Πανεπιστήμιο Πατρών

Μηχανικών Η/Υ και Πληροφορικής

Σχεδιασμός System-on-Chip για επεξεργασία εικόνας και υλοποίηση με FPGA.

Συγγραφέας:
Χαράλαμπος Καλάργαρης

Επιβλέπων:
Θεμιστοκλής Χανιωτάκης

21 Ιουλίου 2012

Για την εκπόνηση αυτής της διπλωματικής θα ήθελα να ευχαριστήσω ιδιαίτερα τον καθηγητή μου κ.Χανιωτάκη που ήταν υπεύθυνος της εργασίας και τον κ.Αδαό για την πολύτιμη βοήθειά του στο εργαστήριο Μικροηλεκτρονικής. Παράλληλα θα ήθελα να ευχαριστήσω τον κ.Αλεξιού που είναι υπεύθυνος του εργαστηρίου Μικροηλεκτρονικής. Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου για την στήριξή τους όλα αυτά τα χρόνια που σπούδασα στην Πάτρα.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

1 Εισαγωγή

Σκοπός

Ο σκοπός αυτής της διπλωματικής εργασίας είναι να δημιουργηθεί ένα System On Chip (SoC) που θα στοχεύει στην επεξεργασία εικόνας. Ουσιαστικά στόχος μας είναι να αναβαθμίσουμε την ήδη υπάρχουσα πλατφόρμα που υπάρχει στο εργαστήριο Μικροηλεκτρονικής. Αυτό θα γίνει με την αλλαγή του επεξεργαστή LEON2 που χρησιμοποιείται τώρα με τον επεξεργαστή OR1200 που παρέχεται ελεύθερα από το Open Cores. Οι κυριότεροι λόγοι που για τους οποίους κρίθηκε αναγκαία αυτή η αλλαγή φαίνονται παρακάτω.

- Ο LEON2 είναι ένας επεξεργαστής που το πρώτο μοντέλο του σχεδιάστηκε το 1997. Τα τελευταία χρόνια όμως οι αναβαθμίσεις που εμφανίζονται για αυτόν τον επεξεργαστή αλλά και η χρησιμοποίησή του σε μοντέρνα ολοκληρωμένα συστήματα ακολουθούν μια φθίνουσα πορεία.

- Σε αντίθεση με το παραπάνω στοιχείο ο επεξεργαστής OR1200 ακολουθεί μία ανοδική πορεία τα τελευταία χρόνια. Οι κυριότεροι λόγοι αυτής της ανόδου έχουν να κάνουν με το γεγονός ότι ο OR1200 είναι ένας επεξεργαστής ελεύθερου λογισμικού με αποτέλεσμα οι περισσότερες ακαδημαϊκές κοινότητες στον κόσμο να τον χρησιμοποιούν για να αναπτύξουν εφαρμογές και να βοηθήσουν τους νέους φοιτητές να εξοικειωθούν με την αρχιτεκτονική RISC στους επεξεργαστές.

- Το μεγαλύτερο πλεονέκτημα του επεξεργαστή OR1200 έχει να κάνει με το γεγονός ότι μπορεί κάποιος να σχεδιάσει το δικό του σύστημα χωρίς να χρησιμοποιήσει εμπορικά προϊόντα που κοστίζουν αφού τόσο ο επεξεργαστής όσο και ο δίαυλος επικοινωνίας (Wishbone) είναι ελεύθερα προϊόντα που διανέμονται από το OpenCores.

- Τέλος ο επεξεργαστής OR1200 λόγο του τρόπου που έχει σχεδιαστεί προσφέρει στον χρήστη μια πληθώρα από επιλογές για την παραμετροποίηση του. Αυτό έχει δυο θετικά στοιχεία. Το πρώτο έχει να κάνει με το γεγονός ότι λόγω της εύκολης αλλά ταυτόχρονα και μεγάλης παραμετροποίησης που παρέχει στον χρήστη, τον κάνει να απευθύνεται σε ένα μεγάλο φάσμα εφαρμογών ειδικού σκοπού. Το δεύτερο πλεονέκτημα απευθύνεται στην χρησιμότητα της παραμετροποίηση για τους νέους φοιτητές. Οι φοιτητές μπορούν μέσω της παραμετροποίησης του επεξεργαστή να ακολουθήσουν πολλά διαφορετικά σενάρια υλοποίησης γεγονός που τους βοηθά σημαντικά στην καλύτερη κατανόηση της αρχιτεκτονικής RISC επεξεργαστών.

Περιορισμοί - Τελικός Σκοπός

Όπως γίνετε εύκολα κατανοητό η δημιουργία ενός ολοκληρωμένου συστήματος είναι μία πολύ χρονοβόρα διαδικασία που απαιτεί την συνεργασία πολλών ανθρώπων για να επιτευχθεί. Επειδή η διπλωματικές εργασίες είναι ατομικές έπρεπε αρχικά να οριστεί ποίος θα ήταν ο στόχος αυτής της διπλωματικής εργασίας. Έτσι μετά από συνεννόηση με τον κ.Χανιωτάκη και τον κ.Αδαό ορίστηκε σαν στόχος η δημιουργία ενός απλοϊκού συστήματος που θα ενσωματώνει τον καινούργιο επεξεργαστή OR1200, τον δίαυλο επικοινωνίας AMBA AHB και μία απλή μνήμη. Με αυτό τον τρόπο δημιουργούμε την βάση του μεγαλύτερου συστήματος που είναι απώτερος σκοπός του εργαστήριο Μικροηλεκτρονικής. Ουσιαστικά μετά την εκπόνηση και άλλων διπλωματικών εργασιών (που θα δημιουργούν άλλες μονάδες για το σύστημα- όπως για επεξεργασία εικόνας) θα είναι δυνατόν να προσθέσουν τις μονάδες που σχεδίασαν πάνω στο σύστημα που εμείς σχεδιάσαμε με αποτέλεσμα να δημιουργηθεί ένα μεγαλύτερο και πολυπλοκότερο σύστημα.

Για να εξασφαλίσουμε την δυνατότητα σε άλλους φοιτητές να σχεδιάζουν τις μονάδες που θέλουν και να τις βάζουν κατευθείαν πάνω στο σύστημα μας, στόχος μας είναι να αλληλεπιδρά αρμονικά το σύστημα με την μνήμη που έχει. Αυτό διασφαλίζει την παραπάνω δυνατότητα για τον λόγο ότι τόσο η μνήμη όσο και οι μονάδες που θα σχεδιάσουν οι φοιτητές θα έχουν τα ίδια σήματα εξόδου και εισόδου πάνω στο δίαυλο επικοινωνίας. Με αυτόν τρόπο εμείς διασφαλίζουμε στους φοιτητές ότι αν σχεδιάσουν σωστά την μονάδα τους και ακολουθήσουν την λογική που απαιτεί ο δίαυλος AMBA AHB τότε η μονάδα τους θα αλληλεπιδρά ορθά με το επεξεργαστή OR1200.

Συνδρομή στο έργο της Ακαδημαϊκής κοινότητας

Ένα από τα σημαντικότερα αποτελέσματα αυτής της διπλωματικής εργασίας είναι η συνδρομή της στο έργο της ακαδημαϊκής κοινότητας. Μέτα το τέλος της διπλωματικής εργασίας τόσο οι φοιτητές όσο και οι καθηγητές του τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής και κυρίως αυτοί που θα ασχοληθούν με το εργαστήριο Μικροηλεκτρονικής θα έχουν στην διάθεσή τους ένα καινούργιο επεξεργαστή για να μπορούν να πειραματιστούν. Αυτή η αλλαγή θα βοηθήσει τόσο το τμήμα όσο και το εργαστήριο Μικροηλεκτρονικής να αναβαθμίσουν τα εργαλεία που παρέχουν στους φοιτητές και ταυτόχρονα να διευρύνουν το πεδίο γνώσης που προσφέρουν.

Παράλληλα το έργο αυτής της διπλωματικής εργασίας θα συμπεριληφθεί σε μία δημοσίευση που θα γίνει στο παγκόσμιο συμπόσιο της IEEE 2012 με θέμα Rapid System Prototyping. Το συμπόσιο θα γίνει στην πόλη Tempere της Φιλανδίας και το θέμα τις δικής μας δημοσίευσης είναι "A Flexible Platform for Developing and Evaluating System-on-Chip Architectures". Οι υπόλοιποι συγγραφείς αυτής της δημοσίευσης φαίνονται παρακάτω.

- Κωνσταντίνος Αδαός.
- Βαγγέλης Μαριάτος.
- Γεώργιος Π.Αλεξιού.

Μορφή Αναφοράς Διπλωματικής εργασίας

Η αναφορά αυτής της διπλωματικής εργασίας δημιουργήθηκε με τέτοιο τρόπο ώστε παρουσιάσει τόσο το έργο που έγινε για την εκπόνηση της όσο και για να αποτελεί ενα εγχειρίδιο χρήσης για τους φοιτητες της σχολής που θέλουν να ασχοληθούν με τον επεξεργαστή OR1200. Ουσιαστικά η διπλωματική εργασία χωρίζεται σε 3 κομμάτια.

1. Το πρώτο εστιάζει στην περιγραφή των κυριότερων χαρακτηριστικών του επεξεργαστή OR1200 που συλλέχθηκαν μέσα από πολλά διαφορετικά εγχειρίδια και αποτελούν μια πολύ καλή εισαγωγή για τους φοιτητές πάνω σε θέματα αρχιτεκτονικής του OR1200.
2. Το δεύτερο αναφέρεται τόσο στον τρόπο που εγκαταστάθηκαν οι κατάλληλες αλυσίδες προγραμμάτων στο εργαστήριο Μικροηλεκτρονικής όσο και ο τρόπος με τον οποίο χρησιμοποιούνται αυτά τα προγράμματα για να προσομοιώσουμε την λειτουργιά του επεξεργαστή. Διαδικασία που είναι απαραίτητη για τους νέους φοιτητές ώστε να είναι πιο εύκολη η εισαγωγή και η κατανόηση του επεξεργαστή OR1200.
3. Το τελευταίο κομμάτι εστιάζει στην δημιουργία του ολοκληρωμένου συστήματος. Ουσιαστικά αποτελεί μια παρουσίαση των παραπάνω κομματιών σε πραγματικό κύκλωμα με παράδειγμα. Ταυτόχρονα περιγράφει τις δυσκολίες που συναντήθηκαν κατά την δημιουργία του ολοκληρωμένου συστήματος.

2 OpenRISC 1200 IP Core

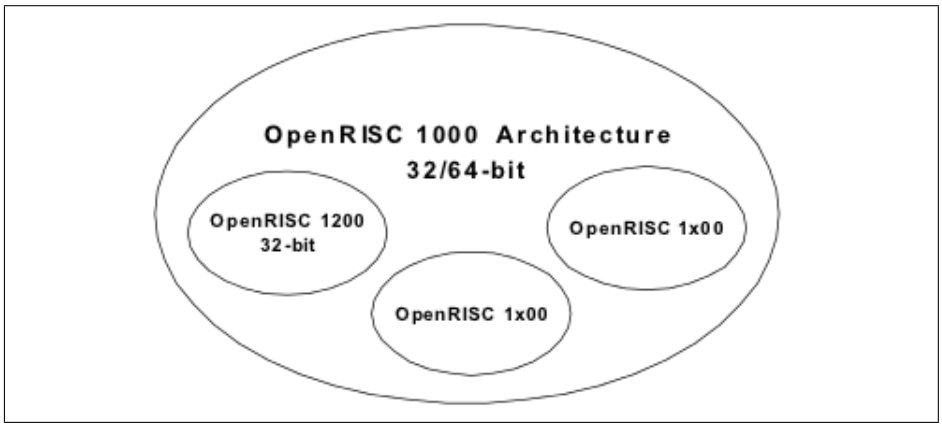
Αυτό το κεφάλαιο αποτελεί μια πρώτη γνωριμία με τον επεξεργαστή OR1200. Θα παρουσιαστούν τα κυριότερα χαρακτηριστικά του και ο τρόπος με τον οποίο υλοποιούνται τόσο από την μεριά του υλικού όσο και από την μεριά της λογικής την οποία ακολουθούν. Κυριότερος σκοπός του κεφαλαίου αυτού είναι να παρέχει στον χρήστη την απαραίτητη γνώση που θα τον βοηθήσει στην ανάπτυξη εφαρμογών σε συμβολική γλώσσα αλλά και στο πως μπορεί να αποτελέσει μέρος ενός ολοκληρωμένου κυκλώματος ειδικού σκοπού (ASIC).

2.1 Εισαγωγή

Σε αυτή την ενότητα θα παρουσιαστούν τα κυριότερα χαρακτηριστικά της οικογένειας επεξεργαστών OpenRISC 1000 και του επεξεργαστή OR1200 που αποτελεί μέρος της.

2.1.1 OpenRISC Οικογένεια

Η OpenRISC 1000 οικογένεια επεξεργαστών αναφέρεται σε μια ελεύθερη, ανοιχτού λογισμικού RISC αρχιτεκτονική κεντρικών μονάδων επεξεργασίας. Σχετικά με την αρχιτεκτονική, η OpenRISC 1000 οικογένεια στοχεύει σε ένα φάσμα υλοποιήσεων που ποικίλουν ως προς την τιμή/απόδοση και το είδος της εφαρμογής. Είναι μια 32/64-bit φόρτωσης και αποθήκευσης (load and store) RISC αρχιτεκτονική που σχεδιάστηκε με έμφαση στην απόδοση, στην απλότητα, στην χαμηλή ενεργειακή κατανάλωση, στην επεκτασιμότητα και στην ευελιξία. Η OpenRISC αρχιτεκτονική στοχεύει σε μεσαία και υψηλή απόδοση δικτύωσης (networking), σε ενσωματωμένα, αυτοκινητοβιομηχανικά και φορητά υπολογιστικά περιβάλλοντα.



Σχήμα 2.1: Σχηματική αναπαράσταση OpenRISC αρχιτεκτονικής.

Όλες οι OpenRISC υλοποιήσεις που το πρώτο ψηφίο στον αριθμό ταυτότητας (identification number) είναι '1' ανήκουν στην OpenRISC 1000 οικογένεια. Το δεύτερο ψηφίο ορίζει ποια χαρακτηριστικά της OpenRISC 1000 αρχιτεκτονικής είναι

υλοποιημένα και με ποιο τρόπο είναι υλοποιημένα. Τα δύο τελευταία ψηφία αναφέρονται στο πως μια υλοποίηση ήταν παραμετροποιημένη πριν χρησιμοποιηθεί σε πραγματική εφαρμογή.

2.1.2 OpenRISC 1200

Ο OR1200 είναι ενας 32-bit βαθμωτός RISC επεξεργαστής με Harvard μικροαρχιτεκτονική, 5 stage integer pipeline, υποστήριξη εικονικής μνήμης (MMU) και βασικές δυνατότητες DSP.

Οι προκαθορισμένες κρυφές μνήμες είναι:

- 1-way direct-mapped 8KB κρυφή μνήμη δεδομένων.
- 1-way direct-mapped 8KB κρυφή μνήμη εντολών.
- Κάθε κρυφή μνήμη έχει γραμμή μεγέθους 16-byte.
- Και οι δύο κρυφές μνήμες είναι φυσικά υλοποιημένες.

Η προκαθορισμένη MMU αποτελείται από:

- 64-entry hash based 1-way direct-mapped data TLB.
- 64-entry hash based 1-way direct-mapped instruction TLB.

Μερικές άλλες επιπρόσθετες λειτουργίες που παρέχει ο OpenRISC 1200 είναι η μονάδα αποσφαλμάτωσης πραγματικού χρόνου (real-time debug unit), υψηλής ανάλυσης χρονιστή, προγραμματιζόμενο ελεγκτή διακοπών (programmable interrupt controller) και μονάδα ρύθμισης των ενεργειακών απαιτήσεων.

Ο OR1200 ουσιαστικά προορίζεται για εφαρμογές σε ενσωματωμένα, φορητά και δικτύωσης συστήματα. Μπορεί να ανταγωνιστεί τους τελευταίους βαθμωτούς 32-bit RISC επεξεργαστές της κλάσης του και να υποστηρίξει αποδοτικά οποιοδήποτε μοντέρνο λειτουργικό σύστημα. Ανταγωνιστές του θεωρούνται οι ARM10, ARC και Tensilica RISC επεξεργαστές.
Συνοπτικά παρουσιάζονται παρακάτω τα χαρακτηριστικά του OR1200:

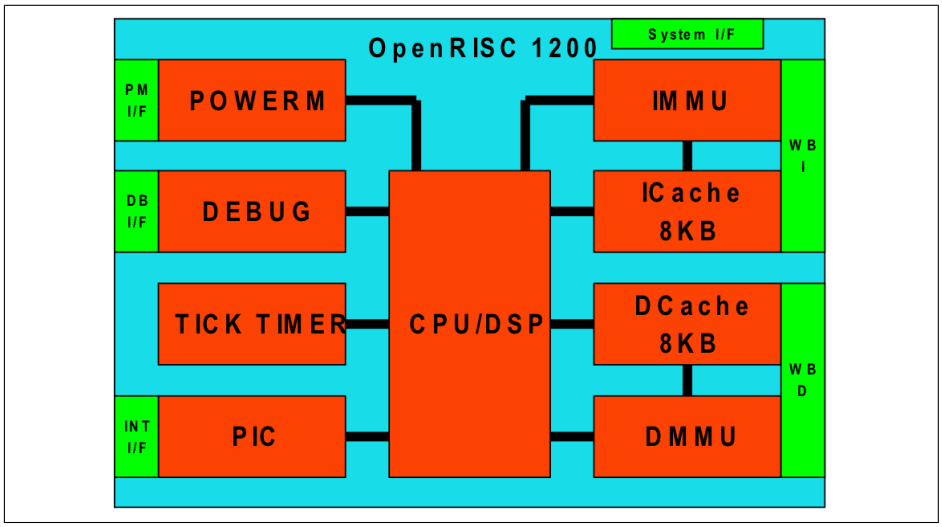
	OR 1200
License	*GNU LGPL*
Platform	*FPGA, ASIC*
Distributed file format	*Verilog*
General	
Architecture	*32-bit RISC*
Byte Ordering	*Big endian*
Pipeline depth	*5*
Issue type	*Single*
Register file	
Organization	*Flat*
# of global registers	*32*
Total # of GPRs	*32*
ISA	
Type	*ORBIS32*
Addressing modes	*Immediate, displacement, pcrelative*
MAC	*32x32-bit, 48-bit Acc*
Custom instructions	*Yes*
Custom coprocessor	*Yes*
Software floating-point support	*IEEE-754 Single and double precision*
Cache	
Hierarchy	*Harvard*
Instruction cache size	*512 byte-8 Kbyte*
Data cache size	*4-8 Kbyte*
Line size	*8-16 byte*
Placement scheme	*Direct-mapped*
Valid bits	*One per cache line*
Line-locking	*Set basis*
System Interface	*Wishbone SoC rev. B 32-bit*
Power Management	*Slow and idle mode, sleep mode, doze mode*
Memory	
On-chip RAM	*Configurable*
Operating system support	*Linux, uClinux, OAR RTEMS RTOS*

Πίνακας 2.1: Overview of OR1200 specifications

2.2 Αρχιτεκτονική

Στο παρακάτω σχήμα βρίσκεται η σχηματική αναπαράσταση του OR1200 επεξεργαστή (δεν απεικονίζεται η μονάδα FPU) όπου αποτελείται από τις εξής υπομονάδες:

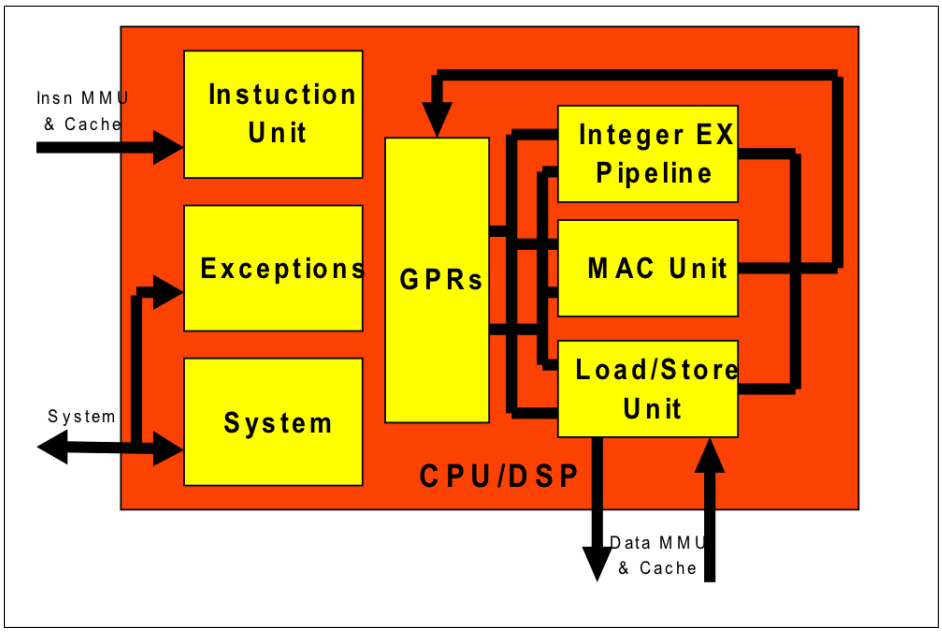
- CPU/FPU/DSP central block.
- Direct-mapped data cache.
- Direct-mapped instruction cache.
- Data MMU based on hash based DTLB.
- Instruction MMU based on hash based ITLB.
- Power management unit and power management interface.
- Tick timer.
- System Interface.
- Debug unit and development interface.
- Interrupt controller and interrupt interface.
- Instruction and Data WISHBONE host interfaces.



Σχήμα 2.2: Core's Architecture

2.2.1 CPU/FPU/DSP

Το μπλόκ CPU/FPU/DSP είναι ένα κεντρικό κομμάτι του επεξεργαστή OR1200 RISC. Το σχήμα 2.3 δείχνει το βασικό μπλόκ διάγραμμα του CPU/FPU/DSP. Η μονάδα του OR1200 CPU/FPU/DSP υλοποιεί μόνο τις ORBIS32 και ORFPX32 ομάδες εντολών. Δεν υπάρχει ακόμα υλοποίηση για τις ομάδες εντολών ORBIS64, ORFBX64 και ORVDX64.



Σχήμα 2.3: CPU/DSP block diagram.

Μονάδα Εντολών

Η μονάδα εντολών (instruction unit) υλοποιεί την βασική pipeline διαδικασία, φέρνει τις εντολές από το υποσύστημα της μνήμης, τις αποστέλλει στις διαθέσιμες μονάδες εκτέλεσης (execution unit) και διατηρεί ενα ιστορικό καταστάσεων ώστε να εξασφαλίσει ένα ακριβές μοντέλο εξαιρέσεων (exception model) για τον σωστό τερματισμό των εργασιών. Επίσης εκτελεί τις υπό όρους και άνευ όρων εντολές διακλάδωσης.

Παράλληλα μπορεί να στείλει διαδοχικές εντολές σε κάθε ρολόι αν η κατάλληλη μονάδα εκτέλεσης είναι διαθέσιμη. Η μονάδα εκτέλεσης είναι αυτή που πρέπει να διακρίνει αν τα δεδομένα που χρειάζεται η εντολή είναι διαθέσιμα και να διασφαλίσει ότι καμία άλλη εντολή δεν χρησιμοποιεί εκείνη την στιγμή τον ίδιο καταχωρητή. Επιπρόσθετα υπολογίζει την αποτελεσματική διεύθυνση (effective address) και προσκομίζει την εντολή από την κρυφή μνήμη εντολών σε ένα κύκλο ρολογιού. Ύστερα η αποτελεσματική διεύθυνση μετατρέπεται σε φυσική διεύθυνση με την βοήθεια της IMMU μονάδας.

Τέλος η μονάδα εντολών χειρίζεται μόνο τις εντολές της κλάσης ORBIS32 και κατά επιλογή ένα υποσύνολο των εντολών ORFPX32. Μερικές εντολές της κλάσης ORFPX32 και όλες οι εντολές των ORFPX3264 και ORVDX64 δεν υποστηρίζονται ακόμα από τον OR1200. Παρακάτω παρουσιάζεται συνοπτικά η λίστα των εντολών που υποστηρίζει ο OR1200 και ποιες είναι προαιρετικές (optional). Για την πλήρη περιγραφή κάθε εντολής μπορείτε να ανατρέξετε στο εγχειρίδιο OpenRISC 1000 System Architecture Manual.

Instruction mnemonic	Opt.	Instruction mnemonic	Opt.	Instruction mnemonic	Opt.	Instruction mnemonic	Opt.
l.add		l.macrc	Y	l.sh		lf.sflt.s	Y
l.addc	Y	l.msb	Y	l.sll		lf.sfne.s	Y
l.addi		l.mfspr		l.slli		lf.sub.s	Y
l.and		l.movhi		l.sra			
l.andi		l.mtspr		l.srai			
l.bf		l.mul	Y	l.srl			
l.bnf		l.muli	Y	l.srli			
l.div	Y	l.nop		l.sub	Y		
l.ff1	Y	l.or		l.sw			
l.fl1	Y	l.ori		l.sys			
l.j		l.rfe		l.trap			
l.jal		l.rori		l.xor			
l.jalr		l.sb		l.xori			
l.jr		l.sfeq		lf.add.s	Y		
l.lbs		l.sfges		lf.div.s	Y		
l.lbz		l.sfgeu		lf.ftoi.s	Y		
l.lhs		l.sfgts		lf.itof.s	Y		
l.lhz		l.sfgtu		lf.mul.s	Y		
l.lws		l.sfleu		lf.sfeq.s	Y		
l.lwz		l.sflts		lf.sfge.s	Y		
l.mac	Y	l.sfltu		lf.sfgt.s	Y		
l.maci	Y	l.sfne		lf.sfle.s	Y		

Πίνακας 2.2: Λίστα εντολών OR1200.

