Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Υπολογιστικων συστηματων

Μελέτη της συμπεριφοράς του διαύλου IEEE 1394 σε εφαρμογές κινούμενης εικόνας

Διπλωματικη Εργασια

Θεόδωρος Β. Καλαματιανός

Επιβλέπων: Νεκτάριος Κοζύρης
Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2008

Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Υπολογιστικων συστηματων

Μελέτη της συμπεριφοράς του διαύλου IEEE 1394 σε εφαρμογές κινούμενης εικόνας

Διπλωματικη Εργασια

Θεόδωρος Β. Καλαματιανός

Επιβλέπων: Νεκτάριος Κοζύρης
Αν. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Νοεμβρίου 2008.

.....	.....	.....
Ν. Κοζύρης	Ν. Παπασπύρου	Π. Τσανάκας
Αν. Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2008

.....
Θεόδωρος Β. Καλαματιανός
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Θεόδωρος Β. Καλαματιανός, 2008
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Ο σκοπός αυτής της διπλωματικής εργασίας ήταν η μελέτη των χαρακτηριστικών και της συμπεριφοράς του σειριακού διαύλου IEEE 1394, με έμφαση στις εφαρμογές κινούμενης εικόνας. Έγινε εκτενής ανάλυση της ομάδας προτύπων IEEE 1394 και υπολογίστηκαν οι θεωρητικές βέλτιστες τιμές για τα βασικά μεγέθη της διαμεταγωγής και της καθυστέρησης, σε συνάρτηση με το μέγεθος των μεταδιδόμενων πακέτων και τις παραμέτρους λειτουργίας του διαύλου. Επιπρόσθετα πραγματοποιήθηκαν πειραματικές δοκιμές σε έναν δίαυλο δύο κόμβων, με κάθε κόμβο να είναι ένας προσωπικός υπολογιστής PC.

Για την εκτέλεση των δοκιμών αναπτύχθηκε, με χρήση της βιβλιοθήκης libraw1394, μια εφαρμογή στο περιβάλλον του λειτουργικού συστήματος Linux, με την οποία είναι δυνατή η χωρίς συμπίεση μετάδοση κινούμενης εικόνας μέσω του διαύλου IEEE 1394. Το λογισμικό αυτό επιτρέπει τον καθορισμό του τύπου των τελούμενων συναλλαγών, του μεγέθους των πακέτων και των παραμέτρων χρονισμού του διαύλου και παράγει στατιστικά στοιχεία για τις μετρούμενες επιδόσεις σε κάθε περίπτωση.

Συγκεκριμένα, πραγματοποιήθηκαν μετρήσεις κατά την τέλεση ασύγχρονων και ισόχρονων συναλλαγών σε όλο το φάσμα των παραμέτρων που επηρεάζουν την απόδοση του συστήματος. Στον κόμβο προορισμού χρησιμοποιήθηκαν ενταμιευτές τόσο στην κύρια μνήμη, όσο και στην αφοσιωμένη μνήμη γραφικών. Στην τελευταία περίπτωση έγινε χρήση των δυνατοτήτων απευθείας πρόσβασης σε φυσικές διευθύνσεις (Physical DMA), που παρέχουν οι ελεγκτές OHCI 1394, για την ανάκτηση και μεταβολή των περιεχομένων της οθόνης ενός απομακρυσμένου κόμβου χωρίς τη δέσμευση πόρων του κεντρικού επεξεργαστή. Η μέθοδος αυτή μπορεί να χρησιμοποιηθεί με οποιαδήποτε συσκευή που εκθέτει τις περιοχές μνήμης της στο χώρο διευθύνσεων του διαύλου συστήματος ενός απομακρυσμένου κόμβου IEEE 1394.

Οι μετρούμενες τιμές ήταν μειωμένες σε σχέση με τις θεωρητικά βέλτιστες, λόγω της παρουσίας ενός μη ιδανικού συστήματος και των επιπλέον περιορισμών που επιβάλλει η λειτουργία ενός υπολογιστή γενικού σκοπού σε ρόλο κόμβου IEEE 1394. Διαπιστώθηκε ότι έχει μεγάλη σημασία ο παραλληλισμός των συναλλαγών από το λογισμικό, όπως επίσης και η ποιότητα υλοποίησης των εμπλεκόμενων ελεγκτών OHCI 1394, η οποία είναι καθοριστική για την επίτευξη της μέγιστης απόδοσης του διαύλου.

Λέξεις Κλειδιά

DMA, Firewire, IEEE 1394, Linux, OHCI, PC, libraw1394, διαμεταγωγή, καθυστέρηση, κινούμενη εικόνα, πρότυπο, σειριακός δίαυλος

Abstract

The scope of this thesis was to study the characteristics and the behaviour of the IEEE 1394 serial bus, with an emphasis on video applications. An extensive analysis of the IEEE 1394 standards set was performed, followed by the calculation of the theoretical optimal values for the basic quantities of throughput and latency, in relation to the transmitted packet size and the bus operational parameters. In addition, experimental tests were performed on a two-node bus, with each node consisting of a PC personal computer.

In order to carry out the tests, an application that uses the libraw1394 library was developed in the Linux operating system environment, allowing the transmission of uncompressed video over the IEEE 1394 bus. The software allows the selection of the performed transaction type, packet size and bus timing parameters and generates statistics on the measured performance for each case.

Specifically, measurements were performed during the execution of asynchronous and isochronous transactions throughout the complete spectrum of the parameters that affect system performance. At the target node the buffers used resided in the main memory, as well as in the dedicated graphics memory. In the latter case, the direct physical address access (Physical DMA) capabilities of the OHCI 1394 controllers were used for the retrieval and modification of the display contents of a remote node, without using any resources of the central processing unit. This method can be used with any device that exposes its memory areas to the system bus address space of a remote IEEE 1394 node.

The measured values were lower than the theoretical best, due to the presence of a non-ideal system and the additional restrictions imposed by the operation of a general purpose computer in the role of an IEEE 1394 node. Having the transactions parallelised by the software was determined to be very important, as well as the implementation quality of the involved OHCI 1394 controllers, which is decisive for achieving the maximum bus performance.

Key Words

DMA, Firewire, IEEE 1394, Linux, OHCI, PC, latency, libraw1394, serial bus, standard, throughput, video

