποια εντολή προκάλεσε την εξαίρεση και δίνει τον έλεγχο σε μία ρουτίνα εξαίρεσης η οποία πολύ πιθανόν θα επιλέξει να εξομοιώσει την εντολή σε software.

2.4.5 **Καθορισμός της Πηγής Εξαίρεσης**

Ο γενικός διαχειριστής διακοπών πρέπει να καθορίσει ποιο ήταν το αίτιο που προκάλεσε μία συγκεκριμένη εξαίρεση, ώστε να μπορεί να μεταφέρει τον έλεγχο στην αντίστοιχη ρουτίνα εξυπηρέτησης εξαίρεσης. Η εικόνα 2.4 παρουσιάζει την παραπάνω διαδικασία σε μορφή διαγράμματος ροής.



Εικόνα 2.4 – Διαδικασία εξαιρέσεων

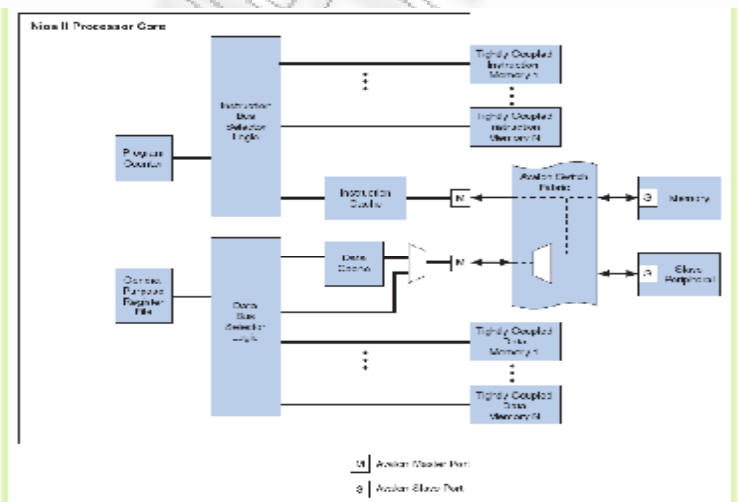
Σύμφωνα με την εικόνα 2.4, εάν το *EPIE* bit του *estatus* καταχωρητή είναι στο λογικό "1" και η τιμή του *ipending* καταχωρητή είναι μη μηδενική, τότε την εξαίρεση την προκάλεσε μία hardware διακοπή. Σε αντίθετη περίπτωση δημιουργήθηκε μία εξαίρεση σχετική με εντολή. Για τον διαχωρισμό μεταξύ *software traps* και *μη-υλοποιημένης εντολής* πρέπει να γίνει ανάγνωση της εντολής στη θέση μνήμης *ea-4*. Αν η εντολή είναι η *trap*, τότε η εξαίρεση είναι μία software trap ενώ αν η εντολή στην θέση αυτή μπορεί να υλοποιηθεί σε software την εξαίρεση την προκάλεσε μία μη-υλοποιημένη εντολή.

2.5 **Οργάνωση Μνήμης και Περιφερειακών**

Η "ευέλικτη" φύση της μνήμης και των περιφερειακών του Nios II είναι η πιο σημαντική διαφορά μεταξύ των Nios II συστημάτων και των παραδοσιακών συστημάτων μικροεπεξεργαστών. Λόγω του ότι τα Nios II συστήματα είναι παραμετροποιήσιμα, τόσο οι μνήμες όσο και οι περιφερειακές συσκευές μπορούν να είναι διαφορετικές από σύστημα σε σύστημα ανάλογα πάντα με τις ανάγκες/προδιαγραφές της εφαρμογής. Ως αποτέλεσμα των παραπάνω, η οργάνωση μνήμης και περιφερειακών αλλάζει επίσης από σύστημα σε σύστημα .Ο επεξεργαστής Nios II χρησιμοποιεί ένα η περισσότερα από τα παρακάτω στοιχεία για να παρέχει πρόσβαση στη μνήμη και τα περιφερειακά :

- Instruction master port – Ένα Avalon-MM master port που συνδέεται με τη μνήμη εντολών μέσω μιας εσωτερικής δομής διασύνδεσης συστήματος (system interconnect fabric).
- Instruction cache – Μία πολύ γρήγορη, εσωτερική, κρυφή μνήμη εντολών του NiosII core.
- Data master port - Ένα Avalon-MM master port που συνδέεται με τη μνήμη δεδομένων και τα περιφερειακά μέσω μιας εσωτερικής δομής διασύνδεσης συστήματος.
- Data cache - Μία πολύ γρήγορη, εσωτερική, κρυφή μνήμη δεδομένων του Nios II core.
- Tightly-coupled instruction or data memory port– Ένας δίαυλος διασύνδεσης με γρήγορη on-chip μνήμη εκτός του Nios II core.

Όπως έχει ήδη αναφερθεί η αρχιτεκτονική του Nios II "κρύβει" τις λεπτομέρειες του hardware, δίνοντας τη δυνατότητα ανάπτυξης εφαρμογών χωρίς ειδική γνώση για την hardware υλοποίηση του συστήματος. Στην εικόνα 2.5 παρουσιάζεται η οργάνωση μνήμης και περιφερειακών του Nios II. Η αρχιτεκτονική του Nios II παρέχει memory-mapped I/O πρόσβαση. Και η μνήμη δεδομένων και τα περιφερειακά είναι χαρτογραφημένα στο χώρο διευθύνσεων (address space) του data master port. Πρέπει επίσης να σημειωθεί ότι η Nios II αρχιτεκτονική είναι little endian. Έτσι τα δεδομένα αποθηκεύονται στη μνήμη με τα πιο σημαντικά bytes στις υψηλότερες/μεγαλύτερες διευθύνσεις. Η αρχιτεκτονική του Nios II δεν ορίζει την ύπαρξη μνήμης και περιφερειακών. Η ποσότητα, ο τύπος και η διασύνδεση μνήμης και περιφερειακών συσκευών εξαρτώνται από το σύστημα και κατά προέκταση από την εφαρμογή προς υλοποίηση. Συνήθως τα βασιζόμενα στον Nios II συστήματα περιλαμβάνουν ένα συνδυασμό γρήγορης on-chip



Εικόνα 2.5 - Η οργάνωση μνήμης και περιφερειακών του Nios II

μνήμης και πιο αργής off-chip μνήμης. Κατά κανόνα τα περιφερειακά βρίσκονται μέσα στο chip (Cyclone II). Υπάρχει όμως και διασύνδεση με off-chip περιφερειακά.

2.5.1 **Δίαυλοι Δεδομένων και Εντολών**

Η αρχιτεκτονική του Nios II κατατάσσεται στην Harvard αρχιτεκτονική, υποστηρίζοντας ξεχωριστούς διαύλους για δεδομένα και εντολές. Και οι δύο δίαυλοι υλοποιούνται ως Avalon-MM master ports που υπακούουν στις προδιαγραφές του Avalon-MM interface. Το data master port συνδέεται στα chip μνήμης αλλά και στις περιφερειακές συσκευές, ενώ το instruction master port συνδέεται μόνο στα chip μνήμης.

Instruction Master Port

Ο δίαυλος εντολών του Nios II υλοποιείται ως ένα Avalon-MM master port των 32-bit. Το instruction master port εκτελεί μια απλή λειτουργία: παρέχει στον επεξεργαστή εντολές προς εκτέλεση ενώ δεν εκτελεί καμία λειτουργία εγγραφής. Το instruction master port είναι ένα pipelined Avalon-MM master port. Η υποστήριξη pipelined Avalon-MM μεταφορών δεδομένων, ελαχιστοποιεί την καθυστέρηση της σύγχρονης μνήμης (synchronous memory), ενώ αυξάνει τη συνολική συχνότητα f_{MAX} του συστήματος. Το instruction master port μπορεί να υλοποιήσει διαδοχικές αιτήσεις ανάγνωσης προτού επιστραφούν τα δεδομένα από προηγούμενες αιτήσεις. Ο Nios II μπορεί να προ-τροφοδοτήσει (prefetch) διαδοχικές εντολές και να εκτελέσει branch prediction ώστε να κρατήσει το pipeline εντολών όσο πιο ενεργό γίνεται. Το instruction master port αλληλεπιδρά πάντα με δεδομένα των 32-bit και βασίζεται στη λογική δυναμικής αλλαγής μεγέθους του διαύλου, που εμπεριέχεται στην εσωτερική δομή διασύνδεσης του συστήματος (system interconnect fabric). Λόγω αυτής της λογικής, κάθε ανάκληση εντολής επιστρέφει μία λέξη εντολής (full instruction word) ανεξάρτητα από το εύρος της μνήμης. Συνεπώς δεν είναι αναγκαίο, στα Nios II συστήματα, να είναι γνωστό το εύρος της μνήμης. Η Nios II αρχιτεκτονική υποστηρίζει επίσης τη χρήση on-chip κρυφής μνήμης με σκοπό τη βελτίωση της απόδοσης κατά την ανάκτηση εντολών από μία πιο αργή μνήμη καθώς και tightly-coupled μνήμες που παρέχουν χαμηλής καθυστέρησης πρόσβαση στην on-chip μνήμη.

Data Master Port

Ο δίαυλος δεδομένων του Nios II υλοποιείται ως ένα Avalon-MM master port των 32-bit. Το Data master port εκτελεί δύο λειτουργίες: i) διαβάζει δεδομένα από μνήμες ή περιφερειακές συσκευές, όταν ο επεξεργαστής εκτελεί μία εντολή φόρτωσης (load instruction) και ii) γράφει δεδομένα σε μνήμες ή περιφερειακές συσκευές όταν ο επεξεργαστής εκτελεί μία εντολή αποθήκευσης (store instruction). Οι λειτουργίες εγγραφής και αποθήκευσης μπορούν να ολοκληρωθούν μέσα σε ένα κύκλο ρολογιού όταν το data master port είναι συνδεδεμένο με μία μνήμη μηδενικής κατάστασης αναμονής (zero-wait-state memory).

Κοινή Μνήμη για Εντολές και Δεδομένα

Συνήθως, τα instruction master port και data master port μοιράζονται μία μνήμη η οποία περιέχει και εντολές και δεδομένα. Ενώ ο επεξεργαστής έχει ξεχωριστούς δίαυλους για εντολές και δεδομένα, το συνολικό Nios II σύστημα μπορεί να παρουσιάζει έναν μόνο κοινό δίαυλο δεδομένων / εντολών “προς τα έξω”. Τέλος, τα instruction master port και data master port λειτουργούν με εντελώς ανεξάρτητο τρόπο, χωρίς να αλληλεπιδρούν μεταξύ τους.

2.5.2 **Κρυφή Μνήμη**

Η Nios II αρχιτεκτονική υποστηρίζει κρυφές μνήμες (cache memories) και για το instruction master port (instruction cache) και για το data master port (data cache). Η κρυφή μνήμη βρίσκεται μέσα στο chip ως ένα αναπόσπαστο κομμάτι του Nios II core. Οι κρυφές μνήμες μπορούν να βελτιώσουν το μέσο χρόνο πρόσβασης μνημών σε ένα Nios II σύστημα που χρησιμοποιεί αργές off-chip μνήμες όπως η SDRAM για αποθήκευση δεδομένων. Οι κρυφές μνήμες εντολών και δεδομένων είναι συνεχώς ενεργοποιημένες κατά τη διάρκεια λειτουργίας του συστήματος. Το σετ εντολών του Nios II παρέχει εντολές για τη διαχείριση της κρυφής μνήμης.

Ρύθμιση Παραμέτρων Κρυφής Μνήμης

Η ανάγκη για μνήμη υψηλής απόδοσης και κατά συνέπεια η ανάγκη για κρυφή μνήμη, εξαρτάται από την εφαρμογή. Πολλές εφαρμογές απαιτούν το μικρότερο δυνατό μέγεθος core επεξεργαστή και μπορούν να θυσιάσουν την απόδοση για το σκοπό αυτό, υπονοώντας ότι η χρήση της κρυφής μνήμης είναι προαιρετική. Το core του Nios II μπορεί να περιέχει μία, και τις δύο ή καμία από τις κρυφές μνήμες δεδομένων και εντολών. Η εισαγωγή της κρυφής μνήμης σε ένα σύστημα δεν επηρεάζει τη λειτουργικότητα των προγραμμάτων, αλλάζει όμως εντελώς την ταχύτητα με την οποία ο επεξεργαστής ανακαλεί εντολές και γράφει/διαβάζει δεδομένα.

Απόδοση Κρυφής Μνήμης

Η περαιτέρω βελτίωση της απόδοσης της κρυφής μνήμης και κατά συνέπεια του συστήματος, βασίζεται στις παρακάτω προϋποθέσεις:

- Η συνήθης μνήμη είναι εκτός chip και ο χρόνος πρόσβασης σε αυτήν είναι πολύ μεγαλύτερος από την on-chip μνήμη.
- Ο μεγαλύτερος βρόχος εντολών (κρίσιμης σημασίας για την απόδοση του συστήματος) είναι μικρότερος από το μέγεθος της κρυφής μνήμης εντολών.
- Το μεγαλύτερο block δεδομένων εντολών (κρίσιμης σημασίας για την απόδοση του συστήματος) είναι μικρότερο από το μέγεθος της κρυφής μνήμης εντολών.

Η αποδοτικότερη ρύθμιση της κρυφής μνήμης εξαρτάται τόσο από το σύστημα όσο και από την εφαρμογή, ωστόσο υπάρχουν καθολικές ρυθμίσεις που επηρεάζουν μεγάλες ομάδες συστημάτων / εφαρμογών. Για παράδειγμα, εάν το NiosII σύστημα περιλαμβάνει μόνο γρήγορες on-chip μνήμες (χωρίς ποτέ να προσπελαύνει off-chip μνήμες) μία κρυφή μνήμη δεδομένων ή εντολών πολύ πιθανόν να μην προσφέρει καμία βελτίωση στην απόδοση του συστήματος. Επίσης, εάν ο κρίσιμος βρόχος εντολών είναι 2Kbytes αλλά η κρυφή μνήμη εντολών είναι μεγέθους 1kByte, τότε η κρυφή μνήμη αυτή δεν θα επιφέρει καμία αύξηση στην ταχύτητα εκτέλεσης, ενδέχεται δε να υποβαθμίσει την ολική απόδοση του συστήματος.

Μέθοδοι Παράκαμψης Κρυφής Μνήμης

Ο Nios II παρέχει δύο μεθόδους παράκαμψης της κρυφής μνήμης : i) I/O εντολές φόρτωσης και αποθήκευσης – εντολές όπως *ldio* και *stio* παρακάμπτουν την κρυφή μνήμη δεδομένων και επιβάλλουν μία Avalon– MM μεταφορά δεδομένων σε/από μία συγκεκριμένη διεύθυνση μνήμης, ii) την bit-31 cache bypass μέθοδο, η οποία χρησιμοποιεί στο data master port το bit-31 της διεύθυνσης ως ετικέτα που υποδεικνύει αν ο επεξεργαστής πρέπει να μεταφέρει δεδομένα από/προς την κρυφή μνήμη, ή να την παρακάμψει. Εάν η εφαρμογή απαιτεί συνεχώς συγκεκριμένα δεδομένα ή κομμάτια κώδικα να βρίσκονται στην κρυφή μνήμη, τότε για λόγους απόδοσης είναι πιο δόκιμη η χρήση tightly-coupled μνήμης.

2.5.3 Tightly Coupled **Μνήμη**

Η tightly-coupled μνήμη παρέχει χαμηλής καθυστέρησης πρόσβαση σε μνήμες για εφαρμογές που απαιτούν υψηλή απόδοση. Συγκρινόμενη με την κρυφή μνήμη, η tightly-coupled μνήμη παρουσιάζει τα παρακάτω πλεονεκτήματα:

- Απόδοση εφάμιλλης της κρυφής μνήμης.
- Κρίσιμα δεδομένα ή κομμάτια κώδικα μπορούν να επιβεβαιωθούν ότι βρίσκονται στην tightly-coupled μνήμη μέσω software.
- Δεν υπάρχει επιβάρυνση (σε πραγματικό χρόνο) για λειτουργίες μνήμης όπως φόρτωση, ακύρωση ή άδειασμα μνήμης.

Η θύρα tightly-coupled μνήμης (tightly-coupled memory port) είναι ένα ξεχωριστό master port στο core του Nios II παρόμοιο με το instruction master port και το data master port. Ο Nios II μπορεί να έχει καμία, μία ή πολλαπλές tightly-coupled μνήμες. Η Nios II αρχιτεκτονική υποστηρίζει tightly-

coupled μνήμες και για εντολές και για δεδομένα. Κάθε θύρα tightly-coupled μνήμης συνδέεται απευθείας με μία μόνο μνήμη, παρέχοντας συγκεκριμένη (χαμηλή) καθυστέρηση πρόσβασης. Η μνήμη αυτή είναι έξω από το core του Nios II και συνήθως μέσα στο chip (on-chip).

Πρόσβαση σε Tightly-Coupled **Μνήμη**

Η tightly-coupled μνήμη καταλαμβάνει συνήθεις θέσεις διευθύνσεων όπως άλλες μνήμες συνδεδεμένες στην εσωτερική δομή διασύνδεσης. Το εύρος διευθύνσεων για την tightly-coupled μνήμη (αν υπάρχει) καθορίζεται κατά την παραγωγή του συστήματος. Η software πρόσβαση σε tightly-coupled μνήμη, επιτυγχάνεται με τις συνήθεις εντολές φόρτωσης και αποθήκευσης και δε διαφέρει από την πρόσβαση σε άλλες μνήμες.

Απόδοση Tightly-Coupled **Μνήμης**

Το σύστημα μπορεί να χρησιμοποιήσει μία tightly-coupled μνήμη για την επίτευξη μέγιστης απόδοσης όσον αφορά την πρόσβαση σε συγκεκριμένα δεδομένα ή κομμάτια κώδικα. Για παράδειγμα, έντονες υπολογιστικά εφαρμογές όπως εφαρμογές ψηφιακής επεξεργασίας σήματος μπορούν να διαμοιράσουν τους buffers δεδομένων σε tightly-coupled μνήμες, για την όσο το δυνατόν γρηγορότερη πρόσβαση στα δεδομένα αυτά.

2.5.4 **Χάρτης Διευθύνσεων** (Address Map)

Ο χάρτης διευθύνσεων για μνήμες και περιφερειακά σε ένα Nios σύστημα, προσαρμόζεται αναλόγως και καθορίζεται κατά την παραγωγή του συστήματος. Υπάρχουν όμως τρεις διευθύνσεις οι οποίες αποτελούν μέρος του επεξεργαστή και χρήζουν ιδιαίτερης προσοχής. Αυτές είναι: i) διεύθυνση επαναφοράς (reset address) ii) διεύθυνση εξαίρεσης (exception address) και iii) διεύθυνση break χειριστή (break handler access).
Ο προσαρμοζόμενος χάρτης διευθύνσεων δεν επηρεάζει την ανάπτυξη των εφαρμογών, αφού η πρόσβαση σε μνήμες και περιφερειακά επιτυγχάνεται με τη χρήση μακροεντολών (macros) και οδηγών-συναρτήσεων (drivers).

2.6 **Μονάδα αποσφαλμάτωσης** JTAG (JTAG debug module)

Η Nios II αρχιτεκτονική υποστηρίζει τη μονάδα αποσφαλμάτωσης JTAG η οποία παρέχει on-chip χαρακτηριστικά εξομοίωσης για τον έλεγχο του επεξεργαστή από έναν απομακρυσμένο υπολογιστή. Το Quartus II και το Nios II IDE επικοινωνούν με τη μονάδα αποσφαλμάτωσης JTAG και παρέχουν λειτουργίες όπως:

- Προγραμματισμός του FPGA με το αρχείο που προέκυψε από τη μεταγλώττιση του συστήματος (.sof, .pof).
- Προγραμματισμός του Nios II με το αρχείο που προέκυψε από τη μεταγλώττιση του προγράμματος/εφαρμογής (.elf).
- Έναρξη και τερματισμός εκτέλεσης προγράμματος.
- Εισαγωγή breakpoints και watchpoints.
- Ανάλυση περιεχομένων καταχωρητών και μνήμης.

Η μονάδα αποσφαλμάτωσης συνδέεται στο JTAG κύκλωμα του FPGA ενώ από την πλευρά του επεξεργαστή η μονάδα αποσφαλμάτωσης συνδέεται με εσωτερικά σήματα του core του επεξεργαστή. Όλοι οι πόροι συστήματος του επεξεργαστή είναι διαθέσιμοι και στη μονάδα αποσφαλμάτωσης. Η μονάδα αποσφαλμάτωσης αποκτά τον έλεγχο του επεξεργαστή είτε με την εισαγωγή ενός hardware break σήματος ή με την εγγραφή μίας break εντολής στη μνήμη του προγράμματος προς εκτέλεση. Και στις δύο περιπτώσεις, ο επεξεργαστής μεταφέρει τον έλεγχο σε μία ρουτίνα που βρίσκεται στη διεύθυνση "***break***". Η διεύθυνση αυτή καθορίζεται κατά την παραγωγή του συστήματος.

Οι soft-core επεξεργαστές, όπως ο Nios II, προσφέρουν επιπλέον δυνατότητες συγκρινόμενοι με τους παραδοσιακούς hard-core επεξεργαστές. Για παράδειγμα, ο Nios II επεξεργαστής παρέχει τη δυνατότητα αποσφαλμάτωσης ενός συστήματος που είναι σε ανάπτυξη, χρησιμοποιώντας ένα πλήρες debug core, επιτρέποντας τη μερική ή πλήρη αφαίρεσή του στο τελικό design για τη διατήρηση πόρων του συστήματος.

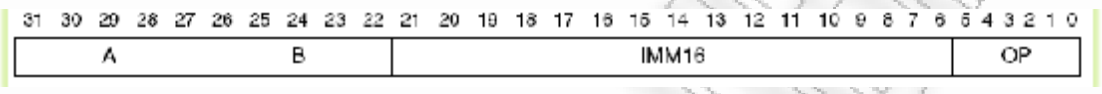
2.7 **Εισαγωγή στο Σετ Εντολών του** Nios II

Όλες οι εντολές που υποστηρίζει ο Nios II ανήκουν σε μία εκ των τριών κατηγοριών: I-Type, R-Type και J-Type. Ο κάθε τύπος εντολών ακολουθεί ξεχωριστή αναπαράσταση εντολών (instruction-word format).

2.7.1 I Type

Οι I-Type εντολές είναι εντολές αριθμητικών και λογικών πράξεων όπως addi και andi. Το χαρακτηριστικό του I-Type instruction-word format, που φαίνεται στην εικόνα, είναι ότι περιέχει μία άμεση τιμή (immediate value) ενσωματωμένη στο instruction word. Οι εντολές I-Type αποτελούνται από :

- Ένα πεδίο opcode (OP) των 6-bit.
- Ένα πεδίο άμεσης τιμής (IMM16) των 16-bit.
- πεδία καταχωρητών (A και B) των 5-bit.

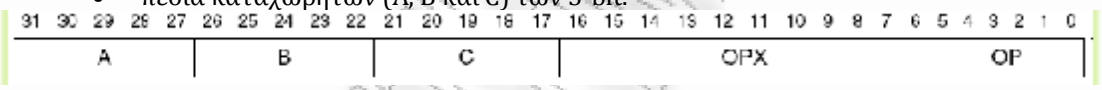


2.7.2 R Type

Οι R-Type εντολές είναι εντολές αριθμητικών και λογικών πράξεων όπως add και nor, πράξεων σύγκρισης όπως cmpeq και cmplt και άλλων πράξεων που απατούν ορίσματα από καταχωρητές Το χαρακτηριστικό του R-Type instruction-word format, που φαίνεται στην εικόνα, είναι ότι όλα τα ορίσματα και τα αποτελέσματα ορίζονται ως καταχωρητές.

Οι εντολές R-Type αποτελούνται από :

- Ένα πεδίο opcode (OP) των 6-bit.
- Ένα πεδίο opcode-extension (OPX) των 11-bit.
- πεδία καταχωρητών (A, B και C) των 5-bit.



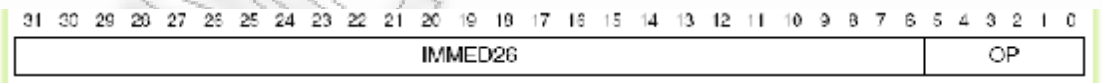
Στις περισσότερες περιπτώσεις τα πεδία A και B ορίζουν του καταχωρητές που παρέχουν τα ορίσματα της πράξης και το πεδίο C ορίζει τον καταχωρητή αποθήκευσης του αποτελέσματος.

2.7.3 J Type

Οι J-Type εντολές μεταφέρουν την εκτέλεση του προγράμματος σε κάποια άλλη διεύθυνση σε διάστημα 256Mbytes. Τέτοιες εντολές είναι η call, η jmpi και άλλες.

Το J-Type instruction-word format, που φαίνεται στην εικόνα, αποτελείται από :

- Ένα πεδίο opcode (OP) των 6-bit.
- Ένα πεδίο άμεσων δεδομένων (IMMED26) των 26-bit.



2.7.4 **Τα** OPcodes **των Εντολών**

Τα πεδία OP και OPX, στις Nios II εντολές, καθορίζουν το opcode τους όπως παρουσιάζονται στους πίνακες εικόνες 2.6 και 2.7.

OP	Εντολή	OP	Εντολή	OP	Εντολή	OP	Εντολή
0x00	call	0x10	cmplti	0x20	cmpeqi	0x30	cmpltui
0x01	jmpi	0x11		0x21		0x31	
0x02		0x12		0x22		0x32	custom
0x03	ldbu	0x13		0x23	ldbuio	0x33	initd
0x04	addi	0x14	ori	0x24	muli	0x34	orhi
0x05	stb	0x15	stw	0x25	stbio	0x35	stwio
0x06	br	0x16	blt	0x26	beq	0x36	bltu
0x07	ldb	0x17	ldw	0x27	ldbio	0x37	ldwio
0x08	cmpgei	0x18	cmpnei	0x28	cmpgeui	0x38	
0x09		0x19		0x29		0x39	
0x0A		0x1A		0x2A		0x3A	R-TYPE
0x0B	ldhu	0x1B	flushda	0x2B	ldhuio	0x3B	flushd
0x0C	andi	0x1C	xori	0x2C	andhi	0x3C	xorhi
0x0D	sth	0x1D		0x2D	sthio	0x3D	
0x0E	bge	0x1E	bne	0x2E	bgeu	0x3E	
0x0F	ldh	0x1F		0x2F	ldhio	0x3F	

Εικόνα 2.6 – Οι εντολές του NIOS II

OPX	Εντολή	OPX	Εντολή	OPX	Εντολή	OPX	Εντολή
0x00		0x10	cmplt	0x20	cmpeq	0x30	cmpltu
0x01	eret	0x11		0x21		0x31	add
0x02	roli	0x12	slli	0x22		0x32	
0x03	rol	0x13	sll	0x23		0x33	
0x04	flushp	0x14		0x24	divu	0x34	break
0x05	ret	0x15		0x25	div	0x35	
0x06	nor	0x16	or	0x26	rdctl	0x36	sync
0x07	mulxuu	0x17	mulxsu	0x27	mul	0x37	
0x08	cmpge	0x18	cmpne	0x28	cmpgeu	0x38	
0x09	bret	0x19		0x29	initi	0x39	sub
0x0A		0x1A	srli	0x2A		0x3A	srai
0x0B	nor	0x1B	srl	0x2B		0x3B	sra
0x0C	flushi	0x1C	nextpc	0x2C		0x3C	
0x0D	jmp	0x1D	callr	0x2D	trap	0x3D	
0x0E	and	0x1E	xor	0x2E	wrctl	0x3E	
0x0F		0x1F	mulxss	0x2F		0x3F	

Εικόνα 2.7 - Οι εντολές του NIOS II

Κεφάλαιο 3

Επεξεργαστές Ειδικού Σκοπού

3.1 Επεξεργαστές Ειδικού Συνόλου Εντολών

Για την βελτίωση των επιδόσεων των ενσωματωμένων συστημάτων πέρα από τις τεχνικές χρησιμοποίησης επεξεργαστών γενικού σκοπού, στα πλαίσια της σχεδίασης σε επίπεδο συστήματος (System Level Design), αναπτύσσονται επεξεργαστές ειδικού συνόλου εντολών. Οι προγραμματιζόμενοι επεξεργαστές Ειδικού Συνόλου Εντολών παρέχουν την δυνατότητα για πιο οικονομική λύση στην ανάπτυξη υπολογιστικά απαιτητικών ενσωματωμένων συστημάτων. Οι λύσεις ASIC έχουν τη δυνατότητα να επιτυγχάνουν υψηλή ταχύτητα επεξεργασίας σε συνδυασμό με χαμηλή κατανάλωση ισχύος, χωρίς όμως να είναι εφικτός ο επαναπρογραμματισμός τους. Ένα ASIC εκτελεί μόνο τους αλγορίθμους οι οποίοι είχαν προβλεφθεί κατά το σχεδιασμό του. Επίσης ο χρόνος ανάπτυξής τους είναι απαγορευτικός για όποιες μεταγενέστερες μετατροπές. Οι επεξεργαστές γενικού σκοπού έχουν την δυνατότητα προγραμματισμού τους, πλην όμως παρουσιάζουν πολλές φορές χαμηλές επιδόσεις, σε σχέση με τις απαιτήσεις των ενσωματωμένων συστημάτων. Η υποστήριξη ειδικών εντολών επέκτασης από τον επεξεργαστή επιφέρει πολλές βελτιώσεις στα χαρακτηριστικά του επεξεργαστή. Κυρίως μπορεί να αυξήσει την επεξεργαστική του ισχύ. Αυτό μπορεί να συμβεί μειώνοντας την καθυστέρηση των σταδίων διοχέτευσης, τα οποία προκαλούνται από *ασυνέπεια δεδομένων* (data hazard). Επίσης μειώνεται το μέγεθος του κώδικα του προγράμματος, αυτό έχει ως αποτέλεσμα μικρότερες απαιτήσεις σε μνήμη εντολών, αλλά και μικρότερη κατανάλωση ενέργειας, αφού οι προσπελάσεις στην μνήμη είναι αρκετά ενεργοβόρες. Έτσι σχεδιάζονται επεξεργαστές με βάση το λογισμικό που πρόκειται να εκτελέσουν, έχοντας υπόψη τα ιδιαίτερα χαρακτηριστικά των εφαρμογών, ώστε να ικανοποιηθούν οι απαιτήσεις σε χρόνο, ισχύ και επιφάνεια υλικού. Μία συμπληρωματική τεχνική είναι η προσθήκη ειδικών εντολών για την εκτέλεση των πιο απαιτητικών τμημάτων του κώδικα, ή η προσθήκη επαναδιαμορφούμενων συνεπεξεργαστών που μπορούν να χρησιμοποιηθούν συμπληρωματικά. Τέτοιες λύσεις υπάρχουν από την ARM (OptimoDE), την MIPS (CorExtend) και σε κατασκευαστές FPGAs. Μία τέτοια διαδικασία παρουσιάζεται στα άρθρα [53],[57].

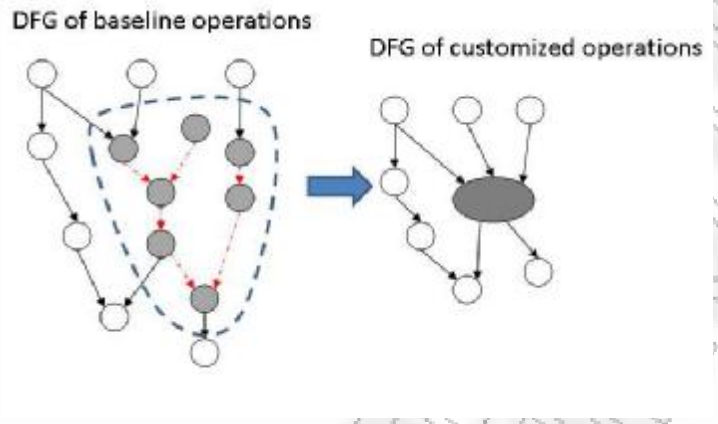
3.2 Διαδικασία για την ανάπτυξη ενός επεξεργαστή Ειδικού Συνόλου Εντολών

Για να φτάσουμε από τον κώδικα μιας εφαρμογής στην σχεδίαση ενός επεξεργαστή Ειδικού Συνόλου Εντολών ακολουθούνται τα παρακάτω βήματα:

1. Καθορίζονται οι περιορισμοί που έχουμε σε χρόνο εκτέλεσης, κατανάλωση ισχύος, επιφάνεια υλικού. Τις περισσότερες φορές δεν είναι δυνατό να ικανοποιηθούν ταυτόχρονα όλοι οι στόχοι, έτσι επιλέγουμε τους πιο σημαντικούς δίνοντας κατάλληλες προτεραιότητες.
2. Αναλύεται ο κώδικας με σκοπό τα συχνά επαναλαμβανόμενα τμήματα να υλοποιηθούν με ειδικές εντολές ή επιταχυντή υλικού. [48],[49] Στο στάδιο της ανάλυσης του κώδικα από την γλώσσα υψηλού επιπέδου C, εξετάζουμε θέματα όπως την παραλληλία της εφαρμογής, συχνότητα εκτέλεσης τμημάτων του κώδικα, εύρος μεταβλητών, τύπους δεδομένων. Σε αυτές τις αναλύσεις στηρίζονται πολλά εργαλεία παραγωγής κώδικα HDL από κώδικα C, όπως ο C to Hardware Compiler της ALTERA. Πολλές φορές παράγεται ένα ενδιάμεσο

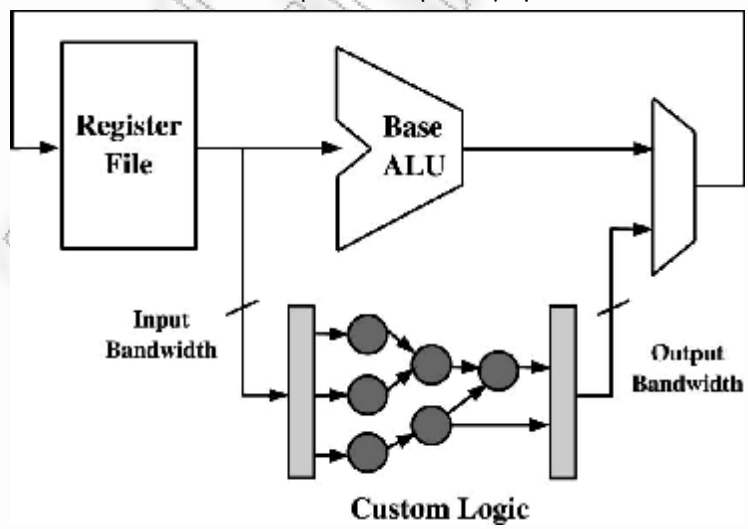
στάδιο μοντελοποίησης TLM (Transaction Level Modeling) που μας επιτρέπει την αναπαράσταση του συστήματος σε επίπεδο επικοινωνίας των επεξεργαστικών μονάδων με σκοπό την σχεδίαση του συστήματος από επίπεδο System Level Design.

3. Ανάλυση των πιθανών αρχιτεκτονικών λύσεων από πλευράς επίτευξης των στόχων. Ανάλογα με τα συμπεράσματα του προηγούμενου σταδίου καθορίζονται τα αρχιτεκτονικά χαρακτηριστικά του επεξεργαστή όπως ο αριθμός των πολλαπλασιαστών, ο αριθμός των σταδίων διοχέτευσης, ο αριθμός των παράλληλων επεξεργαστικών μονάδων, η προσθήκη ή όχι επιταχυντή υλικού.
4. Αναπτύσσονται οι ειδικές εντολές και λειτουργικές μονάδες.[50] Από τον κώδικα της εφαρμογής εξάγεται το CDFG (Control-Data Flow Graph) διάγραμμα, έτσι έχουμε εικόνα για τις εξαρτήσεις ελέγχου και δεδομένων, τους τύπους δεδομένων, τις τοπικές και καθολικές μεταβλητές, το μέγεθος του βασικού μπλοκ, ο αριθμός των λειτουργιών πολλαπλασιασμού και συσσώρευσης, στοιχεία που μας βοηθούν να εντοπίσουμε τα τμήματα του κώδικα που μπορούν να υλοποιηθούν με ειδικές εντολές ή με επιταχυντή υλικού.



Εικόνα 3.1 – Ανάλυση – Μετατροπή του διαγράμματος Ροής - Δεδομένων

5. Γίνεται η σύνθεση και η δοκιμή του υλικού του επεξεργαστή. Εδώ είτε επιλέγουμε εντολές από μία βιβλιοθήκη [36],[45],[55] είτε γίνεται η σύνθεσή τους από την αρχή. Η επιλογή εξαρτάται από τους στόχους που έχουν τεθεί στο στάδιο 1, έχοντας υπόψη την επίτευξη κώδικα με τον μικρότερο αριθμό εντολών, την μεγαλύτερη επαναχρησιμοποίηση των ειδικών εντολών και με μικρό κύκλο εκτέλεσης. Εδώ μπορεί να εξεταστεί η περίπτωση ειδικών εντολών πολλαπλών ορισμάτων εξόδου, όπως και πολλαπλών κύκλων εκτέλεσης. Η δυνατότητα επέκτασης ενός επεξεργαστή σε FPGA όπως ο NIOS II μας επιτρέπει την εισαγωγή ειδικών εντολών απλού κύκλου, πολλαπλού κύκλου ή με διοχέτευση, υλοποίηση του πολλαπλασιαστή σε υλικό ή σε λογισμικό.



Εικόνα 3.2 – Προσθήκη υλικού ειδικών εντολών

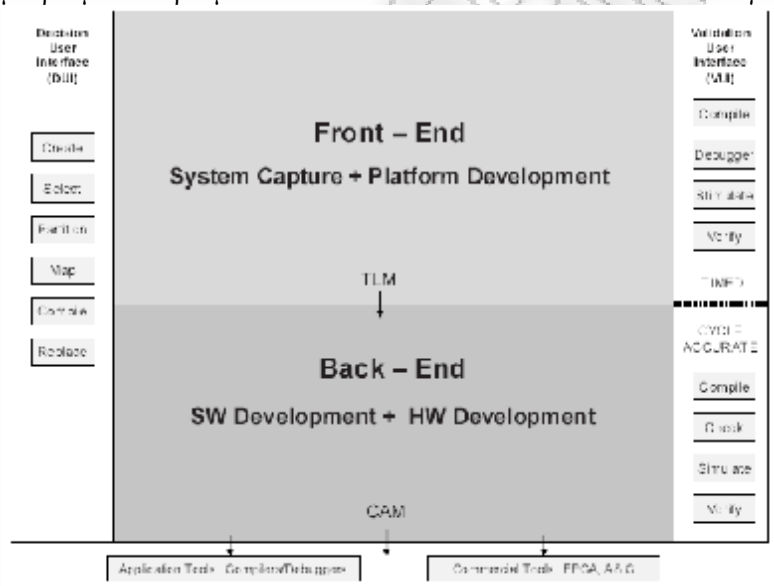
6. Γίνεται η σύνθεση και η δοκιμή του λογισμικού. [49] Εδώ με βάση της νέες εντολές που έχουμε προσθέσει ή τα τμήματα κώδικα που εξυπηρετείται από υλικό, ο νέος κώδικας μεταγλωττίζεται ξανά για το σύνολο εντολών που παρήχθησαν. Τα εργαλεία ανάπτυξης λογισμικού του επεξεργαστή πρέπει να μας επιτρέπουν την εισαγωγή νέων εντολών. Πολλές τεχνολογίες FPGAs υποστηρίζουν soft instruction επεξεργαστές, όπου μία ή περισσότερες soft εντολές μπορούν να υλοποιηθούν χρησιμοποιώντας επαναπροσδιοριζόμενους πόρους, σε ένα μονό chip. Επεξεργαστές όπως ο Xilinx Microblaze και ο Altera Nios είναι διαθέσιμοι από τους FPGA κατασκευαστές. Συχνά αυτοί οι επεξεργαστές υποστηρίζουν προσαρμογή (customization) των πόρων και ειδικές εντολές (custom instructions). Οι ειδικές εντολές έχουν δύο σημαντικά πλεονεκτήματα. Πρώτον, μειώνουν τον χρόνο για το instruction fetch και decode, δεδομένου ότι μια ειδική εντολή αντικαθιστά πολλές κανονικές εντολές. Δεύτερον, επιπλέον πόροι μπορεί να ανατεθούν σε μια ειδική εντολή για να βελτιωθεί η απόδοση.

3.3 **Εργαλεία σχεδίασης επεξεργαστών σε επίπεδο συστήματος**

Για την διαδικασία σχεδίασης σε επίπεδο συστήματος [71] χρειαζόμαστε δύο τύπους εργαλείων (design tools):

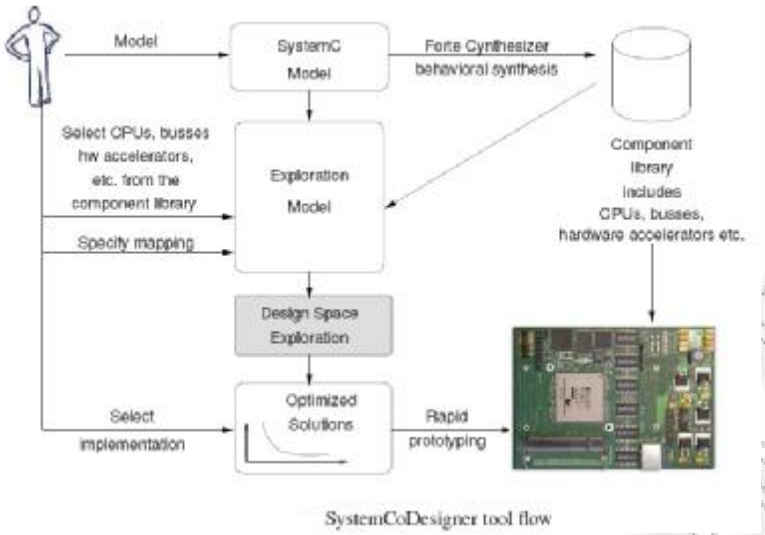
A) front-end tools, τα οποία είναι τα εργαλεία για τους σχεδιαστές συστημάτων που θέλουν να ελέγχουν την σχεδιαστική τους ιδέα. Δέχονται σαν είσοδο την συμπεριφορά του συστήματος σαν ένα υπολογιστικό μοντέλο (MoC) σε μία γλώσσα όπως C, C++, SystemC, Matlab, UML, ή μία αντίστοιχη γραφική αναπαράσταση πχ Simulink και παράγουν ένα μοντέλο TLM (Transaction Level Modeling) για την χρονική ή λειτουργική συμπεριφορά του συστήματος.

B) back-end tools, τα οποία είναι εργαλεία για την ανάπτυξη του υλικού και του λογισμικού του συστήματος. Τα εργαλεία αυτά δέχονται σαν είσοδο το TLM και παράγουν μοντέλα σε HDL για την περιγραφή του υλικού και Instruction Set μοντέλα για την περιγραφή του λογισμικού σύμφωνα με τον επιλεγμένη βασική αρχιτεκτονική επεξεργαστή. Τα δύο αυτά μοντέλα μπορούν να χρησιμοποιηθούν για συν-προσομοίωση λειτουργίας του υλικού και του λογισμικού. Τέλος ανάλογα με τα αποτελέσματα μπορούμε να πάρουμε τον τελικό HDL κώδικα του υλικού και του λογισμικού.



Εικόνα 3.3 – Διαδικασία ανάπτυξης σε επίπεδο συστήματος

Τέτοια εργαλεία ανάπτυξης είναι τα ακόλουθα: **Metropolis, SystemCoDesigner, DAEDALUS, PeaCE(Ptolemy), SCE**. Ένα παράδειγμα της ροής ανάπτυξης για το SystemCoDesigner είναι το ακόλουθο:



Εικόνα 3.4 – Διαδικασία ανάπτυξης σε επίπεδο συστήματος με το SystemCoDesigner

Σαν είσοδο δέχεται υπολογιστικά μοντέλα τύπου ροής δεδομένων δηλωμένα σε γλώσσα SystemC. Οι εφαρμογές μοντελοποιούνται σαν διαγράμματα ατομικών ενεργοποιητών (atomic actors) οι οποίοι επικοινωνούν με ουρές FIFO. Εσωτερικά η συμπεριφορά του κάθε ενεργοποιητή περιγράφεται με ένα διάγραμμα FSMD (Finite State Machine+Data). Ακολούθως παράγεται μια βιβλιοθήκη με τις υλοποιήσεις των ενεργοποιητών σε υλικό (Forte Synthesizer) και σε λογισμικό. Για κάθε έναν συνδυασμό πιθανών υλοποιήσεων παράγεται ένα TLM μοντέλο και γίνεται έλεγχος των προδιαγραφών με προσομοίωση του συστήματος, όπου και επιλέγεται η καλύτερη λύση για την τελική υλοποίηση.

Κεφάλαιο 4

Το υλικό του συστήματος

4.1 Περιγραφή λειτουργίας του συστήματος

Όπως έχει αναφερθεί σκοπός του συστήματος είναι η βελτιστοποίηση του χρόνου εκτέλεσης αλγορίθμου FIR. Αυτό μελετάτε με τους εξής τρόπους – mode λειτουργίας:

Record mode – ηχογραφούμε ηχητικά σήματα 2 καναλιών και τα αποθηκεύουμε στην SDRAM
Play mode – αναπαράγουμε τα ηχογραφημένα σήματα διαβάζοντας τα από την SDRAM
Stand by mode – το σύστημα δεν κάνει τίποτα και περιμένει εντολή.
Sound SW Float mode – διαβάζουμε τα ηχογραφημένα σήματα από την SDRAM, τα επεξεργαζόμαστε σύμφωνα με τον FIR αλγόριθμο χρησιμοποιώντας αριθμητική κινητής υποδιαστολής (οι συντελεστές του φίλτρου είναι δεκαδικοί αριθμοί), και τα αναπαράγουμε επεξεργασμένα. Στην οθόνη εμφανίζουμε τον χρόνο επεξεργασίας των ηχητικών σημάτων.
Sound SW INT mode – διαβάζουμε τα ηχογραφημένα σήματα από την SDRAM, τα επεξεργαζόμαστε σύμφωνα με τον FIR αλγόριθμο χρησιμοποιώντας αριθμητική ακεραίων (οι συντελεστές του φίλτρου είναι ακέραιοι αριθμοί), και τα αναπαράγουμε επεξεργασμένα. Στην οθόνη εμφανίζουμε τον χρόνο επεξεργασίας των ηχητικών σημάτων.
Sound HW ACC mode - διαβάζουμε τα ηχογραφημένα σήματα από την SDRAM, τα αποστέλλουμε μέσω της PIO σε εξωτερικό κύκλωμα FIR επεξεργασίας και αυτά επεξεργασμένα διαβάζονται από τον NIOS II , αποθηκεύονται στην SDRAM και τα αναπαράγουμε επεξεργασμένα. Στην οθόνη εμφανίζουμε τον χρόνο επεξεργασίας των ηχητικών σημάτων.
Sound CI mode - διαβάζουμε τα ηχογραφημένα σήματα από την SDRAM, τα επεξεργαζόμαστε σύμφωνα με τον FIR αλγόριθμο χρησιμοποιώντας ειδική εντολή που έχουμε αναπτύξει που υλοποιεί των αλγόριθμο FIR), και τα αναπαράγουμε επεξεργασμένα. Στην οθόνη εμφανίζουμε τον χρόνο επεξεργασίας των ηχητικών σημάτων.

Επίσης για τον έλεγχο ορθής λειτουργίας των αλγορίθμων και των ειδικών κυκλωμάτων και εντολών που αναπτύχθηκαν, όπως και για την ευκολότερη αξιολόγηση της επίδοσης των διαφόρων υλοποιήσεων του αλγορίθμου FIR, υπάρχουν τα ακόλουθα mode λειτουργίας:

Data SW Float mode - Δίνεται σαν διάνυσμα εισόδου στον αλγόριθμο το ακόλουθο (10000,0,0,0,0,0,0,0,0,0) και σαν συντελεστές οι {0.0324577, 0.0362059, 0.0391191, 0.0409677, 0.0416012, 0.0416012, 0.0409677, 0.0391191, 0.0362059, 0.0324577}, τότε σαν έξοδος αναμένονται οι συντελεστές πολλαπλασιασμένοι Χ 10000. Έτσι ελέγχουμε την ορθότητα του αλγορίθμου και μετράμε τον χρόνο εκτέλεσης για 2 διανύσματα των 10 δειγμάτων το κάθε ένα. Στην οθόνη εμφανίζεται το αποτέλεσμα εξόδου και ο χρόνος επεξεργασίας των διανυσμάτων εισόδου.
Data SW INT mode - Δίνεται σαν διάνυσμα εισόδου στον αλγόριθμο το ακόλουθο (1,0,0,0,0,0,0,0,0,0) και σαν συντελεστές οι {0,0,16,64,127,127,64,16,0,0}, τότε σαν έξοδος αναμένονται οι συντελεστές. Έτσι ελέγχουμε την ορθότητα του αλγορίθμου και μετράμε τον χρόνο εκτέλεσης για 2 διανύσματα των 10 δειγμάτων το κάθε ένα. Στην οθόνη εμφανίζεται το αποτέλεσμα εξόδου και ο χρόνος επεξεργασίας των διανυσμάτων εισόδου.
Data HW ACC mode - Δίνεται σαν διάνυσμα εισόδου στον αλγόριθμο το ακόλουθο {2000000000,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0} και σαν συντελεστές αυτοί που έχει ο FIR HW Accelerator, τότε σαν έξοδος αναμένονται οι συντελεστές. Έτσι ελέγχουμε την ορθότητα του αλγορίθμου και μετράμε τον χρόνο εκτέλεσης για 2 διανύσματα των 37 δειγμάτων το κάθε ένα. Στην οθόνη εμφανίζεται το αποτέλεσμα εξόδου και ο χρόνος επεξεργασίας των διανυσμάτων εισόδου.
Data CI mode - Δίνεται σαν διάνυσμα εισόδου στον αλγόριθμο το ακόλουθο (1000,0,0,0,0,0,0,0,0,0) και σαν συντελεστές οι {0,0,16,64,127,127,64,16,0,0}, τότε σαν έξοδος αναμένονται οι συντελεστές πολλαπλασιασμένοι Χ 1000. Έτσι ελέγχουμε την ορθότητα του αλγορίθμου και μετράμε τον χρόνο

εκτέλεσης για 2 διανύσματα των 10 δειγμάτων το κάθε ένα. Στην οθόνη εμφανίζεται το αποτέλεσμα εξόδου και ο χρόνος επεξεργασίας των διανυσμάτων εισόδου.

4.2 **Οι πυρήνες υλικού του Συστήματος**

Το σύστημα αποτελείται από τον επεξεργαστή Nios II (Nios II/s core) καθώς και από έναν αριθμό περιφερειακών συσκευών, εκ των οποίων ορισμένα είναι μέρος μιας βιβλιοθήκης περιφερειακών συσκευών με όνομα UP IP Cores, ενώ κάποια άλλα παρέχονται ως μέρος του SOPC Builder.

4.2.1 **Ο πυρήνας επεξεργαστή** Nios II/s

Το Nios II/s "standard" core [6], [11], [16], [17] έχει σχεδιαστεί με γνώμονα το μικρό μέγεθος του core θυσιάζοντας κάποιο μέρος της απόδοσης, αφού χρησιμοποιεί περίπου 20% λιγότερη λογική συγκρινόμενο με το Nios II/f core που έχει ως αποτέλεσμα την ελάττωση της απόδοσης κατά 30%. Η σχεδίαση του συγκεκριμένου core έχει γίνει με σκοπό την επίτευξη των ακόλουθων στόχων: i) την ύπαρξη μιας ισορροπίας ανάμεσα στο μέγεθος του core και την απόδοσή του, ii) την απομάκρυνση των χαρακτηριστικών εκείνων (σε hardware) που καταναλώνουν μεγάλο αριθμό on-chip πόρων εις βάρος της απόδοσης. Το συγκεκριμένο Nios II core είναι ιδανικό για εφαρμογές που δεν απαιτούν μεγάλη υπολογιστική ισχύ καθώς και για εφαρμογές που διαχειρίζονται μεγάλο πλήθος δεδομένων όπως ένα πλήρες λειτουργικό σύστημα, όπου η απόδοση δεν αποτελεί τον σημαντικότερο ρόλο.