Καταχωρητές Γενικού Σκοπού

Ο OpenRISC 1200 χρησιμοποιεί 32 καταχωρητές γενικού σκοπού (general-purpose registers) των 32 δυαδικών ψηφίων. Η αρχιτεκτονική του επίσης υποστηρίζει σκιώδη αντίγραφα (shadow copies) του αρχείου καταχωρητών (register file) ώστε να υλοποιεί γρήγορη εναλλαγή μεταξύ των πλαισίων εργασίας (working contexts), όμως αυτή η λειτουργία δεν υποστηρίζεται από την παρούσα υλοποίηση του OR1200.

Ο OR1200 υλοποιεί το αρχείο καταχωρητών γενικού σκοπού σαν δυο σύγχρονες μνήμες διπλής θύρας με χωρητικότητα 32 λέξεων των 32 bit ανά λέξη. Παράλληλα έχει την δυνατότητα να διαβάζει δύο τελούμενα σε κάθε κύκλο ρολογιού και να αποθηκεύει ένα αποτέλεσμα καταχωρητές προορισμού.

Μονάδα Load/Store

Η μονάδα Load/Store (LSU) μεταφέρει όλα τα δεδομένα μεταξύ των καταχωρητών γενικού σκοπού (GPRs) και τον επεξεργαστή διαμέσου του εσωτερικού διαύλου επικοινωνίας. Έχει υλοποιηθεί σαν μια ανεξάρτητη μονάδα εκτέλεσης ώστε οι καθυστερήσεις στο υποσύστημα μνήμης λόγω εξαρτήσεων μεταξύ δεδομένων να επηρεάζει μόνο την pipeline διαδικασία.

Τα κυριότερα χαρακτηριστικά της μονάδας Load/Store παρουσιάζονται παρακάτω:

- όλες οι εντολές Load/Store είναι υλοποιημένες σε επίπεδο hardware (atomic instructions include)
- pipeline λειτουργία
- ευθυγραμμισμένες προσπελάσεις στην μνήμη
- address entry buffer

Όταν κάποια εντολή Load ή Store προορίζεται για εκτέλεση τότε η μονάδα Load/Store (LSU) διερευνά αν όλα τα τελούμενα είναι διαθέσιμα. Τα τελούμενα που ελέγχει είναι τα παρακάτω:

- Οι διευθύνσεις των καταχωρητών που θα χρησιμοποιηθούν.
- Τα δεδομένα στους καταχωρητές που προορίζονται για εκτέλεση (store εντολή).
- Τα δεδομένα στους καταχωρητές που προορίζονται για αποθήκευση (load εντολή).

Η μονάδα Load/Store εκτελεί μια εντολή load κάθε δύο κύκλους ρολογιού, υποθέτοντας ότι έχουμε ευστοχία στην κρυφή μνήμη δεδομένων. Η εκτέλεση της εντολής store απαιτεί ένα κύκλο ρολογιού, υποθέτοντας πάλι ότι έχουμε ευστοχία στην κρυφή μνήμη δεδομένων. Παράλληλα η μονάδα Load/Store υπολογίζει τις αποτελεσματικές διευθύνσεις (effective address), οι οποίες αργότερα μετατρέπονται σε φυσικές διευθύνσεις με την της DMMU μονάδας.

Πράξεις ακεραίων

Ο πυρήνας του OR1200 υποστηρίζει τα παρακάτω είδη εντολών για πράξεις μεταξύ 32 bit ακεραίων (integer execution pipeline):

- Αριθμητικές εντολές.
- Εντολές σύγκρισης.
- Λογικές εντολές.
- Εντολές ολίσθησης και περιστροφής.

Instruction Group	Clock Cycles to Execute
Arithmetic except Multiply/Divide	1
Multiply	3
Divide	32
Compare	1
Logical	1
Rotate and Shift	1
Others	1

Πίνακας 2.3: Χρόνοι εκτέλεσης πράξεων ακεραίων.

Ο πολλαπλασιασμός ακεραίων μπορεί γίνετε είτε σειριακά είτε παράλληλα. Η σειριακή λειτουργία απαιτεί ένα κύκλο ρολογιού για κάθε δυαδικό ψηφίο του τελούμενου, άρα στην προκειμένη περίπτωση (ο OR1200 είναι 32bit επεξεργαστής) 32 κύκλους ρολογιού. Προς το παρών κανένα πρόγραμμα σύνθεσης δεν υποστηρίζει την πράξη της διαίρεσης.

Μονάδα MAC

Η μονάδα MAC εκτελεί λειτουργίες DSP MAC. Οι MAC λειτουργίες είναι 32x32 με 48 bit συσσωρευτή. Η μονάδα MAC είναι υλοποιημένη με την τεχνική pipeline και μπορεί να δεχτεί καινούργιες MAC εργασίες σε κάθε καινούργιο κύκλο.

Ιδιαίτερη προσοχή πρέπει να δοθεί όταν εκτελείται η εντολή *l.macrc* (MAC read and clear) πολύ νωρίτερα από την τελευταία εκτέλεση της εντολής *l.mac* μιας και το τελούμενο που θα ζητάει η εντολή *l.macrc* δεν θα είναι ακόμα έτοιμο. Για αυτό τον λόγο πρέπει μεταξύ αυτών των δύο εντολών να υπάρχουν τουλάχιστον τρεις άλλες εντολές (ακόμα και *l.nops*).

Μονάδα Κινητής Υποδιαστολής

Η υλοποίηση της μονάδας Κινητής Υποδιαστολής(FPU) είναι βασισμένη σε δύο άλλες FPUs που παρέχονται από το OpenCores.org. Για τις συναρτήσεις σύγκρισης και μετατροπής κομμάτια υλοποίησης έχουν παρθεί από την FPU που έχει σχεδιαστεί από τον Rudolf Usselmann, ενώ για τις αριθμητικές λειτουργίες από το πρότζεκτ fpu100 όπου ο Jidan Al-Eryani τις μετέτρεψε σε γλώσσα Verilog HDL.

Όλες οι εντολές ORFPX32 εκτός από τις *lf.madd.s* και *lf.rem.s* υποστηρίζονται από την FPU όταν αυτή έχει ενεργοποιηθεί από τις ρυθμίσεις του OR1200. Οι χρόνοι εκτέλεσης των εντολών κινητής υποδιαστολής παρουσιάζονται στον παρακάτω πίνακα.

Operation	Cycles
Add/subtract	10
Multiply	38
Divide	37
Compare	2
Convert	7

Πίνακας 2.4: Χρόνοι εκτέλεσης εντολών κινητής υποδιαστολής.

Μονάδα Συστήματος

Η μονάδα συστήματος συνδέει όλα τα σήματα της CPU/FPU/DSP μονάδας που δεν είναι συνδεδεμένα με τις διεπαφές (interfaces) των εντολών και των δεδομένων. Παράλληλα υλοποιεί όλους τους καταχωρητές ειδικού σκοπού (π.χ. supervisor registers).

Μονάδα Εξαιρέσεων

Οι εξαιρέσεις του πυρήνα μπορούν να δημιουργηθούν όταν μια συνθήκη εξαίρεσης παρουσιάζεται. Παρακάτω φαίνονται ποιες είναι αυτές οι συνθήκες για τον OR1200.

- Εξωτερικά αιτήματα διακοπών (external interrupt request).
- Ορισμένες συνθήκες πρόσβασης στην μνήμη.
- Εσωτερικά λάθη όπως η προσπάθεια εκτέλεσης μη υλοποιημένης εντολής (undefined opcode).
- Εσωτερικές εξαιρέσεις όπως σημεία διακοπής (breakpoints).

Η μονάδα διαχείρισης εξαιρέσεων είναι αόρατη από το λογισμικό του χρήστη και χρησιμοποιεί τον ίδιο μηχανισμό για να εξυπηρετεί όλα τα είδη των εξαιρέσεων. Όταν μία εξαίρεση παρουσιάζεται τότε ο έλεγχος μεταφέρεται στον διαχειριστή εξαιρέσεων με μία προκαθορισμένη μετατόπιση ανάλογα με το είδος της εξαίρεσης που παρουσιάστηκε. Οι εξαιρέσεις διαχειρίζονται από το supervisor mode.

EXCEPTION TYPE	VECTOR OFFSET	CAUSING CONDITIONS
Reset	0x100	Caused by reset.
Bus Error	0x200	Caused by an attempt to access invalid physical address.
Data Page Fault	0x300	Generated artificially by DTLB miss exception handler when no matching PTE found in page tables or page protection violation for load/store operations.
Instruction Page Fault	0x400	Generated artificially by ITLB miss exception handler when no matching PTE found in page tables or page protection violation for instruction fetch.
Low Priority External Interrupt	0x500	Low priority external interrupt asserted.
Alignment	0x600	Load/store access to naturally not aligned location.
Illegal Instruction	0x700	Illegal instruction in the instruction stream.
High Priority External Interrupt	0x800	High priority external interrupt asserted.
D-TLB Miss	0x900	No matching entry in DTLB (DTLB miss).
I-TLB Miss	0xA00	No matching entry in ITLB (ITLB miss).
System Call	0xC00	System call initiated by software.
Floating point exception	0xD00	FP operation caused flags in FPCSR to become set.
Trap	0xE00	Trap instruction was decoded

Πίνακας 2.5: Λίστα εξαιρέσεων.

2.2.2 Κρυφή Μνήμη Δεδομένων

Η προκαθορισμένες ρυθμίσεις για την κρυφή μνήμη δεδομένων για τον OR1200 είναι 8-KByte, 1-τρόπων πλήρους απεικόνισης (1-way direct-mapped), οι οποίες επιτρέπουν στον πυρήνα ταχεία προσπέλαση των δεδομένων. Παρόλα αυτά η κρυφή μνήμη μπορεί να παραμετροποιηθεί σύμφωνα με τον Πίνακα 2.6.

	Direct mapped
16B/line, 256 lines, 1 way	4KB
16B/line, 512 lines, 1 way	**8KB (default)**
16B/line, 1024 lines, 1 way	16KB
32B/line, 1024 lines, 1 way	32KB

Πίνακας 2.6: Ρυθμίσεις μεγέθους κρυφής μνήμης δεδομένων

Ο OR1200 παρέχει την δυνατότητα η κρυφή μνήμη δεδομένων να χρησιμοποιεί είτε την write-through είτε την write-back στρατηγική, όμως η write-back στρατηγική είναι ακόμα σε πειραματικό στάδιο.

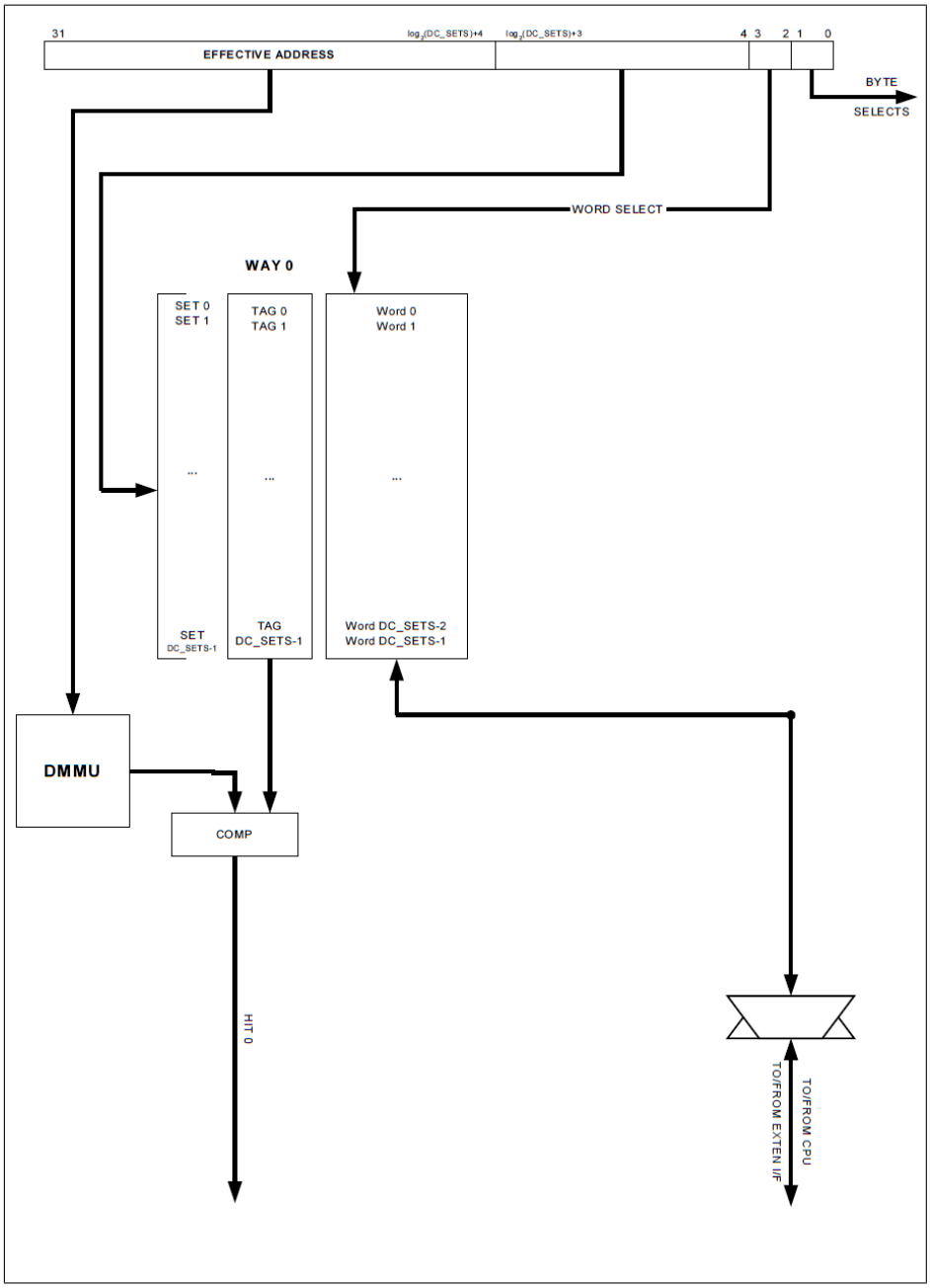
Χαρακτηριστικά:

- Η κρυφή μνήμη δεδομένων είναι ξεχωριστά υλοποιημένη από την κρυφή μνήμη εντολών (Harvard αρχιτεκτονική)
- Η κρυφή μνήμη δεδομένων χρησιμοποιεί τον αλγόριθμο "Λιγότερο Πρόσφατα Χρησιμοποιημένο" για την αντικατάσταση των πλαισίων.
- Ο κατάλογος της κρυφής μνήμης είναι φυσικά διευθυνσιοδοτημένος (physically addressed). Η ετικέτα της φυσικής διεύθυνσης είναι αποθηκευμένη στον κατάλογο της κρυφής μνήμης.
- Write-through ή write-back στρατηγική.
- Ολόκληρη η κρυφή μνήμη μπορεί να απενεργοποιηθεί, γραμμές να ακυρωθούν, να καθαριστούν ή να γραφτούν πίσω, γράφοντας στους καταχωρητές ειδικού σκοπού της κρυφής μνήμης.

Κατά την αστοχία της κρυφής μνήμης και με τις κατάλληλες συνθήκες, η γραμμή της κρυφής μνήμης γεμίζει ή αδειάζει (written back στρατηγική) με ριπές (burst) των 16-byte (default). Οι ριπές πραγματοποιούνται σαν κρίσιμη-λέξη-πρώτα (critical-word-first) λειτουργία˙ η κρίσιμη γράφεται στην κρυφή μνήμη και ταυτόχρονα προωθείται στην μονάδα αιτήσεων (requesting unit), έτσι μειώνονται οι καθυστερήσεις που προκαλούνται από το γέμισμα της γραμμής. Η κρυφή μνήμη παρέχει αποθηκευτικό χώρο για τις ετικέτες και εκτελεί συναρτήσεις αντικατάστασης γραμμών.

Η κρυφή μνήμη δεδομένων είναι στενά συνδεδεμένη με μια εξωτερική διεπαφή ώστε να επιτρέπει την αποδοτική πρόσβαση στον ελεγκτή μνήμης.

Παράλληλα η κρυφή μνήμη δεδομένων προμηθεύει τα δεδομένα στου καταχωρητές γενικού σκοπού διαμέσου της διεπαφής των 32-bit της μονάδας Load/Store. Η μονάδα Load/Store παρέχει όλη την απαραίτητη λογική για τον υπολογισμό της αποτελεσματικής διεύθυνσης (effective addresses), την κατάλληλη αλληλουχία για τις λειτουργίες load/store και χειρίζεται την ευθυγράμμιση των δεδομένων από και προς την κρυφή μνήμη δεδομένων. Οι λειτουργίες εγγραφής στην κρυφή μνήμη δεδομένων μπορούν να γίνουν με βάση το 1 byte, την μισή-λέξη (half-word) ή μια λέξη (word).



Σχήμα 2.4: Οργάνωση κρυφής μνήμης δεδομένων.

Κάθε γραμμή περιέχει τέσσερις συνεχόμενες λέξεις από τη μνήμη που έχουν φορτωθεί από μια οριοθετημένη γραμμή ως προς το μέγεθός της. Ως αποτέλεσμα, οι γραμμές της κρυφής μνήμης είναι ευθυγραμμισμένες με τα όρια της σελίδας.

2.2.3 Κρυφή Μνήμη Εντολών

Η προκαθορισμένες ρυθμίσεις για την κρυφή μνήμη εντολών για τον OR1200 είναι 8-KByte, 1-τρόπων πλήρους απεικόνισης (1-way direct-mapped), οι οποίες επιτρέπουν στον πυρήνα ταχεία προσπέλαση στις εντολές. Παρόλα αυτά η κρυφή μνήμη μπορεί να παραμετροποιηθεί σύμφωνα με τον Πίνακα 2.7.

	Direct mapped
16B/line, 32 lines, 1 way	512B
16B/line, 256 lines, 1 way	4KB
16B/line, 512 lines, 1 way	**8KB (default)**
16B/line, 1024 lines, 1 way	16KB
32B/line, 1024 lines, 1 way	32KB

Πίνακας 2.7: Ρυθμίσεις μεγέθους κρυφής μνήμης εντολών

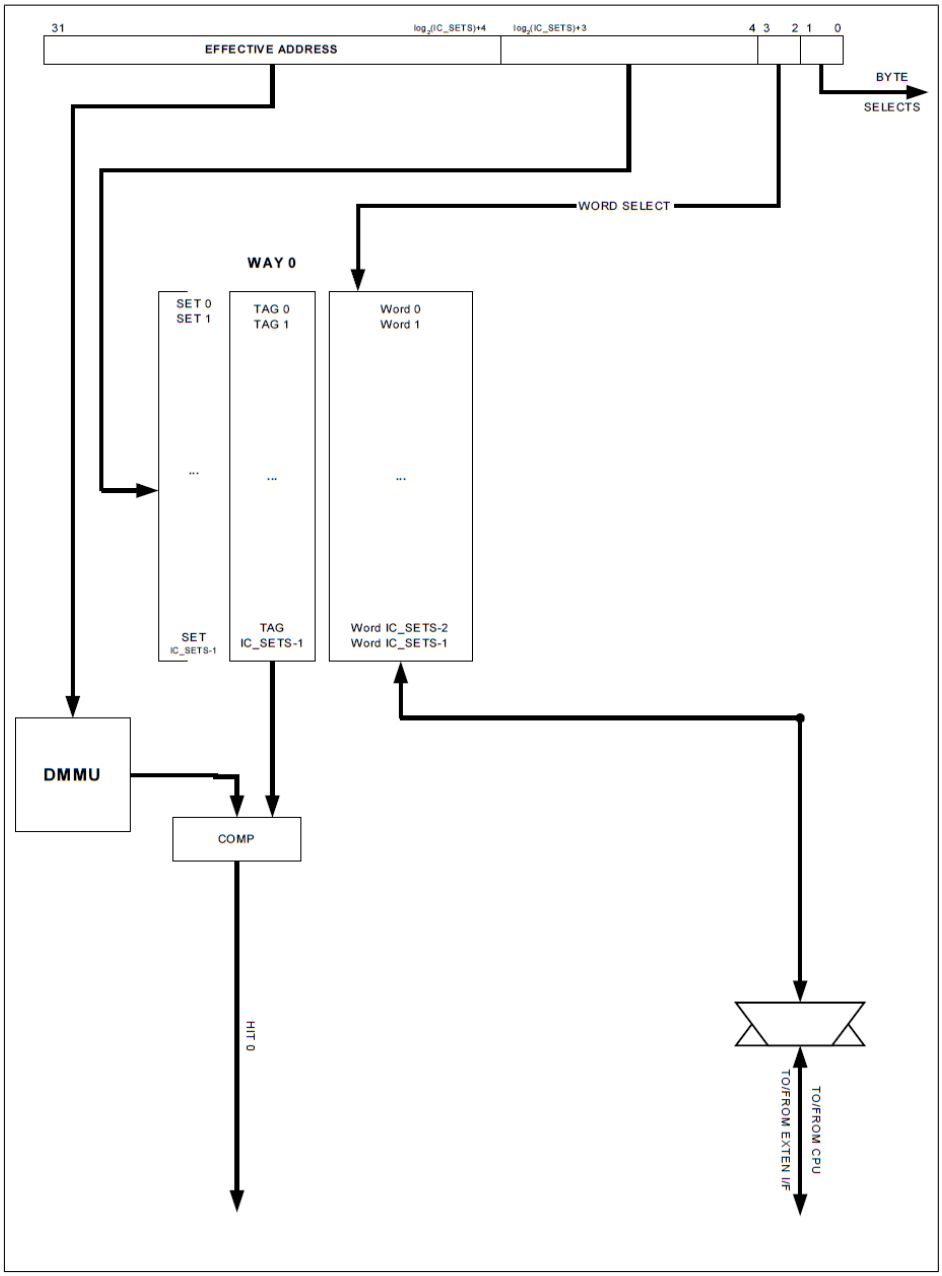
Χαρακτηριστικά:

- Η κρυφή μνήμη εντολών είναι ξεχωριστά υλοποιημένη από την κρυφή μνήμη δεδομένων (Harvard αρχιτεκτονική).
- Η κρυφή μνήμη εντολών χρησιμοποιεί τον αλγόριθμο "Λιγότερο Πρόσφατα Χρησιμοποιημένο" για την αντικατάσταση των πλαισίων.
- Ο κατάλογος της κρυφής μνήμης είναι φυσικά διευθυνσιοδοτημένος (physically addressed). Η ετικέτα της φυσικής διεύθυνσης είναι αποθηκευμένη στον κατάλογο της κρυφής μνήμης.
- Μπορεί να απενεργοποιηθεί ή να ακυρωθεί/ γράφοντας στους καταχωρητές ειδικού σκοπού της κρυφής μνήμης.

Κατά την αστοχία της κρυφής μνήμης η γραμμή της κρυφής μνήμης γεμίζει ή με ριπές (burst) των 16-byte (default). Οι ριπές πραγματοποιούνται σαν κρίσιμη-λέξη-πρώτα (critical-word-first) λειτουργία˙ η κρίσιμη γράφεται στην κρυφή μνήμη και ταυτόχρονα προωθείται στην μονάδα αιτήσεων (requesting unit), έτσι μειώνονται οι καθυστερήσεις που προκαλούνται από το γέμισμα της γραμμής. Η κρυφή μνήμη εντολών παρέχει αποθηκευτικό χώρο για τις ετικέτες και εκτελεί συναρτήσεις αντικατάστασης γραμμών.

Η κρυφή μνήμη εντολών είναι στενά συνδεδεμένη με μια εξωτερική διεπαφή ώστε να επιτρέπει την αποδοτική πρόσβαση στον ελεγκτή μνήμης.

Παράλληλα η κρυφή μνήμη εντολών προμηθεύει τις εντολές προς εκτέλεση διαμέσου της διεπαφής των 32-bit της υπομονάδας προσκόμισης εντολών. Η υπομονάδα προσκόμισης εντολών παρέχει όλη την απαραίτητη λογική για τον υπολογισμό της αποτελεσματικής διεύθυνσης (effective addresses).



Σχήμα 2.5: Οργάνωση κρυφής μνήμης εντολών

Κάθε γραμμή περιέχει τέσσερις συνεχόμενες λέξεις από τη μνήμη που έχουν φορτωθεί από μια οριοθετημένη γραμμή ως προς το μέγεθός της. Ως αποτέλεσμα, οι γραμμές της κρυφής μνήμης είναι ευθυγραμμισμένες με τα όρια της σελίδας.