Περιεχόμενα

1 Εισαγωγή 19
2 Ο δίαυλος IEEE 1394 21
2.1 Γενική περιγραφή. 21
2.2 Τεχνικές προδιαγραφές 22
2.3 Το φυσικό επίπεδο 27
2.3.1 Ηλεκτρικές και μηχανικές προδιαγραφές 27
2.3.2 Τοπολογία δένδρου. 30
2.3.3 Πακέτα φυσικού επιπέδου 35
2.3.3.1 Πακέτα αυτοπεριγραφής (Self-ID). 36
2.3.3.2 Πακέτα ενεργοποίησης (Link-on) 38
2.3.3.3 Πακέτα ρύθμισης (PHY Configuration) 39
2.3.3.4 Εκτεταμένα πακέτα φυσικού επιπέδου (Extended PHY packets) 40
2.3.3.4.1 Πακέτο επερώτησης (Ping) 40
2.3.3.4.2 Πακέτο απομακρυσμένης πρόσβασης (Remote access) 41
2.3.3.4.3 Πακέτο απόκρισης (Remote reply) 41
2.3.3.4.4 Πακέτο απομακρυσμένης εντολής (Remote command) 42
2.3.3.4.5 Πακέτο επιβεβαίωσης (Remote confirmation) . . 43
2.3.3.4.6 Πακέτο ανάνηψης (Resume) 43
2.3.4 Τοπολογία αρτηρίας 44
2.4 Το επίπεδο διασύνδεσης 46
2.4.1 Πρωτεύοντα πακέτα 48
2.4.2 Ασύγχρονα πακέτα 49
2.4.2.1 Ασύγχρονα πακέτα χωρίς φορτίο 51
2.4.2.1.1 Αίτηση για ανάγνωση τετράδας 51
2.4.2.1.2 Απόκριση συναλλαγής εγγραφής 52
2.4.2.2 Ασύγχρονα πακέτα με φορτίο μιας τετράδας 53
2.4.2.2.1 Αίτηση για ανάγνωση τμήματος δεδομένων . . . 53
2.4.2.2.2 Αίτηση για εγγραφή τετράδας 54
2.4.2.2.3 Απόκριση συναλλαγής ανάγνωσης τετράδας. . . 55
2.4.2.2.4 Πακέτο χρονισμού ισόχρονων συναλλαγών Cycle Start. 56
2.4.2.3 Ασύγχρονα πακέτα με φορτίο τμήματος δεδομένων. . . . 57
2.4.2.3.1 Αίτηση για εγγραφή τμήματος δεδομένων. . . . 58
2.4.2.3.2 Απόκριση συναλλαγής ανάγνωσης τμήματος δεδομένων 58
2.4.2.3.3 Αίτηση συναλλαγής κλειδώματος 59
2.4.2.3.4 Απόκριση συναλλαγής κλειδώματος 60
2.4.3 Ισόχρονα πακέτα. 61
2.4.4 Πακέτα ασύγχρονης ροής 63
2.4.5 Πακέτα επιβεβαίωσης λήψης 64

2.5 Το επίπεδο συναλλαγής 65
2.5.1 Συναλλαγές ανάγνωσης 65
2.5.2 Συναλλαγές εγγραφής 67
2.5.3 Συναλλαγές κλειδώματος 68
2.5.4 Διαχείριση λαθών και επανάληψη συναλλαγών 69
2.6 Το επίπεδο διαχείρισης του διαύλου 71
2.6.1 Η αρχιτεκτονική CSR. 72
2.6.1.1 Υποστήριξη συναλλαγών στην αρχιτεκτονική CSR . . . 72
2.6.1.2 Διευθυνσιοδότηση στην αρχιτεκτονική CSR 73
2.6.1.3 Καταχωρητές στο δίαυλο IEEE 1394 75
2.6.1.4 Η μνήμη ρυθμίσεων (Configuration ROM) 77
2.6.2 Επαναλήπτες 80
2.6.3 Κόμβοι με ικανότητα τέλεσης συναλλαγών 81
2.6.4 Κόμβοι με ικανότητα τέλεσης ισόχρονων συναλλαγών 82
2.6.5 Κόμβοι με δυνατότητα παροχής κύκλων χρονισμού ισόχρονων συναλλαγών. 83
2.6.6 Κόμβοι με δυνατότητα διαχείρισης ισόχρονων πόρων 84
2.6.7 Κόμβοι με ικανότητα διαχείρισης διαύλου. 86
2.7 Θεωρητική απόδοση του διαύλου IEEE 1394 87
2.7.1 Ασύγχρονες συναλλαγές 87
2.7.2 Ισόχρονες συναλλαγές 101
3 Ο δίαυλος IEEE 1394 σε υπολογιστικά περιβάλλοντα 105
3.1 Γενικά 105
3.2 Το πρότυπο 1394 Open Host Controller Interface (OHCI) 106
3.3 Ο δίαυλος IEEE 1394 σε προσωπικούς υπολογιστές αρχιτεκτονικής PC 110
3.4 Υποστήριξη του διαύλου IEEE 1394 στο λειτουργικό σύστημα Linux . 113
4 Συμπεριφορά του διαύλου IEEE 1394 σε εφαρμογές κινούμενης εικόνας . 117
4.1 Ανάπτυξη λογισμικού 117
4.2 Πειραματικές δοκιμές και αποτελέσματα 122
4.2.1 Ασύγχρονες συναλλαγές 123
4.2.1.1 Ανάγνωση δεδομένων από την κύρια μνήμη απομακρυσμένου υπολογιστή 125
4.2.1.2 Εγγραφή δεδομένων στην κύρια μνήμη απομακρυσμένου υπολογιστή 130
4.2.1.3 Ανάγνωση δεδομένων από απομακρυσμένο ενταμιευτή . . 133
4.2.1.4 Εγγραφή δεδομένων σε απομακρυσμένο ενταμιευτή . . . 138
4.2.1.5 Εγγραφή δεδομένων στο μέσο χώρο διευθύνσεων απομακρυσμένου υπολογιστή 142
4.2.1.6 Ανάγνωση δεδομένων από τη μνήμη γραφικών απομακρυσμένου κόμβου 145
4.2.1.7 Εγγραφή δεδομένων στη μνήμη γραφικών απομακρυσμένου κόμβου. 150
4.2.2 Ισόχρονες συναλλαγές 154
4.2.3 Μετάδοση κινούμενης εικόνας 156
4.2.3.1 Λήψη κινούμενης εικόνας από απομακρυσμένο ενταμιευτή . 156
4.2.3.2 Εγγραφή κινούμενης εικόνας σε απομακρυσμένο ενταμιευτή 157

4.2.3.3 Λήψη κινούμενης εικόνας από τον ενταμιευτή πλαισίου απομακρυσμένου κόμβου . . . 157
4.2.3.4 Εγγραφή κινούμενης εικόνας στον ενταμιευτή πλαισίου απομακρυσμένου κόμβου . . . 157
4.2.3.5 Μετάδοση κινούμενης εικόνας με ισόχρονες συναλλαγές . 158
4.2.4 Συμπεράσματα . . . 159
5 Επίλογος . . . 161
A Βιβλιογραφία . . . 163
B Τεχνικά χαρακτηριστικά διάταξης δοκιμών . . . 167
B.1 Κόμβος A . . . 167
B.2 Κόμβος B . . . 168
C Πηγαίος κώδικας . . . 169
C.1 cbase.c . . . 169
C.2 cbase.h . . . 173
C.3 clist.c . . . 174
C.4 clist.h . . . 177
C.5 cmenu.c . . . 179
C.6 cmenu.h . . . 187
C.7 cterm.c . . . 188
C.8 cterm.h . . . 199
C.9 fproc.c . . . 202
C.10 fproc.h . . . 205
C.11 frame.c . . . 208
C.12 frame.h . . . 210
C.13 l1394.c . . . 211
C.14 prcdp.c . . . 212
C.15 prcdp.h . . . 214
C.16 prcfm.c . . . 215
C.17 prcfm.h . . . 220
C.18 prcfw.c . . . 221
C.19 prcfw.h . . . 230
C.20 queue.c . . . 231
C.21 queue.h . . . 234
C.22 r1394.c . . . 235
C.23 shlib.c . . . 237
C.24 shlib.h . . . 238
C.25 timer.c . . . 244
C.26 timer.h . . . 245
C.27 u1394.c . . . 246
C.28 u1394.h . . . 268
C.29 v1394.c . . . 272

