

Πτυχιακή εργασία
του Κωνσταντίνου Κρομμύδα

ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΑΠΟΚΩΔΙΚΟΠΟΙΗΤΗ VIDEO AVS ΣΕ ΕΠΑΝΑΔΙΑΤΑΣΣΟΜΕΝΗ ΛΟΓΙΚΗ



Επιβλέποντες καθηγητές: *Μπέλλας Νικόλαος, αναπληρωτής καθηγητής*
Αντωνόπουλος Δ. Χρήστος, επίκουρος καθηγητής

Τμήμα Μηχανικών Η/Υ, Τηλ/νιών και Δικτύων
Πανεπιστήμιο Θεσσαλίας, 2009-2010

Ευχαριστίες

Φτάνοντας στο τέλος των προπτυχιακών μου σπουδών, θα ήθελα να πω ένα μεγάλο ευχαριστώ σε όλους εκείνους που μου στάθηκαν με τον ένα ή τον άλλο τρόπο και που με βοήθησαν να φτάσω αισίως ως εδώ.

Πρώτα από όλα, θα ήθελα να ευχαριστήσω την οικογένειά μου για την αμέριστη στήριξη σε κάθε τομέα.

Στη συνέχεια, ένα μεγάλο ευχαριστώ στους επιβλέποντες καθηγητές μου κ.κ. Νίκο Μπέλλα και Χρήστο Αντωνόπουλο, οι οποίοι μου έδωσαν την ώθηση και την ευκαιρία να προχωρήσω ένα βήμα παραπέρα, τόσο με το ειδικό θέμα του AVS το οποίο κατέληξε σε δημοσίευση –πάντα με τη βοήθειά τους και τη συνεργασία του Γιώργου Τσουμπλέκα- όσο και με την πολύτιμη βοήθειά τους στην προσπάθειά μου να γίνω δεκτός σε κάποιο πανεπιστήμιο της Αμερικής για μεταπτυχιακές σπουδές. Δίπλα τους έμαθα πολλά πράγματα, τόσο συγκεκριμένα και σχετικά με τα μαθήματά τους, όσο και γενικότερα για τον τρόπο σκέψης και δουλειάς ως μηχανικός Η/Υ.

Ευχαριστώ, επίσης, τον καθηγητή μου Ιωάννη Κατσαβουνίδη, για όλα όσα έμαθα από αυτόν και μου χρειάστηκαν στην πορεία, για τον ιδιαίτερο τρόπο διδασκαλίας του, που εμπνέει τους φοιτητές και βεβαίως για τη δική του συνεισφορά και βοήθεια στο ζήτημα των μεταπτυχιακών.

Τέλος, θα ήθελα να ευχαριστήσω όλους τους καθηγητές, και τους φίλους με τους οποίους συνεργάστηκα όλα αυτά τα χρόνια. Ιδιαίτερη μνεία θα ήθελα να κάνω στον Γιώργο Τσουμπλέκα, το Θανάση Καραπατή και το Νίκο Μακρή, ενώ φυσικά δε θα μπορούσα να παραλείψω την αναφορά στον Κωνσταντή Νταλούκα, ο οποίος παρά τον εξαιρετικά μεγάλο προσωπικό φόρτο εργασίας, πάντα έβρισκε λίγο χρόνο για να με βοηθήσει, ως εμπειρότερος, με τυχόν απορίες ή προβλήματα.

Περιεχόμενα

| | |
|--------------------------------------------------------------------------------|-----------|
| Ευχαριστίες..... | 3 |
| 1. ΕΙΣΑΓΩΓΗ | 6 |
| 2. ΣΚΟΠΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ..... | 8 |
| 3. FPGAs | 9 |
| Εισαγωγικά..... | 9 |
| Xilinx ML507 | 10 |
| Power PC | 13 |
| Endianness..... | 15 |
| 4. ΘΕΩΡΙΑ VIDEO CODECS | 18 |
| AVS- Audio Video Standard | 18 |
| Βασικές έννοιες video | 19 |
| Progressive and interlaced video sequence | 19 |
| Picture formats..... | 19 |
| Picture types..... | 21 |
| Picture reordering | 21 |
| Macroblock..... | 22 |
| 8x8 Block..... | 22 |
| Αναλυτική παρουσίαση του Motion Compensation..... | 23 |
| Εισαγωγή | 23 |
| Interpolation process για τα luminance samples..... | 23 |
| Υπολογισμός Half Samples | 24 |
| Υπολογισμός Quarter Samples..... | 25 |
| Interpolation process για τα chrominance samples | 26 |
| 5. ΥΛΟΠΟΙΗΣΗ ΤΟΥ HARDWARE INFRASTRUCTURE..... | 27 |
| Εισαγωγή..... | 27 |
| Δημιουργία/προσθήκη του Motion Compensation accelerator | 28 |
| A) Create Peripheral stage..... | 28 |
| B. Modify stage..... | 30 |
| Γ) Import Stage | 34 |
| Ενσωμάτωση του Motion Compensation Peripheral στο υπάρχον hardware σύστημά.... | 35 |
| Δημιουργία Driver του motion compensation peripheral | 36 |
| Βασικές μακροεντολές εγγραφής/ανάγνωσης καταχωρητών..... | 36 |

| | |
|------------------------------------------------------------------------------------------------|-----------|
| 6. ΥΛΟΠΟΙΗΣΗ ΤΟΥ SOFTWARE INFRASTRUCTURE | 41 |
| Εισαγωγή | 41 |
| Απόφαση για χρησιμότητα και επιλογή λειτουργικού συστήματος | 41 |
| 7. ΜΕΤΡΗΣΗ ΑΠΟΔΟΣΗΣ/ΑΞΙΟΛΟΓΗΣΗ | 44 |
| 8. ΠΡΟΟΠΤΙΚΕΣ ΕΠΕΚΤΑΣΗΣ/ΕΞΕΛΙΞΗΣ | 48 |
| Hardware Accelerators/ Co-processors | 48 |
| Multiprocessor Designs | 49 |
| 9. ΕΠΙΛΟΓΟΣ/ΣΥΝΟΨΗ | 53 |
| ΒΙΒΛΙΟΓΡΑΦΙΑ | 54 |
| Παράρτημα Α | 57 |
| Γλωσσάρι κυριότερων εννοιών video coding/decoding | 57 |
| Παράρτημα Β | 59 |
| Αναλυτικές οδηγίες δημιουργίας hardware infrastructure | 59 |
| Υλοποίηση του βασικού συστήματος | 59 |
| Δημιουργία του dts αρχείου | 62 |
| Παράρτημα Γ | 65 |
| Προετοιμασία του περιβάλλοντος προγραμματισμού | 65 |
| Κατέβασμα και εγκατάσταση των Git Source Control Utilities | 65 |
| Κατεβάσματος των kernel source files | 65 |
| Α) Με το εργαλείο Git που εγκαταστήσαμε προηγουμένως | 66 |
| Β) Κατεβάζοντας απευθείας το snapshot του source tree. | 66 |
| Κατέβασμα και εγκατάσταση του DENX ELDK 4.2 Toolchain | 66 |
| Κάνοντας build το default kernel image | 67 |
| Κατεβάζοντας τη σχεδιάσή μας στο Target Board | 69 |
| Κατέβασμα της σχεδίασης στο board | 69 |
| Κατέβασμα του Kernel Software Image | 69 |
| Φτιάχνοντας ένα μεγαλύτερο root filesystem και περνώντας τα πάντα σε κάρτα Compact Flash | 70 |
| Φτιάχνοντας το elf του kernel για χρήση με filesystem της compact flash | 71 |
| Δημιουργώντας το ACE file | 72 |
| Τρέχοντας την πρώτη μας εφαρμογή στον Powerpc | 72 |

1. Εισαγωγή

Με τον όρο συμπίεση βίντεο αναφερόμαστε στην προσπάθεια για περιορισμό της ποσότητας των δεδομένων που απαιτούνται για την απεικόνιση βίντεο. Η προσπάθεια αυτή εκτείνεται σε δυο τομείς, τον χωρικό (συμπίεση εντός μιας εικόνας/video frame) και τον χρονικό (συμπίεση κατά μήκος μιας ακολουθίας εικόνων/video frames). Υπάρχουν δυο κύριες μορφές συμπίεσης: η lossless συμπίεση, όπου τα αποσυμπιεσμένα δεδομένα ανταποκρίνονται πλήρως στα δεδομένα προ της συμπίεσης, και η lossy συμπίεση, όπου τα πλεονάζοντα δεδομένα ενός βίντεο αφαιρούνται κατά τη διαδικασία της συμπίεσης. Τα δεδομένα αυτά, θεωρούνται πλεονάζοντα, υπό την έννοια ότι η έλλειψή τους, περνάει απαρατήρητη (ή σχεδόν απαρατήρητη) από έναν ανθρώπινο παρατηρητή. Το μεγάλο κέρδος που υπάρχει, λόγω του μεγαλύτερου λόγου συμπίεσης, έχει καταστήσει τη lossy συμπίεση κυρίαρχη.

Υπηρεσίες, όπως η ψηφιακή τηλεόραση, το internet video streaming (YouTube, Google video κλπ), εμπορικά προϊόντα, όπως το DVD video, και νεότερες ανακαλύψεις όπως η τεχνολογία Blue-Ray, τα βιντεοτηλέφωνα και η τρισδιάστατη τηλεόραση (3D-TV), κάνουν τη συμπίεση βίντεο περισσότερο επίκαιρη από ποτέ. Αν και το κόστος ανά bit αποθηκευτικού χώρου, πλέον έχει ελαχιστοποιηθεί σε σχέση με το όχι ιδιαίτερα μακρινό παρελθόν, και αν και το αντίστοιχο κόστος μετάδοσης έχει ακολουθήσει πορεία αντιστρόφως ανάλογη του διαθέσιμου bitrate, η αποθήκευση και μετάδοση ασυμπιεστού (raw) βίντεο εξακολουθεί να είναι ασύμφορη. Το μέχρι τώρα κραταιό standard συμπίεσης, το «γερασμένο» πια MPEG-2, καλείται σιγά σιγά να αποσυρθεί και να δώσει τη θέση του σε νεότερα, πιο αποδοτικά και τεχνολογικά καινοτόμα standards. Τα δύο επικρατέστερα, με όμοια καταγωγή, αλλά εντελώς διαφορετική στόχευση, είναι το MPEG-4 και το H.264. Το πρώτο εξ αυτών, επιχειρεί να δει τη συμπίεση βίντεο από μια εντελώς νέα οπτική σε σχέση με τα παλαιότερα standard, ενώ το H.264, ακολουθεί την πεπατημένη οδό, με μεγαλύτερη όμως επιτυχία από τους προκατόχους του στον τομέα των επιδόσεων. Κάπου ανάμεσα, το AVS video standard, το οποίο θα μας απασχολήσει στην παρούσα εργασία, προσπαθεί να εδραιωθεί ως ο αντικαταστάτης του MPEG-2 στον αγώνα διαδοχής.

Σε αυτά, λοιπόν, τα πλαίσια, καθίσταται φανερή η ανάγκη για τη δυνατότητα γρήγορου encoding, και -πολύ περισσότερο- γρήγορου decoding συμπεσμένου βίντεο. Η ολοένα αυξανόμενη ευκρίνεια του βίντεο μάλιστα (όπως High Definition Video και Ultra High Definition Video), κάνει την ανάγκη αυτή επιτακτική. Στα πλαίσια αυτά, επιχειρείται τόσο στον τομέα της ακαδημαϊκής έρευνας, όσο και στη βιομηχανία η βελτιστοποίηση των μεθόδων κωδικοποίησης/αποκωδικοποίησης βίντεο, όσο και η βελτιστοποίηση των ήδη υπάρχουσών, στο πλαίσιο της παροχής

καλύτερου/πιο αποδοτικού hardware γενικής χρήσης, ή και embedded λύσεων ειδικού σκοπού.

Μια τέτοια προσπάθεια είναι και η παρούσα, η οποία προσπαθεί να συνδυάσει την βελτιστοποίηση του κώδικα του AVS video decoder με την υπολογιστική δύναμη και ευελιξία που μπορεί να παράσχει η λύση της επαναδιατασσόμενης λογικής.

2. Σκοπός της εργασίας

Η παρούσα εργασία, έχει ως σκοπό το porting του AVS video decoder σε υλικό επαναδιατασσόμενης λογικής, το χαρακτηρισμό των επιδόσεων του decoder και τη μετέπειτα σύγκρισή του με εκδόσεις για άλλες πλατφόρμες hardware που έχουν υλοποιηθεί στο τμήμα μας, όπως σε x86 αρχιτεκτονική.

Στα πλαίσια αυτού του porting περιλαμβάνονται τα εξής στάδια:

1. «Χτίσιμο» της απαραίτητης υποδομής, δηλαδή δημιουργία ενός βασικού συστήματος Linux σε FPGA board, το οποίο θα παρέχει όλες τις απαραίτητες υπηρεσίες χαμηλότερου επιπέδου (basic input/output κλπ).
2. Υλοποίηση software βελτιστοποιήσεων, για την αύξηση της ταχύτητας αποσυμπίεσης βίντεο.
3. Μετατροπή του κώδικα, όπου αυτό είναι απαραίτητο, για την επίτευξη συμβατότητας του κώδικα με την αρχιτεκτονική του PowerPC.
4. Μετατροπή του κώδικα, με σκοπό τη συμβατότητα με το interface ενός motion compensation accelerator, όπως αυτός έχει δημιουργηθεί στα πλαίσια εργασίας μεταπτυχιακού φοιτητή, για την επίτευξη ακόμη μεγαλύτερων επιδόσεων, και ενσωμάτωση αυτού στο σύστημα.
5. Μέτρηση επιδόσεων του AVS decoder, κατά τα προηγούμενα στάδια, ποιοτική μελέτη αυτών, καθώς και σύγκριση-αιτιολόγηση ομοιοτήτων και διαφορών σε σχέση με τις υλοποιήσεις στις υπόλοιπες πλατφόρμες.

Παράλληλα, για την επίτευξη των παραπάνω, απαιτείται μελέτη διάφορων ζητημάτων, όπως γενικά της φιλοσοφίας και λειτουργίας των ενσωματωμένων συστημάτων, των FPGAs, των διαφόρων εργαλείων, των κυριότερων χαρακτηριστικών της αρχιτεκτονικής του PowerPC, των internals ζητημάτων του Linux, όπως παραμετροποίηση για χρήση σε embedded solutions, device drivers κ.α.

Τα ζητήματα αυτά θα συζητηθούν, εν παραλλήλω με τη διαδικασία υλοποίησης της εργασίας, όπου αυτά άπτονται των πρακτικών ζητημάτων, ή εν ίδει εισαγωγής και εξοικείωσης του αναγνώστη με τα θέματα που συζητούνται.

Για τη διευκόλυνση του αναγνώστη, επιλέξαμε να ακολουθήσουμε χρονολογικά την πορεία της εργασίας, αναλύοντας τα πρακτικά και θεωρητικά ζητήματα, όπως αυτά παρουσιάστηκαν στη διαδικασία περάτωσης της πτυχιακής αυτής εργασίας.

3. FPGAs

Εισαγωγικά

Η FPGA (field-programmable gate array) είναι ένα integrated circuit, το οποίο, σε αντίθεση, επί παραδειγματι, με το ASIC (application-specific integrated circuit), έχει τη δυνατότητα να επαναπρογραμματίζεται επί τόπου από τον (hardware) programmer. Από αυτή ακριβώς τη δυνατότητά της προέρχεται και ο χαρακτηρισμός "field-programmable" στην ονομασία της. Η λειτουργικότητα μιας FPGA, που την καθιστά ευέλικτη για πληθώρα εφαρμογών, καθορίζεται κυρίως από μια γλώσσα προγραμματισμού υλικού, η "hardware description language", όπως είναι ο αγγλικός όρος, ο οποίος συντέμνεται στο ακρωνύμιο "HDL". Τα γνωστότερα παραδείγματα τέτοιων γλωσσών, είναι η VHDL και η Verilog, με την τελευταία να κρατάει τα σκήπτρα στον τομέα της σύγχρονης βιομηχανίας υλικού και την πρώτη να κατέχει σημαντικότερο μερίδιο στον ακαδημαϊκό τομέα.

Βασικό στοιχείο μιας FPGA είναι τα λεγόμενα "logic blocks", τα οποία είναι στοιχεία προγραμματιζόμενης λογικής, τα οποία συνδέονται μεταξύ τους με ένα επίσης επαναδιατασσόμενο δίκτυο αγωγών διασύνδεσης. Τα λογικά αυτά blocks, μπορούν να υλοποιήσουν απλές λογικές πύλες, και κατ' επέκταση πολυπλοκότερες συνδυαστικές λειτουργίες. Στις περισσότερες FPGAs, μαζί με τα logic blocks περιλαμβάνονται και στοιχεία μνήμης, από απλά flip flops, μέχρι πιο σύνθετα block μνήμης.

Πλέον, η τάση είναι μαζί με τα στοιχεία μιας FPGA, όπως περιγράφηκαν παραπάνω, να περιλαμβάνεται στο board και κάποιος microprocessor. Για παράδειγμα, στη σειρά Virtex-4 και Virtex-5 της Xilinx (η οποία είναι η πιο γνωστή ίσως εταιρεία παραγωγής τέτοιων board) περιλαμβάνεται ο PowerPC embedded microprocessor. Μια παραλλαγή του παραπάνω, είναι η έλλειψη ενός hard processor core, και η χρήση αντί αυτού ενός ή παραπάνω soft processor cores, όπως είναι για παράδειγμα ο Microblaze, πάλι από τη Xilinx. Φυσικά, εφικτή είναι και μια ενδιαμέση λύση η οποία περιλαμβάνει τόσο τη μία, όσο και την άλλη μορφή μικροεπεξεργαστή.

Όσον αφορά τη διαδικασία υλοποίησης ενός FPGA design, αυτό περιλαμβάνει αρκετά στάδια, ξεκινώντας από την περιγραφή, όπως προαναφέραμε σε μια γλώσσα περιγραφής υλικού, ή με τη χρήση ενός schematic design (το οποίο είναι περισσότερο χρήσιμο σε μικρότερες σχεδιάσεις). Εν συνεχεία, με τη χρήση ενός EDA (electronic design automation) tool, προχωρούμε στη δημιουργία ενός netlist για κάποια συγκεκριμένη τεχνολογία. Αυτή η netlist προσαρμόζεται σε μια συγκεκριμένη FPGA, κατά τη διαδικασία του place-and-route, με τη χρήση εξειδικευμένου software, το οποίο κατά κανόνα παρέχεται από την εταιρεία

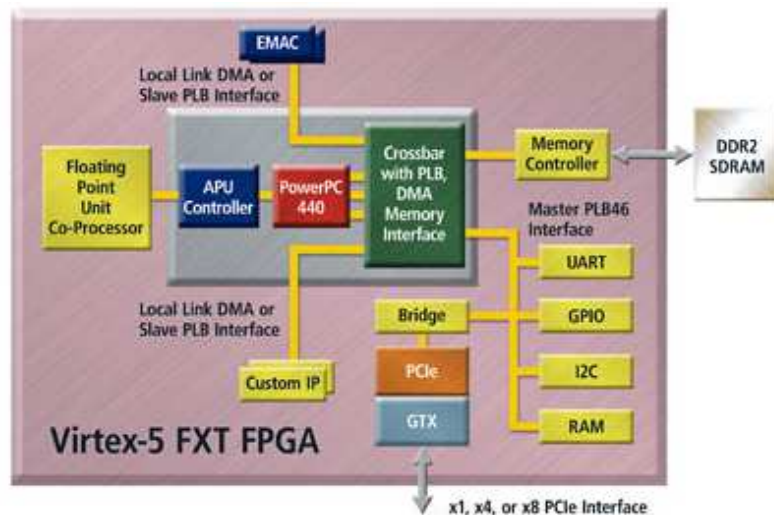
κατασκευής της FPGA. Απαραίτητη πριν το τελικό configuration της FPGA, είναι η διαδικασία του verification, όπου με τεχνικές timing analysis, simulation κ.α. επιβεβαιώνουμε την ορθή λειτουργία του συστήματός μας. Το simulation καλό είναι να γίνεται σε διάφορα στάδια, για την εξασφάλιση της ορθότητας μετά από κάθε στάδιο από αυτά που περιγράφηκαν πιο πάνω. Μετά από την επιτυχή περάτωση αυτού του σταδίου, είμαστε έτοιμοι να δημιουργήσουμε το bitstream προς «κατέβασμα» στην FPGA.

Χρήσιμο θα ήταν να σημειωθεί, πως αντίστοιχα με το σύνηθες προγραμματιστικό πρότυπο, όπου υπάρχουν έτοιμες βιβλιοθήκες συναρτήσεων, βελτιστοποιημένων και ελεγμένων για την ορθή λειτουργία, έτσι και στο hardware design υπάρχουν τα λεγόμενα IP cores. Αυτά δεν είναι τίποτα άλλο, παρά προκαθορισμένα κυκλώματα, τα οποία παρέχονται από τους κατασκευαστές FPGA, είτε από τρίτους κατασκευαστές, συνήθως επί πληρωμή και υπό αυστηρά licences. Δε λείπουν όμως και οι open source λύσεις, όπως τα λεγόμενα OpenCores, τα οποία τυπικά δίδονται υπό άδειες όπως GPL, BSD κλπ.

Στις επόμενες δυο ενότητες, θα εξετάσουμε το board που χρησιμοποιούμε κατά τη διάρκεια της εργασίας αυτής, καθώς και τον επεξεργαστή που είναι ο «πυρήνας» του board και της σχεδίασής μας. Θα εξετάσουμε τα κυριότερα χαρακτηριστικά τους, ώστε να πάρουμε μια εικόνα της υπολογιστικής δύναμης και των εργαλειών που έχουμε στα χέρια μας, ώστε να γνωρίσουμε τα όρια της σχεδίασής μας και το που μπορούμε να φτάσουμε από άποψη επιδόσεων.

Xilinx ML507

Το board που χρησιμοποιήσαμε στα πλαίσια αυτής της εργασίας είναι το Virtex-5 ML507 FXT, το οποίο είναι ένα από τα τελευταία στη σειρά των embedded solutions της κραταιής στο χώρο Xilinx. Πρόκειται για μια πλατφόρμα ειδικά βελτιστοποιημένης για embedded processing, DSP, και memory intensive εφαρμογές, με υψηλότερης ταχύτητας I/O απαιτήσεις. Το μοντέλο FXT, είναι το μοναδικό της σειράς 5, το οποίο περιλαμβάνει ένα ή δυο PowerPC 440 embedded blocks (στο δική μας περίπτωση, έναν), για ακόμη υψηλότερες επιδόσεις. Στην αμέσως επόμενη ενότητα, θα δούμε κάποια βασικά χαρακτηριστικά του PowerPC, που χρησιμοποιούμε στο σύστημα που υλοποιήσαμε. Στην παρακάτω εικόνα (εικόνα 1) μπορούμε να δούμε ένα παράδειγμα συστήματος, το οποίο ομοιάζει πολύ με το σύστημα το οποίο θα χρησιμοποιήσουμε και εμείς στην παρούσα εργασία. Το παρουσιάζουμε μόνο σαν εισαγωγή, και σε επόμενη παράγραφο θα υπεισέλθουμε σε περισσότερες λεπτομέρειες.