2.2.4 Διαχείριση Μνήμης (MMU) Δεδομένων

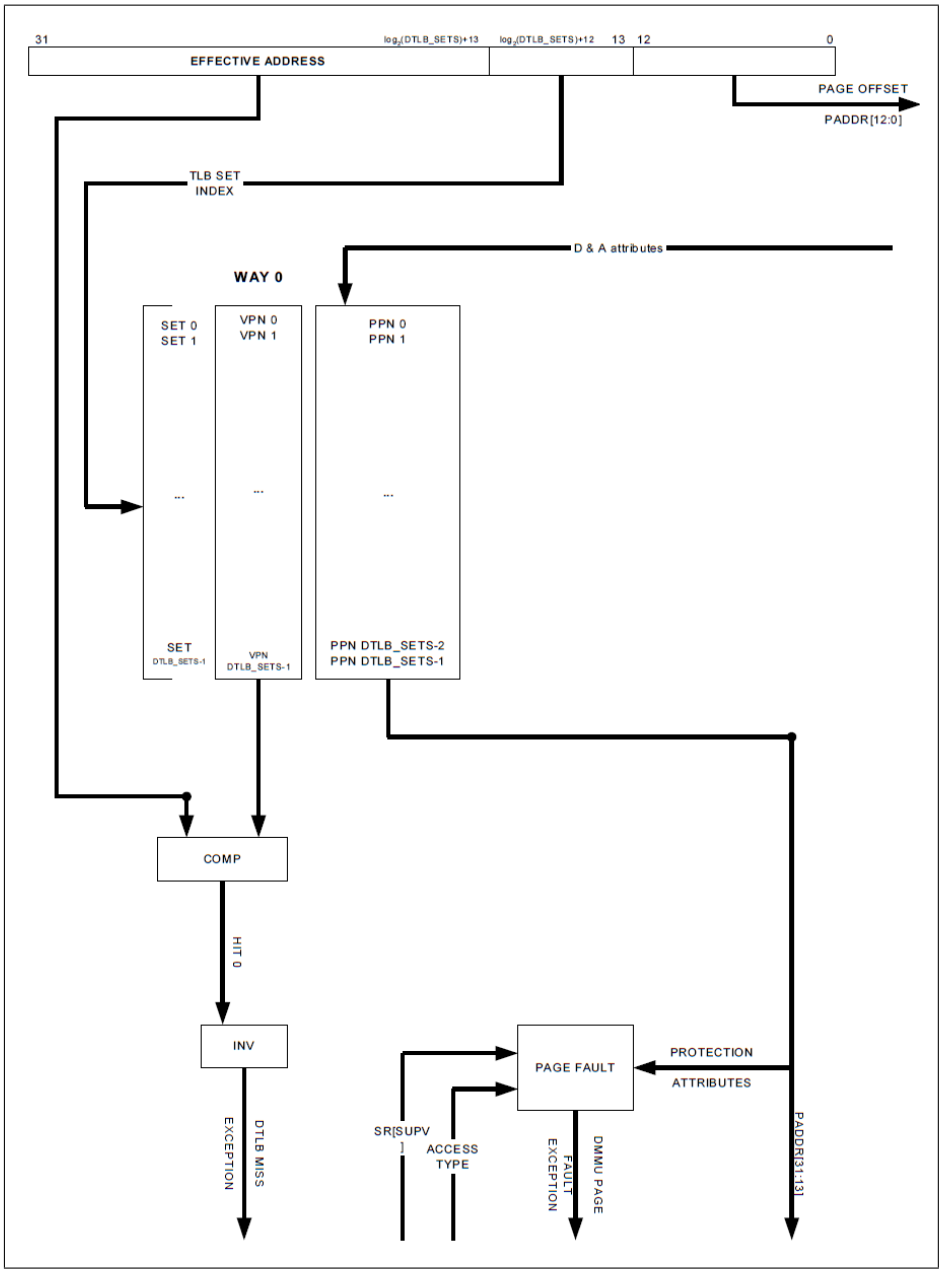
Η OR1200 υλοποιεί ένα εικονικό σύστημα διαχείρισης μνήμης (memomy management unit - MMU) που παρέχει προστασία κατά την πρόσβαση στην μνήμη και αποτελεσματική μετάφραση σε φυσικές διευθύνσεις. Η διακριτότητα της προστασίας είναι όπως ορίζεται από την αρχιτεκτονική OpenRISC 1000 8-Kbyte και 16-Mbyte σελίδες.

	Direct mapped
16 entries per way	16 DTLB entries
32 entries per way	32 DTLB entries
64 entries per way	**64 DTLB entries (default)**
128 entries per way	128 DTLB entries

Πίνακας 2.8: Ρυθμίσεις Δεδομένων TLB (translation lookaside buffer)

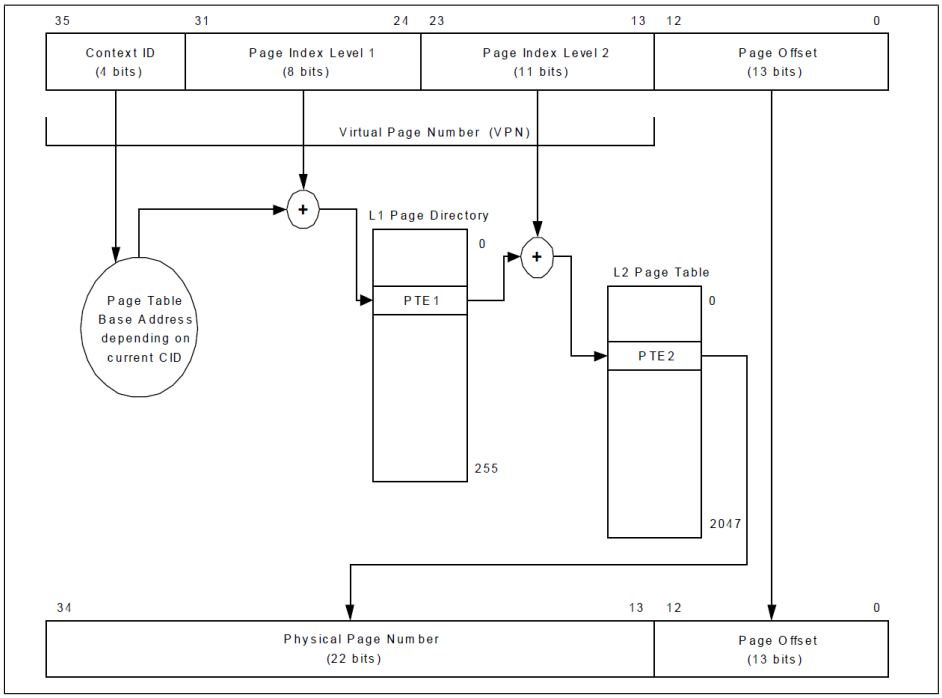
Χαρακτηριστικά:

- Η MMU δεδομένων είναι ξεχωριστή από την MMU εντόλων.
- Το μέγεθός της σελίδας είναι 8-KByte.
- Ολοκληρωμένο σύστημα προστασίας σελίδας.
- Πλήρης απεικόνιση κατακερματισμού βασισμένο στον translation lookaside buffer (DTLB) με προκαθορισμένο 1-τροπο συσχέτισης και τα παρακάτω χαρακτηριστικά:
  - Παροχή εξαιρέσεων στην περίπτωση εξαιρέσεων και λαθών.
  - Software tablewalk.
  - Υψηλή απόδοση λόγο του κατακερματισμού.
  - Τροποποιήσιμο αριθμό καταχωρήσεων στο DLTB με προκαθορισμένες τις 64 καταχωρήσεις ανά τρόπο.



Σχήμα 2.6: Οργάνωση MMU δεδομένων.

Η υλοποίηση της MMU σε υλικό υποστηρίζει δύο-επιπέδων software tablewalk.



Σχήμα 2.7: Μηχανισμός μετάφρασης διευθύνσεων με την χρήση 2-επιπέδων πίνακα σελιδοποίησης.

2.2.5 Διαχείριση Μνήμης (MMU) Εντολών

Η OR1200 υλοποιεί ένα εικονικό σύστημα διαχείρισης μνήμης (memomy management unit - MMU) που παρέχει προστασία κατά την πρόσβαση στην μνήμη και αποτελεσματική μετάφραση σε φυσικές διευθύνσεις. Η διακριτότητα της προστασίας είναι όπως ορίζεται από την αρχιτεκτονική OpenRISC 1000 8-Kbyte και 16-Mbyte σελίδες.

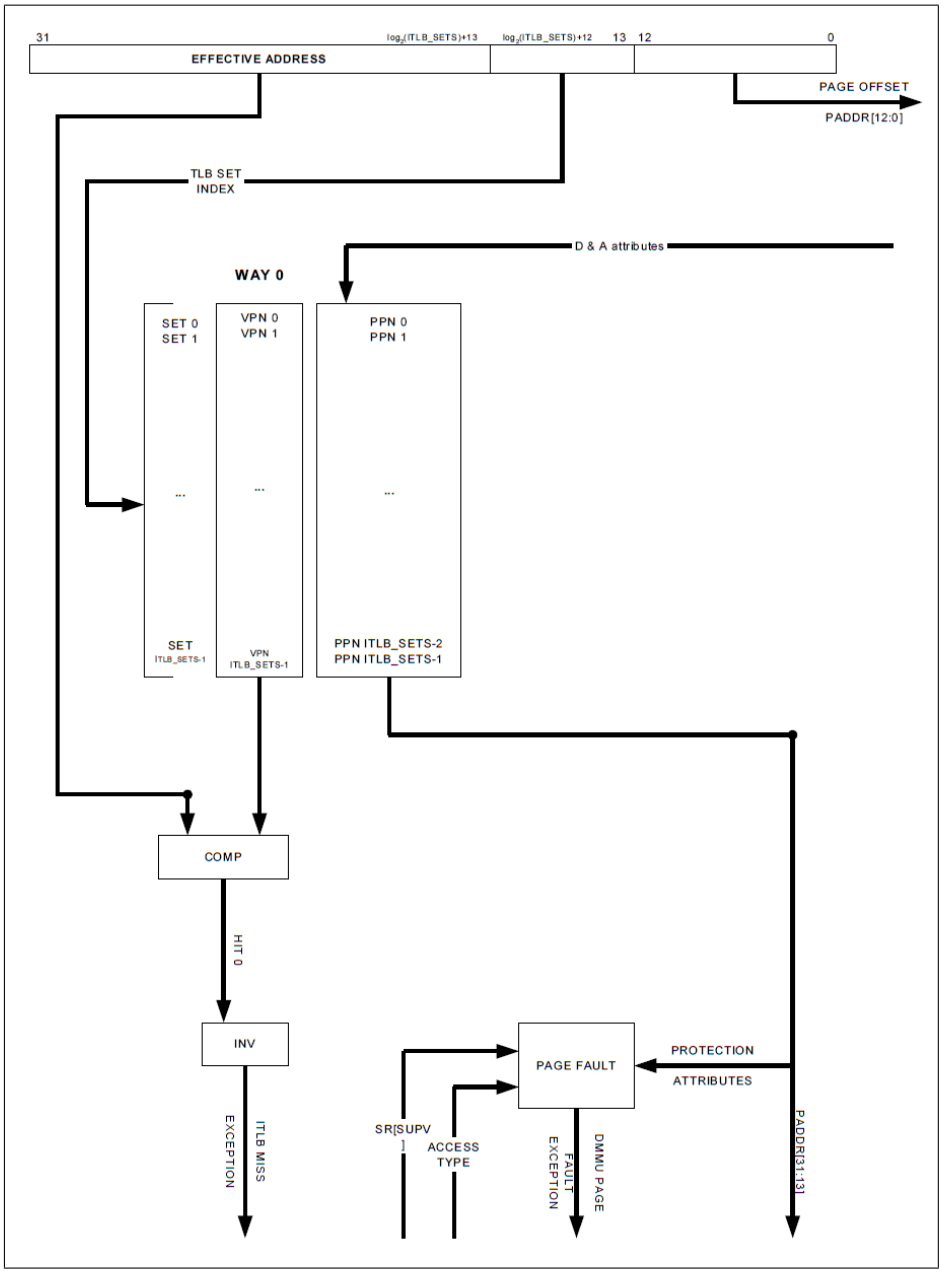
	Direct mapped
16 entries per way	16 ITLB entries
32 entries per way	32 ITLB entries
64 entries per way	**64 ITLB entries (default)**
128 entries per way	128 ITLB entries

Πίνακας 2.9: Ρυθμίσεις Εντολών TLB (translation lookaside buffer)

Χαρακτηριστικά:

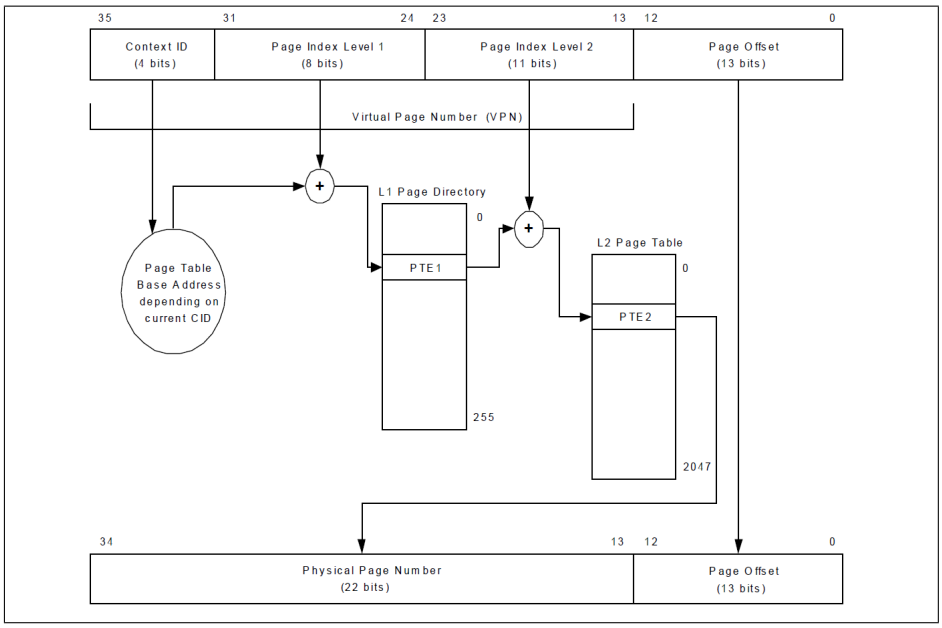
- Η MMU εντόλων είναι ξεχωριστή από την MMU δεδομένων.
- Το μέγεθός της σελίδας είναι 8-KByte.

- Ολοκληρωμένο σύστημα προστασίας σελίδας.
- Πλήρης απεικόνιση κατακερματισμού βασισμένο στον translation lookaside buffer (ITLB) με προκαθορισμένο 1-τρόπο συσχέτισης και τα παρακάτω χαρακτηριστικά:
  - Παροχή εξαιρέσεων στην περίπτωση εξαιρέσεων και λαθών.
  - Software tablewalk.
  - Υψηλή απόδοση λόγο του κατακερματισμού.
  - Τροποποιήσιμο αριθμό καταχωρήσεων στο ILTB με προκαθορισμένες τις 64 καταχωρήσεις ανά τρόπο.



Σχήμα 2.8: Οργάνωση MMU εντολών.

Η υλοποίηση της MMU σε υλικό υποστηρίζει δύο-επιπέδων software tablewalk.



Σχήμα 2.9: Μηχανισμός μετάφρασης διευθύνσεων με την χρήση 2-επιπέδων πίνακα σελιδοποίησης.

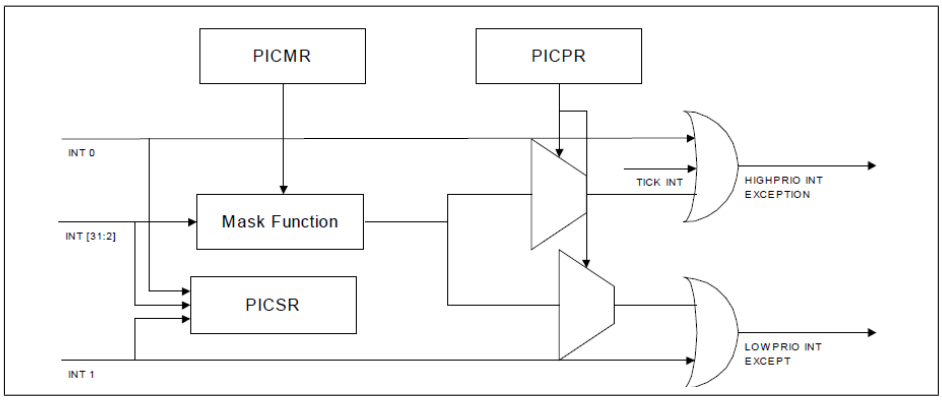
2.2.6 Προγραμματιζόμενος Ελεγκτής Διακοπών

Ο ελεγκτής διακοπών (Interrupt controller) λαμβάνει διακοπές από εξωτερικές πηγές και τις προωθεί ανάλογα με την προτεραιότητά τους (χαμηλή-υψηλή) σαν εξαιρέσεις (exception) στον πυρήνα του επεξεργαστή.

Ο προγραμματιζόμενος ελεγκτής διακοπών έχει 32 σήματα εισόδου διακοπών και τρεις καταχωρητές ειδικού σκοπού, οι οποίοι είναι:

- *PICMR register:* Αυτός ο καταχωρητής χρησιμοποιείται για να φιλτράρει (mask) τα 30 προγραμματιζόμενα σήματα διακοπών.
- *PICPR register:* Αυτός ο καταχωρητής χρησιμοποιείται για να αναθέτει προτεραιότητα (low-high) στα 30 προγραμματιζόμενα σήματα διακοπών.
- *PICSR register:* Αυτός ο καταχωρητής χρησιμοποιείται για να ελέγχει και να τροποποιεί την κατάσταση καθενός ξεχωριστά σήματος διακοπής ανάλογα με τον αν έχει εξυπηρετηθεί ή όχι.

Τα σήματα εισόδου διακοπών *int0* και *int1* είναι πάντα ενεργοποιημένα και συνδεδεμένα με την υψηλή και την χαμηλή προτεραιότητα εισόδου αντίστοιχα, όπως φαίνεται και στο σχήμα 2.10.



Σχήμα 2.10: Μπλοκ διάγραμμα του ελεγκτή διακοπών.

Όταν μια διακοπή προκαλείται σε ένα από τα τριάντα σήματα διακοπών (δηλαδή όταν το συγκεκριμένο σήμα-π.χ *int14*- παίρνει τιμή στον καταχωρητή PICSR), τότε καμία νέα διακοπή δεν μπορεί να εξυπηρετηθεί στο ίδιο σήμα (*int14*) μέχρι να καθαριστεί το δυαδικό ψηφίο που το αντιπροσωπεύει στον καταχωρητή PICSR. Η συνήθης διαδικασία που ακολουθείται όταν μια διακοπή προκαλείται είναι η παρακάτω.

1. Το περιφερειακό προσπαθεί να πάρει τον έλεγχο του επεξεργαστή ενεργοποιώντας το σήμα διακοπής που του έχει αντιστοιχιστεί. Αυτό έχει σαν αποτέλεσμα να πυροδοτείται εκείνη την στιγμή ο διαχειριστής διακοπών.
2. Ο διαχειριστής διακοπών επεξεργάζεται την διακοπή.
3. Ο διαχειριστής ενημερώνει το περιφερειακό ότι η διακοπή που προκάλεσε επεξεργάστηκε (συνήθως μέσω memory-mapped καταχωρητή).
4. Το περιφερειακό αποσύρει την αίτηση που είχε κάνει.
5. Ο διαχειριστής διακοπών καθαρίζει το αντίστοιχο δυαδικό ψηφίο στον PICSR καταχωρητή.

2.2.7 Tick Timer

Ο επεξεργαστής OR1200 έχει υλοποιημένη λειτουργία χρονιστή (tick timer). Βασικά αυτός είναι ένας χρονοδιακόπτης που χρονίζεται από το ρολόι του επεξεργαστή και χρησιμοποιείται από το λειτουργικό σύστημα για ακριβείς μετρήσεις χρόνου και για τον χρονοπρογραμματισμό των διεργασιών τους συστήματος.

Ο OR1200 ακολουθεί αυστηρά τον αρχιτεκτονικό ορισμό του χρονιστή όπου:

- Μέγιστες μετρήσεις του χρονιστή 2^{32} κύκλους ρολογιού.
- Μέγιστη χρονική περίοδο, 2^{28} κύκλους ρολογιού μεταξύ διακοπών.
- Φιλτραρισμένο σήμα διακοπής χρονιστή (Maskable tick timer interrupt).
- Δυνατότητα επανεκκίνησης, μονής και συνεχούς μέτρησης.

2.2.8 Διαχείριση Ενεργειακών Απαιτήσεων

Για την βελτιστοποίηση της κατανάλωσης ενέργειας, ο OR1200 παρέχει καταστάσεις (mode) χαμηλής ενεργειακής κατανάλωσης που μπορούν να χρησιμοποιηθούν ώστε δυναμικά να ενεργοποιούνται και να απενεργοποιούνται ορισμένες εσωτερικές ενότητες (module).

Ο OR1200 έχει τρεις κύριες λειτουργίες για την ελαχιστοποίηση της κατανάλωσης ενέργειας:

- Slow και Idle καταστάσεις (SW controlled clock freq reduction)
- Doze και Sleep καταστάσεις (interrupt wake-up)

Power Minimization Feature	Approx Power Consumption Reduction
Slow and Idle mode	2x – 10x
Doze mode	100x
Sleep mode	200x
Dynamic clock gating	N/A

Πίνακας 2.10: Καταστάσεις βελτιστοποίησης κατανάλωσης ενέργειας.

Η κατάσταση Slow Down εκμεταλλεύεται τους low-power διαιρέτες από το εξωτερικό κύκλωμα παραγωγής ρολογιού ώστε να επιτυγχάνει πλήρης λειτουργικότητα αλλά σε χαμηλότερη συχνότητα, εξοικονομώντας έτσι ενέργεια.

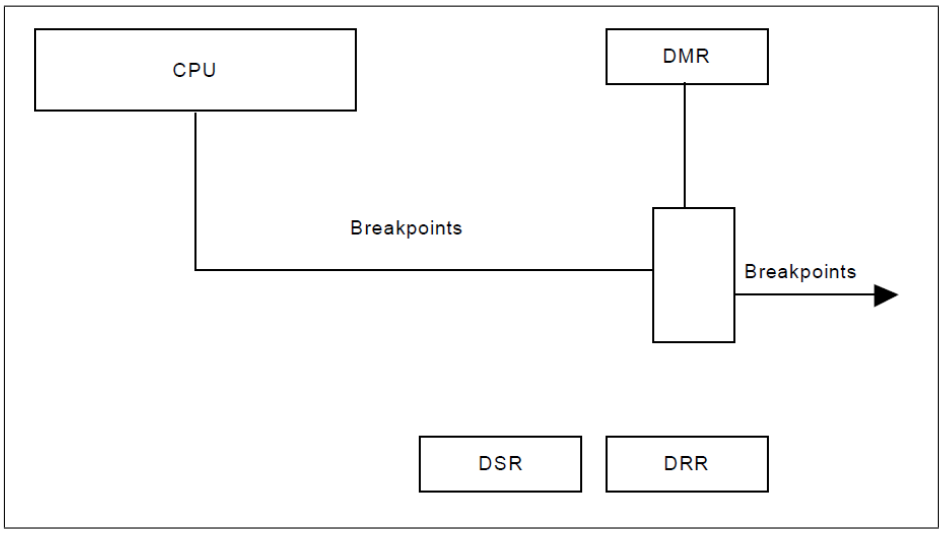
Όταν το λογισμικό εκκινεί την κατάσταση Doze, τότε τα προγράμματα που τρέχουν στον επεξεργαστή αναστέλλονται. Τα ρολόγια στις εσωτερικές ενότητες (modules) του επεξεργαστή απενεργοποιούνται εκτός του χρονιστή (tick timer). Ωστόσο οποιοδήποτε άλλο μπλόκ πάνω στο ολοκληρωμένο κύκλωμα συνεχίζει να λειτουργεί κανονικά. Ο επεξεργαστής OR1200 θα φύγει από την κατάσταση doze και θα επανέλθει σε κανονική κατάσταση όταν κάποια διακοπή (interrupt) παρουσιαστεί.

Στην κατάσταση Sleep, όλες οι εσωτερικές μονάδες του OR1200 απενεργοποιούνται και τα ρολόγια οριοθετούνται (clocks gated). Προαιρετικά (ανάλογα με την υλοποίηση) μπορεί να χαμηλωθεί η παροχή της τάσης στον πυρήνα του OR1200. Ο επεξεργαστής OR1200 θα φύγει από την κατάσταση doze και θα επανέλθει σε κανονική κατάσταση όταν κάποια διακοπή (interrupt) παρουσιαστεί.

Η λειτουργία Dynamic Clock gating δεν υποστηρίζεται προς το παρών από τον OR1200.

2.2.9 Μονάδα Αποσφαλμάτωσης

Η μονάδα αποσφαλμάτωσης βοηθάει τους προγραμματιστές λογισμικού διορθώσουν λάθη στο σύστημα τους. Παρέχει την βασική βοήθεια για αποσφαλμάτωση, χωρίς να παρέχει όμως προηγμένη τεχνολογία σύμφωνα με την αρχιτεκτονική του OpenRISC 1000 όπως watchpoints, breakpoints και πρόγραμμα ελέγχου της ροής των καταχωρητών ελέγχου.



Σχήμα 2.11: Μπλόκ διάγραμμα μονάδας αποσφαλμάτωσης.

2.2.10 Σήματα Χρονισμού και Επανεκκίνησης

Ο πυρήνας του OR1200 χρησιμοποιεί ένα ρολόι για τον χρονισμό κάθε μιας διεπαφή του διαύλου επικοινωνίας Wishbone τόσο για τα δεδομένα όσο και για τις εντολές. Το σήμα ρολογιού *clk_cpu* χρονίζει οτιδήποτε βρίσκεται μέσα στην διεπαφή του διαύλου Wishbone. Η διεπαφή διαύλου δεδομένων Wishbone χρονίζεται από το σήμα *dwd_clk_i*, ενώ των εντολών με το σήμα *iwd_clk_i*.

Ο επεξεργαστής OR1200 παρέχει ένα ασύγχρονο σήμα επανεκκίνησης του συστήματος (asynchronous reset signal) , *rst*. Όταν αυτό σήμα είναι ενεργοποιημένο τότε αμέσως επαναφέρει (resets) όλα τα flip-flops μέσα στον OR1200. Όταν είναι απενεργοποιημένο τότε ο επεξεργαστής λειτουργεί κανονικά.