Κατάλογος Πινάκων

1 Πεδία πακέτου φυσικού επιπέδου 35
2 Πεδία πρώτου πακέτου Self-ID 36
3 Πεδία συμπληρωματικού πακέτου Self-ID 37
4 Πεδία πακέτου PHY Configuration 39
5 Πεδία εκτεταμένου πακέτου PHY 40
6 Πεδία πακέτου Remote access 41
7 Πεδία πακέτου Remote reply 42
8 Πεδία πακέτου Remote command 42
9 Πεδία πακέτου Remote confirmation 43
10 Πεδία ακολουθίας διαιτησίας 44
11 Πεδία πρωτεύοντος πακέτου 48
12 Πεδία ασύγχρονου πακέτου 49
13 Πεδία αίτησης ανάγνωσης τετράδας 51
14 Πεδία απόκρισης συναλλαγής εγγραφής 52
15 Πεδία αίτησης για ανάγνωση τμήματος δεδομένων 53
16 Μέγιστη ποσότητα δεδομένων ασύγχρονου πακέτου 54
17 Πεδία αίτησης για εγγραφή τετράδας 55
18 Πεδία απόκρισης συναλλαγής ανάγνωσης τετράδας 55
19 Πεδία πακέτου Cycle Start 56
20 Πεδία αίτησης για εγγραφή τμήματος δεδομένων 58
21 Πεδία απόκρισης συναλλαγής ανάγνωσης τμήματος δεδομένων . 58
22 Ped'ia a'ithshs sunallag'hs kleid'wmatos 60
23 Πεδία απόκρισης συναλλαγής κλειδώματος 60
24 Πεδία ισόχρονου πακέτου 61
25 Μέγιστη ποσότητα δεδομένων ισόχρονου πακέτου 62
26 Πεδία ασύγχρονου πακέτου ροής 63
27 Πεδία πακέτου ACK 64
28 Πεδία διεύθυνσης IEEE 1394 73
29 Χώρος διευθύνσεων κόμβου IEEE 1394 73
30 Βασικοί καταχωρητές αρχιτεκτονικήςCSR 75
31 Βασικοί καταχωρητές διαύλου IEEE 1394 76
32 Καταχωρητές μονάδας IEEE 1394 76
33 Πεδία καταλόγου μνήμης ρυθμίσεων 77
34 Πεδία ριζικού καταλόγου μνήμης ρυθμίσεων 78
35 Πεδία φύλλου περιγραφής 79
36 Μεγέθη ασύγχρονων πακέτων σε bytes 87
37 Ελάχιστη καθυστέρηση (μs) για συναλλαγές ανάγνωσης ή εγγραφής τετράδας 89
38 Μέγιστη διαμεταγωγή (bytes/sec) με συναλλαγές ανάγνωσης ή εγγραφής τετράδας 89
39 Ελάχιστη καθυστέρηση (μs) για συναλλαγές ανάγνωσης τμήματος δεδομένων 90
40 Μέγιστη διαμεταγωγή (bytes/sec) με συναλλαγές ανάγνωσης τμήματος δεδομένων 91

41 Ελάχιστη καθυστέρηση (μs) για συναλλαγές εγγραφής τμήματος δεδομένων 91
42 Μέγιστη διαμεταγωγή (bytes/sec) με συναλλαγές εγγραφής τμήματος δεδομένων 92
43 Ελάχιστη καθυστέρηση (μs) για συνενωμένες συναλλαγές ανάγνωσης ή εγγραφής τετράδας 93
44 Μέγιστη διαμεταγωγή (bytes/sec) με συνενωμένες συναλλαγές ανάγνωσης ή εγγραφής τετράδας 93
45 Ελάχιστη καθυστέρηση (μs) για συνενωμένες συναλλαγές ανάγνωσης τμήματος δεδομένων 94
46 Μέγιστη διαμεταγωγή (bytes/sec) με συνενωμένες συναλλαγές ανάγνωσης τμήματος δεδομένων 95
47 Ελάχιστη καθυστέρηση (μs) για συνενωμένες συναλλαγές εγγραφής τμήματος δεδομένων 96
48 Μέγιστη διαμεταγωγή (bytes/sec) με συνενωμένες συναλλαγές εγγραφής τμήματος δεδομένων 96
49 Ελάχιστη καθυστέρηση (μs) για ενοποιημένες συναλλαγές εγγραφής τετράδας 97
50 Μέγιστη διαμεταγωγή (bytes/sec) με ενοποιημένες συναλλαγές εγγραφής τετράδας 97
51 Ελάχιστη καθυστέρηση (μs) για ενοποιημένες συναλλαγές εγγραφής τμήματος δεδομένων 98
52 Μέγιστη διαμεταγωγή (bytes/sec) με ενοποιημένες συναλλαγές εγγραφής τμήματος δεδομένων 99
53 Ελάχιστη καθυστέρηση (μs) για ισόχρονες συναλλαγές 102
54 Μέγιστη διαμεταγωγή (bytes/sec) με ισόχρονες συναλλαγές . . 103
55 Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου B 126
56 Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου B 126
57 Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου A 128
58 Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου A 129
59 Καθυστέρηση (μs) κατά την εγγραφή δεδομένων στην κύρια μνήμη του κόμβου B 131
60 Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων στην κύρια μνήμη του κόμβου B 131
61 Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου B 134
62 Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου B 134
63 Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου A 136
64 Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου A 136

65 Καθυστέρηση (μs) κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου B 139
66 Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου B 139
67 Καθυστέρηση (μs) κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου A 141
68 Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου A 141
69 Καθυστέρηση (μs) κατά την εγγραφή δεδομένων στο μέσο χώρο διευθύνσεων του κόμβου B 143
70 Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων στο μέσο χώρο διευθύνσεων του κόμβου B 143
71 Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου B 146
72 Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου B 147
73 Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου A 148
74 Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου A 149
75 Καθυστέρηση (μs) κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου B 151
76 Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου B 151
77 Καθυστέρηση (μs) κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου A 152
78 Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου A 153
79 Διαμεταγωγή (bytes/sec) με ισόχρονες συναλλαγές 154

Κατάλογος Σχημάτων

1 Δομή κόμβου IEEE 1394 25
2 Δενδρική τοπολογία διαύλου IEEE 1394 32
3 Μορφή πακέτου φυσικού επιπέδου 35
4 Μορφή πρώτου πακέτου Self-ID 36
5 Μορφή συμπληρωματικού πακέτου Self-ID 37
6 Μορφή πακέτου Link-on 38
7 Μορφή πακέτου PHY Configuration 39
8 Μορφή εκτεταμένου πακέτου PHY 40
9 Μορφή πακέτου Ping 40
10 Μορφή πακέτου Remote access 41
11 Μορφή πακέτου Remote reply 41
12 Μορφή πακέτου Remote command 42
13 Μορφή πακέτου Remote confirmation 43
14 Μορφή πακέτου Resume 43
15 Μορφή ακολουθίας διαιτησίας 44
16 Μορφή πρωτεύοντος πακέτου 48
17 Μορφή ασύγχρονου πακέτου 49
18 Μορφή ασύγχρονου πακέτου χωρίς φορτίο 51
19 Μορφή αίτησης ανάγνωσης τετράδας 51
20 Μορφή απόκρισης συναλλαγής εγγραφής 52
21 Μορφή ασύγχρονου πακέτου με φορτίο μιας τετράδας 53
22 Μορφή αίτησης για ανάγνωση τμήματος δεδομένων 53
23 Μορφή αίτησης για για εγγραφή τετράδας 54
24 Μορφή απόκρισης συναλλαγής ανάγνωσης τετράδας 55
25 Μορφή πακέτου Cycle Start 56
26 Μορφή ασύγχρονου πακέτου με φορτίο τμήματος δεδομένων . . 57
27 Μορφή αίτησης για εγγραφή τμήματος δεδομένων 58
28 Μορφή απόκρισης συναλλαγής ανάγνωσης τμήματος δεδομένων 59
29 Μορφή αίτησης συναλλαγής κλειδώματος 59
30 Μορφή απόκρισης συναλλαγής κλειδώματος 60
31 Μορφή ισόχρονου πακέτου 61
32 Μορφή ασύγχρονου πακέτου ροής 63
33 Μορφή πακέτου ACK 64
34 Δομή διεύθυνσης IEEE 1394 73
35 Μορφή καταλόγου μνήμης ρυθμίσεων 77
36 Μορφή ριζικού καταλόγου μνήμης ρυθμίσεων 78
37 Μορφή φύλλου περιγραφής 79
38 Τέλεση υποδράσης στο χρόνο 88
39 Μέγιστη διαμεταγωγή με συναλλαγές ανάγνωσης τμήματος δεδομένων 90
40 Μέγιστη διαμεταγωγή με συναλλαγές εγγραφής τμήματος δεδομένων 92
41 Μέγιστη διαμεταγωγή με συνενωμένες συναλλαγές ανάγνωσης τμήματος δεδομένων 94