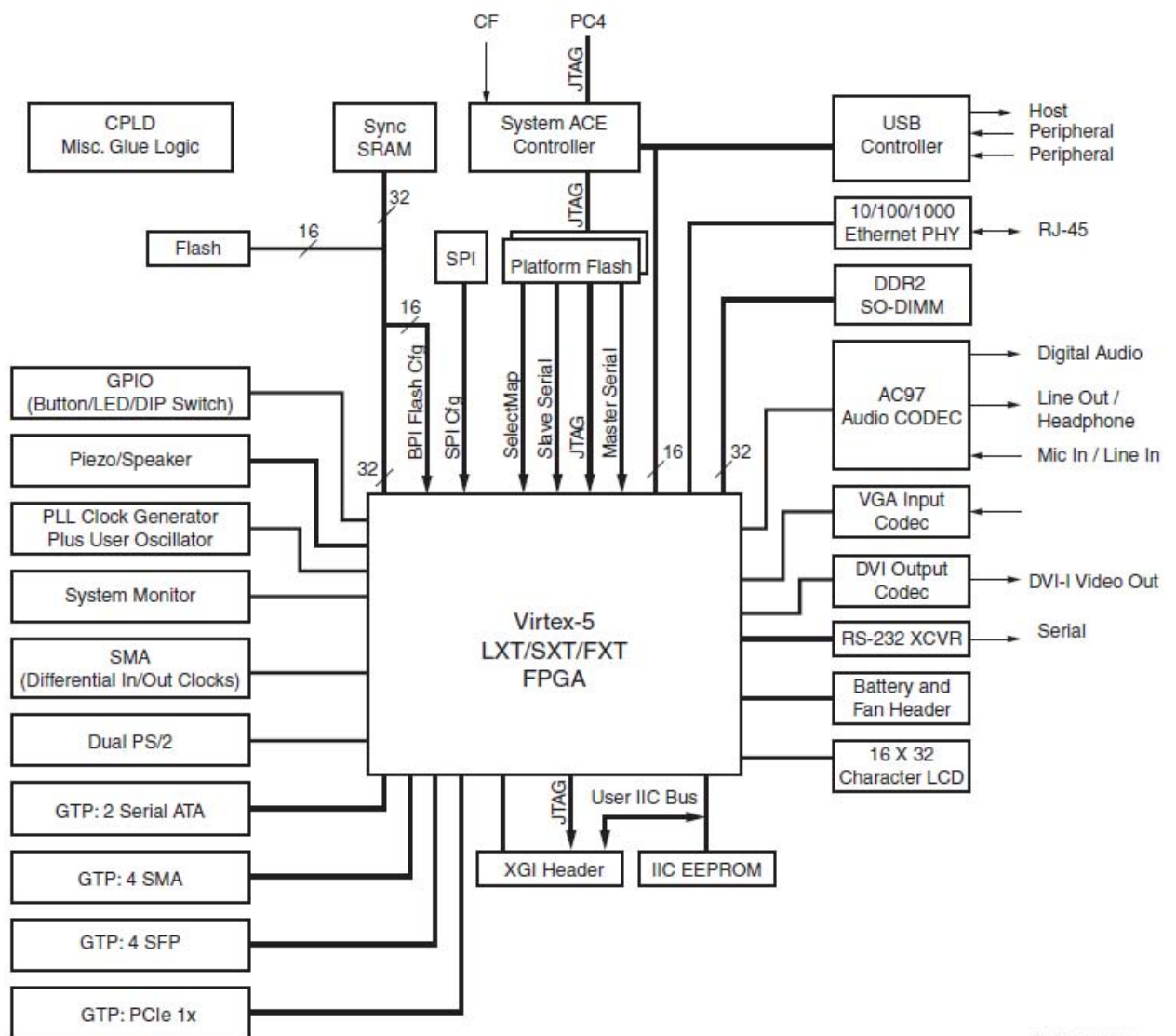


Εικόνα 1: Ένα παράδειγμα ενσωματωμένου συστήματος με PPC.

Θα αναφέρουμε επιγραμματικά μερικά από τα κυριότερα χαρακτηριστικά της πλατφόρμας ML507, για να αντιληφθούμε τις δυνατότητες αυτής:

- Xilinx Virtex-5 FPGA (XC5VFX70T)
- Δυο Xilinx XCF32P Platform Flash PROMs (των 32MB η καθεμιά) για αποθήκευση μεγάλων device configurations.
- Xilinx System ACE CompactFlash configuration controller
- Xilinx XC95144XL CPLD για glue logic
- 64-bit wide, 256MB DDR2 SODIMM
- General purpose DIP switches, LEDs, pushbuttons και rotary encoder
- Stereo AC97 audio codec με line-in, line-out, 50mW headphone, microphone-in jacks, SPDIF digital audio jacks και piezo audio transducer
- RS232 serial port
- LCD display 2 γραμμών, 16 χαρακτήρων η καθεμιά
- Μια 8KB IIC EEPROM
- PS/2 mouse και keyboard connectors
- Video input/output
- 10/100/1000 tri-speed Ethernet PHY transceiver και RJ-45
- USB interface chip και υποδοχές
- JTAG configuration port
- Υποστήριξη PCI express και SATA

Στη παρακάτω εικόνα (εικόνα 2), μπορούμε να δούμε με εύληπτο τρόπο το Block diagram του ML507 evaluation board, το οποίο περιλαμβάνει την Virtex-5 FXT FPGA, καθώς και τα χαρακτηριστικά που περιγράψαμε παραπάνω, μαζί με τα υπόλοιπα τα οποία δεν αναφέραμε για οικονομία χώρου και λόγω της ιδιαίτερα λεπτομερούς φύσης τους, που δε χρειάζεται να απασχολεί τον αναγνώστη.



Εικόνα 2: Virtex-5 FPGA ML507 Evaluation Platform Block Diagram

Power PC

Εσκεμμένα, παραλείψαμε μια πιο λεπτομερή αναφορά στον PowerPC στην παραπάνω ενότητα, γιατί θεωρούμε ότι αξίζει από μόνος του μια ενότητα, στην οποία θα συζητηθούν τα κυριότερα χαρακτηριστικά του, καθώς κάποια από αυτά θα αναφερθούν σε επόμενη ενότητα, καθώς θα παίξουν σαφώς ρόλο στην ανάλυση των επιδόσεων που θα καταγράψουμε για τον AVS decoder.

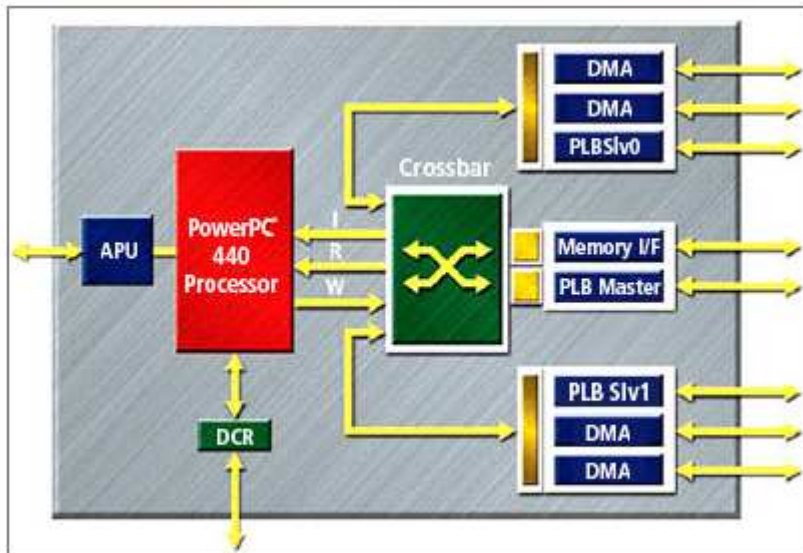
Στα χαρακτηριστικά του PowerPC 440 embedded block περιλαμβάνονται τα εξής:

- ❖ Embedded PowerPC 440 core:
 - Superscalar RISC αρχιτεκτονική
 - Μέχρι 550Mhz συχνότητα λειτουργίας
 - Pipeline 7 επιπέδων
 - Multiple instructions per cycle
 - Out-of-order execution
 - 32Kbyte, 64-way set associative level 1 instruction cache
 - 32Kbyte, 64-way set associative level 1 data cache

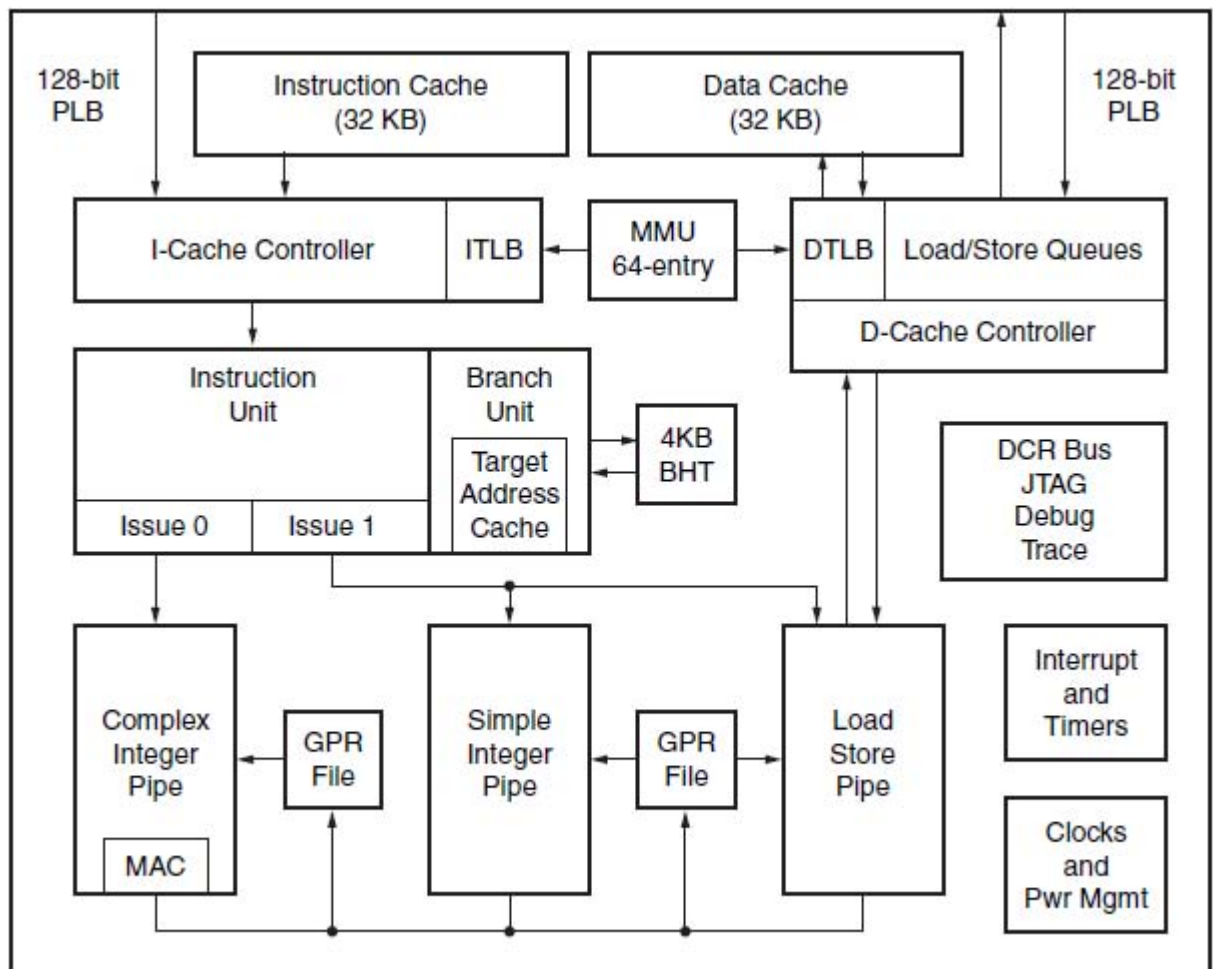
- ❖ Embedded crossbar για βελτίωση των αποδόσεων του συστήματος (βελτιωμένο processor bandwidth, με ελαχιστοποίηση του latency και point-to-point connectivity):
 - 128-bit Processor Local Buses (PLBs)
 - Integrated scatter/gather DMA controllers
 - Dedicated interface για σύνδεση με τον DDR2 memory controller
 - Αυτόματος συγχρονισμός για non-integer PLB-to-CPU clock ratios

- ❖ Auxiliary Processor Unit (APU) Interface & Controller:
 - Άμεση σύνδεση από το PPC440 embedded block σε FPGA fabric-based coprocessors (πχ για αυτόματα αποκωδικοποίηση Floating point instructions)
 - Υποστήριξη custom instructions, δίχως pipeline stalls στον PPC440.
 - Ιδιαίτερα αποδοτικό microcontroller-style interfacing

Τα παραπάνω απεικονίζονται σχηματικά στην εικόνα 3, ενώ στην εικόνα 4 μπορούμε να δούμε την εσωτερική οργάνωση του PPC440 Core:



Εικόνα 3: PowerPC embedded block



UG200_c1_01_022707

Εικόνα 4: Block diagram του PPC440 Embedded Processor

Endianess

Όταν κάποιος μεταφέρει κώδικα από μια αρχιτεκτονική σε μια άλλη, όπως στην περίπτωση μας από IA32 σε Power architecture, , πρέπει να έχει υπόψη του τις διαφορές στην αρχιτεκτονική που μπορεί να έχουν επίπτωση στη λειτουργικότητα των εκτελέσιμων που παράγονται από τον εν λόγω κώδικα.

Ένα από τα ζητήματα αυτά είναι και το endianess του επεξεργαστή. Με τον όρο endianess αναφερόμαστε στο πως αναπαριστώνται δεδομένα παραπάνω του ενός byte, και όπως ήδη αναφέραμε είναι χαρακτηριστικό της αρχιτεκτονικής του συστήματος. Μια συχνή παρανόηση είναι πως το λειτουργικό σύστημα καθορίζει το μοντέλο του endianess που ακολουθείται, όμως στην πραγματικότητα ακόμη και το πώς υλοποιείται το ίδιο το λειτουργικό σύστημα εξαρτάται από το μοντέλο endianess της αρχιτεκτονικής του επεξεργαστή.

Το endianess είναι ένα σημαντικό ζήτημα, μιας και είτε το λογισμικό ή απλώς δεδομένα μοιράζονται ανάμεσα σε συστήματα υπολογιστών, τα οποία μπορεί να έχουν εντελώς διαφορετική αρχιτεκτονική, και συνεπώς διαφορετικό endianness model. Όταν λοιπόν ένα κομμάτι λογισμικού γράφεται για ένα συγκεκριμένο μοντέλο endianness, δίχως να λαμβάνεται υπόψη το ζήτημα του portability, είναι λογικό και αναμενόμενο να προκύπτουν προβλήματα, όταν κάποιος προσπαθήσει να τρέξει το εν λόγω λογισμικό σε ένα σύστημα διαφορετικού endianness, για λόγους στους οποίους θα αναφερθούμε εν συντομία πιο κάτω. Είναι λοιπόν μια καλή προγραμματιστική τεχνική να γράφουμε λογισμικό, το οποίο να είναι, όπως λέμε, endian-neutral. Έτσι, το λογισμικό μας μπορεί να έχει μεγαλύτερη ευκολία χρήσης σε διαφορετικές πλατφόρμες, δίχως να χρειάζεται αλλαγές, είτε από εμάς (ειδικά αν πρόκειται για εμπορικό λογισμικό), είτε από τον χρήστη. Υπάρχουν συγκεκριμένες πρακτικές για να γράφει κανείς endian-neutral κώδικα, αλλά δε θα μας απασχολήσουν ιδιαίτερα στα πλαίσια της παρούσας εργασίας.

Ως endianness, λοιπόν, ορίζουμε το format με το οποίο αποθηκεύονται multi-byte δεδομένα στη μνήμη του υπολογιστή. Για την ακρίβεια, περιγράφει την τοποθεσία του most significant byte (MSB) και του least significant byte (LSB) μιας διεύθυνσης στη μνήμη. Υπάρχουν δυο μοντέλα endianness, το Big-endian και το Little-endian model, καθένα από τα οποία έχει τα πλεονεκτήματα και τα μειονεκτήματά του, και αντίστοιχα τους δικούς του υποστηρικτές και «εχθρούς».

Στο μοντέλο του Big-endian το MSB αποθηκεύεται στη χαμηλότερη διεύθυνση μνήμης, ενώ αντίθετα στο Little-endian model, το LSB είναι αυτό που αποθηκεύεται στη χαμηλότερη διεύθυνση μνήμης. Μιλώντας για χαμηλότερη διεύθυνση μνήμης multi-byte δεδομένων, αναφερόμαστε στην διεύθυνση της μνήμης από όπου ξεκινούν τα δεδομένα. Για καλύτερη κατανόηση του παραπάνω, δίνουμε το παρακάτω σχήμα (εικόνα 5), που απεικονίζει με εύληπτο τρόπο τις διαφορές. Δείχνουμε πως αποθηκεύεται η 32-bit hex τιμή 0x12345678 στα δυο είδη

endianness. Η χαμηλότερη διεύθυνση μνήμης αναπαριστάται στην πιο αριστερή θέση, στο byte 00.

| Endian Order | Byte 00 | Byte 01 | Byte 02 | Byte 03 |
|---------------|----------|---------|---------|----------|
| Big Endian | 12 | 34 | 56 | 78 (LSB) |
| Little Endian | 78 (LSB) | 56 | 34 | 12 |

Εικόνα 5: Παράδειγμα memory addressing σε Big- και Little endian model

Αξίζει να σημειωθεί, για αποφυγή παρανοήσεων, ότι η τιμή του Multi-byte δεδομένου είναι η ίδια και στις δυο περιπτώσεις, όταν η αναφορά σε αυτά γίνεται στον τύπο που έχουν δηλωθεί, όμως το πρόβλημα προκύπτει όταν γίνεται αναφορά σε υποσύνολα των δεδομένων, διαφορετικά από τον τύπο στον οποίο αυτό έχει δηλωθεί.

Αυτή είναι μια περίπτωση στην οποία το endianness μπορεί να δημιουργήσει προβλήματα κατά το porting ενός προγράμματος, από μια αρχιτεκτονική σε μια άλλη, όπως στην περίπτωση μας από x86 σε Power architecture. Στην πρώτη προσπάθεια να τρέξουμε τον decoder στον PowerPC της FPGA, εμφανίστηκε ένα segmentation fault. Αφού έγινε το απαραίτητο debugging με τη χρήση του gdb, διαπιστώθηκε ότι οι τιμές κάποιων μεταβλητών που χρησιμοποιούνταν ως δείκτες ενός δισδιάστατου πίνακα είχαν λανθασμένες τιμές, προκαλώντας προσπέλαση σε θέσεις μνήμης που δεν είχαν δεσμευθεί για τον πίνακα αυτό. Επομένως, υποθέσαμε εύλογα πως κατά τη διαδικασία του bitstream parsing είχε προκύψει κάποιο λάθος, λόγω του endianness. Υπήρχε η περίπτωση, επίσης, τα δεδομένα του αρχείου εισόδου (encoded AVS file), που είχε γίνει encode σε x86 system, να επηρεάζονται από το endianness model, και να προκαλείται εξ αρχής πρόβλημα στο decoding, λόγω αυτού. Διαπιστώσαμε, όμως πως ο encoder γράφει τα δεδομένα στο αρχείο εξόδου του ως single byte data, και έτσι αυτό το πρόβλημα μπορούσε να θεωρηθεί ως μη υπάρχον. Εστιάσαμε, έτσι, στη διαδικασία του variable length decoding, κατά την οποία χρησιμοποιούνταν bit operations και bit fields, καθώς και αναφορές σε μεμονωμένα bytes εντός multi-byte δεδομένων. Οι διαδικασίες αυτές είναι σημεία στα οποία πρέπει να δίδεται προσοχή, σε περιπτώσεις porting προγραμμάτων μεταξύ διαφορετικών αρχιτεκτονικών.

Ξεκινήσαμε λοιπόν το debugging, side-by-side, στον x86 και στο FPGA board, ώστε να διαπιστώσουμε τα σημεία εκείνα τα οποία θα χρειαζόνταν αλλαγές. Προχωρώντας, όμως, διαπιστώνουμε όλο και περισσότερο, ότι δεν υπήρχαν τα προβλήματα που θα αναμέναμε, έτσι μετά από αναζήτηση διαπιστώσαμε ότι οι γλώσσες προγραμματισμού, όπως η C που χρησιμοποιούμε, παρέχουν ένα level of abstraction, όταν πρόκειται για bitwise operations, όπου για παράδειγμα ο τελεστής >> θα κάνει shift τα bits προς το least significant digit, ανεξάρτητα του endianness. Παρ' όλα αυτά, άλλα ζητήματα endianness, όπως cast των pointers κ.α. θα μπορούσαν να δημιουργήσουν προβλήματα, οπότε θα έπρεπε να συνεχίσουμε, μιας και το πρόβλημα, ούτως ή άλλως παρέμενε. Ακολουθώντας την αντίστροφη διαδικασία στο debugging, φτάσαμε σε ένα σημείο, όπου εντοπιζόταν το πρόβλημα. Τελικά, το πρόβλημα ήταν κάτι μη αναμενόμενο, μιας και ήταν κάτι που ίσως γνώριζε κάποιος με εμπειρία προγραμματισμού σε Power architecture, όχι όμως και κάποιος που για πρώτη φορά μεταφέρει κάποιο πρόγραμμα σε αυτόν, και «λειτουργεί» με βάση την εμπειρία του και τα δεδομένα που είχε από τον προγραμματισμό σε x86 architecture. Το standard της ANSI C δεν ορίζει αν με τον προσδιορισμό char, υπονοείται signed ή unsigned char. Αυτό αποφασίζεται συνήθως, ανάλογα με το τι είναι ευκολότερο ή πιο αποδοτικό για το προκείμενο hardware. Έτσι, ενώ σε ένα x86 σύστημα, ο προσδιορισμός char υπονοεί ότι μπορεί να πάρει και αρνητικές τιμές, σε ένα σύστημα βασισμένο σε Power αρχιτεκτονική το PPC EABI ορίζει ότι μια μεταβλητή τύπου char είναι unsigned.

Έτσι, η λύση ήταν πλέον προφανής: αλλαγή των σημείων στον κώδικα, όπου υπήρχαν μεταβλητές τύπου char, με σαφή δήλωσή τους ως signed ή unsigned. Εναλλακτικά, δήλωση κατά το compiling, με το flag -fsigned-char, με την οποία ορίζεται ότι η δήλωση char περιλαμβάνει και αρνητικές τιμές.

Μετά από αυτή την, εύκολη στην υλοποίηση λύση, αλλά δύσκολο στο να γίνει αντιληπτή και να εντοπιστεί αιτία του προβλήματος, ο AVS decoder εκτελέστηκε σωστά και παρήγαγε το αποσυμπιεσμένο βίντεο εξόδου, σε γυν μορφή.

4. Θεωρία Video Codecs

AVS- Audio Video Standard

Το AVS (audio video standard) είναι ένα σχετικά νέο standard συμπίεσης ήχου και βίντεο, το οποίο ξεκίνησε με πρωτοβουλία της κυβέρνησης της Κίνας, σε μια προσπάθεια να ελαττώσει τα υπέρογκα royalty fees τα οποία καλούνταν να πληρώσουν για IP (intellectual property) οι κινέζοι κατασκευαστές audio/video συσκευών, εφόσον χρησιμοποιούσαν codecs οι οποίοι ανήκαν στην πλειοψηφία τους σε ξένες εταιρείες. Ενδεικτικά, ένα DVD player το οποίο κατασκευάζεται στην Κίνα, κοστίζει περίπου \$30 και τα IP charges φθάνουν στα \$20, καταλαμβάνοντας δηλαδή περίπου το 40% του κόστους κατασκευής ενός DVD player. Αντιλαμβάνεται, λοιπόν κανείς, ότι για ακόμη μια φορά επαληθεύεται το ρητό: “money makes the world move round”, αφού οικονομικοί κυρίως λόγοι ήταν αυτοί που ώθησαν στη δημιουργία του AVS.

Όσον αφορά στο τεχνολογικό τομέα, η αποκωδικοποίηση βίντεο, όπως ορίζεται στο AVS standard (με πολλές ομοιότητες με αντίστοιχα video standards), εκτελείται για κάθε frame και το βασικό στοιχείο αποτελεί ένα macroblock (μια υποπεριοχή βίντεο μεγέθους 16x16 pixels). Αντίστοιχα με παλαιότερα και με τα υπόλοιπα σύγχρονα video standards, βασίζεται στις ίδιες αρχές του block-based intra και inter prediction, καθώς και του ίδιου transform-based coding framework. Το intra prediction υπολογίζεται βάσει γειτονικών pixels στο από πάνω, πάνω αριστερά, πάνω δεξιά και αριστερο macroblock. Το AVS standard καθορίζει και υποστηρίζει τέσσερα μεγέθη block (8x8, 8x16, 16x8, 16x16) για inter prediction, σε αντίθεση με το H.264, το οποίο υποστηρίζει μεγέθη block ακόμη και μεγέθους 4x4. Αυτή ήταν μια συνειδητή επιλογή των σχεδιαστών του AVS standard, που είχε ως σκοπό να μειώσει την υπολογιστική πολυπλοκότητα που θα υπεισερχόταν με τη χρήση block 4x4, αν και αυτό θα είχε ως αντιστάθμισμα βελτίωση στην ποιότητα του αποκωδικοποιημένου βίντεο. Η ακρίβεια των motion vectors, από την άλλη, είναι $\frac{1}{4}$ pixel. Τέλος, ο AVS χρησιμοποιεί φίλτρο deblocking στο αποκωδικοποιημένο frame, ώστε να ελαττωθούν τα blocking artifacts στις ακμές των blocks, ειδικά σε περιοχές χαμηλής χωρικής συχνότητας. Το φίλτρο αυτό επανακαθορίζεται αυτόματα ανάλογα με το περιεχόμενο του βίντεο και την τιμή Qp του κβαντιστή κάθε block, τα οποία και έχουν επίπτωση στο ποσοστό δημιουργίας blocking artifacts.