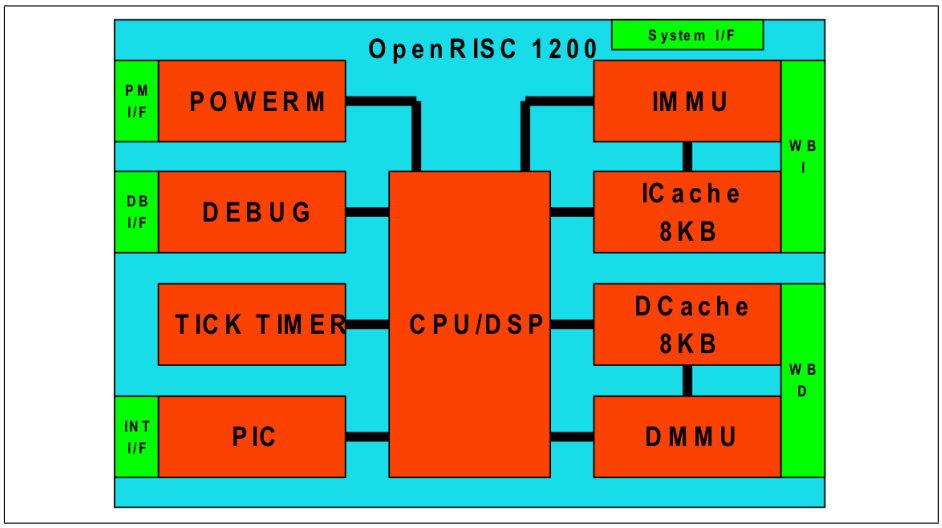
2.2.11 Δίαυλος επικοινωνίας Wishbone

Δύο διεπαφές Wishbone του επεξεργαστή OR1200 συνδέουν τον πυρήνα με τα περιφερειακά και το εξωτερικό υποσύστημα μνήμης. Αυτές είναι συμβατές το WISHBONE SoC Interconnection specification Rev. B3. Η υλοποίηση παρέχει ένα δίαυλο επικοινωνίας πλάτους 32 bit χωρίς να υποστηρίζει άλλα πλάτη.

2.3 I/O Θύρες

Ο επεξεργαστής OR1200 προσφέρει τις παρακάτω διεπαφές για σύνδεση με εξωτερικές λειτουργίες.

- Instruction and data WISHBONE host interfaces
- Power management interface
- Development interface
- Interrupts interface



Σχήμα 2.12: Core's Interfaces

Σε αυτή την ενότητα θα αναλυθούν όλες οι διεπαφές που προσφέρει ο επεξεργαστής OR1200 εκτός από την διεπαφές δεδομένων και εντολών Wishbone που θα περιγραφούν στην κεφάλαιο 4 στην ενότητα 4.3.

2.3.1 Διεπαφή Συστήματος (System I/F)

Με την διεπαφή συστήματος συνδέονται στο επεξεργαστή OR1200 σήματα όπως το reset, το ρολόι (clock) και τα υπόλοιπα που φαίνονται στον παρακάτω πίνακα.

Port	Width	Direction	Description
Rst	1	Input	Asynchronous reset
clk_cpu	1	Input	Main clock input to the RISC
clk_dc	1	Input	Data cache clock
clk_ic	1	Input	Instruction cache clock
clk_dmmu	1	Input	Data MMU clock
clk_immu	1	Input	Instruction MMU clock
clk_tt	1	Input	Tick timer clock

Πίνακας 2.11: Σήματα διεπαφής συστήματος.

2.3.2 Διεπαφή Ανάπτυξης (Development DB I/F)

Η διεπαφή ανάπτυξης συνδέει την εξωτερική θύρα εντοπισμού σφαλμάτων με την εσωτερική μονάδα αποσφαλμάτωσης του επεξεργαστή OR1200. Η μονάδα αποσφαλμάτωσης όπως αναφέρθηκε και παραπάνω επιτρέπει τον έλεγχο της εκτέλεση προγραμμάτων μέσα στον RISC επεξεργαστή, των καθορισμό breakpoints και watchpoints όπως και την ροή δεδομένων και εντολών. Στον παρακάτω πίνακα φαίνονται τα σήματα που παρέχει αυτή η διεπαφή.

Port	Width	Direction	Description
dbg_dat_o	32	Output	Transfer of data from RISC to external development interface
dbg_dat_i	32	Input	Transfer of data from external development interface to RISC
dbg_adr_i	32	Input	Address of special-purpose register to be read or written
dbg_op_I	3	Input	Operation select for development interface
dbg_lss_o	4	Output	Status of load/store unit
dbg_is_o	2	Output	Status of instruction fetch unit
dbg_wp_o	11	Output	Status of watchpoints
dbg_bp_o	1	Output	Status of the breakpoint
dbg_stall_i	1	Input	Stalls RISC CPU core
dbg_ewt_i	1	Input	External watchpoint trigger

Πίνακας 2.12: Development Interface

2.3.3 Διεπαφή Ελέγχου Ενέργειας (Power Management I/F)

Η διεπαφή ελέγχου ενέργειας (Power Management Interface) παρέχει σήματα που συνδέουν τον επεξεργαστή OR1200 με εξωτερικές πηγές ενέργειας. Οι εξωτερικές

πηγές ενέργειας χρειάζονται για την υλοποίηση συναρτήσεων των οποίων η τεχνολογία δεν επιτρέπει την υλοποίηση τους μέσα στον πυρήνα. Στον παρακάτω πίνακα φαίνονται τα σήματα που παρέχει αυτή η διεπαφή.

Port	Width	Direction	Generation	Description
pm_clksd	4	Output	Static (in SW)	Slow down outputs that control reduction of RISC clock frequency
pm_cpustall	1	Input	-	Synchronous stall of the RISC's CPU core
pm_dc_gate	1	Output	Dynamic (in HW)	Gating of data cache clock
pm_ic_gate	1	Output	Dynamic (in HW)	Gating of instruction cache clock
pm_dmmu_gate	1	Output	Dynamic (in HW)	Gating of data MMU clock
pm_immu_gate	1	Output	Dynamic (in HW)	Gating of instruction MMU clock
pm_tt_gate	1	Output	Dynamic (in HW)	Gating of tick timer clock
pm_cpu_gate	1	Output	Static (in SW)	Gating of main CPU clock
pm_wakeup	1	Output	Dynamic (in HW)	Activate all clocks
pm_lvolt	1	Output	Static (in SW)	Lower voltage

Πίνακας 2.13: Σήματα διεπαφής ελέγχου ενέργειας

2.3.4 Διεπαφή Διακοπών (Interrupt I/F)

Η διεπαφή Διακοπών (Interrupt interface) έχει σαν εισόδους τις διακοπές που προκαλούν οι εξωτερικές περιφερειακές μονάδες και τις καθοδηγούν εσωτερικά στον επεξεργαστή για να τις επεξεργαστή. Όλες οι διακοπές δημιουργούνται στην θετική ακμοπυροδότηση του κεντρικού ρολογιού του RISC επεξεργαστή. Στον παρακάτω πίνακα φαίνονται τα σήματα που παρέχει αυτή η διεπαφή.

Port	Width	Direction	Description
pic_ints	PIC_INTS	Input	External interrupts

Πίνακας 2.14: Σήματα διεπαφής διακοπών

2.4 Παραμετροποίηση Επεξεργαστή

Σε αυτή την ενότητα θα γίνει αναφορά στα χαρακτηριστικά του επεξεργαστή που μπορούν να παραμετροποιηθούν σε επίπεδο υλικού από τον χρήστη. Στο παρακάτω πίνακα φαίνονται ποια είναι αυτά τα χαρακτηριστικά και ποιες οι αντίστοιχες μεταβλητές τους.

Variable Name	Range	Default	Description
EADDR_WIDTH	32	32	Effective address width
VADDR_WIDTH	32	32	Virtual address width
PADDR_WIDTH	24-36	32	Physical address width
DATA_WIDTH	32	32	Data width / Operation width
DC_IMPL	0-1	1	Data cache implementation
DC_SETS	256-1024	512	Data cache number of sets
DC_WAYS	1	1	Data cache number of ways
DC_LINE	16-32	16	Data cache line size
IC_IMPL	0-1	1	Instruction cache implementation
IC_SETS	32-1024	512	Instruction cache number of sets
IC_WAYS	1	1	Instruction cache number of ways
IC_LINE	16-32	16	Instruction cache line size in bytes
DMMU_IMPL	0-1	1	Data MMU implementation
DTLB_SETS	64	64	Data TLB number of sets
DTLB_WAYS	1	1	Data TLB number of ways
IMMU_IMPL	0-1	1	Instruction MMU implementation
ITLB_SETS	64	64	Instruction TLB number of sets
ITLB_WAYS	1	1	Instruction TLB number of ways
PIC_INTS	2 – 32	20	Number of interrupt inputs

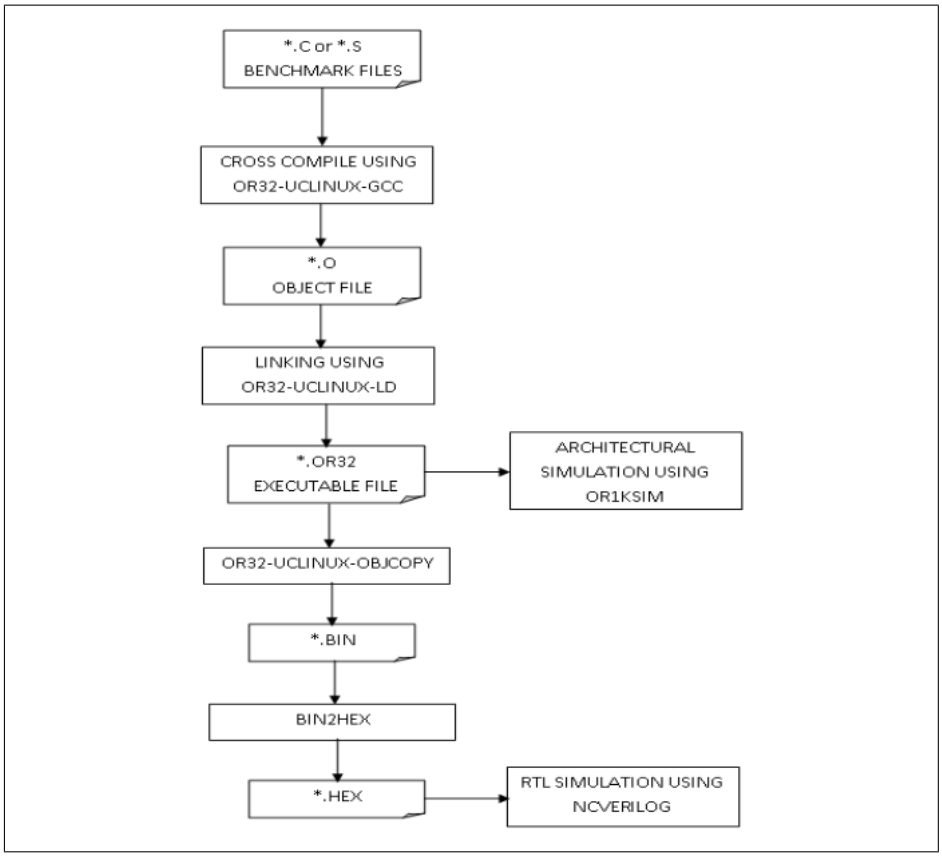
Πίνακας 2.15: Παραμετροποίηση Υλικού.

3 OR1200 Προσομοίωση

3.1 Περιβάλλον Προσομοίωσης

Υπάρχουν δύο περιβάλλοντα με τα οποία γίνετε η προσομοίωση του OR1200 επεξεργαστή. Το πρώτο χρησιμοποιεί τον OpenRISC αρχιτεκτονικό προσομοιωτή *or1ksim* και το δεύτερο χρησιμοποιεί τον *NC-Verilog* προσομοιωτή που κάνει προσομοίωση με βάση το υλικό (hardware based simulation). Στο πρώτο περιβάλλον γίνετε η επαλήθευση της λειτουργικότητας των benchmarks και στο δεύτερο προσομοιώνεται η ουσιαστική λειτουργία του OR1200 σε επίπεδο υλικού με βάση το benchmark που εκτελέσαμε.

Η συνολική ροή της προσομοίωσης παρουσιάζεται στο Σχήμα 3.1. Τα benchmarks είναι είτε .C αρχεία είτε .S αρχεία. Αυτά τα αρχεία, αρχικά γίνονται cross-compiled χρησιμοποιώντας την εντολή or32-uclinux-gcc και παράγουν ένα .O object αρχείο. Το object αρχείο μετά μετατρέπεται σε ενα .OR32 εκτελέσιμο αρχείο χρησιμοποιώντας την συνδετική εντολή or32-uclinux-ld. Αυτό το εκτελέσιμο αρχείο χρησιμοποιείται από τον or1k αρχιτεκτονικό εξομοιωτή. Περαιτέρω το αρχείο .OR32 μετατρέπεται σε δυαδικό (binary) αρχείο χρησιμοποιώντας την εντολή or32-uclinux-objcopy. Στο τέλος δημιουργείται ένα .HEX αρχείο χρησιμοποιώντας τον binary to hex μετατροπέα bin2hex. Το παραγόμενο αρχείο .HEX φορτώνεται στην flash μνημη του RTL κώδικα του OR1200 επεξεργαστή και μετά γινετε η προσομοίωση με βάση το υλικό.



Σχήμα 3.1: Συνολική ροή προσομοίωσης.

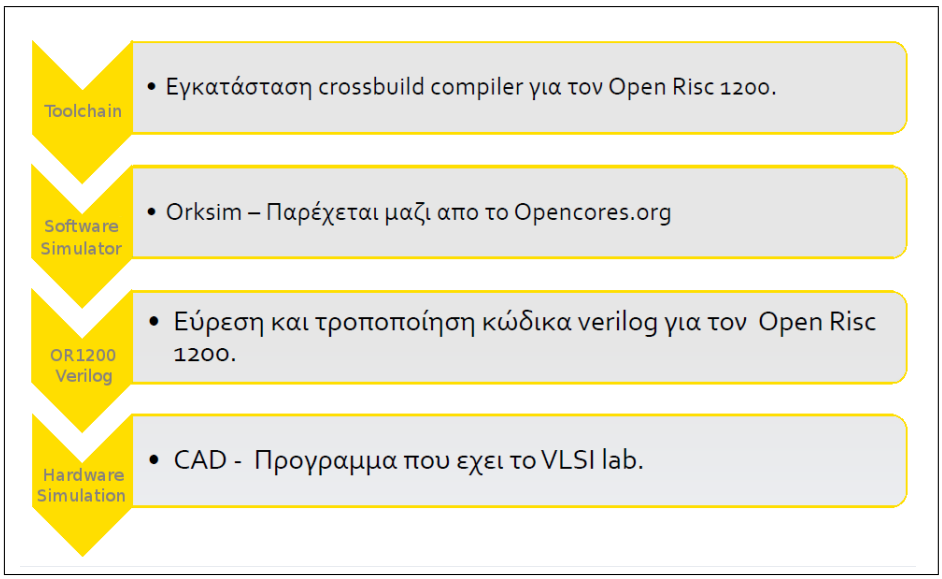
Η συνολική ροή της προσομοίωσης ελέγχετε από το Makefile που παρέχεται από το Opencores μαζί με το πακέτο[1] που περιέχει το επεξεργαστή OR1200. Το συγκεκριμένο Makefile δημιουργεί τα απαραίτητα αρχεία όπως περιγράφηκαν παραπάνω (Σχήμα 3.1) και τα οδηγεί τόσο στον αρχιτεκτονικό προσομοιωτή όσο και στον προσομοιωτή υλικού[2] ώστε να γίνει η προσομοίωση.

[1]Για την δομή των αρχείων σε αυτό το πακέτο δείτε την ενότητα 3.3

[2]Το Makefile τροποιήθηκε όπως φαίνεται στην υπο-ενότητα 3.2.2 για να υποστηρίζει τον προσομοιωτή υλικού NcVerilog

3.2 Εγκατάσταση Αλυσίδας Προσομοίωσης

Σε αυτό το σημείο θα παρουσιάσουμε τον τρόπο με τον οποίο εγκαταστάθηκαν τα κατάλληλα προγράμματα με τα οποία γίνετε η προσομοίωση του επεξεργαστή OR1200 τόσο σε επίπεδο λογισμικού όσο και σε επίπεδο υλικού.
Να σημειωθεί ότι παρόλο που ακολουθήθηκε ένας καθολικός τρόπος εγκατάστασης τέτοιων προγραμμάτων έγιναν αρκετές παραμετροποιήσεις ώστε να μην επηρεαστεί η σωστή λειτουργία άλλων προγραμμάτων που είναι εγκατεστημένα στο εργαστήριο μικροηλεκτρονικής. Στο Σχήμα 3.2 φαίνεται η βασική διαδικασία που ακολουθήθηκε.



Σχήμα 3.2: Διαδικασία Εγκατάστασης Προγραμμάτων

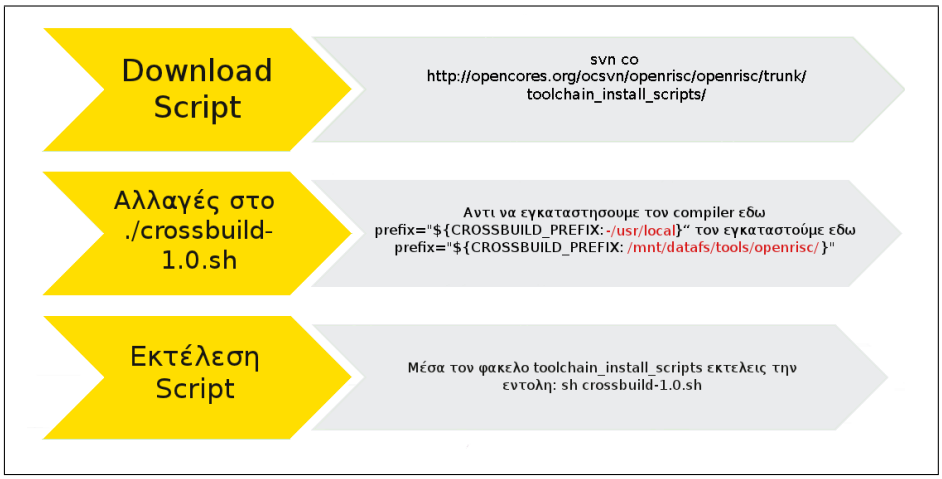
3.2.1 Εγκατάσταση Αρχιτεκτονικού Προσομοιωτή - Or1ksim

Ο αρχιτεκτονικός προσομοιωτής ουσιαστικά είναι ένας τροποποιημένος gcc compiler που συνυπάρχει παράλληλα (crossbuild) με τον gcc compiler του συστήματος και βοηθά στην εκτέλεση αρχείων κώδικα assembly του επεξεργαστή Open Risc 1200.

Η διαδικασία που ακολουθήθηκε είναι η παρακάτω:

1. Κατέβασμα του απαραίτητου script από το www.opencores.org που θα εγκαταστήσει τα απαραίτητα πακέτα για την σωστή λειτουργία του προσομοιωτή.
2. Αλλαγές στο script για την εγκατάσταση των πακέτων σε τέτοιο σημείο στο σύστημά μας ώστε να μπορούν να το χρησιμοποιούν όλοι οι χρήστες του εργαστηρίου.
3. Εκτέλεση του script σε terminal.

Στο Σχήμα 3.3 φαίνεται ακριβώς η παραμετροποίηση που έγινε και το πως εκτελέστηκαν τα παραπάνω βήματα.



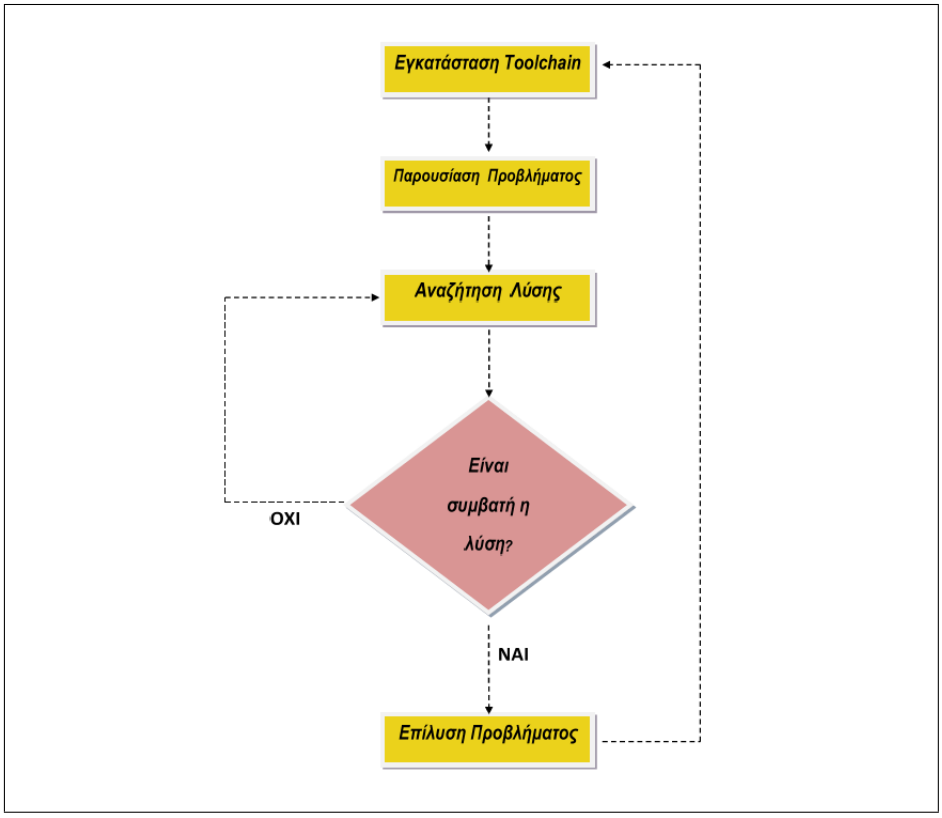
Σχήμα 3.3: Εντολές εγκατάστης του Or1ksim.

Σε αυτό το σημείο είναι απαραίτητο να σημειωθεί ότι η εκτέλεση του δεύτερου βήματος ήταν από τις πιο χρονοβόρες διαδικασίες που έγιναν για την εκπόνηση αυτής της διπλωματικής εργασίας. Οι λόγοι είναι οι παρακάτω:

1. Για την εκτέλεση του script που θα εγκαθιστούσε τον τροποποιημένο GCC compiler για τον OR1200 έπρεπε να αναβαθμιστεί ο υπάρχων GCC compiler και αρκετές άλλες βιβλιοθήκες ώστε να υποστηρίζει στην εκτέλεση κάποιων εσωτερικών εντολών του script.

2. Ο τροποποιημένος GCC compiler θα έπρεπε να λειτουργεί αρμονικά με τον αναβαθμισμένο GCC compiler του συστήματος του εργαστηρίου μικροηλεκτρονικής.

Οι παραπάνω λόγοι αποτελούσαν προβλήματα κατά το στάδιο της εγκατάστασης του αρχιτεκτονικού προσομοιωτή Or1ksim επειδή η εγκατάσταση του έπρεπε να γίνει σε ένα υπάρχων υπολογιστικό σύστημα (του εργαστηρίου μικροηλεκτρονικής) και όχι σε ένα καινούργιο σύστημα. Αυτό είχε σαν συνέπεια να έπρεπε να διατηρηθεί η συμβατότητα των καινούργιων προγραμμάτων με τα προγράμματα που ήταν ήδη εγκατεστημένα στο εργαστήριο και χρησιμοποιούνταν για άλλους σκοπούς. Αυτό αντιμετωπίστηκε με την χρήση της μεθόδου trial and error όπως παρουσιάζεται στο Σχήμα 3.4 παρακάτω.



Σχήμα 3.4: Trial and Error Method

3.2.2 Προσομοιωτής Υλικού NcVerilog

Όπως αναφέρθηκε και στην ενότητα 3.1 η συνολική ροή της προσομοίωσης ελέγχεται από το Makefile που βρίσκεται στο *./sim/bin/* στον φάκελο που περιέχει τον επεξεργαστή OR1200. Το Makefile που παρέχει ελεύθερα το Opencores έχει υλοποιηθεί με τέτοιο τρόπο ώστε στο στάδιο της προσομοίωσης υλικού να δίνει τον έλεγχο είτε στο ICARUS[3] είτε στο Modelsim[4].

Ο προσομοιωτής υλικού που είναι εγκατεστημένος στο εργαστήριο μικροηλεκτρονικής και χρησιμοποιείται για ακαδημαϊκούς σκοπούς είναι ο NcVerilog της εταιρείας Cadence Design Systems που υποστηρίζει γλώσσες περιγραφής υλίκου (HDL) όπως η Verilog και η VHDL.

Όπως γίνετε εύκολα αντιληπτό από τα παραπάνω για να χρησιμοποιηθεί ο NcVerilog προσομοιωτής έπρεπε να τροποποιηθεί το Makefile με τέτοιο τρόπο ώστε να δημιουργεί τα κατάλληλα αρχεία και μετά να τα οδηγεί στο NcVerilog και να εκτελεί εκεί τις κατάλληλες εντολές.