42 Μέγιστη διαμεταγωγή με συνενωμένες συναλλαγές εγγραφής τμήματος δεδομένων 95
43 Μέγιστη διαμεταγωγή με ενοποιημένες συναλλαγές εγγραφής τμήματος δεδομένων 98
44 Τέλεση ισόχρονων συναλλαγών στο χρόνο 101
45 Μέγιστη συνολική διαμεταγωγή με ισόχρονες συναλλαγές . . . 102
46 Κύκλος ζωής πλαισίου 118
47 Διαμεταγωγή κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου B 125
48 Διαμεταγωγή κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου A 128
49 Διαμεταγωγή κατά την εγγραφή δεδομένων στην κύρια μνήμη του κόμβου B 130
50 Διαμεταγωγή κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου B 133
51 Διαμεταγωγή κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου A 135
52 Διαμεταγωγή κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου B 138
53 Διαμεταγωγή κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου A 140
54 Διαμεταγωγή κατά την εγγραφή δεδομένων στο μέσο χώρο διευθύνσεων του κόμβου B 142
55 Διαμεταγωγή κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου B 146
56 Διαμεταγωγή κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου A 148
57 Διαμεταγωγή κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου B 150
58 Διαμεταγωγή κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου A 152

1 Εισαγωγή

Ο σειριακός δίαυλος IEEE 1394, πιο γνωστός με την εμπορική ονομασία Firewire, αποτελεί μια διαρκώς εξελισσόμενη τεχνολογία με ποικίλες εφαρμογές. Ήταν μία από τις πρώτες τεχνολογίες που παρείχαν την ευελιξία ενός αυτορρυθμιζόμενου συστήματος με χαμηλό κόστος υλοποίησης, κάτι που επιτρέπει τη χρήση του από άτομα χωρίς τεχνικές γνώσεις. Επιπλέον, η προδιαγραφή του διαύλου από τον οργανισμό IEEE (Institute of Electrical and Electronics Engineers) τον διαφοροποιεί ποιοτικά από πολλές άλλες εναλλακτικές που, με την υποστήριξη μερικών κατασκευαστών, έχουν απλά αποκτήσει μια de facto διάδοση στην αγορά. Τα ιδιαίτερα χαρακτηριστικά του διαύλου IEEE 1394 τον καθιστούν κατάλληλο τόσο για καταναλωτικές, όσο και για βιομηχανικές και επιστημονικές εφαρμογές, με τις περισσότερες από αυτές να εστιάζονται στον τομέα μετάδοσης ήχου και εικόνας.

Το πρώτο μέρος αυτής της εργασίας συνίσταται από μια εκτενή ανάλυση των τεχνικών χαρακτηριστικών του διαύλου, με έμφαση κυρίως στα στοιχεία που επηρεάζουν τις επιδόσεις του και το προγραμματιστικό μοντέλο με βάση το οποίο γράφεται το λογισμικό των συσκευών που τον υποστηρίζουν. Η ανάλυση αυτή έγινε με βάση τις θεωρητικές προδιαγραφές του προτύπου IEEE 1394 και των τροπολογιών που το ακολούθησαν, παρουσιάζοντας σε κάποιες περιπτώσεις τις διαφοροποιήσεις που αυτές επέφεραν. Η ενότητα αυτή καταλήγει με τον υπολογισμό των βέλτιστων επιδόσεων που επιτρέπονται από το δίαυλο υπό ιδανικές συνθήκες, προκειμένου να καταστεί δυνατή η σύγκριση με τις ποσοτικές μετρήσεις που προκύπτουν από ένα πραγματικό σύστημα.

Το δεύτερο μέρος αφορά τη μελέτη ενός πραγματικού συστήματος, το οποίο αποτελείται από δύο ηλεκτρονικούς υπολογιστές που είναι συνδεδεμένοι με ένα δίαυλο IEEE 1394. Για τη μελέτη αυτή ήταν απαραίτητη η συγγραφή μιας εφαρμογής λογισμικού για την πραγματοποίηση των μετρήσεων. Για τη σύσταση της θεμελιώδους βασικής υποδομής προτιμήθηκε το λειτουργικό σύστημα Linux, καθώς παρέχει ένα οικείο και ιδιαίτερα ευέλικτο προγραμματιστικό περιβάλλον. Επιπλέον η διαθεσιμότητα του κώδικα του πυρήνα του το καθιστά κατάλληλο για εφαρμογές όπου, προκειμένου να ερμηνευθούν συγκεκριμένες παρατηρούμενες συμπεριφορές, είναι κατά περίπτωση απαραίτητη η ανάλυση του κώδικα ελέγχου των συσκευών του υπολογιστή και των βιβλιοθηκών υποστήριξης.

Για την ανάπτυξη του λογισμικού εστιάσαμε στη μετάδοση κινούμενης εικόνας χωρίς συμπίεση. Ο πρώτος λόγος για αυτήν την επιλογή ήταν η διάδοση που έχει ο δίαυλος IEEE 1394 στον τομέα της διασύνδεσης επαγγελματικών και καταναλωτικών συσκευών ήχου και εικόνας, κάτι που υποδεικνύει την καταλληλότητα του. Πέρα από τη συγγένειά της με τις ήδη υπάρχουσες, η εφαρμογή αυτή επιτρέπει τη δοκιμή ενός μεγάλου τμήματος των δυνατοτήτων που παρέχονται για τη λήψη ποσοτικών μετρήσεων. Ο κυριότερος λόγος, όμως, ήταν η δυνατότητα ύπαρξης ενός απτού αποτελέσματος με την προβολή της μεταδιδόμενης εικόνας στην οθόνη του υπολογιστή. Αυτό επιτρέπει την πραγματοποίηση μιας ποιοτικής εκτίμησης, η οποία θα συνοδεύει τα αποτελέσματα των πειραματικών δοκιμών. Με δεδομένο ότι στο χώρο των πολυμέσων το κριτήριο είναι πάντα οι αισθήσεις του χρήστη, δεν θα ήταν υπερβολή να θεωρηθεί ότι η ποιοτική αξιολόγηση κάθε συστήματος είναι τελικά αυτή με τη μεγαλύτερη σημασία.