Βασικές έννοιες video

Progressive and interlaced video sequence

Υπάρχουν δυο βασικά είδη video sequences. Το progressive και το interlaced video sequence, τα οποία θα περιγράψουμε με συντομία παρακάτω.

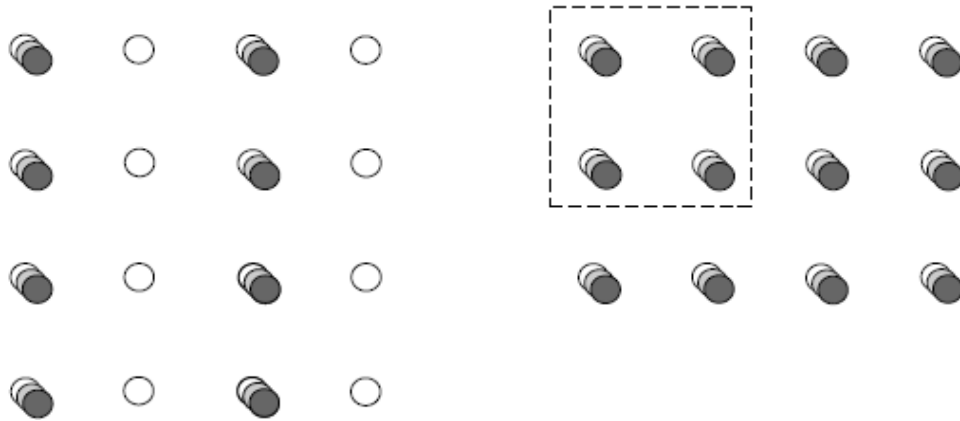
Το κάθε frame αποτελείται από τρία βασικά στοιχεία-πίνακες: τον πίνακα δειγμάτων luminance (Y) και δυο πίνακες δειγμάτων chrominance, τους Cb και Cr. Η σχέση αυτών των στοιχείων με τα βασικά αναλογικά σήματα χρωμάτων για το κόκκινο, πράσινο και μπλε μπορεί να καθορίζεται στο bitstream. Δεν αλλάζει όμως τη διαδικασία του decoding.

Ένα field αποτελείται από interlace γραμμές των παραπάνω τριών πινάκων από samples, που όλες μαζί αποτελούν ένα frame. Οι πρώτη, τρίτη, πέμπτη κ.ο.κ. γραμμές, αποτελούν ένα field, το οποίο ονομάζεται top field, ενώ οι δεύτερη, τέταρτη, έκτη κ.ο.κ. γραμμές αποτελούν το bottom field.

Η έξοδος του decoder είναι μια σειρά από frames και υπάρχει μια χρονική απόσταση μεταξύ δυο frames. Όταν λοιπόν μιλάμε για interlaced sequence, υπάρχει μια μη μηδενική χρονική απόσταση μεταξύ των δυο fields μιας εικόνας. Αντίθετα, για progressive sequences, το time interval ανάμεσα στα δυο fields είναι μηδενικό (εμφανίζονται ταυτόχρονα).

Picture formats

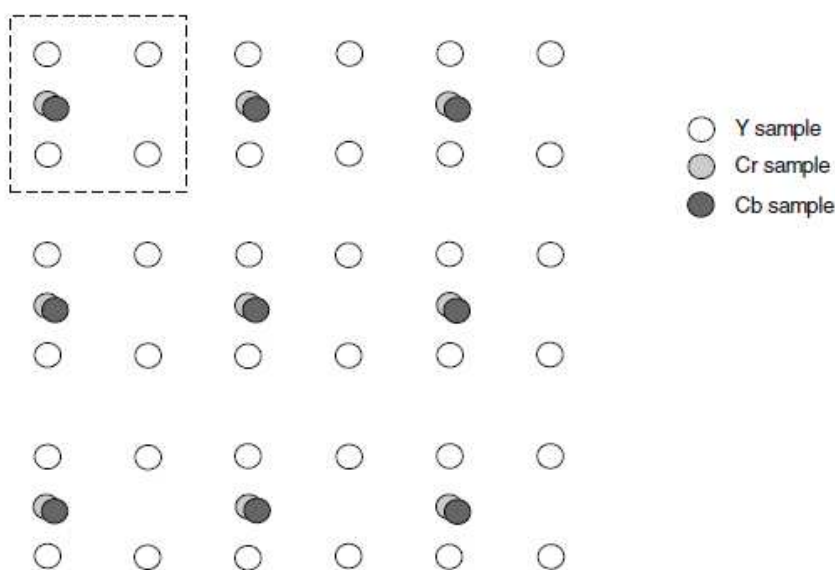
Τα sampling formats που καθορίζονται από το AVS standard είναι τρία, και απεικονίζονται στις εικόνες που έπονται. Όπως αναφέρουμε στο γλωσσάρι των βασικών εννοιών του video (βλ. παράρτημα Α), κάθε εικόνα βίντεο περιέχει τρεις συνιστώσες: μία για το luminance και δύο για το chrominance. Κάθε εικόνα, όπως είναι επίσης γνωστό, αποτελείται από pixels. Όταν λοιπόν οι τρεις προαναφερθείσες συνιστώσες έχουν την ίδια ανάλυση, και επομένως κάθε μια εκ των τριών συνιστωσών εμφανίζεται σε κάθε pixel, τότε μιλάμε για 4:4:4 sampling format. Όταν το chrominance (Cb και Cr) περιορίζονται στο μισό στην οριζόντια διάσταση, τότε μιλάμε για το 4:2:2 sampling. Οι αριθμοί μπορούν να ερμηνευτούν ως η σχετική συχνότητα του κάθε στοιχείου ως προς τα υπόλοιπα. Για παράδειγμα στο 4:2:2 για κάθε τέσσερα luminance samples, υπάρχουν από δύο Cb και Cr, αντίστοιχα. Οι δυο αυτές περιπτώσεις απεικονίζονται στο παρακάτω σχήμα.



Εικόνα 6: 4:2:2 sampling και 4:4:4 sampling

Από τα δυο προαναφερθέντα formats, στο πρώτο διατηρείται η πλήρης πιστότητα των χρωμάτων, ενώ το δεύτερο αν και υπολείπεται του πρώτου, θεωρείται format υψηλής πιστότητας χρωμάτων. Αυτό ισχύει, δεδομένου του τρόπου με τον οποίο το ανθρώπινο μάτι αντιλαμβάνεται τα χρώματα, όπου τη μεγαλύτερη επίδραση έχει η φωτεινότητα (luminance) και όχι οι συνιστώσες του chrominance.

Η τρίτη και πιο συνηθισμένη περίπτωση είναι το 4:2:0 sampling format, όπου η σχετική συχνότητα εμφάνισης των Cb και Cr είναι η μισή, τόσο στην οριζόντια, όσο και στην κάθετη κατεύθυνση. Στην περίπτωση αυτού του format, η ονομασία του δεν ακολουθεί το πρότυπο που αναφέραμε πιο πάνω, αλλά έχει επικρατήσει παραδοσιακά και δε θα πρέπει να συγχέεται. Είναι αυτό το format το οποίο χρησιμοποιείται κατά κανόνα σε εφαρμογές videoconference, στην ψηφιακή τηλεόραση, καθώς και στο DVD video. Μια σχηματική απεικόνιση μπορούμε να δούμε στην ακόλουθη εικόνα.



Εικόνα 7: 4:2:0 sampling

Picture types

Το specification του AVS ορίζει τρεις τύπους εικόνας:

A) Intra-coded picture (I frame): δεν αναφέρεται σε άλλες εικόνες κατά τη διάρκεια του decoding.

B) Predicted inter-coded picture (P frame): αναφέρεται σε άλλες εικόνες, οι οποίες προηγούνται (σε σειρά εμφάνισης- display order) της τρέχουσας εικόνας.

Γ) Bidirectional inter-coded picture (B frame): αναφέρεται σε εικόνες που προηγούνται και έπονται της τρέχουσας εικόνας (κατά display order).

Picture reordering

Αν στο bitstream δεν υπάρχουν B frames, τότε η σειρά των εικόνων, όπως είναι κωδικοποιημένες στο bitstream και η σειρά των εικόνων όπως προκύπτει από τη διαδικασία της αποκωδικοποίησης, είναι η ίδια. Αν όμως υπάρχουν B frames, τότε η σειρά αυτή είναι διαφορετική, επομένως θα πρέπει να υπάρχει αλλαγή της σειράς των εικόνων όπως εμφανίζονται στην έξοδο.

Παρακάτω ακολουθεί ένα παράδειγμα του πως έχει δοθεί το (ασυμπιεστο) βίντεο στην είσοδο του encoder, πως έχει κωδικοποιηθεί αυτό στο bitstream, και πως θα εμφανισθεί τελικά στην έξοδο, μετά τη διαδικασία του decoding.

➤ Στην είσοδο του encoder:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| I | B | B | P | B | B | P | B | B | I | B | B |

➤ Στο bitstream:

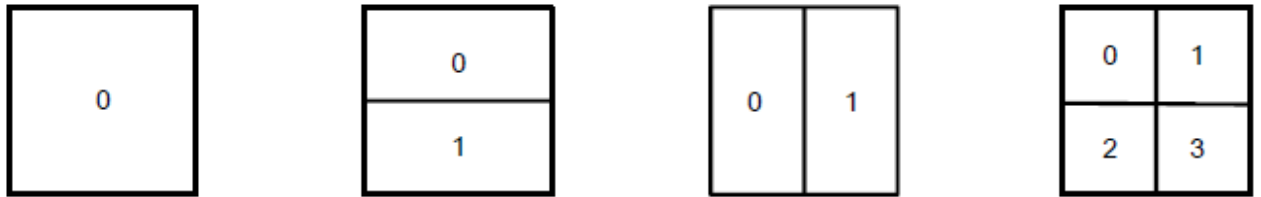
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|----|---|---|----|----|
| 1 | 4 | 2 | 3 | 7 | 5 | 6 | 10 | 8 | 9 | 13 | 11 |
| I | P | B | B | P | B | B | I | B | B | P | B |

➤ Στην έξοδο του decoder (το λεγόμενο display order):

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| I | B | B | P | B | B | P | B | B | I | B | B |

Macroblock

Κάθε εικόνα ενός video sequence χωρίζεται σε ένα αριθμό από macroblocks, τα οποία αντιπροσωπεύουν ορθογώνιες περιοχές εντός της εικόνας. Για τη χρήση στο motion compensation, κάθε macroblock δύναται να χωριστεί σε μικρότερα blocks, τα οποία μπορεί να έχουν μέγεθος 16x16, 16x8, 8x16 και 8x8 pixels. Σχηματικά, οι τρόποι διαχωρισμού εμφανίζονται στην παρακάτω εικόνα:

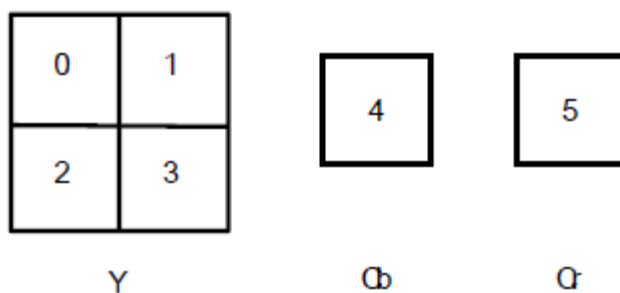


Εικόνα 8: Τρόποι διαχωρισμού των macroblocks για τους σκοπούς του motion compensation

8x8 Block

Το block 8x8 είναι το βασικό στοιχείο του μετασχηματισμού και του block scan, που μπορεί να παριστάνει δεδομένα της αρχικής εικόνας, δεδομένα της reconstructed εικόνας, ή 8x8 δεδομένα του ακέραιου μετασχηματισμού.

Στο 4:2:0 format που υποστηρίζει μέχρι στιγμής το profile του AVS codec, το macroblock περιλαμβάνει τέσσερα 8x8 luminance blocks (Y) και δυο chrominance blocks (Cb και Cr), όπως φαίνεται στο παρακάτω σχήμα:



Εικόνα 9: Τρόπος διαχωρισμού ενός macroblock σε 8x8 blocks στο 4:2:0 format

Αναλυτική παρουσίαση του Motion Compensation

Εισαγωγή

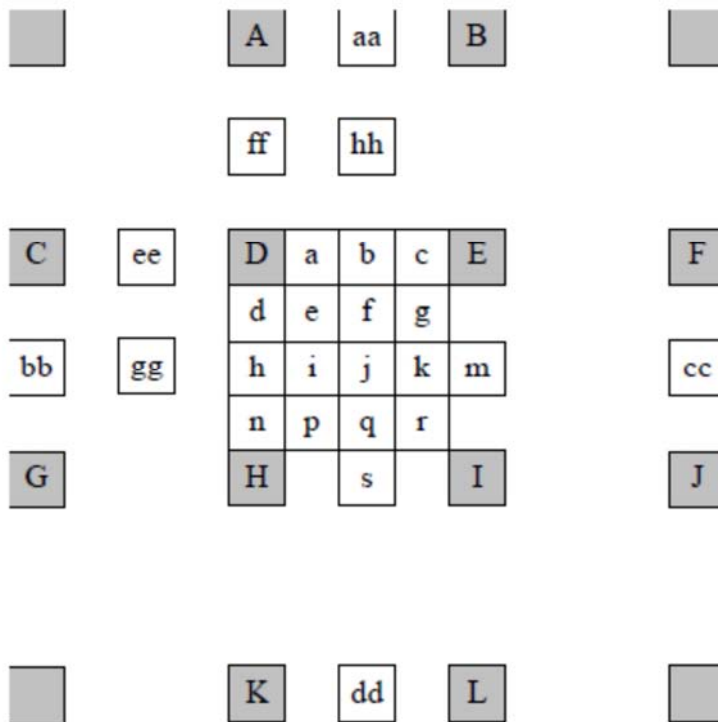
Όπως αναφέραμε ακροθιγώς νωρίτερα, το predictive coding είναι μια ευρέως χρησιμοποιούμενη τεχνική στους video codecs. Είδαμε λοιπόν πως λειτουργεί στην πλευρά του encoder η διαδικασία του motion estimation, για τον υπολογισμό των motion vectors και πως η πληροφορία που ουσιαστικά αποθηκεύεται στο συμπιεσμένο αρχείο είναι η διαφορά (residual frame) του τρέχοντος frame από το frame αναφοράς, καθώς και τα motion vectors. Έτσι, ο decoder, χρησιμοποιώντας τις διαφορές αυτές σε συνδυασμό με τα motion vectors και τις τιμές των δειγμάτων στις περιοχές του reference frame που αυτά υποδεικνύουν, μπορεί να ανακατασκευάσει το αρχικό frame. Η ιδιότητα που αξιοποιούμε με αυτή την τεχνική είναι η μεγάλη ομοιότητα ανάμεσα σε γειτονικά frames, στα οποία οι περισσότερες αλλαγές οφείλονται είτε στην κίνηση της κάμερας, είτε στην κίνηση αντικειμένων εντός του frame. Όπως είδαμε πιο πάνω, για ακόμη μεγαλύτερη ακρίβεια στην πρόβλεψη δε χρησιμοποιούνται μόνο ακέραια pixels, αλλά το motion compensation γίνεται με sub-pixel accuracy. Φυσικά, στο reference frame υπάρχουν μόνο τα ακέραια pixels. Όλα τα υπόλοιπα εξάγονται με τη διαδικασία του interpolation, η οποία και είναι υπολογιστικά ακριβή. Αυτός είναι και ο λόγος που επιθυμούμε να κάνουμε offload τη διαδικασία αυτή από τον επεξεργαστή (PPC) και αντί αυτού να χρησιμοποιήσουμε έναν ειδικό hardware accelerator.

Παρακάτω θα περιγράψουμε αρχικά τη διαδικασία του interpolation για τα luminance samples και αμέσως μετά για τα chrominance samples.

Interpolation process για τα luminance samples

Ξεκινώντας, πρέπει να σημειώσουμε πως ο AVS υποστηρίζει half sample και quarter sample interpolation και όπως αναφέραμε τα αντίστοιχα reference samples προκύπτουν από τη χρήση των motion vectors που υπολογίζονται από τον encoder κατά τη διαδικασία του encoding (στη φάση του motion estimation) και κωδικοποιούνται στο συμπιεσμένο bitstream.

Θα εξηγήσουμε τις διαφορετικές υποπεριπτώσεις interpolation που προκύπτουν στο υπόλοιπο αυτής της ενότητας, αφού πρώτα εξηγήσουμε με μια εικόνα (εικόνα 10) τις θέσεις των samples για την κάθε υποπερίπτωση.



Εικόνα 10: Οι θέσεις των ακεραίων, half και quarter samples

Για τον υπολογισμό των predictive values χρησιμοποιούνται δυο διαφορετικά tap filters, ένα για τα samples που βρίσκονται σε half pixel position και ένα για αυτά που βρίσκονται σε quarter pixel. Και τα δυο είναι 4 tapping filters, εκ των οποίων το πρώτο είναι το F1(-1, 5, 5, -1) και το δεύτερο το F2(1, 7, 7, 1).

Με βάση την εικόνα 10, θα αναλύσουμε το interpolation process αρχικά για τα half samples και στη συνέχεια για τα quarter samples.

Υπολογισμός Half Samples

- **Half sample b**

Αρχικά οι τιμές των τεσσάρων κοντινότερων ακεραίων samples φιλτράρονται με το F1 κατά την οριζόντια διεύθυνση, δίνοντας ως ενδιάμεση τιμή το $b' = (-C + 5D + 5E - F)$ και τελική predictive τιμή $b = \text{clip}((b' + 4) \gg 3)$.

- **Half sample h**

Αρχικά οι τιμές των τεσσάρων κοντινότερων ακεραίων samples φιλτράρονται με το F1 κατά την κάθετη διεύθυνση, δίνοντας ως ενδιάμεση τιμή το $h' = (-A + 5D + 5H - K)$ και τελική predictive τιμή $h = \text{clip}((h' + 4) \gg 3)$.

- **Half sample j**

Αρχικά οι τιμές των τεσσάρων κοντινότερων half samples φιλτράρονται με το F1 κατά την οριζόντια ή κάθετη διεύθυνση, δίνοντας ενδιάμεση τιμή $j' = (-bb + 5h' + 5m' - cc)$ ή $j' = (-aa + 5b' + 5s' - dd)$. Οι ενδιάμεσες τιμές aa, dd, s'

υπολογίζονται σύμφωνα με την υποπερίπτωση του b' που περιγράφηκε νωρίτερα, ενώ τα bb , cc , m' αντίστοιχα με την υποπερίπτωση του h' . Η τελική predictive τιμή υπολογίζεται ως $j = \text{clip}((j' + 32) \gg 6)$, ενώ αξίζει να σημειώσουμε ότι η τιμή που θα πάρουμε είτε με το οριζόντιο, είτε με το κάθετο φίλτρο είναι η ίδια.

Υπολογισμός Quarter Samples

- **Quarter sample a**

Αρχικά οι τιμές των τεσσάρων κοντινότερων samples ee , D' , b' , E' φιλτράρονται με το F2 κατά την οριζόντια διεύθυνση, δίνοντας ως ενδιάμεση τιμή το $a' = (ee + 7D' + 7b' + E')$ και τελική predictive τιμή $a = \text{clip}((a' + 64) \gg 7)$. Τα ee , b' είναι οι ενδιάμεσες τιμές των half samples των αντίστοιχων θέσεων, ενώ τα D' , E' είναι τα ακέραια samples των οποίων οι ενδιάμεσες τιμές είναι μεγεθυμένες κατά 8 φορές. Το interpolation process για το quarter sample c είναι ίδιο με αυτό του a .

- **Quarter sample d**

Αρχικά οι τιμές των τεσσάρων κοντινότερων samples ff , D' , h' , H' φιλτράρονται με το F2 κατά την κάθετη διεύθυνση, δίνοντας ως ενδιάμεση τιμή το $d' = (ff + 7D' + 7h' + H')$ και τελική predictive τιμή $d = \text{clip}((d' + 64) \gg 7)$. Τα ff , h' είναι οι ενδιάμεσες τιμές των half samples των αντίστοιχων θέσεων, ενώ τα D' , H' είναι τα ακέραια samples των οποίων οι ενδιάμεσες τιμές είναι μεγεθυμένες κατά 8 φορές. Το interpolation process για το quarter sample n είναι ίδιο με αυτό του d .

- **Quarter sample i**

Αρχικά οι τιμές των τεσσάρων κοντινότερων samples gg , h'' , j' , m'' φιλτράρονται με το F2 κατά την οριζόντια διεύθυνση, δίνοντας ως ενδιάμεση τιμή το $i' = (gg + 7h'' + 7j' + m'')$ και τελική predictive τιμή $i = \text{clip}((i' + 512) \gg 10)$. Τα gg , j' είναι οι ενδιάμεσες τιμές των half samples των αντίστοιχων θέσεων, ενώ τα h'' , m'' είναι τα ακέραια samples των οποίων οι ενδιάμεσες τιμές είναι μεγεθυμένες κατά 8 φορές. Το interpolation process για το quarter sample k είναι ίδιο με αυτό του i .

- **Quarter sample f**

Αρχικά οι τιμές των τεσσάρων κοντινότερων samples hh , b'' , j' , s'' φιλτράρονται με το F2 κατά την κάθετη διεύθυνση, δίνοντας ως ενδιάμεση τιμή το $f' = (hh + 7b'' + 7j' + s'')$ και τελική predictive τιμή $f = \text{clip}((f' + 512) \gg 10)$. Τα hh , j' είναι οι ενδιάμεσες τιμές των half samples των αντίστοιχων θέσεων, ενώ τα b'' , s'' είναι τα ακέραια samples των οποίων οι ενδιάμεσες τιμές είναι

μεγεθυμένες κατά 8 φορές. Το interpolation process για το quarter sample q είναι ίδιο με αυτό του f.

- **Quarter samples e, g, p, r**

$$e=(D''+j'+64)>>7$$

$$g=(E''+j'+64)>>7$$

$$p=(H''+j'+64)>>7$$

$$r=(I''+j'+64)>>7$$

όπου τα D'' , E'' , H'' , I'' είναι ακέραια samples των αντίστοιχων θέσεων, μεγεθυμένα κατά 64 φορές, και j' είναι η ενδιάμεση τιμή του half sample της αντίστοιχης θέσης.

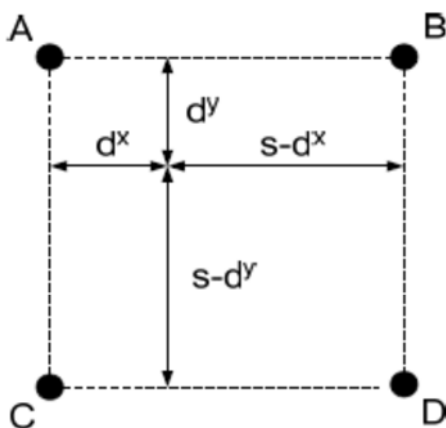
Interpolation process για τα chrominance samples

Το chrominance sample interpolation δεν είναι τόσο πολύπλοκο όσο το αντίστοιχο για το luminance, υπό την έννοια ότι δεν εμπεριέχει υποπεριπτώσεις, αλλά το interpolation είναι ενός είδους. Χρησιμοποιείται το motion vector του αντίστοιχου luminance block, το ονομάζουμε mnE για το παράδειγμά μας, με οριζόντια συνιστώσα το mnE_x και κάθετη το mnE_y .

Για να γίνει ευκολότερα κατανοητή η διαδικασία, την επεξηγούμε με τη βοήθεια της εικόνας 11 παρακάτω, όπου τα A, B, C, D είναι τα integral samples που περιβάλλουν τα samples προς interpolation. Τα dx και dy είναι η οριζόντια και κάθετη αντίστοιχα απόσταση μεταξύ του predictive sample και του A και ισούνται με $mnE_x \& 7$ και $mnE_y \& 7$, αντίστοιχα.