Το core αυτό:

- Διαθέτει κρυφή μνήμη εντολών
- Παρέχει πρόσβαση σε έως 2 GBytes εξωτερικής μνήμης
- Υποστηρίζει tightly-coupled μνήμη για εντολές και δεδομένα (προαιρετικά)
- Χρησιμοποιεί ένα pipeline 5 σταδίων για επίτευξη μέγιστου DMIPS/MHz
- Παρέχει ειδικό hardware πολλαπλασιασμών, διαιρέσεων και ολισθήσεων για βελτίωση της απόδοσης αριθμητικών πράξεων
- Υποστηρίζει την προσθήκη εντολών προσδιοριζόμενων από το χρήστη
- Υποστηρίζει τη μονάδα αποσφαλμάτωσης JTAG (και προαιρετικές βελτιστοποιήσεις του όπως hardware breakpoints και άλλες)

4.2.2 **Η Αριθμητική και Λογική Μονάδα** (ALU)

Το Nios II/s core παρέχει αρκετές hardware ρυθμίσεις για την αριθμητική και λογική μονάδα οι οποίες αποσκοπούν στη βελτίωση της απόδοσης των πράξεων πολλαπλασιασμού και διαίρεσης. Για το σκοπό αυτό, υπάρχουν οι εξής επιλογές:

- DSP blocks – οι πολλαπλασιαστές τοποθετούνται σε DSP block τα οποία είναι διαθέσιμα σε συγκεκριμένα FPGAs της ALTERA
- Ενσωματωμένους πολλαπλασιαστές – οι πολλαπλασιαστές τοποθετούνται σε ειδικούς ενσωματωμένους πολλαπλασιαστές που είναι διαθέσιμοι σε συγκεκριμένα FPGAs της ALTERA
- Λογικά Στοιχεία (LEs) – οι πολλαπλασιαστές τοποθετούνται σε λογικά στοιχεία
- *None* – Δεν περιλαμβάνεται hardware πολλαπλασιασμού και οι πράξεις πολλαπλασιασμού εξομοιώνονται σε software

Επιλογή Υλοποίησης	Λεπτομέρειες Hardware	Κύκλοι Ρολογιού ανά Εντολή	Κύκλοι Καθυστέρησης Αποτελέσματος	Υποστηριζόμενες Εντολές
DSP Block στις οικογένειες Startix, Startix II και Startix III	Η ALU ενσωματώνει 32x32 bits πολ/στές	1	+2	`mul, muli, mulxss, mulxsu, mulxuu`
Ενσωματωμένοι Πολ/στές στις οικογένειες Cyclone, Cyclone II και Cyclone III	Η ALU ενσωματώνει 32x16 bits πολ/στές	5	+2	`mul, muli`
Λογικά Στοιχεία	Η ALU ενσωματώνει 32x4 bits πολ/στές	11	+2	`mul, muli`
None	Εντολές πολ/σμού και διαίρεσης παράγουν μία εξαίρεση	-	-	Καμία
Hardware Διαιρέτες	Η ALU ενσωματώνει ένα multi-cycle κύκλωμα διαίρεσης	4-66	+2	`div, divu`

Πίνακας 4.1 – Επιλογές υλοποίησης εντολών πολλαπλασιασμού / διαίρεσης

Η τιμή της στήλης "κύκλοι ρολογιού ανά εντολή" προσδιορίζει το μέγιστο αριθμό με τον οποίο η ALU μπορεί να εκτελεί εντολές και να παράγει το κάθε αποτέλεσμα. Η τιμή της στήλης "κύκλοι καθυστέρησης αποτελέσματος" καθορίζει πότε το αποτέλεσμα γίνεται διαθέσιμο. Εάν δεν υπάρχουν εντολές που να βασίζονται σε προηγούμενα αποτελέσματα, η τιμή της καθυστέρησης δεν επηρεάζει την απόδοση της ALU, σε αντίθετη περίπτωση, ο επεξεργαστής εισέρχεται σε κατάσταση αναμονής για όσους κύκλους ρολογιού χρειάζεται το αποτέλεσμα προκειμένου να γίνει διαθέσιμο. Η απόδοση των πράξεων ολίσθησης βασίζεται στην επιλογή υλοποίησης των hardware πολλαπλασιαστών. Όταν υπάρχουν hardware πολλαπλασιαστές, η ALU ολοκληρώνει τις πράξεις ολίσθησης / περιστροφής (shift / rotate) σε έναν ή δύο κύκλους ρολογιού. Διαφορετικά η ALU ενσωματώνει ειδικό κύκλωμα ολισθήσεων που πραγματοποιεί ολισθήσεις / περιστροφές με ρυθμό 1-bit-ανά-κύκλο ρολογιού.

4.2.3 Πρόσβαση σε Μνήμη

Το Nios II/s core παρέχει κρυφή μνήμη εντολών (αλλά όχι δεδομένων). Το μέγεθος αυτών των κρυφών μνημών προσδιορίζεται από το χρήστη και κυμαίνεται από 512 bytes έως 64 Kbytes. Υποστηρίζεται επίσης η μέθοδος παράκαμψης κρυφής μνήμης bit-31 bus bypass για πρόσβαση σε I/O που βρίσκονται συνδεδεμένα στο data master port. Στο συγκεκριμένο core το bit-31 έχει πάντα την τιμή μηδέν.

Instruction Master Port και Data Master Port

Το instruction master port είναι ένα pipelined Avalon-MM master port όπως και το data master port . Το instruction master port είναι προαιρετικό και μπορεί να εξαιρεθεί εφόσον το core του επεξεργαστή περιλαμβάνει τουλάχιστον μία tightly-coupled μνήμη που το αντικαθιστά. Η υποστήριξη pipelined Avalon-MM μεταφορών δεδομένων, ελαχιστοποιεί την καθυστέρηση της σύγχρονης μνήμης (synchronous memory). Τα pipelined instruction master port και data master port μπορούν να εξυπηρετήσουν διαδοχικές αιτήσεις ανάγνωσης δεδομένων πριν ολοκληρωθούν οι προηγούμενες.

Κρυφή Μνήμη Εντολών

Η κρυφή μνήμη εντολών έχει τα ακόλουθα χαρακτηριστικά:

- Direct-mapped υλοποίηση
- 32 bytes (8 words) ανά γραμμή
- Το instruction master port διαβάζει μία γραμμή ανά κύκλο ρολογιού από την κρυφή μνήμη

Table 5–8. Instruction Byte Address Fields

31	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5	4	3	2	1	0
tag												line											offset			0	0

Το εύρος του πεδίου “tag” εξαρτάται από το μέγεθος της κρυφής μνήμης καθώς και από το physical address size, ενώ το εύρος του πεδίου “line” εξαρτάται μόνο από το μέγεθος της κρυφής μνήμης. Το πεδίο offset είναι πάντα 3 bits. Η κρυφή μνήμη εντολών , όπως έχει αναφερθεί, είναι προαιρετική. Ωστόσο, η εξαίρεσή της από το Nios II/s core απαιτεί τη χρήση τουλάχιστον μίας tightly-coupled μνήμης εντολών.

Tightly-Coupled **Μνήμη**

Το Nios II/s core προσφέρει προαιρετικά τη χρήση tightly-coupled μνήμης και για δεδομένα και εντολές, ενώ μπορεί να περιλαμβάνει μέχρι και τέσσερις τέτοιες μνήμες. Όταν έχει ενεργοποιηθεί η διασύνδεση με tightly-coupled μνήμη, το core μπορεί να ενσωματώσει ένα επιπλέον master port διασύνδεσης μνήμης. Το core αποκωδικοποιεί τις διευθύνσεις εσωτερικά, ώστε να προσδιορίσει αν τα ζητούμενα δεδομένα (ή εντολές) υπάρχουν μέσα σε κάποια tightly-coupled μνήμη. Αν η διεύθυνση των δεδομένων αυτών (ή των εντολών) υπάρχει στην tightly-coupled μνήμη το core ανακαλεί τα δεδομένα (ή τις εντολές) μέσω της tightly-coupled memory διασύνδεσης. Το software προσπελαύνει την tightly-coupled μνήμη με της συνήθης εντολές φόρτωσης και αποθήκευσης *ldw, ldwio, stw* και *stwio*. Η πρόσβαση στην tightly-coupled μνήμη παρακάμπτει την κρυφή μνήμη.

Απόδοση Εκτέλεσης Εντολών

Το Nios II/s core παρέχει ένα Pipeline 5-σταδίων, όπως φαίνεται και στον πίνακα, για την βελτιστοποίηση του χρόνου εκτέλεσης εντολών.

Γράμμα Σταδίου	Όνομα Σταδίου
F	Fetch
D	Decode
E	Execute
M	Memory
W	Writeback

Μέχρι και μία εντολή εκτελείται ή τερματίζεται ανά κύκλο ρολογιού. Το pipeline εισέρχεται σε κατάσταση αναμονής (stalls) στις εξής περιπτώσεις :

- Όταν εκτελείται μια multi-cycle εντολή
- Όταν προσπελαύνεται ένα Avalon-MM master port για ανάγνωση/εγγραφή
- Όταν υπάρχει εξάρτηση εντολών υψηλής καθυστέρησης (multiply, shift και άλλες) από δεδομένα

Αν το M-στάδιο, και μόνο αυτό, σταματήσει τη λειτουργία του, τότε δεν εισέρχεται καμία νέα τιμή ούτε σε αυτό ούτε σε προηγούμενα στάδια ακόμα και αν το pipeline είναι άδειο. Όλες οι εντολές καταναλώνουν έναν ή περισσότερους κύκλους ρολογιού για να εκτελεστούν. Κάποιες εντολές έχουν ιδιαιτερότητες που σχετίζονται με το χρόνο εκτέλεσης τους, όπως εντολές που απαιτούν την χρήση Avalon-MM μεταφορών δεδομένων. Οι εντολές αυτές πρέπει να “περιμένουν” μέχρι την ολοκλήρωση όλων των προηγούμενων Avalon-MM μεταφορών, ώστε να εκτελεστούν. Ο πίνακας 4.2 παρουσιάζει περιληπτικά την απόδοση εκτέλεσης των εντολών του Nios II/s core.

Εντολή	Κύκλοι Ρολογιού	Penalties
Συνήθεις εντολές της ALU (add κλ)	1	
Συνδυαστικές εντολές προσδιοριζόμενες από τον χρήστη	1	
Multi cycle εντολές προσδιοριζόμενες από τον χρήστη	1	Καθυστερημένο αποτέλεσμα
Εντολές διακλάδωσης (correctly predicted, taken)	2	
Εντολές διακλάδωσης (correctly predicted, not taken)	1	
Εντολές διακλάδωσης (mis predicted)	4	Άδειασμα του pipeline
trap, break, eret, bret, flushp, wrctl και μη-υλοποιημένες εντολές	4	Άδειασμα του pipeline
call, jmpi, jmp, ret, callr	4	

rdctl	1	Καθυστερημένο αποτέλεσμα
load, store	>1	Καθυστερημένο αποτέλεσμα
flushi, initi	4	
Πολλαπλασιασμός	*(1)*	Καθυστερημένο αποτέλεσμα
Διαίρεση	*(1)*	Καθυστερημένο αποτέλεσμα
Ολίσθηση/Περιστροφή (με ενσωματωμένους πολλαπλασιαστές)	3	Καθυστερημένο αποτέλεσμα
Ολίσθηση/Περιστροφή (με λογικά στοιχεία - LEs)	4	Καθυστερημένο αποτέλεσμα
Ολίσθηση/Περιστροφή (χωρίς ειδικό hardware)	1 32	Καθυστερημένο αποτέλεσμα
Υπόλοιπες εντολές	1	Καθυστερημένο αποτέλεσμα

(1) Εξαρτάται από την επιλογή του τρόπου υλοποίησης των πολ/τών και των διαιρέσεων

Πίνακας 4.2 - Απόδοση εκτέλεσης των εντολών του Nios II/s core

Διαχείριση Εξαιρέσεων

Το Nios II/s core υποστηρίζει τους παρακάτω τύπους εξαιρέσεων όπως αυτοί έχουν ήδη αναλυθεί:

- Hardware διακοπές
- Software traps
- Μη-υλοποιημένες εντολές

Μονάδα αποσφαλμάτωσης JTAG

Το Nios II/s Core υποστηρίζει τη μονάδα αποσφαλμάτωσης JTAG, ώστε να παρέχει μια JTAG διασύνδεση με το Quartus II και το Nios II IDE. Υποστηρίζει και επιπλέον προαιρετικά χαρακτηριστικά όπως ανίχνευση δεδομένων, μεταφορά τους εκτός επεξεργαστή και αποθήκευσή τους σε εξωτερικό debug probe.

Μη Υποστηριζόμενα Χαρακτηριστικά

Το Nios II/s core δεν εκτελεί εντολές με μη ορισμένο opcode. Αν ο επεξεργαστής συναντήσει μία τέτοια εντολή το αποτέλεσμα είναι αόριστο.

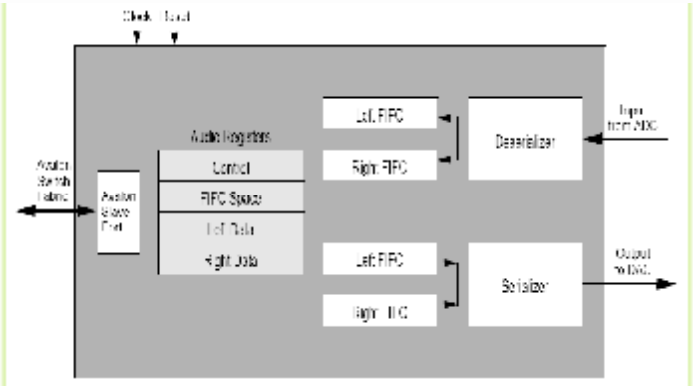
4.3 **Οι πυρήνες περιφερειακών** «UP IP Cores»

Τα UP IP Cores [10] είναι έτοιμα προς χρήση υποκυκλώματα για τον SOPC Builder και περιλαμβάνουν σχεδόν όλες τις περιφερειακές συσκευές του DE1 Board της ALTERA. Τα cores που χρησιμοποιήθηκαν είναι:

- Audio Core for Altera DE1 Board
- Audio/Video Configuration Core for DE1 Boards
- DE Boards External Interface for Altera DE1 Boards
- VGA Core for Altera DE1 Boards
- PIO for Altera DE1 Boards
- SDRAM Controller for Altera DE1Boards
- SRAM Controller
- PS/2 Port
- SYSTEM ID Core
- Performance Counter Core
- RS232 UART Core
- Pixel Buffer DMA Controller
- RGB Resampler
- VIDEO SCALER
- ALPHA BLENDER
- Character Buffer for VGA Display
- DUAL – CLOCK FIFO

4.3.1 **Ο πυρήνας υλικού** «Audio Core»

Το Audio Core [18], [19] αλληλεπιδρά με τον audio codec (EnCoder/DeCoder) της πλακέτας DE1, ο οποίος είναι ο WM8731 της Wolfson Microelectronics παρέχοντας διασύνδεση για είσοδο και έξοδο ήχου (audio). Η αρχικοποίηση του Audio Core επιτυγχάνεται μέσα από τον SOPC Builder (Configuration Wizard). Το core υποστηρίζει ταυτόχρονη είσοδο και έξοδο ήχου. Επίσης για την εξασφάλιση του συγχρονισμού μεταξύ του δεξιού και του αριστερού καναλιού ήχου, δεν αναπαράγεται ήχος έως ότου ληφθούν τα δεδομένα και από τα δύο κανάλια. Αν απαιτείται η αναπαραγωγή ενός μόνο καναλιού, τότε το άλλο κανάλι θα πρέπει να περιέχει μηδενικά ηχητικά δεδομένα (zeros). Στην εικόνα 4.1 παρουσιάζεται το block διάγραμμα του Audio Core.



Εικόνα 4.1 - Το block διάγραμμα του Audio Core

Το Audio Core περιλαμβάνει 4 FIFO (First In First Out) μνήμες, οι οποίες όπως φαίνεται και από το σχηματικό διάγραμμα, χρησιμεύουν στην προσωρινή αποθήκευση των ηχητικών δεδομένων των δύο καναλιών εισόδου και εξόδου. Κάθε FIFO μνήμη μπορεί να αποθηκεύσει έως 255 δεδομένα των 32-bit. Το Audio Core απαιτεί επίσης και κάποια σήματα χρονισμού που εξαρτώνται από το ρυθμό / συχνότητα δειγματοληψίας (sample rate/frequency) που έχει οριστεί. Τα σήματα αυτά τα παρέχει ένα άλλο core της ALTERA, το DE1 Board External Interface, που θα περιγραφεί στη συνέχεια. Περιλαμβάνονται επίσης ένας Serializer και ένας Deserializer που στέλνουν τα δεδομένα σειριακά από και προς τον ADC και DAC αντίστοιχα. Τέλος εμπεριέχεται και η κατάλληλη λογική σύνδεσης του core με το υπόλοιπο σύστημα μέσω του Avalon Slave Port. Οι συναρτήσεις – οδηγοί του core ελέγχουν το core και επικοινωνούν μαζί του μέσω τεσσάρων καταχωρητών των 32-bit. Με την εγγραφή στους καταχωρητές αυτούς και με την ανάγνωση τους επιτυγχάνεται η λήψη ηχητικών δεδομένων από τον Αναλογικό-Ψηφιακό μετατροπέα (Analog-to-Digital Converter - ADC) και η αποστολή τους προς τον Ψηφιακό-Αναλογικό μετατροπέα (Digital-to-Analog Converter - DAC). Ο χάρτης των καταχωρητών (register map) του core φαίνεται στην εικόνα 4.2

Offset in bytes	Register Name	R/W	Bit Description								
			31…24	23…16	15…10	9	8	7…4	3	2	1	0
0	control	RW	(1)			WI	RI	(1)	CW	CR	WE	RE
4	fifospace	R	WS LC	WS RC		RA LC			RA RC			
8	leftdata	RW (2)	Left Data									
12	rightdata	RW (2)	Right Data									

(1) Δεσμευμένα. Οι τιμές ανάγνωσης τους είναι απροσδιόριστες
(2) Μόνο ανάγνωση και εγγραφή εξερχόμενων ηχητικών δεδομένων

Εικόνα 4.2 - Ο χάρτης των καταχωρητών (register map) του Audio core

Ο καταχωρητής *control* (καταχωρητής ελέγχου) είναι αυτός που ουσιαστικά ελέγχει την όλη λειτουργία του core. Ο πίνακας 4.3 περιγράφει τον ρόλο των επιμέρους bit του καταχωρητή ελέγχου.

Αριθμός Bit	Όνομα Bit	Ανάγνωση / Εγγραφή	Περιγραφή
0	RE	A/E	Το bit αυτό ενεργοποιεί τις διακοπές ανάγνωσης (read interrupts). Αν το RE έχει την τιμή "1" και οι δύο FIFO μνήμες ανάγνωσης (left read FIFO, right read FIFO) περιέχουν δεδομένα, τότε το Audio Core παράγει μία αίτηση διακοπής (IRQ).
1	WE	A/E	Το bit αυτό ενεργοποιεί τις διακοπές εγγραφής (write interrupts). Αν το WE έχει την τιμή "1" και οι δύο FIFO μνήμες εγγραφής (left write FIFO, right write FIFO) περιέχουν δεδομένα, τότε το Audio Core παράγει μία αίτηση διακοπής (IRQ).
2	CR	A/E	Το bit αυτό όταν πάρει την τιμή "1", "καθαρίζει" (γράφοντας μηδενικά) τις δύο FIFO μνήμες ανάγνωσης και παραμένει στην τιμή αυτή έως ότου τεθεί στο "0".
3	CW	A/E	Το bit αυτό όταν πάρει την τιμή "1", "καθαρίζει" (γράφοντας μηδενικά) τις δύο FIFO μνήμες εγγραφής και παραμένει στην τιμή αυτή έως ότου τεθεί στο "0".
8	RI	A	Υποδεικνύει ότι εκκρεμεί μία διακοπή ανάγνωσης.
9	WI	A	Υποδεικνύει ότι εκκρεμεί μία διακοπή εγγραφής.

Πίνακας 4.3 – Ο ρόλος των επιμέρους bit του καταχωρητή ελέγχου του Audio Core

Ο καταχωρητής *fifospace* διαχειρίζεται τις FIFO μνήμες ανάγνωσης και εγγραφής. Ειδικότερα, τα πεδία *WSLC* (b_{31}-b_{24}) υποδεικνύουν των αριθμό των εξερχόμενων δεδομένων που είναι διαθέσιμα στις δύο FIFO μνήμες εγγραφής (left write FIFO,right write FIFO).
Αντίστοιχα, τα πεδία *WSRC* (b_{23}-b_{16}) υποδεικνύουν των αριθμό των εξερχόμενων δεδομένων που είναι διαθέσιμα στις δύο FIFO μνήμες ανάγνωσης (left read FIFO, right read FIFO). Ο καταχωρητής *leftdata* είναι καταχωρητής ανάγνωσης/εγγραφής ηχητικών δεδομένων εισόδου/εξόδου του αριστερού καναλιού. Ομοίως, ο καταχωρητής *rightdata* είναι καταχωρητής ανάγνωσης / εγγραφής ηχητικών δεδομένων εισόδου/εξόδου του δεξιού καναλιού. Το Audio Core, όπως έχει αναφερθεί, μπορεί να παράγει διακοπές (interrupts) εφόσον έχει ενεργοποιηθεί η δυνατότητα αυτή. Αναλυτικότερα, προκαλεί μία διακοπή ανάγνωσης όταν η FIFO μνήμη ανάγνωσης είναι γεμάτη κατά 75% ή περισσότερο και μία διακοπή εγγραφής όταν η FIFO μνήμη εγγραφής είναι γεμάτη κατά 25% ή λιγότερο. Η ALTERA παρέχει έτοιμες C συναρτήσεις οι οποίες είναι προσβάσιμες μέσω του Hardware Abstraction Layer (HAL) και υλοποιούν βασικές λειτουργίες του Audio Core. Για τη σωστή λειτουργία των παρεχόμενων συναρτήσεων θα πρέπει στο κυρίως πρόγραμμα/εφαρμογή να συμπεριληφθεί το αρχείο *"altera_up_avalon_audio.h"*. Οι συναρτήσεις αυτές είναι οι εξής :

- alt_up_audio_open_dev

Η συνάρτηση αυτή ενεργοποιεί το Audio Core

- alt_up_audio_enable_read_interrupt

Η συνάρτηση αυτή ενεργοποιεί τις διακοπές ανάγνωσης του Audio Core

- alt_up_audio_disable_read_interrupt

Η συνάρτηση αυτή απενεργοποιεί τις διακοπές ανάγνωσης του Audio Core

- alt_up_audio_reset_audio_core

Μηδενίζει τα δεδομένα των FIFO μνημών εισόδου/εξόδου και για τα δύο κανάλια

- alt_up_audio_read_fifo

Η συνάρτηση αυτή διαβάζει τα περιεχόμενα των δύο FIFO μνημών εισόδου.

- alt_up_audio_write_fifo

Η συνάρτηση αυτή γράφει τα κατάλληλα δεδομένα στις δύο FIFO μνήμες εξόδου.

4.3.2 **Ο πυρήνας υλικού** «Audio/Video Configuration Core»

Το Audio/Video Configuration Core [20] αλληλεπιδρά με τον Audio Codec και παρέχει έναν εύκολο τρόπο για την παραμετροποίηση / αρχικοποίηση των παραπάνω στοιχείων.

Το core αυτό περιέχει τέσσερις καταχωρητές των 32-bit, στους οποίους αποθηκεύονται οι ρυθμίσεις των συσκευών και ένα serializer που στέλνει τις ρυθμίσεις αυτές μέσω του πρωτοκόλλου επικοινωνίας I^2C και διαύλων τριών καλωδίων στα audio και video περιφερειακά. Ο χάρτης των καταχωρητών (register map) του core φαίνεται στην εικόνα 4.3

Offset in bytes	Register Name	R/W/C	31...8	7...4	3	2	1	0
0	control	W	(1)		RW	S	P	R
4	status	R/C	(1)		AIE	AI	TD	ACK
8	address	W	(1)	Addr				
12	data	R/W	(1)	Data				

Εικόνα 4.3 - Ο χάρτης των καταχωρητών (register map) του Audio/Video Configuration Core

Ο καταχωρητής *control* (καταχωρητής ελέγχου) είναι αυτός που ελέγχει τη λειτουργία του core. Ο πίνακας 4.4 περιγράφει το ρόλο των επιμέρους bit του καταχωρητή ελέγχου.

Αριθμός Bit	Όνομα Bit	Ανάγνωση / Εγγραφή	Περιγραφή
0	R	E	Εκτελεί επαναφορά (reset) του core και το επαναρχικοποιεί αν έχει επιλεγεί η ρύθμιση *auto-initialize*. Η audio και video συσκευή του DE2 επαναφέρονται επίσης, όταν τεθεί στο λογικό "1" .
1	P	E	Στέλνει ένα bit τερματισμού στον αντίστοιχο δίαυλο.
2	S	E	Στέλνει ένα bit εκκίνησης και την διεύθυνση της συσκευής στον αντίστοιχο δίαυλο.
3	RW	E	Χρησιμοποιείται για να υποδείξει αν τα εισερχόμενα δεδομένα είναι για ανάγνωση ή εγγραφή

Πίνακας 4.4 - Ο ρόλος των επιμέρους bit του καταχωρητή ελέγχου του Audio/Video Configuration Core

Ο καταχωρητής *status* (καταχωρητής κατάστασης) είναι αυτός που ελέγχει την κατάσταση αποστολής των ρυθμίσεων. Ο πίνακας 4.5 περιγράφει το ρόλο των επιμέρους bit του καταχωρητή κατάστασης.

Αριθμός Bit	Όνομα Bit	Ανάγνωση / Εγγραφή	Περιγραφή
0	ACK	A/E	Acknowledge – Η τιμή "1" υποδηλώνει ότι συνέβη κάποιο σφάλμα κατά την μεταφορά των δεδομένων.

1	TD	A	Transferring Data – Υποδηλώνει ότι τα bits εκκίνησης και τερματισμού, διεύθυνση ή δεδομένα συνεχίζουν να αποστέλλονται.
2	AI	A	Auto-Initialize – Υποδηλώνει ότι η συσκευή βρίσκεται σε κατάσταση αρχικοποίησης.
AIE	RW	A/E	Auto-Initialize Error – Υποδηλώνει ότι συνέβη κάποιο σφάλμα κατά τη διαδικασία αρχικοποίησης της συσκευής.

Πίνακας 4.5 - Ο ρόλος των επιμέρους bit του καταχωρητή κατάστασης του Audio/Video Configuration Core

Ο καταχωρητής address (καταχωρητής διευθύνσεων) χρησιμοποιείται για την αποστολή της διεύθυνσης του καταχωρητή ελέγχου της συσκευής προς αρχικοποίηση/παραμετροποίηση. Για τον audio codec του DE1, ο καταχωρητής αυτός περιλαμβάνει επτά bit για την διεύθυνση και εννέα bit για τα δεδομένα. Επειδή όμως κάθε φορά αποστέλλονται οκτώ bits (ένα byte), το ένατο bit δεδομένων ενώνεται μαζί με τα επτά bit της διεύθυνσης σχηματίζοντας έτσι δύο πακέτα των 8-bit. Έτσι, τα επτά bit ολισθαίνουν κατά μία θέση αριστερά και το ένατο bit δεδομένων τοποθετείται στη θέση 0 του καταχωρητή δεδομένων, όπως φαίνεται στην εικόνα 4.4

Offset in bytes	Register name	R/W/C	31...8	7...1	0
8	address	W	Reserved	Addr	Data (9)

Εικόνα 4.4 - Ο ρόλος των επιμέρους bit του καταχωρητή δεδομένων του Audio/Video Configuration Core

Ο καταχωρητής data (καταχωρητής δεδομένων) χρησιμοποιείται για την αποστολή των δεδομένων από και προς τον καταχωρητή ελέγχου της συσκευής προς αρχικοποίηση/παραμετροποίηση. Ο audio codec του DE1 λαμβάνει μόνο δεδομένα, οπότε δεν πρέπει να γίνεται ανάγνωση του καταχωρητή αυτού.
Οι αλλαγές των ρυθμίσεων του Audio/VideoConfiguration Core επιτυγχάνονται μέσα από τον SOPCBuilder (Configuration Wizard). Το core προσφέρει πληθώρα ρυθμίσεων όπως:

- Γενικές Ρυθμίσεις
- Use Altera's UP IP Audio Controller– Επιλογή ώστε ο audio codec να συνεργαστεί με το UP IP Audio Core
- Power Down Audio Chip – Απενεργοποιεί εντελώς τον audio codec.
- Επιλογή της πηγής ήχου
- Line In to ADC – Επιλέγει το Line In ως είσοδο προς τον ADC.
- Mic In to ADC – Επιλέγει το Mic In ως είσοδο προς τον ADC.
- Mute Mic – Θέτει την ένταση του μικροφώνου στο μηδέν.
- DAC Output – Επιτρέπει στα δεδομένα του DAC να περάσουν στην έξοδο.
- Line In Bypass – Στέλνει το σήμα από την είσοδο Line In κατευθείαν στην έξοδο παρακάμπτοντας τον ADC και τον DAC.
- Mic Bypass – Στέλνει το σήμα από την είσοδο Mic In κατευθείαν στην έξοδο παρακάμπτοντας τον ADC και τον DAC.
- Επιλογές ψηφιακού ελέγχου
- Επιλογές μορφής του ηχητικού σήματος
- Data Format – επιτρέπει την επιλογή της μορφής των δεδομένων (DSP Mode, I2S Format, Left Justified ή Right Justified). Αν έχει επιλεγεί η χρήση του UP IP Audio Core τότε η επιλογή Left Justified επιλέγεται αυτόματα.
- Bit Length – Επιτρέπει την επιλογή του αριθμού των bits για κάθε κανάλι. Επιτρεπτές τιμές είναι αυτές των 16, 20 και 24-bit για όλες τις μορφές δεδομένων, ενώ η τιμή των 32-bit είναι διαθέσιμη μόνο για τις I2S και Left Justified μορφές δεδομένων. Και ο ADC και ο DAC δουλεύουν με δεδομένα των 24-bit. Έτσι αν επιλεγεί διαφορετική ρύθμιση εφαρμόζονται τεχνικές zero-padding ή stripping των LSBits.

- Επιλογές Δειγματοληψίας – Ο audio codec μπορεί να δειγματοληπτεί είτε σε USB Mode είτε σε Normal Mode. Σε USB Mode η συχνότητα του ρολογιού που συγχρονίζει τον Audio codec παραμένει σταθερή και ίση με 12 MHz, ενώ σε Normal Mode η συχνότητα ρολογιού μπορεί να διαφέρει. Επίσης, σε κάθε mode μπορεί να οριστεί ο ρυθμός δειγματοληψίας (sample rate, Base-Over Sampling Rate). Στον πίνακα 4.6 παρουσιάζονται οι διαθέσιμες επιλογές όσον αφορά το ρυθμό/συχνότητα δειγματοληψίας σε Normal Mode.

SAMPLING RATE		MCLK FREQUENCY	SAMPLE RATE REGISTER SETTINGS					DIGITAL FILTER TYPE
ADC	DAC							
kHz	kHz	MHz	BOSR	SR3	SR2	SR1	SR0	
48	48	12.288	0 (256fs)	0	0	0	0	1
		18.432	1 (384fs)	0	0	0	0	
48	8	12.288	0 (256fs)	0	0	0	1	1
		18.432	1 (384fs)	0	0	0	1	
8	48	12.288	0 (256fs)	0	0	1	0	1
		18.432	1 (384fs)	0	0	1	0	
8	8	12.288	0 (256fs)	0	0	1	1	1
		18.432	1 (384fs)	0	0	1	1	
32	32	12.288	0 (256fs)	0	1	1	0	1
		18.432	1 (384fs)	0	1	1	0	
96	96	12.288	0 (128fs)	0	1	1	1	2
		18.432	1 (192fs)	0	1	1	1	
44.1	44.1	11.2896	0 (256fs)	1	0	0	0	1
		16.9344	1 (384fs)	1	0	0	0	
44.1	8	11.2896	0 (256fs)	1	0	0	1	1
	(Note 1)	16.9344	1 (384fs)	1	0	0	1	
8	44.1	11.2896	0 (256fs)	1	0	1	0	1
(Note 1)		16.9344	1 (384fs)	1	0	1	0	
8	8	11.2896	0 (256fs)	1	0	1	1	1
(Note 1)	(Note 1)	16.9344	1 (384fs)	1	0	1	1	
88.2	88.2	11.2896	0 (128fs)	1	1	1	1	2
		16.9344	1 (192fs)	1	1	1	1	

Πίνακας 4.6 - Οι διαθέσιμες επιλογές όσον αφορά το ρυθμό/συχνότητα δειγματοληψίας σε Normal Mode

Η ALTERA παρέχει έτοιμες C συναρτήσεις οι οποίες είναι προσβάσιμες μέσω του Hardware Abstraction Layer (HAL) και με αυτές επιτυγχάνεται ο προγραμματισμός των ρυθμίσεων των συσκευών αυτών σε software. Για τη σωστή λειτουργία των παρεχόμενων συναρτήσεων θα πρέπει στο κυρίως πρόγραμμα/εφαρμογή να συμπεριληφθεί το αρχείο *"altera_up_avalon_audio_video_config.h"*. Οι συναρτήσεις αυτές είναι οι εξής :

- alt_up_av_config_open_dev

Η συνάρτηση αυτή ενεργοποιεί το Audio/Video Configuration Core

- alt_up_audio_av_config_send_audio_cfg

Η συνάρτηση αυτή στέλνει τις αντίστοιχες ρυθμίσεις στον καταχωρητή ελέγχου του audio codec του DE1

- alt_up_audio_av_config_send_video_cfg

Η συνάρτηση αυτή στέλνει τις αντίστοιχες ρυθμίσεις στον καταχωρητή ελέγχου του video controller του DE1

4.3.3 **Ο πυρήνας υλικού** «DE Boards External Interface Core»

Το DE1 Board External Interface Core [22] παράγει τα κατάλληλα σήματα χρονισμού που είναι αναγκαία για τη λειτουργία του Audio Core, του VGA Core και του SDRAM Controller. Αναλυτικότερα, για τη λειτουργία του Audio Core, το DE Board External Interface Core παράγει ένα σήμα ρολογιού με όνομα *AUD_XCK*. Η συχνότητα του σήματος αυτού μπορεί να πάρει διάφορες τιμές (πέντε στο σύνολο) ανάλογα με τις ρυθμίσεις που έχουν γίνει για το ρυθμό δειγματοληψίας του audiocodec.Το σήμα αυτό συνδέεται στον αντίστοιχο ακροδέκτη (pin) του Cyclone II.

Το VGA Core, που θα περιγραφεί στην συνέχεια, απαιτεί την ύπαρξη ενός ρολογιού συχνότητας 25MHz με όνομα *VGA_CLK*, το οποίο θα συνδεθεί στον VGA DAC. Επιπλέον χρειάζεται και ένα ρολόι συχνότητας 50MHz για τα Avalon slave ports, το οποίο όμως παρέχεται από το ρολόι του συστήματος. Το σήμα *VGA_CLK* συνδέεται στον αντίστοιχο ακροδέκτη (pin) του Cyclone II.

Το Board External Interface Core παράγει επίσης και ένα σήμα με όνομα *DRAM_CLK* με 3ns phase shift από το ρολόι του συστήματος (συχνότητας 50MHz) για να εξασφαλιστεί ο σωστός χρονισμός του SDRAM chip του DE1.Το σήμα αυτό συνδέεται στον αντίστοιχο ακροδέκτη (pin) του Cyclone II. Το συγκεκριμένο core έχει ένα καταχωρητή ελέγχου (ανάγνωσης - μόνο) που φαίνεται στην εικόνα 4.5 και περιγράφεται στον πίνακα 4.6.

Offset in bytes	Register Name	R/W	Bit Description			
			31...3	2	1	0
0	control	R	*(1)*	ACL *(2)*	SCL *(2)*	VCL *(2)*

Εικόνα 4.5 – Ο καταχωρητής ελέγχου του DE Boards External Interface Core

Αριθμός Bit	Όνομα Bit	Ανάγνωση / Εγγραφή	Περιγραφή
0	VCL	A	VGA Clock Lock – Παίρνει την τιμή "1" όταν το VGA clock έχει σταθεροποιηθεί
1	SCL	A	SDRAM Clock Lock – Παίρνει την τιμή "1" όταν το SDRAM clock έχει σταθεροποιηθεί
2	ACL	A	Audio Clock Lock – Παίρνει την τιμή "1" όταν το Audio clock έχει σταθεροποιηθεί

Πίνακας 4.6 – Τα bits του καταχωρητή ελέγχου του DE Boards External Interface Core

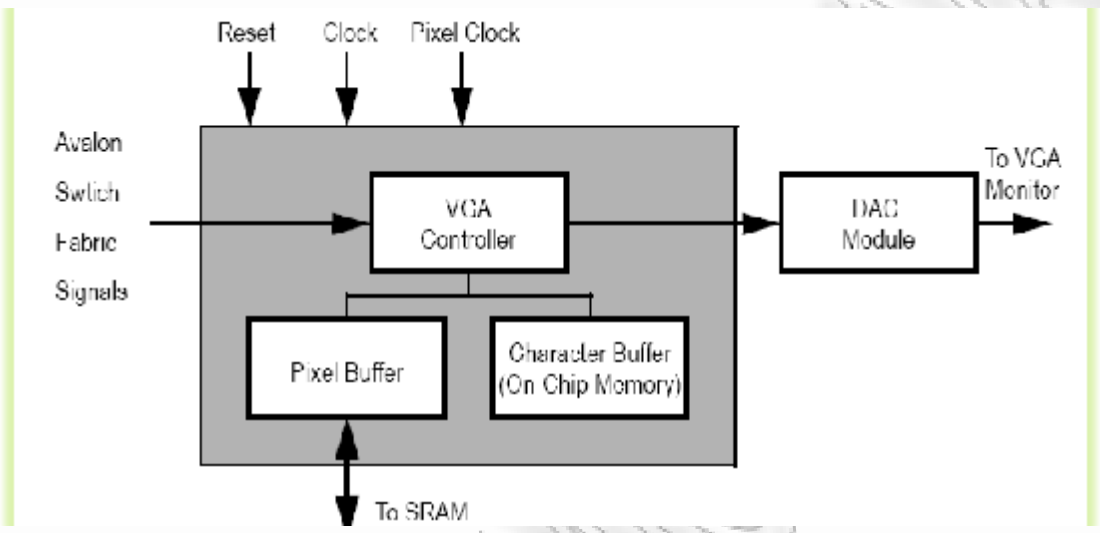
Η παραμετροποίηση του Core γίνεται στο περιβάλλον του SOPC Builder και προσφέρονται οι εξής επιλογές:

- Development and Education Board – Επιλογή της αναπτυξιακής κάρτας (DE2, DE1)
- Required Clocks– Επιλογή των υποκυκλωμάτων (Audio, VGA, SDRAM) που υπάρχουν στο σύστημα και απαιτούν τα αντίστοιχα σήματα ρολογιού που περιγράφθηκαν.
- Audio Clock Settings – Επιλογή της συχνότητας του ρολογιού του audio codec.

4.3.4 **Ο πυρήνας υλικού** «VGA Core»

Το VGACore [29] παράγει τα σήματα που απαιτούνται για απεικόνιση σε VGA οθόνη συμπεριλαμβανομένων των οριζόντιων και κάθετων σημάτων χρονισμού. Για την παραγωγή των σημάτων αυτών απαιτείται ένα ρολόι συχνότητας 25 MHz, το οποίο αναφέρεται ως εσωτερικό ρολόι VGA και παρέχεται από το DE Boards External Interface Core. Τα slave ports του VGA Core, που θα αναλυθούν στη συνέχεια, λειτουργούν σε συχνότητα 50 MHz. Το VGA Core παράγει μία ανάλυση οθόνης 640x480 εικονοστοιχείων (pixels) με ρυθμό ανανέωσης 60 πλαισίων ανά δευτερόλεπτο (60 frames per second). Υποστηρίζονται επίσης μικρότερες αναλύσεις για την επίτευξη των οποίων το VGA Core αντιγράφει την πληροφορία των pixels στα γειτονικά ανάλογα με την επιθυμητή ανάλυση.

Στην εικόνα 4.6 παρουσιάζεται το σχηματικό διάγραμμα του VGA Core. Εκτός από την εμφάνιση pixel στην οθόνη (pixel mode), υποστηρίζεται επιπλέον η εμφάνιση χαρακτήρων (character mode) καθώς και μία μίξη των δύο (character overlay mode) στην οποία οι χαρακτήρες σχηματίζονται πάνω από τα pixel της εικόνας.



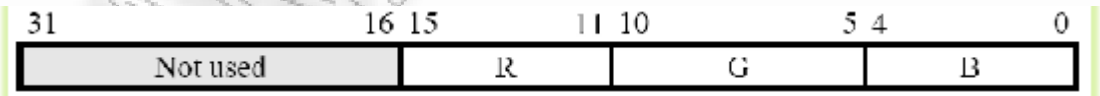
Εικόνα 4.6 - Το σχηματικό διάγραμμα του VGA Core

Τα pixel mode και character mode έχουν τα δικά τους Avalon slave interfaces, *Avalon_pixel_slave* και *Avalon_character_slave.* Αυτά, παρουσιάζονται ως συνεχόμενα διαστήματα μνήμης τόσο μεγάλα όσο χρειάζεται για την αποθήκευση των απαραίτητων δεδομένων. Δηλαδή, συμπεριφέρονται ως απλές διασυνδέσεις μνήμης (memory interfaces), εκτός της περίπτωσης όπου η λειτουργία back buffering, που θα παρουσιαστεί στη συνέχεια, είναι ενεργοποιημένη. Δεν υπάρχουν memory map-mapped καταχωρητές ελέγχου σε κανένα από τα Avalon interfaces.

Pixel Mode

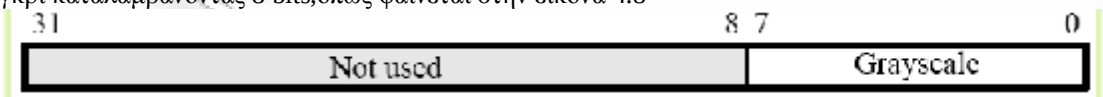
Το VGA Core χρησιμοποιεί το SRAM chip του DE1 board ως pixel buffer. Το mode αυτό είναι κατάλληλο για εμφάνιση μίας εικόνας στη VGA οθόνη. Επίσης μπορεί να οριστεί το χρώμα κάθε pixel σύμφωνα με το RGB (Red Green Blue) μοντέλο χρωμάτων, ακολουθώντας τις εξής ρυθμίσεις:

- 16-bit color mode – σύμφωνα με το οποίο το κόκκινο και μπλε χρώμα απεικονίζονται με πέντε bits ενώ το πράσινο με έξι, όπως φαίνεται στην εικόνα 4.7.



Εικόνα 4.7 – 16 bit color mode

- 8-bit color mode – σύμφωνα με το οποίο τα pixels εμφανίζονται σε αποχρώσεις του γκρι καταλαμβάνοντας 8 bits,όπως φαίνεται στην εικόνα 4.8



Εικόνα 4.8 - 8 bit color mode

Όπως έχει αναφερθεί, το VGA Core υποστηρίζει ανάλυση 640x480=307.200 pixels. Ωστόσο λόγω περιορισμού μεγέθους του pixel buffer, μία χαμηλότερη ανάλυση ίσως είναι αναγκαία. Για την υποστήριξη χαμηλότερων αναλύσεων, το VGACore χρησιμοποιεί τα ***mega-pixels*** όπου κάθε mega-pixel είναι μία ομάδα από pixels που έχουν το ίδιο χρώμα, ελαττώνοντας έτσι την ανάλυση. Οι ακόλουθες αναλύσεις είναι διαθέσιμες:

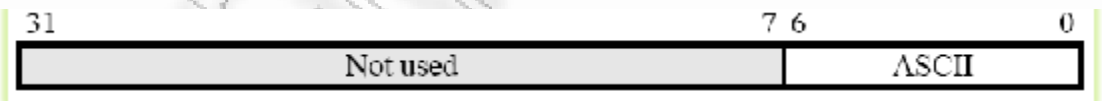
- 640x480 – κάθε mega-pixel είναι ένα απλό pixel
- 320x240 – κάθε mega-pixel είναι μία ομάδα pixels 2x2
- 160x120 – κάθε mega-pixel είναι μία ομάδα pixels 4x4
- 80x60 – κάθε mega-pixel είναι μία ομάδα pixels 8x8
- 40x30 – κάθε mega-pixel είναι μία ομάδα pixels 16x16

Το back buffering είναι μία τεχνική που χρησιμοποιεί δύο image buffers, όπου ο ένας από τους δύο είναι κάθε φορά ενεργός και εμφανίζεται στη VGA οθόνη, ενώ στον άλλο μη ορατό buffer σχηματίζεται κάποια εικόνα. Όταν η εικόνα αυτή ολοκληρωθεί, ο ρόλος των δύο buffers εναλλάσσεται. Το bit b_{31} των δεδομένων χρησιμοποιείται για την αλλαγή του image buffer μέσω ειδικής C συνάρτησης. Επίσης πρέπει να σημειωθεί ότι η αλλαγή του back buffer καταναλώνει κάποιο χρόνο λόγω του γεγονότος ότι το VGA Core πρέπει να περιμένει να ολοκληρωθεί η προς απεικόνιση εικόνα, διαδικασία που μπορεί να πάρει έως και το 1/60 του δευτερολέπτου. Αν το LSB (least significant bit) της θέσης μνήμης του *Avalon pixel slave* έχει την τιμή "1" τότε οι image buffers βρίσκονται στη διαδικασία εναλλαγής και δεν μπορούν να διαβαστούν ή να εγγραφούν νέες τιμές των pixels.

Character Mode

Όταν το VGA Core λειτουργεί σε character mode, η μονάδα απεικόνισης είναι ένας χαρακτήρας. Μία συσκευή στέλνει ASCII χαρακτήρες στο Avalon interface *avalon_char_slave* του VGA Core το οποίο χειρίζεται τη μετατροπή των χαρακτήρων σε pixels. Κατά την αρχικοποίηση του συστήματος, το VGA Core θέτει όλους τους χαρακτήρες στο χαρακτήρα "space" ώστε να μην εμφανίζεται κανένας. Αυτή η λειτουργία εκκαθάρισης της οθόνης μπορεί να καταναλώσει έως και 5000 κύκλους ρολογιού για να ολοκληρωθεί, διαδικασία που δεν αναμένεται να δημιουργήσει προβλήματα. Μετά την αρχικοποίηση, παρέχεται δυνατότητα εκκαθάρισης της οθόνης με την εγγραφή ενός "1" στην υψηλότερη θέση μνήμης του *avalon_char_slave*. Κατά τη διαδικασία αυτή, κανένα δεδομένο δεν μπορεί να γραφτεί ή να διαβαστεί. Το χρώμα κάθε χαρακτήρα μπορεί επίσης να οριστεί χρησιμοποιώντας το RGB μοντέλο χρωμάτων ακολουθώντας τις εξής ρυθμίσεις:

- 1-bit color mode – σύμφωνα με την οποία χαρακτήρες εμφανίζονται με λευκό χρώμα, όπως φαίνεται στην εικόνα 4.9



Εικόνα 4.9 - 1 bit character mode

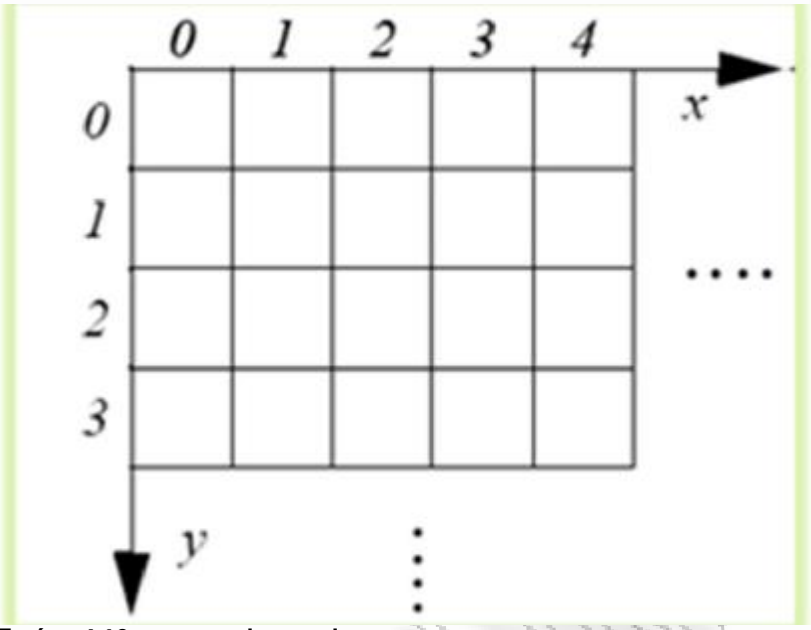
- 4-bit color mode
- 8-bit color mode
- 9-bit color mode

Η ανάλυση των χαρακτήρων ορίζεται από τον αριθμό των χαρακτήρων (του ίδιου μεγέθους) ανά γραμμή και τον αριθμό των γραμμών ανά οθόνη. Τα εξής μεγέθη υποστηρίζονται:

- 8x8 – κάθε χαρακτήρας καταλαμβάνει μια ομάδα pixel 8x8. Συνεπώς, μπορούν να εμφανιστούν 80 χαρακτήρες ανά γραμμή και 60 γραμμές ανά οθόνη.
- 16x16 – κάθε χαρακτήρας καταλαμβάνει μια ομάδα pixel 16x16. Συνεπώς, μπορούν να εμφανιστούν 40 χαρακτήρες ανά γραμμή και 30 γραμμές ανά οθόνη.

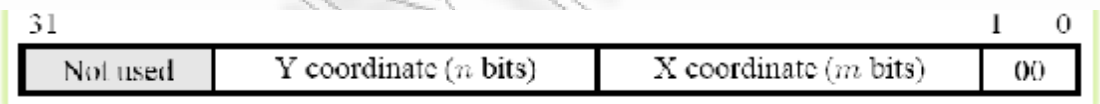
Character Overlay Mode

Στη λειτουργία αυτή, παρέχεται η δυνατότητα εμφάνισης και χαρακτήρων και pixels με τους χαρακτήρες να εμφανίζονται πάνω από τα pixels. Οι συντεταγμένες αποθηκεύονται στις θέσεις μνήμης του διαθέσιμου χώρου του VGA Core. Όλα τα δεδομένα που διαχειρίζεται το VGACore είναι δεδομένα των 32-bit. Ο υποστηριζόμενος τρόπος διευθυνσιοδότησης ονομάζεται X-Y mode, σύμφωνα με τον οποίο η διεύθυνση των δεδομένων περιέχει τις x, y συντεταγμένες όπως φαίνεται στην εικόνα 4.10.

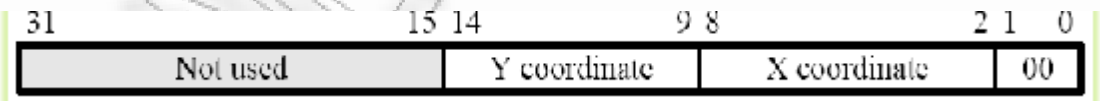


Εικόνα 4.10 – x-y overlay mode

Η μορφή της διεύθυνσης (address format) φαίνεται στην εικόνα όπου οι τιμές των m και n σχετίζονται με την ανάλυση του VGA Core σύμφωνα με τους τύπους: $m=ceil(\log_2 X)$ και $n=ceil(\log_2 Y)$, όπου τα X, Y είναι οι συντεταγμένες των X,Y διευθύνσεων αντίστοιχα.



Για παράδειγμα, για την ανάλυση 80x60 ισχύει: $m=ceil(\log_2 80)=7$ και $n=ceil(\log_2 60)=6$. Έτσι, για αυτή την ανάλυση το address format παρουσιάζεται στην εικόνα 4.11.



Εικόνα 4.11 – 80x60 address format

Στον πίνακα 4.7 παρουσιάζεται η σύνθεση συχνά χρησιμοποιούμενων χρωμάτων. Οι RGB τιμές κυμαίνονται από 0 έως 255. Οποιοσδήποτε συνδυασμός τους παράγει και το αντίστοιχο χρώμα.