Το παρακάτω κομμάτι κώδικα προστέθηκε στο Makefile που βρίσκεται στο *./sim/bin/* ώστε να ενσωματώσει τον προσομοιωτή υλικού NcVerilog. Ουσιαστικά συλλέγει τα μονοπάτια (paths) από όλα τα αρχεία που χρειάζονται και τα τροφοδοτεί στο NcVerilog σαν ορίσματα στις εντολές του για να γίνει η προσομοίωση σε επίπεδο υλικου.

```
1  ################################################
2  # NC-verilog simulator build and run rules     #
3  ################################################
4  # Compile script generation rules:

6  ncverilog_dut.scr: rtl $(RTL_VERILOG_SRC) $(RTL_VERILOG_INCLUDES) $(
       BOOTROM_VERILOG)
7      $(Q)echo "+incdir+"$(RTL_VERILOG_INCLUDE_DIR) > $@;
8      $(Q)echo "+incdir+"$(RTL_SIM_SRC_DIR) >> $@;
9      $(Q)echo "+incdir+"$(BOOTROM_SW_DIR) >> $@;
10     $(Q)echo "+incdir+"$(BENCH_VERILOG_INCLUDE_DIR) >> $@;
11     $(Q)echo "+libext+.v" >> $@;
12     $(Q)for module in $(RTL_VERILOG_MODULES); do if [ -d $(
           RTL_VERILOG_DIR)/$$module ]; then echo "-y " $(RTL_VERILOG_DIR)
           /$$module >> $@; echo $(RTL_VERILOG_DIR)/$$module >> files.txt
           ; fi; done
13     $(Q)for module in $(RTL_VERILOG_MODULES); do if [ -d $(
           RTL_VERILOG_DIR)/$$module ]; then echo  $(RTL_VERILOG_DIR)/
           $$module/*.v >> $@; echo $(RTL_VERILOG_DIR)/$$module >> files.
           txt ; fi; done
14     $(Q)echo >> $@

16 ncverilog_bench.scr: $(BENCH_VERILOG_SRC)
17     $(Q)echo "+incdir+"$(BENCH_VERILOG_DIR) > $@;
18     $(Q)for path in $(BENCH_VERILOG_SRC_SUBDIRS); do echo "+incdir+"
           $$path >> $@; done
19     $(Q)for path in $(BENCH_VERILOG_SRC_SUBDIRS); do echo "-y "$$path
           >> $@; done
20     $(Q)echo "+incdir+"$(RTL_VERILOG_INCLUDE_DIR) >> $@;
21     $(Q)echo "+incdir+"$(RTL_SIM_SRC_DIR) >> $@;
22     $(Q)echo "+libext+.v" >> $@;
23     $(Q)echo "-y" $(RTL_SIM_SRC_DIR) >> $@;
```

[3]Προσομοιωτής υλικού από τον Stephen Williams. Διανέμεται ελεύθερα - Open source programm.
[4]Προσομοιωτής υλικού από την εταιρεία Mentor Graphics. Δεν διανέμεται ελεύθερα.

```
24      $(Q)for vsrc in $(BENCH_VERILOG_SRC); do echo $$vsrc >> $@; done
25      $(Q)echo >> $@

27  # Compile DUT into "work" library
28  worklib: ncverilog_dut.scr #$(RTL_VHDL_SRC)
29  #   $(Q)if [ ! -e $@ ]; then mkdir -p INCA_libs/$@; fi
30  #   $(Q)echo; echo "\t### Compiling VHDL design library ###"; echo
31  #   $(Q)ncvhdl -93 $(QUIET) $(RTL_VHDL_SRC)
32  #   $(Q)echo; echo "\t### Compiling Verilog design library ###"; echo
33  #   $(Q)ncverilog $(QUIET) -f $< $(DUT_TOP)
34      $(Q)ncverilog +nctimescale+1ns/10ps +ncoverride_timescale -f $< $(
            DUT_TOP)

36  # Single compile rule
37  .PHONY : $(NCVERILOG)
38  $(NCVERILOG): ncverilog_bench.scr $(TEST_DEFINES_VLG) $(MGC_VPI_LIB)
        worklib
39      $(Q)echo; echo "\t### Compiling testbench ###"; echo
40      $(Q)ncverilog -c  $(QUIET) $(BENCH_TOP) -f $<
41      $(Q)ncelab $(QUIET) $(RTL_TESTBENCH_TOP) $(VOPT_ARGS) -o tb
42      $(Q)echo; echo "\t### Launching simulation ###"; echo
43      $(Q)ncsim $(MGC_VSIM_ARGS) tb
```

3.3 Δομή Αρχείων

Σε αυτό το μέρος θα παρουσιάσουμε την δομή και τα περιεχόμενα του φακέλου που αποτελεί τον επεξεργαστή OR1200.

Το συγκεκριμένο πακέτο περιέχει:

- **Rtl** : Εδώ βρίσκεται ο κώδικας verilog που περιγράφει σε υλικό τον επεξεργαστή OR1200.
- **Boards** : Περιέχει κατάλληλα αρχεία για να περάσεις τον OR1200 σε συγκεκριμένες πλακέτες FPGA.
- **Sim** : Σε αυτό τον φάκελο δημιουργούνται τα αποτελέσματα της προσομοίωσης σύμφωνα με το Makefile που δημιουργήσαμε για τους σκοπούς του εργαστηρίου.
    1. *sim/bin* : Εδώ βρίσκεται το Makefile που δημιουργήσαμε για τους σκοπούς του εργαστηρίου και χειρίζεται την λειτουργία του επεξεργαστή.
    2. *sim/run* :Εδώ εκτελούνται όλες οι εντολές που σχετίζονται με την προσομοίωση του επεξεργαστή.
    3. *sim/out* :Εδώ τοποθετούνται όλα τα αρχεία που παράγονται μετά την προσομοίωση. Να σημειωθεί ότι οι κυματομορφές τοποθετούνται στο sim/run.
- **Sw** : Εδώ βρίσκονται τα βασικότερα αρχεία που είναι απαραίτητα για το cross-compilation και την σωστή λειτουργία του OpenRISC επεξεργαστή.
    1. *sw/drivers* :Εδώ βρίσκονται οι drivers και τα εργαλεία για τροποποίηση του hardware.

2. *sw/lib* :Εδώ βρίσκεται μια απλή βιβλιοθήκη που σε συνδυασμό με τους drivers κατά την διάρκεια του compile δημιουργούν την βιβλιοθήκη liborpsoc που τοποθετείται στο *sw/lib*.
3. *sw/lib/include* :Εδώ βρίσκεται το αρχείο cpu-utils.h που περιέχει όλες τις συναρτήσεις σχετικές με την CPU του OpenRISC.
4. *sw/tests* :Εδώ βρίσκεται το λογισμικό που χρησιμοποιείται (.C και .S αρχέια) για να δοκιμαστεί η σωστή λειτουργία του επεξεργαστή (testing) σε υπομονάδες όπως ethmac, or1200,sdram, spi και uart. Στον φάκελο κάθε υπομονάδας (πχ sw/test/sdram) υπάρχουν δύο υποφάκελοι board και sim (πχ sw/test/sdram/board και sw/test/sdram/sim). Στον sim φάκελο υπάρχουν τα tests που εκτελούνται κατά την προσομοίωση του επεξεργαστή σε ένα PC και στον φάκελο board υπάρχουν τα tests που εκτελούνται κατά την λειτουργία του επεξεργαστή.

- **Doc** : Εδώ βρίσκεται documentation που χρειαζόμαστε για να καταλάβουμε την φύση των testbenches και οι οδηγίες για το simulation σύμφωνα με το Makefile που δημιουργήσαμε για τους σκοπούς του εργαστηρίου.

3.4 Εντολές Προσομοίωσης

3.4.1 Η βασική διαδικασία

Η διαδικασία με την οποία γίνετε η προσομοίωση του OR1200 είναι η εξής:

1. Το Makefile που ελέγχει την προσομοίωση βρίσκετε στο */sim/bin/* και είναι προσπελάσιμο και από το */sim/run/* .
2. Στον φάκελο */sim/run/* εκτελώντας την εντολή *make rtl-tests* κάνει compile τον rtl (verilog) κώδικά του OR1200 και εκτελεί όλα τα testbenches (assembly) που βρίσκονται στο *sw/tests/or1200*.
3. Τα αποτελέσματα του παραπάνω βήματος τοποθετούνται στο */sim/out/*. Αυτά είναι (στα αγγλικά για καλύτερη κατανόηση):
   - *test-name-executed.log* : A trace of the processor after each executed instruction
   - *test-name-sprs.log* : A list of processor special purpose registers (SPR) accesses is created
   - *test-name-lookup.log* : A list of when each instruction was executed is generated
   - *test-name-general.log* : The use of the processor's report mechanism is commonplace in the test software, as it allows for the checking of intermediate values after simulation.

3.4.2 Εκτέλεση ενος συγκεκριμένου test

Η εξομοίωση ενός συγκεκριμένου test γίνετε με την παρακάτω εντολή.

```
make rtl-test TEST=test-name
```

Πρέπει το αρχείο *test-name.c*[5] (ή *test-name.s*)να είναι τοποθετημένο στο *sw/tests/module /sim/* όπου module είναι η υπομονάδα που θέλουμε να ελέξουμε με κάποιο από τα tests που μας παρέχει (πχ sw/tests/sdram/sim/sdram-rows.c).

3.4.3 Εκτέλεση συγκεκριμένων tests μαζί

Η εξομοίωση πολλών συγκεκριμένων tests γίνετε με την εντολή *make rtl-test TEST=" test-name1 test-name2 ...".*

```
make rtl-tests TESTS="sdram-rows uart-simple or1200-mmu or1200-fp"
```

3.4.4 Παρέχοντας μια προσαρμοσμένη VMEM εικόνα (image)

Είναι δυνατό να καθορίσουμε το μονοπάτι μιας ήδη υπάρχουσας VMEM εικόνας που θα την χρησιμοποιήσουμε αντί να κάνουμε πάλι από την αρχή test το software. Χρησιμοποιώντας την μεταβλητή *USER_VMEM* μπορούμε να καθορίσουμε το μονοπάτι της VMEM εικόνας που θέλουμε να τρέξουμε. Για παράδειγμα

```
make rtl-test USER_VMEM=/path/to/myapp.vmem
```

Αυτή η εικόνα θα αντιγραφεί στο φάκελο στον οποίο εργαζόμαστε και θα μετονομαστεί σύμφωνα με το τι η μνήμη στην εξομοίωση απαιτεί.

3.4.5 Παρέχοντας ένα "precompiled" εκτελέσιμο .ELF αρχείο

Είναι δυνατό να καθορίσουμε το μονοπάτι ενός OR32 ELF εκτελέσιμου αρχείου που θα το χρησιμοποιήσουμε αντί να κάνουμε πάλι από την αρχή test το software. Χρησιμοποιώντας την μεταβλητή *USER_ELF* μπορούμε να καθορίσουμε το μονοπάτι στο οποίο βρίσκετε αυτό το αρχείο. Για παράδειγμα

```
make rtl-test USER_ELF= /path/to/myapp.elf
```

Το ELF αρχείο θα μετατραπεί σε δυαδίκη μορφή και μετά σε VMEM και θα φορτωθεί στο μοντέλο για να εκτελεστέι.

[5] test-name: Είναι η ονομασία του test που θέλουμε να εκτελέσουμε.

3.4.6 Καθαρισμός αρχείων

Με την παρακάτω εντολή μέσα στον φάκελο */sim/run/* μπορούμε να καθαρίσουμε τα περιεχόμενα του φακέλου */sim/out/* που περιέχουν τα αποτελέσματα προηγούμενων προσομοίωσεων που έχουμε κάνει και δεν μας χρειάζονται πια.

```
make clean
```

3.4.7 Κυματομορφές

Παράλληλα με το simulation ενός ή περισσοτέρων testbench παράγονται και οι κυματομορφές των εξομοιώσεων που μας βοηθάνε στην καλύτερη κατανόηση της λειτουργίας του τεστ. Τα αρχεία που παράγονται βρίσκονται σε φακέλους μέσα στο */sim/run/* που έχουν την ονομασία *test-name.shm*. Για να εμφανίσεις τις κυματομορφές αυτές πρέπει μέσω κονσόλας να οδηγηθείς στο /sim/run/ και μετά να εκτελέσεις την παρακάτω εντολή.

```
simvision test-name.shm
```

Να σημειωθεί ότι η παραπάνω οδηγία αναφέρεται όταν έχουμε διαλέξει σαν Simulator τον NcVerilog που είναι εγκατεστημένος στο εργαστήριο. Στην περίπτωση που διαλέξουμε να κάνουμε προσομοίωση υλικού με τον προσομοιωτή Icarus (που είναι δωρεάν) τότε θα πρέπει στην εντολή προσομοίωσης να προσθέσουμε το όρισμα VCD=1. Για παράδειγμα

```
make rtl-test TEST=test-name VCD=1
```

Αυτό θα έχει σαν αποτέλεσμα να δημιουργηθεί μέσα στον φάκελο */sim/out* το αρχείο test-name.vcd το οποίο μπορεί να ανοιχθεί είτε με μια από τις παρακάτω εντολές.

```
gtkwave test-name.vcd &
##########or#############
simvision test-name.vcd
```

3.4.8 Επιπρόσθετες επιλογές στις εντολες

Παρακάτω παρουσιάζονται μερικές μεταβλητές που μας βοηθούν να επιλέξουμε κάποιες συγκεκριμένες λειτουργίες.

- *END_TIME :*Αναγκάζει την εξομοίωση να τερματίσει ($finish).Πχ *make rtl-test TEST="or1200-mul" END_TIME=100* όπου είναι ίδιο με το *#100 $finish* σε αρχείο verilog.
- *DISABLE_PROCESSOR_LOGS :* Απενεργοποιεί την οθόνη παρακολούθησης του επεξεργαστή που συλλέγει πληροφορίες κατά την εκτέλεση μιας εξομοίωσης. Αυτό βοηθάει στην επιτάχυνση της εξομοίωσης αφού απαιτείται λιγότερος χρόνος στην εγγραφή αρχείων και αποτρέπει την δημιουργία πολύ μεγάλων αρχείων σε χρονοβόρες εξομοιώσεις.
- *SIMULATOR :*Επιλέγουμε τον εξομοιωτή υλικού που θέλουμε να χρησιμοποιήσουμε. Προκαθορισμένος εξομοιωτής είναι ο NC-Verilog.Άλλη επιλογή είναι το ICARUS.

3.4.9 Απομόνωση .bin αρχείων

Κατά τον σχεδιασμό εφαρμογών σε συστήματα ειδικού σκοπού είναι σημαντικό να παίρνουμε το αρχείο .bin που δημιουργήθηκε από την προσομοίωση και να το φορτώνουμε κατευθείαν σε ένα ASIC (Application Specific Integrated Circuit) και να κάνουμε την προσομοίωση. Αναλυτικότερα η διαδικασία με την οποία γίνετε αυτό φαίνεται παρακάτω.

1. Αρχικά δημιουργούμε την εφαρμογή είτε σε γλώσσα C είτε σε Assembly (.S) και την αποθηκεύουμε στο *orpsocv2/sw/tests/or1200/sim/*.Για παράδειγμα *test_programm.c ή test_programm.s*
2. Μετά μέσα στο φάκελο */orpsocv2/sw/* και με την παρακάτω εντολή δημιουργούμε το αρχείο test_programm.elf που είναι απαραίτητο για την δημιουργία του .bin αρχείου.

```
make –C ./tests/or1200/sim/ test_programm.elf
```

3. Τέλος για να δημιουργήσουμε το αρχείο .bin καθώς βρισκόμαστε στον φάκελο */orpsocv2/sw/* εκτελούμε την παρακάτω εντολή με την οποία δημιουργείται το .bin αρχείο που είναι και το ζητούμενο και το οποίο τοποθετείται μέσα στον φάκελο *orpsocv2/sw/tests/or1200/sim/*.

```
or32-elf-objcopy –O binary ./tests/or1200/sim/test_programm.elf
test_programm.bin
```

3.4.10 Επιβεβαίωση ορθής προσομοίωσης

Η επιβεβαίωση ορθής προσομοίωσης που θα αναφέρουμε παρακάτω ελέγχει αν το τεστ μας έγινε compile κανονικά, έτρεξε και τελείωσε κανονικά. Για λογικά

λάθη πρέπει ο προγραμματιστής να ελέγξει τα αρχεία που δημιουργούνται κατά το τέλος της αρχιτεκτονικής προσομοίωσης στον φάκελο *sim/out* που καταγράφουν τα περιεχόμενα των καταχωρήτων και τις εντολές που εκτελέστηκαν καθώς και τις κυματομορφές που δημιουργούνται στον φάκελο */sim/run/*.

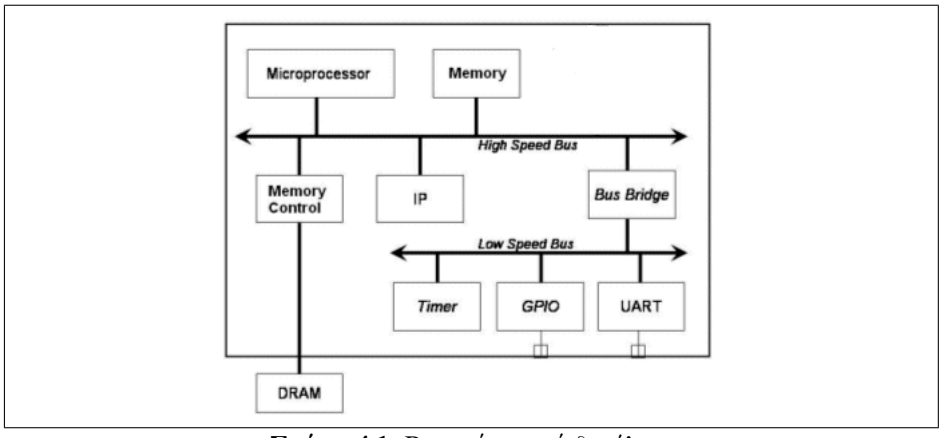
Για να ελένξουμε λοιπόν αν το τεστ μας έγινε compile, έτρεξε και τελείωσε κανονικά πρέπει στο τέλος της προσομοίωσης στην κονσόλα να πάρουμε το παρακάτω μήνυμα.

```
report (0x8000000d);
exit(00000000)
```

4 Διαδικασία Γεφύρωσης

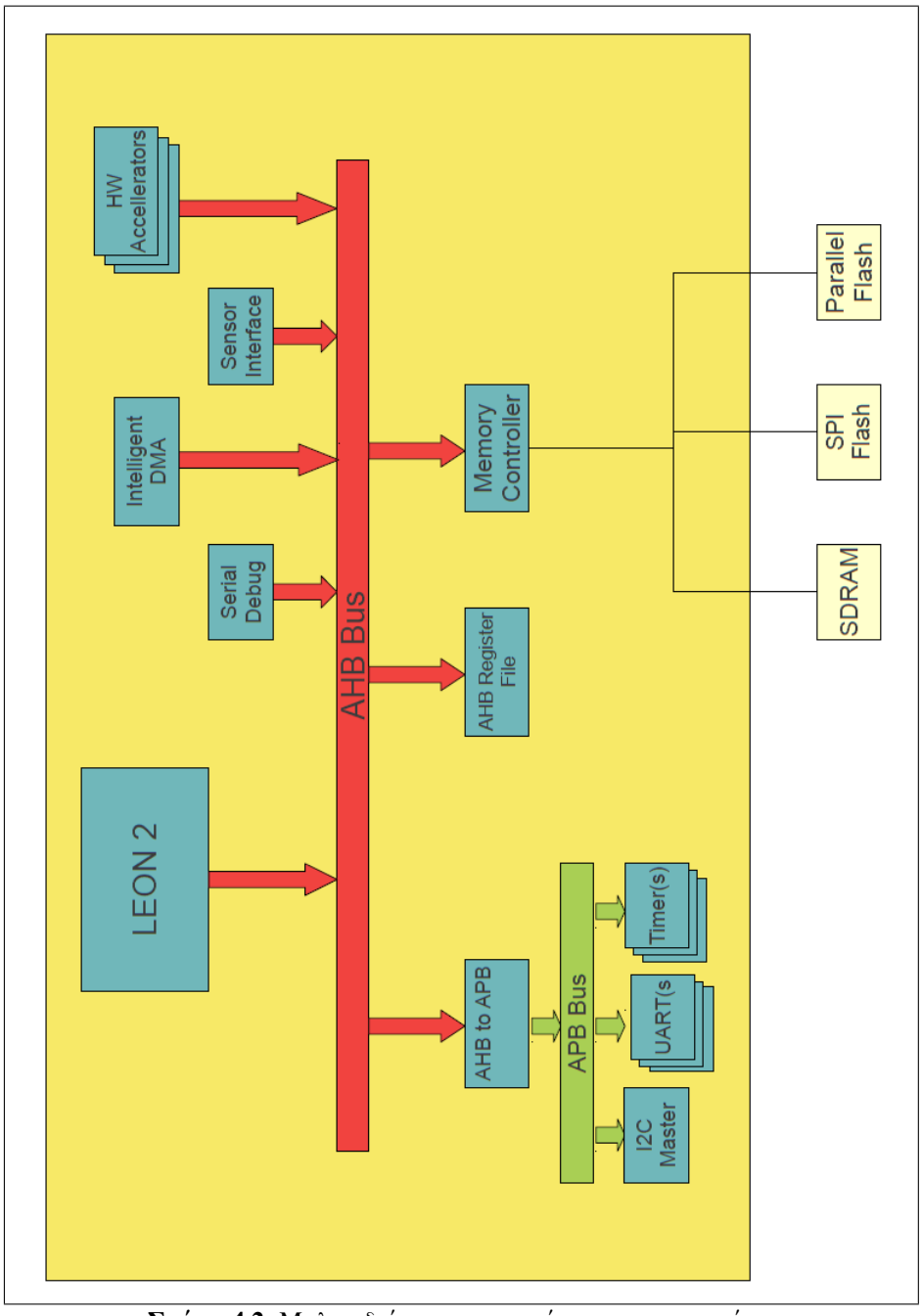
4.1 Σκοπός

Ο επεξεργαστής Or1200 απευθύνεται κυρίως σε ASIC συστήματα. Το κυριότερο συστατικό στοιχείο των συστημάτων ειδικού σκοπού (ASIC) είναι ο δίαυλος επικοινωνίας που χρησιμοποιούν. Πάνω σε αυτόν τον δίαυλο τοποθετούνται όλες οι επιμέρους μονάδες που αποτελούν το ASIC σύστημά μας. Στα μοντέρνα ολοκληρωμένα συστήματα ο δίαυλος επικοινωνίας χωρίζεται σε ένα δίαυλο υψηλής ταχύτητας πάνω στον οποίο τοποθετούνται οι μονάδες που θέλουν συχνή και γρήγορη επικοινωνία με άλλες μονάδες και στον άλλο δίαυλο τοποθετούνται μονάδες που χρησιμοποιούν κυρίως αργά πρωτοκολλά και δεν είναι άμεσα απαραίτητες για την σωστή λειτουργία τους συστήματος. Παρακάτω φαίνεται μια γενική απεικόνιση ενός μοντέρνου συστήματος ειδικού σκοπού.



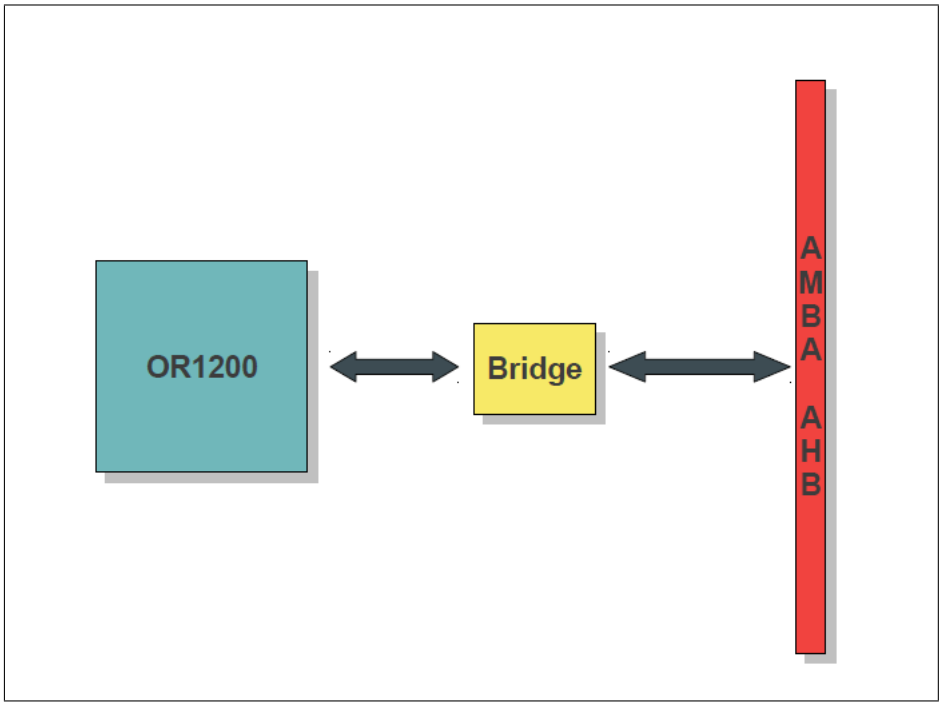
Σχήμα 4.1: Βασική μορφή διαύλου.

Για την δημιουργία ενός ολοκληρωμένου συστήματος επεξεργασίας εικόνας γίνετε εύκολα αντιληπτό ότι ο δίαυλος επικοινωνίας που θα χρησιμοποιηθεί αποτελεί κυρίαρχο στοιχείο. Επειδή η εργασία που γίνετε στα πλαίσια εκπόνησης διπλωματικής εργασίας και ουσιαστικά συνεχίζει-αναβαθμίζει το έργο που ήδη έχει γίνει στο εργαστήριο Μικροηλεκτρονικής θα πρέπει αρχικά να μελετηθεί το υπάρχων σύστημα που υπάρχει διαθέσιμο. Παρακάτω παρουσιάζεται το διάγραμμα του συστήματος που υπάρχει στο εργαστήριο (κάποιες μονάδες είναι ακόμα στο στάδιο της υλοποίησης από άλλες διπλωματικές).