2 Ο δίαυλος IEEE 1394

2.1 Γενική περιγραφή

Ο δίαυλος IEEE 1394, πιο ευρέως γνωστός με την εμπορική ονομασία Firewire, είναι ένας σειριακός δίαυλος υψηλών επιδόσεων, σχεδιασμένος για τη διασύνδεση τόσο μεμονωμένων συσκευών, όσο και για τη γεφύρωση άλλων τύπων διαύλων. Η ανάπτυξή του ξεκίνησε στα τέλη της δεκαετίας του 1980 από την εταιρεία Apple Inc. με σκοπό τη δημιουργία μιας σειριακής τεχνολογίας χαμηλού κόστους που θα παρείχε παρεμφερείς δυνατότητες με παράλληλους διαύλους όπως ο SCSI (Small Computer System Interface), ενώ ταυτόχρονα θα ήταν κατάλληλος και για τη διασύνδεση ψηφιακών συσκευών ήχου και κινούμενης εικόνας.

Στα τέλη του 1995 η επιτροπή P1394 του IEEE (Institute of Electrical and Electronics Engineers) ενέκρινε το πρότυπο IEEE Std 1394-1995 IEEE Standard for a High Performance Serial Bus, επιτρέποντας έτσι την ευρεία χρήση του διαύλου σε πληθώρα εφαρμογών. Το πρότυπο του 1995 ακολούθησαν τρεις τροπολογίες (IEEE Std 1394a-2000, IEEE Std 1394b-2002 και IEEE Std 1394c-2006), οι οποίες αντιμετώπισαν κάποιες από τις τεχνικές αδυναμίες της αρχικής έκδοσης, ενώ αύξησαν δραστικά τη μέγιστη ταχύτητα μεταφοράς δεδομένων ακολουθώντας την εξέλιξη της διαθέσιμης τεχνολογίας.

Ο δίαυλος IEEE 1394 χρησιμοποιείται ευρέως, αν και το ελαφρώς μεγαλύτερο κόστος των συσκευών που τον υποστηρίζουν έχει περιορίσει τη χρήση του κυρίως σε εφαρμογές όπου οι επιδόσεις και τα ιδιαίτερα τεχνικά χαρακτηριστικά του είναι κρίσιμα. Είναι η μόνη υποστηριζόμενη μορφή διασύνδεσης για την πλειοψηφία των καταναλωτικών συσκευών σύλληψης και αναπαραγωγής κινούμενης εικόνας που υποστηρίζουν το πρότυπο DV, ενώ χρησιμοποιείται συχνά στο χώρο της επαγγελματικής επεξεργασίας ήχου, καθώς και σε επιστημονικές, ιατρικές και βιομηχανικές εφαρμογές, όπως η μικροσκοπία και η ενδοσκοπία. Μια σειρά πρωτοκόλλων με βάση το IEEE 1394, όπως τα SBP (Serial Bus Protocol) και IEC 61883 επιτρέπουν την διασύνδεση πλήθους καταναλωτικών προϊόντων με ομοιόμορφο τρόπο. Επιπλέον, η τεχνολογία IEEE 1394 έχει χρησιμοποιηθεί σε αυτοκίνητα, αεροσκάφη και διαστημικά οχήματα, όπου το χαμηλότερο βάρος και κόστος μιας σειριακής καλωδίωσης είναι σημαντικό προσόν έναντι των παράλληλων εναλλακτικών με αντίστοιχες δυνατότητες.

Στο χώρο των ηλεκτρονικών υπολογιστών συναντώνται αρκετά συχνά μονάδες που υποστηρίζουν το πρότυπο IEEE 1394, ωστόσο το χαμηλότερο κόστος υλοποίησης του παρομοίων ονομαστικών χαρακτηριστικών διαύλου USB (Universal Serial Bus) έχει οδηγήσει στην ευρύτατη διάδοση του τελευταίου, εν μέρει εις βάρος του IEEE 1394, παρά το γεγονός ότι στην πράξη έχει χαμηλότερες επιδόσεις. Παρ΄ όλα αυτά, τα περισσότερα σύγχρονα λειτουργικά συστήματα διαθέτουν υποστήριξη τόσο για την πλειοψηφία των ελεγκτών IEEE 1394, όσο και για τη χρήση αρκετών ομάδων περιφερειακών, όπως μονάδες αποθήκευσης δεδομένων και συσκευές σύλληψης, αποθήκευσης, επεξεργασίας και αναπαραγωγής ήχου και εικόνας.

2.2 Τεχνικές προδιαγραφές

Το πρότυπο IEEE 1394 σχεδιάστηκε λαμβάνοντας αρκετά στοιχεία του από δοκιμασμένα προϋπάρχοντα πρότυπα, σε πολλές περιπτώσεις με απευθείας παραπομπές σε αυτά, μειώνοντας έτσι τη δυσκολία αφομοίωσής του από την επιστημονική και βιομηχανική κοινότητα και καθιστώντας το δυνητικά πιο ευέλικτο. Η πιο σημαντική ίσως επιλογή ήταν η χρήση της αρχιτεκτονικής CSR που περιγράφεται στο πρότυπο IEEE Std 1212-1994 Control and Status Registers (CSR) Architecture for Microcomputer Buses (ISO/IEC 13213: 1994). Η αρχιτεκτονική αυτή έχει επιλεγεί και για μια σειρά άλλων διαύλων που έχουν προτυποποιηθεί από το IEEE, όπως οι παράλληλοι δίαυλοι IEEE Std 896-1991 FutureBus+ και IEEE Std 1596-1992 Scalable Coherent Interface (SCI). Η χρήση κοινής αρχιτεκτονικής επιτρέπει την εύκολη διασύνδεση ανάμεσα σε αυτούς τους διαύλους με τη βοήθεια σχετικά απλών μονάδων γεφύρωσης.

Η αρχιτεκτονική CSR προβλέπει τη χρήση διευθύνσεων 64 bit, από τα οποία τα 16 είναι η διεύθυνση του υποδεικνυόμενου κόμβου και τα υπόλοιπα 48 αναφέρονται σε κάποια θέση εντός του χώρου διευθύνσεων του κόμβου αυτού. Στα πλαίσια του IEEE 1394 τα 16 δυαδικά ψηφία της διεύθυνσης χωρίζονται επιπλέον, με 10 bit (0-1023) να δίνουν τον κωδικό αριθμό ταυτοποίησης του διαύλου και τα υπόλοιπα 6 (0-63) να υποδεικνύουν κάποιο συγκεκριμένο κόμβο, όπως ακριβώς προβλέπεται και για τους FutureBus+ και SCI. Αυτό επιτρέπει τη διασύνδεση τους με τον IEEE 1394 χρησιμοποιώντας έναν κοινό χώρο διευθύνσεων, απλά με τη χρήση διαφορετικού κωδικού αριθμού για κάθε συνδεδεμένο δίαυλο. Με τον κωδικό αριθμό 1023, ο οποίος υποδηλώνει τον τοπικό δίαυλο κάθε κόμβου, να είναι δεσμευμένος επιτρέπεται συνολικά η συνύπαρξη 1023 διαύλων στον ίδιο χώρο διευθύνσεων.

Από τα 6 bit ταυτοποίησης των κόμβων ενός διαύλου, η τιμή 63 είναι δεσμευμένη για την ταυτόχρονη διευθυνσιοδότηση όλων των κόμβων (broadcast), οπότε είναι δυνατή η σύνδεση 63 διαφορετικών συσκευών. Το πρότυπο IEEE 1394 προβλέπει τη χρήση τόσο μιας τοπολογίας αρτηρίας (backplane environment) όπου όλες οι συσκευές συνδέονται ηλεκτρικά στις ίδιες γραμμές επικοινωνίας, όσο και μιας ακυκλικής δενδρικής τοπολογίας (cable environment) όπου κάθε συσκευή συνδέεται με μία ή περισσότερες άλλες μέσω συνδέσεων σημείο προς σημείο.