Όνομα Χρώματος	R	G	B	Χρώμα
Red	255	0	0	
Green	0	255	0	
Blue	0	0	255	
Yellow	255	255	0	
Cyan	0	255	255	
Magenta	255	0	255	

Πίνακας 4.7 - Η σύνθεση συχνά χρησιμοποιούμενων χρωμάτων

Το VGA Core μπορεί να ενσωματωθεί εύκολα σε ένα σύστημα και να παραμετροποιηθεί μέσω του SOPC Builder όπου παρέχονται οι ακόλουθες ρυθμίσεις:

- Επιλογή τρόπου λειτουργίας (mode tab)
- Pixel Mode – ενεργοποιεί τον τρόπο λειτουργίας pixel mode. Επίσης, ρυθμίζει την ανάλυση και το χρώμα των pixels του VGA Core καθώς και τη λειτουργία του back buffering.
- Character Mode – ενεργοποιεί τον τρόπο λειτουργίας character mode. Ρυθμίζει επίσης το μέγεθος και το χρώμα των χαρακτήρων του VGA Core.
- Character Overlay Mode – ενεργοποιεί τον τρόπο λειτουργίας character overlay mode.

Έτοιμες C συναρτήσεις παρέχονται και για το VGA Core οι οποίες υλοποιούν τις βασικές λειτουργίες ενός VGA Controller. Για τη σωστή λειτουργία των συναρτήσεων αυτών, θα πρέπει στο κυρίως πρόγραμμα/εφαρμογή να συμπεριληφθεί το αρχείο “altera_up_avalon_vga.h”. Επίσης, όταν έχει επιλεχθεί ο τρόπος λειτουργίας Character Overlay Mode, πρέπει να ορισθεί ένα σύμβολο με όνομα OVERLAY_MODE και να προστεθεί στα flags του compiler. Οι συναρτήσεις αυτές είναι οι εξής :

- alt_up_vga_open_dev

Η συνάρτηση αυτή ενεργοποιεί το VGA Core.

- alt_up_vga_draw_pixel

Η συνάρτηση αυτή εμφανίζει ένα pixel σε συγκεκριμένες συντεταγμένες (x,y) πάνω στη VGA οθόνη.

- alt_up_vga_draw_pixel_with_back_buffer

Η συνάρτηση αυτή εμφανίζει ένα pixel σε συγκεκριμένες συντεταγμένες (x,y) πάνω στη VGA οθόνη με εναλλαγή του back buffer.

- alt_up_vga_draw_char_1b

Η συνάρτηση αυτή εμφανίζει ένα χαρακτήρα σε συγκεκριμένες συντεταγμένες (x,y) πάνω στη VGA οθόνη με λευκό χρώμα.

4.3.5 **Ο πυρήνας υλικού** «PIO Core»

Το PIO (Parallel Input/Output) Core [23] παρέχει τη δυνατότητα μεταφοράς δεδομένων ανάμεσα στον Nios II και τις I/O συσκευές (όπως LEDS, switches κτλ). Η μεταφορά αυτή πραγματοποιείται με παράλληλο τρόπο (1-32 bits). Το συγκεκριμένο core περιέχει τέσσερις καταχωρητές, όπως φαίνεται στον πίνακα 4.8, που είναι προσβάσιμοι ως θέσεις μνήμης.

Offset in Bytes	Register Name		Read/Write	Bits (n-1)...0
0	data	Input	R	Δεδομένα που υπάρχουν στα PIOs.
		Output	W	Δεδομένα που πρέπει να εγγραφούν στα PIOs.
4	direction		R/W	Ανεξάρτητος έλεγχος διεύθυνσης για κάθε IO port. Η τιμή "0" θέτει τη διεύθυνση ως είσοδο, ενώ η τιμή "1" ως έξοδο.
8	interruptmask		R/W	Ενεργοποίηση/απενεργοποίηση IRQ για κάθε είσοδο. Η τιμή "1" ενεργοποιεί τις διακοπές για την αντίστοιχη είσοδο.
12	edgecapture		R/W	Edge detection για κάθε είσοδο.

Πίνακας 4.8 – Οι καταχωρητές του PIO core

Ο *data καταχωρητής* (καταχωρητής δεδομένων) αποθηκεύει τα δεδομένα (n-bits) που μεταφέρονται μεταξύ των PIOs και του Nios II. Μπορεί να υλοποιηθεί ως καταχωρητής εισόδου, εξόδου ή εισόδου-εξόδου (bidirectional).
Ο *address καταχωρητής* (καταχωρητής διεύθυνσης) ορίζει τη διεύθυνση των δεδομένων για κάθε ένα από τα n-bit δεδομένα. Η τιμή "0" θέτει την διεύθυνση ως είσοδο, ενώ η τιμή "1" ως έξοδο.
Ο *interruptmask* καταχωρητής χρησιμοποιείται για την ενεργοποίηση των διακοπών των PIOs.
Ο *edgecapture* καταχωρητής υποδεικνύει αλλαγή της λογικής τιμής των σημάτων συνδεδεμένα στα PIOs.
Ο αριθμός *n* των bits καθώς και η διεύθυνση της μεταφοράς (Input, Output, Bidirectional) καθορίζονται μέσω του SOPC Builder.

4.3.6 **Ο πυρήνας υλικού** «SDRAM Controller Core»

Ο SDRAM Controller [12] επικοινωνεί με το SDRAM chip του DE1Board και παρέχει την κατάλληλη διασύνδεση για τη χρήση του SDRAM chip. Αναλυτικότερα, ο SDRAM controller παρέχει μία byte-addressable διασύνδεση προς το SDRAM chip του DE1 Board. Με αυτόν τον τρόπο μία master συσκευή (όπως ο επεξεργαστής Nios II), μπορεί να διαβάσει ή/και να γράψει διαδεδομένα στον SDRAM controller. Ο SDRAM controller χρειάζεται περίπου 100μs για να εκκινήσει μετά το άνοιγμα του DE1 board πριν να μπορεί να είναι προσβάσιμος. Θα υπάρχει επίσης κάποια καθυστέρηση στην ανάγνωση/εγγραφή δεδομένων λόγω του χρόνου ανανέωσης της SDRAM μνήμης. Το απαραίτητο ρολόι για το core αυτό παρέχεται, όπως έχει ήδη αναφερθεί, από το DE Boards External Interface Core. Ο συγκεκριμένος controller μπορεί να ενσωματωθεί σε οποιοδήποτε σύστημα μέσα από τον SOPC Builder χωρίς την ανάγκη αρχικοποίησης / παραμετροποίησης.

4.3.7 **Ο πυρήνας υλικού** «SRAM Controller Core»

Ο SRAM Controller [26] επικοινωνεί με μία 256Kx16 ασύγχρονη CMOS στατική RAM η οποία περιέχεται στην DE1 board. Παρέχει την απαραίτητη διασύνδεση τύπου διευθυνσιοδότησης – λέξης (byte addressable) για την λειτουργία του SRAM ολοκληρωμένου της κάρτας. Μετατρέποντας τα σήματα από το Avalon Switch Fabric σε κατάλληλα για το SRAM ολοκληρωμένο, επιτρέπει στους χρήστες να γράφουν ή να διαβάζουν την SRAM από μία master συσκευή όπως ο NIOS II σαν μία απλή διαδικασία μνήμης. Το Avalon Switch Fabric διευθυνσιοδοτεί αυτόματα και υποστηρίζει 8, 16 και 32-bit μεταφορές ανάγνωσης/εγγραφής. Για 32-bit δεδομένα το Avalon Switch Fabric χωρίζει τα δεδομένα σε δύο λέξεις των 16-bit και τις μεταφέρει μία μία. Έτσι έχουμε μία καθυστέρηση πέντε κύκλων ρολογιού για ανάγνωση και δύο κύκλων για εγγραφή στην SRAM.

4.3.8 **Ο πυρήνας υλικού** «PS/2 Port Core»

Η PS/2 θύρα [28] χρησιμοποιείτε για την διασύνδεση πληκτρολόγιου ή ποντικιού με την κάρτα. Το PS/2 Core χειρίζεται τον χρονισμό του σειριακού πρωτοκόλλου μετάδοσης δεδομένων. Με τον κατάλληλο οδηγό μπορούμε να επικοινωνήσουμε διαβάζοντας ή γράφοντας δεδομένα και καταχωρητές ελέγχου. Το Core περιέχει ένα FIFO 256 λέξεων για την αποθήκευση των δεδομένων που λαμβάνονται από την PS/2 θύρα. Οι οδηγοί της συσκευής ελέγχουν και επικοινωνούν με το Core μέσω δύο 32-bit καταχωρητών, οι οποίοι γράφονται και διαβάζονται διαμέσου του Avalon Slave Port. Ο παρακάτω πίνακας δείχνει τις λεπτομέρειες των καταχωρητών.

Table 1. PS/2 Core register map

Offset in bytes	Register Name	R/W/C	Bit description						
			31…16	15…11	10	9	8	7…1	0
0	data	R/W	RAVAIL	*(1)*				DATA	
4	control	R/C	*(1)*		CE	*(1)*	RI	*(1)*	RE

Πίνακας 4.9 – Οι καταχωρητές του PS2 Core

Table 2. data register bits

Bit number	Bit name	Read/Write/Clear	Description
7…0	DATA	R/W	The value to transfer to/from the PS/2 core. When writing, the DATA field is interpreted as a command to be sent to the PS/2 device. When reading, the DATA field is data from the PS/2 device.
31…16	RAVAIL	R	The number of data items remaining in the read FIFO (including this read).

Πίνακας 4.10 – Οι καταχωρητές δεδομένων του PS2 Core

Table 3. control register bits

Bit number	Bit name	Read/Write/Clear	Description
0	RE	R/W	Interrupt-enable bit for read interrupts.
8	RI	R	Indicates that a read interrupt is pending.
10	CE	C	Indicates that an error occurred while trying to send a command to a PS/2 device.

Πίνακας 4.11 – Ο καταχωρητής ελέγχου του PS2 core

Το PS/2 Core παρέχει συναρτήσεις C μέσω του Hardware Abstraction Layer (HAL), οι οποίες υλοποιούν βασικές λειτουργίες της PS/2 θύρας. Γι' αυτό τον σκοπό πρέπει να συμπεριλάβουμε στον κώδικα την ακόλουθη δήλωση: #include "altera_up_avalon_ps2.h"

4.3.9 **Ο πυρήνας υλικού** «SYSTEM ID Core»

Το SYSTEM ID Core [5], [6] είναι μία απλή συσκευή μόνο για ανάγνωση που παρέχει τα συστήματα SOPC με μία μοναδική ταυτότητα. Τα συστήματα με επεξεργαστή NIOS II χρησιμοποιούν το SYSTEM ID Core για να πιστοποιείτε ότι το εκτελέσιμο πρόγραμμα έχει μεταγλωττιστεί για ένα και μοναδικό σύστημα υλικού. Το SYSTEM ID Core παρέχει μία διασύνδεση μόνο ανάγνωσης τύπου Avalon Memory-Mapped slave (Avalon-MM), αυτή έχει δύο 32-bit καταχωρητές, η τιμή του κάθε καταχωρητή καθορίζεται κατά τον χρόνο δημιουργίας του συστήματος και πάντα μας δίνει μία σταθερή τιμή.

Offset	Register Name	R/W	Description
0	id	R	A unique 32-bit value that is based on the contents of the SOPC Builder system. The id is similar to a check-sum value; SOPC Builder systems with different components, different configuration options, or both, produce different id values.
1	timestamp	R	A unique 32-bit value that is based on the system generation time. The value is equivalent to the number of seconds after Jan. 1, 1970.

4.3.10 **Ο πυρήνας υλικού** «Performance Counter Core»

Το Performance Counter Core [5] παρέχει την δυνατότητα για ακριβή σε πραγματικό χρόνο μέτρηση του χρόνου εκτέλεσης ενός προγράμματος, σε πολλαπλά τμήματα αυτού. Χρειάζεται μόνο να προσθέσουμε μία εντολή στην αρχή και το τέλος του τμήματος του κώδικα που θέλουμε να μετρήσουμε. Το Performance Counter Core περιέχει δύο μετρητές για κάθε τμήμα του κώδικα, έναν 64-bit που μετράει κύκλους ρολογιού και ένα 32-bit που μετράει γεγονότα (event counter) δηλαδή πόσες φορές εκτελείτε το τμήμα του κώδικα που θέλουμε να μετρήσουμε. Ένας κεντρικός μετρητής (Global Counter) των 64-bit ελέγχει όλους τους τμηματικούς μετρητές.

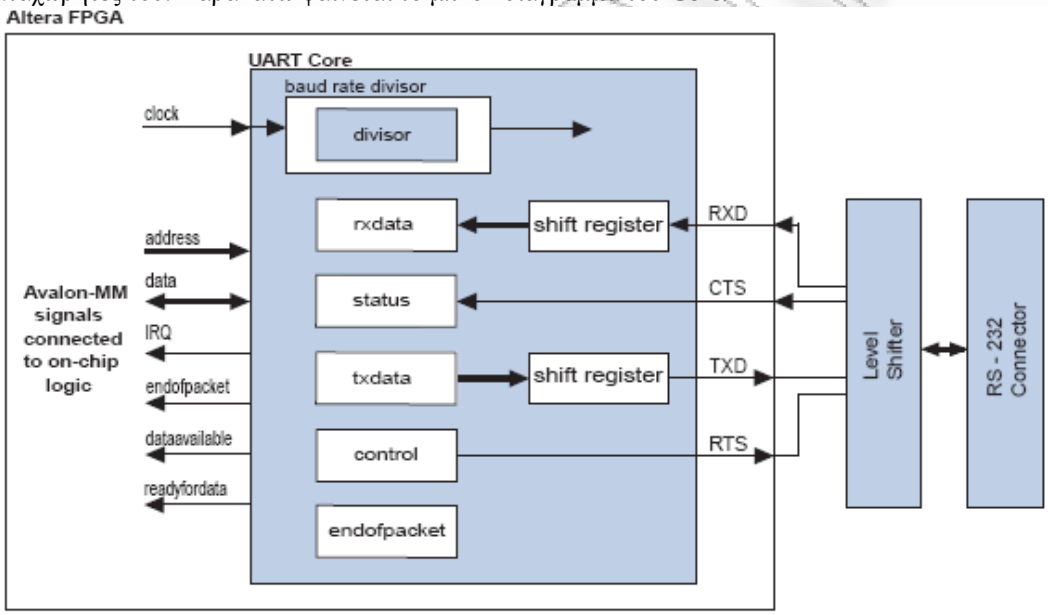
Offset	Register Name	Bit Description		
		Read	Write	
		31 ... 0	31 ... 1	0
0	$T[0]_{lo}$	global clock cycle counter [31: 0]	*(1)*	0 = STOP 1 = RESET
1	$T[0]_{hi}$	global clock cycle counter [63:32]	*(1)*	0 = START
2	Ev[0]	global event counter	*(1)*	*(1)*
3	—	*(1)*	*(1)*	*(1)*
4	$T[1]_{lo}$	section 1 clock cycle counter [31:0]	*(1)*	0 = STOP
5	$T[1]_{hi}$	section 1 clock cycle counter [63:32]	*(1)*	0 = START
6	Ev[1]	section 1 event counter	*(1)*	*(1)*
7	—	*(1)*	*(1)*	*(1)*
8	$T[2]_{lo}$	section 2 clock cycle counter [31:0]	*(1)*	0 = STOP
9	$T[2]_{hi}$	section 2 clock cycle counter [63:32]	*(1)*	0 = START
10	Ev[2]	section 2 event counter	*(1)*	*(1)*
11	—	*(1)*	*(1)*	*(1)*
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
$4n+0$	$T[n]_{lo}$	section *n* clock cycle counter [31:0]	*(1)*	0 = STOP
$4n+1$	$T[n]_{hi}$	section *n* clock cycle counter [63:32]	*(1)*	0 = START
$4n+2$	Ev[n]	section *n* event counter	*(1)*	*(1)*
$4n+3$	—	*(1)*	*(1)*	*(1)*

Για τον έλεγχο από το λογισμικό του Core πρέπει να ενσωματώσουμε την παρακάτω επικεφαλίδα : altera_avalon_performance_counter.h, με αυτήν μας παρέχονται τα ακόλουθα macros και συναρτήσεις.

Name	Summary
PERF_RESET()	Stops and disables all counters, resetting them to 0.
PERF_START_MEASURING()	Starts the global counter and enables section counters.
PERF_STOP_MEASURING()	Stops the global counter and disables section counters.
PERF_BEGIN()	Starts timing a code section.
PERF_END()	Stops timing a code section.
perf_print_formatted_report()	Sends a formatted summary of the profiling results to stdout.
perf_get_total_time()	Returns the aggregate global profiling time in clock cycles.
perf_get_section_time()	Returns the aggregate time for one section in clock cycles.
perf_get_num_starts()	Returns the number of counter events.
alt_get_cpu_freq()	Returns the CPU frequency in Hz.

4.3.11 **Ο πυρήνας υλικού** «RS232 UART Core»

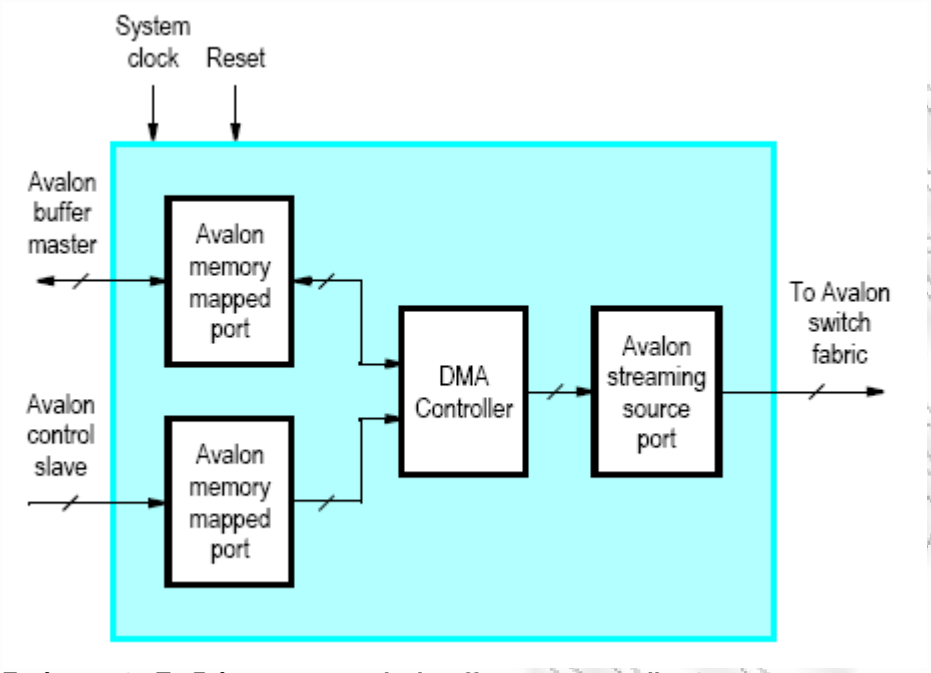
Το RS232 UART Core [25] υλοποιεί το πρωτόκολλο RS232 και παρέχει ρυθμιζόμενο baud rate, parity, stop και data bits. Ο NIOS II επικοινωνεί με το Core γράφοντας και διαβάζοντας τους καταχωρητές του. Παρακάτω φαίνεται το μπλοκ διάγραμμα του Core:



Εικόνα 4.12 - Το μπλοκ διάγραμμα του RS232 UART Core

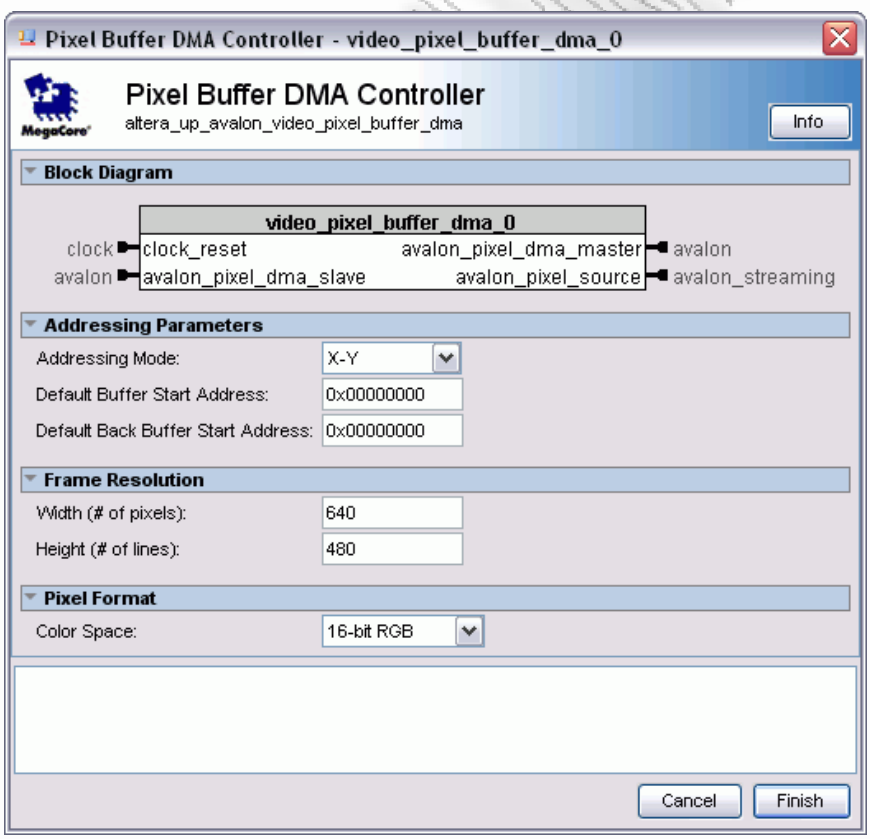
4.3.12 **Ο πυρήνας υλικού** «Pixel Buffer DMA Controller Core»

Το διάγραμμα του Controller [29] φαίνεται παρακάτω. Ο ελεγκτής χρησιμοποιεί την Avalon MM Master διασύνδεση που διαθέτει για να διαβάζει πλαίσια video από μία εξωτερική μνήμη. Ακολούθως στέλνει αυτά τα πλαίσια έξω μέσω της διασύνδεσης Avalon Streaming. Η διασύνδεση Avalon MM Slave χρησιμοποιείτε για επικοινωνία του NIOS II με τους καταχωρητές του ελεγκτή.



Εικόνα 4.13 - Το διάγραμμα του Pixel Buffer DMA Controller Core

Οι ακόλουθες ρυθμίσεις μπορούν να γίνουν στον SOPC.
Addressing Mode: Επιλογή μεταξύ διαδοχικού και X-Y τρόπου διευθυνσιοδότησης
Default buffer Start Address: Η διεύθυνση έναρξης του buffer



Εικόνα 4.14 – Οι ρυθμίσεις του Pixel Buffer DMA Controller Core

Οι καταχωρητές με τους οποίους για να επικοινωνήσουμε πρέπει να ενσωματώσουμε στον κώδικα την επικεφαλίδα #include "altera_up_avalon_pixel_buffer.h" είναι οι ακόλουθοι:

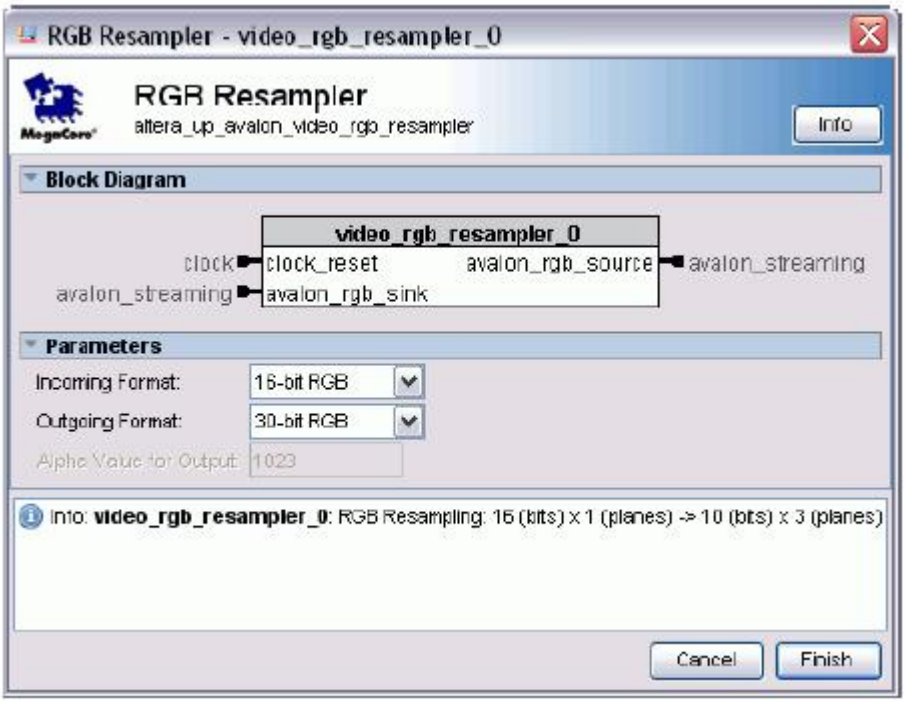
Table 4. Pixel Buffer register map

Offset in bytes	Register Name	R/W	Bit Description							
			31...24	23...16	15...8	7...4	3	2	1	0
0	Buffer	R	Buffer's start address							
4	BackBuffer	R/W	Back buffer's start address							
8	Resolution	R	Y		X					
12	Status	R	m	n	*(1)*	B	*(1)*		A	S

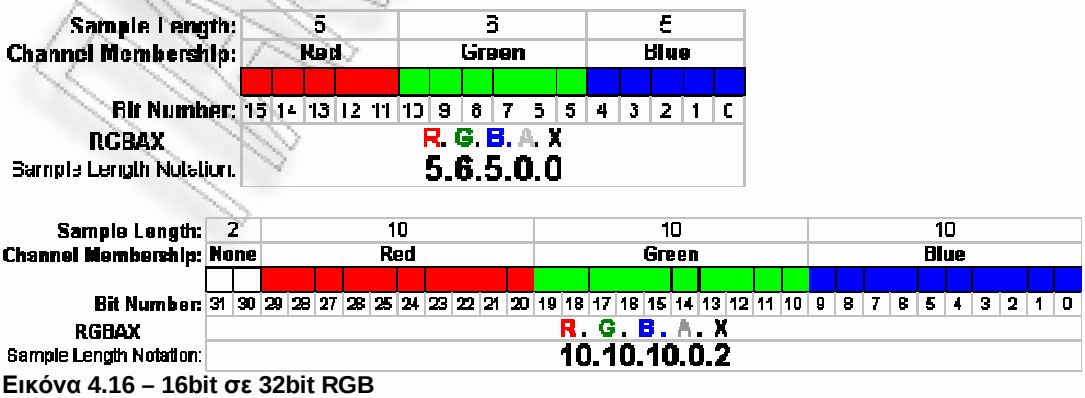
Πίνακας 4.12 – Οι καταχωρητές του Pixel Buffer DMA Controller Core

4.3.13 **Ο πυρήνας υλικού** «RGB Resampler Core»

Ο RGB Resampler [29] μετατρέπει το εύρος σε bit των video streams



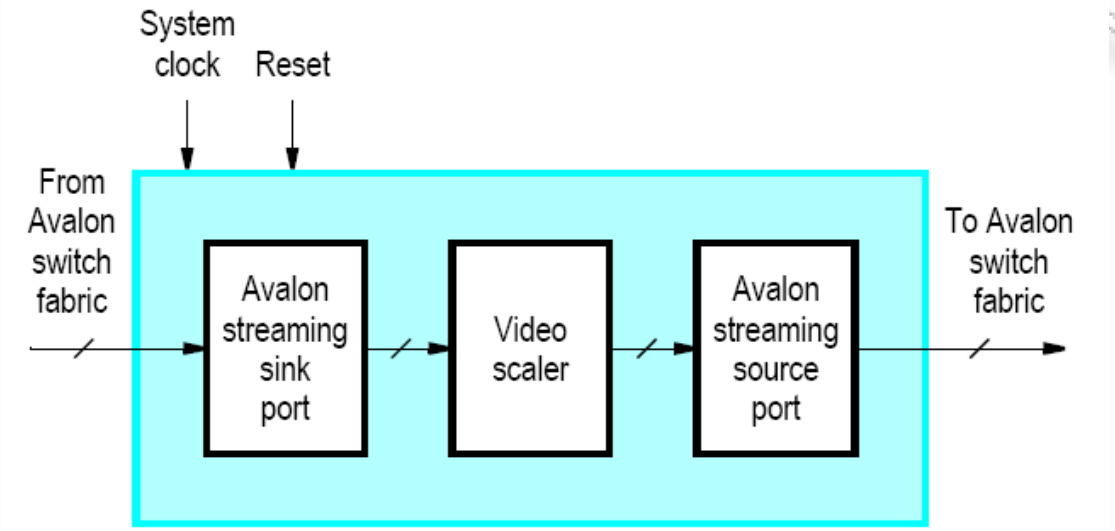
Εικόνα 4.15 – Οι ρυθμίσεις του RGB Resampler Core



Εικόνα 4.16 – 16bit σε 32bit RGB

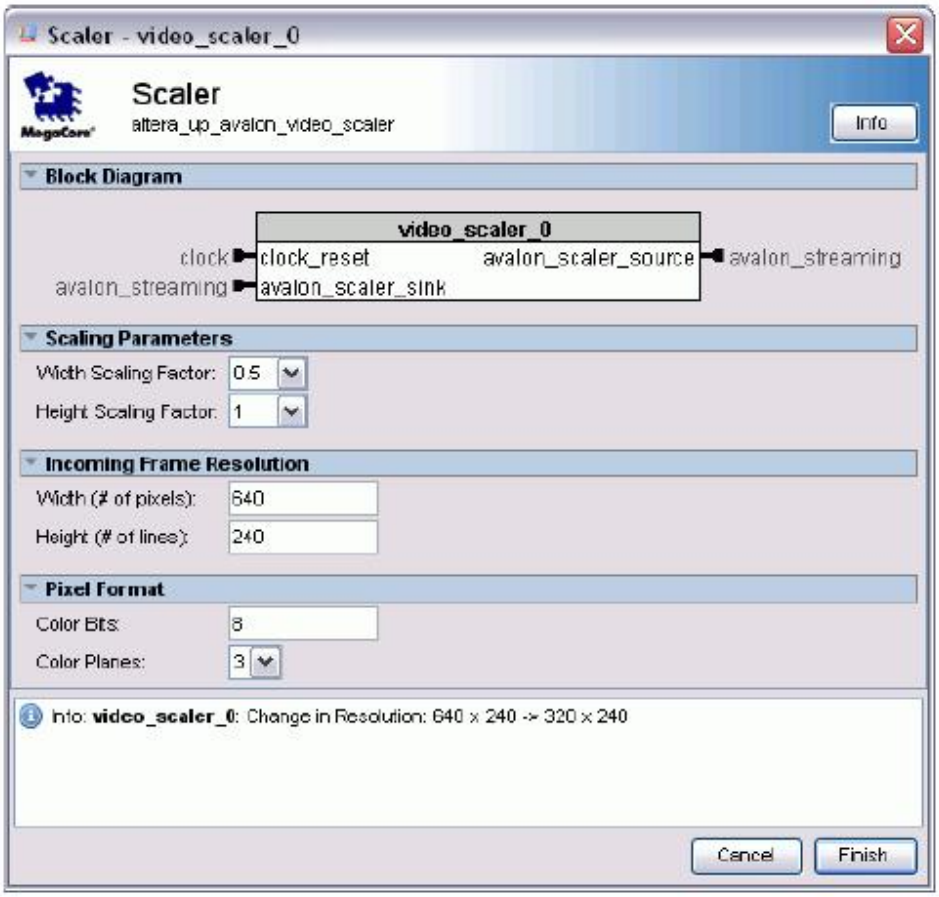
4.3.14 **Ο πυρήνας υλικού** «VIDEO Scaler Core»

Ο VIDEO SCALER [29] μετατρέπει την ανάλυση των video streams, προσθέτοντας ή αφαιρώντας ολόκληρες γραμμές ή στήλες από πίξελ. Το διάγραμμα του Core είναι:



Εικόνα 4.17 - Το διάγραμμα του VIDEO Scaler Core

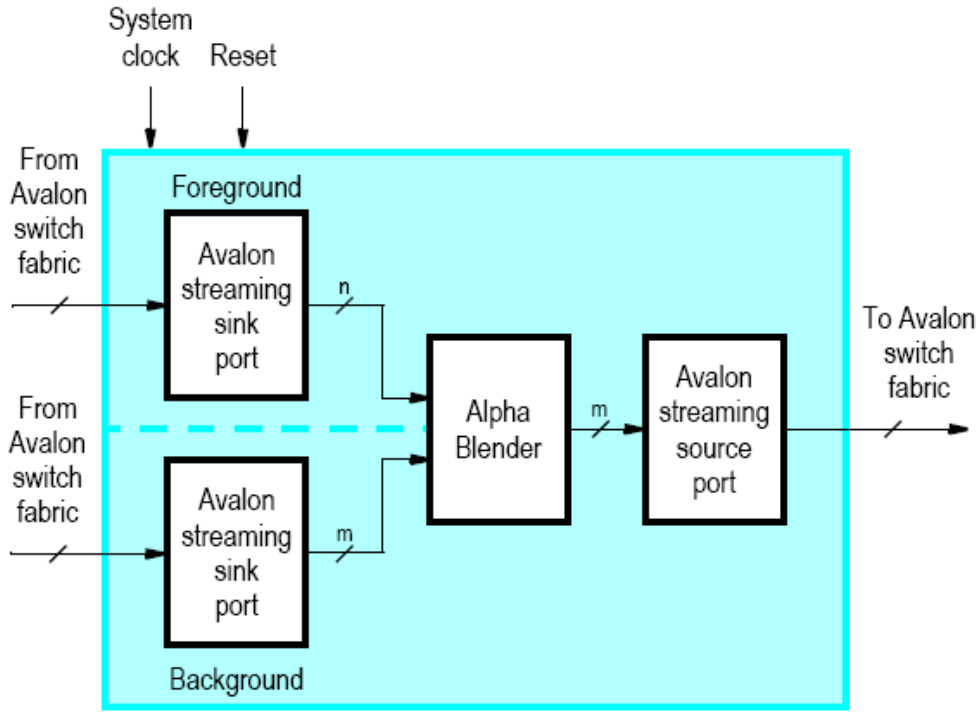
Οι παρακάτω ρυθμίσεις γίνονται στον SOPC:



Εικόνα 4.18 – Οι ρυθμίσεις του VIDEO Scaler Core

4.3.15 **Ο πυρήνας υλικού** "ALPHA BLENDER Core»

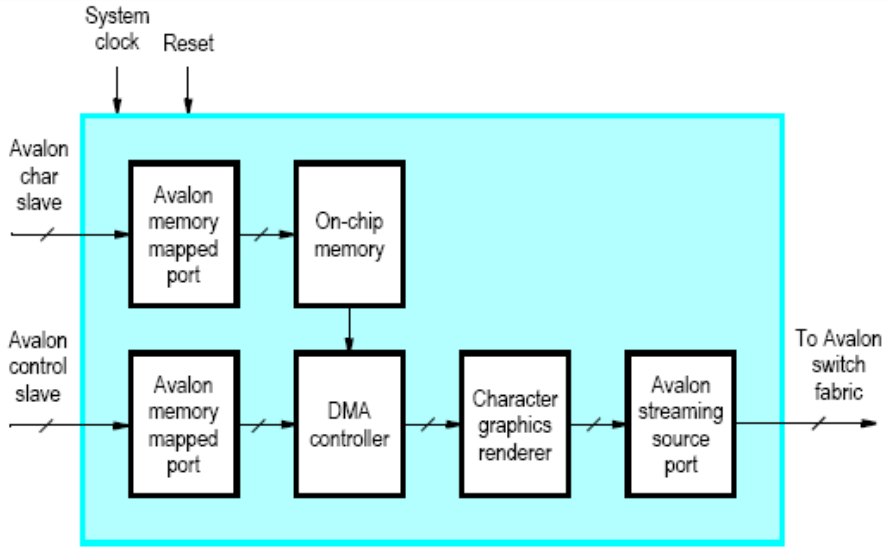
Ο ALPHA BLENDER [29] συνδυάζει δύο video streams σε ένα. Τα δύο streams εισόδου που αποκαλούνται foreground και background, αναμειγνύονται για να δημιουργήσουν ένα stream εξόδου. Το foreground πρέπει να είναι 40-bit RGB και το background 30-bit RGB για να μας δώσουν έξοδο 30-bit RGB.



Εικόνα 4.19 – το διάγραμμα του ALPHA BLENDER Core

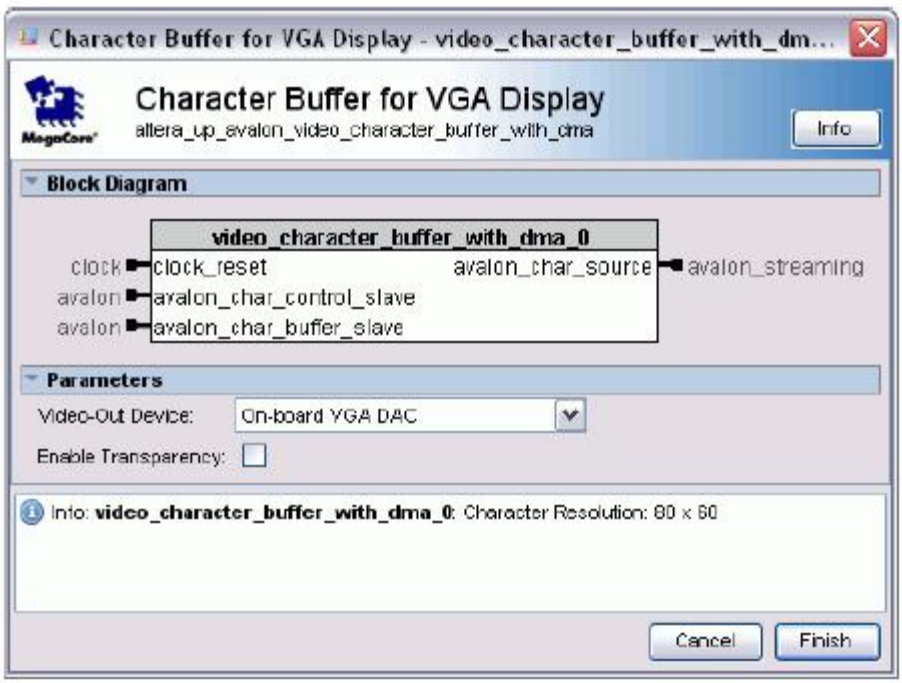
4.3.16 **Ο πυρήνας υλικού** «Character Buffer for VGA Display Core»

Το Core αυτό [29] εμφανίζει ASCII χαρακτήρες στην οθόνη. Ο NIOS II στέλνει χαρακτήρες στην Avalon char MM Slave θύρα του περιφερειακού, αυτό τους αποθηκεύει στην εσωτερική του μνήμη. Ο DMA Controller τους διαβάζει και τους στέλνει στον Character Renderer όπου μετατρέπονται στην αντίστοιχη γραφική τους αναπαράσταση. Το διάγραμμα φαίνεται παρακάτω.



Εικόνα 4.20 – Το διάγραμμα του Character Buffer for VGA Display Core

Οι ακόλουθες ρυθμίσεις μπορούν να γίνουν στο SOPC.



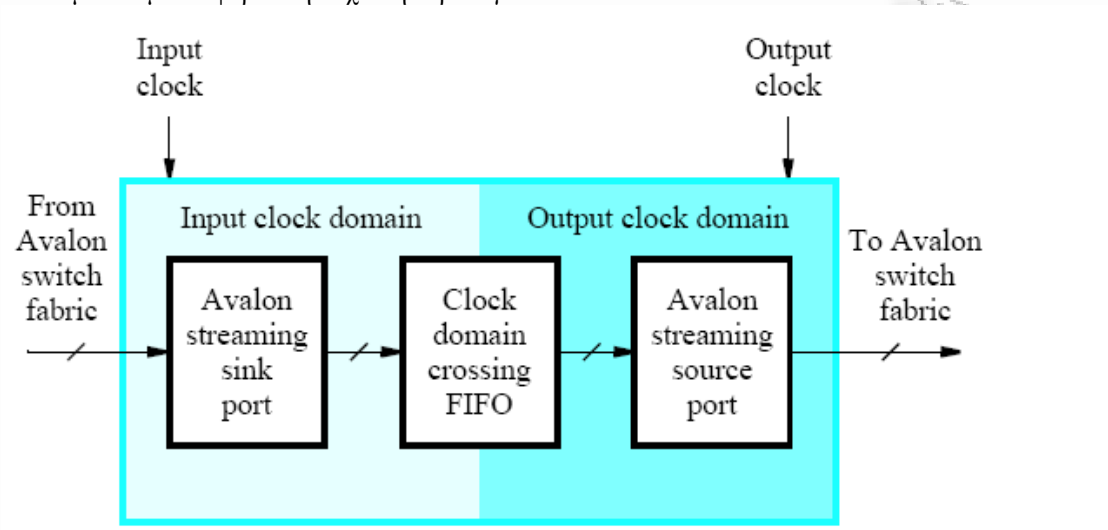
Εικόνα 4.21 – Οι ρυθμίσεις του Character Buffer for VGA Display Core

Στο λογισμικό πρέπει να ενσωματώσουμε την ακόλουθη δήλωση, για χρησιμοποιήσουμε τις απαραίτητες συναρτήσεις για την αξιοποίηση του περιφερειακού:

```
#include "altera_up_avalon_character_buffer.h"
```

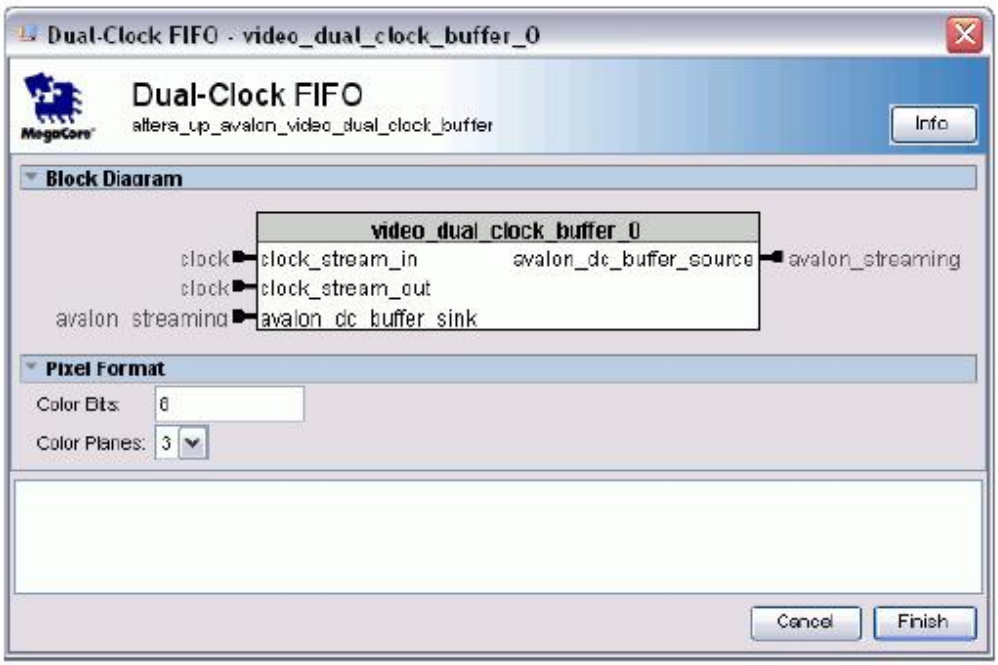
4.3.17 **Ο πυρήνας υλικού** «DUAL – CLOCK FIFO Core»

Ο DUAL – CLOCK FIFO [29] αποθηκεύει τα δεδομένα video και τα μεταφέρει μεταξύ δύο κυκλωμάτων με διαφορετική συχνότητα ρολογιού.



Εικόνα 4.22 – Το διάγραμμα του DUAL – CLOCK FIFO Core

Οι ακόλουθες ρυθμίσεις γίνονται στον SOPC



Εικόνα 4.23 – Οι ρυθμίσεις του DUAL – CLOCK FIFO Core

4.4 **Οι πυρήνες περιφερειακών** «IP Cores»

Τα IP Cores που παρέχονται από την ALTERA, εγκαθίστανται μαζί με το Quartus και μπορούν να ενσωματωθούν σε οποιοδήποτε SOPC σύστημα. Τα περισσότερα IP Cores υποστηρίζονται και από τις κατάλληλες software συναρτήσεις για πλήρη εκμετάλλευση των δυνατοτήτων τους από τον επεξεργαστή Nios II. Τα cores που χρησιμοποιήθηκαν είναι τα εξής:

- On-chip Memory
- JTAG UART
- Timer Core

4.4.1 On chip **Μνήμη**

Τα FPGAs της ALTERA ενσωματώνουν block μνήμης (on-chip memory blocks) [5] τα οποία μπορούν να διαμορφωθούν είτε ως RAM μνήμη είτε ως ROM μνήμη στα SOPC συστήματα. Η on-chip μνήμη παρουσιάζει τα ακόλουθα πλεονεκτήματα:

- Γρήγορος χρόνος πρόσβασης συγκρινόμενη με off-chip μνήμη.
- Ο SOPC builder αρχικοποιεί αυτόματα την on-chip μνήμη μέσα στο σύστημα εξαλείφοντας την ανάγκη για χειροκίνητες συνδέσεις
- Συγκεκριμένα block μνήμης μπορούν να έχουν αρχικοποιημένα δεδομένα κατά την ενεργοποίηση του FPGA (power up)(boot code).

Ένα μειονέκτημα της on-chip μνήμης είναι ότι το μέγεθός της δεν μπορεί να ξεπερνάει το 1 MByte στα μεγαλύτερα FPGAs. Η παραμετροποίηση της on-chip μνήμης επιτυγχάνεται μέσω του περιβάλλοντος του SOPC builder (configuration wizard). Οι διαθέσιμες ρυθμίσεις είναι οι εξής: Memory Type (τύπος μνήμης), Size (μέγεθος) και Read Latency (καθυστέρηση ανάγνωσης).

Memory Type – Η ρύθμιση αυτή ορίζει την εσωτερική δομή της on-chip μνήμης.

- RAM (writable) – δημιουργία μνήμης ανάγνωσης – εγγραφής
- ROM (read only) – δημιουργία μνήμης μόνο ανάγνωσης
- Dual-port access – δημιουργία μνήμης με δύο slave ports επιτρέποντας σε δύο master ports να έχουν ταυτόχρονη πρόσβαση στη μνήμη
- Block type – επιλογή από το Quartus συγκεκριμένου τύπου block μνήμης, κατά τη διαδικασία του fitting
- Auto – το Quartus επιλέγει την καταλληλότερη υλοποίηση της on-chip μνήμης
- M512 – το Quartus επιλέγει να χρησιμοποιήσει τα M512 block μνήμης
- M4K – το Quartus επιλέγει να χρησιμοποιήσει τα M4K block μνήμης
- M-RAM – το Quartus επιλέγει να χρησιμοποιήσει τα M-RAM block μνήμης

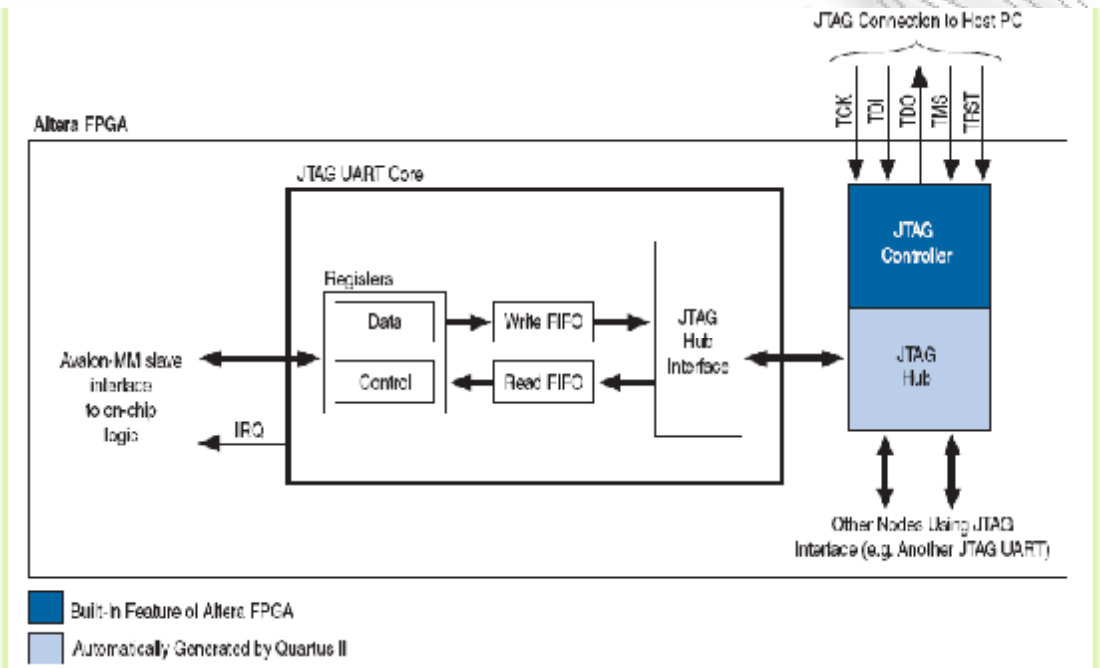
Size – Η ρύθμιση αυτή ορίζει το μέγεθος και το εύρος της on-chip μνήμης.

- Data width – προσδιορισμός του εύρους της μνήμης (8,16,32,64,128,256,512 ή 1024 bits)
- Total memory size – προσδιορισμός του μεγέθους της on-chip μνήμης

Read Latency – Τα στοιχεία της on-chip μνήμης χρησιμοποιούν σύγχρονα, pipelined Avalon-MM slave ports. Το pipeline βελτιώνει την f_{MAX} προσθέτει όμως επιπλέον κύκλους καθυστέρησης κατά τη διαδικασία ανάγνωσης της μνήμης. Η επιλογή **Read Latency** επιτρέπει τον προσδιορισμό των κύκλων καθυστέρησης που απαιτούνται για την ανάγνωση των δεδομένων της μνήμης (η επιλογή Dual-port access επιτρέπει τον ορισμό διαφορετικών κύκλων καθυστέρησης για κάθε slave port).

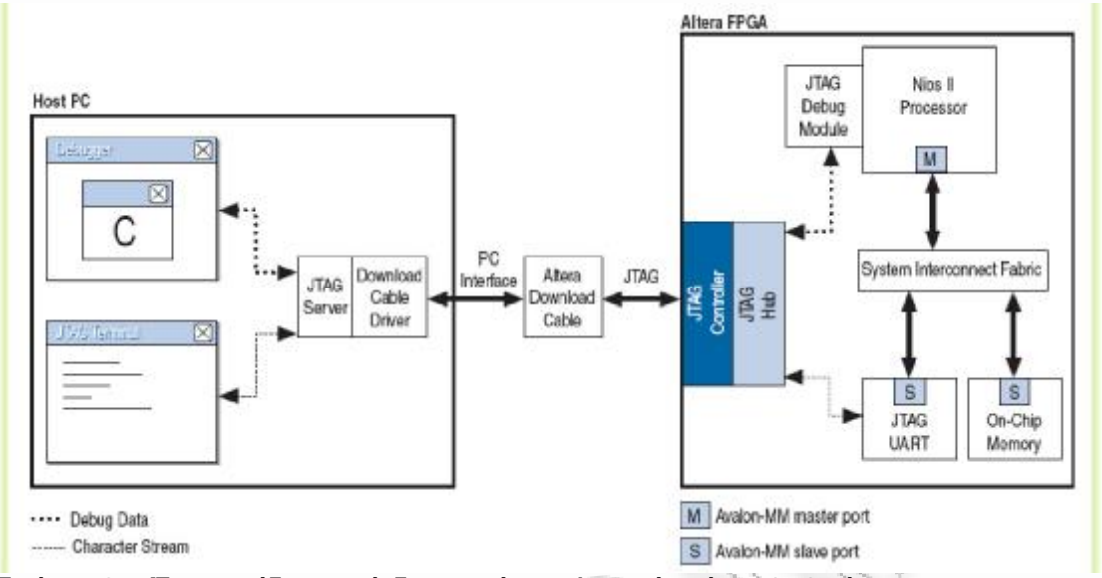
4.4.2 **Ο πυρήνας υλικού** «JTAG UART Core»

Το JTAG UART (Universal Asynchronous Receiver/Transmitter) Core [5] υλοποιεί μια μέθοδο σειριακής επικοινωνίας μεταξύ ενός PC και ενός SOPC συστήματος που βρίσκεται στο FPGA. Το JTAG core μία απλή register-mapped διασύνδεση, έτσι τα κυρίως περιφερειακά (όπως ο Nios II επεξεργαστής) επικοινωνούν με το συγκεκριμένο core διαβάζοντας ή γράφοντας σε συγκεκριμένους καταχωρητές (καταχωρητές ελέγχου και καταχωρητές δεδομένων). Στην εικόνα 4.24 παρουσιάζεται το σχηματικό διάγραμμα του JTAG UART Core .