Σχήμα 4.2: Μπλοκ διάγραμμα συστήματος εργαστηρίου.

Παρατηρώντας το παραπάνω διάγραμμα βλέπουμε ότι το σύστημά μας χρησιμοποιεί το AMBA AHB σαν κεντρικό δίαυλο επικοινωνίας. Σκοπός αυτής της διπλωματικής εργασίας ήταν αναβάθμιση του συστήματος αυτού αντικαθιστώντας τον επεξεργαστή LEON 2 με τον επεξεργαστή OR1200. Αυτό για να μπορεί να είναι υλοποιήσημο πρέπει να σχεδιαστεί μια γέφυρα επικοινωνίας μεταξύ του διαύλου AMBA AHB και του επεξεργαστή OR1200. Ο λόγος ύπαρξης αυτής της γέφυρας οφείλεται στο γεγονός ότι ο επεξεργαστής OR1200 είναι σχεδιασμένος με τέτοιο τρόπο ώστε να είναι άμεσα συμβατός με τον Wishbone δίαυλο και όχι με τον AMBA AHB.



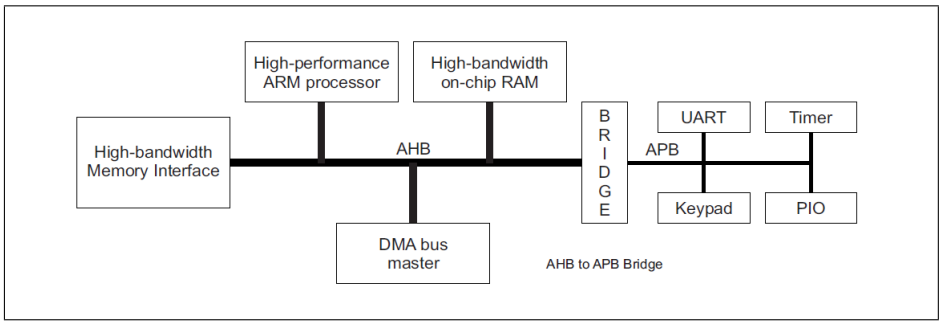
Σχήμα 4.3: Τοποθεσία Γέφυρας Επικοινωνίας

Στις παρακάτω ενότητες θα αναλυθούν τα σήματα που παρέχει τόσο ο δίαυλος AMBA AHB όσο και ο επεξεργαστής OR1200, ο τρόπος με τον οποίο έγινε η σύνδεση τους μέσω της γέφυρας και μια επιτυχημένη μεταφορά δεδομένων μεταξύ του επεξεργαστή και της μνήμης κλείνοντας με αυτόν το τρόπο αυτή την διπλωματική εργασία.

4.2 AMBA AHB Δίαυλος

Ο δίαυλος επικοινωνίας AMBA AHB είναι ένα προϊόν της εταρείας ARM. Στα σύγχρονα συστήματα ειδικού σκοπού αποτελεί ένα βιομηχανικό στάνταρ η ύπαρξη του. Ο δίαυλος AHB είναι ένας δίαυλος που τοποθετείται υψηλότερα (πιο κοντά στο επεξεργαστή) από τον δίαυλο APB και προσφέρει μια πληθώρα από χαρακτηριστικά με στόχο την μεγαλύτερη αποδοτικότητα τους συστήματος. Κάποια από τα χαρακτηριστικά αυτά φαίνονται παρακάτω.

- Μεταφορές δεδομένων κατα ριπές.(burst transfers)
- Ξεχωριστές αναμεταδόσεις. (split transactions)
- Sigle clock edge operation.
- Non-tristate implementation.
- Πιο πλατή δίαυλο δεδομένων (65/128 bits).
- Single cycle bus master handover.



Σχήμα 4.4: A typical AMBA AHB-based system

Αυτός ο δίαυλος επικοινωνίας είναι εγκατεστημένος στο σύστημα στο οποίο δουλεύουμε. Κατά συνέπεια θα πρέπει να μελετηθεί η λειτουργία του καθώς και τα σήματα που μας παρέχει ώστε να σχεδιάσουμε την γέφυρα επικοινωνίας μεταξύ του επεξεργαστή OR1200 και αυτού του διαύλου.

Σχετικά με την λειτουργία του διαύλου AMBA AHB ανατρέξαμε στο εγχειρίδιο που παρέχει η εταιρεία ARM. Όσο αναφορά τα σήματα τα οποία παρέχει και είναι υλοποιημένα στο σύστημα μας φαίνονται παρακάτω.

Πίνακας 4.1: AMBA AHB Signals

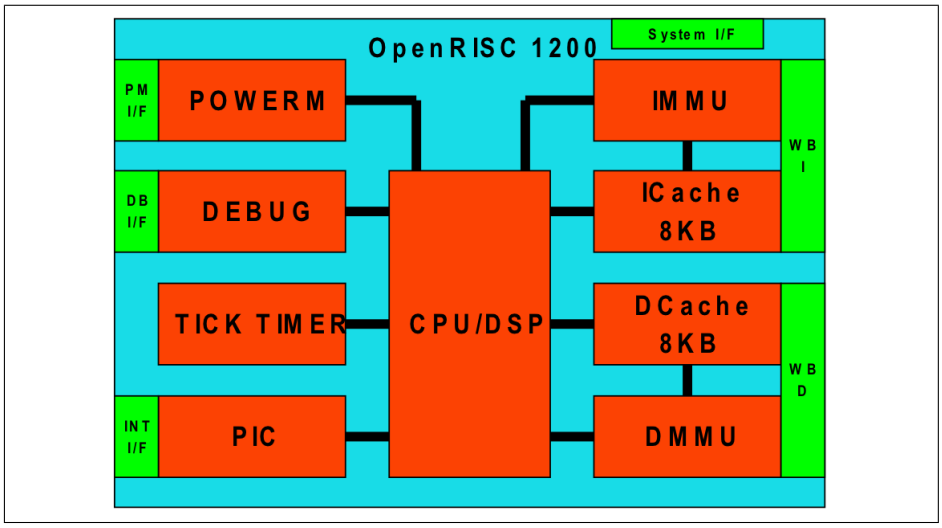
Name	Source	Description
HCLK Bus clock	Clock source	This clock times all bus transfers. All signal timings are related to the rising edge of **HCLK**.
HRESETn Reset	Reset controller	The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.
HADDR[31:0] Address bus	Master	The 32-bit system address bus.
HTRANS[1:0] Transfer type	Master	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE Transfer direction	Master	When HIGH this signal indicates a write transfer and when LOW a read transfer.
HSIZE[2:0] Transfer size	Master	Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes up to a maximum of 1024 bits.
HBURST[2:0] Burst type	Master	Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.
HWDATA[31:0] Write data bus	Master	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HRDATA[31:0] Read data bus	Slave	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HREADY Transfer done	Slave	When HIGH the **HREADY** signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. Note: Slaves on the bus require **HREADY** as both an input and an output signal.

HRESP[1:0] Transfer response	Slave	The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.
HBUSREQx Bus request	Master	A signal from bus master x to the bus arbiter which indicates that the bus master requires the bus. There is an **HBUSREQx** signal for each bus master in the system, up to a maximum of 16 bus masters.
HGRANTx Bus grant	Arbiter	This signal indicates that bus master x is currently the highest priority master. Ownership of the address/control signals changes at the end of a transfer when **HREADY** is HIGH, so a master gets access to the bus when both **HREADY** and **HGRANTx** are HIGH.

4.3 OR1200 - Wishbone Interface

Στο κεφάλαιο 2 περιγράφηκαν αναλυτικά οι I/O θύρες που παρέχει ο επεξεργαστής OR1200 εκτός από τις θύρες που χρησιμοποιούνται για την σύνδεση του επεξεργαστή με τον δίαυλο επικοινωνίας.

Όπως βλέπουμε στο παρακάτω σχήμα ο επεξεργαστής προσφέρει οι θύρες για την επικοινωνία με τον δίαυλο. Η μια είναι για να μπορούν τα δεδομένα να μεταφέρονται σε άλλες μονάδες και η άλλη είναι για την μεταφορά εντολών. Ο λόγος που έχει δυο θύρες επικοινωνίας οφείλεται στο γεγονός ότι ο επεξεργαστής OR1200 ακολουθώντας την λογική της αρχιτεκτονικής Open Risc 1000 αντιμετωπίζει ξεχωριστά τα δεδομένα και τις εντολές (π.χ ξεχωριστή κρυφή μνήμη για τα δεδομένα από την κρυφή μνήμη των εντολών).



Σχήμα 4.5: Core's Architecture

Ο επεξεργαστής OR1200 προσφέρει μια "master" διεπαφή ώστε να μπορεί να συνδέει τον πυρήνα του με το υποσύστημα μνήμης ώστε να προσκομίζει εντολές ή instruction cache lines. Παρακάτω φαίνονται τα σήματα που προσφέρει αυτή η διεπαφή.

Port	Width	Direction	Description
iwb_CLK_I	1	Input	Clock input
iwb_RST_I	1	Input	Reset input
iwb_CYC_O	1	Output	Indicates valid bus cycle (core select)
iwb_ADR_O	32	Outputs	Address outputs
iwb_DAT_I	32	Inputs	Data inputs
iwb_DAT_O	32	Outputs	Data outputs
iwb_SEL_O	4	Outputs	Indicates valid bytes on data bus (during valid cycle it must be 0xf)
iwb_ACK_I	1	Input	Acknowledgment input (indicates normal transaction termination)
iwb_ERR_I	1	Input	Error acknowledgment input (indicates an abnormal transaction termination)
iwb_RTY_I	1	Input	In OR1200 treated same way as iwb_ERR_I.
iwb_WE_O	1	Output	Write transaction when asserted high
iwb_STB_O	1	Outputs	Indicates valid data transfer cycle

Πίνακας 4.2: Instruction WISHBONE Master Interface' Signals

Ο επεξεργαστής OR1200 προσφέρει μια "master" διεπαφή ώστε να μπορεί να συνδέει τον πυρήνα του με το υποσύστημα μνήμης και εξωτερικά περιφερειακά για σκοπούς ανάγνωσης ή εγγραφής δεδομένων και data cache lines. Παρακάτω φαίνονται τα σήματα που προσφέρει αυτή η διεπαφή.

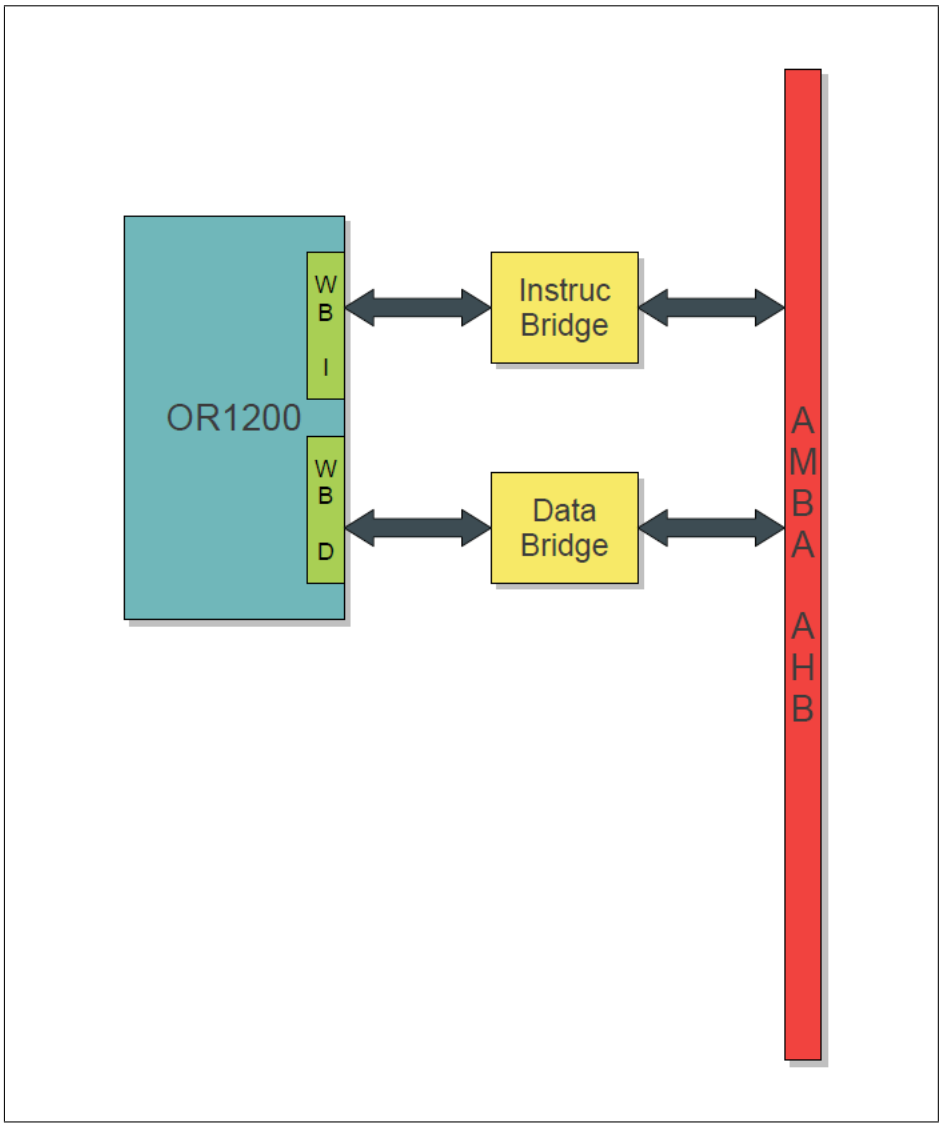
Port	Width	Direction	Description
dwb_CLK_I	1	Input	Clock input
dwb_RST_I	1	Input	Reset input
dwb_CYC_O	1	Output	Indicates valid bus cycle (core select)
dwb_ADR_O	32	Outputs	Address outputs
dwb_DAT_I	32	Inputs	Data inputs
dwb_DAT_O	32	Outputs	Data outputs
dwb_SEL_O	4	Outputs	Indicates valid bytes on data bus (during valid cycle it must be 0xf)
dwb_ACK_I	1	Input	Acknowledgment input (indicates normal transaction termination)
dwb_ERR_I	1	Input	Error acknowledgment input (indicates an abnormal transaction termination)
dwb_RTY_I	1	Input	In OR1200 treated same way as dwb_ERR_I.
dwb_WE_O	1	Output	Write transaction when asserted high
dwb_STB_O	1	Outputs	Indicates valid data transfer cycle

Πίνακας 4.3: Data WISHBONE Master Interface' Signals

4.4 Δημιουργία Γέφυρας

4.4.1 Θεωρητική ανάλυση

Μελετώντας τις παραπάνω ενότητες γίνετε αντιληπτό ότι πρέπει να σχεδιαστούν δύο γέφυρες που θα συνδέουν το επεξεργαστή OR1200 με τον δίαυλο επικοινωνίας AMBA AHB. Η μία γέφυρα θα συνδέει την διεπαφή εντολών που παρέχει ο επεξεργαστής και η άλλη θα συνδέει την διεπαφή δεδομένων όπως φαίνεται και στο παρακάτω σχήμα.



Σχήμα 4.6: Τοποθεσία γεφυρών στο σύστημα μας

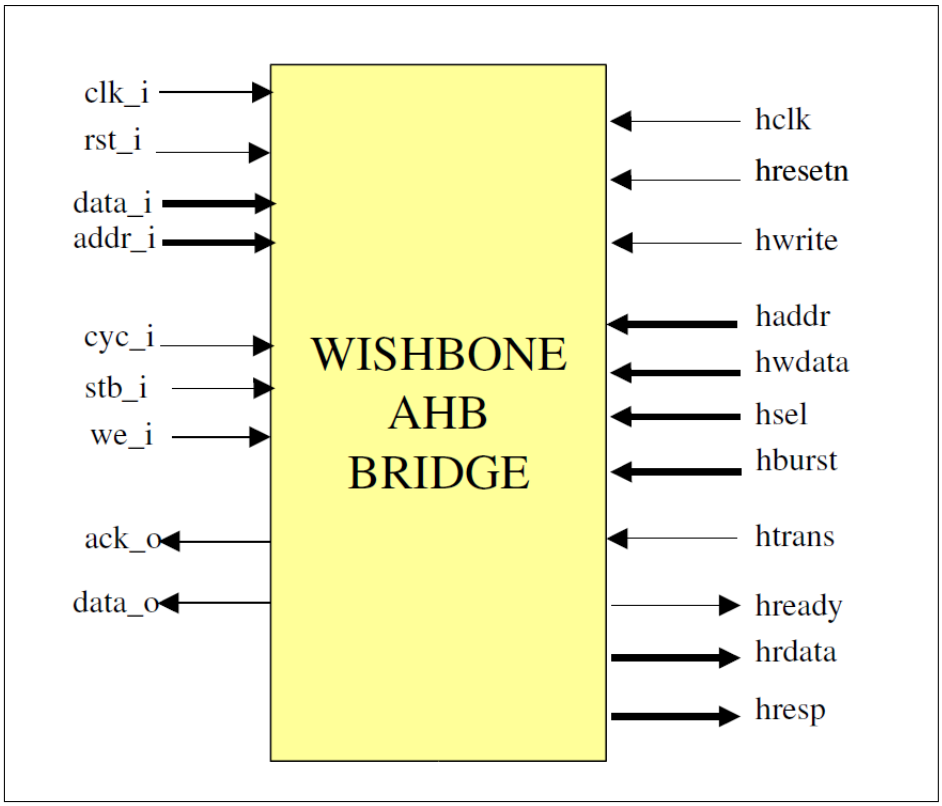
Οι δυο αυτές γέφυρες θα είναι με το ίδιο τρόπο υλοποιημένες άπλα χρείαζονται δύο τέτοιες υπομονάδες για να πετύχουμε την σύνδεση του OR1200 με τον δίαυλο

επικοινωνίας.

Για την υλοποίηση αυτών των γεφυρών επικοινωνίας χρησιμοποιήθηκε σαν βάση μια ήδη υπάρχουσα υλοποίηση από την εταιρεία TooMuch Semiconductor Solutions που παρέχεται ελεύθερα από το Open Cores. Η συγκεκριμένη γέφυρα προσφέρει την απαραίτητη λογική ώστε να μπορεί να γίνει η σύνδεση του επεξεργαστή με τον δίαυλο με τέτοιο τρόπο ώστε να μπορούν να γίνονται απλές μεταφορές δεδομένων μεταξύ του επεξεργαστή και των άλλων υπομονάδων του συστήματος μας. Κάποιοι περιορισμοί που υπάρχουν σε αυτήν την υλοποίηση αλλά δεν αποτελούν πρόβλημα για την δική μας περίπτωση φαίνονται παρακάτω.

- Ο σχεδιασμός αυτός δεν υποστηρίζει την διαδοχική μεταφορά δεδομένων. Αντίθετα μπορεί να μεταφέρει μεγάλα μπλοκ δεδομένων θεωρώντας όμως ότι δεν είναι διαδοχικές μεταφορές δεδομένων.
- Η λέξη (word) έχει μέγεθος 32 bits.
- Δεν υποστηρίζονται οι καταστάσεις λάθους, επανάληψης και διαχωρισμός (error, retry and split).

Το μπλόκ διάγραμμα που περιγράφει την γέφυρα επικοινωνίας καθώς και τα σήματα που χρησιμοποιούνται φαίνονται παρακάτω.



Σχήμα 4.7: Μπλοκ διαγραμμα γέφυρας επικοινωνίας.

Port Name	Direction	Size	Sourse	Description
haddr	out	32	AHB Master	Address Bus
htrans	out	2	AHB Master	Transfer Type
hwrite	out	1	AHB Master	Read/Write enable
hsize	out	3	AHB Master	not used
hburst	out	3	AHB Master	Burst Type
hwdata	out	32	AHB Master	Write data bus
hrdata	out	32	AHB Master	Read data bus
hready	in	32	AHB Slave	To indicate bridge is ready
hresp	in	2	AHB Slave	Response
data_o	out	32	WISHBONE Slave	Read data bus
data_i	in	32	WISHBONE Master	Write data bus
addr_i	in	32	WISHBONE Master	Address Bus
clk_i	in	1	WISHBONE Master	Clock
rst_i	in	1	WISHBONE Master	Active Hign Sync. Reset
cyc_i	in	1	WISHBONE Master	To indicate valid bus cycle
stb_i	in	1	WISHBONE Master	To indicate valid data transfer cycle
sel_i	in	4	WISHBONE Decoder	Selection
we_i	in	1	WISHBONE Master	Read/Write enable
ack_o	out	1	WISHBONE Slave	To indicate current transfer status
hclk	in	1	AHB Master	Unconnected
hresetn	in	1	AHB Master	Unconnected

Πίνακας 4.4: Σήματα γέφυρας επικοινωνίας.

Παρατηρώντας τα παραπάνω σήματα που παρέχει αυτή η γέφυρα βλέπουμε να λείπουν τα σήματα *hbusreq* και *hgrantx* που χρησιμοποιεί η δικιά μας υλοποίηση του δίαυλου AMBA AHB. Αυτά τα σήματα χρησιμοποιούνται όταν πάνω στον δίαυλο επικοινωνίας τοποθετούνται παραπάνω από μία μονάδες που θα πρέπει να ανταγωνίζεται η μία την άλλη για να αποκτήσει πρόσβασή πάνω στο δίαυλο. Ουσιαστικά το *hbusreq* είναι ένα σήμα που δέχεται ο δίαυλος AHB από κάποια μονάδα master όταν αυτή η μονάδα θέλει να αποκτήσει πρόσβαση πάνω στον δίαυλο. Το αντίστοιχο σήμα που στέλνει ο επεξεργαστής OR1200 όταν επιθυμεί να αποκτήσει πρόσβασή πάνω στον δίαυλο είναι το *dwb_CYC_O* για μεταφορά δεδομένων και το *iwb_CYC_O* για μεταφορά εντολών. Επειδή όπως προαναφέρθηκε και οι δυο γέφυρες επικοινωνίας θα έχουν την ίδια υλοποίηση τόσο για αυτήν που θα συνδεθεί με την διεπαφή δεδομένων όσο και αυτή με την διεπαφή εντολών τα σήματα θα αναγράφονται με * μπροστά από την ονομασία τους αντί για *(*i) ή *(*d) αντίστοιχα. Δηλαδή στην προκειμένη περίπτωση **wb_CYC_O*. Άρα για να υλοποιεί η γέφυρα επικοινωνίας τους σκοπούς αυτών των δύο σημάτων πρέπει μέσα στην γέφυρα να ενώνονται αυτά και όταν ο επεξεργαστής θέλει αν αποκτήσει πρόσβαση στον δίαυλο (ενεργοποιώντας το σήμα **wb_CYC_O*) τότε αυτό το σήμα θα οδηγείται στην γέφυρα και εκείνη με την σειρά της θα θα ενεργοποιεί το σήμα *hbusreq* που θα οδη-

γείται πάνω στον δίαυλο και θα μεταφέρει ουσιαστικά το αίτημα του επεξεργαστή.

Το άλλο σήμα που και αυτό δεν υλοποιείται είναι το *hgrantx*. Ουσιαστικά αύτο το σήμα είναι το αποτέλεσμα του σήματος *hbusreq*. Δηλαδή όπως αναφέρθηκε και παραπάνω όταν το *hbusreq* ενεργοποιηθεί τότε ο δίαυλος με την βοήθεια του διαιτητή (arbiter) πρέπει να ενεργοποιήσει το σήμα *hgrantx* στην αντίστοιχη μονάδα που το αιτήθηκε ώστε να να της επιτρέψει να αναλάβει τον έλεγχο του διαύλου. Από την στιγμή που αναλάβει τον έλεγχο τότε όταν και το *hready* ενεργοποιηθεί θα πραγματοποιήσει την μεταφορά.

Η παραπάνω λογική ενσωματώνεται στην γέφυρα μας προσθέτοντας το σήμα *hgrant* σαν είσοδο στην γέφυρα μας και συνδέοντας το με το *hgrantx* του διαύλου. Εσωτερικά στην γέφυρα περνάμε τα σήματα *hgrant* και *hready* από μία λογική πύλη AND της οποίας την έξοδο την ανατροφοδοτούμε στο σήμα εξόδου της γέφυρας *ack_o*. Αυτό το σήμα (*ack_o*) επειδή συνδέεται με το σήμα **wb_ACK_I* του επεξεργαστή έχει σαν αποτέλεσμα όταν ενεργοποιείται να ξεκινάει η μεταφορά δεδομένων, που ουσιαστικά είναι η επιθυμητή κατάσταση. Ταυτόχρονα για ασφάλεια βάζουμε όλες τις συναρτήσεις που υλοποιούνται στην γέφυρα μέσα σε ένα if (hgrant and hready) do then fuctions ώστε να ξεκινάει η λειτουργία της γέφυρας όταν τόσο το *hgrant* όσο και το *hready* είναι ενεργοποιημένα. Με αυτό τον τρόπο αποφεύγουμε την άσκοπη λειτουργία της γέφυρας.

Τέλος τα σήματα **wb_ERR_I* και **wb_RTY_I* του επεξεργαστή τα οδηγούμε στο 0 μίας και η υλοποίηση τους δεν ήταν μέρος της διπλωματικής. Ο λόγος που τα οδηγούμε στο 0 είναι ότι με αυτό τον τρόπο εξασφαλίζουμε ότι ποτέ δεν θα φτάσει αίτημα στον επεξεργαστή για επανάληψη μεταφοράς δεδομένων ή σε περίπτωση λάθους.