Το predictive sample matrix, έστω $predMatrix[y][x]$ υπολογίζεται ως εξής:

$$predMatrix[y][x]=((8-dx)x(8-dy)xA+dx \ x(8-dy)xB+(8-dx)xdyxC+dx \ x \ dyxD+32)>> 6$$



Εικόνα 11: Chroma sample interpolation

5. Υλοποίηση του hardware infrastructure

Εισαγωγή

Για την υλοποίηση του συνολικού συστήματος, είναι απαραίτητη η δημιουργία ενός hardware υποστρώματος, το οποίο θα δεχθεί στη συνέχεια, όπως θα δούμε, το λειτουργικό σύστημα, πάνω από το οποίο θα εκτελείται η εφαρμογή της αποκωδικοποίησης video. Όσον αφορά το hardware, στην παρούσα εργασία αναφερόμαστε σε hardware, τόσο με την τυπική έννοια (πχ hard core επεξεργαστής), όσο και με την έννοια του hardware δημιουργούμενου με τη μορφή επαναδιατασσόμενης λογικής (πχ soft core IPs, όπως το custom motion compensation peripheral που θα δημιουργήσουμε).

Μια σχεδίαση hardware, θα πρέπει να λαμβάνει υπόψη της τις ανάγκες της εφαρμογής την οποία καλείται να εκτελέσει, αν πρόκειται για κάποιο σύστημα ειδικού σκοπού (όπως στην περίπτωση μας), είτε να γίνεται μετά από σχετική μελέτη της μέσης περίπτωσης εκτέλεσης, αν πρόκειται για σύστημα γενικού σκοπού. Όλα αυτά, λαμβάνοντας υπόψη τις δυνατότητες, αλλά και τους περιορισμούς των διαθέσιμων λύσεων.

Αν αυτό είναι εφικτό, είναι απόλυτα επιθυμητό η σχεδίαση να περιλαμβάνει και επιπλέον δυνατότητες, αν αυτό δεν αντίκειται στους περιορισμούς χώρου ή ταχύτητας εκτέλεσης (ειδικά για συστήματα πραγματικού χρόνου). Έτσι, καθίσταται δυνατή η εύκολη επέκταση του συστήματος, χωρίς την ανάγκη για εκ βάθρων αλλαγές, οι οποίες θα είναι όχι μόνο πιο χρονοβόρες, αλλά ενδέχεται να κοστίζουν και σε χρήμα, όταν αυτές γίνονται σε ύστερα στάδια της δημιουργίας, μετά το αρχικό. Ως ενδεικτικό παράδειγμα, αναφέρουμε τη δυνατότητα δικτύωσης, η οποία αν και μη απαραίτητη για πολλές εφαρμογές (όπως για παράδειγμα στην αποκωδικοποίηση βίντεο), δεν αποκλείεται να βρει χρησιμότητα σε ένα επεκταμένο μελλοντικό μοντέλο.

Στο παράρτημα Β, παρουσιάζουμε αναλυτικές οδηγίες για τη δημιουργία της υποδομής hardware, την οποία θα χρησιμοποιήσουμε σαν βάση για τη συνέχεια της εργασίας.

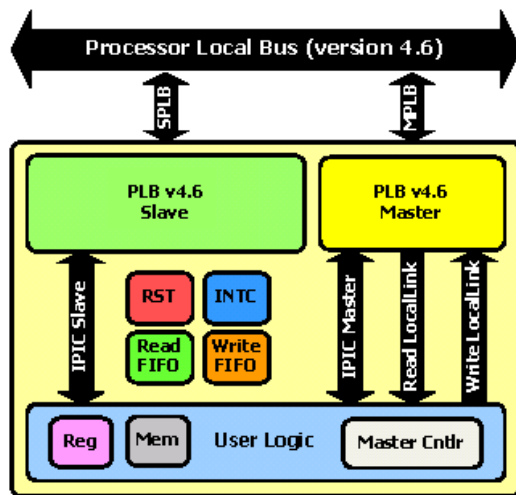
Στη συνέχεια, θεωρώντας δεδομένη την ύπαρξη του παραπάνω βασικού συστήματος, υπεισερχόμαστε στη δημιουργία του motion compensation accelerator, επεξηγώντας παράλληλα διάφορες χρήσιμες θεωρητικές έννοιες καθώς προχωράμε. Για το λόγο αυτό παρουσιάζουμε τη διαδικασία στο κύριο σώμα της εργασίας και όχι σε κάποιο ξεχωριστό παράρτημα.

Δημιουργία/προσθήκη του Motion Compensation accelerator

Για τη δημιουργία του motion compensation peripheral, ακολουθούμε τη μέθοδο που προσδιορίζει η Xilinx, μέσω του “Peripheral Wizard”. Μέσω αυτού του wizard, απλοποιείται σημαντικά η δημιουργία ενός custom peripheral, η οποία ολοκληρώνεται σε τρεις διακριτές φάσεις, όπως θα παρουσιάσουμε παρακάτω. Η πρώτη είναι η φάση του “Create”, η δεύτερη του “Modify” και τελευταία η φάση του “Import”, όπως θα τις ονομάσουμε. Σε κάθε βήμα, θα εξηγούμε, όπου χρειάζεται εκτός από το ποιες επιλογές κάνουμε, και γιατί τις κάνουμε, καθώς και το τι σημαίνουν αυτές.

A) Create Peripheral stage

- 1) Αφού ξεκινήσουμε τον οδηγό, από το μενού Hardware-> Create or import peripheral, πατάμε next και στη δεύτερη οθόνη που καλούμαστε να επιλέξουμε το Flow, διαλέγουμε “Create templates for a new peripheral”.
- 2) Επιλέγουμε ως χώρο αποθήκευσης, το τρέχον XPS project μας (σε περίπτωση δημιουργίας ενός custom peripheral γενικής χρήσης, καλό είναι να επιλέγουμε αποθήκευση σε ένα πιο generic user repository).
- 3) Δίνουμε ένα αναγνωριστικό όνομα και προαιρετική περιγραφή για το περιφερειακό μας.
- 4) Επιλέγουμε το bus interface, ανάλογα με το bus στο οποίο θα προσαρτηθεί το περιφερειακό μας. Στην περίπτωσή μας επιλέγουμε το PLB bus (Processor Local Bus). Το PLB bus interface, είναι η ιδανική επιλογή όταν το ζητούμενο είναι η υψηλή ταχύτητα και επιδόσεις, παρέχοντας εκτός των άλλων ξεχωριστούς διαύλους (32bit address και 64bit data buses, για τα instruction και data sides). Στην περίπτωσή μας το motion compensation peripheral θα συνδεθεί στο δίαυλο, ως slave peripheral.
- 5) Η επόμενη οθόνη, μας καλεί να επιλέξουμε τα IPIF services, τα οποία απαιτούνται από το περιφερειακό μας. Τα IPIF, είναι τα IP interface modules, τα οποία παρέχουν ένα γρήγορο και τυποποιημένο τρόπο υλοποίησης του interface ανάμεσα στο PLB



Εικόνα 12

interconnect και το user logic. Κάποια από τα τυπικά slave services, που χρησιμοποιούνται από τα περιφερειακά, είναι το “software reset”, “read/write FIFO”, “Interrupt control”, “user logic software register”, “user logic memory space”, “include data phase timer”. Στη δική μας περίπτωση, επιλέγουμε μόνο το “user logic software register” (θα δούμε παρακάτω τι ακριβώς σημαίνει αυτό).

6) Ακολουθώντας, κάνουμε κάποιες επιλογές όσον αφορά το slave interface του περιφερειακού μας. Η κυριότερη επιλογή αφορά την ενεργοποίηση ή μη, του burst-mode, το οποίο είναι μεγάλης σημασίας σε ότι έχει να κάνει με το performance. Συνεπώς, ενεργοποιούμε το Burst-mode, και αλλάζουμε την τιμή του native data bus width σε 64 bit, όσο δηλαδή και το width του data bus του PLB. Για το write-data buffer επιλέγουμε write buffer depth, 0.

7) Στην επόμενη οθόνη επιλέγουμε τον αριθμό των software accessible registers του περιφερειακού μας. Μέσω αυτών είναι εφικτή η αλληλεπίδραση του software με το περιφερειακό, υπό την έννοια ότι μπορούμε να αντιστοιχήσουμε σήματα εισόδου και εξόδου (σε χαμηλό επίπεδο) του περιφερειακού, σε registers, τους οποίους θα μπορούμε να γράφουμε και να διαβάζουμε, σαν να ήταν κανονικοί hardware registers. Όπως, όμως μαρτυρά και το όνομά τους, πρόκειται για μια software υλοποίηση-αφαίρεση. Οι S/W registers διευθυνσιοδοτούνται στα όρια byte, half-word, word, double-word, quad-word, ανάλογα με την εκάστοτε σχεδίαση. Μετά από καταγραφή των ports εισόδου και εξόδου του motion compensation module, όπως αυτά εμφανίζονται στη δήλωση του σε VHDL, επιλέγουμε τον αριθμό των S/W registers που απαιτούνται για την περίπτωσή μας, που ανέρχεται στον αριθμό των οκτώ.

8) Η επόμενη οθόνη έχει να κάνει με το λεγόμενο IP interconnect (IPIC). Όπως είδαμε νωρίτερα, το περιφερειακό μας συνδέεται με το PLB bus, μέσω κατάλληλου IPIF slave module. Η custom λογική, από την άλλη, από το user-logic module,

συνδέεται με το παραπάνω IPIF module, μέσω ενός συνόλου σημάτων, τα οποία αποτελούν το IPIF interface. Κάποια από αυτά τα σήματα είναι απαραίτητα (όπως για παράδειγμα το clock), ενώ άλλα επιλέγονται ανάλογα με την επιθυμητή λειτουργικότητα που θέλουμε να προσδώσουμε στο υπό υλοποίηση περιφερειακό. Στην περίπτωσή μας, δε χρειάζεται να προχωρήσουμε σε κάποια αλλαγή, μιας και ο wizard έχει προεπιλέξει τα απαραίτητα ports, με βάση τις επιλογές που κάναμε στα προηγούμενα βήματά του.

9) Ακολουθούν δυο προαιρετικές οθόνες επιλογών, εκ των οποίων προσπερνάμε την πρώτη. Στη δεύτερη ο wizard μας ενημερώνει πως με το τέλος του, θα παραχθούν τα απαραίτητα synthesizable HDL files που υλοποιούν τα IPIF services που έχουν επιλεγεί. Παράλληλα, θα παραχθεί και ένα stub “user-logic” module, το οποίο θα πρέπει να τροποποιήσουμε κατάλληλα, για την περίπτωσή μας (περισσότερες λεπτομέρειες θα παρουσιαστούν παρακάτω). Επιλέγουμε το 2^ο και το 3^ο checkbox, ώστε να παραχθούν ISE και XST project files, για την υλοποίηση του περιφερειακού με XST flow, και ώστε να παραχθούν template driver files, που θα μας βοηθήσουν να υλοποιήσουμε το software interface (χρήση των S/W registers).

10) Με την τελευταία οθόνη ολοκληρώνεται η δημιουργία του περιφερειακού μας, όμως πρέπει να γίνουν αρκετά ακόμη βήματα ώστε να καταστεί λειτουργικό. Αυτά περιγράφονται στην αμέσως επόμενη παράγραφο.

B. Modify stage

Σε αυτή τη φάση, θα τροποποιήσουμε τα template αρχεία που παρήγαγε ο wizard στην Α' φάση, ώστε να υλοποιήσουμε ουσιαστικά το συγκεκριμένο περιφερειακό που εκτελεί το motion compensation, καθώς η Α' φάση, αν και σαφώς επηρεάζεται από τις συγκεκριμένες απαιτήσεις του εκάστοτε περιφερειακού, πραγματοποιεί μια template υλοποίηση σε ένα υψηλότερο αφαιρετικό επίπεδο.

Δεδομένου ότι έχουμε ζητήσει από το wizard στην προηγούμενη φάση να παραγάγει ISE και XST project files, θα χρησιμοποιήσουμε το ISE για την απαραίτητη επεξεργασία αυτής της φάσης.

Τα σημεία στα οποία θα πρέπει να γίνουν οι αλλαγές για να προστεθεί η επιθυμητή λειτουργικότητα, βρίσκονται στο αρχείο user_logic.vhd.

Στην επόμενη εικόνα (εικόνα 13), βλέπουμε το σημείο έναρξης του implementation του user_logic, και τον ορισμό των σημάτων που αντιστοιχούν στους 8 software καταχωρητές, και κάποια άλλα βοηθητικά σήματα που έχουν να κάνουν με το IPIF και παράγονται από τον peripheral wizard και στο τελικό επίπεδο ανάγονται σε σύνδεση με το ίδιο το PLB bus και τον arbiter αυτού.

Εκτός, από τα αυτόματα παραγόμενα σήματα, προσθέτουμε τα custom σήματα, που απαιτούνται για τις εξόδους του module μας (εικόνα 14).

```

-----
-- Architecture section
-----

architecture IMP of user_logic is

    --USER signal declarations added here, as needed for user logic

    -----
    -- Signals for user logic slave model s/w accessible register
    -----

    signal slv_reg0      : std_logic_vector(0 to C_SLV_DWIDTH-1);
    signal slv_reg1      : std_logic_vector(0 to C_SLV_DWIDTH-1);
    signal slv_reg2      : std_logic_vector(0 to C_SLV_DWIDTH-1);
    signal slv_reg3      : std_logic_vector(0 to C_SLV_DWIDTH-1);
    signal slv_reg4      : std_logic_vector(0 to C_SLV_DWIDTH-1);
    signal slv_reg5      : std_logic_vector(0 to C_SLV_DWIDTH-1);
    signal slv_reg6      : std_logic_vector(0 to C_SLV_DWIDTH-1);
    signal slv_reg7      : std_logic_vector(0 to C_SLV_DWIDTH-1);
    signal slv_reg_write_sel : std_logic_vector(0 to 7);
    signal slv_reg_read_sel  : std_logic_vector(0 to 7);
    signal slv_ip2bus_data   : std_logic_vector(0 to C_SLV_DWIDTH-1);
    signal slv_read_ack      : std_logic;
    signal slv_write_ack     : std_logic;

```

Εικόνα 13

```

-----
--custom signals for outputs
-----

signal Stall_L      : std_logic;
signal Stall_C      : std_logic;
signal valid_Out     : std_logic;
signal C_Out         : std_logic;
signal L_Out         : std_logic;
signal L_state       : std_logic_vector(1 downto 0);
signal C_state       : std_logic_vector(3 downto 0);
signal PBO           : std_logic_vector(7 downto 0);
signal PB1           : std_logic_vector(7 downto 0);
signal PB2           : std_logic_vector(7 downto 0);
signal PB3           : std_logic_vector(7 downto 0);
signal PB4           : std_logic_vector(7 downto 0);
signal PB5           : std_logic_vector(7 downto 0);
signal PB6           : std_logic_vector(7 downto 0);
signal PB7           : std_logic_vector(7 downto 0);

```

Εικόνα 14

Μετά από τα σήματα αυτά, ορίζουμε και το component που αντιστοιχεί στο motion compensation module, και το οποίο θα χρησιμοποιήσουμε στην υλοποίηση της αρχιτεκτονικής (εικόνα 15). Αυτό το module υψηλότερου επιπέδου, είναι υπεύθυνο για την επίτευξη όλης της λειτουργικότητας του motion compensation, και είναι αυτό το οποίο καλεί εντός του ένα πλήθος άλλων components, που είναι απαραίτητα για τον τελικό υπολογισμό των motion compensated samples του βίντεο.

```

component avs_mc_accelerator
Port
(
    clk, reset          : in  std_logic;
    write_req, read_req : in  std_logic;
    LC, ref_res, AVG     : in  std_logic;
    mode_dims           : in  std_logic_vector(1 downto 0);
    dxdy                : in  std_logic_vector(3 downto 0);
    valid_bytes         : in  std_logic_vector(7 downto 0);
    RB0, RB1, RB2, RB3  : in  std_logic_vector(7 downto 0);
    RB4, RB5, RB6, RB7  : in  std_logic_vector(7 downto 0);
    Stall_L, Stall_C    : out std_logic;
    valid_Out           : out std_logic;
    C_Out, L_Out        : out std_logic;
    L_state             : out std_logic_vector(1 downto 0);
    C_state             : out std_logic_vector(3 downto 0);
    PB0, PB1, PB2, PB3  : out std_logic_vector(7 downto 0);
    PB4, PB5, PB6, PB7  : out std_logic_vector(7 downto 0)
);
end component;

```

Εικόνα 15

Παρακάτω, μετά το begin statement, οφείλουμε να προχωρήσουμε στο instantiation του component που προσθέσαμε (εικόνα 16). Αντιστοιχίζουμε δηλαδή (port mapping) τα σήματα εισόδου/εξόδου του motion compensation component, στα αντίστοιχα σήματα της τρέχουσας αρχιτεκτονικής. Εν προκειμένω, αντιστοιχίζουμε τα σήματα σε τμήματα των software registers, όπως φαίνεται στην εικόνα 16. Το σημείο στο οποίο θα έπρεπε να δώσουμε ίσως λίγο περισσότερη προσοχή, είναι το σήμα για το ρολόι (clk), και το σήμα valid_bytes, τα οποία αντιστοιχίζουμε στο σήμα ρολογιού που έρχεται από το bus (Bus2IP_Clk) και το Bus2IP_BE, αντίστοιχα.

Στον υπόλοιπο τμήμα του αρχείου για το user logic, έχει παραχθεί κώδικας που υλοποιεί το γράψιμο και το διάβασμα στους/από τους software registers, μέσω αντίστοιχων processes, και που είναι συμβατός με τον template driver που παράγεται επίσης από τον peripheral wizard.

Στην περίπτωση μας, τροποποιήσαμε το process που αφορά την ανάγνωση των καταχωρητών, όπου προσθέσαμε στο sensitivity list, τα αντίστοιχα signals που είχαμε κάνει port map στα σήματα εξόδου. Έτσι, όταν ζητούνται προς ανάγνωση οι καταχωρητές εξόδου, οι οποίοι είναι οι 4,5,6 και 7, επιστρέφονται στο slv_ip2bus_data τα αντίστοιχα signals concatenated, και με προσθήκη μηδενικών, όπου απαιτείται, για τη συμπλήρωση των 32 bits που εγγράφονται εν τέλει στο PLB bus (IP2Bus_Data), όταν το απαιτούμενο σήμα slv_read_ack έχει την τιμή 1.


```

--USER logic implementation
uut_avs_mc_accelerator_0: avs_mc_accelerator
port map
(
    clk=>Bus2IP_Clk,
    reset=>slv_reg1(31),
    write_req=>slv_reg0(23),
    read_req=>slv_reg0(31),
    LC=>slv_reg1(7),
    ref_res=>slv_reg1(15),
    AVG=>slv_reg1(23),
    mode_dims=>slv_reg0(14 to 15),
    dxdy=>slv_reg0(4 to 7),
    valid_bytes=>Bus2IP_BE,
    RB0=>slv_reg2(0 to 7),
    RB1=>slv_reg2(8 to 15),
    RB2=>slv_reg2(16 to 23),
    RB3=>slv_reg2(24 to 31),
    RB4=>slv_reg3(0 to 7),
    RB5=>slv_reg3(8 to 15),
    RB6=>slv_reg3(16 to 23),
    RB7=>slv_reg3(24 to 31),
    Stall_L=>slv_reg7(15),
    Stall_C=>slv_reg7(23),
    valid_Out=>slv_reg7(7),
    C_Out=>slv_reg6(7),
    L_Out=>slv_reg6(15),
    L_state=>slv_reg6(22 to 23),
    C_state=>slv_reg6(28 to 31),
    PB0=>slv_reg4(0 to 7),
    PB1=>slv_reg4(8 to 15),
    PB2=>slv_reg4(16 to 23),
    PB3=>slv_reg4(24 to 31),
    PB4=>slv_reg5(0 to 7),
    PB5=>slv_reg5(8 to 15),
    PB6=>slv_reg5(16 to 23),
    PB7=>slv_reg5(24 to 31)
);

```

Εικόνα 16

```

-- implement slave model software accessible register(s) read mux
SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1, slv_reg2, slv_reg3,
slv_reg4, slv_reg5, slv_reg6, slv_reg7, Stall_L, Stall_C, valid_Out, C_Out, L_Out,
L_state, C_state, PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7 ) is
begin

    case slv_reg_read_sel is
        when "10000000" => slv_ip2bus_data <= slv_reg0;
        when "01000000" => slv_ip2bus_data <= slv_reg1;
        when "00100000" => slv_ip2bus_data <= slv_reg2;
        when "00010000" => slv_ip2bus_data <= slv_reg3;
        when "00001000" => slv_ip2bus_data <= PB0 & PB1 & PB2 & PB3;
        when "00000100" => slv_ip2bus_data <= PB4 & PB5 & PB6 & PB7;
        when "00000010" => slv_ip2bus_data <= "00000000" & C_out & "000000000000"
            & L_out & "000000000000" & L_state & "0000" & C_state;
        when "00000001" => slv_ip2bus_data <= "00000000" & valid_out & "00000000"
            & Stall_L & "00000000" & Stall_C & "00000000";
        when others => slv_ip2bus_data <= (others => '0');
    end case;

end process SLAVE_REG_READ_PROC;

```

Εικόνα 17

Εννοείται, ότι θα πρέπει να ενημερώσουμε το ISE με όλα τα αρχεία της σχεδίασης μας, τα οποία περιέχουν τα entities όλων των components του top-level motion compensation module, τα entities των components των sub-components κ.ο.κ.

Σε αυτό το σημείο, μπορούμε να κάνουμε synthesis της σχεδίασης και να διορθώσουμε τυχόν λάθη που έχουν διαφύγει.

Γ) Import Stage

Με αυτό το τρίτο στάδιο, ολοκληρώνεται η σχεδίαση και η εισαγωγή του motion compensation custom περιφερειακού μας. Η διαδικασία που χρειάζεται να ακολουθηθεί είναι τετριμμένη και δεν απαιτεί κάποιες ιδιαίτερες επιλογές από την πλευρά μας, θα παρουσιαστεί όμως για λόγους πληρότητας.

1) Αφού εκκινήσουμε το XPS, επιλέγουμε από το μενού “Hardware” τον οδηγό δημιουργίας περιφερειακού “Create or Import Peripheral”, όπου στη δεύτερη οθόνη, σε αντίθεση με την Α’ φάση, επιλέγουμε “Import existing peripheral”.

2) Στην επόμενη οθόνη, επιλέγουμε το Project το οποίο είχαμε δημιουργήσει κατά την Α’ φάση δημιουργίας του περιφερειακού.

3) Ακολούθως, επιλέγουμε από το drop-down menu, το όνομα του top VHDL entity του περιφερειακού μας και τα ίδια στοιχεία για το version που είχαμε εισάγει στην Α’ φάση, πατώντας στη συνέχεια Next, και απαντώντας θετικά στην προειδοποίηση για overwrite του υπάρχοντος περιφερειακού.

4) Στην οθόνη που ακολουθεί, με τίτλο “Source File Types”, επιλέγουμε τον τύπο των αρχείων που αποτελούν το περιφερειακό μας, τα οποία στην περίπτωση μας είναι μόνο HDL source files (συγκεκριμένα .vhd).

5) Ακολούθως στην οθόνη “HDL Source Files”, επιλέγουμε το “Use data (*.mpd) collected during a previous invocation of this tool”, και κάνουμε browse για το αρχείο mpd που παρήχθη κατά την Α’ φάση, και το οποίο βρίσκεται στο φάκελο “project_name\rcores\ονομα_περιφερειακού\data\”. Για τον εντοπισμό των HDL αρχείων που απαρτίζουν τη σχεδίασή μας, καθώς και των εξαρτώμενων βιβλιοθηκών, επιλέγουμε την επιλογή “Use an XST project file”, όπου και οδηγούμε το εργαλείο στο αρχείο .prj που παρήχθη κατά την πρώτη εκτέλεση του οδηγού, και το οποίο βρίσκεται στο φάκελο “project_name\rcores\ονομα_περιφερειακού\dev\projnav\”.