Εικόνα 4.24 - Το σχηματικό διάγραμμα του JTAG UART Core

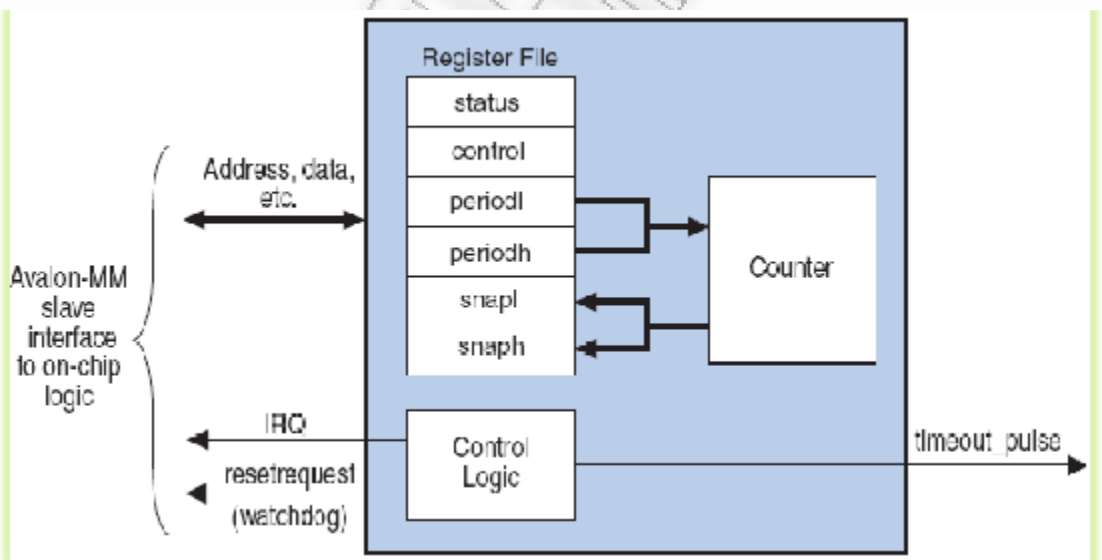
Το JTAG UART Core παρέχει ένα Avalon slave interface προς το JTAG κύκλωμα του FPGA της Altera. Η ορατή στο χρήστη διασύνδεση με το JTAG UART Core αποτελείται από δύο καταχωρητές των 32-bit, δεδομένων και ελέγχου (data και control register) οι οποίοι είναι προσβάσιμοι μέσω του Avalon slave port. Μία Avalon master συσκευή όπως ο επεξεργαστής Nios II, προσπελαύνει τους καταχωρητές για τον έλεγχο του core και για τη μεταφορά δεδομένων μέσω της JTAG σύνδεσης. Το JTAG UART Core παρέχει δύο FIFOs διπλής κατεύθυνσης (bidirectional) για τη βελτίωση της ταχύτητας της JTAG σύνδεσης. Το μέγεθος των FIFOs αυτών είναι παραμετροποιήσιμο ώστε να προσαρμόζεται στην εκάστοτε on-chip μνήμη του συστήματος. Παρέχεται επίσης η δυνατότητα υλοποίησης των FIFOs είτε με blocks μνήμης είτε με καταχωρητές. Τα FPGAs της ALTERA περιέχουν ενσωματωμένο JTAG κύκλωμα ελέγχου. Στην εικόνα 4.25 παρουσιάζεται ένα παράδειγμα σύνδεσης ενός υπολογιστή με ένα SOPC σύστημα που ενσωματώνει το JTAG UART Core.



Εικόνα 4.25 - Ένα παράδειγμα σύνδεσης ενός υπολογιστή με ένα SOPC σύστημα που ενσωματώνει το JTAG UART Core

4.4.3 **Ο πυρήνας υλικού** «Timer Core»

Το Timer core [5] είναι ένας εσωτερικός χρονομέτρης-χρονιστής των 32-bit ή 64-bit με χαρακτηριστικά όπως έλεγχος εκκίνησης, τερματισμού και επαναφοράς. Παρέχει δύο τρόπους απαρίθμησης: απλή αντίστροφη απαρίθμηση και συνεχή αντίστροφη απαρίθμηση που υλοποιούνται με ένα καταχωρητή περιόδου απαρίθμησης (count-down period register) και πραγματοποιεί λειτουργίες όπως δημιουργία διακοπής (interrupt) και παραγωγή παλμών, όταν η απαρίθμηση φτάσει στο μηδέν. Στην εικόνα 4.26 παρουσιάζεται το σχηματικό διάγραμμα του Timer Core.



Εικόνα 4.26 - Το σχηματικό διάγραμμα του Timer Core

Όλοι οι καταχωρητές είναι εύρους 16-bit καθιστώντας τον Timer συμβατό με επεξεργαστές των 16-bit, 32-bit (2 timers των 16-bit) και 64-bit (4 timers των 16-bit). Μερικοί καταχωρητές υπάρχουν μόνο εάν απαιτείται από το hardware. Για παράδειγμα, αν ο timer έχει ρυθμιστεί να δουλεύει με συγκεκριμένη περίοδο που δεν αλλάζει (fixed) τότε οι καταχωρητές περιόδου (*periodl, periodh*) δεν υπάρχουν. Η κύρια λειτουργία του timer έχει ως εξής:

- Μία Avalon-MM master περιφερειακή συσκευή όπως ο επεξεργαστής Nios II γράφει δεδομένα στον καταχωρητή ελέγχου (*control* register) του Timer Core ώστε να αρχίσει ή να σταματήσει τον

timer, να ενεργοποιήσει ή να απενεργοποιήσει τις IRQ ή να καθορίσει τη μέθοδο της αντίστροφης μέτρησης (count-down once ή continuous count-down).

- Ο επεξεργαστής διαβάζει τον καταχωρητή κατάστασης (***status*** register) για την παρούσα κατάσταση του timer.
- Ο επεξεργαστής μπορεί να καθορίσει την περίοδο του timer με την εγγραφή της κατάλληλης τιμής στους καταχωρητές περιόδου ***periodl*** και ***periodh.***
- Ένας εσωτερικός μετρητής μετράει αντίστροφα προς το μηδέν και όποτε φτάσει στο μηδέν επαναφορτώνεται άμεσα από τους καταχωρητές περιόδου και αν έχουν ενεργοποιηθεί οι διακοπές, μία IRQ παράγεται, η προαιρετική επιλογή watchdog εκτελεί reset στο σύστημα.
- Ο επεξεργαστής μπορεί να διαβάσει την τιμή του counter γράφοντας αρχικά σε έναν από τους ***snapl*** ή ***snaph*** καταχωρητές για να ζητήσει ένα στιγμιότυπο του μετρητή και έπειτα διαβάζει τη ζητούμενη τιμή από τους ***snapl*** και ***snaph*** καταχωρητές.

Ο Timer Core υλοποιεί μία απλή Avalon_MM slave διασύνδεση για να παρέχει πρόσβαση στο αρχείο καταχωρητών. Το Avalon_MM slave port χρησιμοποιεί το σήμα resetrequest για την υλοποίηση της συμπεριφοράς του timer ως watchdog. Το σήμα αυτό είναι ένα σήμα επαναφοράς το οποίο οδηγεί την είσοδο reset όλων των Avalon_MM περιφερειακών του συστήματος στον SOPC builder. Το Timer core παρέχεται έτοιμο για χρήση στον SOPC Builder με δυνατότητες παραμετροποίησης. Το συγκεκριμένο core προσφέρει τις παρακάτω ρυθμίσεις οι οποίες επηρεάζουν τη δομή του.

- Simple periodic interrupt – η ρύθμιση αυτή είναι χρήσιμη για συστήματα όπου απαιτείται μία περιοδική γεννήτρια διακοπών. Η περίοδος είναι σταθερή και ο timer δεν μπορεί να σταματήσει αλλά τα IRQ μπορούν να απενεργοποιηθούν.
- Full featured – η ρύθμιση αυτή είναι χρήσιμη για συστήματα που απαιτούν έναν timer με μεταβλητή περίοδο, ο οποίος μπορεί να αρχίσει τη μέτρηση ή να τη σταματήσει, υπό τον έλεγχο του επεξεργαστή.
- Watchdog - η ρύθμιση αυτή είναι χρήσιμη για συστήματα τα οποία απαιτούν ένα μετρητή watchdog για την επαναφορά του συστήματος σε περίπτωση που το σύστημα σταματήσει να ανταποκρίνεται.

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	status	RW	*(1)*					RUN	TO
1	control	RW	*(1)*			STOP	START	CONT	ITO
2	periodl	RW	Timeout Period – 1 (bits 15..0)						
3	periodh	RW	Timeout Period – 1 (bits 31..16)						
4	snapl	RW	Counter Snapshot (bits 15..0)						
5	snaph	RW	Counter Snapshot (31..16)						

Ο καταχωρητής ελέγχου (control register) έχει τέσσερα bits η λειτουργία των οποίων παρουσιάζεται στον πίνακα 4.12

Αριθμός Bit	Όνομα Bit	Ανάγνωση / Εγγραφή/ Εκκαθάριση	Περιγραφή
0	ITO	A/E	Αν το ITO bit έχει πάρει την τιμή "1", το timer core παράγει μία IRQ όταν το TO bit του *status* καταχωρητή έχει την τιμή "1". Αντίθετα το core δεν παράγει κάποια IRQ.
1	CONT	A	Το CONT (continuous) bit καθορίζει την συμπεριφορά του εσωτερικού μετρητή όταν αυτός φτάσει στο μηδέν. Αν έχει την τιμή "1", ο μετρητής απαριθμεί συνεχώς έως ότου τερματιστεί από το STOP bit. Αν έχει την τιμή "0", ο μετρητής σταματάει όταν φτάσει στο μηδέν και επαναφορτώνεται με την τιμή που είναι αποθηκευμένη στους *periodl* and *periodh* καταχωρητές, ανεξάρτητα από την τιμή του CONT bit.
2	START	E	Για την εκκίνηση του απαριθμητή (αντίστροφη μέτρηση) αρκεί να γραφτεί η τιμή "1" στο START bit. Αν ο μετρητής έχει σταματήσει την απαρίθμηση τότε αν γραφτεί η τιμή "1" στο START bit ο μετρητής ξαναρχίζει από τον αριθμό που είχε σταματήσει.
3	STOP	E	Για τον τερματισμό του απαριθμητή αρκεί να γραφτεί η τιμή "1" στο STOP bit που σταματάει τη διαδικασία απαρίθμησης. Αν ο μετρητής έχει σταματήσει την απαρίθμηση τότε αν γραφτεί η τιμή "1" στο STOP bit δεν επηρεάζεται η λειτουργία του.

Πίνακας 4.12 - Ο καταχωρητής ελέγχου (control register) του Timer core

Ο καταχωρητής κατάστασης (status register) έχει δύο bits η λειτουργία των οποίων παρουσιάζεται στον πίνακα 4.13

Αριθμός Bit	Όνομα Bit	Ανάγνωση / Εγγραφή/ Εκκ.	Περιγραφή
0	TO	A/EK	Το TO (timeout) bit τίθεται στην τιμή "1" όταν ο μετρητής φτάσει στο μηδέν. Και μένει στην τιμή αυτή έως ότου "εκκαθαριστεί" από κάποια master περιφερειακή συσκευή.
1	RUN	E	Το RUN bit διαβάζει την τιμή "1" όσο ο εσωτερικός μετρητής απαριθμεί. Σε αντίθετη περίπτωση διαβάζει την τιμή "0".

Πίνακας 4.13 - Ο καταχωρητής κατάστασης (control register) του Timer core

Οι καταχωρητές περιόδου (*periodl* και *periodh*) αποθηκεύουν την τιμή της περιόδου (time out period). Στον *periodl* αποθηκεύονται τα λιγότερο σημαντικά 16-bits, ενώ στον *periodh* τα περισσότερο

σημαντικά 16-bits. Ο εσωτερικός μετρητής του core φορτώνεται με την τιμή που έχει αποθηκευθεί σε αυτούς τους καταχωρητές είτε όταν εκτελεστεί μία λειτουργία εγγραφής στους καταχωρητές αυτούς ή όταν ο εσωτερικός μετρητής φτάσει στο μηδέν. Οι καταχωρητές "στιγμιότυπου" (*snapl* και *snaph*) παρέχουν ένα στιγμιότυπο της κατάστασης του εσωτερικού μετρητή σε κάποια master περιφερειακή συσκευή, όταν αυτή εκτελέσει μία λειτουργία εγγραφής στους καταχωρητές αυτούς (στον *snapl* αποθηκεύονται τα λιγότερο σημαντικά 16-bits, ενώ στον *snaph* τα περισσότερο σημαντικά 16-bits). Τότε, η τιμή του μετρητή αντιγράφεται στους snap_n καταχωρητές. Το στιγμιότυπο λαμβάνεται είτε λειτουργεί ο μετρητής είτε όχι, και δεν αλλάζει την εσωτερική λειτουργία του καταχωρητή.

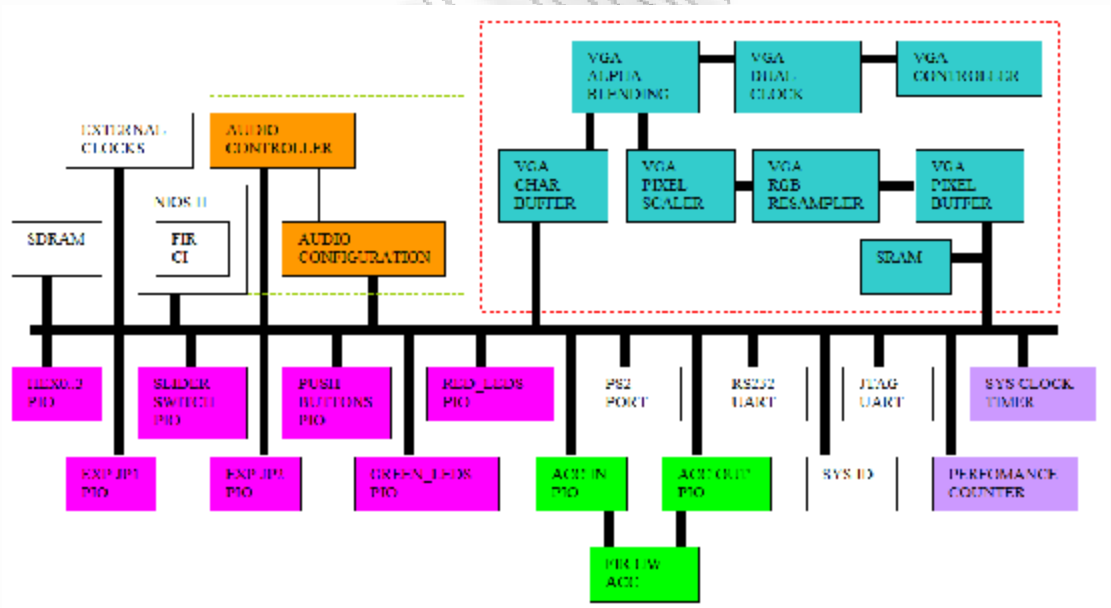
Το Timer Core παράγει μία IRQ κάθε φορά που ο εσωτερικός μετρητής πάρει την τιμή μηδέν και η τιμή του ITO bit του καταχωρητή ελέγχου είναι "1". Η απενεργοποίηση των IRQs γίνεται με έναν από τους ακόλουθους δύο τρόπους: i) εκκαθάριση του TO bit του καταχωρητή κατάστασης ή ii) απενεργοποίηση των διακοπών με εκκαθάριση του ITO bit του καταχωρητή ελέγχου. Η ALTERA δίνει C συναρτήσεις-οδηγούς που είναι προσβάσιμες μέσω του HAL με τους οποίους επιτυγχάνεται η πρόσβαση στους καταχωρητές του core και ο προγραμματισμός του. Η ALTERA παρέχει οδηγούς και για τα δύο μοντέλα του HAL timer: system clock timer και timestamp timer.

4.4.4 **Οδηγός χρονισμού συστήματος** (System Clock Driver)

Όταν ο timer [5] έχει ρυθμιστεί ως system clock, τότε λειτουργεί συνεχώς σε περιοδική βάση, χρησιμοποιώντας την τιμή της περιόδου που έχει οριστεί από τον SOPC Builder. Ο οδηγός είναι καθοδηγούμενος-από-διακοπές (driven-by-interrupts) έτσι θα πρέπει τα σήματα διακοπών του να είναι συνδεδεμένα στο κατάλληλο hardware του συστήματος. Το HAL παρέχει υλοποιήσεις συνήθων UNIX συναρτήσεων όπως gettimeofday(), settimeofday()και άλλες.

4.5 **Περιγραφή δομής υλικού του συστήματος**

Το διάγραμμα του συστήματος με όλα τα Cores [4], [5], [6], [16], [17], [31] είναι το ακόλουθο:



Ο επεξεργαστής NIOS II είναι το κεντρικό core του συστήματος. Επικοινωνεί με τα υπόλοιπα core μέσω του Avalon MM Data bus σαν data master συσκευή, καθώς και για τις μνήμες του συστήματος και με το Avalon MM Instruction bus σαν instruction master συσκευή. Έχουν χρησιμοποιηθεί μια σειρά από core παράλληλης εισόδου εξόδου για επικοινωνία με τον χρήστη. Αυτά είναι : το HEX0..3-PIO για την έξοδο δεδομένων στο 7seg της κάρτας, τα EXP JP1 και EXP JP2 για έξοδο δεδομένων στους ακροδέκτες της κάρτας, το SLIDER SWITCH –PIO για είσοδο δεδομένων από τα 10 slider switch, το PUSH BUTTONS –PIO για είσοδο δεδομένων από τα 4 push buttons, τα RED LEDS και GREEN LEDS –PIO για έξοδο από τα led της κάρτας. Επίσης δύο PIO ένα για είσοδο και ένα για έξοδο χρησιμοποιούνται για την αποστολή και την λήψη δεδομένων προς το εξωτερικό κύκλωμα του FIR Hardware Accelerator. Ένα PS2 core χρησιμοποιείτε για επικοινωνία με πληκτρολόγιο. Το RS232

core έχει τοποθετηθεί για πιθανή χρήση, αποστολής περισσότερων δεδομένων σε ένα PC. Το SYS ID είναι απαραίτητο για την ανάπτυξη κάθε συστήματος SOC με NIOS II. Το JTAG χρησιμοποιείτε για την αποσφαλμάτοση του συστήματος. Ο Perfomance Counter χρησιμοποιείτε για την μέτρηση του χρόνου εκτέλεσης τμημάτων του κώδικα. Ο SYS Clock Timer χρησιμοποιείτε για τον χρονισμό του λειτουργικού συστήματος FreeRtos. Η κύρια μνήμη του συστήματος είναι η SDRAM. Το External Clocks Core δημιουργεί τα απαραίτητα σήματα χρονισμού, ειδικότερα 25MHz για το VGA Core, 12.288MHz για το AUDIO Core και 50MHz ανάλογα καθυστερημένο για την SDRAM. Το AUDIO Configuration Core στέλνει τις κατάλληλες ρυθμίσεις μέσω I2C bus στο AUDIO Codec, ενώ το AUDIO Controller Core παρέχει την μεταφορά δεδομένων μεταξύ επεξεργαστή και AUDIO Codec.

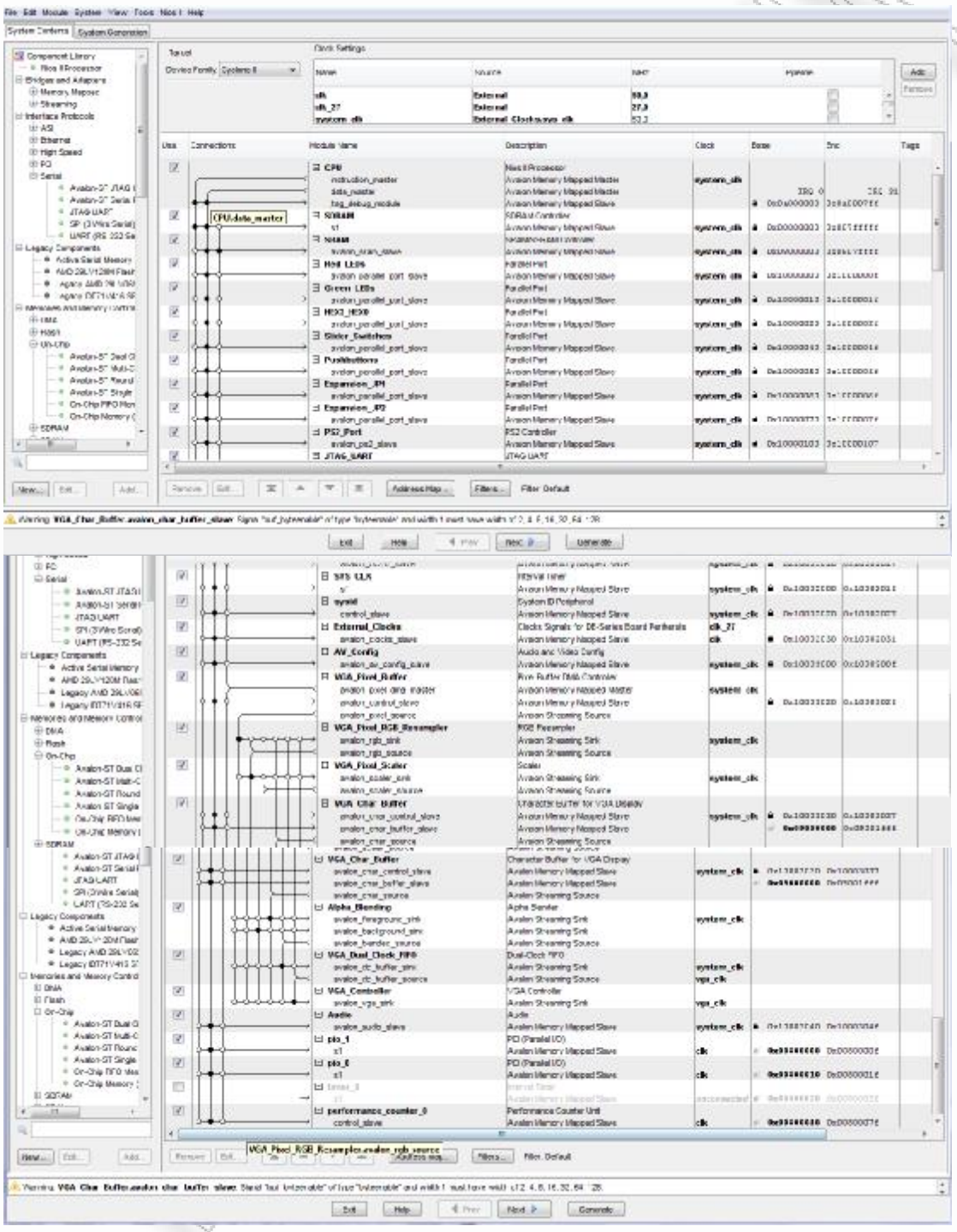
Το τμήμα γραφικών λειτουργεί ως εξής: Ο VGA Pixel Buffer διαβάζει δεδομένα μέσω DMA από μία μνήμη SRAM, όπου εκεί γράφονται από τον επεξεργαστή και τα στέλνει στο VGA Resampler όπου από 16-bit RGB format μετατρέπονται σε 30- bit RGB format. Ακολούθως με τον VGA Scaler μετατρέπουμε την ανάλυση από 320X240 σε 640X480. Από την άλλη με τον VGA Character Buffer δημιουργούμε τους χαρακτήρες που θέλουμε να εμφανίζονται στην οθόνη. Με τον Alpha Blender συνδυάζουμε τα δύο σήματα γραφικά και χαρακτήρες και τα στέλνουμε στον VGA Controller με χρήση του Dual Clock FiFo (από 50MHz σε 25MHz).

Οι διακοπές του επεξεργαστή κατανέμονται ως εξής:

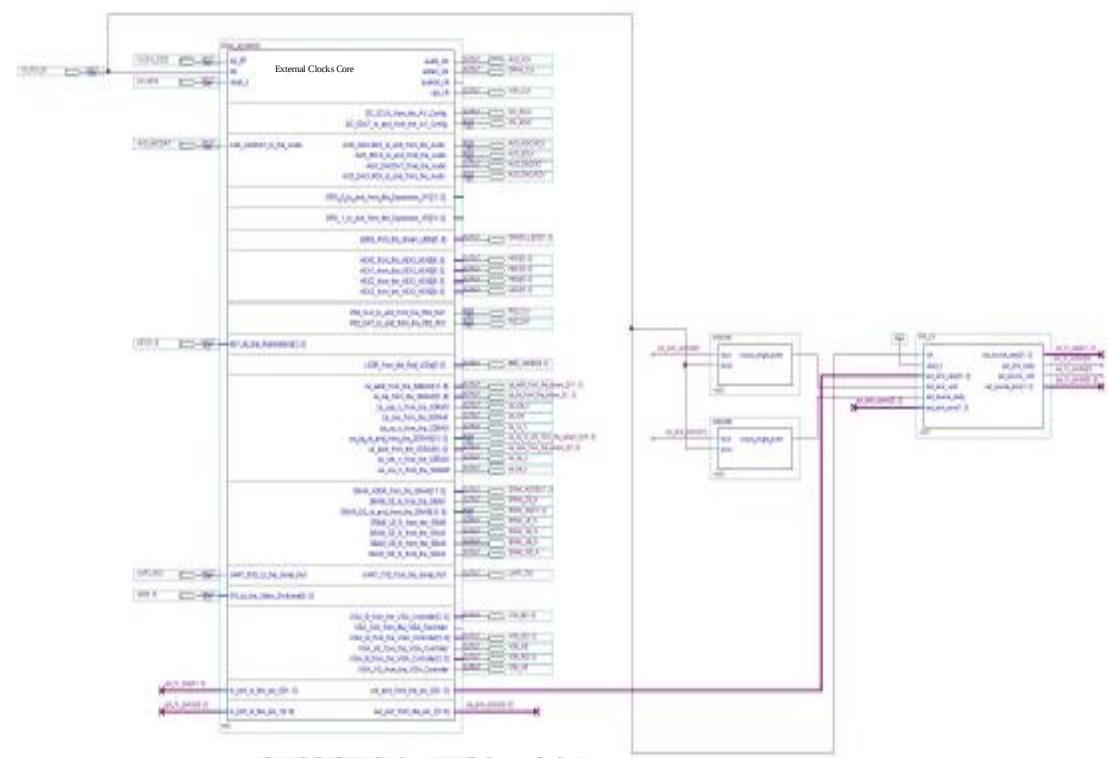
0- SYS CKL TIMER
1- PushButtons
6- Audio Core
7- PS2 Controller Core
8- JTAG_UART Core
10 RS232 Core
11-ExpansionJP1 PIO
12-ExpansionJP2 PIO

Η διευθυνσιοδότηση όλων των πυρήνων του συστήματος φαίνεται στον παρακάτω πίνακα:

Cores Name	Cores Type	clk	Memory starts	Memory Ends
jtag_debug_module	Avalon Memory Mapped Slave		0x0a000000	0x0a0007ff
SDRAM	SDRAM Controller	system_clk	0x00000000	0x007fffff
SRAM	SRAM/SSRAM Controller	system_clk	0x08000000	0x0807ffff
Red_LEDs	Parallel Port	system_clk	0x10000000	0x1000000f
Green_LEDs	Parallel Port	system_clk	0x10000010	0x1000001f
HEX3_HEX0	Parallel Port	system_clk	0x10000020	0x1000002f
Slider_Switches	Parallel Port	system_clk	0x10000040	0x1000004f
Pushbuttons	Parallel Port	system_clk	0x10000050	0x1000005f
Expansion_JP1	Parallel Port	system_clk	0x10000060	0x1000006f
Expansion_JP2	Parallel Port	system_clk	0x10000070	0x1000007f
PS2_Port	PS2 Controller	system_clk	0x10000100	0x10000107
JTAG_UART	JTAG UART	system_clk	0x10001000	0x10001007
Serial_Port	RS232 UART	system_clk	0x10001010	0x10001017
SYS_CLK	Interval Timer	system_clk	0x10002000	0x1000201f
sysid	System ID Peripheral	system_clk	0x10002020	0x10002027
External_Clocks	Clocks Signals for DE-Series Board Periherals	*multiple*	0x10002030	0x10002031
AV_Config	Audio and Video Config	system_clk	0x10003000	0x1000300f
VGA_Pixel_Buffer	Pixel Buffer DMA Controller	system_clk	0x10003020	0x1000302f
VGA_Pixel_RGB_Resampler	RGB Resampler	system_clk		
VGA_Pixel_Scaler	Scaler	system_clk		
VGA_Char_Buffer	Character Buffer for VGA Display	system_clk	*10003030* 09000000	*10003037* 09001ffff
Alpha_Blending	Alpha Blender	system_clk		
VGA_Dual_Clock_FIFO	Dual-Clock FIFO			
VGA_Controller	VGA Controller	vga_clk		
Audio	Avalon Memory Mapped Slave	system_clk	0x10003040	0x1000304f
pio_1	Avalon Memory Mapped Slave	clk	0x00800000	0x0080000f
pio_0	Avalon Memory Mapped Slave	clk	0x00800010	0x0080001f
performance_counter_0	Avalon Memory Mapped Slave	clk	0x00800040	0x0080007f



Το σχηματικό διάγραμμα του συστήματος μαζί με τα εξωτερικά υποκυκλώματα του Επιταχυντή φαίνονται στο παρακάτω σχήμα:



Κεφάλαιο 5

Προσεγγίσεις υλοποίησης FIR φίλτρων

5.1. Εισαγωγή στα ψηφιακά φίλτρα

Ένα ψηφιακό φίλτρο [63], [67], [68] είναι ένας μαθηματικός αλγόριθμος οποίος πραγματοποιείται με την χρήση hardware ή software. Τα ψηφιακά φίλτρα παίζουν έναν πολύ σημαντικό ρόλο στην Ψηφιακή Επεξεργασία Σήματος με πολλές εφαρμογές όπως, συμπίεση δεδομένων, βιο-ιατρική επεξεργασία σήματος, επεξεργασία ομιλίας, επεξεργασία εικόνας, μετάδοση δεδομένων, ψηφιακός ήχος. Τα πλεονεκτήματα των ψηφιακών φίλτρων είναι:

- Με τα ψηφιακά φίλτρα μπορούμε να επιτύχουμε χαραχτηριστικές οι οποίες δεν είναι δυνατόν να επιτευχθούν με αναλογικά φίλτρα, όπως η γραμμική απόκριση φάσης.
- Σε αντίθεση με τα αναλογικά φίλτρα, η απόδοση των ψηφιακών δεν επηρεάζεται από θερμοκρασιακούς παράγοντες. Αυτό έχει σαν αποτέλεσμα να μην χρειάζονται επαναρύθμιση σε τακτά χρονικά διαστήματα όπως τα αναλογικά φίλτρα.
- Η απόκριση συχνότητας μπορεί εύκολα να μεταβληθεί αν το ψηφιακό φίλτρο υλοποιείται από προγραμματιζόμενο μικρο-επεξεργαστή ή από ψηφιακά στοιχεία προγραμματιζόμενης λογικής π.χ. FPGAs.
- Στην πράξη η εξασθένηση η οποία μπορεί να επιτευχθεί με την χρήση αναλογικών φίλτρων φτάνει τα 60db έως 70db λόγω προβλημάτων ευστάθειας και ακρίβειας του αναλογικού συστήματος. Με τα ψηφιακά φίλτρα δεν υπάρχουν τέτοιοι περιορισμοί.
- Η απόδοση των ψηφιακών φίλτρων είναι σταθερή (repeatable) από σύστημα σε σύστημα.
- Τα ψηφιακά φίλτρα μπορούν να χρησιμοποιηθούν σε πολύ χαμηλές συχνότητες, όπου η χρήση αναλογικών φίλτρων είναι αδύνατη.

Είδη Ψηφιακών Φίλτρων

Τα ψηφιακά φίλτρα γενικός διαιρούνται σε δύο μεγάλες κατηγορίες, στα φίλτρα άπειρης κρουστικής απόκρισης IIR (Infinite Impulse Responce), και στα φίλτρα πεπερασμένης κρουστικής απόκρισης FIR (Finite Impulse Responce). Η είσοδος και η έξοδος του φίλτρου συσχετίζονται με τις παρακάτω εξισώσεις για τα IIR και FIR φίλτρα αντίστοιχα:

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k)$$

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

Είναι γεγονός όπως φαίνεται και από τις παραπάνω εξισώσεις ότι για τα IIR φίλτρα η κρουστική απόκριση έχει άπειρη κρουστική απόκριση, ενώ για τα FIR φίλτρα έχει πεπερασμένη διάρκεια γιατί η h(k) έχει μόνο N τιμές.. Για τα IIR χρησιμοποιούμαι κυρίως την ακόλουθη εξίσωση:

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k) = \sum_{k=0}^{N} a_k\, x(n-k) - \sum_{k=1}^{M} b_k\, y(n-k)$$

Όπου a_k και b_k οι συντελεστές του φίλτρου.

Οι παραπάνω εξισώσεις αποτελούν τις εξισώσεις διαφορών για τις δύο κατηγορίες ψηφιακών φίλτρων. Εναλλακτικές μορφές απεικόνισης είναι οι ακόλουθες:

$$H(z) = \sum_{k=0}^{N-1} h(k)\, z^{-k}$$ για FIR φίλτρα χωρίς ανάδραση

$$H(z) = \frac{\sum_{k=0}^{N} a_k z^{-k}}{\left(1 + \sum_{k=1}^{M} b_k z^{-k}\right)}$$ για IIR φίλτρα με ανάδραση

Όπως είναι φανερό η θεωρία των γραμμικών ψηφιακών φίλτρων βασίζεται στις γραμμικές εξισώσεις διαφορών με σταθερούς συντελεστές. Οι εξισώσεις αυτές λύνονται με την βοήθεια του μετασχηματισμού Z, ενώ η γραμμικές διαφορικές εξισώσεις που περιγράφουν τα αναλογικά φίλτρα λύνονται με τον μετασχηματισμό Laplace στο πεδίο S.

Επιλέγοντας μεταξύ FIR και IIR Φίλτρων

- Τα FIR φίλτρα μπορούν να έχουν εντελώς γραμμική απόκριση φάσης. Έτσι δεν έχουμε καθόλου παραμόρφωση φάσης στο σήμα εισόδου. Η απόκριση φάσης στα IIR φίλτρα είναι μη γραμμική κυρίως στις συχνότητες αποκοπής.
- Τα FIR φίλτρα υλοποιούνται χωρίς ανάδραση και γι' αυτό τον λόγο είναι πάντοτε σταθερά συστήματα. Η σταθερότητα των IIR συστημάτων λόγο της ανάδρασης δεν είναι πάντοτε γεγονός.
- Τα προβλήματα λόγο πεπερασμένου αριθμού απεικόνισης των ψηφιολέξεων (roundoff noise, coefficient quantization errors), είναι περισσότερο αισθητά στα IIR φίλτρα από ότι στα FIR.
- Τα FIR φίλτρα χρειάζονται μεγαλύτερο αριθμό συντελεστών (coefficients), για μεγάλη κλήση από ότι τα IIR. Έτσι για μία δεδομένη απόκριση περισσότερος χρόνος επεξεργασίας και χώρος αποθήκευσης απαιτείται από τα FIR φίλτρα.
- Τα αναλογικά φίλτρα μπορούν πολύ εύκολα να μετατραπούν σε αντίστοιχα IIR ψηφιακά φίλτρα τα οποία έχουν παρόμοιες χαραχτηριστικές. Αυτό όμως δεν είναι δυνατόν για τα FIR φίλτρα τα οποία δεν έχουν αντίστοιχες αναλογικές υλοποιήσεις. Πάντως με τα FIR φίλτρα είναι ευκολότερο να υλοποιήσει κανείς δεδομένες αποκρίσεις πλάτους οι οποίες δίνονται σε αριθμητική μορφή.
- Γενικώς για τον υπολογισμό και την σχεδίαση των FIR φίλτρων απαραίτητη είναι η χρησιμοποίηση συστημάτων CAE.

Από την παραπάνω λίστα βγαίνει σαν συμπέρασμα ότι:
Πρέπει να χρησιμοποιούμε IIR φίλτρα όταν θέλουμε μεγάλη κλήση στην ζώνη αποκοπής και μεγάλη ταχύτητα στον υπολογισμό του αλγορίθμου, μια και αυτά χρειάζονται μικρότερο αριθμό συντελεστών.
Πρέπει να χρησιμοποιούμαι FIR φίλτρα αν ο απαιτούμενος αριθμός συντελεστών για την δεδομένη απόκριση δεν είναι πολύ μεγάλος, και επίσης όταν χρειαζόμαστε γραμμική απόκριση φάσης.

Διαδικασία σχεδίασης Ψηφιακών Φίλτρων

Η σχεδίαση των ψηφιακών φίλτρων απαιτεί τα πέντε ακόλουθα βήματα:

- Καθορισμός των χαραχτηριστικών του φίλτρου.
- Υπολογισμός των απαιτούμενων συντελεστών.
- Απεικόνιση του φίλτρου σε επιθυμητή μορφή μπλοκ διαγράμματος (realization).
- Ανάλυση των φαινομένων πεπερασμένου μήκους ψηφιολέξεων στην απόδοση του φίλτρου.
- Υλοποίηση του φίλτρου με software ή hardware.

Από τα παραπάνω βήματα κρίνεται σκόπιμο να αναλυθούν περισσότερο τα δύο τελευταία βήματα:

Ανάλυση των φαινομένων πεπερασμένου μήκους ψηφιολέξεων

Οι κύριοι παράγοντες μείωσης της απόδοσης του φίλτρου λόγο πεπερασμένου μήκους ψηφιολέξεων είναι οι ακόλουθοι:

- *Δειγματοληψία Εισόδου/Εξόδου* .Από την διαδικασία δειγματοληψίας εισάγεται θόρυβος (quantization error) ο οποίος και θα αναλυθεί αργότερα.
- *Μείωση της ακρίβειας των συντελεστών*. Αυτό το φαινόμενο έχει σαν συνέπεια στην απόκριση της συχνότητας αποκοπής στα FIR και στα IIR φίλτρα, όπως και την αστάθεια των IIR συστημάτων.

- *Σφάλματα μειωμένης αριθμητικής/υπόλογιστικής ακρίβειας (Arithmetic roundoff errors).* Η χρησιμοποίηση πεπερασμένης αριθμητικής ακρίβειας για την πραγματοποίηση του ψηφιακού αλγορίθμου εκτός από σφάλματα στην ακρίβεια του φίλτρου μπορεί να οδηγήσει σε αστάθεια τα IIR φίλτρα.
- *Υπερφόρτωση (Overflow).*Αυτό το φαινόμενο συμβαίνει όταν ένα αποτέλεσμα μίας πράξης ξεπεράσει το εύρος bit του συστήματος οδηγεί σε παρόμοια με τα παραπάνω προβλήματα.

Υλοποίηση του φίλτρου

Όπως έχει προαναφερθεί για την υλοποίηση των αλγορίθμων ψηφιακής επεξεργασίας σήματος, κυρίως πραγματοποιούνται πράξεις πολλαπλασιασμού, πρόσθεσης/αφαίρεσης και καθυστέρησης. Έτσι για την υλοποίηση των φίλτρων χρειάζονται τα ακόλουθα στοιχεία:

- Μνήμη (για παράδειγμα ROM) για την αποθήκευση των συντελεστών του φίλτρου.
- Μνήμη (για παράδειγμα RAM) για την αποθήκευση των παρόντων και των προηγουμένων εισόδων και εξόδων του φίλτρου, δηλαδή των {x(n), x(n-1),.....} και {y(n), y(n-1),.....} .
- Πολλαπλασιαστές σε Hardware ή Software μορφή.
- Αθροιστές ή Αριθμητική Λογική Μονάδα ALU.

Σε συνθήκες λειτουργίας πραγματικού χρόνου το φίλτρο απαιτείται να παράγει το σήμα εξόδου πριν την άφιξη του επόμενου σήματος εισόδου. Ειδικότερα ακολουθείται η παρακάτω φιλοσοφία. Οι παράγοντες που καθορίζουν την πολυπλοκότητα του συστήματος που υλοποιεί ένα ψηφιακό φίλτρο είναι:

- Ο αριθμός των πολλαπλασιασμών που πρέπει να γίνουν ανά δευτερόλεπτο.
- Ο αριθμός των αθροίσεων ανά δευτερόλεπτο.
- Το μέγεθος της μνήμης που απαιτείται.
- Το μέγεθος των κυκλωμάτων που απαιτούνται για την μονάδα ελέγχου του ψηφιακού συστήματος (control unit).

Συγκρίνοντας τους δύο πρώτους παράγοντες συμπεράνουμε ότι σημαντικότερος είναι ο αριθμός των αθροιστών γιατί για μία είσοδο Β αριθμού bit χρειάζονται κατά εκτίμηση Β πολλαπλασιαστές και Β^2 αθροιστές.

5.2 ΥΛΟΠΟΙΗΣΗ FIR ΣΥΣΤΗΜΑΤΩΝ

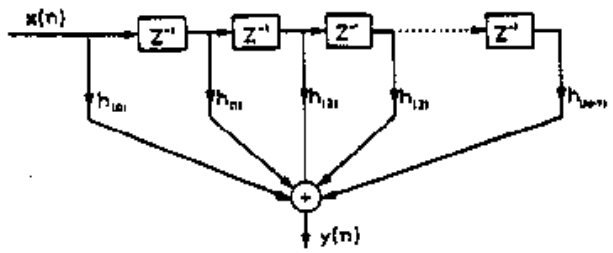
Στα FIR συστήματα η διακριτή κρουστική απόκριση είναι πεπερασμένης διάρκειας και η συνάρτηση H(Z) γράφεται συνήθως με τη μορφή:

$$H(z) = \sum_{n=0}^{N-1} h(n)\cdot z^{-n}$$

Η συνάρτηση h(n) αποτελείται από Ν δείγματα με αποτέλεσμα η H(Z) να είναι πολυώνυμο N-1 τάξης ως προς Z^-1. Δηλαδή η H(Z) έχει N-1 πόλους στο Z=0 και N-1 μηδέν. Τα FIR συστήματα χαρακτηρίζονται από μη αναδρομικές εξισώσεις διαφοράς που είναι της μορφής:

$$y(n) = \sum_{k=0}^{N-1} h(x)\cdot x(n-k)$$

Στο παρακάτω σχήμα απεικονίζεται η υλοποίηση της τελευταίας σχέσης με την ευθεία μορφή:



Στη μορφή καταρράκτη η H(Z) γράφεται υπό μορφή γινομένου παραγόντων πρώτης και δεύτερης τάξης:

$$H(z) = \prod_{n=1}^{k} H_n(z)$$

όπου οι παράγοντες πρώτης τάξης είναι:

$$H_n(z) = a_{0n} + a_{1n} \cdot z^{-1}$$

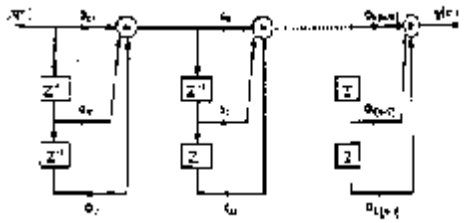
και οι παράγοντες δεύτερης τάξης είναι:

$$H_n(z) = a_{0n} + a_{1n} \cdot z^{-1} + a_{2n} \cdot z^{-2}$$

Η παρακάτω σχέση ισοδυναμεί με την αρχική σχέση της παραγράφου όπου αν N είναι άρτιος αριθμός, ένας από τους συντελεστές α2n θα είναι μηδέν.

$$H(z) = \left[\prod_{n=1}^{\frac{N}{2}} a_{0n} + a_{1n} \cdot z^{-1} + a_{2n} \cdot z^{-2} \right]$$

Στο ακόλουθο σχήμα πραγματοποιείται η υλοποίηση της μορφής καταρράκτη όπου κάθε υποσύστημα είναι σε ευθεία μορφή:



5.3 Υλοποίηση του FIR φίλτρου με λογισμικό

Με αυτήν την υλοποίηση χρησιμοποιείτε μία συνάρτηση filterloop, η οποία αναλαμβάνει να φιλτράρει τα ηχητικά δεδομένα και να τα αποθηκεύσει σε ένα προσωρινό πίνακα ώστε να αναπαραχθούν στη συνέχεια από τον Nios II. Ο κώδικας αυτής της συνάρτησης είναι ο ακόλουθος:

```
void  filterfir_float_sound(unsigned int *input_left, unsigned int *input_right, int  left_samples ,int right_samples)
 {

// fir basic implemntation
float  yn_left_float=0;
float  yn_right=0;
int tl=0,rl=0,bl=0;
int tr=0,rr=0,br=0;
// Filter Left Channel samples
for (bl=0; bl<left_samples; bl++)
  {
    yn_left_float=0;
    dly_left[0] = input_left[bl];
for (tl=0; tl<N; tl++) {
//implementation of FIR filter
       yn_left_float += ( h_float[tl] * ((float  ) dly_left[tl]) ); }   //calculating Sum of Products
            out_buff_array_EQ_LEFT[bl] = (yn_left_float);

for (rl = left_samples-1; rl > 0; rl--)
     {dly_left[rl] = dly_left[rl-1];}
  }

// Filter Right Channel samples
for (br=0; br<right_samples; br++)
  {
    yn_right=0;
    dly_right[0] = input_right[br];
for (tr=0; tr<N; tr++) {
//implementation of FIR filter
    yn_right += ( h_float[tr] * (float ) dly_right[tr]);  }  //calculating Sum of Products
    out_buff_array_EQ_RIGHT[br] = (yn_right);
for (rr = right_samples-1; rr > 0; rr--) {
     dly_right[rr] = dly_right[rr-1]; }
  }
}
```

Οι συντελεστές της κρουστικής απόκρισης διευθετούνται από έναν buffer (πίνακα) έτσι ώστε ο πρώτος συντελεστής *h(0)*, να αποτελεί την χαμηλότερη διεύθυνση μνήμης και ο τελευταίος συντελεστής *h(N-1)*, την υψηλότερη διεύθυνση μνήμης. Τα καθυστερημένα δείγματα οργανώνονται στη μνήμη, έτσι ώστε το νεότερο δείγμα *x(n)*, να είναι στο ξεκίνημα των δειγμάτων του buffer, ενώ το παλαιότερο δείγμα, *x(n-(N-1))*, να είναι στο τέλος του buffer.Οι συντελεστές και τα δείγματα μπορούν να διευθετηθούν στην μνήμη όπως φαίνεται στον πίνακα. Αρχικά όλα τα δείγματα ορίζονται μηδέν.

i	Συντελεστές	Δείγματα
0	*h(0)*	*x(n)*
1	*h(1)*	*x(n-1)*
2	*h(2)*	*x(n-2)*
⋮	⋮	⋮
N-1	*h(N-1)*	*x(n-(N-1))*

Χρόνος *n*.
Το νεότερο δείγμα, *x(n)*, την στιγμή *n* προκύπτει από έναν ADC και αποθηκεύεται στην αρχή του buffer. Η έξοδος του φίλτρου την χρονική στιγμή *n* υπολογίζεται από την εξίσωση της συνέλιξης ή διαφορετικά:

$$y(n)=h(0)x(n)+h(1)x(n-1)+\cdots+h(N-2)x\left(n-(N-2)\right)+h(N-1)x\left(n-(N-1)\right)$$

Τα καθυστερημένα δείγματα ενημερώνονται, έτσι ώστε το *x(n-k)=x(n+1-k)* για να μπορεί να χρησιμοποιηθεί στον υπολογισμό του *y(n+1)*, της εξόδου την επόμενη χρονική στιγμή ή αλλιώς στην επόμενη περίοδο T_s. Όλα τα δείγματα ενημερώνονται εκτός από το νεότερο δείγμα. Για παράδειγμα, *x(n-1)=x(n)*, και *x(n-(N-1))=x(n-(N-2))*. Αυτή η προοδευτική ενημέρωση έχει ως αποτέλεσμα την μετακίνηση των δεδομένων (κάτω) στην μνήμη.

Χρόνος *n+1*.
Σε χρόνο *n+1*, ένα νέο δείγμα προκύπτει ως είσοδος *x(n+1)* και αποθηκεύεται στην κορυφή του buffer, όπως φαίνεται στον πίνακα 4.χχ. Η έξοδος *y(n+1)*,μπορεί τώρα να υπολογιστεί ως:

$$y(n+1)=h(0)x(n+1)+h(1)x(n)+\cdots+h(N-2)x\left(n-(N-3)\right)+h(N-1)x\left(n-(N-2)\right)$$

Τα δείγματα τότε ενημερώνονται για την επόμενη μονάδα του χρόνου.

Συντελεστές	Δείγματα		
	Time n	*Time n+1*	*Time n+2*
h(0)	*x(n)*	*x(n+1)*	*x(n+2)*
h(1)	*x(n-1)*	*x(n)*	*x(n+1)*
h(2)	*x(n-2)*	*x(n-1)*	*x(n)*
⋮	⋮	⋮	⋮
h(N-3)	*x(n-(N-3))*	*x(n-(N-4))*	*x(n-(N-5))*
h(N-2)	*x(n-(N-2))*	*x(n-(N-3))*	*x(n-(N-4))*
h(N-1)	*x(n-(N-1))*	*x(n-(N-2))*	*x(n-(N-3))*

Χρόνος *n+2*.
Σε χρόνο *n+2* ,ένα νέο δείγμα εισόδου *x(n+2)*, προκύπτει. Η έξοδος γίνεται:

$$y(n+2)=h(0)x(n+2)+h(1)x(n+1)+\cdots+h(N-1)x\left(n-(N-3)\right)$$

Αυτή η διαδικασία συνεχίζεται για να υπολογιστεί η έξοδος του φίλτρου και να ενημερωθούν τα καθυστερημένα δείγματα σε κάθε μονάδα χρόνου (περίοδος δειγματοληψίας).
Στην παραπάνω υλοποίηση μπορούν να γίνουν καποιες τροποποιήσης όπως η επίλογή ακέραιων συντελεστών και όχι με αριθμητική κινητής υποδιαστολής, επιλογή που μειώνει τον χρόνο εκτέλεσης του αλγόριθμου. Επίσης έχει δοθεί στον compiler οδηγία -mhw-mul και -mhw-mulx για να

χρησιμοποιήσουμε τα κυκλώματα πολλαπλασιαστών που έχουμε επιλέξει να περιέχει ο επεξεργαστής NIOS II κατά την υλοποίηση του.
Ο παρακάτω κώδικας είναι η υλοποίηση με συντελεστές ακέραιους αριθμούς.