4.4.2 Υλοποίηση σε Verilog

Ουσιαστικά σε αύτη την ενότητα θα παρουσιάσουμε υλοποίηση σε κώδικά Verilog των λειτουργιών που περιγράφηκαν παραπάνω. Παρακάτω βρίσκετε ο κώδικας που δημιουργήθηκε σχολιασμένος.

```
1
2  `timescale 1 ns / 1 ns
3  //DEFINES
4  //TOP MODULE
5
6  module AHBMAS_WBSLV_TOP (
7
8    hclk, hresetn,
9  // AHB Master Interface (Connect to AHB Slave)
10   haddr, htrans, hwrite, hsize,
11   hburst, hwdata, hrdata, hready, hresp, hbusreq, hgrant,
12
13 // WISHBONE Slave Interface (Connect to WB Master)
14   data_o, data_i, addr_i, clk_i, rst_i,
15   cyc_i, stb_i, sel_i, we_i, ack_o
16 );
17
18
19 //PARAMETER
20       parameter AWIDTH = 32,DWIDTH = 32; //Address Width, Data Width
21
22 //INPUTS AND OUTPUTS
```

```
23 // ------------------------------------------
24 //Top level ports for AHB
25 input hresetn;        //AHB Clk
26 input hclk;           //AHB Active Low Reset
27
28 // AHB Master Interface (Connect to AHB Slave)
29 input [DWIDTH-1:0]hrdata;          //Read data bus
30
31 //Transfer Response from AHB Slave
32 input [1:0]hresp;
33 input hready;
34 input hgrant;
35
36 //Address and Control Signals
37 output [AWIDTH-1:0]haddr;          //Address
38 output hwrite;                     //Write/Read Control
39 output [2:0]hsize;                 //Size of Data Control
40 output [2:0]hburst;                //Burst Control
41 output [31:0]hwdata;               //Write data bus
42 output [1:0]htrans;                //Transfer type
43 output hbusreq;
44
45
46 // ------------------------------------------
47 // WISHBONE Slave Interface (Connect to WB Master)
48 output     [DWIDTH-1:0]          data_o;    //Wishbobe Data Ouput
49 output                           ack_o;       //Wishbone Acknowledge
50
51 input      [DWIDTH-1:0]          data_i;    //Wishbone Data Input
52 input      [AWIDTH-1:0]          addr_i;    //Wishbone Address Input
53 input                            cyc_i;    //Wishbone Cycle Input
54 input                            stb_i;    //Wishbone Strobe Input
55 input      [3:0]                 sel_i;    //Wishbone Selection Input
56 input                            we_i;       //Wishbone Write/Read Control
57 input                            clk_i;     //Wishbone Clk Input
58 input                            rst_i;     //Wishbone Active High
59                          //Reset Input
60
61 // datatype declaration
62 reg [AWIDTH-1:0]haddr;
63 wire hwrite;
64 wire hbusreq;
65 reg [2:0]hsize;
66 reg [2:0]hburst;
67 reg [31:0]hwdata;
68 reg [1:0]htrans;
69 reg  [DWIDTH-1:0]data_o;
70 reg  ack_o;
71
72 //SIGNAL DECLARATIONS
73      reg flag;
74      reg hready_temp;
75     reg hgrant_temp;
76
77 //*****************************************************************
78 // WISHBONE logic Write and Read Operation
79 //*****************************************************************
80
81 //ASSIGN STATEMENTS
82 assign #2 hwrite = we_i;
83 assign #2 hbusreq = cyc_i; //implementetion of hbusreq
84
```

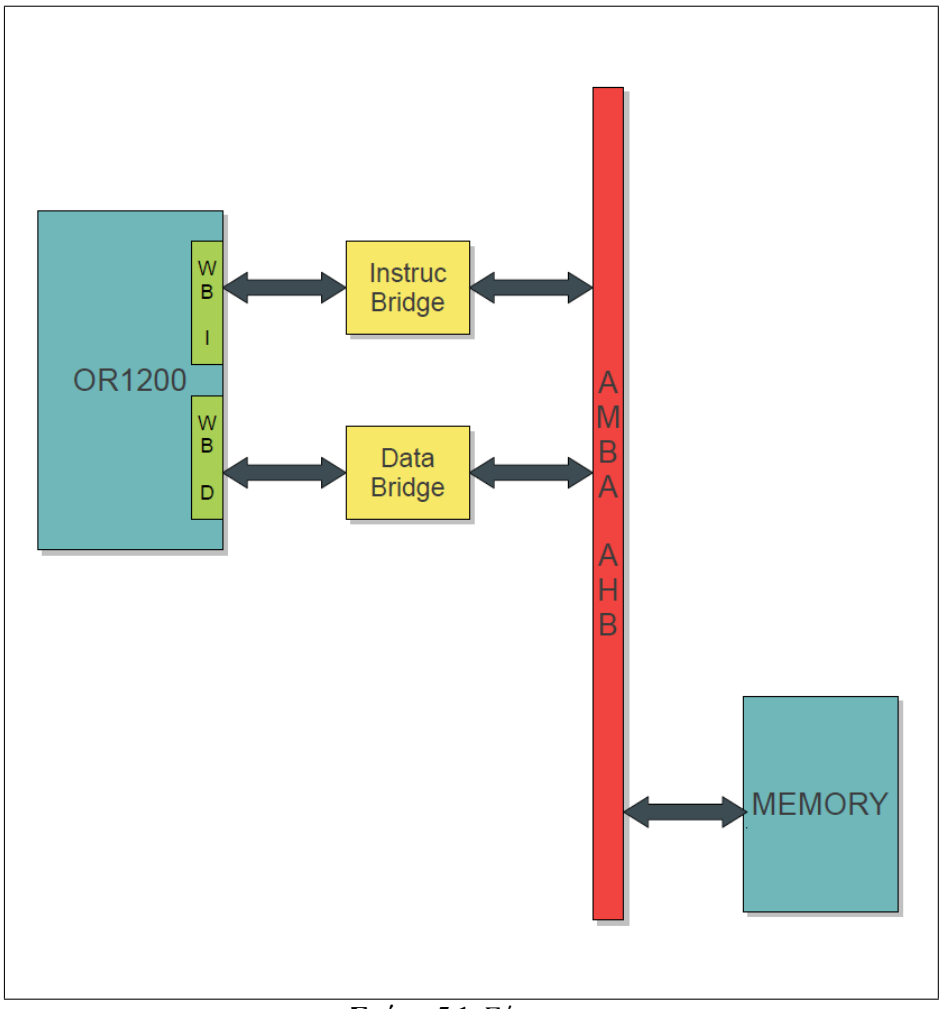
```
85
86  //Sysncronous Reset
87  always @ (posedge clk_i)
88       begin
89       //implementetion using hgrant
90       if (hgrant) begin
91           hgrant_temp<= hgrant;
92          end
93       if (rst_i) begin
94                 hsize = 3'b010;           //Size of Data Control
95                 hburst = 3'b000;          //Burst Control
96                 hgrant_temp <= 'b0;
97                 flag <= 'b1;
98                     end
99
100 //Write Operation : Wait for a valid Cycle,
101 //Strobe and Active High Write enable signal
102    else if ( hgrant_temp) begin
103                if (cyc_i & stb_i) begin
104                     if (we_i) begin //Write Cycle: No Need To Check
105                                     //for hready signal for data to be send out
106                          hwdata <= data_i;
107                     end
108 //Read Operation : Wait for a valid Cycle, Strobe and
109 // Active Low Write enable signal
110                  else begin          //    Read Cycle
111                     if (hready) begin
112                          if(flag) begin
113                               flag <=  'b0;
114                               end
115                          else begin
116                               flag <=  'b1;
117                               end
118                     end
119
120                end
121           end
122     end
123           else begin
124           ack_o<='b0;
125           //hwdata <= data_i;//when stb goes active low
126           //send asyncronously the data
127           end
128 end
129
130 always @ (we_i or stb_i or addr_i or flag or hready or hrdata or hgrant) begin
131       if(we_i) begin
132            if (hready) begin
133                 haddr <= addr_i;
134            end
135       end
136       else begin
137            if (flag) begin
138                 haddr <= addr_i;       //During Flag set Accept Address
139
140                 end
141            else begin
142                 data_o <= #2 hrdata;        //During Flag reset Accept Data
143                 end
144       end
145 end
146
```

```
147 //Logic for Acknowledge from Wishbone Slave
148 always @(we_i or addr_i or hrdata or hready or flag )  begin
149       if (rst_i) begin
150             ack_o<='b0;
151             end
152       else if (we_i) begin
153             ack_o <= hgrant_temp & hready;
154           // hgrant_temp <= 'b0;
155           end
156       else
157             ack_o<=!flag & hready & hgrant_temp;
158           #2 hgrant_temp <= 'b0;
159       end
160
161 //Logic for Transfer Type
162 always @(cyc_i or stb_i or clk_i) begin
163       if (rst_i) begin
164             htrans<=2'b00;
165             end
166       else if (cyc_i) begin
167             if (stb_i) begin
168                   htrans <= 2'b10;     //Transfer type Non Sequential
169             end
170             else begin
171                   htrans <= 2'b01;     //Transfer type Busy
172             end
173       end
174       else begin
175       htrans<=2'b00; //Transfer type Idle
176       end
177 end
178
179
180 endmodule
```

5 Υλοποίηση Συστήματος

5.1 Συνδεσμολογία συστήματος

Σκοπός αυτής της διπλωματικής εργασίας είναι να δημιουργήσουμε ένα ολοκληρωμένο σύστημά για την επεξεργασία εικόνας. Αυτό βέβαια προϋποθέτει την ύπαρξη και την σχεδίαση άλλων υπομονάδων από ένα σύνολο ανθρώπων. Για αυτό τον λόγο επειδή κάποια κομμάτια δεν είναι έτοιμα ακόμα σχεδιάστηκε μια πιο απλή μορφή συστήματος που αποδεικνύει την σωστή λειτουργία όσων αναφέρθηκαν στις προηγούμενες ενότητες. Το σύστημα που δημιουργήθηκε φαίνεται παρακάτω.



Σχήμα 5.1: Σύστημα.

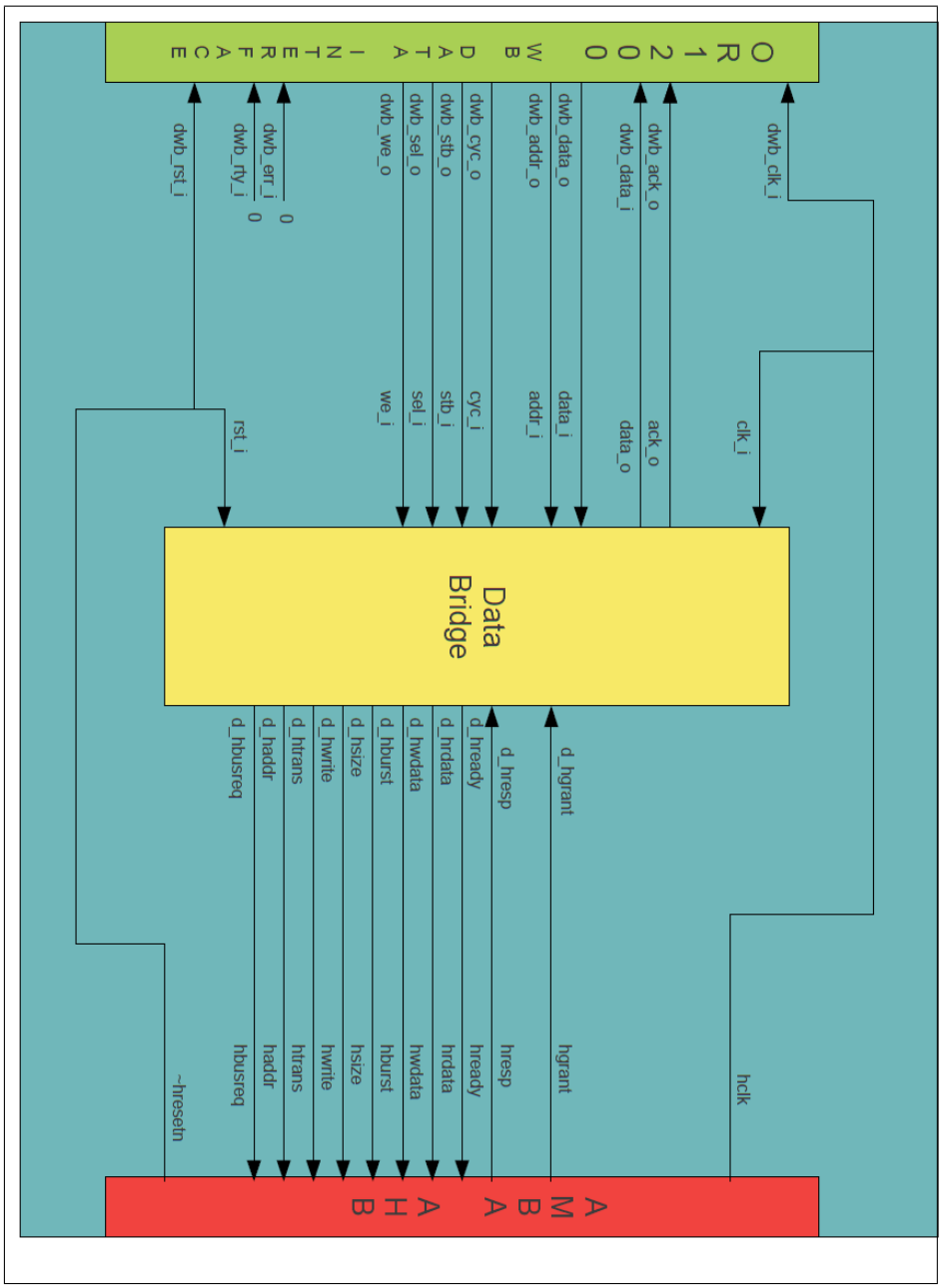
Όπως φαίνεται στο παραπάνω σχήμα το σύστημα μας αποτελείται:

- Το επεξεργαστή OR1200.
- Τις δύο γέφυρες επικοινωνίας.
- Τον δίαυλο επικοινωνίας AMBA AHB.
- Μια τυπική μνήμη.

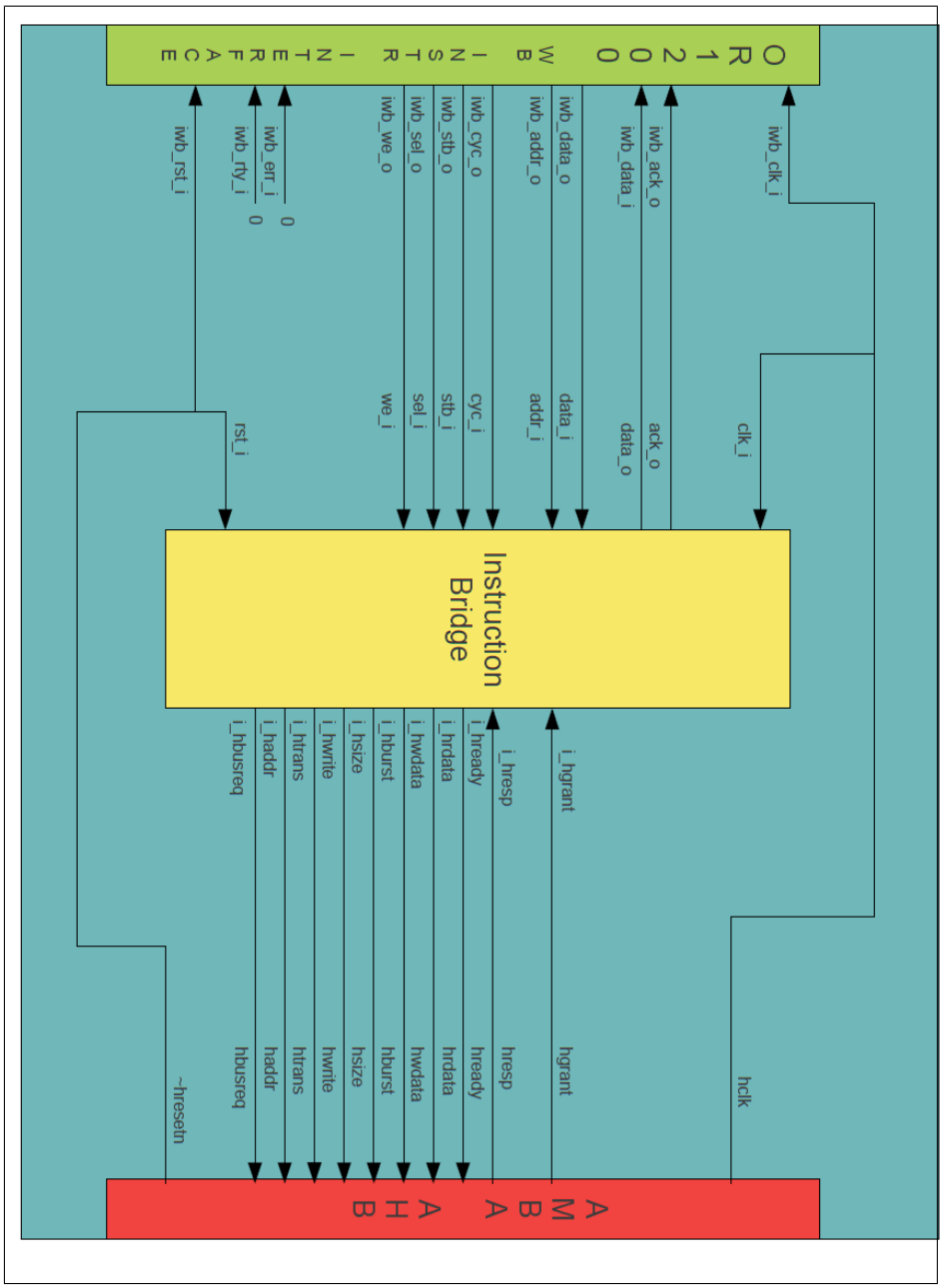
Ουσιαστικά με αυτό το σύστημα μπορούμε να ελένξουμε τόσο την σωστή λειτουργία του επεξεργαστή όσο και την λειτουργία των γεφυρών επικοινωνίας.

Στόχος μας είναι να πραγματοποιήσουμε απλές μεταφορές δεδομένων από και προς την μνήμη. Με αυτό τον τρόπο εξασφαλίζουμε τόσο την αρμονική λειτουργία του επεξεργαστή με τις γέφυρες όσο και την ορθή επικοινωνία των γεφυρών με δίαυλο AMBA AHB. Να σημειωθεί σε αυτό το σημείο ότι δημιουργία του διαύλου AMBA AHB και της μνήμης έγινε από Κώστα Αδαό που είναι υπεύθυνος του εργαστηρίου Μικροηλεκτρονικής.

Στα παρακάτω σχέδια παρουσιάζεται η συνδεσμολογία που δημιουργήθηκε μεταξύ του επεξεργαστή OR1200, των γεφυρών επικοινωνίας και του διαύλου AMBA AHB.



Σχήμα 5.2: Συνδεσμολογία Γεφυρας δεδομένων



Σχήμα 5.3: Συνδεσμολογία Γεφυρας Εντολών

5.2 Προσομοίωση Συστήματος

Για να γίνει η προσομοίωση του παραπάνω συστήματος αφού δημιουργήθηκε από το κ.Αδαό ο δίαυλος επικοινωνίας και η μνήμη στο αρχειο *ahb_test_final* ακολουθήθηκαν βήματα που περιγράφονται παρακάτω.

1. Προσθήκη κατάλληλων αρχείων Verilog

Αρχικά προστέθηκε μέσα στον φάκελο *ahb_test_final* ένας καινούργιος φάκελος με την ονομασία *or1200_top_ahb* μέσα στον οποίο είναι τοποθετημένο το top αρχείο που περιγράφει την λειτουργία της γέφυρας καθώς και το initialization του επεξεργαστή OR1200 που ουσιαστικά συνδέει όλα τα σήματα που του OR1200 με τέτοιο τρόπο ώστε να καλύπτει τις ανάγκες του συστήματος. Το αρχείο αυτό έχει την ονομασία *or1200_top_ahb.v* και παρακάτω βρίσκεται ο σχολιασμένος κώδικας που σχεδιάστηκε.

```
1  module or1200_top_ahb (
2
3      hclk                ,
4      hresetn             ,
5
6      i_haddr             ,
7      i_htrans            ,
8      i_hwrite            ,
9      i_hsize             ,
10     i_hburst            ,
11     i_hwdata            ,
12     i_hrdata            ,
13     i_hready            ,
14     i_hresp             ,
15     i_hbusreq            ,
16     i_hgrant             ,
17
18     d_haddr             ,
19     d_htrans            ,
20     d_hwrite            ,
21     d_hsize             ,
22     d_hburst            ,
23     d_hwdata            ,
24     d_hrdata            ,
25     d_hready            ,
26     d_hresp             ,
27     d_hbusreq            ,
28     d_hgrant             ,
29
30     or1200_pic_ints
31
32
33 );
34
35 input           hclk;
36 input           hresetn;
37
38
39 input [31:0]    i_hrdata;
40 input [1:0]     i_hresp;
```

```
41  input           i_hready;
42  input           i_hgrant;//grant
43  output [31:0]   i_haddr;   //Address
44  output          i_hwrite;  //Write/Read Control
45  output [2:0]    i_hsize;   //Size of Data Control
46  output [2:0]    i_hburst;  //Burst Control
47  output [31:0]   i_hwdata;  //Write data bus
48  output [1:0]    i_htrans;  //Transfer type
49  output          i_hbusreq; //bus_req
50
51  input [31:0]    d_hrdata;
52  input [1:0]     d_hresp;
53  input           d_hready;
54  input           d_hgrant;  // grant
55  output [31:0]   d_haddr;   //Address
56  output          d_hwrite;  //Write/Read Control
57  output [2:0]    d_hsize;   //Size of Data Control
58  output [2:0]    d_hburst;  //Burst Control
59  output [31:0]   d_hwdata;  //Write data bus
60  output [1:0]    d_htrans;  //Transfer type
61  output          d_hbusreq; //bus_req
62
63  input [19:0]    or1200_pic_ints;
64
65
66  wire [31:0]      wbm_i_or12_adr_o;
67  wire [31:0]      wbm_i_or12_dat_o;
68  wire [3:0]       wbm_i_or12_sel_o;
69  wire                wbm_i_or12_we_o;
70  wire                wbm_i_or12_cyc_o;
71  wire                wbm_i_or12_stb_o;
72
73  wire [31:0]      wbm_i_or12_dat_i;
74  wire                wbm_i_or12_ack_i;
75
76  wire [31:0]      wbm_d_or12_adr_o;
77  wire [31:0]      wbm_d_or12_dat_o;
78  wire [3:0]       wbm_d_or12_sel_o;
79  wire                wbm_d_or12_we_o;
80  wire                wbm_d_or12_cyc_o;
81  wire                wbm_d_or12_stb_o;
82
83  wire [31:0]      wbm_d_or12_dat_i;
84  wire                wbm_d_or12_ack_i;
85
86
87
88  AHBMAS_WBSLV_TOP i_bridge (
89
90     .hclk      ( hclk                  ),
91     .hresetn   ( hresetn               ),
92
93  // AHB Master Interface (Connect to AHB Slave)
94     .haddr     ( i_haddr               ),
95     .htrans    ( i_htrans              ),
96     .hwrite    ( i_hwrite              ),
97     .hsize     ( i_hsize               ),
98     .hburst    ( i_hburst              ),
99     .hwdata    ( i_hwdata              ),
100    .hrdata    ( i_hrdata              ),
101    .hready    ( i_hready              ),
102    .hresp     ( i_hresp               ),
```

```
103    .hgrant     ( i_hgrant                ),
104    .hbusreq     ( i_hbusreq              ),
105
106  // WISHBONE Slave Interface (Connect to WB Master)
107    .data_o     ( wbm_i_or12_dat_i        ),
108    .data_i     ( wbm_i_or12_dat_o        ),
109    .addr_i     ( wbm_i_or12_adr_o        ),
110
111    .clk_i      ( hclk                    ),
112    .rst_i      ( ~hresetn                ),
113    .cyc_i      ( wbm_i_or12_cyc_o        ),
114    .stb_i      ( wbm_i_or12_stb_o        ),
115    .sel_i      ( wbm_i_or12_sel_o        ),
116    .we_i       ( wbm_i_or12_we_o         ),
117    .ack_o      ( wbm_i_or12_ack_i        )
118  );
119
120
121  AHBMAS_WBSLV_TOP d_bridge (
122
123    .hclk       ( hclk                    ),
124    .hresetn    ( hresetn                 ),
125
126  // AHB Master Interface (Connect to AHB Slave)
127    .haddr      ( d_haddr                 ),
128    .htrans     ( d_htrans                ),
129    .hwrite     ( d_hwrite                ),
130    .hsize      ( d_hsize                 ),
131    .hburst     ( d_hburst                ),
132    .hwdata     ( d_hwdata                ),
133    .hrdata     ( d_hrdata                ),
134    .hready     ( d_hready                ),
135    .hresp      ( d_hresp                 ),
136    .hgrant      (d_hgrant                 ),
137    .hbusreq     ( d_hbusreq               ),
138
139  // WISHBONE Slave Interface (Connect to WB Master)
140    .data_o     ( wbm_d_or12_dat_i        ),
141    .data_i     ( wbm_d_or12_dat_o        ),
142    .addr_i     ( wbm_d_or12_adr_o        ),
143
144    .clk_i      ( hclk                    ),
145    .rst_i      ( ~hresetn                ),
146    .cyc_i      ( wbm_d_or12_cyc_o        ),
147    .stb_i      ( wbm_d_or12_stb_o        ),
148    .sel_i      ( wbm_d_or12_sel_o        ),
149    .we_i       ( wbm_d_or12_we_o         ),
150    .ack_o      ( wbm_d_or12_ack_i        )
151  );
152
153
154
155     //
156     // Instantiation
157     //
158     or1200_top or1200_top0
159         (
160       // Instruction bus, clocks, reset
161       .iwb_clk_i                (hclk),
162       .iwb_rst_i                (~hresetn),
163       .iwb_ack_i                (wbm_i_or12_ack_i),
164       .iwb_err_i                (1'b0),
```

```
165       .iwb_rty_i                (1'b0),
166       .iwb_dat_i                (wbm_i_or12_dat_i),
167
168       .iwb_cyc_o                (wbm_i_or12_cyc_o),
169       .iwb_adr_o                (wbm_i_or12_adr_o),
170       .iwb_stb_o                (wbm_i_or12_stb_o),
171       .iwb_we_o             (wbm_i_or12_we_o),
172       .iwb_sel_o                (wbm_i_or12_sel_o),
173       .iwb_dat_o                (wbm_i_or12_dat_o),
174       .iwb_cti_o                (),
175       .iwb_bte_o                (),
176
177       // Data bus, clocks, reset
178       .dwb_clk_i                (hclk),
179       .dwb_rst_i                (~hresetn),
180       .dwb_ack_i                (wbm_d_or12_ack_i),
181       .dwb_err_i                (1'b0),
182       .dwb_rty_i                (1'b0),
183       .dwb_dat_i                (wbm_d_or12_dat_i),
184
185       .dwb_cyc_o                (wbm_d_or12_cyc_o),
186       .dwb_adr_o                (wbm_d_or12_adr_o),
187       .dwb_stb_o                (wbm_d_or12_stb_o),
188       .dwb_we_o             (wbm_d_or12_we_o),
189       .dwb_sel_o                (wbm_d_or12_sel_o),
190       .dwb_dat_o                (wbm_d_or12_dat_o),
191       .dwb_cti_o                (),
192       .dwb_bte_o                (),
193
194       // Debug interface ports
195       .dbg_stall_i          (1'b0),
196       .dbg_ewt_i                (1'b0),
197       .dbg_lss_o                (),
198       .dbg_is_o             (),
199       .dbg_wp_o             (),
200       .dbg_bp_o             (),
201
202       .dbg_adr_i                (0),
203       .dbg_we_i             (1'b0 ),
204       .dbg_stb_i                (1'b0),
205       .dbg_dat_i                (0),
206       .dbg_dat_o                (),
207       .dbg_ack_o                (),
208
209
210       .pm_clksd_o               (),
211       .pm_dc_gate_o         (),
212       .pm_ic_gate_o         (),
213       .pm_dmmu_gate_o           (),
214       .pm_immu_gate_o           (),
215       .pm_tt_gate_o         (),
216       .pm_cpu_gate_o        (),
217       .pm_wakeup_o          (),
218       .pm_lvolt_o               (),
219
220       // Core clocks, resets
221       .clk_i                    ( hclk              ),
222       .rst_i                    (~hresetn           ),
223
224       .clmode_i             ( 2'b00             ),
225       // Interrupts
226       .pic_ints_i               ( or1200_pic_ints   ),
```

```
227         .sig_tick           (                   ), /// ??????????????
228         /*
229          .mbist_so_o            (),
230          .mbist_si_i            (0),
231          .mbist_ctrl_i          (0),
232          */
233
234         .pm_cpustall_i      ( 1'b0              )
235
236         );
237
238
239
240 endmodule
```

2. Τροποποίηση testbench

Ακολουθώντας την μεθοδολογία που περιγράφηκε στο τρίτο κεφαλαίο κάναμε μερικές εξομοιώσεις και εξάγαμε το παρακάτω συμπέρασματα.