Η τοπολογία αρτηρίας συναντάται σπανίως και προορίζεται κυρίως για τη συμπλήρωση ενός παράλληλου διαύλου εντός της ίδιας μονάδας, παραδείγματος χάρη για τον έλεγχο των υπομονάδων ενός υπολογιστή. Η δενδρική τοπολογία, αντίθετα, είναι ευρέως διαδεδομένη, με τις περισσότερες συσκευές να παρέχουν τουλάχιστον δυο θύρες IEEE 1394 για την σημείο-προς-σημείο σύνδεση με άλλες συσκευές, αν και η τοπολογία ενός διαύλου που αποτελείται αποκλειστικά από κόμβους με δυο το πολύ θύρες ουσιαστικά εκφυλλίζεται σε τοπολογία αλυσίδας. Το πρότυπο επιβάλλει ως όριο έναν μέγιστο αριθμό 16 συνδέσεων ανάμεσα σε οποιουσδήποτε δυο κόμβους, απαγορεύοντας έτσι την ύπαρξη μεγάλων αλυσίδων και αποφεύγοντας την ανάγκη για τη διαχείριση των πολύ μεγαλύτερων καθυστερήσεων που θα προέκυπταν. Στη δενδρική τοπολογία δεν υπάρχει προκαθορισμένος κόμβος-ρίζα και είναι δυνατόν κατά τη διαδικασία της αυτορρύθμισης του διαύλου δυο ή περισσότεροι κόμβοι να ανταγωνιστούν για τη θέση της ρίζας του δένδρου και για τις υπόλοιπες θέσεις διαχείρισης της λειτουργίας του διαύλου.

Τα 48 bit του χώρου διευθύνσεων ενός κόμβου συνήθως απεικονίζονται απευθείας στις φυσικές διευθύνσεις του εσωτερικού διαύλου του. Παρά την ύπαρξη κάποιων δεσμευμένων από την αρχιτεκτονική CSR περιοχών, επιτρέπουν τη διευθυνσιοδότηση περίπου 256 Terrabytes, ποσό που τουλάχιστον προς το παρόν υπερβαίνει τη διαθέσιμη μνήμη ακόμα και των μεγαλύτερων υπολογιστικών συστημάτων. Στην περίπτωση ενός προσωπικού υπολογιστή μια θύρα IEEE 1394 δίνει τυπικά άμεση πρόσβαση στο σύνολο της φυσικής του μνήμης (Direct Memory Access, DMA), αν και σε κάποιες περιπτώσεις η χρήση μονάδων διαχείρισης του χώρου διευθύνσεων των συσκευών (Input Output Memory Management Unit, IOMMU) επιβάλλει περιορισμούς τόσο για λόγους ασφάλειας των δεδομένων της κύριας μνήμης, όσο και για την προστασία από συσκευές που παρουσιάζουν προβλήματα στη λειτουργία τους.

Η δυνατότητα απευθείας πρόσβασης στην κύρια μνήμη ενός κόμβου επιτρέπει να χαρακτηρίσουμε το δίαυλο IEEE 1394 ως δίαυλο επέκτασης συστήματος, γεγονός που τον διαφοροποιεί από τους διαύλους περιφερειακών, όπως για παράδειγμα ο USB, όπου όλες οι διεργασίες πρέπει να ξεκινήσουν από τον ελεγκτή του διαύλου (host controller). Στις περισσότερες υλοποιήσεις η πρόσβαση στα δεδομένα πραγματοποιείται χωρίς τη βοήθεια του επεξεργαστή του κόμβου, γεγονός που επιτρέπει τη λειτουργία σε σταθερά υψηλούς ρυθμούς μεταφοράς δεδομένων και με μικρούς χρόνους καθυστέρησης (latency), χωρίς ταυτόχρονα να απορροφά επεξεργαστική ισχύ.

Στην αρχική του έκδοση, το πρότυπο IEEE Std 1394-1995 παρείχε τρεις δυνατούς ρυθμούς μεταφοράς δεδομένων για την τοπολογία δένδρου, στα 100, 200 και 400 Mbit/sec περίπου, που είναι γνωστοί στην τεχνική κοινότητα ως S100, S200 και S400 αντιστοίχως. Στη δεύτερη τροπολογία του IEEE Std 1394b-2002 όμως, επετράπησαν επιπλέον ταχύτητες 800, 1600 και 3200 Mbit/sec - S800, S1600 και S3200 αντιστοίχως - με τη χρήση κατάλληλου υλικού και καλωδίωσης. Με αυτή τη σημαντική αναβάθμιση στις επιδόσεις ο δίαυλος IEEE 1394 γίνεται ιδιαίτερα ελκυστικός για κάποιες εφαρμογές, ενώ αποκτά καθαρό τεχνικό προβάδισμα απέναντι στον USB, που στην έκδοση 2.0 (2000) επιτρέπει ρυθμό μεταφοράς δεδομένων 480 Mbit/sec.

Για την τοπολογία αρτηρίας προβλέπονται ταχύτητες μόλις 25 και 50 Mbit/sec (S25 και S50), αν και υπάρχουν ορισμένες υλοποιήσεις στην αγορά που παρέχουν και την υψηλότερη ταχύτητα S100, παύοντας όμως τότε να είναι συμβατές με το πρότυπο IEEE 1394. Είναι προφανές ότι σε αυτήν την τοπολογία επιτυγχάνονται μικρότερες καθυστερήσεις μετάδοσης, καθώς όλοι οι κόμβοι βρίσκονται συνδεδεμένοι στο ίδιο κύκλωμα και δεν υπάρχουν οι χρόνοι αναμετάδοσης από κόμβο σε κόμβο που παρατηρούμε στην τοπολογία δένδρου. Επιπλέον στην τοπολογία αυτή προβλέπεται υποστήριξη για την ύπαρξη κόμβων με υψηλή προτεραιότητα κατά τη διαιτησία (priority arbitration), βελτιώνοντας έτσι ακόμα περισσότερο τους μέγιστους χρόνους καθυστέρησης για χρονικά κρίσιμες μεταδόσεις δεδομένων. Αυτή η μορφή διαιτησίας δεν υπάρχει στην τοπολογία δένδρου, όπου υποστηρίζεται μόνο μια μορφή αμερόληπτης διαιτησίας (fair arbitration). Η δραστικά χαμηλότερη ταχύτητα της τοπολογίας αρτηρίας και η απουσία προτυποποιημένων ηλεκτρικών προδιαγραφών, όμως, έχουν περιορίσει τη διάδοσή της σε ορισμένες πολύ εξειδικευμένες εφαρμογές.

Η βασική μονάδα μεταφοράς δεδομένων για το πρότυπο IEEE 1394 είναι η τετράδα (quadlet), η οποία αποτελείται από τέσσερα bytes ή 32 δυαδικά ψηφία. Η μετάδοση των δεδομένων γίνεται με το πιο σημαντικό ψηφίο να εκπέμπεται πρώτο. Όλα τα πακέτα που μεταδίδονται έχουν μέγεθος πολλαπλάσιο αυτής της βασικής μονάδας, με τα δεδομένα να συμπληρώνονται με επιπλέον μηδενικά bytes (padding) όταν αυτό είναι απαραίτητο. Εξαίρεση σε αυτόν τον κανόνα είναι τα πακέτα επιβεβαίωσης λήψης (acknowledgement packets, ACK), τα οποία έχουν μήκος ακριβώς ένα byte, χαρακτηριστικό που τα διαχωρίζει από οποιοδήποτε άλλο πακέτο.