6) Η οθόνη που εμφανίζεται στη συνέχεια (“HDL Analysis Information”) δείχνει όλα τα αρχεία HDL που απαρτίζουν το περιφερειακό μας. Αφού βεβαιωθούμε ότι δεν έχει υπάρξει κάποια ασυνέπεια, προχωράμε στις υπόλοιπες οθόνες επιβεβαιώνοντας τις default επιλογές, και ολοκληρώνοντας τη δημιουργία του περιφερειακού μας.

Ενσωμάτωση του Motion Compensation Peripheral στο υπάρχον hardware σύστημά

Πλέον, έχοντας έτοιμο το motion compensation IP, δε μένει παρά να το ενσωματώσουμε στο σύστημα που έχουμε φτιάξει. Η διαδικασία είναι πολύ απλή.

Το νέο IP που δημιουργήσαμε θα είναι προσβάσιμο μέσω του XPS, στο tab “IP Catalog” στο υπομενού “Project local Pcores”. Από εκεί, μπορούμε να το προσθέσουμε στη σχεδιάσή μας και να το συνδέσουμε ως slave στο PLBv46 (επιλέγοντας “plb” στο drop-down menu, για το tag “SPLB”).

Αυτό που απομένει να γίνει, είναι να υπολογίσουμε εκ νέου το memory map, ώστε αυτό να περιλαμβάνει και το άρτι προστεθέν περιφερειακό. Από το tab “Addresses” πραγματοποιούμε, λοιπόν, “Generate addresses”, και δημιουργείται με αυτόματο τρόπο το memory map του συστήματός μας.

Κατά τα γνωστά πλέον, προχωρούμε στο implementation της σχεδιάσής μας. Πλέον επόμενο βήμα είναι η δημιουργία ενός driver, ο οποίος θα έχει ως σκοπό την αλληλεπίδραση του λογισμικού με το τμήμα του hardware που υλοποιεί το περιφερειακό μας. Το ζήτημα αυτό θα καλυφθεί στην επόμενη ενότητα.

Δημιουργία Driver του motion compensation peripheral

Με το τέλος του προηγούμενου βήματος, έχουμε στη διάθεση μας ένα σύστημα, το οποίο περιλαμβάνει το motion compensation peripheral. Για να επικοινωνήσουμε όμως με αυτό το τμήμα του hardware, θα χρειαστούμε τον αντίστοιχο driver, ο οποίος προσφέρει τη λειτουργικότητα, την οποία θα εκμεταλλεύεται το software το οποίο θα γράψει ο προγραμματιστής.

Ο peripheral wizard του XPS, με βάση τις επιλογές που κάνουμε, δημιουργεί έναν sample driver με κάποιες βασικές λειτουργίες ανάγνωσης και εγγραφής των software καταχωρητών. Στη συνέχεια, με βάση αυτές τις βασικές λειτουργίες, θα δημιουργήσουμε έναν πιο περίπλοκο driver, ο οποίος θα γράφει και θα διαβάζει συγκεκριμένα τμήματα, τα οποία αντιστοιχούν στα διάφορα σήματα εισόδου και εξόδου του περιφερειακού. Οι συναρτήσεις αυτές ουσιαστικά θα είναι το interface του προγράμματος μας (του AVS decoder) με το hardware, υλοποιώντας το μεγάλο αυτό πλεονέκτημα του software-hardware codesign.

Στη συνέχεια, θα εξηγήσουμε περιληπτικά τα σημαντικότερα σημεία των αρχείων που παράγονται αυτόματα από τον peripheral wizard, και ακολούθως θα εξετάσουμε από σημασιολογικής έννοιας ποιες είναι οι συναρτήσεις οι οποίες θα πρέπει να παρέχονται σε ένα προγραμματιστή που θα χρησιμοποιήσει το custom περιφερειακό μας. Τέλος, θα παρουσιάσουμε κάποια ενδεικτικά τμήματα αυτών των συναρτήσεων.

Βασικές μακροεντολές εγγραφής/ανάγνωσης καταχωρητών

Οι βασικές εντολές εγγραφής και ανάγνωσης, παράγονται ως function-like defines στο .h αρχείο που δημιουργείται από τον peripheral wizard (inline functions δηλαδή). Στα πλαίσια της παροχής της αναγκαίας λειτουργικότητας στο αρχείο αυτό εμφανίζονται κάποιες σταθερές (constants) και συναρτήσεις, οι οποίες είναι άγνωστες στο μη εξοικωμένο αναγνώστη, επομένως είναι σκόπιμη μια σύντομη περιγραφή, απαραίτητη για την κατανόηση του επεκταμένου driver που σχεδιάζουμε, και αναγκαία σε όποιον επιθυμεί να επεκτείνει περαιτέρω μελλοντικά τη λειτουργικότητα του motion compensation peripheral.

```
#define MOTION_COMP_PER_USER_SLV_SPACE_OFFSET (0x00000000)
#define MOTION_COMP_PER_SLV_REG0_OFFSET (MOTION_COMP_PER_USER_SLV_SPACE_OFFSET + 0x00000000)
#define MOTION_COMP_PER_SLV_REG1_OFFSET (MOTION_COMP_PER_USER_SLV_SPACE_OFFSET + 0x00000004)
#define MOTION_COMP_PER_SLV_REG2_OFFSET (MOTION_COMP_PER_USER_SLV_SPACE_OFFSET + 0x00000008)
#define MOTION_COMP_PER_SLV_REG3_OFFSET (MOTION_COMP_PER_USER_SLV_SPACE_OFFSET + 0x0000000C)
```

Εικόνα 18

Στην εικόνα 18, βλέπουμε τον τρόπο με τον οποίο ορίζεται το offset του κάθε s/w register. Να υπενθυμίσουμε στον αναγνώστη, πως οι καταχωρητές αυτοί είναι “software”, επομένως αντιστοιχούν σε κάποια διεύθυνση της μνήμης, η οποία έχει αποδοθεί στο εν λόγω περιφερειακό. Για αποφυγή πλεονασμού, απεικονίζουμε μόνο τους τέσσερις πρώτους καταχωρητές. Παρατηρούμε, ότι ο κάθε καταχωρητής απέχει από τον επόμενό του 4 bytes, ήτοι 32 bits, όπως και είχε καθοριστεί κατά τη δημιουργία του περιφερειακού.

Η πρώτη από τις δυο βασικές ενέργειες που εκτελούνται πάνω σε αυτούς τους καταχωρητές είναι η εγγραφή, η οποία υλοποιείται επίσης ως inline function και είναι η εξής:

```
#define MOTION_COMP_PER_mWriteReg(BaseAddress, RegOffset, Data) \  
    XIo_Out32((BaseAddress) + (RegOffset), (Xuint32)(Data))
```

Όπου “BaseAddress”, η διεύθυνση βάσης του περιφερειακού, “RegOffset” η απόσταση του καταχωρητή από τη διεύθυνση βάσης και “Data” τα δεδομένα (32 bits) τα οποία επιθυμούμε να εγγράψουμε στον καταχωρητή.

Παρατηρούμε τη συνάρτηση XIo_Out32, καθώς και το casting σε τύπο δεδομένων Xuint32. Πρόκειται για συναρτήσεις και τύπους δεδομένων, τα οποία ορίζονται στα αρχεία “xio.h” και “xbasic_types.h”, που γίνονται include. Το πρώτο εκ των δυο καθορίζει το interface για το XIo component, το οποίο εγκλείει τις συναρτήσεις εισόδου/εξόδου για την αρχιτεκτονική του PowerPC. Θα πρέπει να τονιστεί, πως τα στοιχεία αυτού του αρχείου είναι architecture-dependent. Το δεύτερο, περιλαμβάνει τον ορισμό διαφόρων τύπων δεδομένων για Xilinx software IP. Στην ουσία πρόκειται για διάφορα typedefs στους γνωστούς τύπους δεδομένων.

Με παρόμοιο τρόπο, ορίζεται και η συνάρτηση για την ανάγνωση καταχωρητή:

```
#define MOTION_COMP_PER_mReadReg(BaseAddress, RegOffset) \  
    XIo_In32((BaseAddress) + (RegOffset))
```

Η κύρια διαφορά έγκειται στη χρήση της συνάρτησης XIo_In32, η οποία διαβάσει 32 bits δεδομένων από τη ζητούμενη θέση.

Στη συνέχεια, μιας και η υλοποίησή μας αντιμετωπίζει τα περιεχόμενα των καταχωρητών ως τιμές 8 bits, δηλαδή θεωρούμε ότι κάθε καταχωρητής εμπεριέχει 4 διακριτές εννοιολογικά τιμές, γράφουμε παρόμοια inline functions που με οδηγό τα παραπάνω, υλοποιούν τη ζητούμενη εγγραφή/ανάγνωση σε 8-μπιτα τμήματα των καταχωρητών. Ενδεικτικές τέτοιες συναρτήσεις είναι οι παρακάτω (εγγραφή και ανάγνωση αντίστοιχα):

```

// Write to an 8bit part of register 0
#define MOTION_COMP_PER_mWriteSlaveReg0_8(BaseAddress, RegOffset, Value) \
    XIo_Out8((BaseAddress) + (MOTION_COMP_PER_SLV_REGO_OFFSET) + (RegOffset), (Xuint8)(Value))

// Read from an 8bit part of register 0
#define MOTION_COMP_PER_mReadSlaveReg0_8(BaseAddress, RegOffset) \
    XIo_In8((BaseAddress) + (MOTION_COMP_PER_SLV_REGO_OFFSET) + (RegOffset))

```

Όπως γίνεται αντιληπτό, αρχίζουμε και εξειδικεύουμε, όσον αφορά τη λειτουργικότητα των συναρτήσεων, όμως ακόμη δεν έχουμε φτάσει στο επίπεδο αφαίρεσης το οποίο πρέπει να αντιλαμβάνεται ο προγραμματιστής εφαρμογών. Εδώ, θα πρέπει να κάνουμε μια μικρή παρένθεση, αναφέροντας ότι πολλοί θεωρούν ότι ο driver θα πρέπει να παρέχει χαμηλού επιπέδου πρόσβαση στο περιφερειακό, και λειτουργίες υψηλότερου επιπέδου να υλοποιούνται κατά βούληση από τον προγραμματιστή εφαρμογών. Εν πάση περιπτώσει, εμείς προχωρούμε υλοποιώντας συναρτήσεις που γράφουν και διαβάζουν συγκεκριμένα (σημασιολογικά) πεδία, με αντίστοιχα περιγραφικά ονόματα. Για λόγους επεξήγησης, θα παρουσιάσουμε τις αντίστοιχες συναρτήσεις, που αφορούν τα πεδία του πρώτου καταχωρητή (εικόνες 20 και 21).

Για λόγους πληρότητας, να αναφέρουμε ότι η τιμή BaseAddress, είναι γνωστή κατά τη σχεδίαση του συστήματος, στη διαδικασία δημιουργίας του memory map, και αναφέρεται στο αρχείο “xparameters.h” ως “XPAR_MOTION_COMP_PER_0_BASEADDR”

```

// -----MACROS FOR SETTING THE VALUES OF THE FIRST SW REGISTER-----
// set the dx dy value
#define setDXDY(BaseAddress, dx, dy) \
    XIo_Out8((BaseAddress) + (MOTION_COMP_PER_SLV_REGO_OFFSET), (Xuint8)((dx<<2)+dy))
// set the ModeDims value
#define setModeDims(BaseAddress, mode_dims) \
    XIo_Out8((BaseAddress) + (MOTION_COMP_PER_SLV_REGO_OFFSET)+(1), (Xuint8)(mode_dims))
// set the WriteReq value
#define setWriteReq(BaseAddress, write_req) \
    XIo_Out8((BaseAddress) + (MOTION_COMP_PER_SLV_REGO_OFFSET)+(2), (Xuint8)(write_req))
// set the ReadReq value
#define setReadReq(BaseAddress, read_req) \
    XIo_Out8((BaseAddress) + (MOTION_COMP_PER_SLV_REGO_OFFSET)+(3), (Xuint8)(read_req))
//-----

```

Εικόνα 20

```

// -----MACROS FOR READING THE VALUES OF THE FIRST SW REGISTER -----
// set the dxdy value
#define readDXY(BaseAddress) \
    xil_printf("    - mode_dims:%d\n\r", MOTION_COMP_PER_mReadSlaveReg0_8(BaseAddress,0));
// set the ModeDims value
#define readModeDims(BaseAddress) \
    xil_printf("    - mode_dims:%d\n\r", MOTION_COMP_PER_mReadSlaveReg0_8(BaseAddress,1));
// set the WriteReq value
#define readWriteReq(BaseAddress) \
    xil_printf("    - write_req:%d\n\r", MOTION_COMP_PER_mReadSlaveReg0_8(BaseAddress,2));
// set the ReadReq value
#define readReadReq(BaseAddress) \
    xil_printf("    - read_req: %d\n\r", MOTION_COMP_PER_mReadSlaveReg0_8(BaseAddress,3));
//-----

```

Εικόνα 21

Η ορθότητα των παραπάνω, καθώς και των υπολοίπων συναρτήσεων ελέγχθηκε αρχικά σε ένα dummy peripheral, ένα περιφερειακό δηλαδή χωρίς κάποια λειτουργικότητα. Στο περιφερειακό αυτό αναθέσαμε οκτώ καταχωρητές, όπου έγινε σύγκριση των τιμών που εγγράφονταν και των τιμών που διαβάζονταν, ώστε να επιβεβαιώσουμε ότι οποιοδήποτε πρόβλημα εμφανιστεί σε επόμενο στάδιο δεν οφείλεται στις βασικές αυτές συναρτήσεις, αλλά στο ίδιο το περιφερειακό. Ο έλεγχος πραγματοποιήθηκε σύμφωνα με τα πρότυπα των παρακάτω ενδεικτικών τμημάτων κώδικα (εικόνες 22 και 23).

Δε θα μπορούμε σε λεπτομέρειες υλοποίησης software, μέσω του XPS, καθώς θεωρούμε ότι ο αναγνώστης είναι σχετικά εξοικειωμένος, και δεδομένου ότι δεν παρουσιάζει κάποια ιδιαίτερη δυσκολία. Αρκεί η δημιουργία του C αρχείου με τον κώδικα ελέγχου, και η δημιουργία του H header file, τα οποία καλό είναι να τοποθετούνται στους αντίστοιχους καταλόγους (sources & headers). Τέλος, είναι αναγκαία η δημιουργία ενός linker script (μενού “Software”, επιλογή “Generate new linker script” και επιλογή της εφαρμογής μας), και φυσικά το compilation για τη δημιουργία του αρχείου ELF.

```

// Now we will test reading and writing to each of the 4 8bit areas within a register
// |BYTE0|BYTE1|BYTE2|BYTE3| <-32 bit software register
xil_printf("    Now going to try and read/write to 8bit areas within the registers\n\n\r");
xil_printf("    - write 15 to slave register 0 byte 0\n\r");
MOTION_COMP_DUMMY_mWriteSlaveReg0_8(baseaddr, 0, 15);
Reg8Value = MOTION_COMP_DUMMY_mReadSlaveReg0_8(baseaddr, 0);
xil_printf("    - read %d from register 0 byte 0\n\r", Reg8Value);
if ( Reg8Value != (Xuint8) 15 )
{
    xil_printf("    - slave register 0 byte 0 write/read failed\n\r");
    return XST_FAILURE;
}

```

Εικόνα 22

```
// Now we will test the macros for writing and reading to the parts of the first register
xil_printf("    - write 2 to dx and 3 for dy (i.e. 1011(binary)=11(decimal))\n\r");
setDXY(baseaddr, 2, 3);
readDXY(baseaddr);

xil_printf("    - write 2 for mode_dims\n\r");
setModeDims(baseaddr, 2);
readModeDims(baseaddr);

xil_printf("    - write 1 for write_req\n\r");
setWriteReq(baseaddr, 1);
readWriteReq(baseaddr);

xil_printf("    - write 0 for read_req\n\r");
setReadReq(baseaddr, 0);
readReadReq(baseaddr);
```

Εικόνα 23

6. Υλοποίηση του Software Infrastructure

Εισαγωγή

Το κεφάλαιο αυτό, αφορά θέματα software που αφορούν το σύστημά μας. Συγκεκριμένα, θα εξετάσουμε τι επιλογές έχουμε όσον αφορά στο λειτουργικό σύστημα, τα πλεονεκτήματα και τα μειονεκτήματα της καθεμιάς, και τέλος θα καταλήξουμε στο λειτουργικό σύστημα που θα χρησιμοποιήσουμε. Στο παράρτημα Γ, υπάρχουν αναλυτικές οδηγίες για τη δημιουργία της απαραίτητης υποδομής development (δημιουργία cross development toolchain σε ένα host μηχανήμα), το building του λειτουργικού συστήματος (Linux), το κατέβασμα της σχεδίασης και του kernel image στο FPGA board, ζητήματα του root filesystem, και τέλος τη δημιουργία ενός ενιαίου ACE file, που καθιστά δυνατή την εκτέλεση του συνολικού συστήματος από κάρτα CompactFlash.

Απόφαση για χρησιμότητα και επιλογή λειτουργικού συστήματος

Θεωρήσαμε απαραίτητη την ύπαρξη κάποιου, υποτυπώδους έστω πυρήνα, ώστε να αποφύγουμε τα προβλήματα και την πολυπλοκότητα που θα δημιουργούσε η ανάγκη συγγραφής standalone κώδικα. Έτσι, με ένα λειτουργικό σύστημα (και ειδικά με τη λύση του Linux που επιλέξαμε, όπως θα εξηγήσουμε παρακάτω), αρκεί η συγγραφή κώδικα σε ένα υψηλότερο αφαιρετικό επίπεδο, αντί για κώδικα χαμηλότερου επιπέδου ή κώδικα επιπέδου micro-controller, που θα χρειαζόταν σε διαφορετική περίπτωση. Εκτός αυτού, το σημαντικότερο, μιας και ο κώδικας του video decoder μας, όπως και οι περισσότερες εφαρμογές, βασίζονται σε υπηρεσίες όπως file systems, time management, network management κ.α., η ύπαρξη ενός λειτουργικού συστήματος που παρέχει αυτές τις υπηρεσίες ελαττώνει εκθετικά την προσπάθεια που θα απαιτούνταν δίχως αυτό.

Υπήρξε αρχικά η σκέψη να χρησιμοποιηθεί το Xilkernel, το οποίο παρέχεται από τη Xilinx. Πρόκειται για ένα μικρό, modular kernel, με αρκετές δυνατότητες παραμετροποίησης, επιτρέποντας στους χρήστες του να το φέρουν στα μέτρα τους, τόσο όσον αφορά στο μέγεθος, όσο και στη λειτουργικότητα. Υποστηρίζει το Microblaze, αλλά και τον PowerPC, που είναι η δική μας περίπτωση και παρέχει δυνατότητες όπως POSIX threads, POSIX synchronization services (semaphores και mutex locks), POSIX IPC services (message queues και shared memory), software timers και άλλα, ενώ είναι integrated στο Platform Studio της Xilinx και συνεργάζεται με το Embedded Development Kit (EDK) της ίδιας εταιρείας. Αν και σε

γενικές γραμμές, ένα πρόγραμμα που γράφεται για το Xilkernel είναι συμβατό (λόγω POSIX interface) με λειτουργικά όπως το Linux ή το SunOS σε ένα συμβατικό PC, η αντίστροφη διαδικασία μπορεί να αποτελέσει πηγή προβλημάτων, λόγω της ιδιαίτερης φύσης των ενσωματωμένων συστημάτων, τα οποία καλείται να «συντονίσει» το Xilkernel. Αν και επιχειρήθηκε η μεταφορά του AVS decoder πάνω από το Xilkernel, γρήγορα έγινε αντιληπτό ότι η προσπάθεια που θα απαιτούνταν για τη μετατροπή του κώδικα, όπως ασυμβατότητες μεταξύ των LibXil standard C libraries και των τυπικών βιβλιοθηκών της C, το LibXil FAT file system (για την πρόσβαση στο SystemACE compact flash device κ.α. καθιστούσαν ασύμφορη αυτή την επιλογή.

Η άλλη επιλογή που είχαμε ήταν να χρησιμοποιήσουμε το γνωστό και μη εξαιρετέο Linux, με όλα τα πλεονεκτήματα τα οποία αυτό έχει. Τελικά, επιλέξαμε να υλοποιήσουμε ένα σύστημα, το οποίο θα βασίζεται στο Linux.

Οι λόγοι που μας οδήγησαν σε αυτή την επιλογή είναι οι παρακάτω:

Ο κώδικας του Linux kernel, έχει πλέον πολλά χρόνια στην πλάτη του και οι περισσότερες «παιδικές αρρώστιες» των πρώτων χρόνων έχουν γιατρευτεί. Πρόκειται, λοιπόν, για έναν σταθερό kernel, ο οποίος προσφέρει όλα εκείνα τα χαρακτηριστικά που προσδίδουν ποιότητα και αξιοπιστία, όπως αναμενόμενη συμπεριφορά στα πλαίσια ενός καλώς καθορισμένου framework, καλή υποδομή ανάνηψης από σφάλματα, η οποία να περιλαμβάνει τουλάχιστον έγκαιρη ειδοποίηση του system administrator με διαγνωστικά μηνύματα, μακροβιότητα και δυνατότητα λειτουργίας χωρίς επίβλεψη για μακρά χρονικά διαστήματα. Χαρακτηριστικά, δηλαδή, ιδανικά για μια embedded λύση, όπως στην περίπτωση μας.

Επιπλέον, για έναν γνώστη των internals του Linux kernel, η ευκολία διόρθωσης τυχόν λαθών είναι τετριμμένη, ειδικά αν συνυπολογίσει κανείς το επαρκές documentation και την ύπαρξη εμπειρίας.

Η δυνατότητα πλήρους παραμετροποίησης και επέκτασης των δυνατοτήτων του kernel, είναι επίσης ένα απόλυτα θεμιτό χαρακτηριστικό, ειδικά στον χώρο των ενσωματωμένων συστημάτων, όπου η high-performance φύση και τα προβλήματα σχετικά με την κατανάλωση ενέργειας είναι αρκετά κρίσιμα.

Πρόκειται για ένα πολυδοκιμασμένο λειτουργικό σύστημα, με μεγάλη προϊστορία, η οποία χαρακτηρίζεται με θετικό πρόσημο. Το open source development model και γενικά η open source φύση και λογική των υποστηρικτών του, έχει πολλά πλεονεκτήματα, σε σχέση με το proprietary software διαφόρων εταιρειών. Το κυριότερο αυτών είναι αναμφισβήτητα η δυνατότητα διάδοσης της γνώσης, της εμπειρίας και η βοήθεια στην επίλυση προβλημάτων. Για την επίλυση ενός

προβλήματος σε άλλες περιπτώσεις, θα έπρεπε να στηριχθεί κανείς στο support της εταιρείας, να χαθεί σημαντικός χρόνος και επιπλέον να πληρωθούν μεγάλα – συνήθως- χρηματικά ποσά. Αντίθετα, στην open source community, στα διάφορα fora και στις mailing lists, μπορεί κανείς να βρει άμεση βοήθεια, από άτομα που έχουν ήδη αντιμετωπίσει το πρόβλημα και που έχουν βρει λύση, ή στις περισσότερες περιπτώσεις μπορεί να μιλήσει με τον ίδιο τον άνθρωπο που έγραψε τον προβληματικό κώδικα και να ζητήσει κάποιο patch.

Επιπλέον, αν και το αφήσαμε για το τέλος, η ίδια (αυτονόμητη για το Linux) δυνατότητα του να έχεις πρόσβαση στον κώδικα, δε θεωρείται δεδομένη για άλλα λειτουργικά συστήματα, και ειδικά για παραδοσιακά embedded OSes, όπου ο κώδικας είναι κλειστός και θα πρέπει να πληρώσει κανείς μεγάλα χρηματικά ποσά για να αποκτήσει πρόσβαση σε αυτόν και μάλιστα υπό το καθεστώς αυστηρότατων licenses. Η δυνατότητα αυτή όμως είναι πολύ σημαντική, αφού ο developer μπορεί αφενός να διορθώσει μόνος του τυχόν προβλήματα, χωρίς να χρειαστεί να απευθυνθεί στον κατασκευαστή, και αφετέρου μπορεί να δει και να κατανοήσει καλύτερα τις εσωτερικές λειτουργίες του λειτουργικού συστήματος, και να τις εκμεταλλευτεί στο έπακρο.