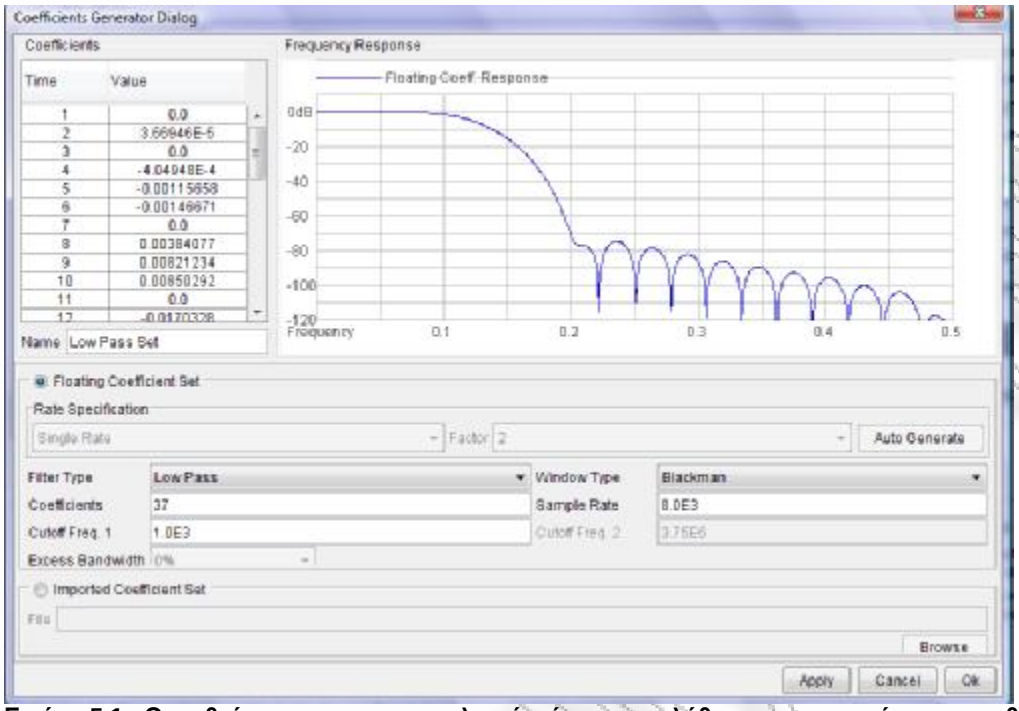
```
void  filterloop_float_sound(unsigned int *input_left, unsigned int *input_right, int  left_samples ,int right_samples)
 {

// fir basic implemntation
alt_16 temp;
alt_64 yn_left_int=0;
 alt_64 yn_right_int=0;
float  yn_right=0;
int tl=0,rl=0,bl=0;
int tr=0,rr=0,br=0;
// Filter Left Channel samples
for (bl=0; bl<left_samples; bl++)
  {
    yn_left_float=0;
    yn_left_int=0;
    dly_left[0] = input_left[bl];
for (tl=0; tl<N; tl++) {
//implementation of FIR filter
      temp=h_int[tl]+128;
      yn_left_int += ( temp * ((float  ) dly_left[tl]) ); }   //calculating Sum of Products
out_buff_array_EQ_LEFT[bl] = (yn_left_int);

for (rl = left_samples-1; rl > 0; rl--)
     {dly_left[rl] = dly_left[rl-1];}
  }

// Filter Right Channel samples
for (br=0; br<right_samples; br++)
  {
    yn_right=0;
    dly_right[0] = input_right[br];
for (tr=0; tr<N; tr++) {
//implementation of FIR filter
temp=h_int[tl]+128;
      yn_right_int += (temp * (float ) dly_right[tr]);  }  //calculating Sum of Products

    out_buff_array_EQ_RIGHT[br] = (yn_right_int);
for (rr = right_samples-1; rr > 0; rr--) {
      dly_right[rr] = dly_right[rr-1]; }
  }
}
```

5.4 **Υλοποίηση του** FIR **φίλτρου με χρήση εξωτερικού κυκλώματος επιταχυντή** (Hardware Accelerator)

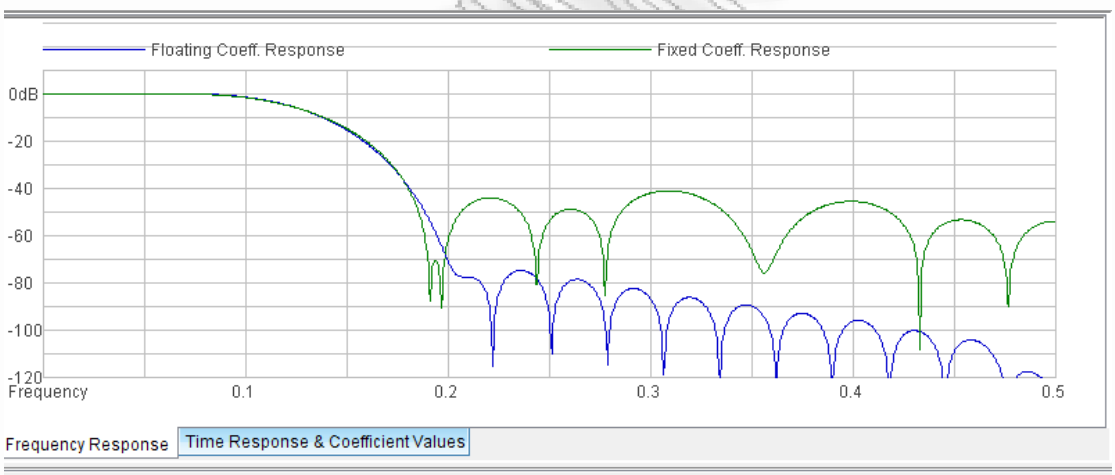
Για το σκοπό αυτό, χρησιμοποιήθηκε ο FIR Compiler της ALTERA [30], ο οποίος υλοποιεί ένα FIR φίλτρο (MegaCore Function). Το φίλτρο που υλοποιήθηκε για το συγκεκριμένο Nios II σύστημα, είναι ένα βαθυπερατό φίλτρο με συχνότητα δειγματοληψίας 8 KHz και συχνότητα αποκοπής 1 KHz. Το βαθυπερατό φίλτρο, ορίζεται εκτός από τη συχνότητα αποκοπής του και από τους συντελεστές του. Ο FIR Compiler παρέχει τη δυνατότητα παραγωγής των συντελεστών αυτών. Οι ρυθμίσεις για τους συντελεστές, όπως το πλήθος τους και ο τύπος παραθύρου του FIR φίλτρου, παρουσιάζονται στην εικόνα 5.1. Συγκεντρωτικά, οι ρυθμίσεις έχουν ως εξής:

- Floating Coefficient Set
- Filter Type: Low Pass
- Coefficients: 37
- Cutoff Freq.1: 1000
- Window Type: Rectangular
- Sample Rate: 8000

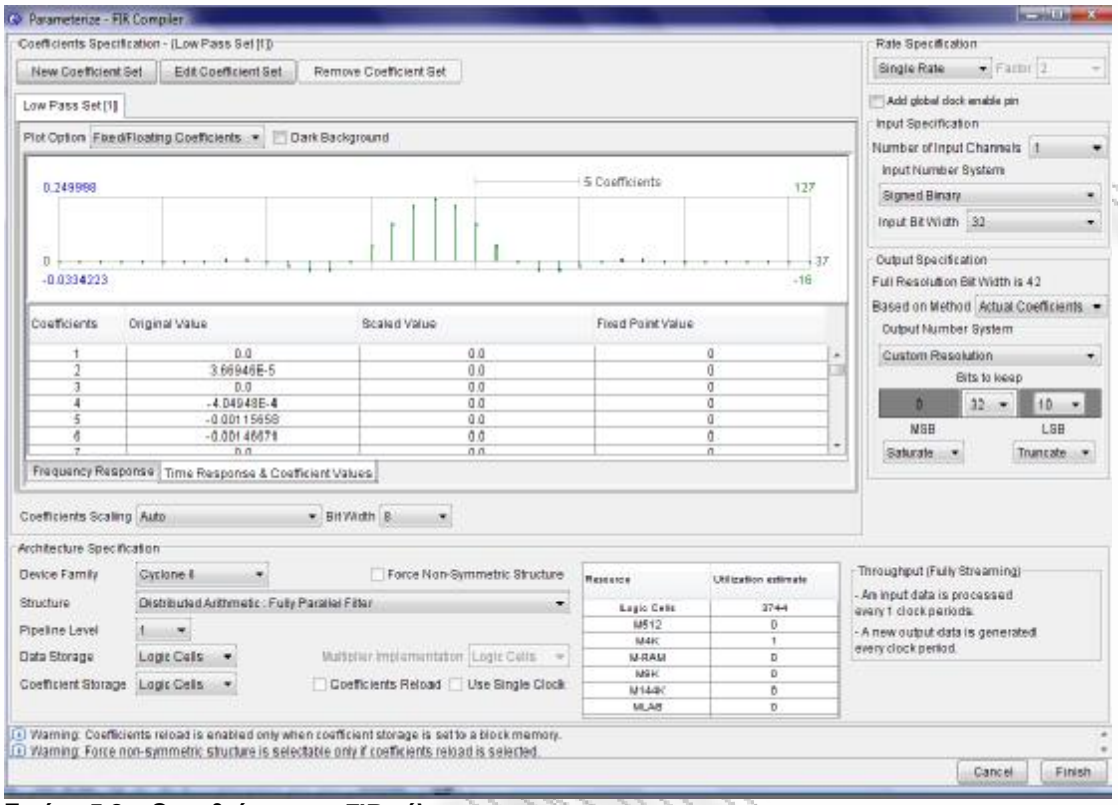
Οι ρυθμίσεις του FIR φίλτρου παρουσιάζονται στην εικόνα 5.3, όπου φαίνεται και η απόκριση συχνότητας του φίλτρου. Αναλυτικότερα, ο αριθμός των καναλιών εισόδου τίθεται στο “1”, τα δεδομένα στην είσοδο του FIR φίλτρου ορίζονται να είναι προσημασμένα “Signed Binary” με εύρος 32-bit. Το εύρος των δεδομένων στην έξοδο του FIR φίλτρου, εάν χρησιμοποιηθεί η επιλογή “Full Resolution”, τίθεται αυτόματα από τον FIR Compiler στα 42-bit. Επειδή όμως το Nios II σύστημα δουλεύει με δεδομένα των 32-bit επιλέγεται η ρύθμιση “Custom Resolution” σύμφωνα με την οποία από τα 42 bits χρησιμοποιούνται τα 32. τα δέκα λιγότερο σημαντικά ψηφία (Least Significant Bits - LSB) επιλέγονται να αποκοπούν (Truncate). Στη ρύθμιση “Saturate” εάν η φιλτραρισμένη έξοδος είναι μεγαλύτερη από τη μέγιστη θετική ή τη μέγιστη αρνητική τιμή η οποία μπορεί να αναπαρασταθεί, τότε η έξοδος τίθεται ίση με τη μέγιστη αυτή τιμή, ενώ στη ρύθμιση “Truncate” το φίλτρο απορρίπτει τα συγκεκριμένα bits. Στην επιλογή Coefficients Scaling επιλέγεται η ρύθμιση “Auto” σύμφωνα με την οποία παρέχεται το μέγιστο SNR, ενώ η ρύθμιση Coefficients Width Bit τίθεται στον αριθμό “8” όπως υπολογίζεται από τον FIR Compiler. Η ρύθμιση “Device Family” τίθεται στην επιλογή CycloneII και επιλέγεται η δομή υλοποίησης του FIR φίλτρου: “Distributed Arithmetic: Fully Parallel Filter”. Η δομή αυτή αν και καταλαμβάνει τη μεγαλύτερη επιφάνεια, παράγει ένα πολύ γρήγορο φίλτρο αφού υπολογίζει την έξοδο του φίλτρου σε έναν απλό κύκλο ρολογιού. Το σχηματικό διάγραμμα της δομής αυτής παρουσιάζεται στην εικόνα 5.5. Στο πεδίο Pipeline Level επιλέγεται η ρύθμιση “1” όπου υλοποιείται ένα μόνο επίπεδο Pipeline. Στο πεδίο Data Storage επιλέγεται η ρύθμιση “LogicCells”, κατά την οποία η αποθήκευση των δεδομένων πραγματοποιείται στα λογικά κελιά του FPGA. Στο πεδίο Coefficients Storage επιλέγεται η ρύθμιση “Logic Cells”, κατά την οποία η αποθήκευση των συντελεστών πραγματοποιείται στα λογικά κελιά του FPGA.



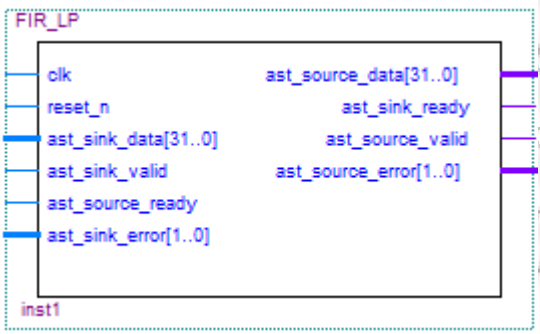
Εικόνα 5.1 - Οι ρυθμίσεις για τους συντελεστές, όπως το πλήθος τους και ο τύπος παραθύρου του FIR φίλτρου



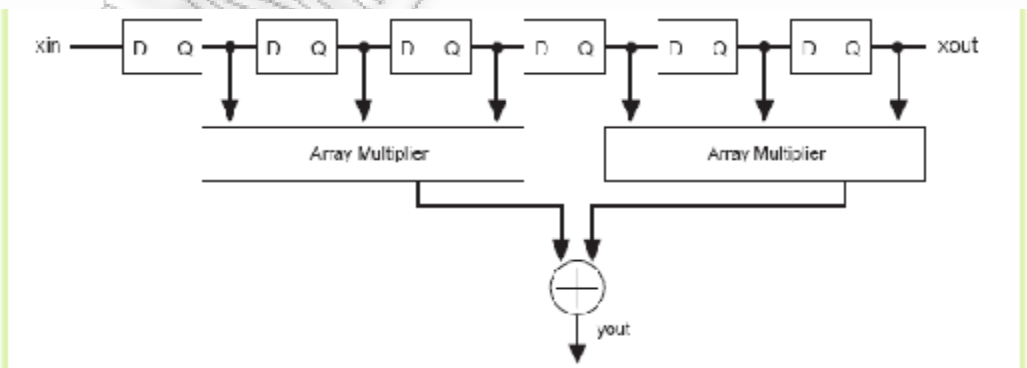
Εικόνα 5.2 – Η απόκριση συχνότητας με βάση τους συντελεστές



Εικόνα 5.3 – Οι ρυθμίσεις του FIR φίλτρου



Εικόνα 5.4 – Το μπλοκ του παραγόμενου FIR φίλτρου



Εικόνα 5.5 – Το δομικό διάγραμμα του FIR φίλτρου

Στις εικόνες 5.1 και 5.2 παρουσιάζεται η απόκριση χρόνου του φίλτρου και οι τιμές των συντελεστών.

Το υποκύκλωμα του FIR φίλτρου φαίνεται στην εικόνα 5.4 όπως αυτό παράχθηκε από τον FIR Compiler. Τα σήματα εισόδου του είναι τα ακόλουθα:

- clk – το σήμα αυτό είναι το σήμα ρολογιού που συγχρονίζει όλους τους εσωτερικούς καταχωρητές του FIR φίλτρου.
- reset_n – το σήμα αυτό επαναφέρει το κύκλωμα ελέγχου του FIR φίλτρου κατά τη θετική ακμή του clk και πρέπει να διαρκεί πάνω από ένα κύκλο ρολογιού.
- ast_sink_valid – το σήμα αυτό τίθεται στο λογικό 1 όταν τα δεδομένα εισόδου είναι έγκυρα. Εάν το σήμα αυτό δεν έχει την τιμή 1 το φίλτρο σταματάει την επεξεργασία του εάν απαιτούνται νέα δεδομένα και δεν έχουν μείνει δεδομένα στην Avalon-ST input FIFO.
- ast_sink_data – το σήμα αυτό αναπαριστά τα δεδομένα εισόδου.
- ast_source_ready – τίθεται στο λογικό 1 όταν ο Nios II είναι έτοιμος να λάβει δεδομένα.
- ast_sink_error – το σήμα αυτό υποδεικνύει κάποιο σφάλμα κατά τη μεταφορά δεδομένων από τη μεριά του επεξεργαστή. Όταν έχει την τιμή 00 δεν υπάρχει σφάλμα.

Τα σήματα εξόδου του είναι τα ακόλουθα:

- ast_sink_ready – το σήμα αυτό τίθεται στο λογικό 1 από το FIR φίλτρο, όταν είναι έτοιμο να λάβει δεδομένα στον τρέχων κύκλο ρολογιού.
- ast_source_valid – το σήμα αυτό τίθεται στο λογικό 1 από το FIR φίλτρο, όταν υπάρχουν έγκυρα δεδομένα στην έξοδό του.
- ast_source_data – το σήμα αυτό αναπαριστά τα δεδομένα εξόδου του φίλτρου, το εύρος των οποίων καθορίζεται από τις ρυθμίσεις του FIR Compiler.
- ast_source_error – το σήμα αυτό υποδεικνύει κάποιο σφάλμα κατά τη μεταφορά δεδομένων από τη μεριά του φίλτρου. Όταν έχει την τιμή 00 δεν υπάρχει σφάλμα.

Όπως έχει ήδη αναφερθεί σε προηγούμενες παραγράφους, ο Nios II ελέγχει τις μεταφορές δεδομένων από και προς το FIR φίλτρο μέσω συγκεκριμένων PIO Cores και μέσω των σημάτων ast_sink_valid (για αποστολή δεδομένων στο FIR) και ast_source_ready (για λήψη δεδομένων από το FIR). Η διαδικασία αυτή έχει ως εξής:

i. Θα πρέπει να γράφεται ένα δεδομένο στο ast_sink_data και στη συνέχεια να δημιουργείται το σήμα ast_sink_valid που να έχει διάρκεια ενός κύκλου ρολογιού έτσι ώστε να διαβάζεται το αντίστοιχο δεδομένο από το FIR. Εάν το σήμα αυτό έχει την τιμή 1 για περισσότερους από έναν κύκλους ρολογιού, τότε θα εγγραφούν τα ίδια δεδομένα πολλές φορές και το FIR δε θα δίνει τα σωστά αποτελέσματα.
ii. Κάθε φορά που το FIR φίλτρο πρέπει να διαβάσει ένα δεδομένο δημιουργείται ένας παλμός διάρκειας ενός κύκλου ρολογιού στο σήμα ast_source_ready.

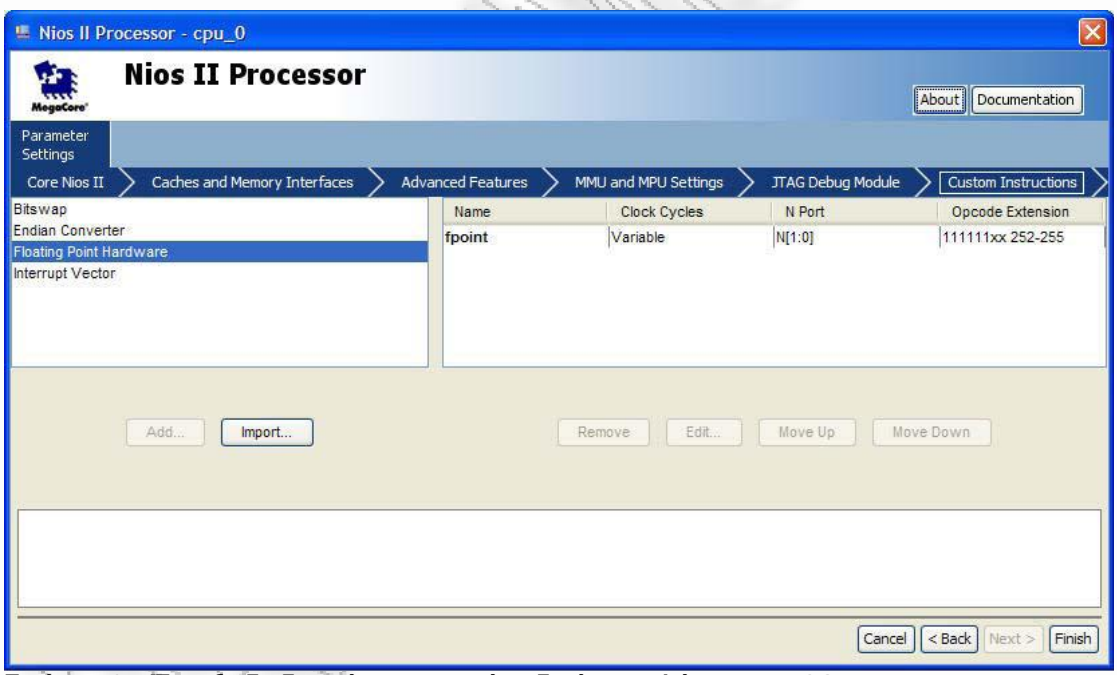
Για το σκοπό αυτό, πρέπει ανάμεσα στις εξόδους των PIO Cores για τα σήματα ast_sink_valid και ast_source_ready και στις αντίστοιχες εισόδους του FIR να προστεθεί ένα κύκλωμα (one_pulse). Σύμφωνα με το κύκλωμα αυτό, όταν εγγράφεται η τιμή 1 από το πρόγραμμα (π.χ. στο σήμα ast_sink_valid), το σήμα αυτό θα μπορεί να παραμένει για περισσότερους από έναν κύκλους ρολογιού στην τιμή 1, αλλά στο FIR θα καταλήγει μόνο ένας παλμός.

5.5 Υλοποίηση του FIR φίλτρου με χρήση ειδικών εντολών (Custom Instruction) του NIOS II

Για την υλοποίηση του FIR φίλτρου με χρήση ειδικών εντολών έχουν ακολουθηθεί δύο επιλογές. Η πρώτη είναι η χρησιμοποίηση έτοιμων ειδικών εντολών, που μας παρέχει η ALTERA, που επεκτείνουν το ρεπερτόριο εντολών του επεξεργαστή για εκτέλεση αριθμητικών πράξεων (πρόσθεσης, αφαίρεσης, πολλαπλασιασμού, διαίρεσης) κινητής υποδιαστολής. Για την χρησιμοποίηση αυτών τον εντολών από τον Compiler πρέπει να του δοθεί η ακόλουθη οδηγία -mcustom-fpu-cfg=60-2. Για την επιλογή ή όχι αυτών τον εντολών από τον κώδικα υπάρχουν οι παρακάτω οδηγίες:

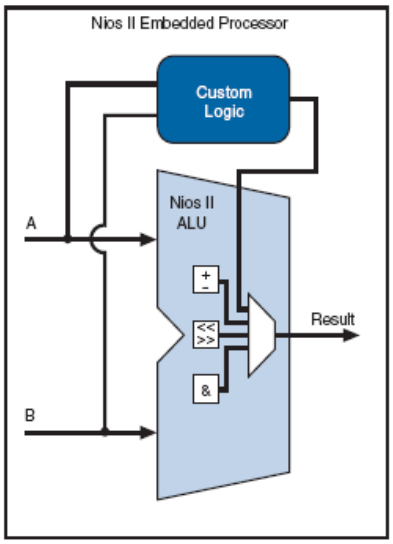
- **#pragma no_custom_fadds** – Μη επιλογή υλοποίησης πρόσθεσης κινητής υποδιαστολής με χρήση των ειδικών εντολών.
- **#pragma no_custom_fsubs** – Μη επιλογή υλοποίησης αφαίρεσης κινητής υποδιαστολής με χρήση των ειδικών εντολών.
- **#pragma no_custom_fmuls** – Μη επιλογή υλοποίησης πολλαπλασιασμού κινητής υποδιαστολής με χρήση των ειδικών εντολών.
- **#pragma no_custom_fdivs** – Μη επιλογή υλοποίησης διαίρεσης κινητής υποδιαστολής με χρήση των ειδικών εντολών.

Για την προσθήκη αυτών των ειδικών εντολών ακολουθούμε την παρακάτω διαδικασία:
Μέσα από το περιβάλλον του SOPC Builder, ανοίγουμε την παραμετροποίηση του επεξεργαστή NIOS II και επιλέγουμε την σελίδα Custom Instructions, εκεί επιλέγουμε το πακέτο ειδικών εντολών **Floating Point Hardware** [8], [62] και κάνουμε add για να εισαχθούν αυτόματα στο ρεπερτόριο εντολών του επεξεργαστή. Επαναδημιουργόντας όλο το project και εισάγοντας το στο NIOS IDE, στο αρχείο system.h έχουν εισαγχεί οι περιγραφές των νέων ειδικών εντολών και είναι έτοιμες να χρησιμοποιηθούν από εμάς σε μορφή γλώσσας προγραμματισμού υψηλού επιπέδου.



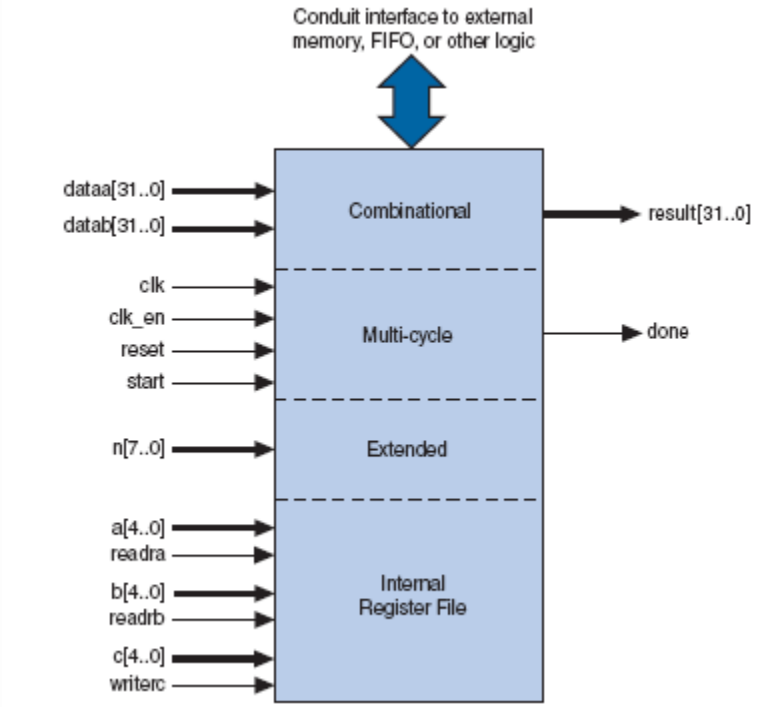
Εικόνα 5.6 – Έναρξη διαδικασίας εισαγωγής ειδικών εντολών στον NIOS II

Η δεύτερη περίπτωση είναι να εισάγουμε στον επεξεργαστή ειδικές εντολές [7] πού εμείς οι ίδιοι έχουμε αναπτύξει, έτσι ώστε να επιταχυνθεί η εκτέλεση κρίσιμων όσο αναφορά τον χρόνο εκτέλεσης τμημάτων κώδικα. Χρησιμοποιώντας ειδικές εντολές μειώνουμε μία σύνθετη ακολουθία εντολών σε μία μόνο εντολή που υλοποιείτε από υλικό. Μπορούμε να χρησιμοποιήσουμε αυτήν την δυνατότητα σε μεγάλη γκάμα εφαρμογών, όπως για την βελτιστοποίηση λογισμικού που περιέχει πολλαπλές εσωτερικές επαναλήψεις (inner loops) για ψηφιακή επεξεργασία σήματος. Με ένα ειδικό περιβάλλον που μας παρέχεται από τον SOPC Builder μπορούμε να εισάγουμε μέχρι 256 ειδικές εντολές.
Το λογικό κύκλωμα (Custom Logic) των ειδικών εντολών συνδέεται απευθείας στην αριθμητική λογική μονάδα του NIOS II όπως φαίνεται παρακάτω



Εικόνα 5.7 - Το λογικό κύκλωμα (Custom Logic) των ειδικών εντολών

Το μπλοκ διάγραμμα του υλικού μιας ειδικής εντολής φαίνεται παρακάτω



Εικόνα 5.8 - Το μπλοκ διάγραμμα του υλικού μιας ειδικής εντολής

Η βασική λειτουργία του υλικού μίας ειδικής εντολής του NIOS II είναι να δέχεται είσοδο από τις θύρες dataa και datab και να δίνει το αποτέλεσμα στην θύρα result. Το υλικό της ειδικής εντολής μας δίνει αποτέλεσμα που βασίζεται στις εισόδους που παρέχονται από τον επεξεργαστή. Ο NIOS II υποστηρίζει διαφορετικούς τύπους ειδικών εντολών. Στην παραπάνω εικόνα φαίνονται όλες οι επιπλέον θύρες που απαιτούνται για την υλοποίηση όλων των διαφορετικών τύπων ειδικών εντολών. Επίσης μας παρέχεται η δυνατότητα για την υλοποίηση των ειδικών εντολών να έχουμε διασύνδεση με εξωτερικό υλικό εκτός του επεξεργαστή π.χ. μνήμη, FIFO.
Για κάθε μία ειδική εντολή το περιβάλλον ανάπτυξης λογισμικού NIOS IDE δημιουργεί macro στο αρχείο System.h έτσι μπορούμε να ενσωματώσουμε στον κώδικά μας εύκολα τις νέες ειδικές εντολές. Παράλληλα υπάρχει η δυνατότητα να τις χρησιμοποιήσουμε και σε γλώσσα assembly.

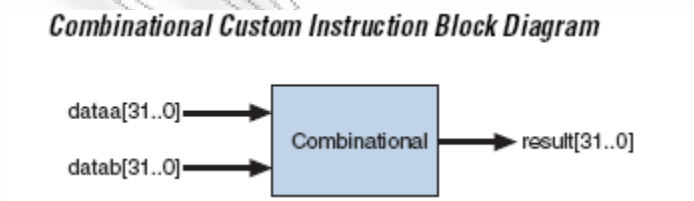
Υπάρχουν πολλοί τύποι ειδικών εντολών για να ικανοποιήσουν τις απαιτήσεις της εφαρμογής μας. Ο τύπος που επιλέγουμε καθορίζει και και την διασύνδεση του υλικού της ειδικής εντολής. Ο παρακάτω πίνακας μας δείχνει τους διαθέσιμους τύπους ειδικών εντολών και τις θύρες του υλικού που αντιστοιχούν σε κάθε τύπο.

Custom Instruction Types, Application and Hardware Ports		
Type	**Application**	**Hardware Ports**
Combinational	Single clock cycle custom logic blocks	• dataa[31..0] • datab[31..0] • result[31..0]
Multi-cycle	Multi clock cycle custom logic block of fixed or variable durations	• dataa[31..0] • datab[31..0] • result[31..0] • clk • clk_en*(1)* • start • reset • done
Extended	Custom logic blocks that are capable of performing multiple operations	• dataa[31..0] • datab[31..0] • result[31..0] • clk • clk_en*(1)* • start • reset • done • n[7..0]

Custom Instruction Types, Application and Hardware Ports		
Type	**Application**	**Hardware Ports**
Internal Register File	Custom logic blocks that access internal register files for input and/or output	• dataa[31..0] • datab[31..0] • result[31..0] • clk • clk_en • start • reset • done • n[7..0] • a[4..0] • readra • b[4..0] • readrb • c[4..0] • writerc
External Interface	Custom logic blocks that interface to logic outside of the Nios II processor's data path	Standard custom instruction ports, plus user-defined interface to external logic.

Πίνακας 5.1 - οι διαθέσιμοι τύποι ειδικών εντολών και οι θύρες του υλικού που αντιστοιχούν σε κάθε τύπο

Συνδυαστικές (Combinational) ειδικές εντολές
Οι συνδυαστικού τύπου ειδικές εντολές αποτελούνται από ένα μπλοκ λογικών κυκλωμάτων που είναι ικανό να ολοκληρώνει την λειτουργία του σε ένα κύκλο του ρολογιού.

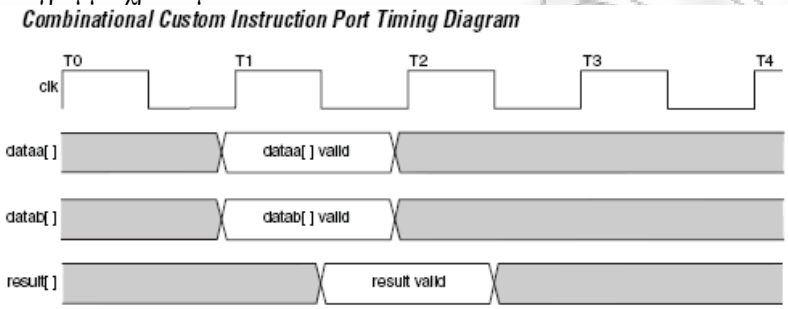


Εικόνα 5.9 - Συνδυαστικές (Combinational) ειδικές εντολές

Στις συνδυαστικού τύπου ειδικές εντολές χρησιμοποιούνται οι θύρες dataa και datab σαν είσοδοι και το αποτέλεσμα εμφανίζεται στην θύρα result. Επειδή η λογική υλοποιήτε σε έναν κύκλο ρολογιού θύρες ελέγχου δεν χρειάζονται.

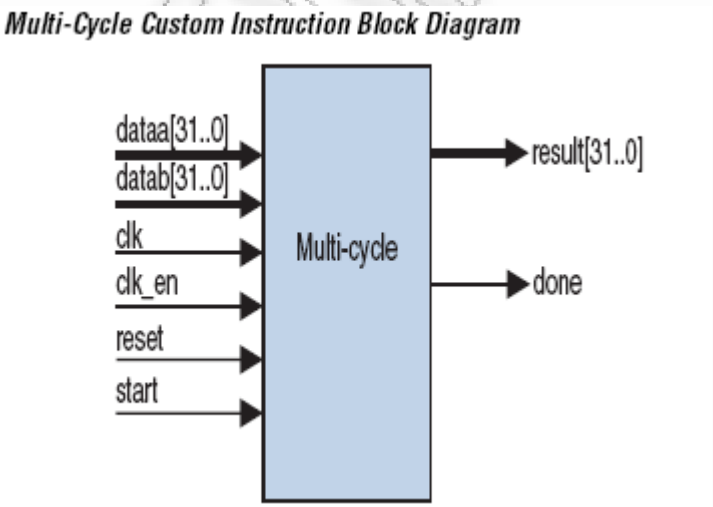
Combinational Custom Instruction Ports			
Port Name	**Direction**	**Required**	**Application**
dataa[31..0]	Input	No	Input operand to custom instruction
datab[31..0]	Input	No	Input operand to custom instruction
result[31..0]	Output	Yes	Result from custom instruction

Η μόνη απαραίτητη θύρα είναι αυτή που μας δίνει το αποτέλεσμα η result.Οι θύρες dataa και datab είναι προαιρετικές και χρησιμοποιούνται μόνο αν η ειδική εντολή έχει ορίσματα εισόδου.
Το διάγραμμα χρονισμού του συνδυαστικού τύπου ειδικών εντολών είναι το ακόλουθο:



Εικόνα 5.10 – Χρονισμός Συνδυαστικών (Combinational) ειδικών εντολών

Πολλαπλών – Κύκλων ειδικές εντολές
Πολλαπλών – Κύκλων ή ακολουθιακές ειδικές εντολές αποτελούνται από ένα μπλοκ λογικής που χρειάζεται δύο ή περισσότερους κύκλους ρολογιού για να ολοκλήρωση την λειτουργία του. Πρόσθετες θύρες ελέγχου χρειάζονται για την υλοποίηση τέτοιου τύπου εντολών.



Εικόνα 5.11 - Πολλαπλών – Κύκλων ειδικές εντολές

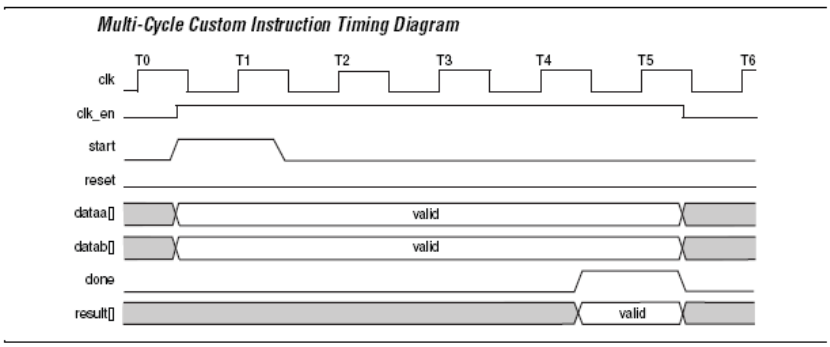
Οι πολλαπλών κύκλων ειδικές εντολές ολοκληρώνουν την λειτουργία τους σε σταθερό ή μεταβαλλόμενο αριθμό κύκλων ρολογιού.

- Σταθερού κύκλου: Καθορίζουμε τον αριθμό των κύκλων κατά την φάση δημιουργίας της ειδικής εντολής
- Μεταβαλλόμενου κύκλου: Οι θύρες start και done χρησιμοποιούνται για έχουμε ένα είδος χειραψίας με τον επεξεργαστή που ορίζει πότε μια εντολή έχει ολοκληρώσει την λειτουργία της.

Οι θύρες μιας πολλαπλού κύκλου ειδικής εντολής είναι:

Multi-Cycle Custom Instruction Ports			
Port Name	**Direction**	**Required**	**Application**
clk	Input	Yes	System clock
clk_en	Input	Yes	Clock enable
reset	Input	Yes	Synchronous reset
start	Input	No	Commands custom instruction logic to start execution
done	Output	No	Custom instruction logic indicates to the processor that execution is complete.
dataa[31..0]	Input	No	Input operand to custom instruction
datab[31..0]	Input	No	Input operand to custom instruction
result[31..0]	Output	No	Result from custom instruction

Το διάγραμμα χρονισμού του πολλαπλού κύκλου ειδικών εντολών είναι το ακόλουθο:

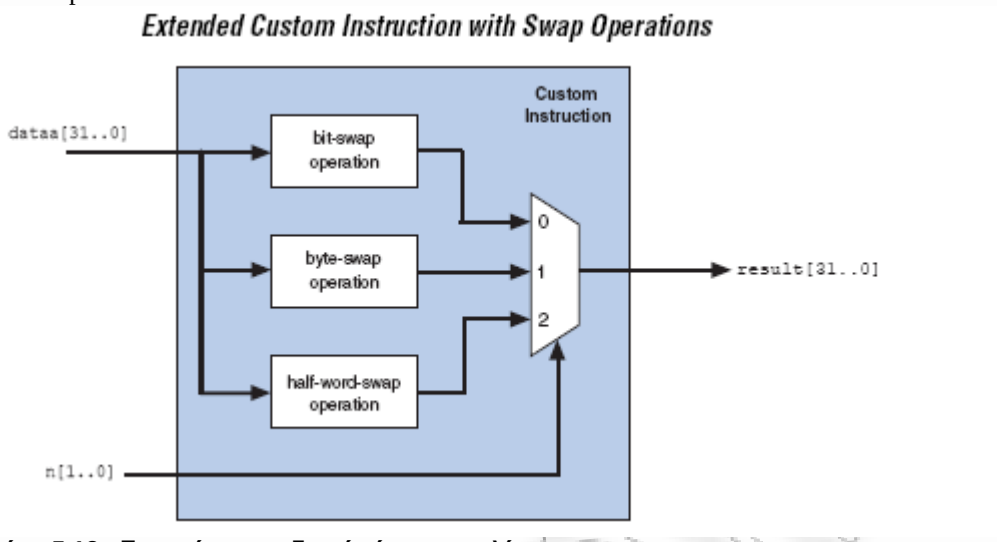


Εικόνα 5.12 - Το διάγραμμα χρονισμού του πολλαπλού κύκλου ειδικών εντολών

Επεκτάσιμες ειδικού τύπου εντολές

Οι επεκτάσιμες ειδικού τύπου εντολές επιτρέπουν μία ειδική εντολή να υλοποιεί πολλές διαφορετικές λειτουργίες. Χρησιμοποιείτε ένας δείκτης που μας δείχνει ποια λειτουργία από το μπλοκ λογικής της εντολής εκτελείτε. Ο δείκτης είναι 8-bit που επιτρέπει μέχρι και 256 διαφορετικές λειτουργίες.

Παρακάτω φαίνετε ένα παράδειγμα με τρεις διαφορετικές λειτουργίες: bit-swap, byte-swap και half-word swap.



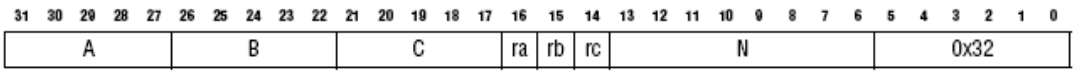
Εικόνα 5.13 - Επεκτάσιμες ειδικού τύπου εντολές

Εσωτερικού αρχείου καταχωρητών ειδικές εντολές

Ο NIOS II επιτρέπει στις ειδικές εντολές να δέχονται ορίσματα εισόδου που προέρχονται από δικό τους εσωτερικό αρχείο καταχωρητών και όχι από αυτό του επεξεργαστή, επίσης το όρισμα εξόδου μπορεί να οδηγείτε στο τοπικό αρχείο καταχωρητών.Οι ειδικές εντολές που περιέχουν εσωτερικό αρχείο καταχωρητών χρησιμοποιούν τις θύρες readra, readrb και writerc για να καθορίσουν αν η ειδική εντολή πρέπει να χρησιμοποιήσει το εσωτερικό αρχείο καταχωρητών ή τα σήματα data,datab και result. Επιπρόσθετα οι θύρες a,b και c καθορίζουν από ποιο εσωτερικό καταχωρητή να διαβάσει ή σε ποιον να γράψει η ειδική εντολή.

Instruction Fields:

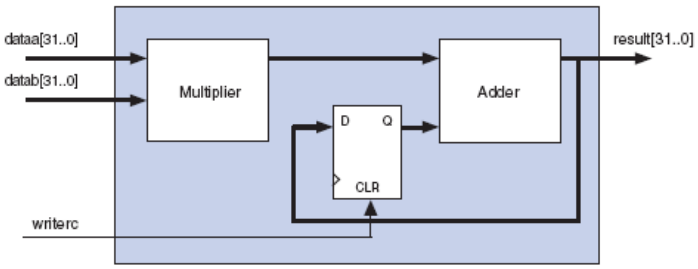
A = Register index of operand A
B = Register index of operand B
C = Register index of operand C
readra = 1 if instruction uses rA, 0 otherwise
readrb = 1 if instruction uses rB, 0 otherwise
writerc = 1 if instruction provides result for rC, 0 otherwise
N = 8-bit number that selects instruction



Internal Register File Custom Instruction Ports

Port Name	Direction	Required	Application
readra	Input	No	If readra is high, the Nios II processor supplies dataa. If readra is low, custom instruction logic reads the internal register file indexed by a.
readrb	Input	No	If readrb is high, the Nios II processor supplies datab. If readrb is low, custom instruction logic reads the internal register file indexed by b.
writerc	Input	No	If writerc is high, the Nios II processor writes to the result port. If writerc is low, custom instruction logic writes to the internal register file indexed by c.
a[4..0]	Input	No	Custom instruction internal register file index
b[4..0]	Input	No	Custom instruction internal register file index
c[4..0]	Input	No	Custom instruction internal register file index

Multiply-Accumulate Custom Logic Block



Εικόνα 5.14 - Εσωτερικού αρχείου καταχωρητών ειδικές εντολές

Όπως προαναφέρθηκε κατά την διαδικασία δημιουργίας της εφαρμογής με το NIOS IDE αυτόματα δημιουργούνται κατάλληλα macro στο αρχείο system.h που μας επιτρέπουν την χρησιμοποίηση των ειδικών εντολών σε γλώσσα C. Ένα τέτοιο παράδειγμα ακολουθεί:

Bit Swap Macro Definition

```
#define ALT_CI_BITSWAP_N 0x00

#define ALT_CI_BITSWAP(A) __builtin_custom_ini(ALT_CI_BITSWAP_N,(A))
```

Bit Swap Instruction Usage

```
1. #include "system.h"
2.
3.
4. int main (void)
5. {
6.  int a = 0x12345678;
7.  int a_swap = 0;
8.
9.  a_swap = ALT_CI_BITSWAP(a);
10. return 0;
11.}
```

Ο NIOS II χρησιμοποιεί ενσωματωμένες συναρτήσεις του μεταγλωττιστή gcc για την αντιστοίχηση των ειδικών εντολών. Μία ειδική εντολή με ακέραια ορίσματα καθορίζεται στο system.h αρχείο. Ενώ με μη ακέραια ορίσματα, χρησιμοποιώντας τις παραπάνω ενσωματωμένες συναρτήσεις μπορούμε να

έχουμε 52 διαφορετικούς συνδυασμούς ορισμάτων. Οι ενσωματωμένες αυτές συναρτήσεις ακολουθούν τον παρακάτω τύπο:
__builtin_custom_<return type>n<parameter types>

32-Bit Types Support by Custom Instructions	
Type	**Built-In Function Abbreviation**
`int`	i
`float`	f
`void *`	p

- Returning void
- Returning int
- Returning float
- Returning a pointer

Built-In Functions Returning void

```
void __builtin_custom_n (int n);
void __builtin_custom_ni (int n, int dataa);
void __builtin_custom_nf (int n, float dataa);
void __builtin_custom_np (int n, void *dataa);
void __builtin_custom_nii (int n, int dataa, int datab);
void __builtin_custom_nif (int n, int dataa, float datab);
void __builtin_custom_nip (int n, int dataa, void *datab);
void __builtin_custom_nfi (int n, float dataa, int datab);
void __builtin_custom_nff (int n, float dataa, float datab);
void __builtin_custom_nfp (int n, float dataa, void *datab);
void __builtin_custom_npi (int n, void *dataa, int datab);
void __builtin_custom_npf (int n, void *dataa, float datab);
void __builtin_custom_npp (int n, void *dataa, void *datab);
```

Built-in Functions Returning int

```
int __builtin_custom_in (int n);
int __builtin_custom_ini (int n, int dataa);
int __builtin_custom_inf (int n, float dataa);
int __builtin_custom_inp (int n, void *dataa);
int __builtin_custom_inii (int n, int dataa, int datab);
int __builtin_custom_inif (int n, int dataa, float datab);
int __builtin_custom_inip (int n, int dataa, void *datab);
int __builtin_custom_infi (int n, float dataa, int datab);
int __builtin_custom_inff (int n, float dataa, float datab);
int __builtin_custom_infp (int n, float dataa, void *datab);
int __builtin_custom_inpi (int n, void *dataa, int datab);
int __builtin_custom_inpf (int n, void *dataa, float datab);
int __builtin_custom_inpp (int n, void *dataa, void *datab);
```

Built-in Functions Returning float

```
float __builtin_custom_fn (int n);
float __builtin_custom_fni (int n, int dataa);
float __builtin_custom_fnf (int n, float dataa);
float __builtin_custom_fnp (int n, void *dataa);
float __builtin_custom_fnii (int n, int dataa, int datab);
float __builtin_custom_fnif (int n, int dataa, float datab);
float __builtin_custom_fnip (int n, int dataa, void *datab);
float __builtin_custom_fnfi (int n, float dataa, int datab);
float __builtin_custom_fnff (int n, float dataa, float datab);
float __builtin_custom_fnfp (int n, float dataa, void *datab);
float __builtin_custom_fnpi (int n, void *dataa, int datab);
float __builtin_custom_fnpf (int n, void *dataa, float datab);
float __builtin_custom_fnpp (int n, void *dataa, void *datab);
```

Built-in Functions Returning a Pointer

```
void *__builtin_custom_pn (int n);
void *__builtin_custom_pni (int n, int dataa);
void *__builtin_custom_pnf (int n, float dataa);
void *__builtin_custom_pnp (int n, void *dataa);
void *__builtin_custom_pnii (int n, int dataa, int datab);
void *__builtin_custom_pnif (int n, int dataa, float datab);
```

```
void *__builtin_custom_pnip (int n, int dataa, void *datab);
void *__builtin_custom_pnfi (int n, float dataa, int datab);
void *__builtin_custom_pnff (int n, float dataa, float datab);
void *__builtin_custom_pnfp (int n, float dataa, void *datab);
void *__builtin_custom_pnpi (int n, void *dataa, int datab);
void *__builtin_custom_pnpf (int n, void *dataa, float datab);
void *__builtin_custom_pnpp (int n, void *dataa, void *datab);
```

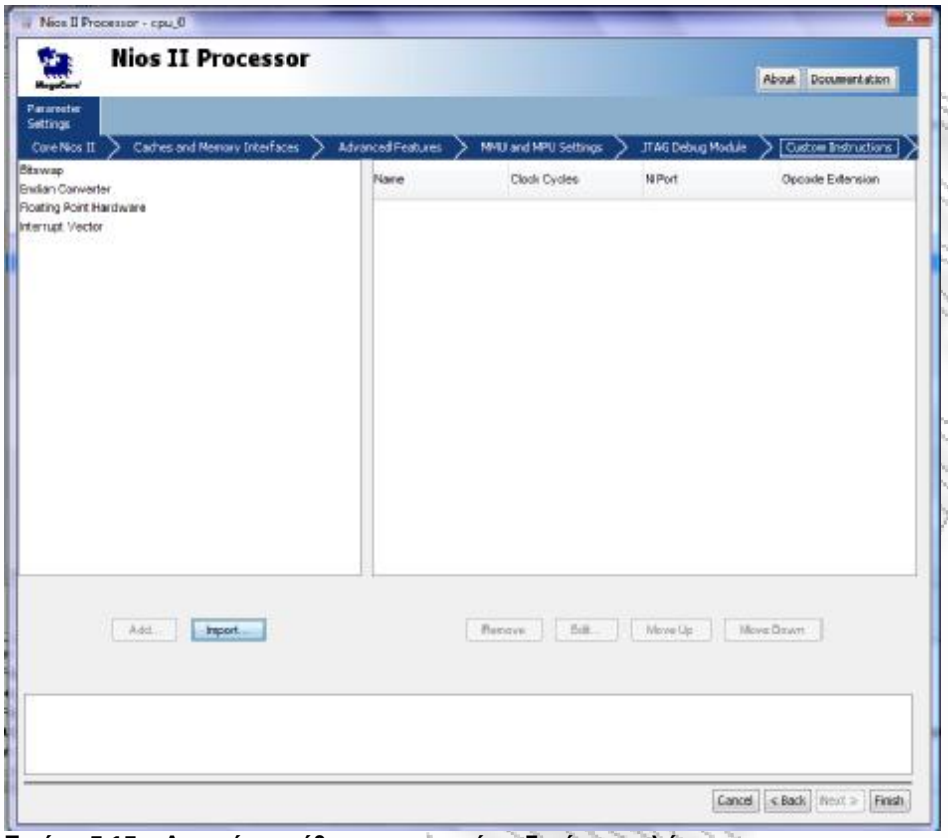
Custom Instruction Macro Usage

```
1. /* define void udef_macro1(float data); */
2. #define UDEF_MACRO1_N 0x00
3. #define UDEF_MACRO1(A) __builtin_custom_nf(UDEF_MACRO1_N, (A));
4. /* define float udef_macro2(void *data); */
5. #define UDEF_MACRO2_N 0x01
6. #define UDEF_MACRO2(B) __builtin_custom_fnp(UDEF_MACRO2_N, (B));
7.
8. int main (void)
9. {
10. float a = 1.789;
11. float b = 0.0;
12. float *pt_a = &a;
13.
14. UDEF_MACRO1(a);
15. b = UDEF_MACRO2((void *)pt_a);
16. return 0;
17. }
```

Στο παραπάνω παράδειγμα στις γραμμές 2 μέχρι 6 ορίζονται τα macro και αντιστοιχούνται στις ενσωματωμένες συναρτήσεις του μεταγλωττιστή. Το macro UDEF_MACRO1 δέχεται έναν αριθμό κινητής υποδιαστολής (float) σαν όρισμα εισόδου και δεν παράγει τίποτα. Το macro UDEF_MACRO2 δέχεται έναν δείκτη (pointer) σαν είσοδο και παράγει έναν αριθμό κινητής υποδιαστολής (float). Στις γραμμές 14 και 15 χρησιμοποιούνται οι δύο ειδικές εντολές.