- Όταν ξεκίνα την λειτουργία του ο επεξεργαστής παίρνει δεδομένα από την μνήμη χρησιμοποιώντας την διεπαφη εντολών.
- Οι πρώτες 3 διευθύνσεις που προσπελαύνει είναι οι 00000100, 00000104 και 00000108. Απο αυτές τις διευθύνσεις παίρνει τα δεδομένα που έχουν μεσά.

Εκμεταλλευόμενοι τα παραπάνω συμπεράσματα και λαμβάνοντας υπόψιν τον σκοπό αυτή της διπλωματικής εργασίας δημιουργήσαμε ένα testbech που αρχικά θα βάζει στις παραπάνω διευθύνσεις της μνήμης κάποια δεδομένα που εμείς επιλέξαμε και μετά θα ενεργοποιεί τον επεξεργαστή και θα περιμένουμε λογικά να τραβήξει αυτά τα δεδομένα από την μνήμη μέσω της γέφυρας εντολών ο επεξεργαστής OR1200.
Αυτό έγινε τροποποιώντας το αρχείο *ahb_master.v* που βρίσκεται μέσα στο φάκελο *ahb_test_final/ahb_models/* όπως φαίνεται παρακάτω.

```

task ahb_master_commands;
    begin

        #1_000;

        repeat(200) @(posedge hclk);

        set_size(BUS_32); $display;
        raddr = 32'h0000_0100; ahb_write(raddr, 32'h1234_5678, 1);
        raddr = 32'h0000_0104; ahb_write(raddr, 32'h0000_0001, 1);
        raddr = 32'h0000_0108; ahb_write(raddr, 32'h9999_9999, 1);


    end

endtask
```

3. Τροποποίηση Makefile

Μετά την τοποθέτηση των κατάλληλων αρχείων Verilog προστέθηκαν οι παρακάτω γραμμές κώδικα μέσα στο αρχείο *Makefile* που βρίσκεται μέσα στο φάκελο *ahb_test_final* μέσα στο section sim:.

```
      or1200_top_ahb/or1200_top_ahb.v \
      or1200_top_ahb/ahbmas_wbslv_top.v \
      -y rtl/verilog/or1200      \
      +incdir+../orpsocv2/rtl/verilog/include \
      +libext+.v \
      -y ../orpsocv2/rtl/verilog/or1200      \
      +incdir+/mnt/datafs/users/kalargaris/orpsocv2/sim/run/../../
bench/verilog/include \
```

Το παραπάνω Makefile ελέγχει την ροή προσομοίωσης όποτε αυτό που κάναμε ήταν να συμπεριλάβουμε τα αρχεία που περιγράφουν τον επεξεργαστή καθώς και τα τεστ του επεξεργαστή όπως φαίνεται παραπάνω.

4. Στάδιο Προσομοίωσης

Για να γίνει η προσομοίωση εκτελέστηκαν οι παρακάτω εντολές με την σειρά που αναγράφονται μέσα στον φάκελο *or1200_top_ahb* με την βοήθεια του terminal.

```
make clean
#clean directories from results from previus simulations
```

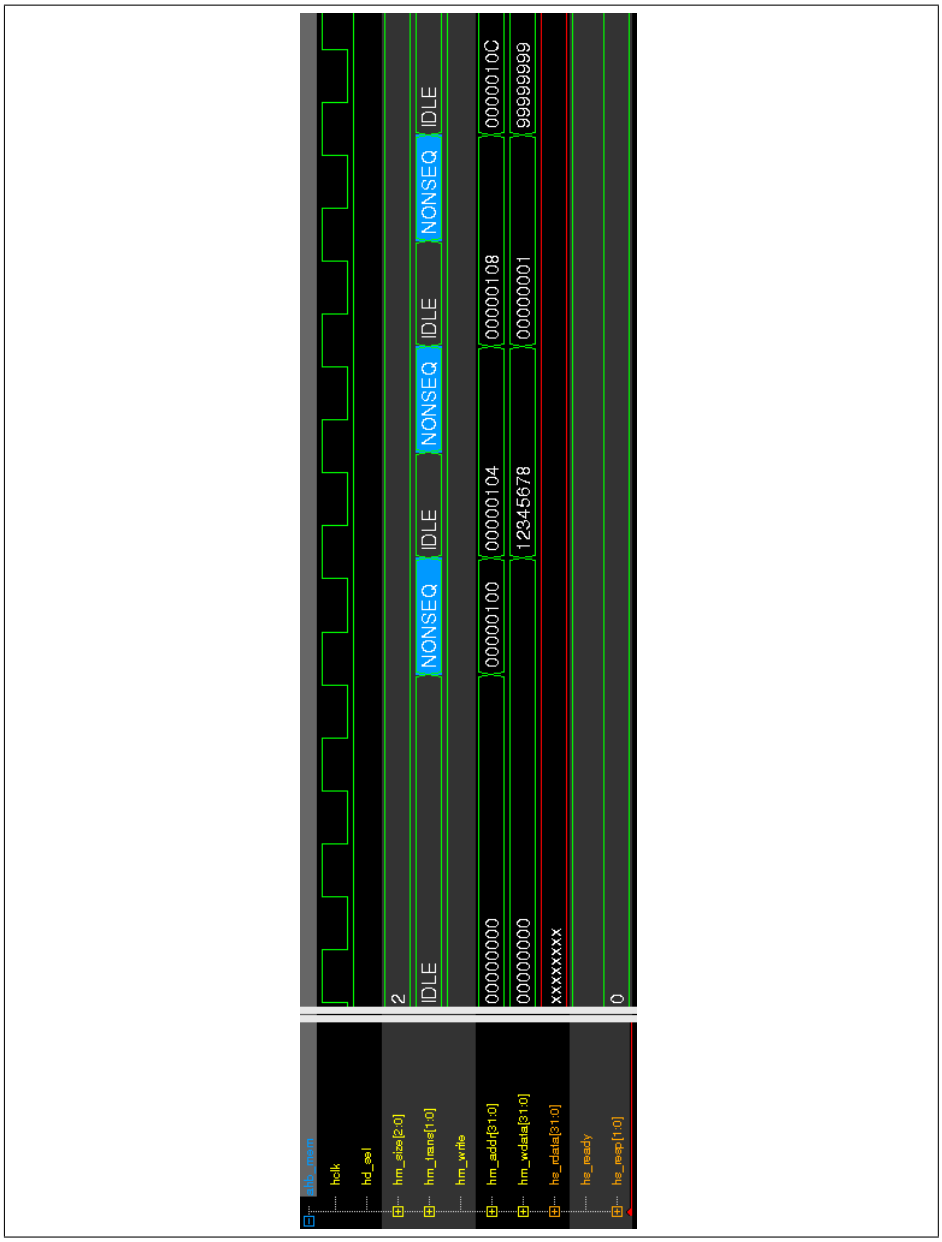
```
make sim
#to simulate ouy system
```

5. Προβολή Κυματομορφών - Σχολιασμός

Με την παρακάτω εντολή μπορούμε να δούμε τις κυματομορφές της προσομοίωσης που κάναμε παραπάνω και να διαπιστώσουμε αν το σύστημα μας λειτουργεί αρμονικά.

```
make waves
# to open waveforms with simvision
```

Εκτελώντας την παραπάνω εντολή παίρνουμε τις παρακάτω κυματομορφές. Να σημειωθεί ότι επιλέχθηκαν μόνο εκείνα τα σήματα που χρειάζονταν να αναπαρασταθούν ώστε να δείξουμε την αρμονική λειτουργία του συστήματος. Δηλαδή οτί ξεκινά ο επεξεργαστής την λειτουργία του και αλληλεπιδρά σωστά με την μνήμη όπως ορίσαμε στο τρίτο βήμα παραπάνω.

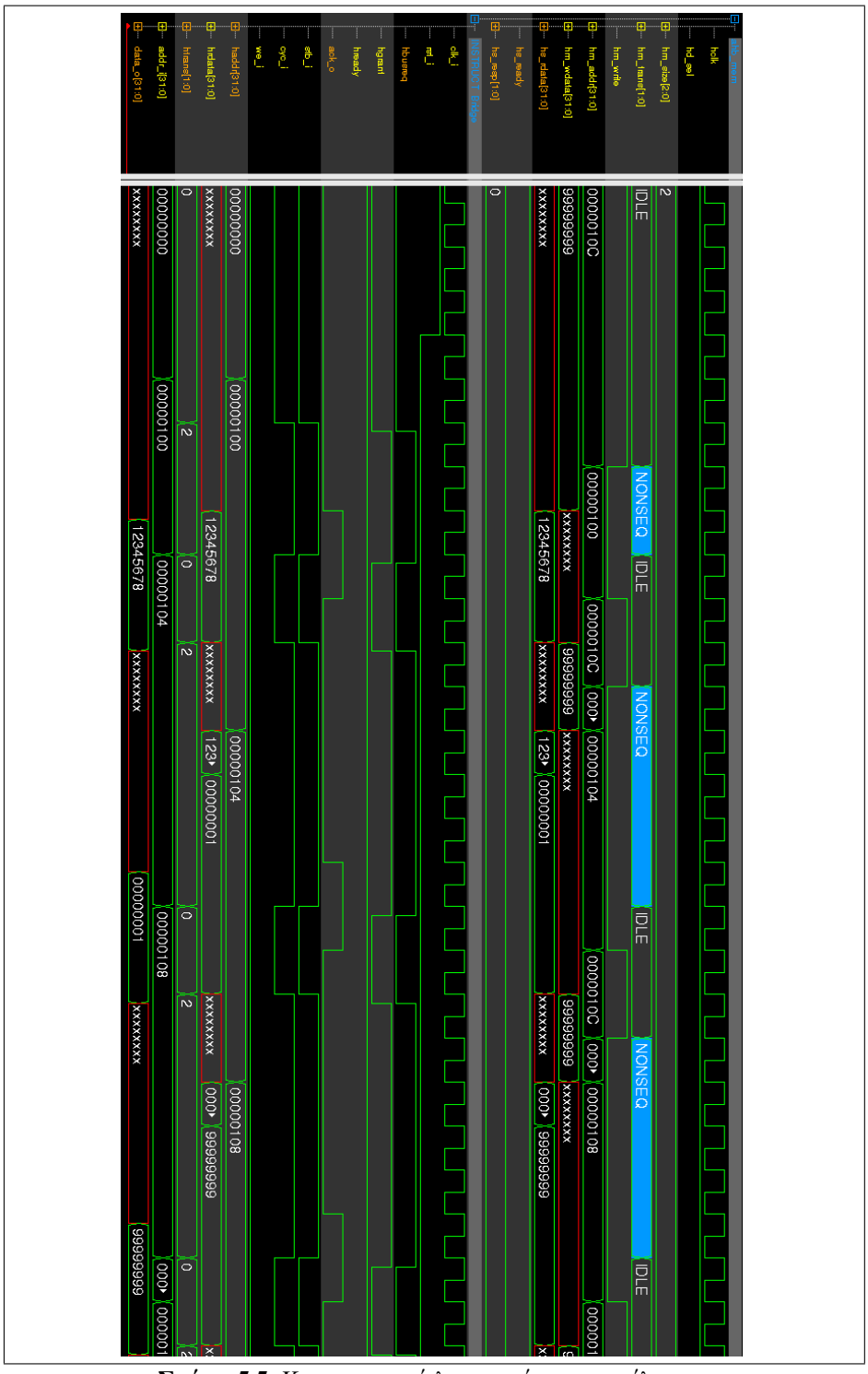


Σχήμα 5.4: Κυματομορφή λειτουργίας μνήμης.

Στην παραπάνω κυματομορφή εμφανίζεται η λειτουργία της μνήμης. Βρισκόμαστε στο αρχικό στάδιο που γράφουμε στην μνήμη στις διευθύνσεις 00000100, 00000104 και 00000108 τα δεδομένα 12345678, 00000001 και 99999999 αντίστοιχα.

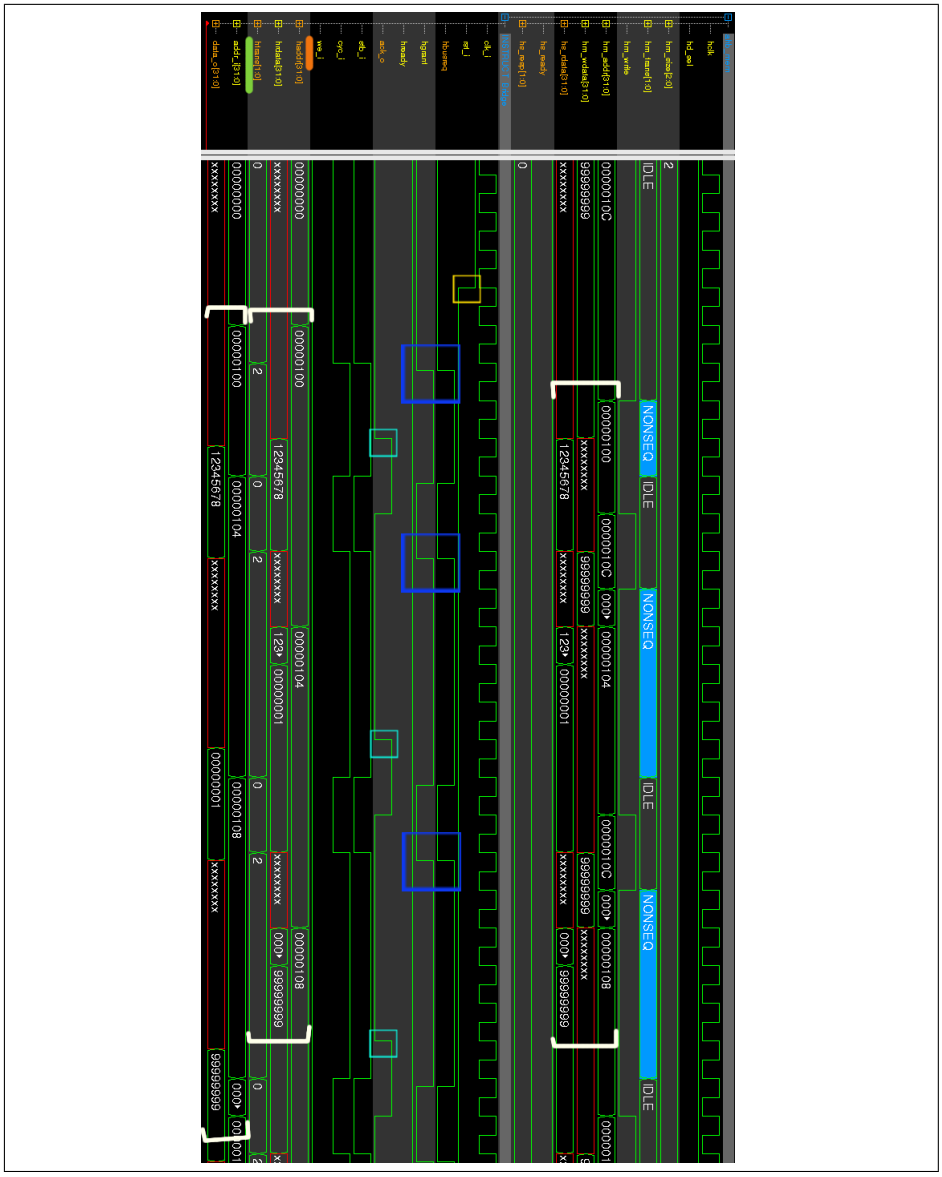
Παρατηρούμε ότι γίνετε εγγραφή αφού χρησιμοποιείται το σήμα *hwdata* που υποδηλώνει την εγγραφή και το σήμα *hrdata* που υποδηλώνει την ανάγνωση.

Η παρακάτω κυματομορφή δείχνει την λειτουργία του επεξεργαστή όταν θέλει να προσπελάσει τα δεδομένα στην μνήμη.



Σχήμα 5.5: Κυματομορφή λειτουργίας προσπέλασης.

Επειδή είναι δύσκολο να σχολιαστεί αυτή η κυματομορφή παρακάτω παρουσιάζεται η επεξεργασμένη κυματομορφή που θα μας βοηθήσει να σχολιάσουμε την λειτουργία.



Σχήμα 5.6: Επεξεργασμένη κυματομορφή.

Παρατηρώντας την παραπάνω κυματομορφή διακρίνουμε τα εξής χαρακτηριστικά που δηλώνουν την σωστή λειτουργία τους συστήματος μας.

1. Κίτρινο τετράγωνο: Εδώ παρατηρούμε το σήμα *rst_i* που όταν απενεργοποιείται ουσιαστικά ενεργοποιεί τον επεξεργαστή και την γέφυρα.

2. Μπλε τετράγωνο: Εδώ παρατηρούμε τα σήματα *hbusreq*,*hready* και *hgrant*. Ουσιαστικά αφου ξεκίνησε ο επεξεργαστής απαιτεί πρόσβαση στο δίαυλο ενεργοποιώντας το σήμα *cyc_i*. Αυτό το σήμα μεταφέρεται διαμέσου της γέφυρας εντολών στον δίαυλο με την ενεργοποίηση του σήματος *hbusreq*. Παρατηρούμε ότι ο δίαυλος του επιστρέφει αμέσως την έγκριση πρόσβασης στο δίαυλο με την ενεργοποίηση του σήματος *hgrant* και συνδυασμός με το σήμα *hready* παρατηρούμε την ενεργοποίηση του σήματος *ack_o*.

3. Γαλάζιο τετράγωνο: Εδώ παρατηρούμε το σήμα *ack_o* που ενεργοποιείται όπως αναφέραμε παραπάνω και ουσιαστικά λέει στον επεξεργαστή να ξεκινήσει την μεταφορά δεδομένων.

4. Πορτοκαλί υπογράμμιση: Εδώ παρατηρούμε το σήμα *we_i* που είναι απενεργοποιημένο. Αυτό δείχνει ότι θα εκτελεστούν αναγνώσεις από την μνήμη και όχι εγγραφές σε αυτήν.Πράγμα το οποίο περιμέναμε ουσιαστικά.

5. Άσπρες αγκύλες: Ουσιαστικά τα σήματα που περιέχουν αυτές οι αγκύλες παρουσιάζουν την λειτουργίας της μεταφοράς δεδομένων. Αρχικά στο σήμα *addr_i[31:0]* ο επεξεργαστής τοποθετεί την διεύθυνση της μνήμης από την οποία θέλει να πάρει τα δεδομένα. Μέτα αυτό πηγαίνει στην γέφυρα και η οποία βγάζει σαν έξοδο το σήμα *haddr[31:0]* που περιέχει την διεύθυνση μνήμης και την τοποθετεί πάνω στον δίαυλο επικοινωνίας. Ύστερα ο δίαυλος επικοινωνίας μεταφέρει αυτό το σήμα σαν είσοδο στην μνήμη στο σήμα *hm_addr[31:0]*. Μετά η μνήμη παίρνει τα δεδομένα που βρίσκονται σε αυτή την διεύθυνση και τα μεταφέρει στο σήμα *hs_rdata[31:0]* που χρησιμοποιείται μόνο για σκοπούς ανάγνωσης. Ύστερα αυτό το σήμα που περιέχει τα δεδομένα που θέλουμε μεταφέρεται διαμέσου του διαύλου στον γέφυρα εντολών η οποία με την σειρά της το μεταφέρει στον επεξεργαστή διάμεσου του σήματος *data_o[31:0]* οπότε και φτάνουν τα δεδομένα που είχε ζητήσει ο επεξεργαστής αρχικά σε αυτόν.

6. Πράσινη υπογράμμιση: Εδώ παρατηρούμε το σήμα *htrans[1:0]* που περιγράφει το είδος της μεταφοράς. Όταν έχει την τιμή 2 τότε πραγματοποιείται μια απλή μεταφορά δεδομένου (Nonseq transfer) και όταν είναι 0 δείχνει ότι δεν πραγματοποιείται εκείνη την στιγμή κάποια μεταφορά.

Συνοψίζοντας τα παραπάνω συμπεράσματα από τις κυματομορφές μπορούμε να διαπιστώσουμε ότι ο επεξεργαστής OR1200 λειτουργεί αρμονικά με την μνήμη που ήταν και ο αρχικός μας στόχος. Υπάρχουν πολλά περιθώρια βελτίωσης των παραδειγμάτων που αναφέρθηκαν σε αυτήν την διπλωματική άλλα με την χρήση πιο σύνθετων παραδειγμάτων δεν θα ήταν εύκολο να αναλυθεί η λειτουργία του συστήματος μας. Μέτα την παραπάνω υλοποίηση έχει σχεδιαστεί η ραχοκοκκαλιά

του συστήματος πάνω στο οποίο θα μπορέσουν μελλοντικά οι φοιτητές να τοποθετήσουν τις δικές τους μονάδες.

Αναφορές

[1] ARM Holdings plc, *"AMBA™ Specification (Rev 2.0)",* Released:May 13, 1999. www.arm.com.

[2] Damjan Lampret, *"OpenRISC 1200 IP Core Specification",* Rev. 0.11, Released:January, 2011.

[3] Deepak Kumar Tala, *"Verilog Tutorial",* Released:October 25, 2003.

[4] Helmut Kopka, Patrick W. Daly, *A Guide to LaTeXand Electronic Publishing,* Addison-Wesley Publishing Company, Fourth edition, Released:2004.

[5] Julius Baxter, OpenCores Organization, *"ORPSoC User Guide",* Released:2010. www.opencores.org.

[6] Mattsson, Christensson, *"Evaluation of synthesizable CPU cores",* Master's Thesis, Chalmers University of Technology, 2004.

[7] OpenCores Organization, *"OpenRISC 1000 Architecture Manual",* Released:April 5, 2006. www.opencores.org.

[8] OpenCores Organization, *"OpenCores HDL modeling guidelines",* Released:2009. www.opencores.org.

[9] Richard Herveille, OpenCores Organization, *"WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores", Revision: B.3,* Released:September 7, 2002. www.opencores.org.

[10] TooMuch Semiconductor Solutions, *"WISHBONE-AHB BRIDGE",* Rev. 1.0, Released:2007.

[11] Vinayak Pai, Swapnil S. Lotlikar, *"Modeling Lifetime Reliability of Processor Cores",* Department of ECE,Texas A&M University,College Station, Texas.