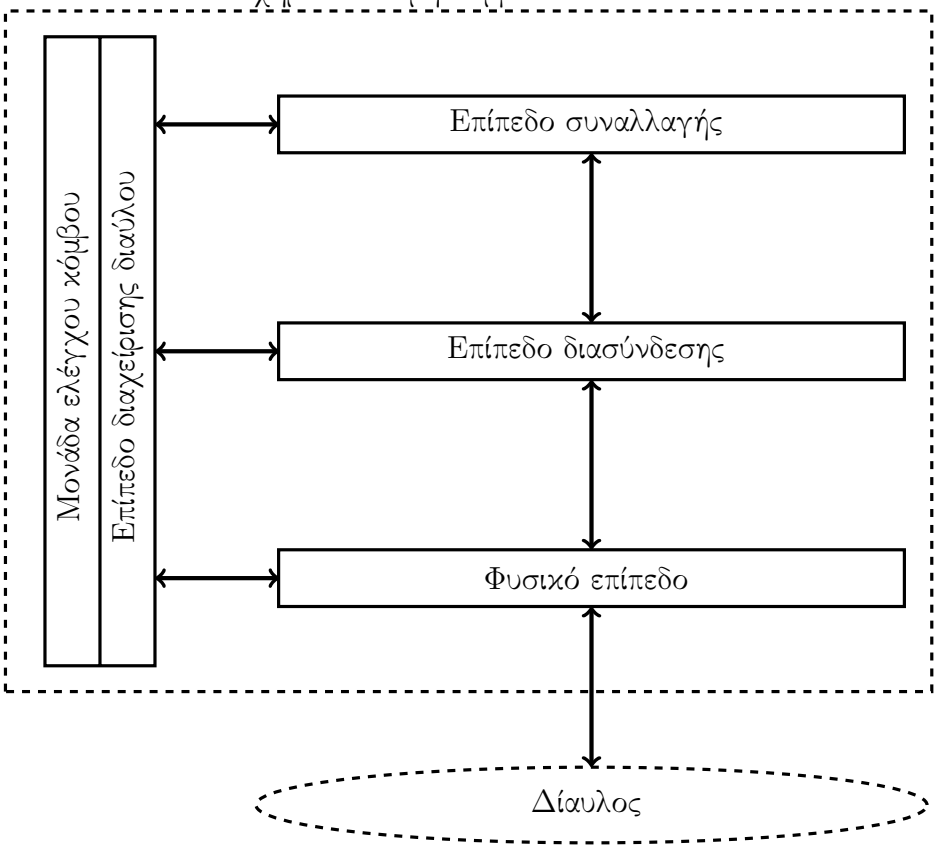
Ακολουθώντας την αρχιτεκτονική CSR, ο δίαυλος IEEE 1394 μεταφέρει δεδομένα σε μορφή πακέτων και παρέχει εντολές εγγραφής, ανάγνωσης και κλειδώματος. Οι εντολές αυτές πραγματοποιούνται ασύγχρονα, δηλαδή ακολουθούν τη λογική αίτησης-απόκρισης. Οι εντολές εγγραφής και ανάγνωσης επιτρέπουν την αξιόπιστη μεταφορά δεδομένων, ενώ οι εντολές κλειδώματος χρησιμοποιούνται για λειτουργίες συγχρονισμού. Εκτός από τις ασύγχρονες εντολές, προβλέπεται και η δυνατότητα για ισόχρονη μεταφορά δεδομένων, όπου διατίθενται εξασφαλισμένα σταθερός ρυθμός μετάδοσης και μικροί χρόνοι καθυστέρησης, με αντίτιμο, όμως, την απουσία ελέγχου της εγκυρότητας των αποστελλόμενων δεδομένων και της επανεκπομπής αλλοιωμένων πακέτων. Οι ισόχρονες λειτουργίες απευθύνονται κατά κύριο λόγο σε εφαρμογές όπου ο χρόνος καθυστέρησης και η ταχύτητα μετάδοσης είναι κρίσιμες, ενώ τυχόν χαμένα πακέτα λόγω σφαλμάτων είναι πλέον άχρηστα και δεν έχει νόημα να επανεκπεμπούν. Τέτοιες εφαρμογές είναι οι εφαρμογές συλλογής και καταγραφής δεδομένων, καθώς και κάποιες που σχετίζονται με ήχο και κινούμενη εικόνα.

Το τελευταίο βασικό χαρακτηριστικό του διαύλου IEEE 1394 είναι οι δυνατότητες αυτόματης ρύθμισης που διαθέτει. Με την προσθήκη ή αφαίρεση κάθε συσκευής, κάτι που μπορεί να πραγματοποιηθεί με το σύστημα σε λειτουργία, εκχωρούνται αυτόματα νέες διευθύνσεις στους κόμβους και επαναπροσδιορίζεται ο κόμβος που θα αποτελέσει τη ρίζα του δένδρου. Οι αρμοδιότητες που σχετίζονται με τη διαχείριση πόρων και τη διαιτησία των λειτουργιών του διαύλου ανατίθενται στους κόμβους εκείνους που μπορούν να τις επιτελέσουν και βρίσκονται όσο το δυνατόν πιο κοντά στη ρίζα της δενδρικής δομής. Αυτή η διαδικασία υποβοηθείται σημαντικά από την αρχιτεκτονική CSR, η οποία προβλέπει ότι κάθε κόμβος διαθέτει σε συγκεκριμένη διεύθυνση μια μνήμη ROM ρυθμίσεων (Configuration ROM) που περιγράφει τις δυνατότητες του κόμβου, ενώ η εξειδίκευση του IEEE 1212 στο πρότυπο IEEE 1394 περιέχει μια σειρά καταχωρητών για την πραγματοποίηση της αυτορρύθμισης του διαύλου.

Για την περιγραφή του διαύλου, το πρότυπο IEEE Std 1394-1995 προτείνει μια δομή τεσσάρων επιπέδων και ορίζει συγκεκριμένα με ποιο τρόπο επικοινωνούν τα επίπεδα αυτά μεταξύ τους:

- Φυσικό επίπεδο (physical layer, PHY)
- Επίπεδο διασύνδεσης (link layer)
- Επίπεδο συναλλαγής (transaction layer)
- Επίπεδο διαχείρισης του διαύλου (bus management)

Σχήμα 1: Δομή κόμβου IEEE 1394



Το φυσικό επίπεδο εκτελεί τρεις βασικές λειτουργίες: καθορίζει τα ηλεκτρικά και μηχανικά χαρακτηριστικά του διαύλου, πραγματοποιεί την εκπομπή και τη λήψη των σειριακών δεδομένων και παρέχει υπηρεσίες διαιτησίας με βάση τις οποίες ο κόμβος διεκδικεί τον έλεγχο του διαύλου, έτσι ώστε να εξασφαλίζεται η αμερόληπτη πρόσβαση των κόμβων στο δίαυλο. Κατά την αυτόματη διαμόρφωση του διαύλου, το φυσικό επίπεδο είναι υπεύθυνο για την απόδοση των κωδικών ταυτοποίησης στους κόμβους και για την επιλογή του κόμβου που θα αποτελέσει τη ρίζα τη δενδρικής δομής. Επιπλέον φροντίζει να καλύψει τις λεπτομέρειες της τοπολογίας, παρέχοντας την εικονική επίπεδη τοπολογία που χρησιμοποιούν τα ανώτερα επίπεδα.