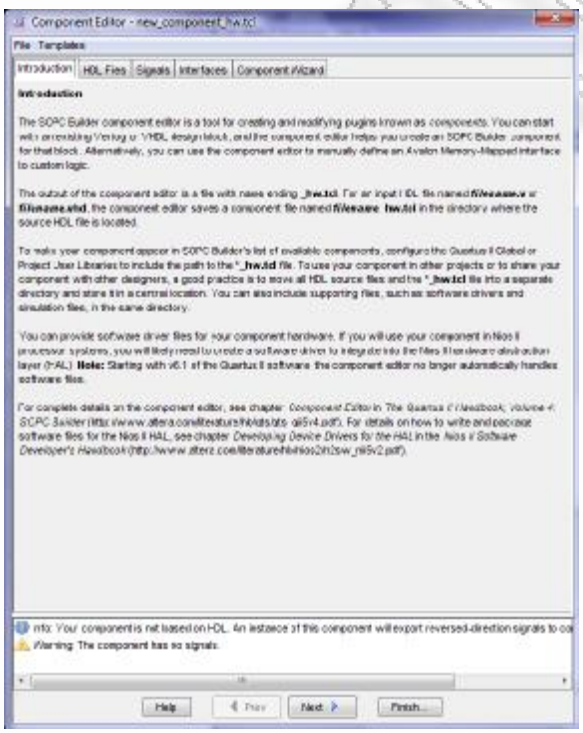
Δημιουργία μιας ειδικής εντολής

Για να δημιουργήσουμε μία νέα ειδική εντολή ακολουθούμε την παρακάτω διαδικασία:
1.Ανοίγουμε το περιβάλλων του SOPC Builder
2.Κάνουμε διπλό κλικ στην CPU Nios
3.Στις επιλογές παραμέτρων (Parameter Settings) του Nios επιλέγουμε Custom Instructions



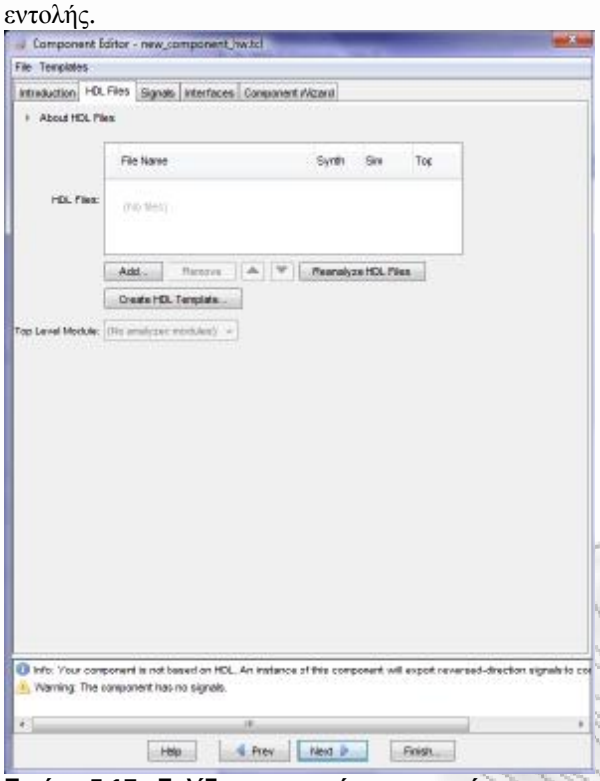
Εικόνα 5.15 – Αρχικό παράθυρο εισαγωγής ειδικών εντολών

4.Πατάμε την επιλογή Import και εμφανίζεται το παρακάτω μενού



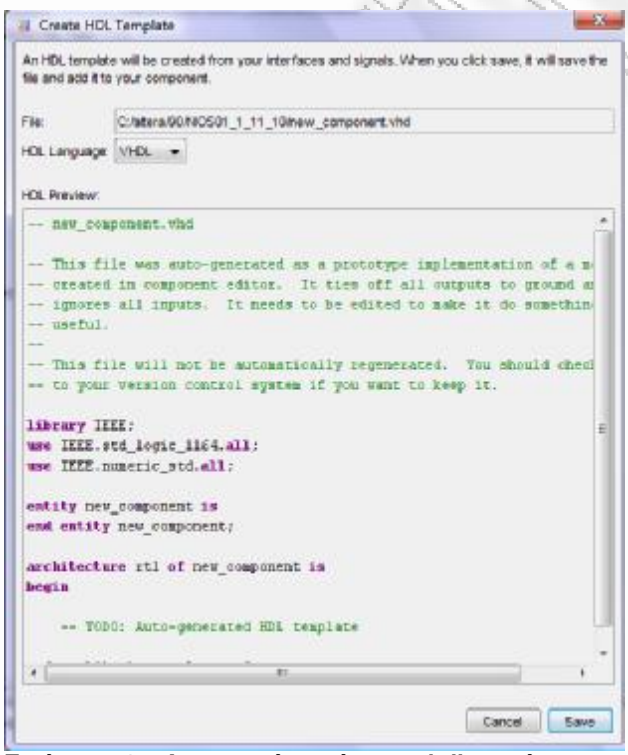
Εικόνα 5.16

5.Πατάμε Next και περνάμε στην σελίδα εισαγωγής του αρχείο που περιγράφει το υλικό της ειδικής εντολής.



Εικόνα 5.17 - Σελίδα εισαγωγής του αρχείο που περιγράφει το υλικό της ειδικής εντολής

6.Πατάμε Create HDL Template



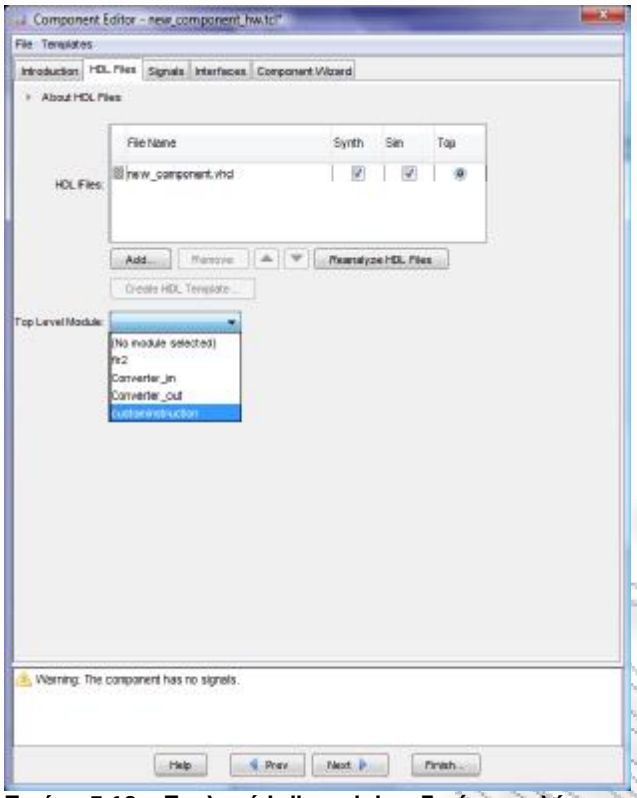
Εικόνα 5.18 – Δημιουργία πρότυπου hdl αρχείου

7.Στην επιλογή file επιλέγουμε το αρχείο vhdl που περιγράφει το υλικό της ειδικής εντολής και που έχουμε δημιουργήσει χρησιμοποιώντας το παρακάτω οδηγό που μας παρέχει η ALTERA.

VHDL Template Sample VHDL template file:

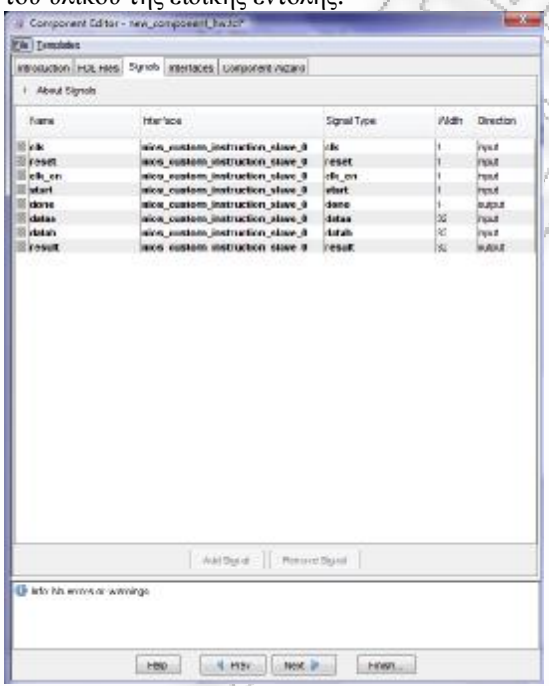
```
-- VHDL Custom Instruction Template File for Internal Register Logic
library ieee;
use ieee.std_logic_1164.all;
entity custominstruction is
port(
signal clk: in std_logic;-- CPU system clock (required for multi-cycle or extended multi-cycle)
signal reset: in std_logic;-- CPU master asynchronous active high reset (required for multi-cycle
or extended multi-cycle)
signal clk_en: in std_logic;-- Clock-qualifier (required for multi-cycle or extended multi-cycle)
signal start: in std_logic;-- Active high signal used to specify that inputs are valid (required
for multi-cycle or extended multi-cycle)
signal done: out std_logic;-- Active high signal used to notify the CPU that result is valid
(required for variable multi-cycle or extended variable multi-cycle)
signal n: in std_logic_vector(7 downto 0);-- N-field selector (required for extended), modify
width to match the number of unique operations within the custom instruction
signal dataa: in std_logic_vector(31 downto 0);-- Operand A (always required)
signal datab: in std_logic_vector(31 downto 0);-- Operand B (optional)
signal a: in std_logic_vector(4 downto 0);-- Internal operand A index register
signal b: in std_logic_vector(4 downto 0);-- Internal operand B index register
signal c: in std_logic_vector(4 downto 0);-- Internal result index register
signal readra: in std_logic;-- Read operand A from CPU (otherwise use internal operand A)
signal readrb: in std_logic;-- Read operand B from CPU (otherwise use internal operand B)
signal writerc: in std_logic;-- Write result to CPU (otherwise write to internal result)
signal result: out std_logic_vector(31 downto 0)-- result (always required)
);
end entity custominstruction;
architecture a_custominstruction of custominstruction is
-- local custom instruction signals
begin
-- custom instruction logic (note: external interfaces can be used as well)
-- use the n[7..0] port as a select signal on a multiplexer to select the value to feed result[31..0]
end architecture a_custominstruction;
```

8.Πατάμε Reanalyze HDL Files και επιλέγουμε το Top Level Module του αρχείου μας, στην περίπτωσή μας custominstruction



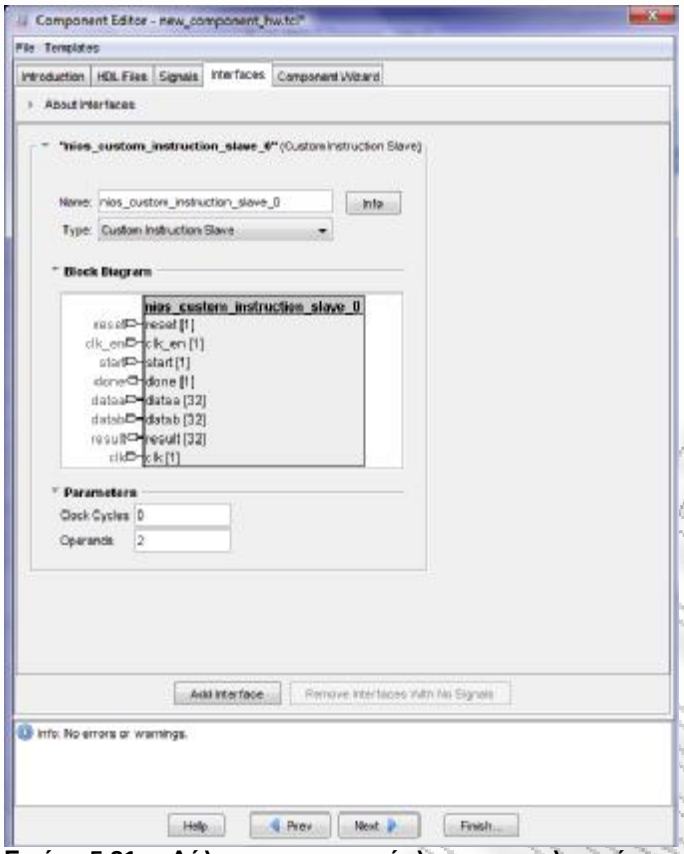
Εικόνα 5.19 – Επιλογή hdl module ειδικής εντολής

9.Πατάμε Next και περνάμε στο παράθυρο όπου εμφανίζονται τα σήματα (Signals) δηλαδή οι θύρες του υλικού της ειδικής εντολής.



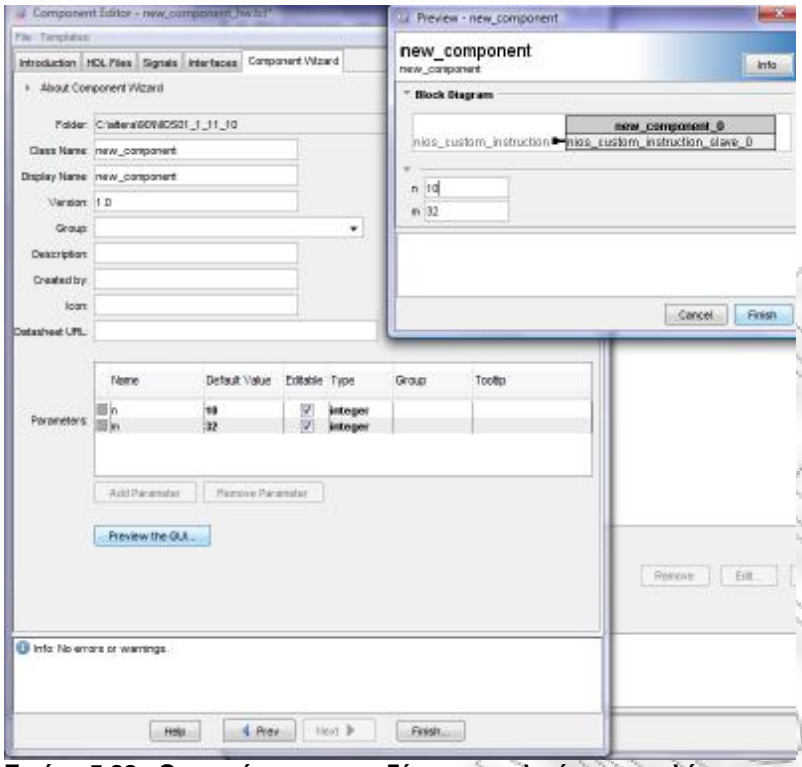
Εικόνα 5.20 - Τα σήματα (Signals) δηλαδή οι θύρες του υλικού της ειδικής εντολής

10.Με Next εμφανίζεται το ακόλουθο παράθυρο όπου δηλώνουμε τους κύκλους του ρολογιού που θα εκτελείτε η ειδική εντολή (0 για μεταβαλλόμενο πολλαπλού κύκλου) και τον αριθμό των ορισμάτων της εντολής.



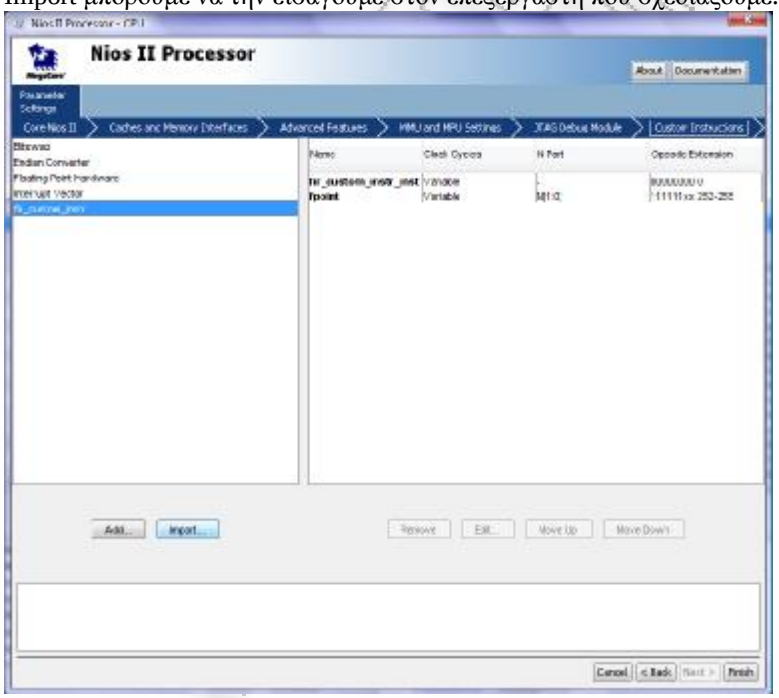
Εικόνα 5.21 – Δήλωση για τους κύκλους του ρολογιού που θα εκτελείτε η ειδική εντολή (0 για μεταβαλλόμενο πολλαπλού κύκλου) και τον αριθμό των ορισμάτων της εντολής

11.Με Next περνάμε στο τελευταίο στάδιο όπου φαίνονται και οι παράμετροι που δέχεται το υλικό της εντολής στην περίπτωσή μας n=10 ο αριθμός των συντελεστών του φίλτρου και m=32 τα bits του ορίσματος εξόδου.



Εικόνα 5.22 - Οι παράμετροι που δέχεται το υλικό της εντολής

12.Πατόντας Finish ολοκληρώνουμε την διαδικασία δημιουργίας της ειδικής εντολής. Αυτή τότε εμφανίζεται στην λίστα των διαθέσιμων ειδικών εντολών του Nios και επιλέγοντας την πατώντας Import μπορούμε να την εισάγουμε στον επεξεργαστή που σχεδιάζουμε.



Εικόνα 5.23 - Ολοκλήρωση της διαδικασίας δημιουργίας της ειδικής εντολής

Ακολουθώντας όλη την παραπάνω διαδικασία εισάγαμε στον Nios μία ειδική εντολή [35], [37], [38] που υλοποιεί τον FIR αλγόριθμο. Ο VHDL κώδικας που υλοποιεί το LP FIR φίλτρο [65],[66] είναι:

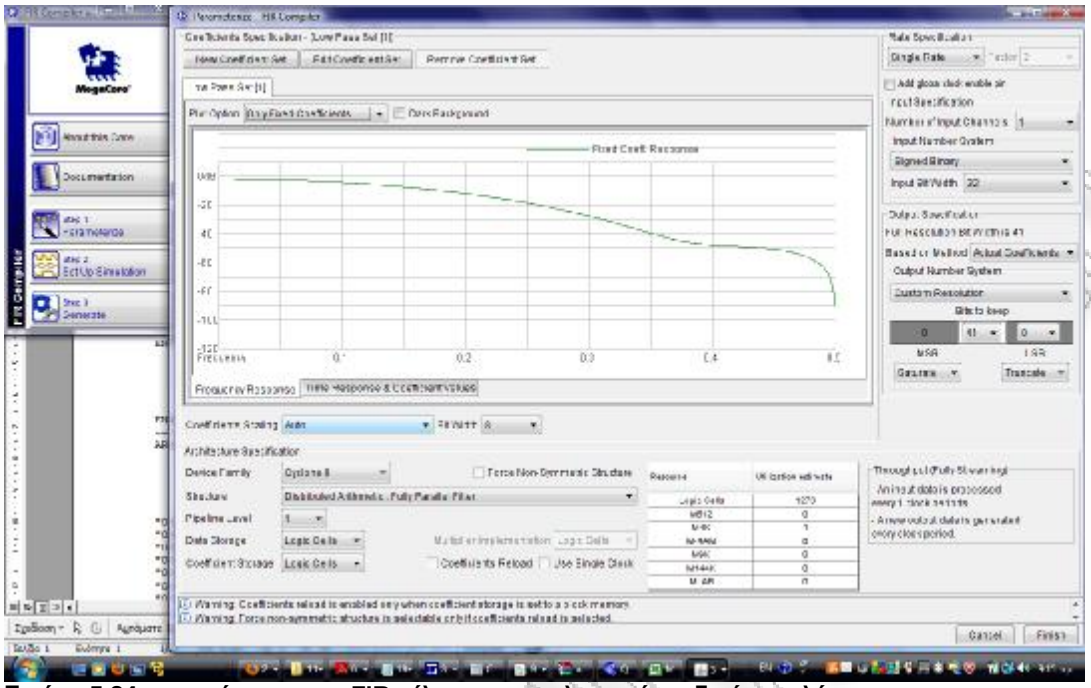
```
------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all; --package needed for SIGNED
------------------------------------------------------
ENTITY fir2 IS
        GENERIC (n :INTEGER :=10; m: INTEGER :=32);
        --n = # of coef. , m = # of bits of input and coef.
                PORT ( x: in signed (m-1 DOWNTO 0);
                  clk, rst, clk_en: IN STD_LOGIC;
                  y : out signed(2*m-1 DOWNTO 0));
END fir2;
------------------------------------------------------
ARCHITECTURE rtl OF fir2 IS
        TYPE registers IS ARRAY (n-2 DOWNTO 0) OF signed (m-1 DOWNTO 0);
        TYPE coefficients IS ARRAY (n-1 DOWNTO 0) OF signed (m-1 DOWNTO 0);
        SIGNAL reg: registers;
        CONSTANT coef : coefficients :=(

"00000000000000000000000000000000", --0
"00000000000000000000000000000000", --0
"00000000000000000000000000010000", --16
"00000000000000000000000001000011", --67
"00000000000000000000000001111111", --127
"00000000000000000000000001111111", --127
"00000000000000000000000001000011", --67
"00000000000000000000000000010000", --16
"00000000000000000000000000000000", --0
"00000000000000000000000000000000" --0
 );
BEGIN
        PROCESS (clk,rst)
                VARIABLE acc, prod: signed (2*m-1 DOWNTO 0):= (OTHERS=>'0');
                VARIABLE sign: STD_LOGIC;
        BEGIN
                -------reset : ---------------------------
                IF (rst= '1') THEN
                        FOR i IN n-2 DOWNTO 0 LOOP
                                FOR j IN m-1 DOWNTO 0 LOOP
                                        reg(i)(j)<='0';
                                END LOOP;
                        END LOOP;
                ----register inference +MAC : -----------
                ELSIF (clk 'EVENT AND clk='1' AND clk_en='1') THEN
                                acc := coef(0)*x;
                                FOR i IN 1 TO n-1 LOOP
                                        sign := acc(2*m-1);
                                        prod := coef(i)*reg(n-1-i);
                                        acc:= acc+prod;
                                        ---overflow check:-----------
                                        IF (sign=prod(prod'left)) AND (acc(acc'left)/=sign)
                                        THEN
                                        acc := (acc'LEFT => sign, OTHERS => NOT sign);
                                        END IF;
                                END LOOP;
                                reg <= x & reg(n-2 DOWNTO 1);
                        END IF;
```

```
                y<=acc;
            END PROCESS;
        END rtl;
        ---------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY Converter_in IS
PORT(
a_in : out signed (31 downto 0);
b_in : in std_logic_vector (31 downto 0)
);
END Converter_in;

ARCHITECTURE struct OF Converter_in IS
begin

label1: for i in 0 to a_in'LENGTH-1 generate
a_in(i) <= b_in(i);
end generate label1;
END struct;

        ---------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY Converter_out IS
PORT(
a_out : IN signed (63 downto 0);
b_out : OUT std_logic_vector (31 downto 0);
done : out std_logic
);
END Converter_out;

ARCHITECTURE struct OF Converter_out IS
SIGNAL TEMP : std_logic_vector (63 downto 0);
begin

label1: for i in 0 to TEMP'LENGTH-1 generate
TEMP(i) <= a_out(i);
done <= '1';
end generate label1;
b_out<=TEMP(43 DOWNTO 12);
END struct;
---------------------------------------------------------------------------
        --Copyright (C) 1991-2004 Altera Corporation
--Any megafunction design, and related net list (encrypted or decrypted),
--support information, device programming or simulation file, and any other
--associated documentation or information provided by Altera or a partner
--under Altera's Megafunction Partnership Program may be used only to
--program PLD devices (but not masked PLD devices) from Altera. Any other
--use of such megafunction design, net list, support information, device
--programming or simulation file, or any other related documentation or
```

```
--information is prohibited for any other purpose, including, but not
--limited to modification, reverse engineering, de-compiling, or use with
--any other silicon devices, unless such use is explicitly licensed under
--a separate agreement with Altera or a megafunction partner. Title to
--the intellectual property, including patents, copyrights, trademarks,
--trade secrets, or maskworks, embodied in any such megafunction design,
--net list, support information, device programming or simulation file, or
--any other related documentation or information provided by Altera or a
--megafunction partner, remains with Altera, the megafunction partner, or
--their respective licensors. No other licenses, including any licenses
--needed under any third party's intellectual property, are provided herein.
--Copying or modifying any file, or portion thereof, to which this notice
--is attached violates this copyright.


-- VHDL Custom Instruction Template File for Multi-Cycle Logic

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all; --package needed for SIGNED


entity custominstruction is
GENERIC (n :INTEGER :=10; m: INTEGER :=32);
port(
        signal clk: in std_logic; -- CPU system clock (always required)signal reset: in std_logic;
                -- CPU master asynchronous active high reset (always required)
        signal clk_en: in std_logic;-- Clock-qualifier (always required)
        signal start: in std_logic; -- Active high signal used to specify that inputs are valid (always
                                    required)
        signal done: out std_logic; -- Active high signal used to notify the CPU that result is valid
                                    (required for variable multi-cycle)
        signal dataa: in std_logic_vector(31 downto 0); -- Operand A (always required)
        signal datab: in std_logic_vector(31 downto 0); -- Operand B (optional)
        signal result: out std_logic_vector(31 downto 0) -- result (always required)
        --signal result: out signed(63 downto 0) -- result (always required)
);

end entity custominstruction;



architecture a_custominstruction of custominstruction is

component fir2 is
port(
         x: in signed (m-1 DOWNTO 0);
         clk, rst,clk_en : IN STD_LOGIC;
         y : out signed(2*m-1 DOWNTO 0)
        );
end component;
-------------------------------------------
component Converter_in is
port(
         a_in : out signed (31 downto 0);
  b_in : in std_logic_vector (31 downto 0)
        );
```

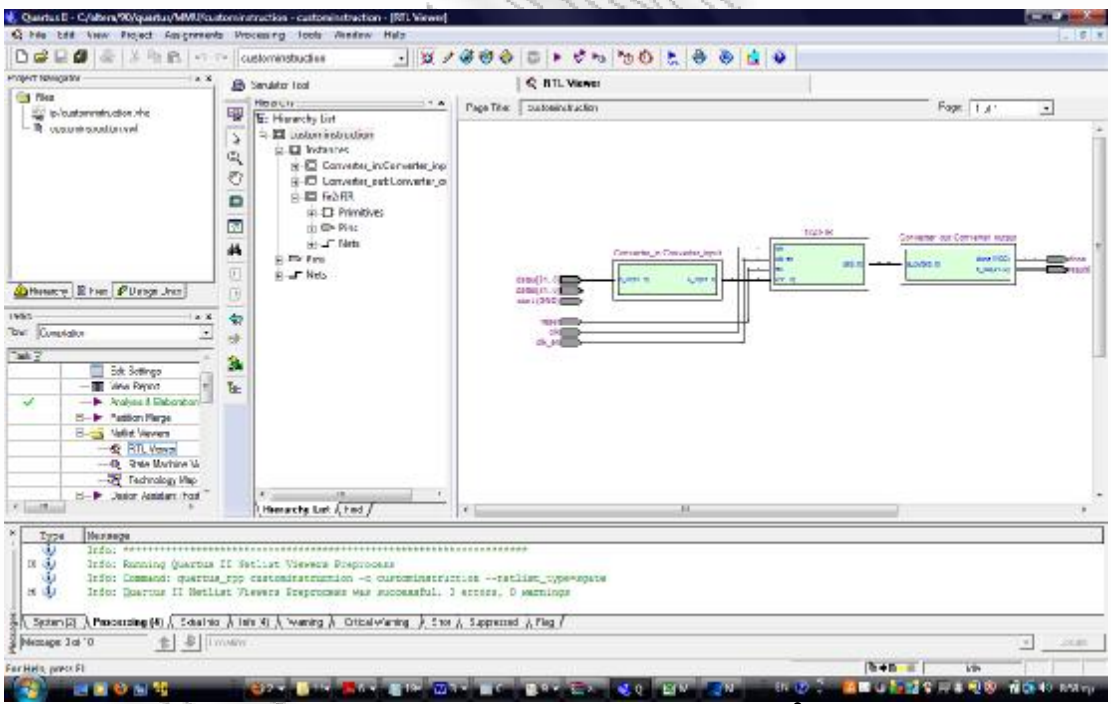
```
end component;
-------------------------------------
-----------------------------------------
component Converter_out is
port(
            a_out : IN signed (63 downto 0);
  b_out : OUT std_logic_vector (31 downto 0);
  done : out std_logic
            );
end component;
            -- local custom instruction signals
signal a_in_s: signed (31 downto 0);
signal b_in_s: std_logic_vector(31 downto 0);
signal a_out_s: signed(63 downto 0);
signal b_out_s: std_logic_vector(63 downto 0);

begin

            -- custom instruction logic (note: external interfaces can be used as well)
FIR:fir2
port map (
                        x=>a_in_s,
  clk=>clk,
  clk_en=>clk_en,
  rst=>reset,
  y=>a_out_s
  --y=>result
                        );
-----------------------------------------------------
Converter_input:Converter_in
port map (
                        b_in=>dataa,
                        a_in=>a_in_s
                        );

Converter_output:Converter_out
port map (
                        b_out=>result,
  a_out=>a_out_s,
  done=>done
                        );
end architecture a_custominstruction;
```

Οι συντελεστές είναι (0,0,16,67,127,127,67,16,0,0), αυτοί μας δίνουν μία απόκριση



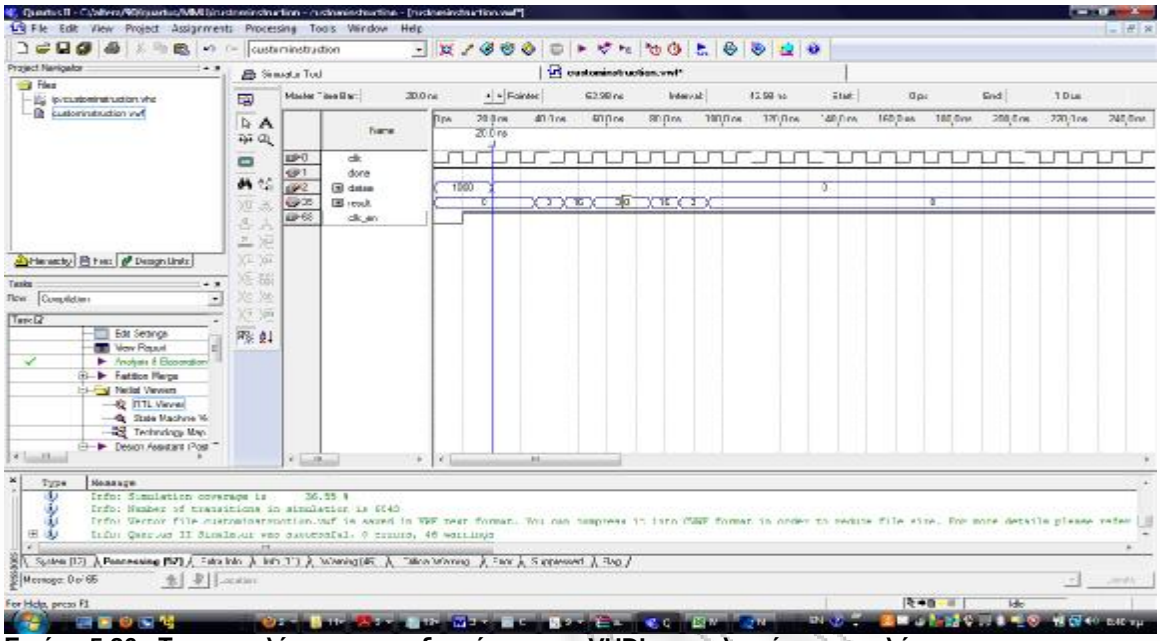
Εικόνα 5.24 – η απόκριση του FIR φίλτρου που υλοποιεί η ειδική εντολή

με frequensy 0.5 =4khz.Στην έξοδο παίρνουμε από τα (63 downto 0) τα (43 downto 12) bits για να έχουμε τα 32bits που απαιτεί ο επεξεργαστής. (Κάνουμε ένα scaling στην τιμή της εξόδου.)



Εικόνα 5.25 – Το δομικό διάγραμμα του υλικού της ειδικής εντολής (1^{ο} επίπεδο)

Τα αποτελέσματα της εξομοίωσης σε VHDL του υλικού της εντολής είναι:



Εικόνα 5.26 - Τα αποτελέσματα της εξομοίωσης σε VHDL του υλικού της εντολής

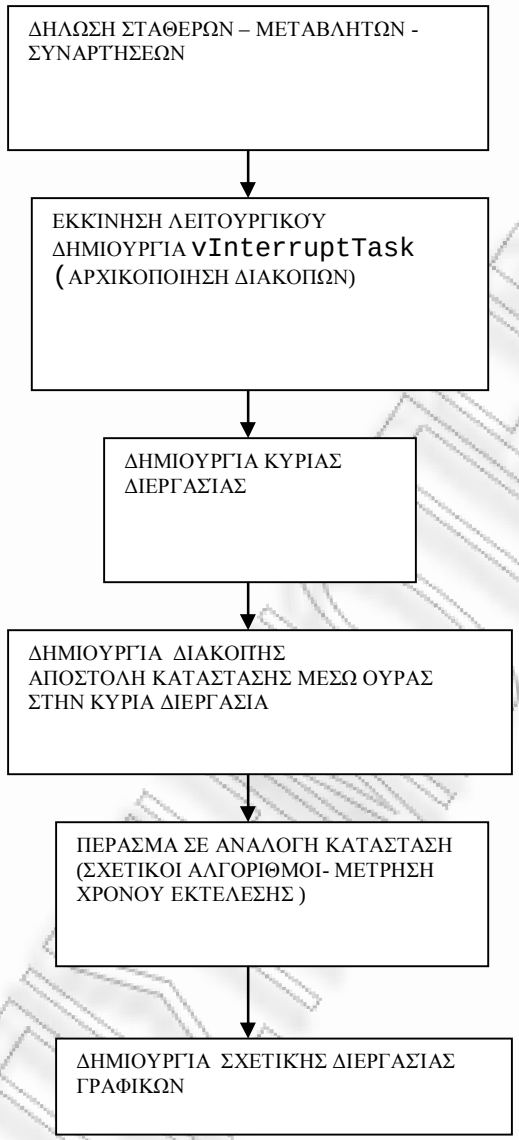
Τα σχετικά macros που δημιουργούνται στο system.h αρχείο είναι:

```
* custom instruction macros
 *
 */

#define ALT_CI_FIR_CUSTOM_INSTR_INST_N 0x00000000
#define ALT_CI_FIR_CUSTOM_INSTR_INST(A)
__builtin_custom_ini(ALT_CI_FIR_CUSTOM_INSTR_INST_N,(A))
```

5.6 **Το Λογισμικό του συστήματος**

Το λογισμικό έχει αναπτυχθεί με το NIOS II IDE v9.0 της ALTERA [32], [33], [34], [69]. Έχει επιλεγεί για λόγους ευελιξίας κατά την ανάπτυξη της εφαρμογής να δοκιμαστεί η δυνατότητα φόρτωσης ενός λειτουργικού συστήματος πραγματικού χρόνου. Για αυτό τον σκοπό επιλέγηκε το RTOS FreeRTOS ένα ελεύθερο λειτουργικό σύστημα που μας παρέχεται από την Real Time Engineers LTD, με αρκετή απλότητα στην χρήση του, αλλά χωρίς μεγάλη εφαρμογή στον επεξεργαστή NIOS II, σε αντίθεση με άλλους εμπορικούς επεξεργαστές (AVR32, ARM). Η βασική δομή του λογισμικού είναι η ακόλουθη:

Αρχικά δημιουργείται μία διεργασία υψηλής προτεραιότητας η `vInterruptTask` η οποία χρησιμοποιείται για την αρχικοποίηση των διακοπών του συστήματος και σβήνεται μόλις εκτελεστεί μία φορά κατά την έναρξη λειτουργίας του συστήματος. Παράλληλα μία άλλη διεργασία χαμηλότερης προτεραιότητας δημιουργείτε και ξεκινάει να τρέχει μόλις η `vInterruptTask` σβηστεί. Αυτή η διεργασία είναι η κύρια διεργασία του συστήματος και μέσα σε αυτήν με δομές if then else, όταν συμβεί μία διακοπή από το πάτημα ενός πλήκτρου, ανάλογα με την τιμή μίας μεταβλητής που μεταδίδεται μέσω δομών ουρών από την ρουτίνα εξυπηρέτησης διακοπής στην κύρια διεργασία, το σύστημα περνάει στην ανάλογη κατάσταση λειτουργίας. Τότε για κάθε κατάσταση δημιουργείται μία νέα διεργασία γραφικών υψηλής προτεραιότητας η οποία αφού εμφανίσει τα σχετικά γραφικά σβήνεται.



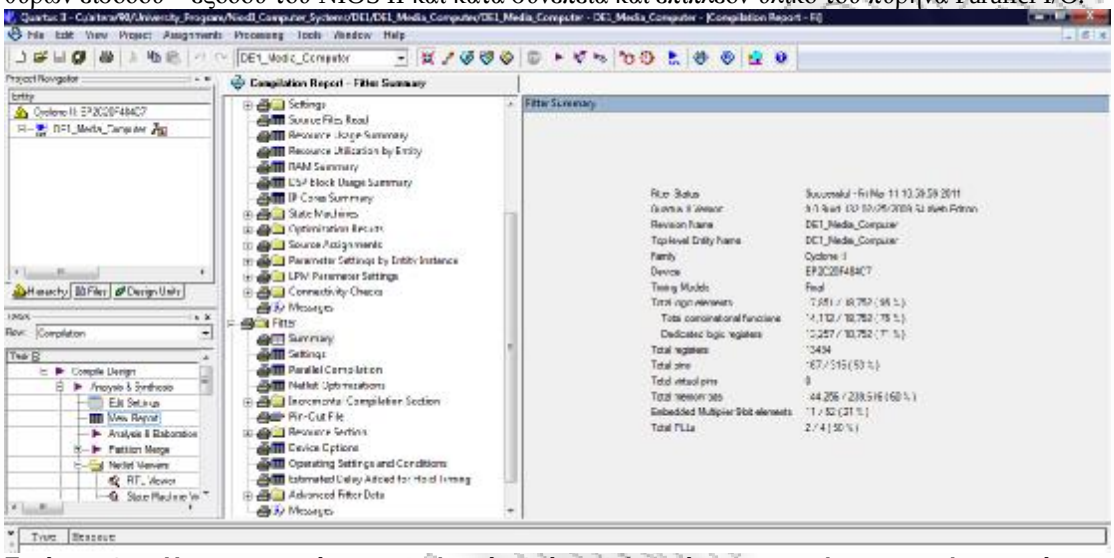
Με την χρήση του FreeRTOS δεν επηρεάζεται ο τρόπος χρησιμοποίησης των διακοπών μια και αυτές είναι συμβατές με το ALTERA HAL. Οι διακοπές δηλώνονται με την χρήση της συνάρτησης alt_irq_register(). Στο τέλος κάθε ρουτίνας εξυπηρέτησης διακοπής καλείται το macro, portend_SWITCING_ISR(), με τιμή 0 αν δεν χρειάζεται από την ρουτίνα διακοπής να μεταβούμε σε μία άλλη διεργασία από αυτήν που ξεκινήσαμε, ή τιμή 1 αν επιστρέφουμε στην αρχική διεργασία.
Για την χρησιμοποίηση του FreeRTOS απαιτείτε να υπάρχει σε υλικό ένας χρονιστής με όνομα “sys_clk” και περίοδο 1ms, στοιχείο που έχει υλοποιηθεί.

Οι ρυθμίσεις που δίνονται για την διαμόρφωση του λειτουργικού συστήματος στο αρχείο FreeRTOSConfig.h είναι οι παρακάτω:

```
/*-----------------------------------------------------------
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF
THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *-----------------------------------------------------------*/

#define configUSE_PREEMPTION                    1
#define configUSE_IDLE_HOOK                     0
#define configUSE_TICK_HOOK                     0
#define configTICK_RATE_HZ                      ( ( portTickType ) 1000)
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) SYS_CLK_FREQ )
#define configMAX_PRIORITIES                    ( ( unsigned portBASE_TYPE ) 8 )
#define configMINIMAL_STACK_SIZE                ( 1024)
#define configISR_STACK_SIZE                    configMINIMAL_STACK_SIZE
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 1024*80 ) )
#define configMAX_TASK_NAME_LEN                 ( 28 )
#define configUSE_TRACE_FACILITY                0
#define configUSE_16_BIT_TICKS                  0
#define configIDLE_SHOULD_YIELD                 0
#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configUSE_COUNTING_SEMAPHORES           1
#define configCHECK_FOR_STACK_OVERFLOW2
#define configQUEUE_REGISTRY_SIZE               0

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES                   0
#define configMAX_CO_ROUTINE_PRIORITIES         ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet                1
#define INCLUDE_uxTaskPriorityGet               1
#define INCLUDE_vTaskDelete                     1
#define INCLUDE_vTaskCleanUpResources           0
#define INCLUDE_vTaskSuspend                    1
#define INCLUDE_vTaskDelayUntil                 1
#define INCLUDE_vTaskDelay                      1
#define INCLUDE_uxTaskGetStackHighWaterMark     1

/* The priority at which the tick interrupt runs. This should probably be
kept at 1. */
#define configKERNEL_INTERRUPT_PRIORITY                 0x01

/* The maximum interrupt priority from which FreeRTOS.org API functions can
be called. Only API functions that end in ...FromISR() can be used within
interrupts. */
#define configMAX_SYSCALL_INTERRUPT_PRIORITY    0x03

#endif /* FREERTOS_CONFIG_H */
```

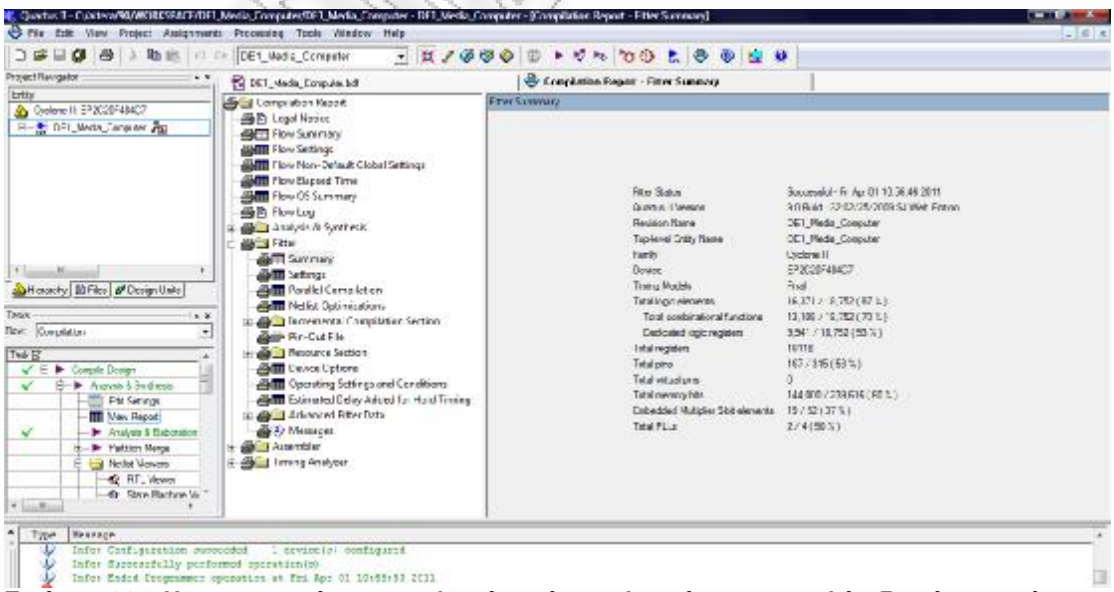
5.7 **Σχολιασμός του τελικού συστήματος**

Η παρακάτω εικόνα μας δείχνει την χρησιμοποίηση του υλικού κατά την υλοποίηση με Hardware Accelerator, όπου φτάνουμε το 95% των διαθέσιμων πόρων, χρησιμοποιώντας 11 πολλαπλασιαστές του FPGA μία λύση αρκετά επιβαρυντική σε υλικό. Επίσης με αυτήν την υλοποίηση απαιτήθηκε επιπλέον κώδικας για την διασύνδεση του επεξεργαστή με τον επιταχυντή μέσω των παράλληλων θυρών εισόδου – εξόδου του NIOS II και κατά συνέπεια και επιπλέον υλικό του πυρήνα Parallel I/O.

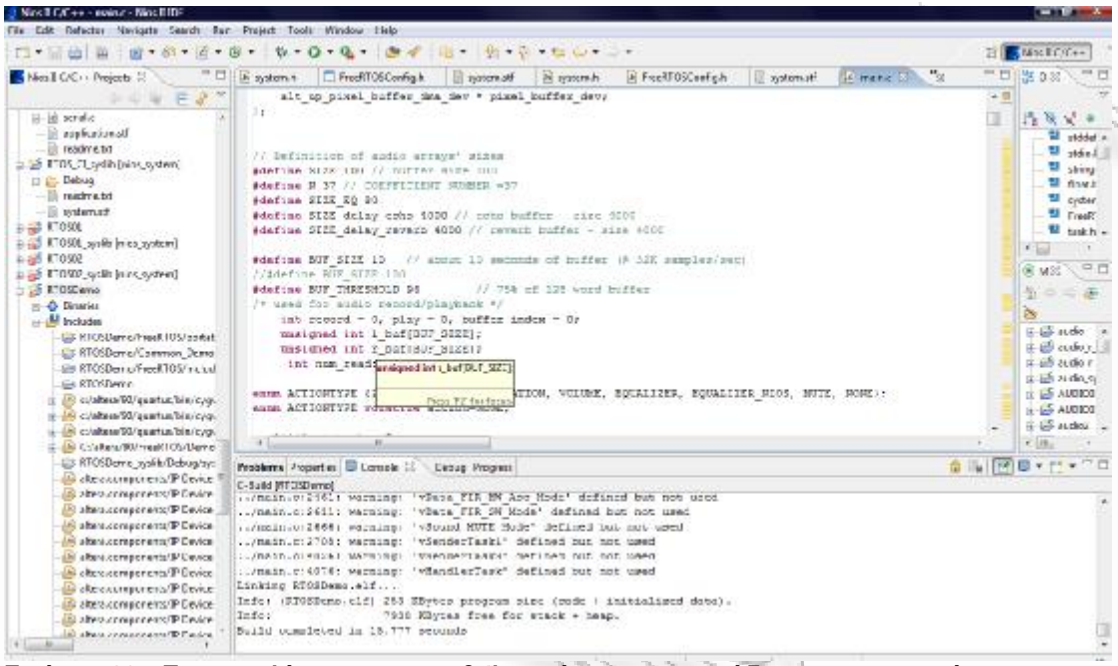


Εικόνα 5.27 – Η χρησιμοποίηση του υλικού κατά την υλοποίηση με Hardware Accelerator, όπου φτάνουμε το 95% των διαθέσιμων πόρων

Η παρακάτω εικόνα μας δείχνει την χρησιμοποίηση του υλικού κατά την υλοποίηση με εντολή ειδικού σκοπού, όπου φτάνουμε το 87% των διαθέσιμων πόρων, χρησιμοποιώντας 19 πολλαπλασιαστές λόγο της πλήρης παραλληλίας του υλικού που υλοποιεί την ειδική εντολή, μία λύση λιγότερο επιβαρυντική σε υλικό, που πλεονεκτεί όπως θα φανεί στην συνέχεια και στην ταχύτητα επεξεργασίας και αποτελεί την πλέον ενδεδειγμένη λύση για το σύστημα μας.



Εικόνα 5.28 – Η χρησιμοποίηση του υλικού κατά την υλοποίηση με εντολή ειδικού σκοπού, όπου φτάνουμε το 87% των διαθέσιμων πόρων



Εικόνα 5.29 – Τα αποτελέσματα της συμβολομετάφρασης του κώδικα της εφαρμογής

Εδώ παρατηρούμε ότι ο κώδικας έχει μέγεθος 253Kbytes μέγεθος που θεωρητικά είναι μικρότερο με την χρησιμοποίηση ειδικών εντολών, αφού τότε δεν έχουμε εντολές για την διασύνδεση με τον επιταχυντή υλικού.

5.8 **Μετρήσεις και αξιολόγηση επιδόσεων**

Για τον έλεγχο ορθής λειτουργίας των αλγορίθμων και των ειδικών κυκλωμάτων και εντολών που αναπτύχθηκαν, όπως και για την ευκολότερη αξιολόγηση της επίδοσης των διαφόρων υλοποιήσεων του αλγορίθμου FIR, υπάρχουν τα ακόλουθα mode λειτουργίας:

Data SW Float mode - Δίνεται σαν διάνυσμα εισόδου στον αλγόριθμο το ακόλουθο (10000,0,0,0,0,0,0,0,0,0) και σαν συντελεστές οι {0.0324577, 0.0362059, 0.0391191, 0.0409677, 0.0416012, 0.0416012, 0.0409677, 0.0391191, 0.0362059, 0.0324577}, τότε σαν έξοδος αναμένονται οι συντελεστές πολλαπλασιασμένοι Χ 10000000. Έτσι ελέγχουμε την ορθότητα του αλγορίθμου και μετράμε τον χρόνο εκτέλεσης για 2 διανύσματα των 10 δειγμάτων το κάθε ένα. Στην οθόνη εμφανίζεται το αποτέλεσμα εξόδου και ο χρόνος επεξεργασίας των διανυσμάτων εισόδου.
Σαν αποτέλεσμα παίρνουμε: **207504** κύκλους και σαν έξοδο {324577, 362059,391191, 409677, 416012, 416012, 409677, 391191, 362059, 324577) ακολουθία που επιβεβαιώνει και την ορθότητα του αλγορίθμου.

Data SW INT mode - Δίνεται σαν διάνυσμα εισόδου στον αλγόριθμο το ακόλουθο (1,0,0,0,0,0,0,0,0,0) και σαν συντελεστές οι {0,0,16,64,127,127,64,16,0,0},τότε σαν έξοδος αναμένονται οι συντελεστές του φίλτρου. Έτσι ελέγχουμε την ορθότητα του αλγορίθμου και μετράμε τον χρόνο εκτέλεσης για 2 διανύσματα των 10 δειγμάτων το κάθε ένα. Στην οθόνη εμφανίζεται το αποτέλεσμα εξόδου και ο χρόνος επεξεργασίας των διανυσμάτων εισόδου.
Σαν αποτέλεσμα παίρνουμε: **21297** κύκλους και σαν έξοδο {0,0,16,64,127,127,64,16,0,0} ακολουθία που επιβεβαιώνει και την ορθότητα του αλγορίθμου.

Data HW ACC mode - Δίνεται σαν διάνυσμα εισόδου στον αλγόριθμο το ακόλουθο {2000000000,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0} και σαν συντελεστές αυτοί που έχει ο FIR HW Accelerator, τότε σαν έξοδος αναμένονται οι συντελεστές . Έτσι ελέγχουμε την ορθότητα του αλγορίθμου και μετράμε τον χρόνο εκτέλεσης για 2 διανύσματα των 37 δειγμάτων

το κάθε ένα. Στην οθόνη εμφανίζεται το αποτέλεσμα εξόδου και ο χρόνος επεξεργασίας των διανυσμάτων εισόδου.
Σαν αποτέλεσμα παίρνουμε: **11031** κύκλους και σαν έξοδο για τις τιμές [7..16] έχουμε {0,1953125, 7812500, 7812500, 0, -15625000, -31250000, -31250000, 0, 66406250} ακολουθία που επιβεβαιώνει και την ορθότητα του αλγορίθμου, μία και είναι ίδια με τους συντελεστές [7..16].

Data CI mode - Δίνεται σαν διάνυσμα εισόδου στον αλγόριθμο το ακόλουθο (1000,0,0,0,0,0,0,0,0,0) και σαν συντελεστές οι {0,0,16,64,127,127,64,16,0,0}, τότε σαν έξοδος αναμένονται οι συντελεστές πολλαπλασιασμένοι Χ 1000. Έτσι ελέγχουμε την ορθότητα του αλγορίθμου και μετράμε τον χρόνο εκτέλεσης για 2 διανύσματα των 10 δειγμάτων το κάθε ένα. Στην οθόνη εμφανίζεται το αποτέλεσμα εξόδου και ο χρόνος επεξεργασίας των διανυσμάτων εισόδου.
Σαν αποτέλεσμα παίρνουμε: **1558** κύκλους και σαν έξοδο {0,0,0,3,16,31,31,16,3,0} ακολουθία που επιβεβαιώνει και την ορθότητα του αλγορίθμου.

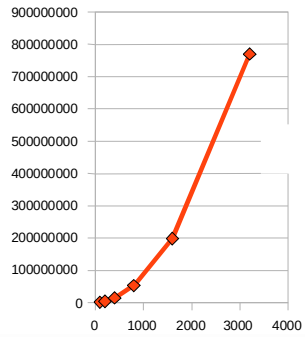
Οι επόμενες μετρήσεις μας παρουσιάζουν τους κύκλους και την επιτάχυνση που έχουμε κατά τις διαφορετικές προσεγγίσεις υλοποίησης του FIR φίλτρου, για διαφορετικό αριθμό δειγμάτων εισόδου.