Στα πλεονεκτήματα του Linux, ως λειτουργικού γενικά, αλλά και συγκεκριμένα για την περίπτωσή μας, περιλαμβάνεται το σημαντικό hardware support, τόσο σε ότι αφορά τις αρχιτεκτονικές γενικότερα, όσο και συγκεκριμένες πλατφόρμες και συσκευές. Πέραν των device drivers, οι οποίοι υποστηρίζονται από απλούς προγραμματιστές όταν τυχόν η εταιρεία σταματήσει μια σειρά συσκευών και την επίσημη υποστήριξη, το σημαντικότερο είναι ότι το Linux υποστηρίζει πάρα πολλές αρχιτεκτονικές. Σε αυτές δε μπορεί παρά να περιλαμβάνεται και η Power architecture, που ακολουθεί ο PowerPC που περιλαμβάνει το Virtex 5 evaluation board που χρησιμοποιούμε για την εργασία αυτή. Έτσι, καθίσταται εύκολο το porting του κώδικα που έχουμε ήδη γράψει για IA32 architecture, χωρίς ιδιαίτερα προσκόμματα στην όλη διαδικασία.

Τα tools που είναι απαραίτητα για software development, debugging, profiling, καθώς και η πληθώρα των βοηθητικών εργαλείων που συνοδεύουν κάθε τυπικό Linux distribution, είναι κοινά και συνεπώς δεν απαιτείται χρόνος για την εγκατάσταση, configuration και πολύ περισσότερο για την εκμάθηση νέων εργαλείων.

Τέλος, αν και στην περίπτωση μας, στο video decoding, δεν είναι τόσο εμφανές αυτό το πλεονέκτημα, το Linux υποστηρίζει μια πληθώρα από communication protocols, και ακολουθεί όλα τα καθιερωμένα software standards. Έτσι, σε αντίθεση με παραδοσιακά ενσωματωμένα λειτουργικά συστήματα, είναι πολύ πιο εύκολες διαδικασίες όπως το networking της ενσωματωμένης μας πλατφόρμας με άλλες πλατφόρμες και λειτουργικά (πχ Linux-Windows με χρήση Samba).

7. Μετρήσεις απόδοσης-αξιολόγηση

Το profiling έγινε με τη χρήση του εργαλείου GNU profiler (gprof) και τη χρήση ενδεικτικού απαιτητικού αρχείου βίντεο (calendar), το οποίο ενσωματώνει στοιχεία δύσκολα στην αποκωδικοποίηση, όπως κινούμενα γράμματα και αριθμούς. Η ανάλυση του είναι 576p (δηλαδή 720x576 pixels, progressive) και αποτελείται από 247 frames. Για τη δημιουργία του εκτελέσιμου χρησιμοποιήθηκε ο gcc compiler, με ενεργοποιημένες τις βελτιστοποιήσεις `-O3` και `-ffast-math`.

Αρχικά, η συλλογή δεδομένων του profiler, έδωσε τα αποτελέσματα της εικόνας 24. Η ανάλυση των κυριότερων τμημάτων του decoder φαίνεται στον πίνακα 1. Παρατηρούμε, πως ένα ποσοστό της τάξης του 9.56% δαπανάται σε συναρτήσεις που αφορούν δεδομένα κινητής υποδιαστολής και ένα ποσοστό του 5.44% σε πράξεις memory. Ο συνολικός χρόνος εκτέλεσης ήταν 77.35sec, το οποίο μεταφραζόμενο σε frames per second αντιστοιχεί σε 3.19fps. Το ποσοστό επί του χρόνου εκτέλεσης στις πράξεις δεδομένων κινητής υποδιαστολής, είναι δικαιολογημένο, αν λάβουμε υπόψη μας πως δεν έχουμε στη διάθεση μας floating point unit. Έτσι, τα αποτελέσματα των εν λόγω πράξεων προκύπτουν από τη διαδικασία του software emulation, η οποία είναι υπολογιστικά χρονοβόρα. Πέραν αυτού, όμως, ο συνολικός χρόνος εκτέλεσης, και η μετρική των frames per second, είναι πολύ μικρότερη εν συγκρίσει με την αντίστοιχη εκτέλεση σε x86 αρχιτεκτονική. Το ζήτημα εδώ, πέραν φυσικά της τεράστιας διαφοράς στη συχνότητα λειτουργίας, είναι η έλλειψη cache, του συστήματός μας. Το caching, και ειδικά σε τέτοιες εφαρμογές, όπως ένας video decoder, είναι ιδιάζουσας σημασίας, λόγω του μεγάλου χωρικού και χρονικού correlation των δεδομένων που προσπελούνται. Έτσι, αντιλαμβάνεται κανείς το μεγάλο πλεονέκτημα ενός Intel i7 (όπου έχουνε πραγματοποιηθεί αντίστοιχες μετρήσεις), τη στιγμή που αυτός διαθέτει multilevel data caches, με πολύ καλά στοιχεία απόδοσης (σε σχέση με παλαιότερα cores). Πιο συγκεκριμένα, κάθε πυρήνας διαθέτει δικές του L1 instruction και data caches μεγέθους 32KB η καθεμιά, μια L2 instruction & data cache μεγέθους 256KB. Τέλος, υπάρχει μια τεράστια L3 unified cache, η οποία είναι κοινή για όλους τους πυρήνες, και η οποία ακολουθεί μια inclusive cache policy για την ελάττωση της κίνησης εξαιτίας των snoops.

Δεδομένου ότι υπήρχε η δυνατότητα να προστεθεί hardware για floating point arithmetic, μέσω του FPU, όπου δημιουργείται το αντίστοιχο hardware στο FPGA fabric και συνδέεται στον PowerPC μέσω του FPU, υπήρξε αρχικά η σκέψη να χρησιμοποιηθεί. Όμως, η σκέψη αυτή εγκαταλείφθηκε μεμιάς, αφού οι video decoders, είναι εξ ορισμού εφαρμογές οι οποίες δεν απαιτούν floating point

arithmetic, οπότε ήταν βέβαιο, πως τα όποια floating point operations υπήρχαν στον κώδικα, θα μπορούσαν να εξαλειφθούν/αντικατασταθούν με fixed point arithmetic ή με lookup tables. Το κυριότερο παράδειγμα πράξης floating point, η οποία και καταλάμβανε το μεγαλύτερο ποσοστό από αυτό που αναλογεί στις fp operations, ήταν η power. Οι σχεδιαστές του reference κώδικα του AVS, μη έχοντας υπόψη τους high performance ζητήματα (άλλωστε δεν είναι αυτό το άμεσο ζητούμενο του reference code ενός video standard), χρησιμοποιούσαν τη συνάρτηση για ύψωση σε δύναμη. Η συνάρτηση αυτή, μπορούσε κάλλιστα να αντικατασταθεί, είτε με τον επαναληπτικό αλγόριθμο σταθερής υποδιαστολής, είτε με bitwise operations (αριστερή ολίσθηση), όπως και έγινε. Ο χρόνος εκτέλεσης πλέον μειώθηκε στα 69.25sec (3.57fps), δηλαδή κατά ένα ποσοστό της τάξης του 10% μειωμένος. Τα αποτελέσματα του profiling φαίνονται στην εικόνα 25, όπου και είναι εμφανής η σχεδόν πλήρης εξάλειψη των floating point operations, ενώ στον πίνακα 2 γίνεται μια ομαδοποίηση των ποσοστών ανά computation kernel.

| | |
|----------------------------|--------|
| Inverse Transform | 10.84% |
| Deblocking | 6.04% |
| Bitstream Parsing | 7.45% |
| Luma Motion Compensation | 21.44% |
| Chroma Motion Compensation | 12.71% |
| Floating point operations | 9.56% |
| Initialization/Other | 31.96% |

Πίνακας 1

| | |
|----------------------------|--------|
| Inverse Transform | 13.84% |
| Deblocking | 3.49% |
| Bitstream Parsing | 6.07% |
| Luma Motion Compensation | 33.04 |
| Chroma Motion Compensation | 14.73% |
| Initialization/Other | 28.83% |

Πίνακας 2

8. Προοπτικές επέκτασης/εξέλιξης

Δεδομένης της φύσης ενός FPGA board, οι δυνατότητες που υπάρχουν για πειραματισμό και επεκτάσεις είναι πάρα πολλές. Παλαιότερα, ο σχεδιασμός του hardware και του software αποτελούσαν διακριτές διαδικασίες, με μικρή ή και καθόλου αλληλεπίδραση μεταξύ τους. Η εισαγωγή και εφαρμογή της έννοιας του hardware/software co-design είναι μια νεότερη τάση η οποία προσφέρει μεγάλη ευελιξία, προσαρμογή σε συγκεκριμένες ανάγκες και σχεδιαστικούς στόχους, με άμεση συνεπαγωγή καλύτερες επιδόσεις, είτε αυτές μετρώνται στα πλαίσια της ταχύτητας εκτέλεσης, είτε της θερμικής/ενεργειακής απόδοσης, είτε με κάποια άλλη παράμετρο.

Έτσι, λοιπόν, ένας σχεδιαστής ενός συστήματος σε FPGA, όπως στην περίπτωση μας, έχει το προνόμιο να μπορεί να προχωρά στη σχεδίαση του υλικού και του λογισμικού ταυτόχρονα, και να προσαρμόζει το ένα στο άλλο, με απώτερο σκοπό την υψηλή επίδοση, η οποία στην περίπτωση του AVS video decoder αφορά στο χρόνο εκτέλεσης της διαδικασίας της αποκωδικοποίησης video.

Hardware Accelerators/ Co-processors

Πολύπλοκες υπολογιστικές διαδικασίες δύνανται να απεμπλακούν από το κομμάτι του λογισμικού και να υλοποιηθούν στο hardware, όπου η ταχύτητα εκτέλεσης είναι πολύ μεγαλύτερη. Αυτή η απεμπλοκή αφορά στην αλλαγή του κώδικα λογισμικού και τη «μεταφορά» της εκτέλεσης από τον επεξεργαστή γενικού σκοπού (τον PowerPC στην περίπτωση μας), σε ένα hardware accelerator/co-processor με συγκεκριμένη λειτουργικότητα. Ως γνωστόν, όσο πιο συγκεκριμένη είναι η λειτουργικότητα ενός κομματιού υλικού, τόσο πιο αποδοτικά θα μπορεί αυτή να επιτελείται, μιας και σε hardware γενικού σκοπού, είναι πολλά τα ζητήματα για τα οποία θα πρέπει να αποφασιστεί μια μέση λύση από το σχεδιαστή.

Στα πλαίσια της παρούσας εργασίας, παράδειγμα του παραπάνω αποτελεί η ενσωμάτωση του motion compensation accelerator. Δεν είναι όμως το μοναδικό module του AVS decoder, το οποίο θα μπορούσε να υλοποιηθεί σε hardware. Ενδεικτικά, σαν επέκταση του συστήματος, θα μπορούσε να υλοποιηθεί και να ενοποιηθεί με το παρόν σύστημα ένας hardware accelerator για τη διαδικασία του inverse transform, η οποία είναι υπολογιστικά πολύπλοκη, όπως φανερώνει και η αρχική διαδικασία του profiling, και ένας για τη διαδικασία του deblocking.

Φυσικά, σε κάθε περίπτωση, πριν από κάθε τέτοια απόφαση σχεδιασμού, θα πρέπει να ληφθεί υπόψη η χρησιμότητα σε σχέση με το κόστος υλοποίησης (σε

χρόνο/προσπάθεια). Έτσι, σε κάθε βήμα βελτιστοποίησης είναι αναγκαίο το profiling της εκτέλεσης, για τον εντοπισμό των χρονοβόρων εκείνων τμημάτων του κώδικα, των οποίων το off-loading σε hardware θα ωφελούσε τη συνολική σχεδιάσή μας.

Multiprocessor Designs

Ένα σημαντικό βήμα το οποίο θα έδινε μεγάλη ώθηση στη σχεδίαση μας, θα ήταν η προσθήκη ενός ή και περισσότερων ακόμα επεξεργαστών. Ένα σύστημα πολυεπεξεργαστών είναι πρακτικά χρήσιμο σε συγκεκριμένες εφαρμογές και περιπτώσεις. Θα αναφέρουμε με συντομία τις γενικές αυτές κατηγορίες, στη συνέχεια θα εξετάσουμε το κατά πόσο ένα σύστημα πολυεπεξεργαστών θα μπορούσε να χρησιμοποιηθεί και πόσο θα ωφελούσε, στην περίπτωση του AVS video decoder και τέλος θα συζητήσουμε τις γενικές αρχές σχεδίασης και τα προβλήματα τέτοιων συστημάτων.

Προφανώς, η πρώτη περίπτωση στην οποία ένα σύστημα πολυεπεξεργαστών είναι χρήσιμο, είναι αυτή στην οποία η σχεδίασή αποτελείται από πολλαπλές και ανεξάρτητες μεταξύ τους λειτουργίες. Έτσι, η ανάθεση καθεμιάς από αυτές σε ξεχωριστούς επεξεργαστές με ξεχωριστούς πόρους, αναμφισβήτητα θα οδηγούσε σε αύξηση των επιδόσεων.

Μια δεύτερη περίπτωση, η οποία είναι συνηθισμένη στον τομέα των ενσωματωμένων συστημάτων, είναι η περίπτωση μιας σχεδίασης η οποία περιλαμβάνει λειτουργίες με και χωρίς real-time απαιτήσεις. Αν χρησιμοποιείται ένας μοναδικός επεξεργαστής και για τις δύο, η εκπλήρωση των real-time απαιτήσεων καθίσταται δύσκολη και μη εγγυημένη. Έτσι, η ανάθεση των λειτουργιών με απαιτήσεις πραγματικού χρόνου σε κάποιο ξεχωριστό επεξεργαστή, μπορεί –υπό προϋποθέσεις- να εγγυηθεί την εκπλήρωση όλων των απαιτήσεων απόδοσης.

Η τρίτη και πολύ συνηθισμένη ακόμη και σε συστήματα γενικού σκοπού περίπτωση, αφορά εφαρμογές με μεγάλο παραλληλισμό σε ότι αφορά τα δεδομένα. Σε αυτή την περίπτωση, ένας κύριος επεξεργαστής αναλαμβάνει το συνολικό co-ordination, τυχόν αρχικοποιήσεις και την ανάθεση σε ένα σύνολο επεξεργαστών των επιμέρους εργασιών. Παραδείγματα τέτοιων εφαρμογών αποτελούν αλγόριθμοι ασφάλειας, streaming media processing κ.α.

Η περίπτωσή μας, δηλαδή η αποκωδικοποίηση video, θα λέγαμε ότι έχει στοιχεία και από τις τρεις αυτές κατηγορίες:

- Το bitstream parsing, δηλαδή η ανάγνωση του encoded αρχείου σε συνδυασμό με το variable length decoding (VLD), είναι μια διαδικασία η οποία μπορεί να απεμπλακεί τελείως από τους υπόλοιπους υπολογισμούς

και διαδικασίες του αποκωδικοποιητή. Βέβαια, η γενικότερη σειριακή φύση της αποκωδικοποίησης video, δε μας δίνει τη δυνατότητα της πλήρους απεμπλοκής, αλλά παρόλα αυτά δεν παύει να μας δίνει τη δυνατότητα εξόρυξης ενός επιπλέον επιπέδου παραλληλισμού.

- Αν και μη απαραίτητο, είναι άκρως επιθυμητό για την αποκωδικοποίηση βίντεο να γίνεται σε συνθήκες πραγματικού χρόνου, το οποίο μεταφράζεται σε αποκωδικοποίηση ενός ελάχιστου αριθμού frames ανά δευτερόλεπτο (25-30 fps). Η απαίτηση αυτή δύσκολα εκπληρούται, δίχως την υιοθέτηση επιπλέον τεχνικών, όπως η χρήση επιπλέον επεξεργαστών.
- Η αποκωδικοποίηση βίντεο, είναι εν γένει μια διαδικασία με έμφυτο παραλληλισμό. Έχοντας ήδη υλοποιήσει μια multithreaded έκδοση του AVS video decoder, όπου το data partitioning γίνεται δυναμικά σε επίπεδο macroblock, θα μπορούσαμε να μεταφέρουμε τον κώδικα αυτόν, με τις κατάλληλες μετατροπές, σε ένα FPGA board, με πολλαπλούς πυρήνες (Microblaze, PowerPC, ή συνδυασμό αυτών). Με περισσότερες λεπτομέρειες θα αναφερθούμε σε ξεχωριστή ενότητα.

Όπως αναφέραμε παραπάνω, μια σχεδίαση με χρήση πολυεπεξεργαστών, μπορεί να περιλαμβάνει είτε ομογενείς, είτε ετερογενείς επεξεργαστές. Μπορούμε να εκμεταλλευτούμε, δηλαδή, δυο PowerPC, που περιλαμβάνονται σε κάποια μοντέλα της σειράς Virtex5, ή δύο (ή και παραπάνω, ανάλογα με τις απαιτήσεις σε χώρο στην FPGA) Microblaze επεξεργαστές. Είναι, όμως εφικτός και ο συνδυασμός και των δυο σε μια ενιαία σχεδίαση.

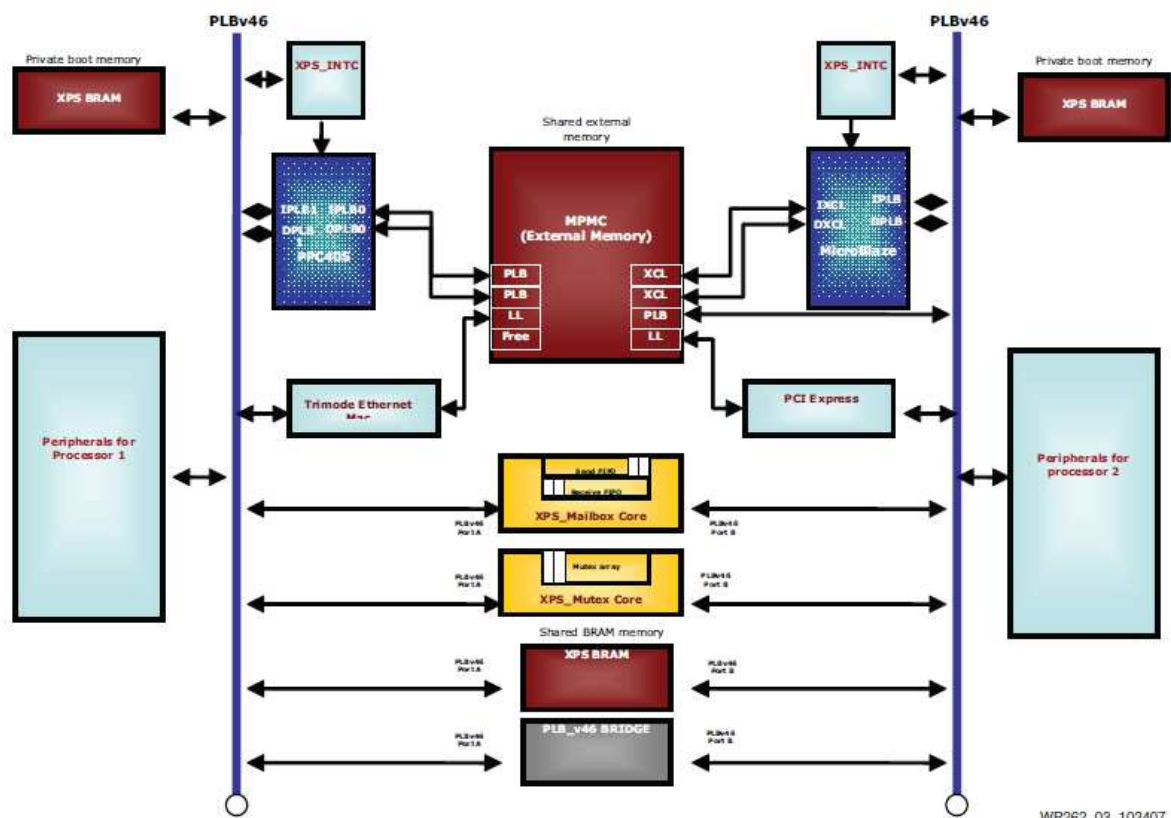
Σαν υπόδειγμα, θα εξετάσουμε ένα σύστημα δυο επεξεργαστών, ενός PowerPC και ενός Microblaze (εικόνα 26).

Παρατηρούμε στα μπλε πλαίσια τους δυο επεξεργαστές, καθένας από τους οποίους συνδέεται στο δικό του PLB bus, από όπου επικοινωνεί με τα περιφερειακά του, καθώς και με την ατομική του μνήμη. Κάθε επεξεργαστής έχει, επίσης, έναν interrupt controller, ώστε να διαχειρίζεται τα διάφορα interrupts του συστήματος. Είναι εμφανής η συμμετρική φύση του συστήματος μας, ενώ στη βάση του σχήματος παρατηρούμε ένα PLB to PLB bridge, το οποίο συνδέει το δεύτερο επεξεργαστή με το system bus του πρώτου, πράγμα το οποίο απαιτείται για το διαμοιρασμό περιφερειακών τα οποία δεν είναι εγγενώς multi-ported (πχ UART, SPI, I2C peripherals). Τα multi-ported peripherals, από την πλευρά τους, επιτρέπουν σε κάθε PLB system bus να είναι ανεξάρτητο από το άλλο. Αυτή η ιδιότητα αποτρέπει το κλείδωμα του bus για έναν επεξεργαστή ή περιφερειακό του, τη στιγμή που πραγματοποιείται κάποιο transaction στον άλλον. Τα multi-ported peripherals, υλοποιούν το απαραίτητο arbitration εσωτερικά.

Όσον αφορά την κοινή μνήμη, αυτή είναι δυο τύπων:

- Εξωτερική κοινή μνήμη DDR, στην οποία οι δυο επεξεργαστές έχουν πρόσβαση μέσω ενός MPMC memory controller. Αυτό είναι ένας ειδικός memory controller που παρέχεται από το Platform Studio και παρέχει έναν αριθμό διαφορετικών interfaces στην ίδια κοινή εξωτερική μνήμη, με τρόπο που να επιτυγχάνει πολύ μικρό latency, παράλληλα με πολύ μεγάλο bandwidth.
- Κοινή εσωτερική BRAM μνήμη, η οποία είναι περιορισμένου μεγέθους, όμως αντίστοιχα γρήγορη και αποδοτική για τη μεταφορά μικρού μεγέθους δεδομένων μεταξύ των επεξεργαστών.

Τέλος, αξίζει να κάνουμε μια αναφορά στα δυο cores που εμφανίζονται με κίτρινο χρώμα στην εικόνα 26, το XPS_Mailbox_core και το XPS_Mutex_core. Πρόκειται για δυο cores, τα οποία χρησιμοποιούνται για να παρέχουν υπηρεσίες συγχρονισμού και επικοινωνία μέσω μηνυμάτων.



Εικόνα 26: Μια αρχιτεκτονική δυο επεξεργαστών

Προτού κλείσουμε την αναφορά στα συστήματα πολυεπεξεργαστών, θα πρέπει να αναφέρουμε το ζήτημα του memory coherency, το οποίο θα πρέπει να ληφθεί υπόψη από μελλοντικούς σχεδιαστές της επέκτασης του συστήματός μας. Αν η περιοχή κοινής μνήμης είναι cacheable από τους επεξεργαστές του συστήματός, θα πρέπει ο ίδιος ο χρήστης να μεριμνήσει ώστε να εφαρμόζεται το αντίστοιχο coherency model, καθώς ούτε ο Microblaze, αλλά ούτε και ο PowerPC παρέχουν hardware υλοποίηση cache coherency. Υπάρχουν διάφορες τεχνικές για επίλυση αυτού του προβλήματος.

Δε θα επεκταθούμε περισσότερο όσον αφορά τα ζητήματα πολυεπεξεργαστών, στα πλαίσια της παρούσης εργασίας, αλλά παραπέμπουμε τους ενδιαφερόμενους στην παραπομπή [27], για αναλυτικότερη παρουσίαση ζητημάτων interprocessor communication και synchronization στα πλαίσια του hardware, communication και synchronization στα πλαίσια του software design, στην παραπομπή [28] για κάποια reference designs της Xilinx που αφορούν πολυεπεξεργαστικές σχεδιάσεις και ζητήματα και στην παραπομπή [26] για μια ενδεικτική υλοποίηση coherent shared memories για FPGAs.

9. Επίλογος/Σύνοψη

Η παρούσα εργασία ξεκίνησε με κάποια εισαγωγικά σχόλια όσον αφορά τη χρησιμότητα για αποδοτικό video decoding και προχώρησε παρουσιάζοντας τον σκοπό αυτής. Στη συνέχεια, αναφερθήκαμε στις τεχνολογίες επαναδιατασσόμενης λογικής και εξετάσαμε τη συγκεκριμένη πλατφόρμα ανάπτυξης που χρησιμοποιήθηκε για τους σκοπούς της εργασίας, της Virtex-5 ML507, καθώς και του επεξεργαστή PowerPC 440, που αυτή φέρει. Συζητήθηκε επίσης το ζήτημα του endianness, το οποίο είναι συχνά πηγή προβλημάτων στη διαδικασία του porting software από μια αρχιτεκτονική σε άλλη, και εξηγήθηκε η διαδικασία του porting αυτή καθ' αυτή. Παρουσιάστηκαν μετρήσεις από την εκτέλεση του AVS, τόσο πριν, όσο και μετά από μια σειρά αλλαγών και έγινε μια αποτίμηση και αξιολόγηση των αποτελεσμάτων. Ακολουθώντας, ασχοληθήκαμε με το θεωρητικό υπόβαθρο του video decoding γενικά, όσο και του AVS και του motion compensation procedure πιο συγκεκριμένα. Υλοποιήσαμε το κατάλληλο hardware design για την εκτέλεση του video decoder σε Linux, ενώ παράλληλα υλοποιήθηκε ένα motion compensation custom peripheral και ο αντίστοιχος driver αυτού, τα οποία ελέγχθηκαν σε standalone μορφή, έξω από τα πλαίσια του AVS video decoder. Όσον αφορά το software infrastructure, αρχικά έγινε μια αξιολόγηση των διαθέσιμων επιλογών λειτουργικών συστημάτων, και αφού επιλέχθηκε η λύση του Linux, προχωρήσαμε στην υλοποίηση αυτής της επιλογής, με τη δημιουργία του κατάλληλου περιβάλλοντος προγραμματισμού και το building του kernel (με βάση και το hardware design που είχαμε σχεδιάσει). Τέλος, το hardware και το software κομμάτι, ενσωματώθηκαν σε ένα ενιαίο ace αρχείο, ώστε να είναι δυνατή η εκτέλεση του συστήματος από κάρτα CompactFlash, αφού αυτή εισαχθεί στο FPGA board. Κλείνοντας την παρούσα εργασία, εξετάζουμε τις προοπτικές εξέλιξης και επέκτασης της σχεδίασης, με σκοπό την επίτευξη μεγαλύτερων επιδόσεων, στα πλαίσια ενός πολυεπεξεργαστικού συστήματος με παράλληλη χρήση custom accelerators.

Βιβλιογραφία

- [1] *H264 and MPEG-4 Video Compression*, Iain E. G. Richardson, Wiley
- [2] *Building Embedded Linux Systems*, Karim Yaghmour, O'Reilly
- [3] *Embedded Linux Primer: a practical real-world approach*, Christopher Hallinan, Prentice Hall
- [4] *Linux Device Drivers*, Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, O'Reilly
- [5] *Video Compression Wikipedia article:*
http://en.wikipedia.org/wiki/Video_compression
- [6] *AVS Wikipedia article:* http://en.wikipedia.org/wiki/Audio_Video_Standard
- [7] *Royalty fees on video codecs:*
<http://www.eetimes.com/story/OEG20020524S0091>
- [8] *Mapping and optimization of the AVS video decoder on a high performance chip multiprocessor*, Konstantinos Krommydas, George Tsoublekas, Christos Antonopoulos, Nikolaos Bellas, IEEE ICME 2010
- [9] *Implementantion and performance comparison of the motion compensation kernel of the AVS video decoder on FPGA, Cell and multi-core processors*, Muhsen Owaida, Christos D. Antonopoulos, Nikolaos Bellas, Konstantis Daloukas, Konstantinos Krommydas, Georgios Tsoumplekas
- [10] *ELDK toolchain:* <http://www.denx.de/wiki/DULG/Manual>
- [11] *Linux on Xilinx FPGA project:* <http://xilinx.wikidot.com/>
- [12] *Xilinx XAPP1107 Getting Started Using Git*, Application Note:
http://www.xilinx.com/support/documentation/application_notes/xapp1107.pdf
- [13] *Xilinx forum on creating partitions and loading system from CF card:*
<http://forums.xilinx.com/xlnx/board/message?message.uid=39019>
- [14] *OSL System ACE driver (on generating ACE file & building kernel):*
<http://xilinx.wikidot.com/osl-sysace-dricer>

- [15] *Xilinx XAP969* (section: Generating an ACE File and booting Linux on ML405), Xilinx:
http://www.xilinx.com/support/documentation/application_notes/xapp969.pdf
- [16] *Busybox*: <http://www.busybox.com>
- [17] *AVS Video Standard*
- [18] *FPGA Wikipedia article*: http://en.wikipedia.org/wiki/Field-programmable_gate_array
- [19] *Xilkernel*, Xilinx:
http://www.xilinx.com/ise/embedded/edk91i_docs/xilkernel_v3_00_a.pdf
- [20] *Endianness White Paper*, Intel, 2004:
<http://www.intel.com/design/intarch/papers/endian.pdf>
- [21] *Virtex-5 Family overview (ds100)*, Xilinx:
http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf
- [22] *ML505/506/507 Evaluation Platform User Guide (ug347)*, Xilinx:
http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf
- [23] *Embedded processor block in Virtex-5 FPGAs (ug200)*, Xilinx:
http://www.xilinx.com/support/documentation/user_guides/ug200.pdf
- [24] *Virtex-5 FXT PowerPC 440 and Microblaze Edition Kit reference systems (ug511)*, Xilinx:
http://www.xilinx.com/support/documentation/boards_and_kits/ug511.pdf
- [25] *Motion Compensation Wikipedia article*:
http://en.wikipedia.org/wiki/Motion_compensation
- [26] *Coherent Shared Memories for FPGAs*, David Woods, Master Thesis, University of Toronto:
https://tspace.library.utoronto.ca/bitstream/1807/19001/1/Woods_David_E_20091_1_MASc_thesis.pdf
- [27] *Designing multiprocessor systems in Platform Studio (WP262)*, Xilinx:
http://www.xilinx.com/support/documentation/white_papers/wp262.pdf
- [28] *Dual Processor reference design suite, Application note (XAPP996)*, Xilinx:
http://www.xilinx.com/support/documentation/application_notes/xapp996.pdf
- [29] *Processor Local Bus (PLBv4.6)*, Xilinx:
http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf

[30] *EDK peripheral wizard tutorial:*

www.cosmiac.org/pdfs/EDK_Peripheral_Wizard_Tutorial.pdf

[31] *Xilinx Forum on solving S/W register issues:* <http://forums.xilinx.com/t5/EDK-and-Platform-Studio/about-custom-ip-with-S-W-accessible-registers-cannot-get-data/m-p/64909>

[32] *Intel i7 Nehalem caches:* <http://www.tomshardware.com/reviews/Intel-i7-nehalem-cpu,2041-10.html>

Σημείωση: Οι εικόνες των σελίδων 11,12,13,29,51,62 έχουν ληφθεί από τους οδηγούς της Xilinx.

Παράρτημα Α

Γλωσσάρι κυριότερων εννοιών video coding/decoding

Backward prediction: η πρόβλεψη με βάση επόμενη reference picture (με βάση το display order)

Bidirection prediction: η πρόβλεψη με βάση περασμένες και επόμενες reference pictures (με βάση το display order)

Bitstream: stream δυαδικών δεδομένων, το οποίο προέρχεται από τη διαδικασία του encoding

Compensation: η διαδικασία πρόσθεσης των υπολοίπων που προέρχονται από το decoding των syntax elements του bitstream και των αντίστοιχων predictive τιμών

Decoding order: η σειρά του decoding όπως αυτή γίνεται με βάση τις σχέσεις prediction ανάμεσα στις εικόνες του βίντεο και η οποία μπορεί να είναι διαφορετική από το Display order

Display order: η σειρά απεικόνισης των decoded pictures, όπως προκύπτει από ενδεχόμενο reordering

Forward Inter-coded picture (P frame): picture η οποία έχει γίνει encode με forward prediction

Forward prediction: η πρόβλεψη με βάση προηγούμενη reference picture (με βάση το display order)

Inter-frame coding: κωδικοποίηση ενός macroblock ή εικόνας με χρήση inter-frame prediction

Inter-frame prediction: η διαδικασία δημιουργίας των sample predictive values της τρέχουσας εικόνας με βάση προηγούμενως αποκωδικοποιηθείσας εικόνας

Intra encoded picture (I-frame): εικόνα η οποία έχει κωδικοποιηθεί μόνο με intra-frame coding

Intra-frame coding: κωδικοποίηση ενός macroblock ή εικόνας με χρήση intra-frame prediction

Intra-frame prediction: η διαδικασία παραγωγής των predictive values του τρέχοντος sample, με βάση προηγούμενως αποκωδικοποιηθέντα sample values της ίδιας εικόνας

Luminance: αναφέρεται στον πίνακα sample data ενός μοναδικού σήματος χρώματος, απεικονισμένο ως μείξη των βασικών χρωμάτων ή ένα μοναδικό sample data. Συμβολίζεται με Y

Macroblock: λογική αφαίρεση που περιλαμβάνει ένα block 16x16 δειγμάτων luminance, καθώς και το αντίστοιχο chrominance sample data block

Motion vector: δισδιάστατο διάνυσμα που χρησιμοποιείται στο inter-frame prediction, και που παρέχει ένα offset από τις συντεταγμένες στο decoded picture, στις συντεταγμένες του reference frame

Output order: ισοδύναμο με το Display order

Parsing process: η διαδικασία λήψης των syntax elements από το bitstream

Picture reordering: η διαδικασία αλλαγής σειράς των decoded pictures, στην περίπτωση που το decoding order είναι διαφορετικό από το display order

Raster scan: η διαπέραση και το mapping ενός δισδιάστατου ορθογώνιου χώρου σε μονοδιάστατο, του οποίου η αρχή ξεκινά από την πρώτη γραμμή του αρχικού δισδιάστατου προς την τελευταία και από αριστερά προς τα δεξιά, γραμμή προς γραμμή

Reference pictures: εικόνες οι οποίες χρησιμοποιούνται ως εικόνες αναφοράς για την αποκωδικοποίηση προηγούμενων ή επόμενων εικόνων με βάση τη διαδικασία του inter-frame prediction

Residual: το υπόλοιπο της αφαίρεσης της τιμής του reconstructed syntax element από την προβλεφθείσα τιμή (predicted value)

Sample value: η τιμή του πλάτους του δείγματος

Variable length coding: αναστρέψιμη διαδικασία κωδικοποίησης εντροπίας, η οποία αποδίδει μικρότερες κωδικές λέξεις σε σύμβολα με μεγάλη συχνότητα εμφάνισης και μεγαλύτερες κωδικές λέξεις σε σύμβολα με χαμηλή συχνότητα εμφάνισης, με σκοπό την επίτευξη καλύτερης συμπίεσης

Video sequence: η υψηλότερη συντακτική δομή του encoded bitstream, η οποία περιέχει μια ή περισσότερες encoded pictures

Παράρτημα Β

Αναλυτικές οδηγίες δημιουργίας hardware infrastructure

Σε αυτό το κεφάλαιο, θα προχωρήσουμε στην υλοποίηση του βασικού συστήματος, πάνω στο οποίο θα τρέξει στη συνέχεια το λειτουργικό σύστημα και ο AVS decoder. Θα ξεκινήσουμε με ένα σύστημα στο οποίο θα τρέξουμε αρχικά τον decoder, θα πραγματοποιήσουμε μετρήσεις, και εν συνεχεία, αφού δημιουργήσουμε το custom motion compensation peripheral, θα δημιουργήσουμε τον κατάλληλο driver και θα επαληθεύσουμε τη λειτουργία του, ώστε να είναι εφικτή η προσθήκη αυτού στο σύστημα μας.

Υλοποίηση του βασικού συστήματος

Για την υλοποίηση του βασικού συστήματος, θα χρησιμοποιήσουμε τον wizard που παρέχει η Xilinx στο XPS, ο οποίος ονομάζεται Base System Builder (BSB) και θα παράγουμε ένα dts (device tree source) αρχείο, το οποίο θα χρησιμοποιήσουμε κατά τη διαδικασία του building του λειτουργικού.

Δε θα μπούμε σε πολλές λεπτομέρειες, σε ότι αφορά τη χρήση του BSB, μιας και τα περισσότερα είναι αυτοεξηγούμενα, όμως θα εστιάζουμε στα σημεία που θεωρούμε πως είναι απαραίτητο.

Το σύστημα μας, βασίζεται στον PowerPC 440, ο οποίος τρέχει στα 400Mhz. Το memory interface block του επεξεργαστή συνδέεται στο PPC440MC DDR2 memory controller και η συχνότητα λειτουργίας του ορίζεται στα 200Mhz. Στο MPLB port του processor block συνδέουμε το PLB v4.6 bus, πάνω στο οποίο θα συνδεθούνε ως slaves τα διάφορα περιφερειακά που θα προσαρτήσουμε. Αναφέρουμε, επίσης, πως θα συνδέσουμε το LocalLink connection του XPS_LL_TEMAC core στο Hard DMA device του processor block, από όπου στη συνέχεια, μέσω του Linux που θα εγκαταστήσουμε, θα έχουμε tri-mode Ethernet MAC, ανάλογα με το δίκτυο που θα συνδεθούμε.

Αφού επιλέξουμε να ανοίξουμε το BSB, δώσουμε το όνομα και σώσουμε τη σχεδιάσή μας ως .bsb αρχείο, επιλέγουμε τη δημιουργία νέας σχεδίασης και φθάνουμε στην οθόνη επιλογής board, όπου κάνουμε τις επιλογές που φαίνονται στην εικόνα ΠΒ.1:

Base System Builder

Welcome **Board** System Processor Peripheral Cache Application Summary

Board Selection

Select a target development board.

Board

☒ I would like to create a system for the following development board

Board Vendor: Xilinx

Board Name: Virtex 5 ML507 Evaluation Platform

Board Revision: A

☐ I would like to create a system for a custom board

Board Information

Architecture: virtex5 Device: xc5vfx70t Package: ff1136 Speed Grade: -1

☐ Use Stepping

Reset Polarity: Active Low

Related Information

[Vendor's Website](#)

[Vendor's Contact Information](#)

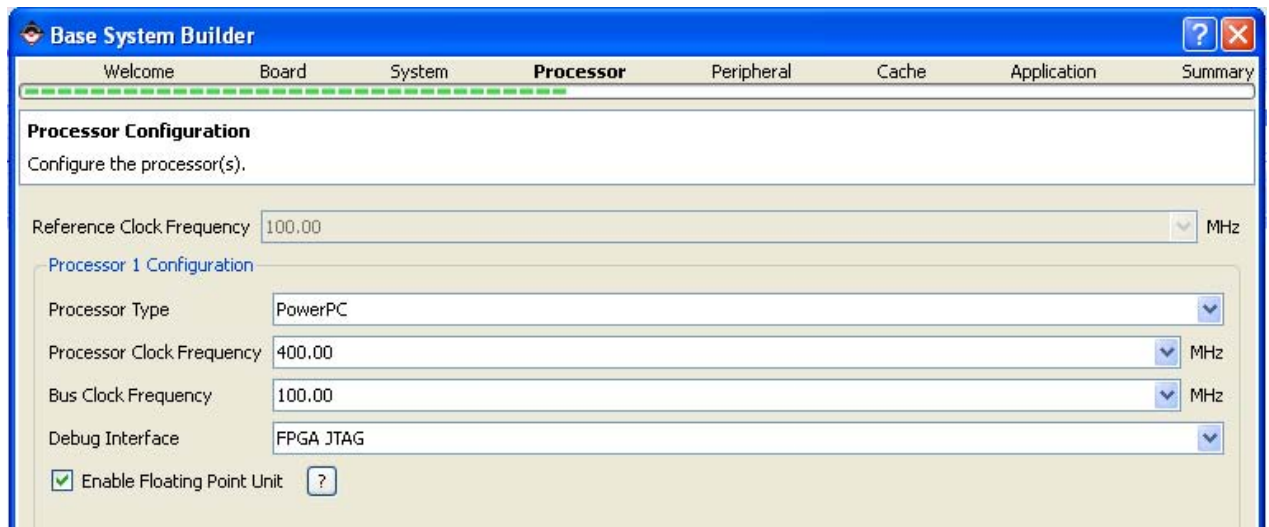
[Third Party Board Definition Files Download Website](#)

The ML507 board is intended to showcase and demonstrate Virtex-5 technology. The ML507 board utilizes Xilinx Virtex 5 XC5VFX70T-FF1136 device. The board includes Tri-Mode Ethernet MAC/PHY, 256MB DDR2 SDRAM SODIMM memory, 1MB ZBT SRAM, 32MB of Commodity Flash, 8kb IIC EEPROM, CPU Debug and CPU Trace connectors, System ACE CF controller and 2 RS232 serial ports.

More Info < Back Next > Cancel

Εικόνα ΠΒ.1

Στην επόμενη οθόνη του οδηγού επιλέγουμε απλά “Single Processor System” και προχωράμε στη μεθεπόμενη οθόνη, όπου καλούμαστε να κάνουμε τις πρώτες ουσιαστικές επιλογές, δηλαδή το Processor Configuration. Μιας και ούτως ή άλλως σκοπός μας είναι οι υψηλές επιδόσεις, επιλέγουμε να χρησιμοποιήσουμε τον PowerPC στην υψηλότερη δυνατή συχνότητα που μας επιτρέπεται, δηλαδή στα 400Mhz. Στο Bus clock frequency επιλέγουμε επίσης την υψηλότερη δυνατή τιμή, που είναι τα 100Mhz. Ως debug interface αφήνουμε το προεπιλεγμένο “FPGA JTAG”. Οι επιλογές μας θα πρέπει να είναι αυτές που φαίνονται στην εικόνα ΠΒ.2:



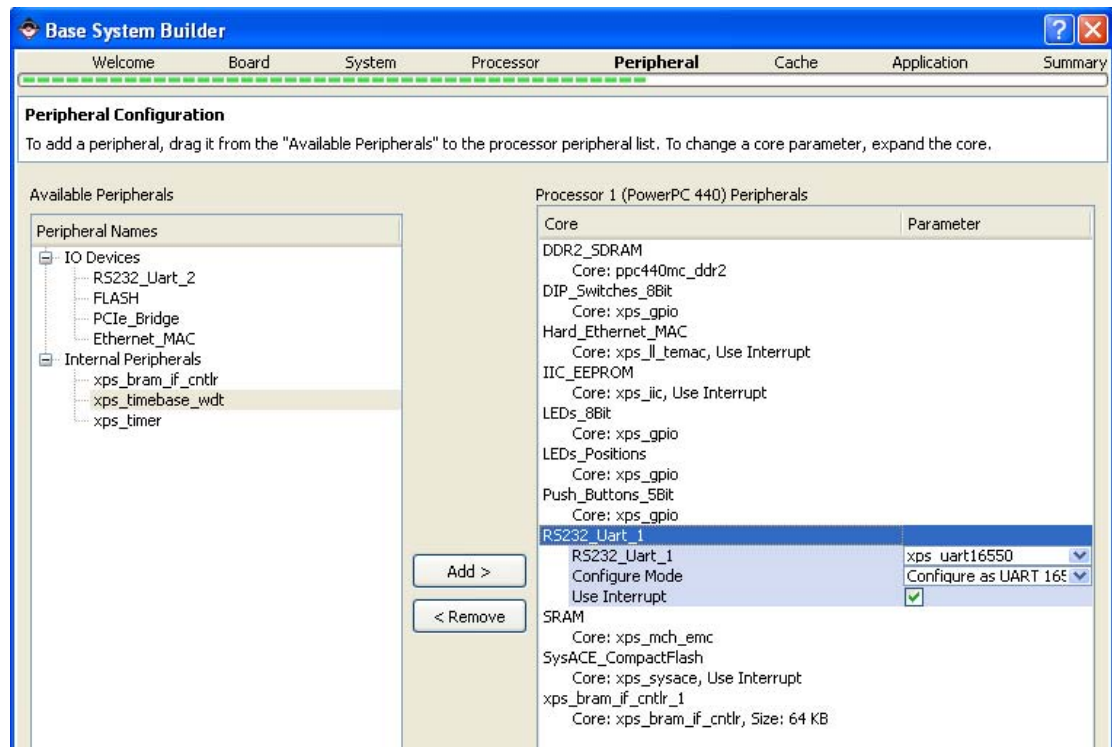
Εικόνα ΠΒ.2

Περνάμε, τώρα στην τελευταία ουσιαστική οθόνη, που αναφέρεται στα περιφερειακά που επιθυμούμε να συμπεριλάβουμε στο σύστημά μας. Εδώ θα χρειαστεί να συμπεριλάβουμε κάποια περιφερειακά τα οποία είναι απαραίτητα για τη λειτουργικότητα της σχεδιάσής μας, και κάποια άλλα τα οποία μπορεί να μην είναι απαραίτητα, είναι όμως επιθυμητό να υπάρχουν.

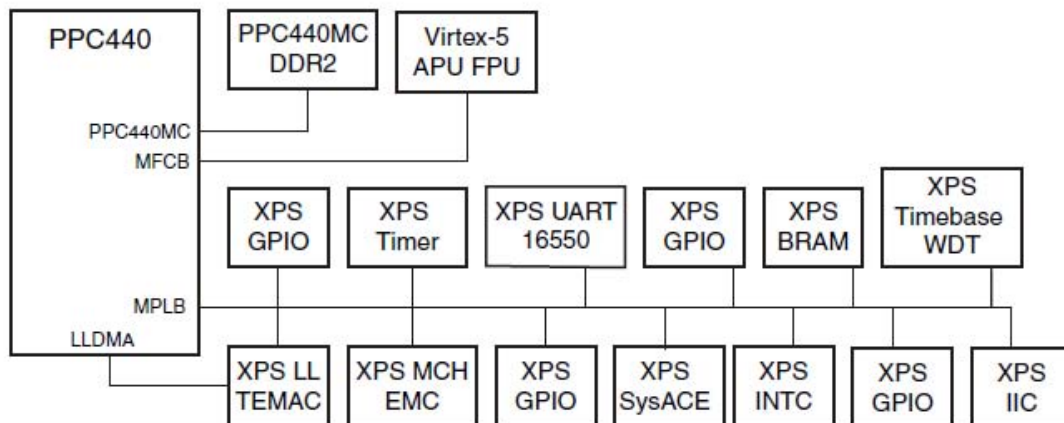
Εδώ θα επιλέξουμε τα “PCIe_bridge”, “Ethernet_MAC”, “RS232_Uart_2” και θα τα αφαιρέσουμε από το προεπιλεγμένο σύστημα, ενώ θα προσθέσουμε το “Hard_Ethernet_MAC”, στο οποίο και θα κάνουμε enable το πεδίο “Use interrupt”. Το ίδιο κάνουμε και για το “IIC_EEPROM”, “RS232_Uart_1” και “SysACE_CompactFlash”. Τελειώνοντας τις αλλαγές σε αυτή την οθόνη, θέτουμε την επιλογή “Size” του “xps_bram_if_cntlr_1” peripheral στα 64KB και στο “RS232_Uart_1” επιλέγουμε “xps_uart16550”. Οι επιλογές μας πριν φύγουμε από αυτή την οθόνη, θα πρέπει να είναι όμοιες με αυτές της εικόνας ΠΒ.3.

Στην επόμενη οθόνη, φροντίζουμε να έχουμε ενεργοποιημένες τις caches (ούτως ή άλλως πρόκειται για embedded caches). Απλώς σε αυτή την οθόνη μπορούμε να επιλέξουμε να κάνουμε cache πολλαπλά memory regions.

Στις επόμενες δυο οθόνες, απλά πατάμε Next, οπότε αφού γίνει και το memory map, με βάση τα περιφερειακά που έχουμε επιλέξει να ενσωματώσουμε στη σχεδιάσή μας, ο BSB ολοκληρώνεται και έχουμε μπροστά μας το System Assembly View του συστήματος που μόλις δημιουργήσαμε. Το block diagram της σχεδίασης φαίνεται στην εικόνα ΠΒ.4.



Εικόνα ΠΒ.3



Εικόνα ΠΒ.4: Το block diagram της σχεδίασής μας.