ΔΕΙΓΜΑΤΑ	FLOAT CYCLES			INT CYCLES	CI CYCLES	ΕΠΙΤΑΧΥΝΣΗ ΤΗΣ FLOAT 1	HACC CYCLES	ΕΠΙΤΑΧΥΝΣΗ ΤΗΣ FLOAT 2
100	**1361302**			**1136898**	**13131**	103,67	**32612**	41,74
200	**4167301**			**3718378**	**29680**	140,41	**61394**	67,88
400	**14637365**			**13246807**	**55794**	262,35	**162197**	90,24
800	**52957352**			**49786443**	**151657**	349,19	**283267**	186,95
1600	**198920144**			**192675681**	**249225**	798,15	**694478**	286,43
3200	**769776812**	**15,40**	**SEC**	**757793015**	**476360**	1615,96	**1009218**	762,75

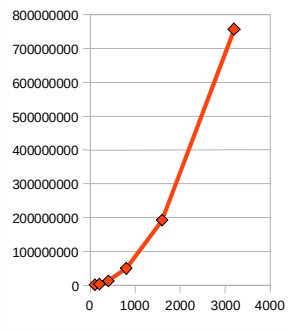
Όπου στην στήλη ΔΕΙΓΜΑΤΑ εμφανίζονται 6 διαφορετικοί αριθμοί δειγμάτων για τα οποία πήραμε τις ανάλογες μετρήσεις. Στην στήλη FLOAT CYCLES εμφανίζονται οι κύκλοι για φίλτρο με συντελεστές αριθμούς κινητής υποδιαστολής και υλοποίηση της αριθμητικής με ειδικές εντολές κινητής υποδιαστολής της ALTERA. Ο χρόνος 15,40 sec είναι μία ένδειξη του χρόνου που απαιτείται για το φιλτράρισμα 3200 δειγμάτων με τον παραπάνω τρόπο, μία πολύ μεγάλη τιμή. Στην στήλη INT CYCLES εμφανίζονται οι κύκλοι για φίλτρο με συντελεστές ακέραιους αριθμούς. Στην στήλη CI CYCLES εμφανίζονται οι κύκλοι για υλοποίηση του φίλτρου με ειδική εντολή. Στην στήλη ΕΠΙΤΑΧΥΝΣΗ ΤΗΣ FLOAT 1 εμφανίζεται η επιτάχυνση που επιτυγχάνεται από την υλοποίηση του φίλτρου με ειδική εντολή σε σχέση με την υλοποίηση με κώδικα και υλοποίηση της αριθμητικής με ειδικές εντολές κινητής υποδιαστολής της ALTERA. Στην στήλη HACC CYCLES εμφανίζονται οι κύκλοι για υλοποίηση του φίλτρου με επιταχυντή υλικού. Στην στήλη ΕΠΙΤΑΧΥΝΣΗ ΤΗΣ FLOAT 2 εμφανίζεται η επιτάχυνση που επιτυγχάνεται από την υλοποίηση του φίλτρου με επιταχυντή υλικού σε σχέση με την υλοποίηση με κώδικα και υλοποίηση της αριθμητικής με ειδικές εντολές κινητής υποδιαστολής της ALTERA.

Από αυτές παρατηρούμε ότι λόγο της μορφής του αλγορίθμου υλοποίησης του FIR φίλτρου σε λογισμικό, για κάθε διπλασιασμό του αριθμού των δειγμάτων έχουμε 3,9 φορές αύξηση του χρόνου εκτέλεσης. Ενώ λόγο της παράλληλης υλοποίησης του αλγορίθμου σε υλικό τόσο με ειδικές εντολές όσο και με επιταχυντή για κάθε διπλασιασμό του αριθμού των δειγμάτων έχουμε 2 φορές αύξηση του χρόνου εκτέλεσης. Έτσι σαν τελικό συμπέρασμα έχουμε ότι όσο αυξάνει ο αριθμός της ακολουθίας των δειγμάτων εισόδου, τόσο μεγαλύτερη είναι η επιτάχυνση που επιτυγχάνεται με υλοποιήσεις υλικού, με πλέον αποδοτικότερη από πλευράς ταχύτητας και υλικού αυτήν της ειδικής εντολής.

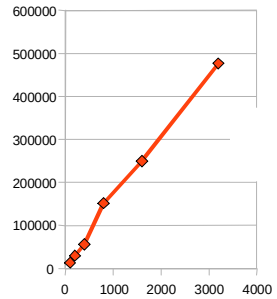
Οι εντολές mul και muli που υλοποιούν την πράξη του πολλαπλασιασμού με ακεραίους αριθμούς χρειάζονται 7 κύκλους αν η μονάδα πολλαπλασιαστή υλοποιηθεί με έτοιμους πολλαπλασιαστές του FPGA [2], αν τώρα λάβουμε υπόψη, ότι χρησιμοποιώντας τις ειδικές εντολές για αριθμητικές πράξεις κινητής υποδιαστολής [8] επιτυγχάνουμε βελτίωση 17 φορές για την πράξη του πολλαπλασιασμού και 20 φορές για την πράξη της πρόσθεσης, αν αυτές υλοποιούνταν από έτοιμες βιβλιοθήκες κώδικα αριθμητικής κινητής υποδιαστολής.



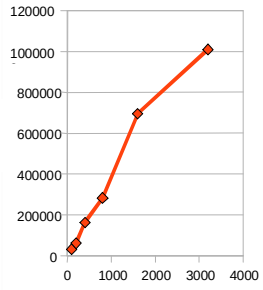
FLOAT



INT

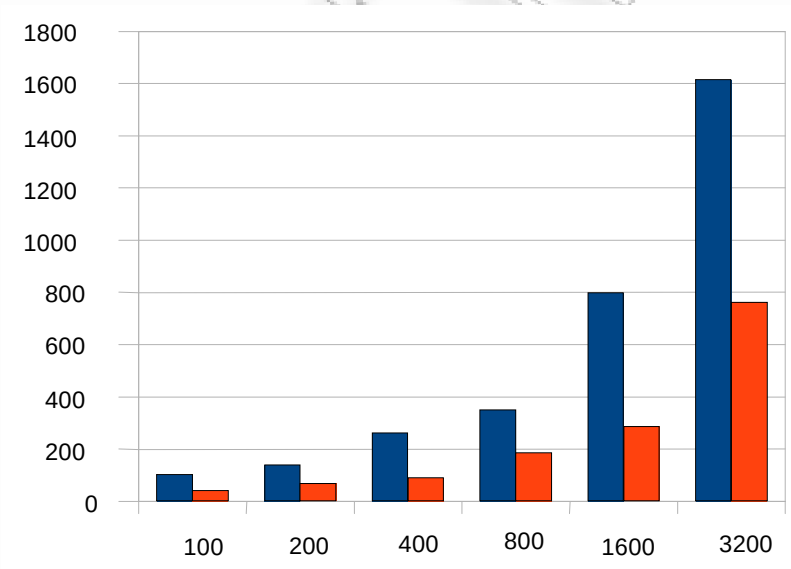


CI



HACC

Στα παραπάνω διαγράμματα παρουσιάζονται ο αριθμός των κύκλων σε σχέση με των αριθμό των δειγμάτων. Στο ακόλουθο διάγραμμα παρουσιάζεται οι επιτάχυνση που επιτυγχάνεται με τις δύο υλοποιήσεις με υλικό, σε σχέση με την υλοποίηση με κώδικα και υλοποίηση της αριθμητικής με ειδικές εντολές κινητής υποδιαστολής της ALTERA. Όπου με μπλε είναι για CI και πορτοκαλί για HACC.



Κεφάλαιο 6

Επίλογος

6.1 Συμπεράσματα

Έχοντας σαν στόχο την επιτάχυνση της εκτέλεσης του αλγορίθμου ψηφιακού φίλτρου πεπερασμένης κρουστικής απόκρισης χρησιμοποιώντας soft-core επεξεργαστή σε προγραμματιζόμενη συσκευή, καταλήξαμε στο συμπέρασμα ότι η χρήση ειδικών εντολών στον επεξεργαστή μας επιφέρει την μεγαλύτερη βελτίωση της απόδοσης επιτρέποντας την ανάπτυξη, ειδικού υψηλής παραλληλίας υλικού, που εξυπηρετεί την εντολή, έχοντας την δυνατότητα ο προγραμματιστής να χρησιμοποιεί τις ειδικές εντολές χωρίς να έχει γνώση για την διασύνδεση αυτού του υλικού με τον επεξεργαστή πράγμα που συμβαίνει στην περίπτωση χρήσης επιταχυντή υλικού.

6.2 Προβλήματα

Πρέπει να τονιστεί και το γεγονός ότι κατά τη διάρκεια υλοποίησης του συστήματος δημιουργήθηκαν διάφορα προβλήματα, από τα οποία αλλά επιλύθηκαν και άλλα ξεπεράστηκαν ή αγνοήθηκαν.
Αυτά τα προβλήματα / δυσκολίες μπορούν να συνοψιστούν στα εξής παρακάτω σημεία:

- Στην διαδικασία προσαρμογής του λειτουργικού συστήματος πραγματικού χρόνου δαπανήθηκε πολύς χρόνος λόγω της ελλιπούς σχετικής τεκμηρίωσης. Το κυριότερο πρόβλημα ήταν η αδυναμία του χρονοδρομολογητή των διεργασιών να ξεκινήσει ύστερα από μία διακοπή χρησιμοποιώντας σημαφόρους, έτσι επιλέγηκε μία εντελώς διαφορετική αρχιτεκτονική του λογισμικού από αυτήν που είχε αρχικά σχεδιαστεί και τελικά πολλές διεργασίες ενσωματώθηκαν σε μία μεγάλη όπου ο έλεγχος και η επιστροφή από την ρουτίνα διακοπών έγινε με ουρές.
- Σχετικά με τους πυρήνες υλικού, έπρεπε να επιλεγούν αυτοί με την κατάλληλη έκδοση για να μπορούν να συνεργαστούν μεταξύ τους, αλλά και με την έκδοση των εργαλείων ανάπτυξης που είχαμε στην διάθεση μας, θέμα που επηρεάζει και τον τρόπο φόρτωσης του λειτουργικού συστήματος.
- Δεν κατέστη δυνατό λόγω του χρόνου που απαιτεί ο CoDec για την εγγραφή και την αναπαραγωγή του ηχητικού σήματος, και έχοντας ως περιορισμό την ελάχιστη συχνότητα δειγματοληψίας που μας επέτρεπε ο οδηγός του σχετικού πυρήνα υλικού, να πετύχουμε επεξεργασία σήματος (είσοδος – έξοδος) σε πραγματικό χρόνο χωρίς παραμόρφωση.
- Λόγο περιορισμού στην χωρητικότητα υλικού του FPGA της κάρτας, ο επιταχυντής υλικού και η προσθήκη ειδικών εντολών γίνεται σε δύο διαφορετικές σχεδιάσεις.
- Επίσης λόγο περιορισμού στην χωρητικότητα υλικού η υλοποίηση με ειδικές εντολές δεν επέτρεπε στο υλικό του ψηφιακού φίλτρου να έχει τον ίδιο αριθμό συντελεστών με τις άλλες προσεγγίσεις.

6.3 Περαιτέρω Προσθήκες – Βελτιώσεις

Το συγκεκριμένο Nios σύστημα δύναται να βελτιωθεί /εμπλουτιστεί με επιπλέον δυνατότητες που δεν έγινε εφικτό να υλοποιηθούν μέσα στο διαθέσιμο χρονικό διάστημα.
Μερικές από τις προτεινόμενες βελτιώσεις είναι οι εξής:

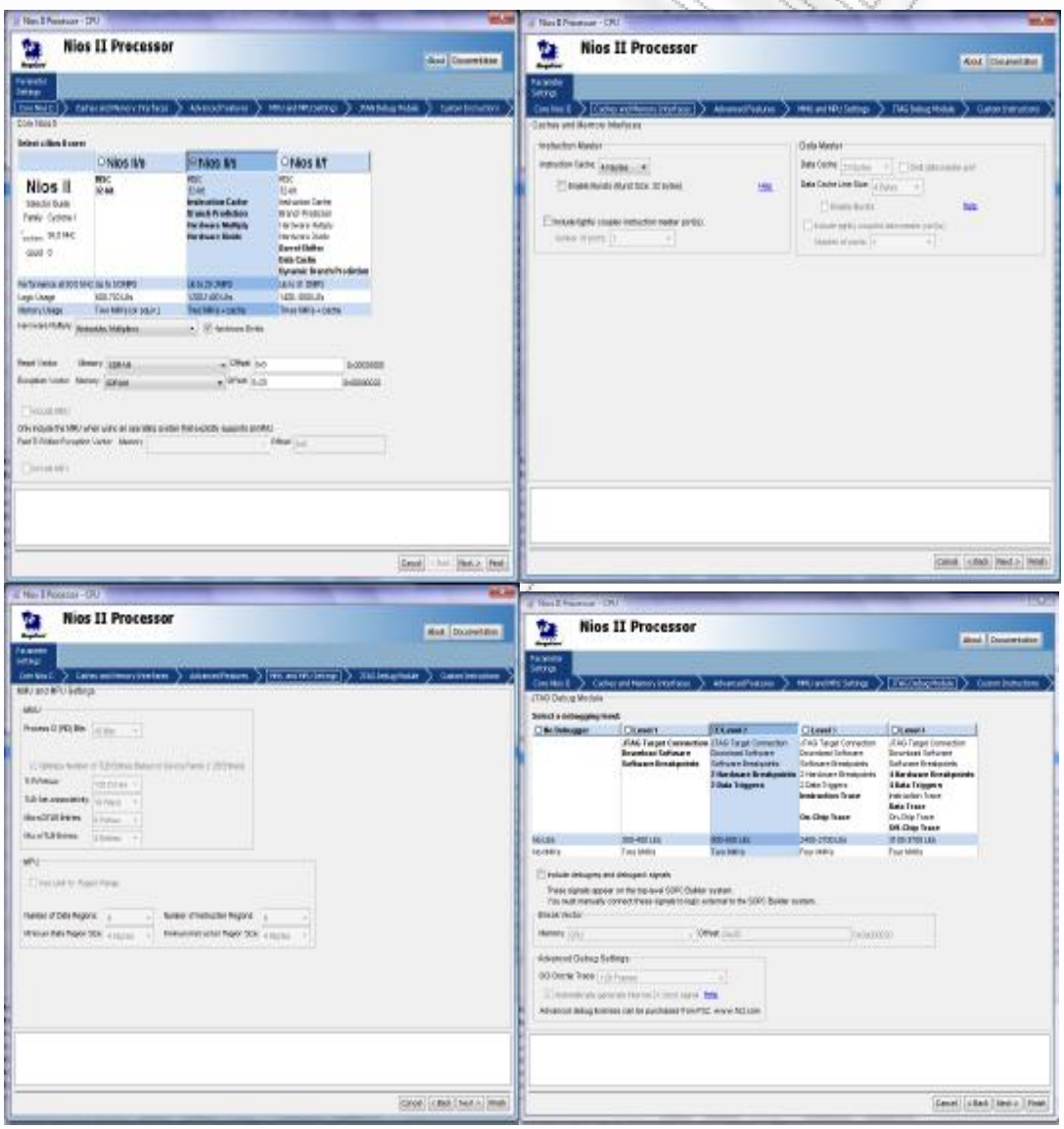
- Δυνατότητα επιλογής της πηγής εισόδου του ήχου (usb, line-in, mic-in, SD-card, ethernet), ώστε να είναι δυνατή η αναπαραγωγή ήχου από εξωτερικές συσκευές όπως mp3 player ή ακόμα και το internet.
- Υλοποίηση ηχητικών εφέ όπως Echo, Distortion, Chorus και άλλα.
- Περαιτέρω αξιοποίηση του VGA Core, όπως εμφάνιση ηχητικών μπάρων για την απεικόνιση της ένταση του ήχου ή εμφάνιση ενός GUI (Graphical User Interface).

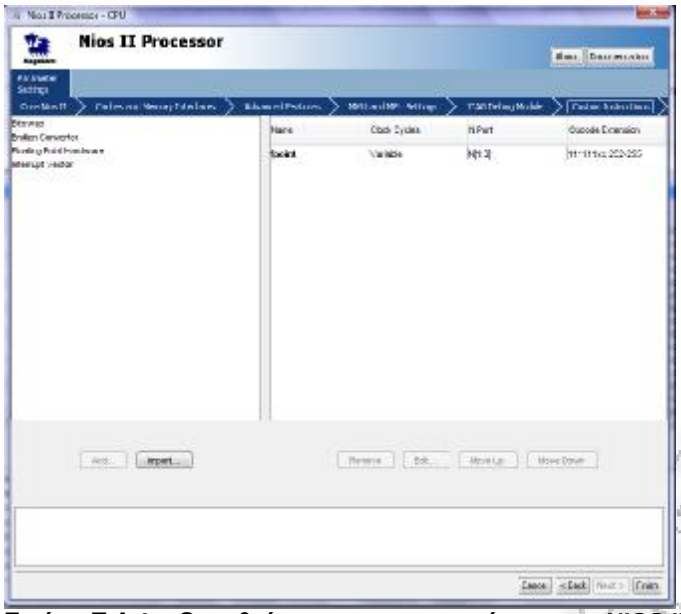
Το επόμενο λογικό βήμα στην έρευνα είναι η ανάπτυξη πολυπύρηνων επεξεργαστών (multi-core processors) προκειμένου την εκμετάλλευση της έμφυτης παραλληλίας των εφαρμογών σε επίπεδο συναρτήσεων ή νημάτων επεξεργασίας (processing threads). Η αρχιτεκτονική των πολυπύρηνων επεξεργαστών που θα χρησιμοποιούνται σε ενσωματωμένα συστήματα θα είναι ετερογενής ώστε να είναι εξειδικεύσιμη στις εφαρμογές του ενδιαφέροντος. Σημαντικό ρόλο θα έχει η ανάπτυξη και αυτοματοποίηση τεχνικών για τη διερεύνηση καταμερισμού εργασίας ανάμεσα στους επεξεργαστές. Επίσης ένα ακόμη ζήτημα είναι η διερεύνηση της αρχιτεκτονικής διασύνδεσης των μονάδων (δίαυλος ή δίκτυο-σε-ολοκληρωμένο) και ιδιαίτερα για τη δεύτερη περίπτωση, ο καθορισμός της δομής του πακέτου δεδομένων/ελέγχου και της πολιτικής μεταγωγής. Το δεύτερο σημαντικό ζήτημα στην ανάπτυξη πολυπυρηνικών SoC είναι η οργάνωση μνήμης, η οποία θα πρέπει να μπορεί να αποδώσει μεγάλο εύρος προσπέλασης για τις επεξεργαστικές μονάδες.
Στα πλαίσια της έρευνας, ενδιαφέρον παρουσιάζουν και οι μετασχηματισμοί κώδικα υψηλού επιπέδου προκειμένου τη διαμόρφωση του κώδικα μιας εφαρμογής ώστε να αναδεικνύει πλεονεκτικές χρήσεις προκαθορισμένου υλικού, είτε αυτό έχει σχεδιαστεί εμπειρικά είτε έχε παραχθεί αυτόματα (με γέννηση ειδικών εντολών) και καταχωρηθεί σε βιβλιοθήκη λειτουργικών μονάδων. Ακόμη, αν και κάποιες φορές είναι επιβεβλημένη η ανάπτυξη μεταγλωττιστή ο οποίος να εκμεταλλεύεται τα ιδιαίτερα χαρακτηριστικά των επεξεργαστών ειδικού σκοπού. Οι επαναστοχεύσιμοι μεταγλωττιστές αποτελούν ερευνητικό πεδίο με έντονο βιομηχανικό ενδιαφέρον. Ιδανικά, μεταγλωττιστής, λοιπά εργαλεία ανάπτυξης εφαρμογών, προσομοιωτής και το συνθέσιμο μοντέλο του επεξεργαστή, πρέπει να παράγονται από ένα κοινό μοντέλο αναφοράς, πιθανότατα σε κάποια γλώσσα περιγραφής συστήματος σε επίπεδο Transaction Level Modelling (TLM).
Γενικά, το πεδίο των ενσωματωμένων συστημάτων και ειδικά τα επιμέρους ερευνητικά θέματα που διέπουν το σχεδιασμό επεξεργαστών ειδικού σκοπού παρουσιάζουν έντονο ενδιαφέρον για ενασχόληση στο άμεσο μέλλον.

Παράρτημα Α – Ρυθμίσεις του συστήματος

Π.Α.1. Ρυθμίσεις του πυρήνα υλικού «Nios II Core»

Το Core του Nios II (*cpu_0*) που χρησιμοποιήθηκε, όπως αναλύθηκε και στο κεφάλαιο 2 είναι το Nios II/s (standard). Για τους πολλαπλασιασμούς χρησιμοποιούνται τα embedded multipliers blocks, ενώ ο reset vector και ο exception vector τοποθετούνται στην SDRAM μνήμη του συστήματος. Επιπλέον, το μέγεθος της κρυφής μνήμης εντολών (InstructionCache) επιλέγεται να είναι 4 Kbytes, ενώ το μέγεθος της κρυφής μνήμης δεδομένων (Data Cache) είναι προκαθορισμένο και ίσο με 2 Kbytes.Ανκαι υπάρχει η δυνατότητα για προσθήκη MMU η οποία μπορεί να χρησιμοποιηθεί από ένα λειτουργικό σύστημα πχ. Linux, το λειτουργικό σύστημα που έχουμε χρησιμοποιήσει, το FreeRtos δεν απαιτεί την ύπαρξη MMU. Στις ρυθμίσεις για την μονάδα αποσφαλμάτωσης JTAG επιλέγεται το επίπεδο αποσφαλμάτωσης 2, ενώ χρησιμοποιούνται / ορίζονται πρόσθετες εντολές στο πεδίο Custom Instructions για floating point αριθμητική. Οι παραπάνω ρυθμίσεις παρουσιάζονται στην εικόνα Π.Α.1.

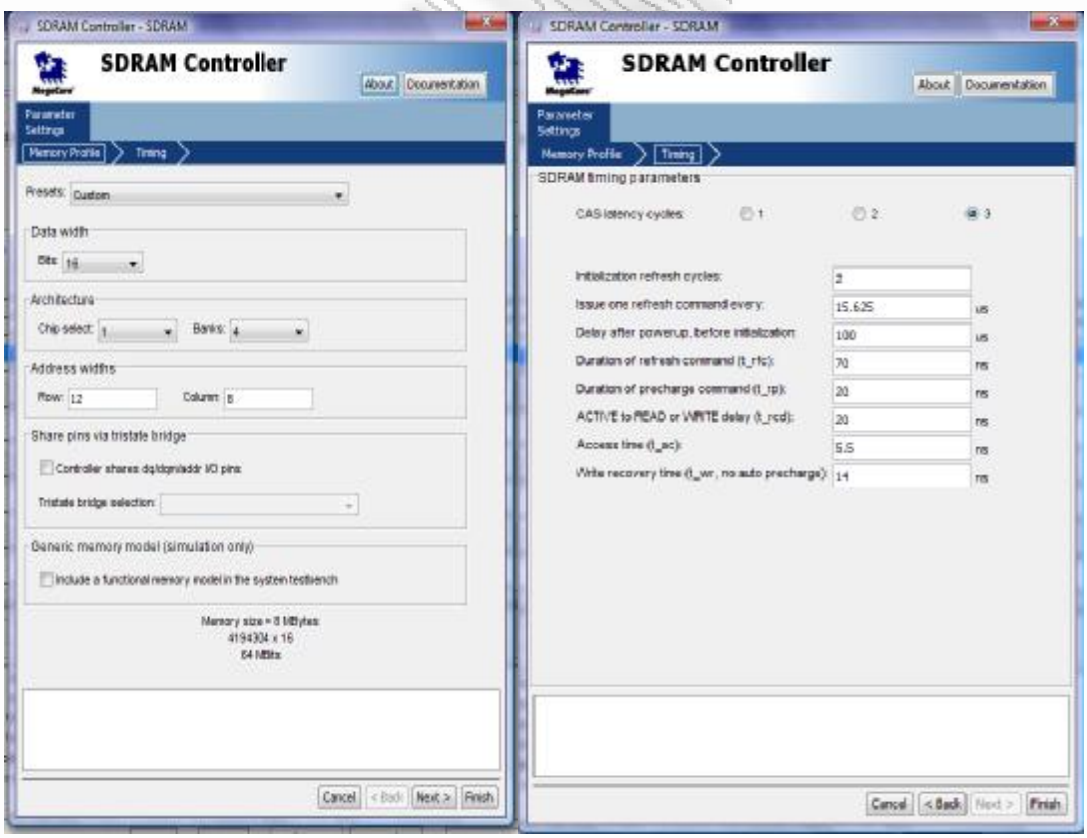




Εικόνα Π.Α.1 – Οι ρυθμίσεις παραμετροποίησης του NIOS II Core

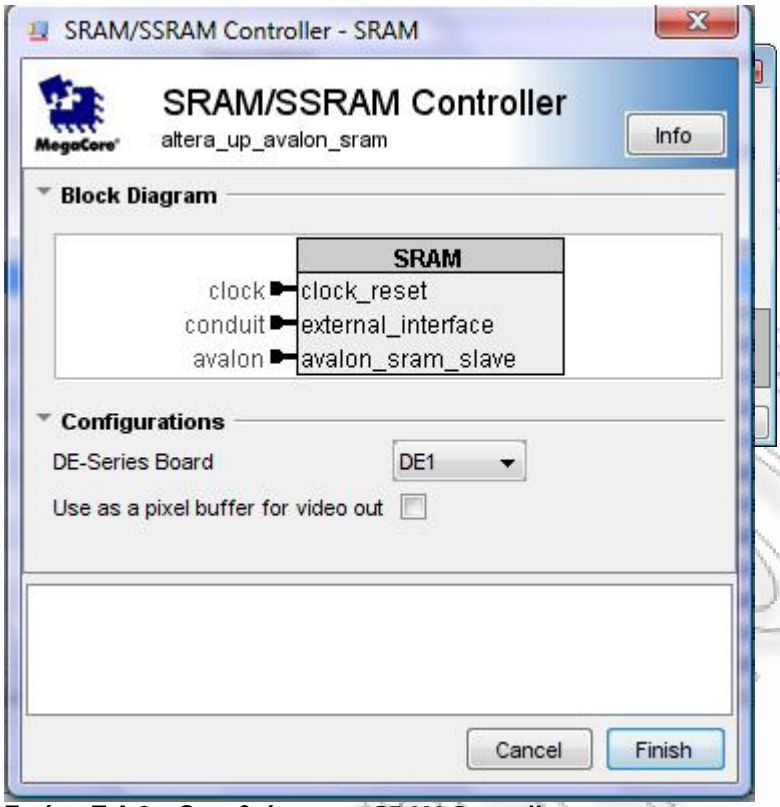
Π.Α.2 Οι ρυθμίσεις της μνήμης του συστήματος

Η κύρια μνήμη του συστήματος επιλέγεται η SDRAM με μέγεθος 8MB με τις ακόλουθες ρυθμίσεις:



Εικόνα Π.Α.2 - Οι ρυθμίσεις της μνήμης του συστήματος

Η SRAM επιλέγεται σαν μνήμη γραφικών και χρησιμοποιείτε από το Pixel Buffer DMA Controller Core. Το μέγεθος που μας παρέχει η κάρτα είναι 512Kbytes. Έχουμε τις ακόλουθες ρυθμίσεις:



Εικόνα Π.Α.3 – Οι ρυθμίσεις του SRAM Controller

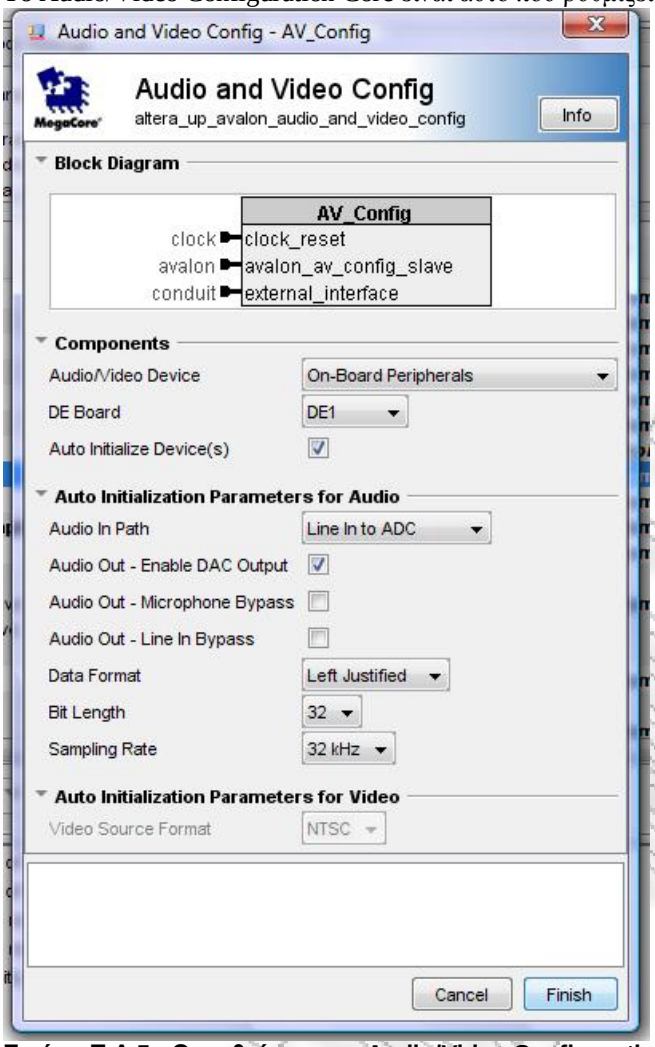
Π.Α.3 Ρυθμίσεις του πυρήνα υλικού «Audio Core»

Οι ρυθμίσεις για το Audio Core (*audio_0*) είναι τρεις, αφού οι επιμέρους λεπτομέρειες της λειτουργίας του καθορίζονται από το Audio/Video Configuration Core. Έτσι, ο CODEC επιλέγεται να λειτουργεί ως είσοδος-έξοδος ηχητικών δεδομένων και το εύρος των δεδομένων αυτών να είναι 32-bit.

Εικόνα Π.Α.4 – Οι ρυθμίσεις του Audio Core

Π.Α.4 **Ρυθμίσεις του πυρήνα υλικού** «Audio/Video Configuration Core»

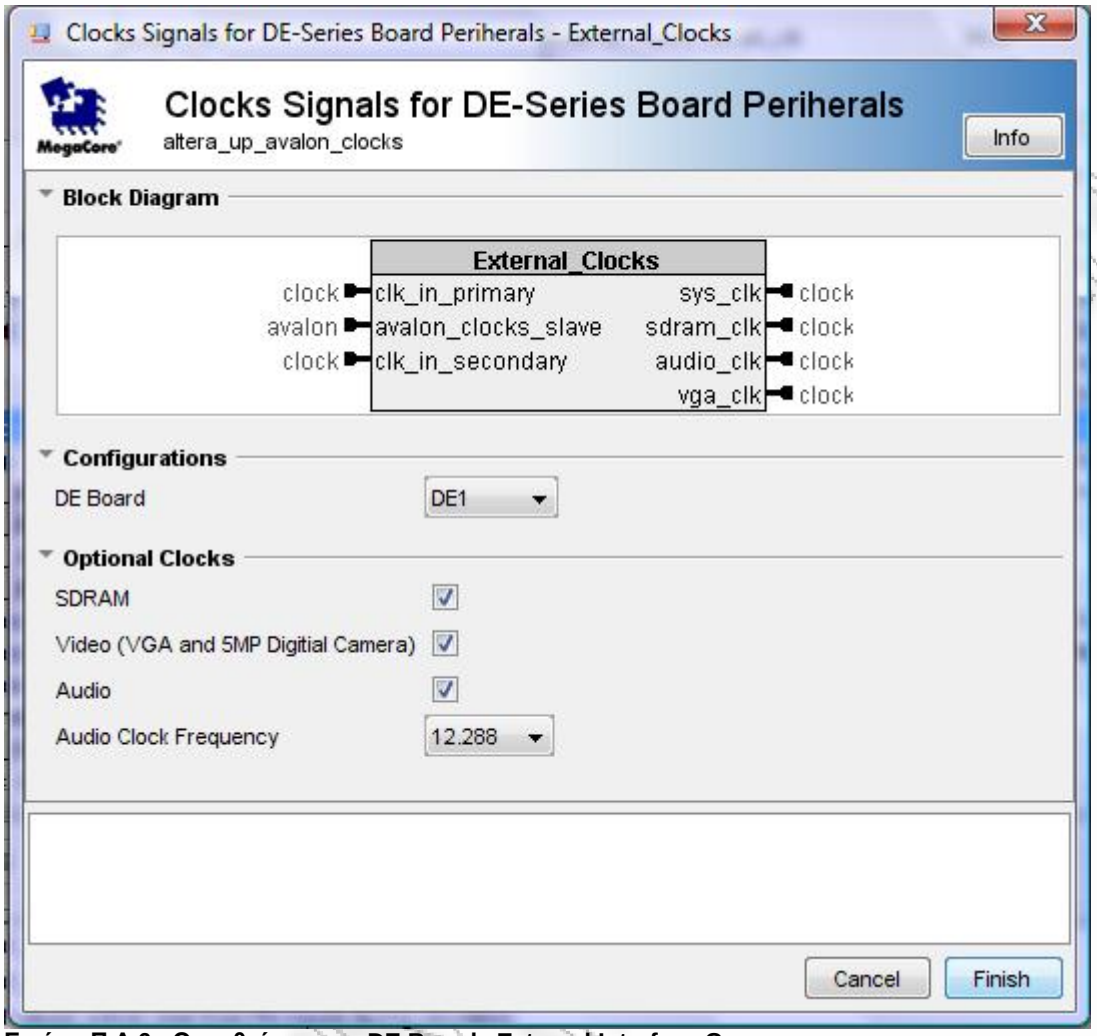
Το Audio/Video Configuration Core είναι αυτό που ρυθμίζει την όλη λειτουργία του Audio CODEC.



Εικόνα Π.Α.5 - Οι ρυθμίσεις του Audio/Video Configuration Core

Π.Α.5. **Ρυθμίσεις του πυρήνα υλικού** «DE Boards External Interface Core»

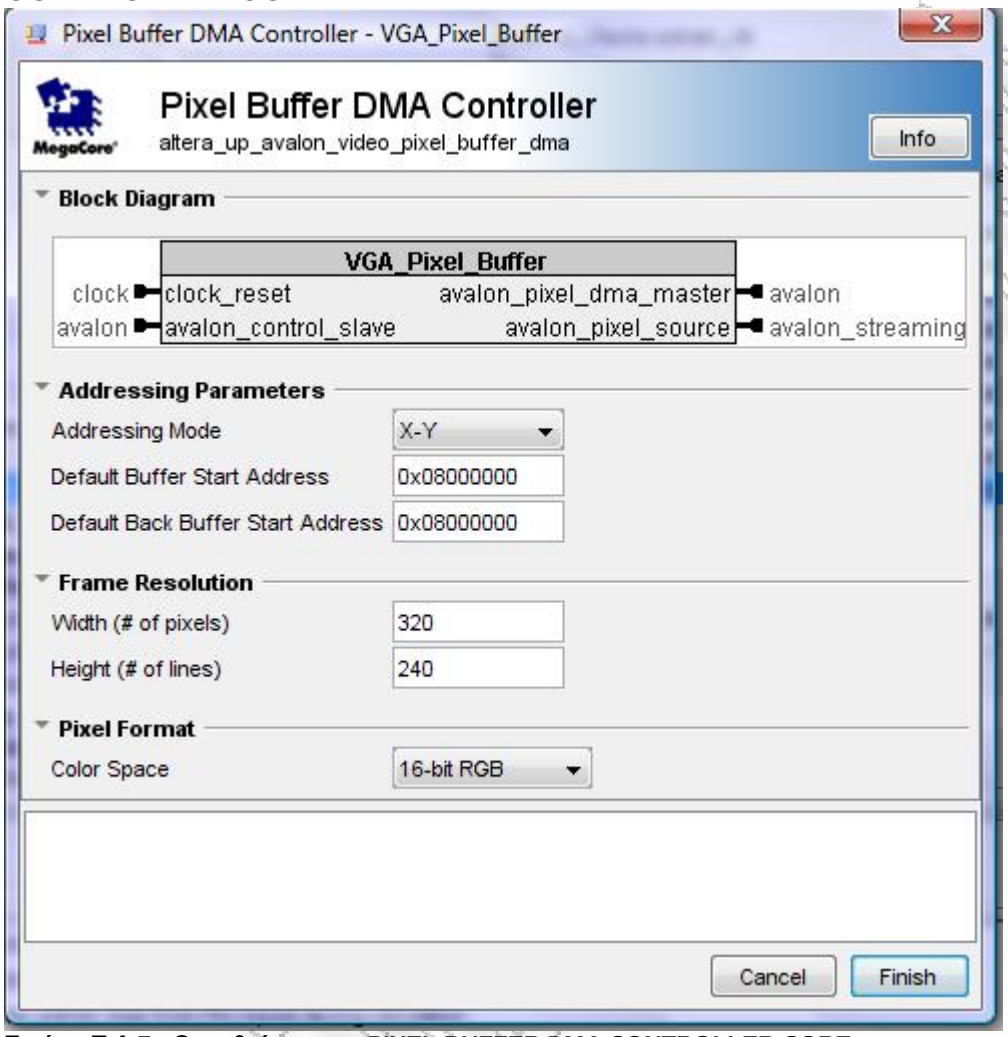
Το DE1 Boards External InterfaceCore παράγει τα κατάλληλα σήματα χρονισμού που είναι αναγκαία για τη λειτουργία του Audio CODEC, του VGA DAC και του SDRAM Controller. Στο συγκεκριμένο Nios II σύστημα είναι αναγκαίο το core αυτό να παράγει και τα τρία σήματα ρολογιού, αφού χρησιμοποιούνται και Audio CODEC, και ο VGA DAC και ο SDRAM Controller της DE1 κάρτας. Επιπλέον, σύμφωνα με τα παραπάνω, η συχνότητα για το ρολόι που χρονίζει-συντονίζει τον Audio CODEC επιλέγεται να είναι ίση με 12.288 MHz.



Εικόνα Π.Α.6 - Οι ρυθμίσεις του DE Boards External Interface Core

Για τα VGA Cores έχουμε τις ακόλουθες ρυθμίσεις:

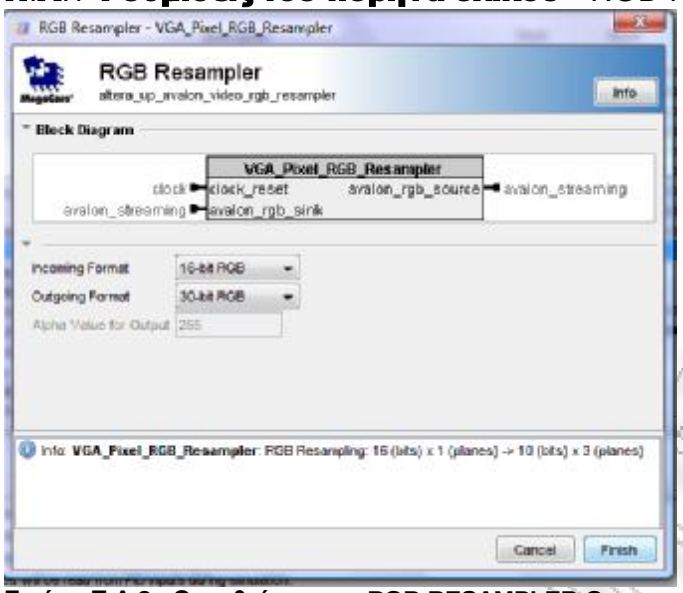
Π.Α.6 **Ρυθμίσεις του πυρήνα υλικού** «PIXEL BUFFER DMA CONTROLLER CORE»



Εικόνα Π.Α.7 - Οι ρυθμίσεις του PIXEL BUFFER DMA CONTROLLER CORE

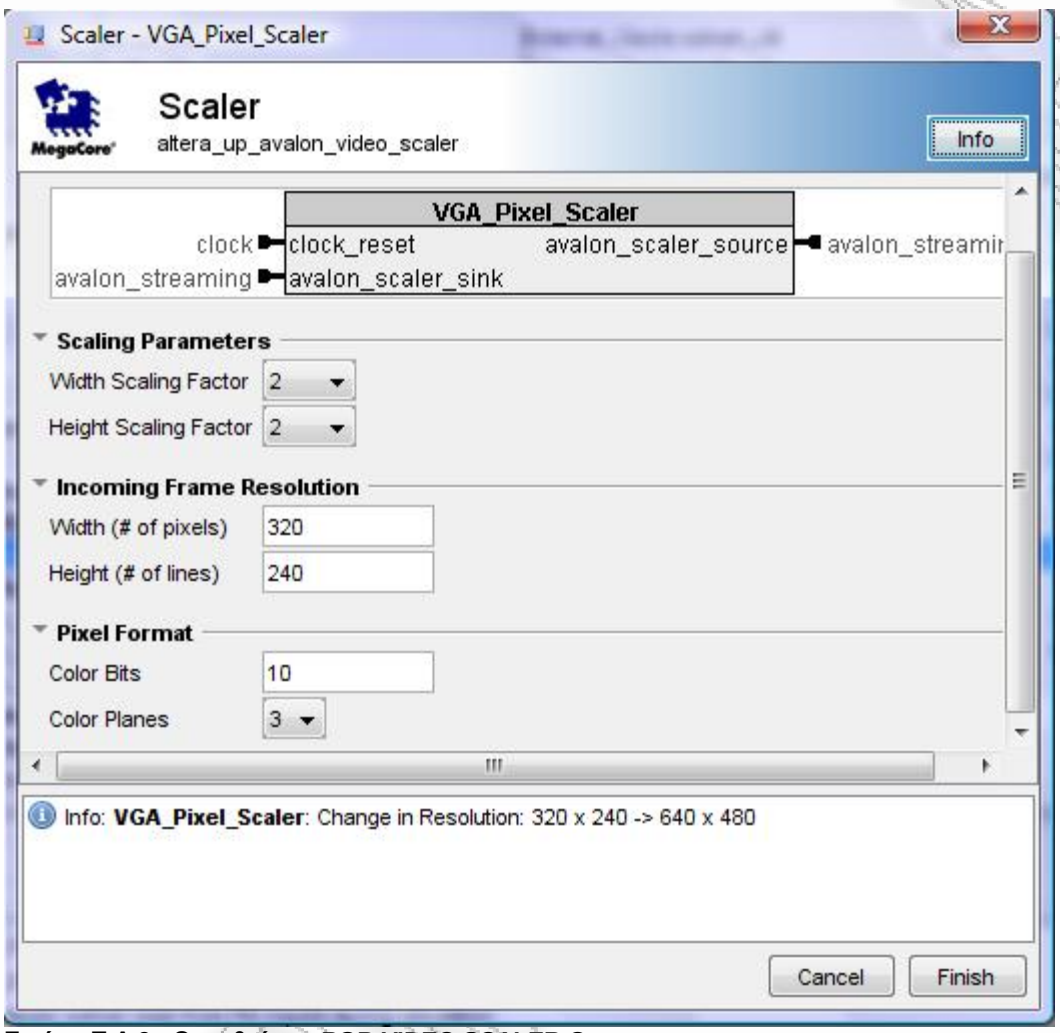
όπου σαν buffer start address επιλέγεται η διεύθυνση έναρξης της SRAM.

Π.Α.7 **Ρυθμίσεις του πυρήνα υλικού** «RGB RESAMPLER Core»



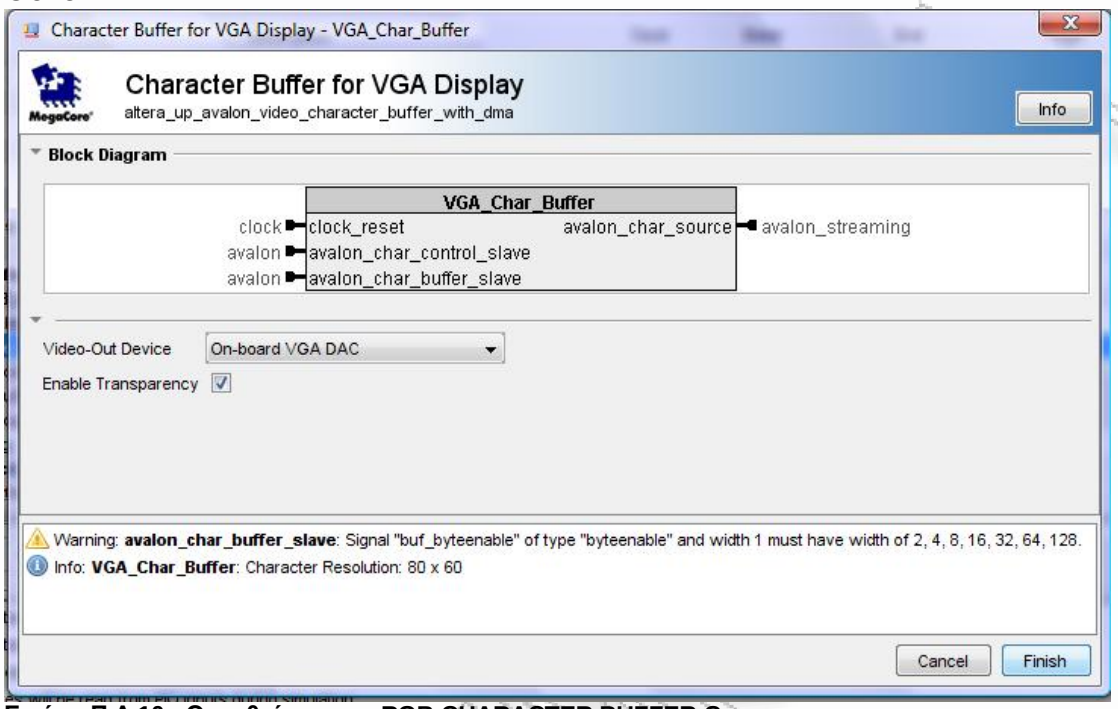
Εικόνα Π.Α.8 - Οι ρυθμίσεις του RGB RESAMPLER Core

Π.Α.8 **Ρυθμίσεις του πυρήνα υλικού** «RGB VIDEO SCALER Core»



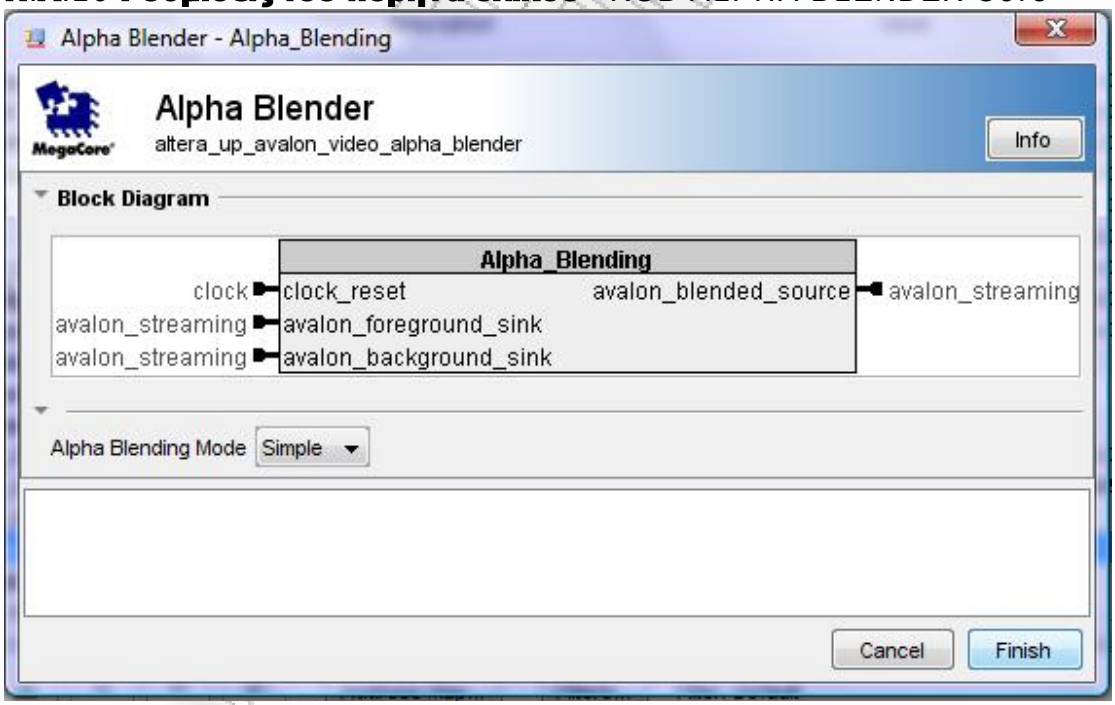
Εικόνα Π.Α.9 - Οι ρυθμίσεις RGB VIDEO SCALER Core

Π.Α.9 **Ρυθμίσεις του πυρήνα υλικού** «RGB CHARACTER BUFFER Core»



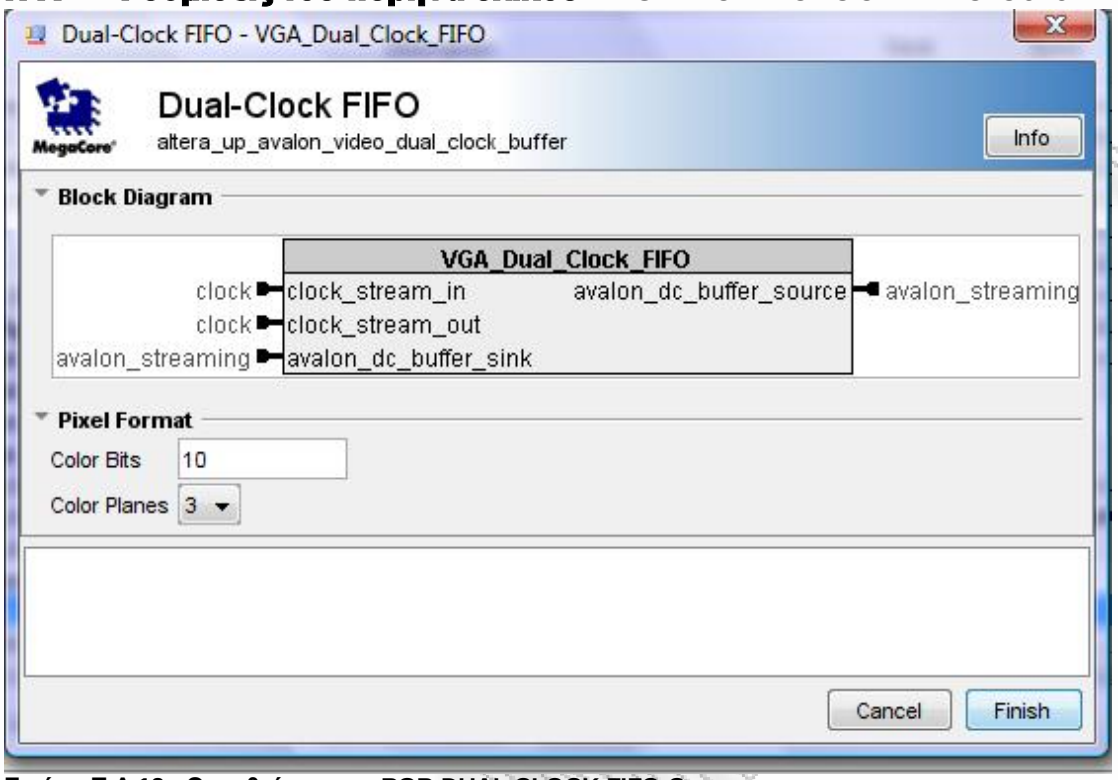
Εικόνα Π.Α.10 - Οι ρυθμίσεις του RGB CHARACTER BUFFER Core

Π.Α.10 **Ρυθμίσεις του πυρήνα υλικού** «RGB ALPHA BLENDER Core»



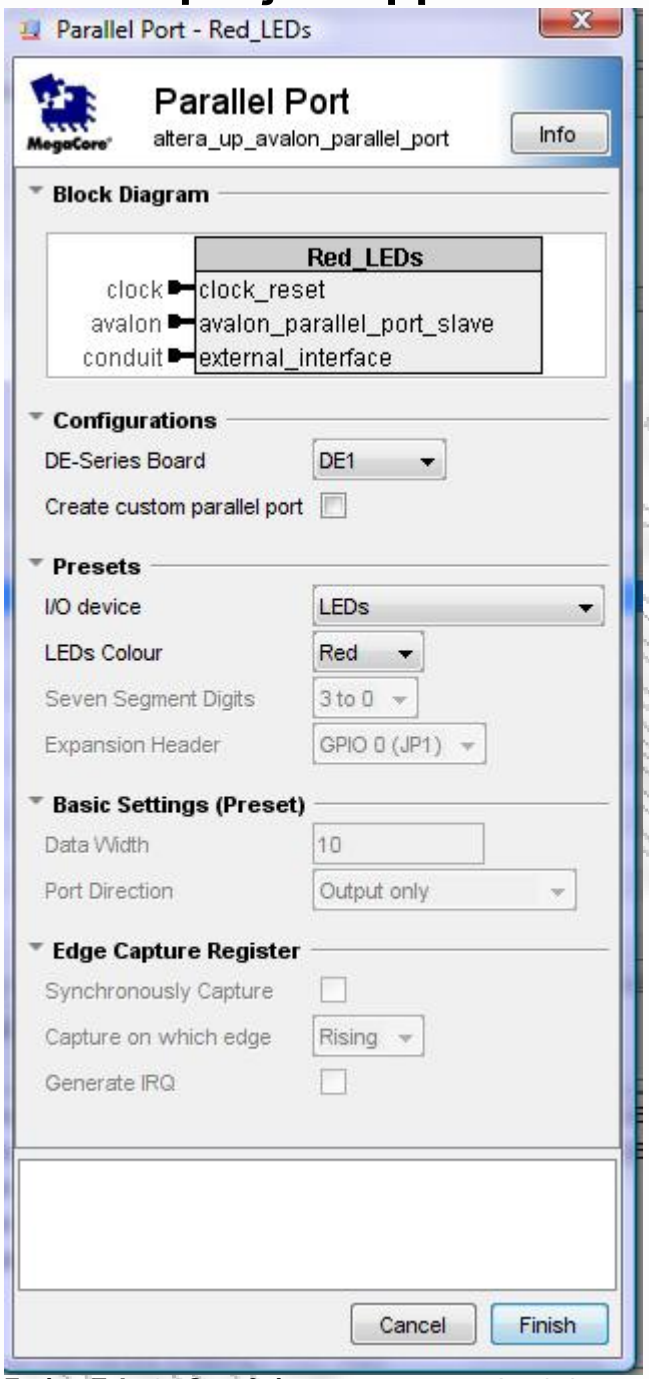
Εικόνα Π.Α.11 - Οι ρυθμίσεις του RGB ALPHA BLENDER Core

Π.Α.11 **Ρυθμίσεις του πυρήνα υλικού** «RGB DUAL CLOCK FIFO Core»



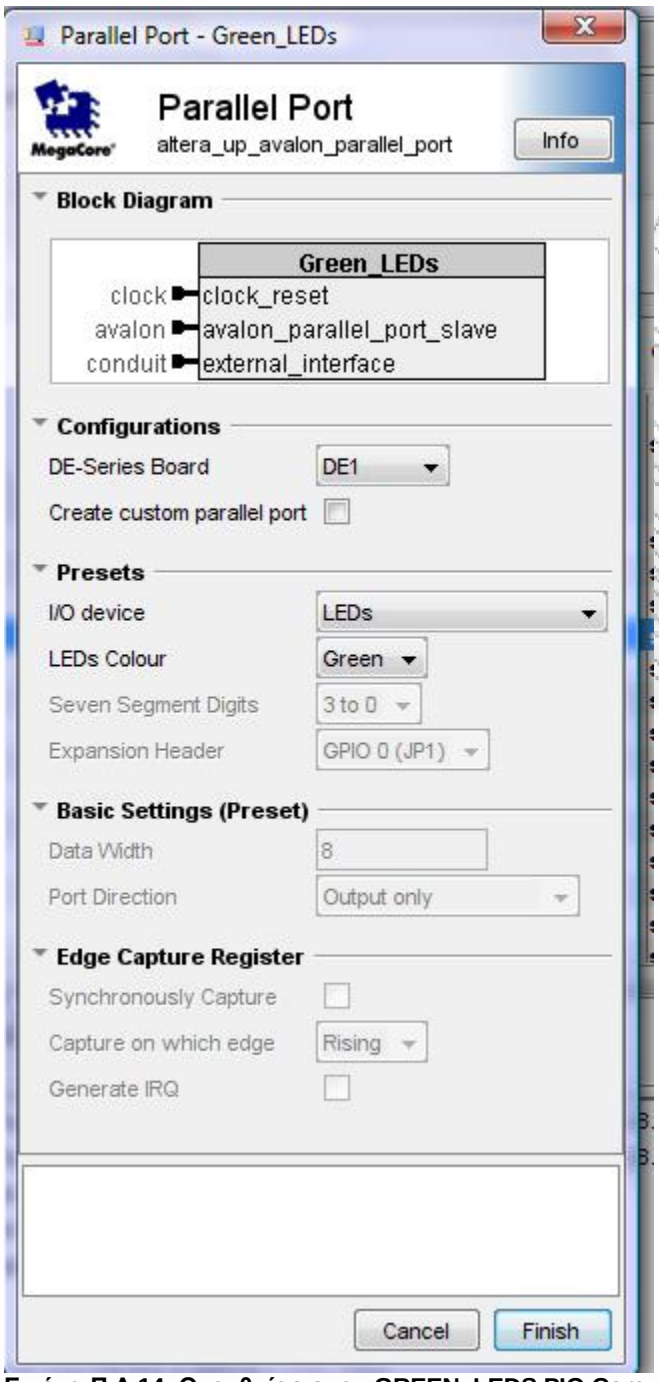
Εικόνα Π.Α.12 - Οι ρυθμίσεις του RGB DUAL CLOCK FIFO Core

Π.Α.12 **Ρυθμίσεις του πυρήνα υλικού** «RED_LEDS PIO Core»



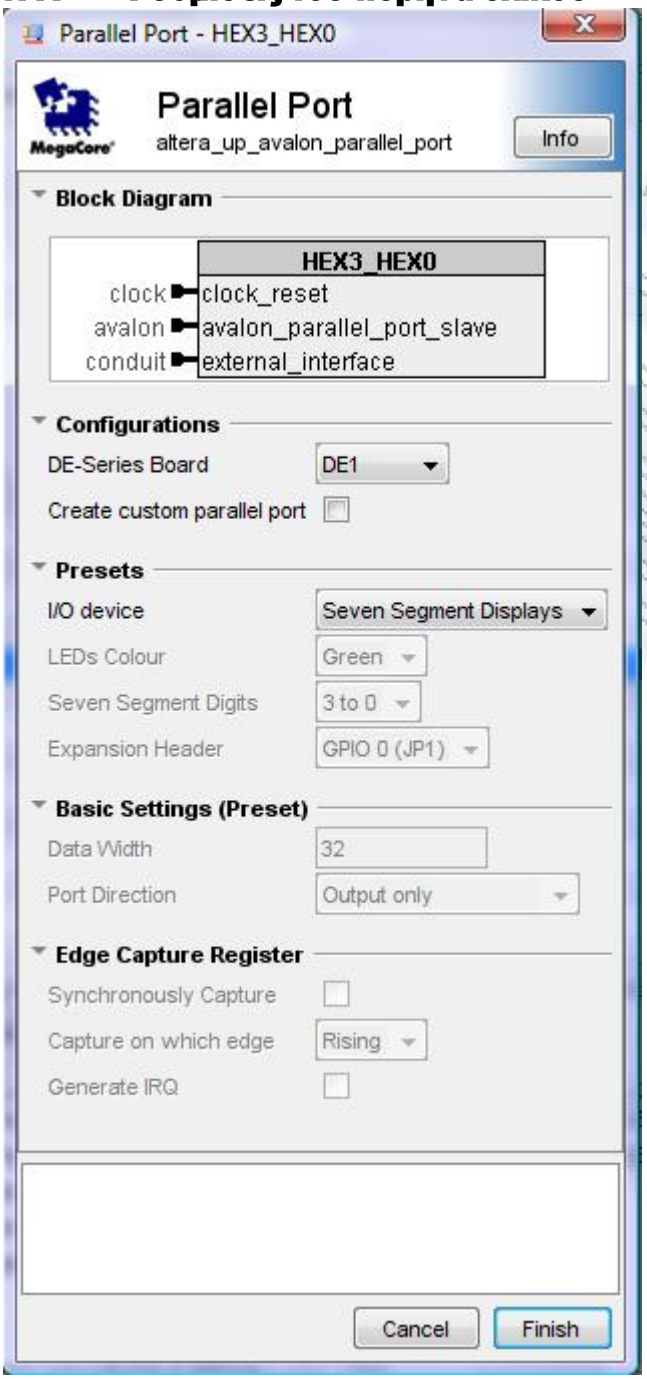
Εικόνα Π.Α.13 - Οι ρυθμίσεις του RED_LEDS PIO Core

Π.Α.13 **Ρυθμίσεις του πυρήνα υλικού** «GREEN_LEDS PIO Core»



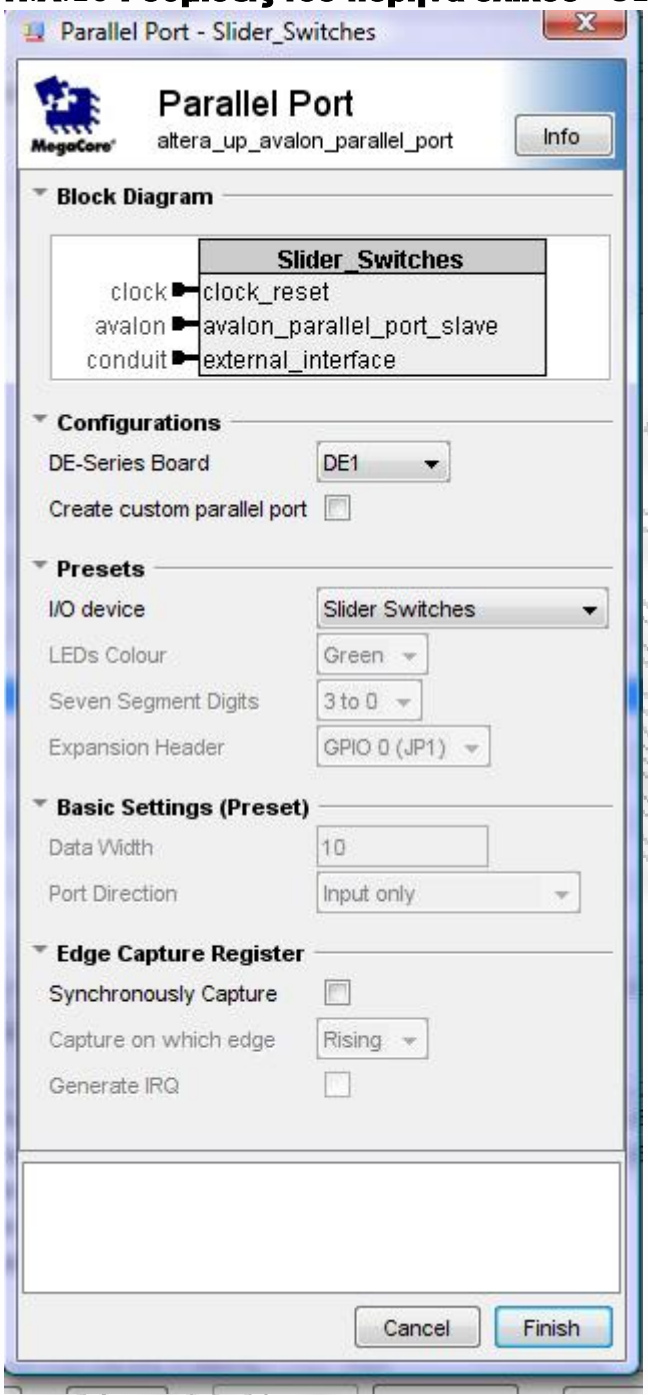
Εικόνα Π.Α.14- Οι ρυθμίσεις του GREEN_LEDS PIO Core

Π.Α.14 **Ρυθμίσεις του πυρήνα υλικού** «HEX3..HEX0 PIO Core»



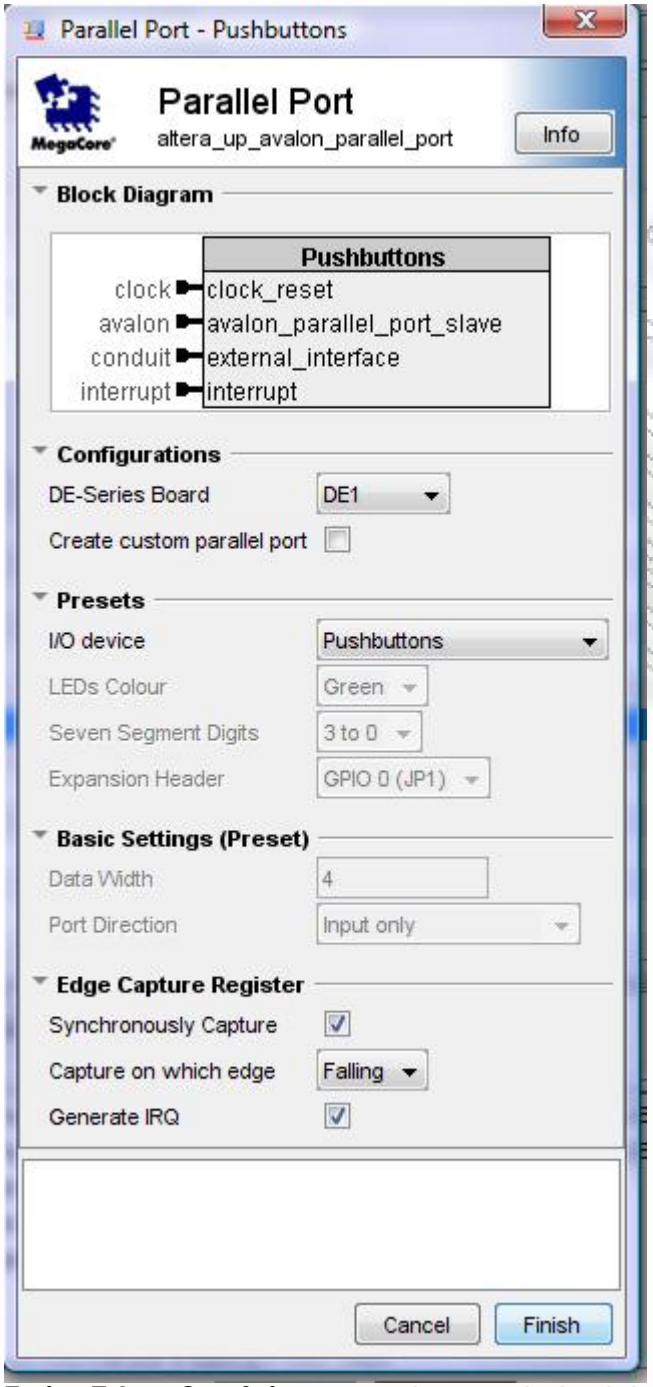
Εικόνα Π.Α.15 - Οι ρυθμίσεις του HEX3..HEX0 PIO Core

Π.Α.15 **Ρυθμίσεις του πυρήνα υλικού** «SLIDER_SWITCHES PIO Core»



Εικόνα Π.Α.16 - Οι ρυθμίσεις του SLIDER_SWITCHES PIO Core

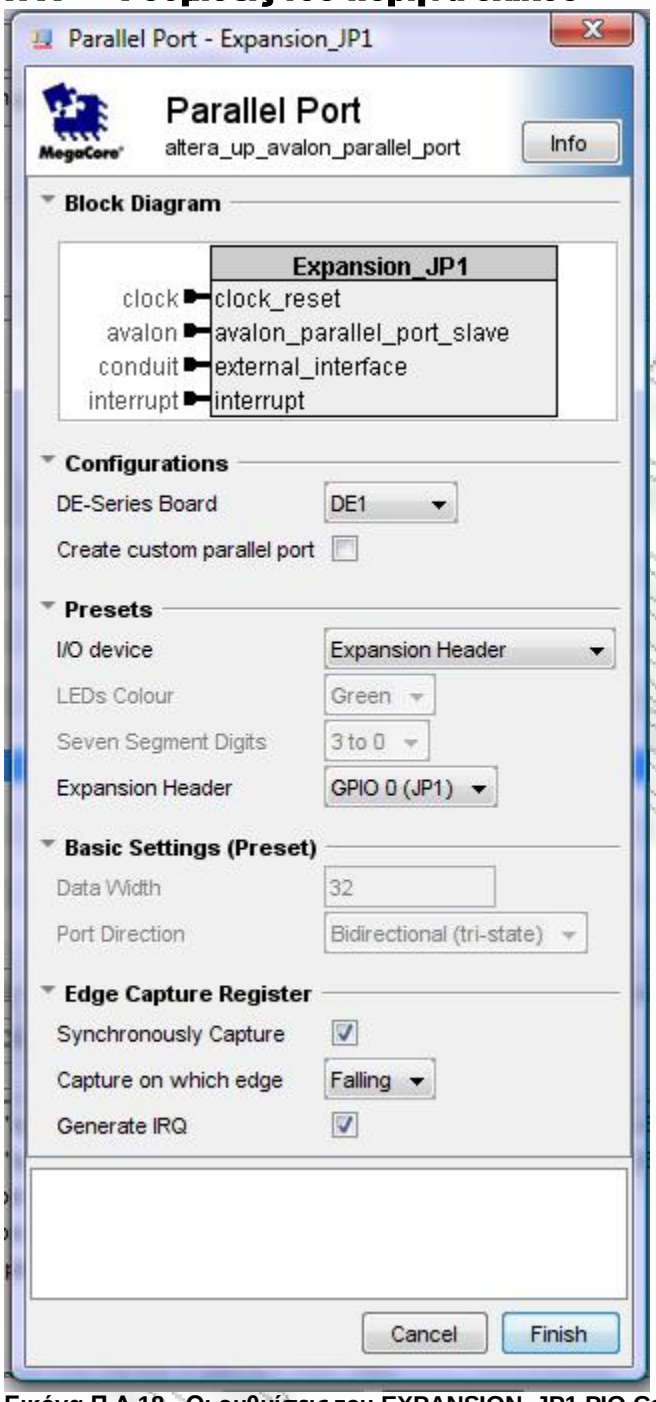
Π.Α.16 **Ρυθμίσεις του πυρήνα υλικού** «PUSH_BUTTONS PIO Core»



Εικόνα Π.Α.17- Οι ρυθμίσεις του PUSH_BUTTONS PIO Core

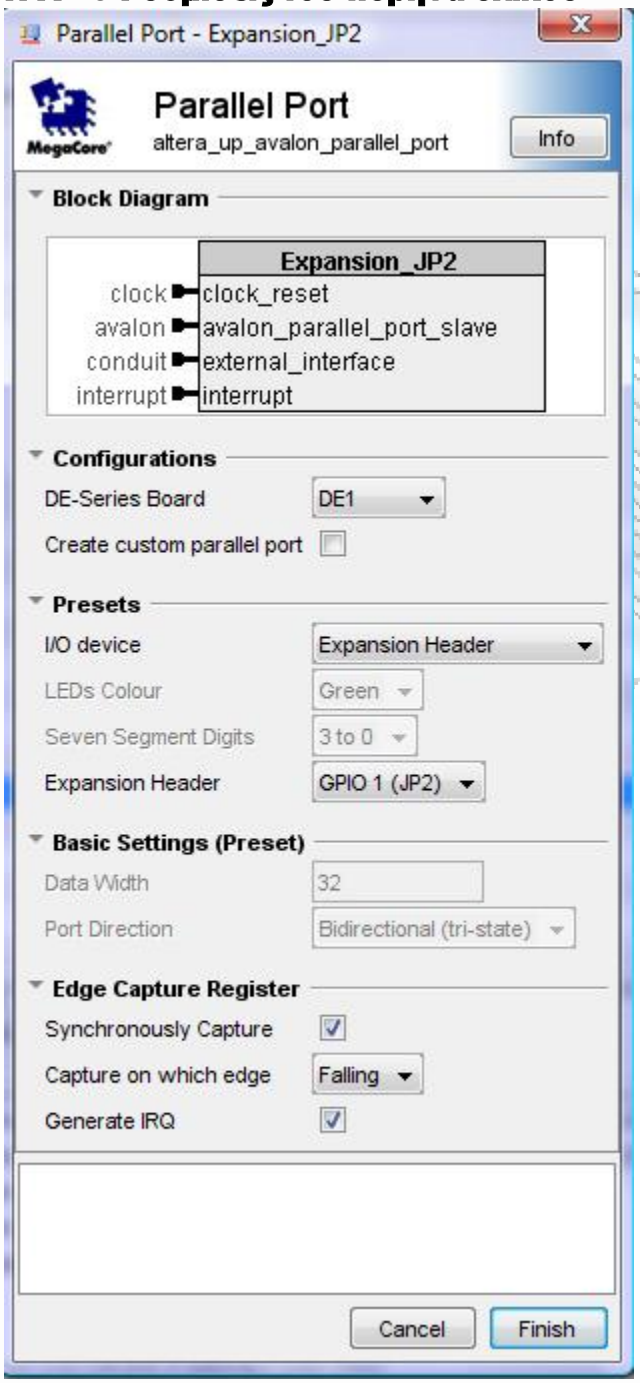
όπου έχουμε δηλώσει ότι επιθυμούμε να δημιουργούνται διακοπές.

Π.Α.17 **Ρυθμίσεις του πυρήνα υλικού** «EXPANSION_JP1 PIO Core»



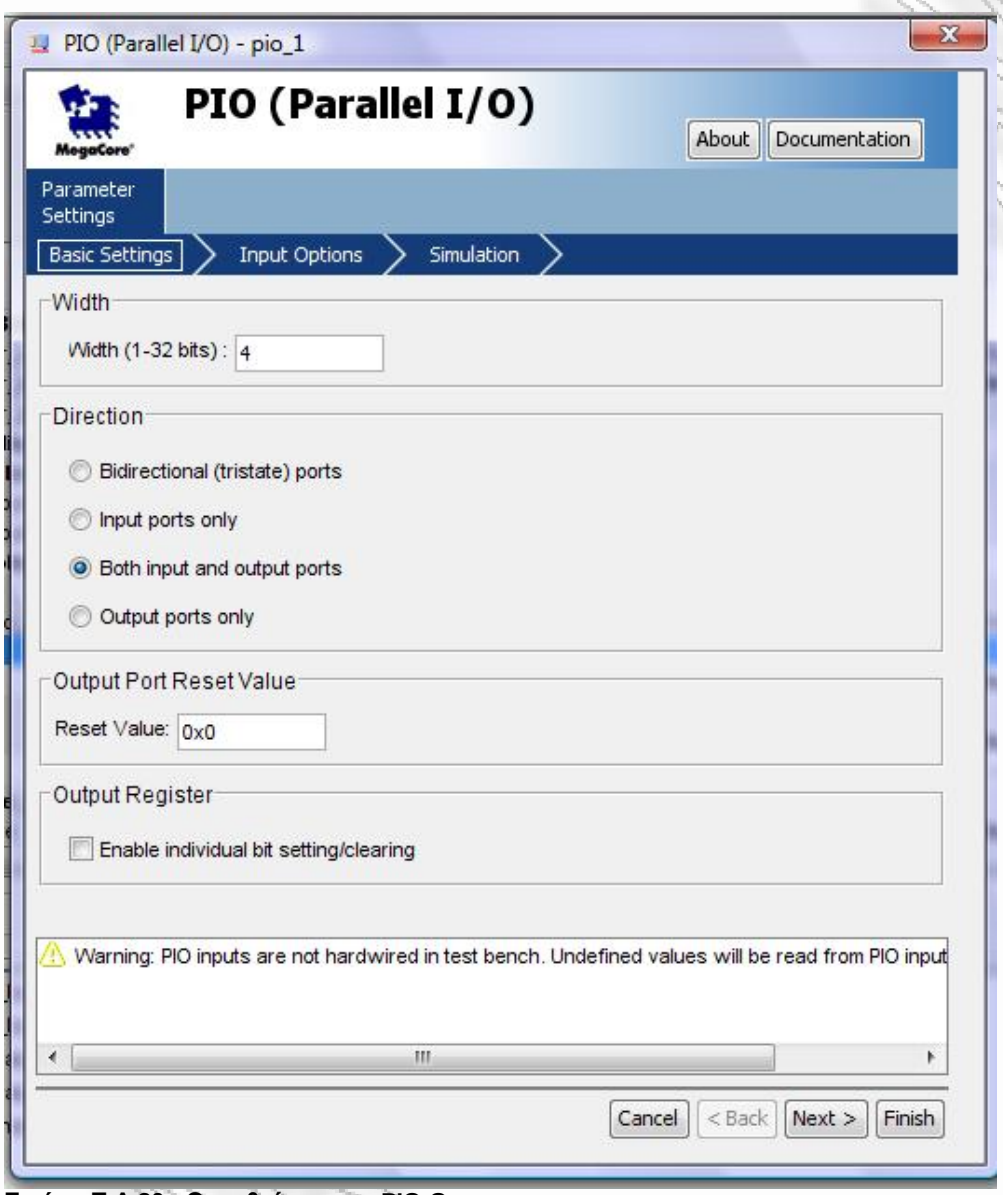
Εικόνα Π.Α.18 - Οι ρυθμίσεις του EXPANSION_JP1 PIO Core

Π.Α.18 **Ρυθμίσεις του πυρήνα υλικού** «EXPANSION_JP2 PIO Core»

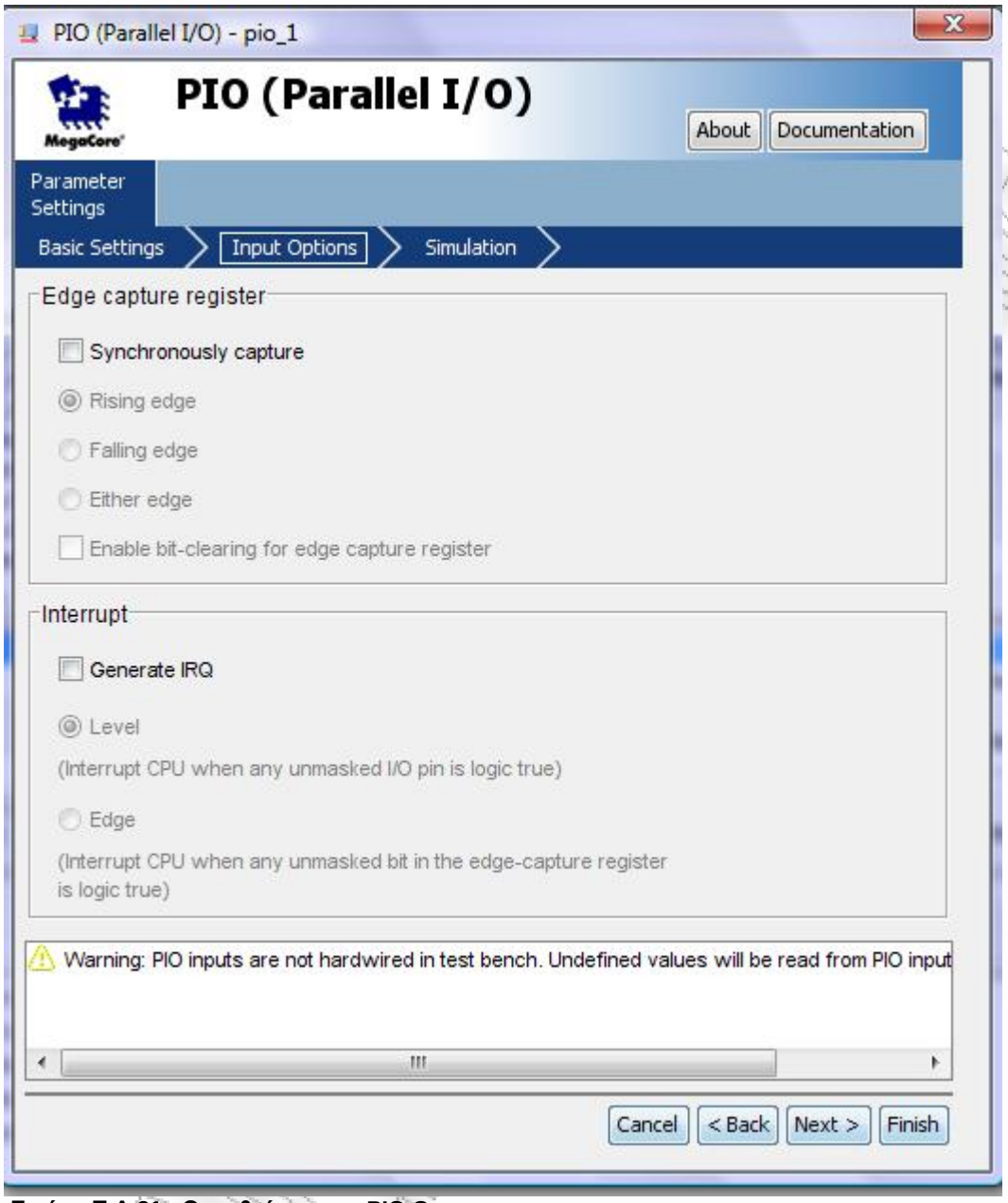


Εικόνα Π.Α.19 - Οι ρυθμίσεις του EXPANSION_JP2 PIO Core

Π.Α.19 **Ρυθμίσεις του πυρήνα υλικού** «PIO1 Core»

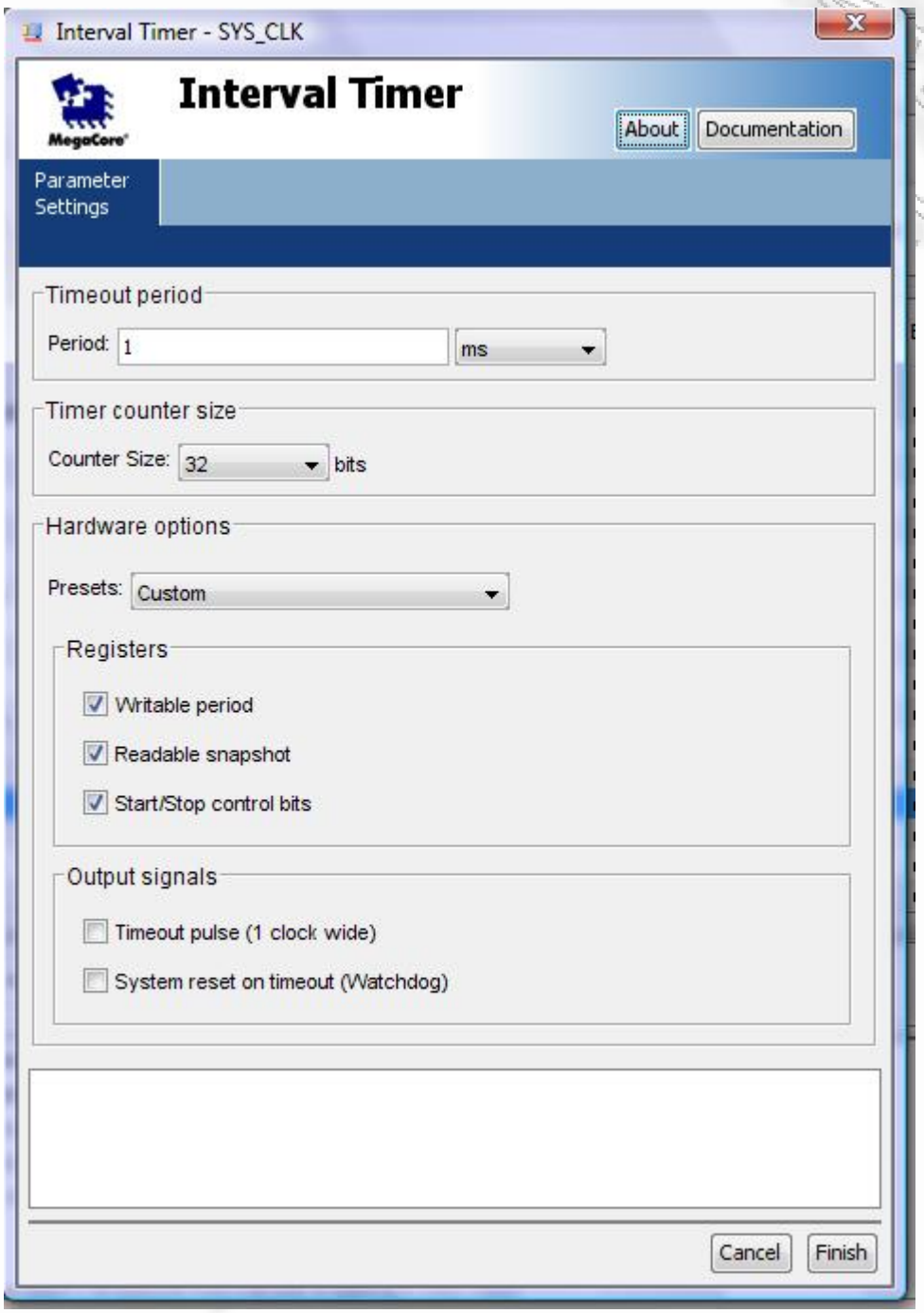


Εικόνα Π.Α.20 - Οι ρυθμίσεις του PIO Core



Εικόνα Π.Α.21 - Οι ρυθμίσεις του PIO Core

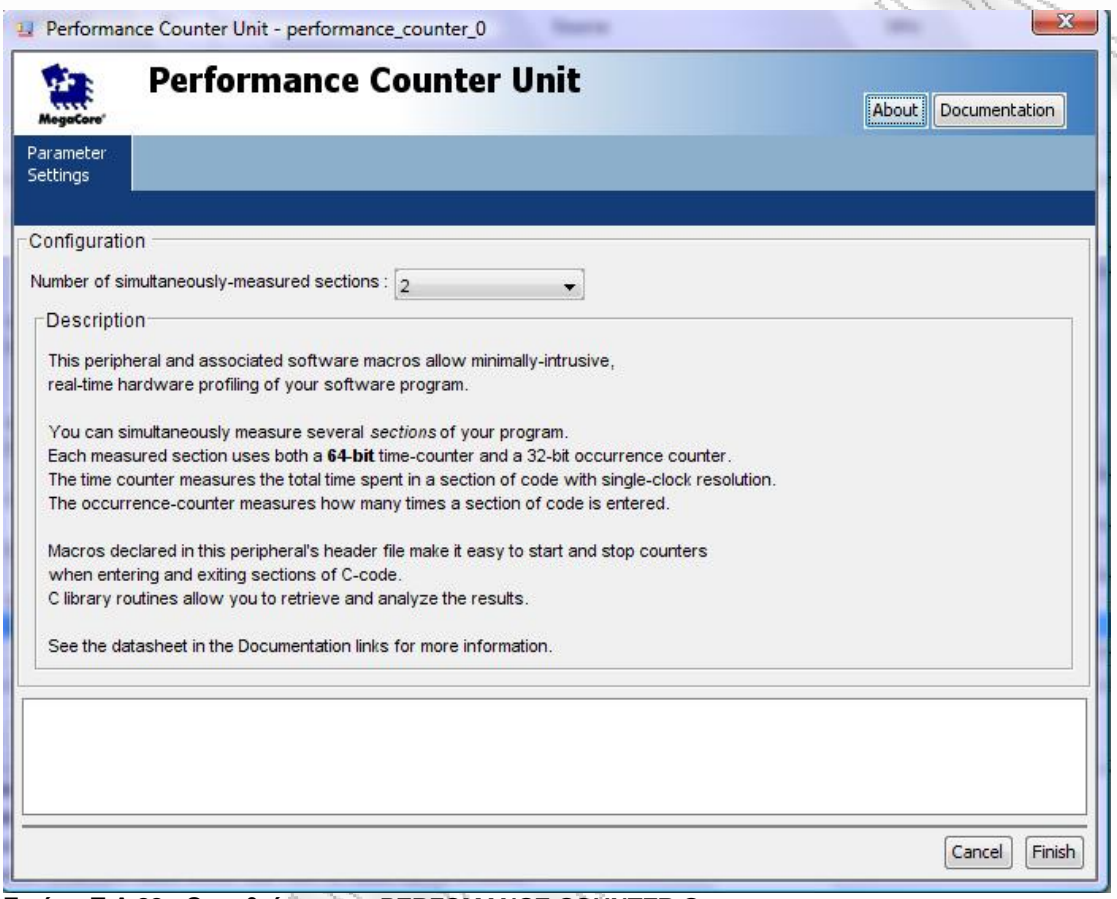
Π.Α.20 **Ρυθμίσεις του πυρήνα υλικού** «SYS CLK TIMER Core»



Εικόνα Π.Α.22 - Οι ρυθμίσεις του SYS CLK TIMER Core

όπου επιλέγεται timeout period 1ms δηλαδή όσο ο tick timer του FreeRtos.

Π.Α.21 **Ρυθμίσεις του πυρήνα υλικού** «PERFOMANCE COUNTER Core»



Εικόνα Π.Α.23 - Οι ρυθμίσεις του PERFOMANCE COUNTER Core

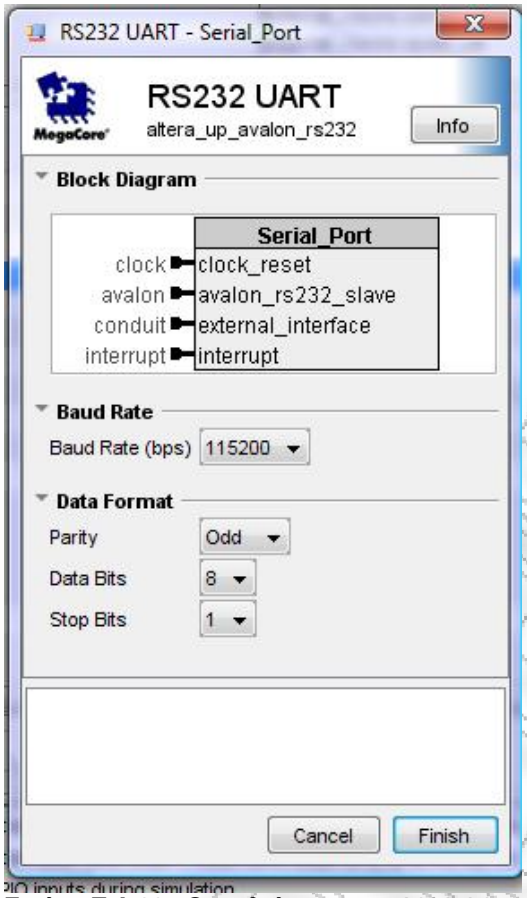
Π.Α.22 **Ρυθμίσεις του πυρήνα υλικού** «PS2 CONTROLLER Core»

Εικόνα Π.Α.24 - Οι ρυθμίσεις του PS2 CONTROLLER Core

Π.Α.23 **Ρυθμίσεις του πυρήνα υλικού** «JTAG_UART Core»

Εικόνα Π.Α.25 - Οι ρυθμίσεις του JTAG_UART Core

Π.Α.24 **Ρυθμίσεις του πυρήνα υλικού** «RS232 CONTROLLER Core»



Εικόνα Π.Α.26 - Οι ρυθμίσεις του RS232 CONTROLLER Core

Παράρτημα Β

Συνοπτικές Οδηγίες Χρήσης

Παρακάτω θα παρουσιαστούν τα βήματα που πρέπει να πραγματοποιηθούν προκειμένου το σύστημα να λειτουργήσει σωστά.

- Το πρώτο βήμα είναι να συνδεθεί η κάρτα DE1 με τον υπολογιστή μέσω του USB Blaster καλωδίου και να τεθεί σε λειτουργία.
- Επίσης, θα πρέπει ο διακόπτης RUN/PROG να είναι στην θέση RUN. Θα πρέπει επίσης στην θύρα Line-In της κάρτας DE1 να συνδεθεί η πηγή ήχου, στην θύρα Line-Out ένα ζεύγος ηχείων ή ακουστικών και στην VGA υποδοχή μία συμβατή οθόνη VGA.
- Το δεύτερο βήμα είναι να εκτελεστεί το Quartus και μέσα από αυτό να ανοιχθεί το project. (File>>Open Project>> DE1_Media_Computer.qpf)
- Το FPGA θα πρέπει να προγραμματιστεί με το σωστό .sof αρχείο, το οποίο παράγεται από το Quartus κατά τη διαδικασία μεταγλώττισης του συστήματος. Το αρχείο αυτό περιλαμβάνει όλες τις ρυθμίσεις (ανάθεση ακροδεκτών και άλλες) που είναι απαραίτητες ώστε το σύστημα που έχει αναπτυχθεί να υλοποιηθεί στο FPGA. Αυτό επιτυγχάνεται με τη χρήση του Quartus Programmer (Tools >> Programmer). Με την εκτέλεση του Quartus Programmer, θα πρέπει να εμφανιστεί και το αντίστοιχο αρχείο .sof. Αν η επιλογή MODE δεν είναι ρυθμισμένη στην επιλογή JTAG θα πρέπει να ρυθμιστεί σε αυτήν την επιλογή και να έχει επιλεγεί η ρύθμιση Program/Configure. Τέλος με την επιλογή Start το αρχείο .sof προγραμματίζει το FPGA.

Τώρα πλέον το Nios II σύστημα είναι υλοποιημένο στο FPGA.
Τα παραπάνω βήματα αφορούσαν το Hardware. Τα επόμενα βήματα αφορούν την εκτέλεση της εφαρμογής στον Nios.

- Θα πρέπει να εκτελεστεί το Nios II IDE.
- Έπειτα ακολουθεί η εισαγωγή του project (που περιλαμβάνει όλους τους πηγαίους κώδικες *.c/c++ και τα αρχεία κεφαλίδας *.h) στο χώρο εργασίας (workspace) του IDE (File>>Import>>Altera NiosII>>Existing Nios II project into workspace>>Browse>> προσδιορισμός του φακέλου του project) και η εισαγωγή της απαραίτητης βιβλιοθήκης συστήματος (File>>Import>>Altera Nios II>> Existing NiosII project into workspace>>Browse>> προσδιορισμός του φακέλου της βιβλιοθήκης του συστήματος: όνομα_ project_syslib)
- Τέλος, θα πρέπει να εκτελεστεί το πρόγραμμα. Αυτό επιτυγχάνεται με την εντολή Run(Run>>Run As>Nios II Hardware>>Run). Με την εντολή αυτή το project θα μεταγλωττιστεί, θα παραχθεί ένα αρχείο *.elf και αυτό το αρχείο θα εκτελεστεί από τον Nios II.

Βιβλιογραφία:

1. ALTERA -JAN_2009 - **Embedded Design Handbook**
2. ALTERA – MAR_2009 - **Nios II Processor Reference Handbook**
3. ALTERA - MAR_2009 - **Nios II Software Developer's Handbook**
4. ALTERA - MAR_2009 - **Quartus II Handbook Version 9.0 Volume 4: SOPC Builder**
5. ALTERA - MAR_2009 - **Quartus II Handbook Version 9.0 Volume 5: Embedded Peripherals**
6. ALTERA - OCT_2007 - **Nios II Hardware Development Tutorial**
7. ALTERA - MAY_2008 - **Nios II Custom Instruction User Guide**
8. ALTERA - MAY_2006 - **Using Nios II Floating-Point Custom Instructions**
9. ALTERA - MAR_2008 - **Avalon Interface Specifications**
10. ALTERA - JAN_2009- **Altera University Program Overview**
11. ALTERA - JUN_2009- **Nios II System Architect Design Tutorial**
12. ALTERA - SEP_2010- **Using the SDRAM on Altera's DE1 Board with VHDL Designs**
13. ALTERA - JUL_2010- **Clock Signals for Altera DE Series Boards**
14. ALTERA - OCT_2006- **Cyclone II FPGA Starter Development Board Reference Manual**
15. ALTERA - OCT_2006- **Cyclone II FPGA Starter Development Kit User Guide**
16. ALTERA - JUL_2010- **Basic Computer System for the Altera DE1 Board**
17. ALTERA - JUL_2010- **Media Computer System for the Altera DE1 Board**
18. ALTERA - JUL_2010- **Audio Core for Altera DE2/DE1 Boards**
19. WOLFSON MICROELECTRONICS- APR_2004- **WM8731 AUDIO CODEC**
20. ALTERA - JUL_2010- **Audio/Video Configuration Core for DE2/DE1 Boards**
21. ALTERA - NOV_2008 - **Character Buffer for Altera DE2/DE1 Boards**
22. ALTERA - MAY_2007 - **DE Boards External Interface for Altera DE2/DE1 Boards**
23. ALTERA - MAY_2007 - **PIO for Altera DE2/DE1 Boards**
24. ALTERA - MAY_2007 - **PS2 Core for Altera DE2/DE1 Boards**
25. ALTERA - MAY_2007 - **RS232 UART Core for Altera DE2/DE1 Boards**
26. ALTERA - MAY_2007 - **SRAM Controller for Altera DE2/DE1 Boards**
27. ALTERA - JUL_2010- **Parallel Port for Altera DE-Series Boards**
28. ALTERA - JUL_2010- **PS/2 Core for Altera DE-Series Boards**
29. ALTERA - JUL_2010- **Video IP Cores for Altera DE Series Boards**
30. ALTERA - MAR_2009 - **FIR Compiler User Guide**
31. ALTERA - SEP_2010- **Introduction to the Altera SOPC Builder Using VHDL Designs**
32. FREERTOS - **Altera Nios II FreeRTOS Demo Running on a Cyclone III FPGA**
33. Richard Barry - **FreeRTOS Reference Manual - API Functions and Configuration Options** - Real Time Engineers Ltd. 2011
34. Richard Barry - **Using the FreeRTOS Real Time Kernel - a Practical Guide**- Real Time Engineers Ltd. 2011
35. Yassine Aoudni-**Custom Instruction Integration Method within Reconfigurable SoC and FPGA Devices**- The 18th International Confernece on Microelectronics (ICM) 2006
36. Joseph A. Fisher-**Customized Instruction-Sets For Embedded Processors** - Hewlett-Packard Laboratories Cambridge
37. Leonardo Amaral, Guido Araujo, Julio Lopez -**HW/SW Co-Design of Identity-Based Encryption using a Custom Instruction Set** - Institute of Computing, University of Campinas,BRAZIL
38. Kilian Foerster-**Improving AES128 software for Altera Nios II processor using custom instructions**- OCT_2005
39. **Instruction Set Synthesis with Efficient Instruction Encoding for Configurable Processors**- JONG-EUN LEE Samsung Electronics Co., Korea -KIYOUNG CHOI Seoul National University, Korea and NIKIL D. DUTT University of California, Irvine

40. Y. Aoudni, Kais Loukil G. Gogniat J.L. Philippe M. Abid1LESTER - **Mapping SoC architecture Solutions for an Application based on PACM Model** -, Université de Bretagne Sud CNRS FRE 2734, Lorient, France
41. Huynh Phung Huynh and Tulika Mitra -**Runtime Reconfiguration of Custom Instructions for Real-Time Embedded Systems**- School of Computing-National University of Singapore
42. Hamblen J., (2008), **Rapid Prototyping Of Digital Systems: SOPC Edition**, Springer
43. H. P. Huynh and T. Mitra. **Instruction-set customization for real-time embedded systems**. In DATE '07
44. K. Atasu, C. Ozturan, G. Dundar, O. Mencer, and W. Luk. CHIPS: **Custom hardware instruction processor synthesis**. In IEEE TCAD'08.
45. J. Sato, M. Imai, T. Hakata, A. Alomary, and N. Hikichi, **"An integrated design environment for application specific integrated processor,"** Proceedings of the International Conference on Computer Design, pp. 414-417, Cambridge, MA, USA, Oct. 1991
46. E. Borin, F. Klein, N. Moreano, R. Azevedo, and G. Araujo, **"Fast instruction set customization,"** Proceedings of the 2nd Workshop on Embedded systems for Real-Time Multimedia (ESTIMedia'04), pp. 53-58, Sep. 2004, Stockholm, Sweden.
47. H. Choi, J.-S. Kim, C.-W. Yoon, I.-C. Park, S.H. Hwang, and C.-M. Kyung, **"Synthesis of application specific instructions for embedded DSP software,"** IEEE Transactions on Computers, Vol. 48, No. 6, pp. 603-614, Jun. 1999.
48. N. Clark, H. Zhong, W. Tang, and S. Mahlke, **"Automatic design of application specific instruction set extensions through dataflow graph exploration,"** International Journal of Parallel Programming, Vol. 31, No. 6, pp. 429-449, Dec. 2003.
49. Nathan T. Clark, Hongtao Zhong, and Scott A. Mahlke, **"Automated custom instruction generation for domain-specific processor acceleration,"** IEEE Transactions on Computers, Vol. 54, No. 10, pp. 1258-1270, Oct. 2005.
50. J. Cong, Y. Fan, G. Han, and Z. Zhang, **"Application-specific instruction generation for configurable processor architectures,"** In Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, pp. 183-189, Feb. 2004, Monterey, California, USA.
51. N. Dutt, and K. Choi, **"Configurable processors for embedded computing,"** IEEE Computer, Vol. 36, No. 1, pp. 120-123, Jan. 2003.
52. R. Gonzalez, **"Xtensa: A configurable and extensible processor,"** IEEE Micro, Vol. 20, No. 2, pp. 60-70, Mar.-Apr. 2000.
53. R. Leupers, K. Karuri, S. Kraemer, and M. Pandey, **"A design flow for configurable embedded processors based on optimized instruction set extension synthesis,"** Proceedings of the Design, Automation and Test in Europe Conference, pp. 581-586, Mar. 2006, Messe Munich, Germany
54. F. Sun, S. Ravi, A. Raghunathan, and N.K. Jha, **"Custom-instruction synthesis for extensible-processor platforms,"** IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 23, No. 2, pp. 216-228, Feb. 2004.
55. Wang, E. Killian, D. Maydan, and C. Rowen, **"Hardware/software instruction set configurability for system-on-chip processors,"** Proceedings of the 38th ACM/IEEE Design Automation Conference, pp. 184-188, Las Vegas, Nevada, USA, Jun. 2001.
56. P. Yiannacouras, J.G.Steffan and J. Rose, **"Exploration and customization of FPGA-based soft processors,"** IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 26, No. 2, pp. 266-277, February 2007.
57. P. Yu and T. Mitra, **"Scalable custom instructions identification for instruction-set extensible processors,"** Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems, pp. 69-78, Washington, DC, USA, Sep. 2004.
58. P. Yu, and T. Mitra, **"Characterizing Embedded Applications for Instruction-Set Extensible Processors,"** Proceedings of the 41st ACM/IEEE Design Automation Conference, pp. 723-728, San Diego, CA, USA, Jun. 2004.
59. G. De Micheli, **Synthesis and Optimization of Digital Circuits**, *McGraw-Hill, International Editions*, 1994.
60. S.P. Seng, W. Luk, and P.Y.K. Cheung, "**Flexible instruction processors".** *Proc. Int. Conf. On Compilers,* Arch. and Syn. For Embedded Systems ACM Press, 2000.

61. K. Compton and S. Hauck, **"Reconfigurable Computing: A Survey of Systems and Software",** in *ACM Computing Surveys*, vol. 34, no. 2, pp.171-210, June 2002.
62. J. Liang, R. Tessier, and O. Mencer, **"Floating point unit generation and evaluation for FPGAs"**, *Proc. Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 2003.
63. Δ. Παυλίδης, Ε.Ε.Ι.Π -Α.Πικράκης, Λέκτορας -**Ψηφιακή Επεξεργασία Σήματος Επεξεργασία Φωνής-Εργαστηριακές Ασκήσεις.** Άνοιξη 2009 ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ
64. Δάκης Παυλίδης, Ε.Ε.Ι.Π. -Δημήτρης Γκιζόπουλος, Αναπληρωτής Καθηγητής **FreeRTOS σε μικροελεγκτές Atmel AVR32**- Απρίλιος 2010 ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ
65. P.J. Ashenden -**Digital design: an embedded systems approach using VHDL**, Morgan Kaufmann, 2008.
66. Volnei A. Pedroni -**Circuit Design with VHDL**, MIT Press.
67. Ifeachor Emmanuel, **Digital Signal Processing –a practical approach**, ADDISON-WESLEY,1993.
68. Μαλτέζος Ιωάννης , **ΨΗΦΙΑΚΑ ΦΙΛΤΡΑ – ΘΕΩΡΙΑ ΕΦΑΡΜΟΓΗ**, Πτυχιακή εργασία Τμήμα Ηλεκτρονικής ΑΣΕΤΕΜ/ΣΕΛΕΤΕ 1997.
69. Qing Li and Carolyn Yao ,**Real-Time Concepts for Embedded Systems** CMP Books © 2003
70. ALTERA-JUN_2009-**Advanced Synthesis Cookbook: A Design Guide for Stratix II, Stratix III, and Stratix IV Devices.**
71. Daniel D. Gajski • Samar Abdi Andreas Gerstlauer • Gunar Schirner -**Embedded System Design - Modeling, Synthesis and Verification**, Springer ,2009.