Δημιουργία του dts αρχείου

Όπως αναφέραμε νωρίτερα, θα χρειαστούμε ένα dts αρχείο με πληροφορίες της σχεδίασής μας, για τη διαδικασία του build του Linux kernel.

Αφού κατεβάσουμε τα Git source control utilities, εκτελούμε την εντολή:

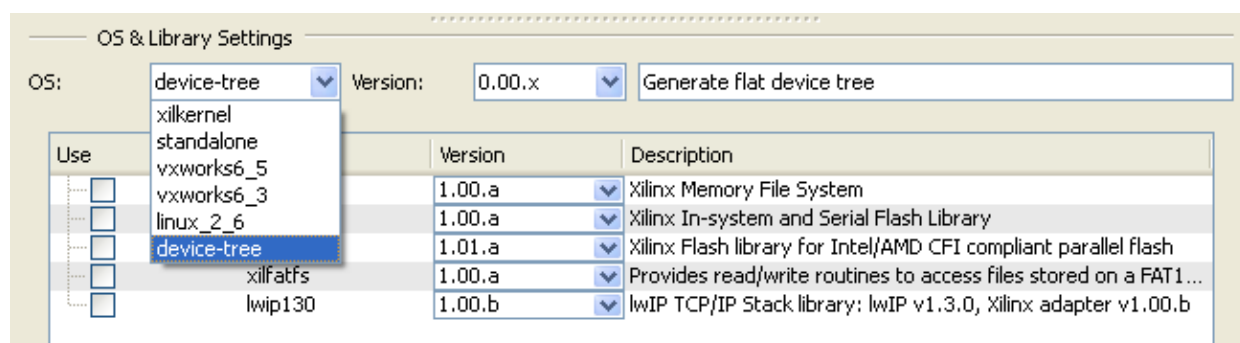
```
git clone git://git.xilinx.com/device-tree.git
```

και στη συνέχεια αντιγράφουμε τα αρχεία στο φάκελο του project μας, ως εξής:

```
cp -r ~/git_master/device-tree/bsp <project_directory>/
```

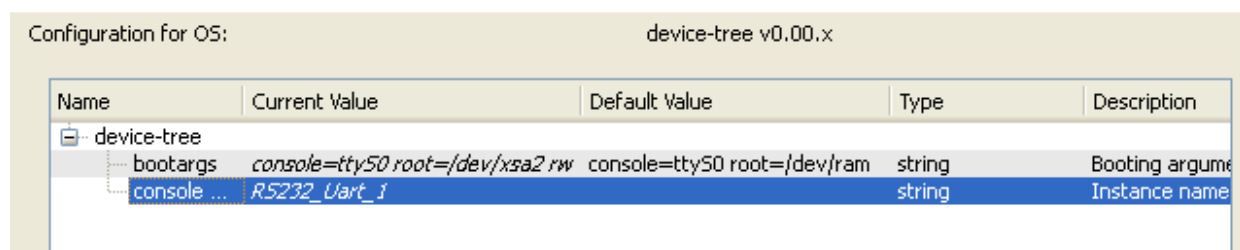
Για να γίνει αντιληπτή η αλλαγή αυτή από το XPS, θα πρέπει είτε να κλείσουμε και να ξανανοίξουμε το project μας, είτε να επιλέξουμε Project→Rescan user repositories.

Τώρα, επιλέγουμε από το μενού Software→Software platforms settings και στην καρτέλα που εμφανίζεται επιλέγουμε “device-tree” από το drop-down μενού στο panel “OS & Library settings”.



Εικόνα ΠΒ.5

Από τις επιλογές αριστερά στο παράθυρο, επιλέγουμε τώρα “OS and Lib configuration” και αφού ανοίξουμε το tree στη δεξιά μεριά του παραθύρου, συμπληρώνουμε την τιμή για το πεδίο Console με το όνομα της UART που έχουμε προσθέσει στο σύστημά μας, ενώ στο πεδίο bootargs συμπληρώνουμε την τιμή που φαίνεται στην πιο κάτω εικόνα (εικόνα ΠΒ.6) αν σκοπεύουμε να χρησιμοποιούμε root file system από την CompactFlash, ή το αφήνουμε ως έχει για τη χρήση ramdisk (οι επιλογές αυτές θα συζητηθούν παρακάτω στην ενότητα του Software infrastructure).



Εικόνα ΠΒ.6

Τελειώνοντας, αφού εφαρμόσουμε τις αλλαγές αυτές, πηγαίνουμε στο μενού “Software” και επιλέγουμε “Generate Libraries and BSPs”. Αυτό θα έχει ως αποτέλεσμα τη δημιουργία ενός αρχείου με την ονομασία xilinx.dts στο φάκελο rrc440/libsrc/device-tree/. Αυτό το αρχείο θα το χρησιμοποιήσουμε αργότερα, αφού το μετονομάσουμε σε virtex405-m1405.dts, αντικαθιστώντας το ομώνυμο αρχείο στο Linux kernel tree (περισσότερες λεπτομέρειες δίνονται στην αντίστοιχη ενότητα του Software infrastructure).

Παράρτημα Γ

Προετοιμασία του περιβάλλοντος προγραμματισμού

Παρακάτω περιγράφουμε τη διαδικασία εγκατάστασης ενός περιβάλλοντος χρήσης για το building ενός Linux kernel χρησιμοποιώντας το Xilinx Git tree.

Η διαδικασία πραγματοποιήθηκε σε υπολογιστή με εγκατεστημένο το OpenSuse Linux 10.2

Κατέβασμα και εγκατάσταση των Git Source Control Utilities

Α' ΤΡΟΠΟΣ:

- 1) Κατεβάζουμε το Git Source Package tar file με την εντολή:
`wget http://kernel.org/pub/software/scm/git/git-1.6.3.3.tar.gz`
- 2) Αποσυμπιέζουμε τα αρχεία χρησιμοποιώντας την εντολή tar:
`tar xzf git-1.6.3.3.tar.gz`
- 3) Αλλάζουμε στο directory που περιέχει το source και τα scripts και κάνουμε build τα source files με τις παρακάτω εντολές:
`cd git-1.6.3.3`
`make prefix=/usr all`
- 4) Όταν τελειώσει το compilation, τα tools μπορούν να εγκατασταθούν στο /usr directory. Ως root εκτελούμε την εγκατάσταση με τις εξής εντολές:
`make prefix=/usr install`
- 5) Για να επιβεβαιώσουμε την ορθή εγκατάσταση των εργαλείων εκτελούμε:
`git --version`

Β' ΤΡΟΠΟΣ

Εναλλακτικά, μπορούμε να κατεβάσουμε και να εγκαταστήσουμε αυτοματοποιημένα το εργαλείο Git, με τον package manager της διανομής μας.

Παράδειγμα: `apt-get install git`
`apt-get install git-core`

Κατεβασμα των kernel source files

Για να κατεβάσουμε τα kernel files, χρειαζόμαστε τα ονόματα του Git server και το όνομα του project το οποίο πρόκειται να κατεβάσουμε.

Η Xilinx παρέχει έναν browser-enabled Git server στη διεύθυνση
<http://git.xilinx.com>

Έχουμε τη δυνατότητα να κατεβάσουμε το project με δυο τρόπους:

A) Με το εργαλείο Git που εγκαταστήσαμε προηγουμένως

Αρχικά, φτιάχνουμε ένα directory για να αποθηκεύσουμε τα source files και στη συνέχεια σε αυτό το φάκελο κάνουμε check out τα source files με τις εξής εντολές:

```
mkdir ~/git_master  
  
cd ~/git_master  
  
git clone git://git.xilinx.com/linux-2.6-xlnx.git
```

Στην οθόνη φαίνεται η εξέλιξη της διαδικασίας (το μέγεθος του download είναι περίπου 390MB), ενώ στο τέλος στο φάκελο θα βρίσκεται το kernel source tree.

Σημείωση: Για να διατηρούμε την πιο πρόσφατη έκδοση του source tree από το git.xilinx.com, μπορούμε να χρησιμοποιούμε την εξής εντολή για να κατεβάζουμε τα updated kernel source files από το kernel root directory:

```
git pull
```

B) Κατεβάζοντας απευθείας το snapshot του source tree.

Αν δε θέλουμε να εγκαταστήσουμε τα Git Tools, μπορούμε να κατεβάσουμε τα kernel sources με το snapshot feature. Για να κάνουμε αυτό, πηγαίνουμε στη σελίδα <http://git.xilinx.com>, όπου επιλέγουμε την επιλογή tree δίπλα από τα project: linux-2.6-xlnx.git και device-tree.git και στη συνέχεια κάνουμε click στην επιλογή snapshot (πάνω αριστερά) για να κατεβάσουμε σε συμπιεσμένη μορφή τα projects.

Για τη συνέχεια αυτού του οδηγού, θα κάνουμε την υπόθεση ότι αυτά τα αρχεία έχουν γίνει extract στους φακέλους:

/<user_directory>/git_master/linux-2.6-xlnx και

/<user_directory>/git_master/device-tree, αντίστοιχα

Κατέβασμα και εγκατάσταση του DENX ELDK 4.2 Toolchain

Για να κάνουμε build το kernel image για το PowerPC 405 ή το PowerPC 440 που βρίσκεται στις FPGA της Xilinx που χρησιμοποιούμε, χρειαζόμαστε ένα GNU tool chain. Για να δημιουργήσουμε το περιβάλλον στο οποίο θα κάνουμε build τον kernel, θα χρησιμοποιήσουμε το DENX ELDK 4.2 tool chain, το οποίο διατίθεται στο <http://www.denx.de>. Εκεί, μπορεί κανείς να βρει το πλήρες manual και λεπτομερείς

οδηγίες εγκατάστασης (<http://www.denx.de/wiki/DULG/ELDK>). Εδώ, θα δούμε με συντομία τη διαδικασία εγκατάστασης:

- 1) Κατεβάζουμε το ELDK 4.2 ISO image με την εξής εντολή:
`wget ftp://ftp.sunet.se/pub/Linux/distributions/eldk/4.2/ppc-linux-x86/iso/ppc-2008-04-01.iso`

Το μέγεθος του image είναι περίπου 1.9GB

- 2) Ως root user, δημιουργούμε ένα mount folder, και κάνουμε mount το ISO image για reading. Στη συνέχεια δημιουργούμε ένα φάκελο για την εγκατάσταση του ELDK:

```
mkdir /mnt/iso
chmod ag+r /mnt/iso
mount -o loop ppc-2008-04-01.iso /mnt/iso
mkdir /opt/eldk
chown <your_user_name> /opt/eldk
```

Βγαίνοντας από root user, εγκαθιστούμε το ELDK:

```
cd /mnt/iso
./install -d /opt/eldk
```

Πληκτρολογούμε 'γ' στην ερώτηση επιβεβαίωσης της εγκατάστασης και η εγκατάσταση αρχίζει, διαρκώντας περίπου 45 λεπτά.

Σημείωση: Βάζοντας επιπλέον την παράμετρο <cru_family> στο τέλος της εντολής install, προσδιορίζουμε την target CPU family για την οποία θέλουμε να εγκαταστήσουμε. Αν παραλείψουμε μία ή περισσότερες τέτοιες παραμέτρους, θα εγκατασταθεί υποστήριξη για όλα τα συμβατά target CPU families. Εμάς μας ενδιαφέρει μόνο υποστήριξη για ppc_4xx, αλλά στο παραπάνω παράδειγμα εγκαθιστούμε όλα τα υποστηριζόμενα CPU families.

Κάνοντας build το default kernel image

- 1) Δημιουργούμε ένα working directory για το project και αντιγράφουμε τα downloaded kernel sources σε αυτό το directory. Στο παράδειγμά μας, υποθέτουμε ότι έχουμε ακολουθήσει τις ως τώρα οδηγίες:

```
mkdir test_ml405
cd test_ml405
```

```
git clone ~/git_master/linux-2.6-xlnx
cd linux-2.6-xlnx
```

Για να κάνουμε build τον kernel χρησιμοποιώντας το make file, θα πρέπει να προσδιορίσουμε την αρχιτεκτονική του target processor, καθώς και το όνομα του cross compiler. Για τον PowerPC η αρχιτεκτονική είναι powerpc, ενώ ο cross compiler του ELDK για τον PowerPC είναι ο ppc-4xx-.

- 2) Για να βεβαιωθούμε ότι τα ELDK tools είναι στο path του τρέχοντα user, εκτελούμε το configuration script μέσα από το root directory του ELDK, ως εξής:
- ```
source /opt/eldk/eldk_init ppc_4xx
```

Η εντολή αυτή θέτει το path και τις cross compiler environment variables, αλλά εκ προεπιλογής το ELDK καθορίζει τη μεταβλητή ARCH ως ppc. Όμως, πλέον το PowerPC kernel development έχει μετακινηθεί στο powerpc kernel folder, επομένως ο χρήστης θα πρέπει να κάνει build χρησιμοποιώντας το powerpc, αντί του ppc. Αυτό το κάνουμε κάνοντας το εξής:

```
export ARCH=powerpc
```

- 3) Τα kernel sources, παρέχονται με default configuration settings για τα ML405 και ML507 reference designs (βλέπε [xilinx.wikidot.com](http://xilinx.wikidot.com)). Κάνουμε configure τον kernel με το σωστό configuration για το board που χρησιμοποιούμε χρησιμοποιώντας την εξής εντολή:

```
make 40x/virtex4_defconfig (για ppc405/ml405)
ή
make 44x/virtex5_defconfig (για ppc440/ml507)
```

- 4) Τα default kernel images απαιτούν ένα RAM disk να βρίσκεται στο kernel source tree για να συμπεριφέρεται σαν το root file system. Αντιγράφουμε το RAM disk image από το [xilinx.wikidot.com](http://xilinx.wikidot.com) στο source tree στο φάκελο arch/powerpc/boot/

- 5) Τέλος, κάνουμε build τον kernel, χρησιμοποιώντας την επόμενη εντολή:

```
make simpleImage.initrd.virtex405-ml405 (για ppc405/ml405)
ή
make simpleImage.initrd.virtex440-ml507 (για ppc440/ml507)
```

Το kernel image elf file δημιουργείται στο φάκελο arch/powerpc/boot με τα παραπάνω ονόματα και την κατάληξη .elf.

Αυτά τα images περιλαμβάνουν το device tree image και ένα ramdisk image και επιτρέπουν σε ένα self contained kernel να κατεβεί σε ένα board χωρίς τη χρήση κάποιου boot-loader, όπως του U-Boot.

## Κατεβάζοντας τη σχεδίασή μας στο Target Board

Η διαδικασία περιλαμβάνει δυο βήματα:

- A) Κατέβασμα της σχεδίασης (αρχείο .bit) στο board
- B) Κατέβασμα του elf αρχείου στο σύστημα

### Κατέβασμα της σχεδίασης στο board

Στο σημείο αυτό θα κατεβάσουμε το αρχείο που περιέχει το configuration του hardware της FPGA. Η διαδικασία αυτή πραγματοποιείται με τη βοήθεια του εργαλείου iMPACT. Στο παράδειγμά μας χρησιμοποιούμε το reference design, το οποίο παρέχεται από τη Xilinx στο <http://git.xilinx.com> και στο <http://xilinx.wikidot.com>. Προσοχή, θα πρέπει να κατεβάσουμε το αρχείο που αντιστοιχεί στο target board (στην περίπτωση μας το ML405), διαφορετικά το σύστημα δε θα είναι λειτουργικό.

Θέτουμε σε λειτουργία το board, αφού έχουμε κάνει τις απαραίτητες συνδέσεις (τροφοδοσία, JTAG) και εκτελούμε το iMPACT, με την εντολή iMPACT (\*να έχουν εκτελεστεί οι οδηγίες source κλπ από το φυλλάδιο οδηγιών). Για ευκολία θα χρησιμοποιήσουμε το GUI mode. Δημιουργούμε (προαιρετικά) ένα νέο project και κάνουμε διπλό κλικ στην επιλογή “Boundary scan” στο παράθυρο “Impact Flows”. Κάνουμε δεξί κλικ στο λευκό χώρο για να κάνουμε Initialize τη JTAG chain (Ctrl+I). Κάνουμε bypass τα devices που δε μας ενδιαφέρουν, και κάνουμε assign το .bit configuration file στο xc4vfx20 (δεξί κλικ-> Assign configuration file). Τέλος, κάνουμε δεξί κλικ στο xc4vfx20 και επιλέγουμε “Program”->OK για να «κατεβάσουμε» το bitstream στην FPGA. Ενδεικτικό μήνυμα στην οθόνη μας πληροφορεί για την επιτυχία της διαδικασίας.

### Κατέβασμα του Kernel Software Image

- 1) Από τη γραμμή εντολών ή το EDK shell, εκκινούμε το Xilinx Microprocessor Debugger (XMD) με την εντολή  
xmd
- 2) Συνδεόμαστε στον επεξεργαστή με την εντολή  
XMD% connect prc hw  
Ο XMD θα μας ενημερώσει για την επιτυχία της σύνδεσης με το μήνυμα:

```
Connected to "ppc" target. Id=0
Starting GDB server for "ppc" target (id=0) at TCP port 1234
```

- 3) Κάνουμε download το kernel image στον επεξεργαστή. Για το ML405, που είναι η περίπτωση μας, δίνουμε την εντολή:  
XMD% dow simpleImage.initrd.virtex405-ml405.elf  
(καλώντας το xmd, μέσα από το φάκελο όπου βρίσκεται το αρχείο elf, διαφορετικά χρησιμοποιούμε το απόλυτο path για το αρχείο).
- 4) Αφού έχουμε συνδέσει ένα serial cable στο board, και ανοίξει ένα RS232 terminal με ρυθμίσεις 9600 8N1, συνεχίζουμε την εκτέλεση στον επεξεργαστή με την εντολή:  
XMD% con
- 5) Θα πρέπει πια στο terminal να βλέπουμε τα μηνύματα που παρουσιάζονται κατά την εκκίνηση του Linux.

### **Φτιάχνοντας ένα μεγαλύτερο root filesystem και περνώντας τα πάντα σε κάρτα Compact Flash**

Θα χρησιμοποιήσουμε την Compact Flash (CF) Card που παρέχεται μαζί με το ML405 evaluation board (σε περίπτωση που κάποια στιγμή επιθυμήσουμε να επαναφέρουμε την κάρτα στην εργοστασιακή της κατάσταση, ακολουθούμε τις οδηγίες στον αντίστοιχο οδηγό της Xilinx).

Η κάρτα, ως παρέχεται, είναι χωρισμένη σε δυο partitions, εκ των οποίων το δεύτερο είναι το Linux partition (393MB). Το πρώτο partition είναι σε FAT16 (για τον ACE controller). Με τη βοήθεια κάποιου Card Reader, διαγράφουμε τα περιεχόμενα του δεύτερου partition (πιθανότατα θα είναι mounted στο /media/disk-1). Το πρώτο partition μπορούμε να το αφήσουμε ως έχει (με τα υπόλοιπα demonstration projects), και να τοποθετήσουμε στο φάκελο myace, το ACE file του δικού μας project (βλ. παρακάτω). Σε αυτή την περίπτωση με το power-on του board θα επιλέγουμε την επιλογή 6 (My own ACE file), ανάμεσα στα υπόλοιπα demo projects (σημείωση: το default Linux project που παρέχεται από τη Xilinx δε θα λειτουργεί, αφού έχουμε αντικαταστήσει το filesystem του με το δικό μας). Εναλλακτικά, μπορούμε να σβήσουμε τα περιεχόμενα του πρώτου partition, ώστε με το που θα ξεκινά το board, να φορτώνεται το Linux.

Αφού, λοιπόν, έχουμε διαγράψει τα περιεχόμενα του δεύτερου partition της CF card, θα δημιουργήσουμε εκεί το filesystem μας, με βάση το ramdisk που χρησιμοποιήσαμε νωρίτερα (από το site της Xilinx/wikidot).

```
sudo mount /media/disk-1 /mnt (ή όπως αλλιώς έχει γίνει mount το 2ο partition της CF card)
```

```
mkdir ramdisk
```

```
cd ramdisk
```

```
wget http://xilinx.wikidot.com/local--files/powerpc-linux/ramdisk.image.gz
```

```
gunzip ramdisk.image.gz
```

```
mkdir mnt
```

```
sudo mount -o loop ramdisk.image mnt
```

```
cd mnt
```

```
sudo cp -r . /mnt
```

```
cd ..
```

```
sudo umount mnt
```

Εναλλακτικά, χωρίς να περιπλέκουμε τη διαδικασία με εντολές, κατεβάζουμε το ramdisk στο 2<sup>ο</sup> partition της CF card και το αποσυμπιέζουμε εκεί.

### **Φτιάχνοντας το elf του kernel για χρήση με filesystem της compact flash**

Για να χρησιμοποιήσουμε το Linux kernel root file system που δημιουργήσαμε, θα πρέπει να κάνουμε κατάλληλα build τον kernel, τροποποιώντας την (αυτοματοποιημένη) διαδικασία που χρησιμοποιήσαμε πιο νωρίς σε αυτό τον οδηγό.

- Στο configuration file που χρησιμοποιήσαμε νωρίτερα κατά το build του kernel (φάκελος 40x/virtex4\_defconfig) ενεργοποιούμε το ACE support στα device drivers (γραμμή 475) – όχι τον παλιό driver (υπάρχουν δύο, εκ των οποίων ο ένας με τον προσδιορισμό “old”).
- Επίσης, στο ίδιο configuration file, ενεργοποιούμε το ext3 filesystem support).
- Στο device tree file, virtex405-m1405.dts (στο φάκελο arch/powerpc/boot/dts), αλλάζω την παράμετρο boot, ως:

```
boot=/dev/xsa2 rw
```

(υποθέτοντας ότι όντως έχουμε βάλει το root file system στο 2<sup>ο</sup> Partition της CF).

Κάνουμε build τον kernel με την εντολή:

```
make simpleImage.virtex405-m1405 (προσοχή, όχι initrd, όπως
νωρίτερα)
```

Στο φάκελο arch/powerpc/boot θα έχει δημιουργηθεί τώρα το elf αρχείο του kernel.

Στη συνέχεια, θα συνδυάσουμε το elf αυτό αρχείο, με το bit αρχείο του reference hardware design, ώστε να εκκινεί κατευθείαν από την CF card, χωρίς να χρειάζεται να «κατεβάζουμε» με το impact κάθε φορά το σύστημα στην FPGA.

### Δημιουργώντας το ACE file

Για να δημιουργήσουμε το top.ace file, εκτελούμε το XMD με τις εξής παραμέτρους:

```
xmd -tcl genace.tcl -jprog -board m1405 -hw [path to .bit file] -elf
[path to .elf file] -ace top.ace -start_address 0x00400000 -target
ppc_hw
```

Αυτή η εντολή θα έχει ως αποτέλεσμα την παραγωγή του αρχείου ACE top.ace, το οποίο θα πρέπει να τοποθετήσουμε στην CF card στον φάκελο m1405/myace (αν δεν έχουμε διαγράψει τα περιεχόμενα του πρώτου partition, όπως αναφέραμε νωρίτερα περιληπτικά). Διαφορετικά, απλά το αντιγράφουμε στο πρώτο partition της CF card, από όπου και θα εκκινήσει, όταν ενεργοποιήσουμε το board, με την επιλογή SYSACE στο αντίστοιχο switch, κάτω από την LCD οθόνη- τα 6 switches μπορείτε να τα αφήσετε όλα στο off).

### Τρέχοντας την πρώτη μας εφαρμογή στον Powerpc

Το development των εφαρμογών μας θα γίνεται στο PC στο οποίο έχουμε εγκαταστήσει τα ELDK tools.

Κάνουμε:

```
source /opt/eldk/eldk_init ppc_4xx
```

και στη συνέχεια κάνουμε compile το πρόγραμμά μας με:

```
ppc_4xx-gcc -o -static exec_name file_name.c κοκ
```

Μπορούμε να μεταφέρουμε αρχεία κλπ στο file system της CF card, είτε συνδέοντας την με κάποιο card reader στο host pc, είτε μεταφέροντας τα κατευθείαν στην CF, ενώ αυτή χρησιμοποιείται στην FPGA, μέσω FTP.

Μπορούμε να εκτελέσουμε εντολές στο target system, είτε μέσω του RS232 terminal (πχ minicom), είτε με telnet από το host system στην IP του board (την οποία μπορούμε να δούμε με ifconfig, έχοντας συνδέσει το board στο δίκτυο).