Το υπόστρωμα διασύνδεσης παρέχει τις λειτουργίες αποστολής ασύγχρονων πακέτων και της απόκρισης με πακέτα επιβεβαίωσης της λήψης. Επιπλέον αναλαμβάνει την αποστολή και τη λήψη ισόχρονων πακέτων, διακινώντας δεδομένα με σταθερό προκαθορισμένο ρυθμό.

Το επίπεδο διασύνδεσης χρησιμοποιείται από το υπόστρωμα συναλλαγής για την υλοποίηση των βασικών εντολών του διαύλου - εγγραφή, ανάγνωση και κλείδωμα - που προβλέπονται από την αρχιτεκτονική CSR. Το πρότυπο IEEE 1394 έχει ως ελάχιστη απαίτηση για κάθε κόμβο την υλοποίηση των εντολών εγγραφής και ανάγνωσης που είναι απαραίτητες για την πρόσβαση στους πόρους CSR, έτσι ώστε να μπορεί να λάβει μέρος στις διαδικασίες αυτορρύθμισης του διαύλου.

Τέλος, το πρότυπο προδιαγράφει το υπόστρωμα διαχείρισης του διαύλου, το

οποίο επικοινωνεί ταυτόχρονα και με τα τρία προαναφερθέντα επίπεδα. Σε αυτό το επίπεδο βρίσκονται οι λειτουργίες του παροχέα κύκλων χρονισμού (Cycle Master, CM), του διαχειριστή ισόχρονων πόρων (Isochronous Resource Manager, IRM) και του διαχειριστή διαύλου (Bus Manager, BM). Είναι προφανές ότι αυτό το υπόστρωμα δεν υφίσταται σε κόμβους που δεν έχουν τη δυνατότητα να παρέχουν τουλάχιστον μια από τις προαναφερθείσες υπηρεσίες.

Αν και δεν αποτελεί μέρος του προτύπου IEEE 1394, θα πρέπει να αναφερθούμε και στο επίπεδο εφαρμογής. Αυτό χρησιμοποιεί τις υπηρεσίες που προσφέρουν τα επίπεδα του διαύλου για να υλοποιήσει μια συγκεκριμένη λειτουργία. Πρωτόκολλα όπως το SBP (Serial Bus Protocol) και πρότυπα όπως το IEC 61883 προσδιορίζουν αυτό ακριβώς το υψηλότερο επίπεδο, επιτρέποντας τη χρήση ομοειδών συσκευών IEEE 1394 με τις ίδιες μεθόδους και μειώνοντας δραστικά τις ανάγκες για εξειδικευμένο λογισμικό ελέγχου.

Σε όλες τις συσκευές που υποστηρίζουν το πρότυπο IEEE 1394, το φυσικό επίπεδο και το υπόστρωμα διασύνδεσης είναι υλοποιημένα σε αφοσιωμένα ολοκληρωμένα κυκλώματα (dedicated ICs), λόγω των αυστηρών χρονικών ορίων που επιβάλλονται κατά το χειρισμό του διαύλου. Το επίπεδο συναλλαγής μπορεί να είναι υλοποιημένο είτε σε υλικό, είτε σε λογισμικό στον επεξεργαστή του κόμβου.

Στους προσωπικούς υπολογιστές χρησιμοποιούνται συνήθως ελεγκτές IEEE 1394 οι οποίοι υλοποιούν την προδιαγραφή 1394 Open Host Controller Interface (OHCI). Σύμφωνα με αυτήν, το υλικό του ελεγκτή παρέχει το φυσικό επίπεδο και το υπόστρωμα διασύνδεσης, όπως επίσης και κάποιες λειτουργίες υποστήριξης για τα επίπεδα συναλλαγής και διαχείρισης διαύλου, τα οποία όμως κατά τα άλλα υλοποιούνται σε λογισμικό στον επεξεργαστή του συστήματος. Στην περίπτωση των ελεγκτών OHCI είναι αξιοσημείωτη η δυνατότητά τους να εξυπηρετούν αυτόνομα ασύγχρονες αιτήσεις εγγραφής και ανάγνωσης από άλλους κόμβους. Συγκεκριμένα, αποκτούν απευθείας πρόσβαση μέσω DMA στις διευθύνσεις μνήμης της αίτησης μέσω του διαύλου συστήματος του υπολογιστή, χωρίς τη δέσμευση πόρων της κεντρικής μονάδας επεξεργασίας. Ανοίγει έτσι ο δρόμος για μια σειρά από ενδιαφέρουσες εφαρμογές, όπως η αποσφαλμάτωση (debugging) και η επιτήρηση (monitoring) του συστήματος, καθώς και η απομακρυσμένη απευθείας πρόσβαση σε συσκευές του διαύλου συστήματος που διαθέτουν χαρτογραφημένες περιοχές μνήμης (memory mapped I/O).

2.3 Το φυσικό επίπεδο

Ο κύριοι ρόλοι του φυσικού επιπέδου του διαύλου IEEE 1394 είναι η αποστολή και λήψη δυαδικών δεδομένων, η εκτέλεση των λειτουργιών διαιτησίας για την πρόσβαση στο δίαυλο, καθώς και η γεφύρωση των ανώτερων επιπέδων με το φυσικό μέσο μεταφοράς. Στην πλειοψηφία των περιπτώσεων το μέσο αυτό αποτελείται από ηλεκτρικά καλώδια, αλλά η δεύτερη τροπολογία του προτύπου IEEE Std 1394b-2002 επιτρέπει και τη χρήση οπτικών ινών. Το φυσικό μέρος έχει διαφορετικά χαρακτηριστικά ανάμεσα στις δυο δυνατές τοπολογίες, αλλά παρουσιάζει την ίδια σχεδόν συμπεριφορά στα ανώτερα επίπεδα.

2.3.1 Ηλεκτρικές και μηχανικές προδιαγραφές

Η αρχική έκδοση του προτύπου προέβλεπε μόνο τη χρήση χάλκινων ηλεκτρικών καλωδίων. Ο δίαυλος υλοποιείται με τρία ζεύγη καλωδίων, εκ των οποίων τα δυο, που είναι συστραμμένα (Twisted Pair A/B, TPA/TPB), χρησιμοποιούνται για τη μετάδοση των δεδομένων και το τρίτο (VP/VG) για την παροχή ηλεκτρικού ρεύματος σε όσους κόμβους δε διαθέτουν δική τους τροφοδοσία. Το καλώδιο διαθέτει μεταλλικό πλέγμα το οποίο λειτουργεί ως θωράκιση για τη μείωση του ηλεκτρομαγνητικού θορύβου, ενώ ταυτόχρονα παρέχει το δυναμικό αναφοράς για όλο το δίαυλο. Στην τροπολογία IEEE Std 1394b-2002 προστέθηκε ένα ακόμα ζεύγος καλωδίων που παρέχει ισορροπημένα σημεία αναφοράς στα δυο ζεύγη σηματοδοσίας για την υποστήριξη των ταχυτήτων S800, S1600 και S3200, καθώς και ένα επιπλέον καλώδιο για μελλοντική χρήση.

Για την τοπολογία δένδρου οι κόμβοι διαθέτουν συνδετήρες έξι ακροδεκτών, που δίνουν πρόσβαση στα τρία αυτά ζεύγη καλωδίων. Εδώ πρέπει να σημειωθεί ότι, παρά την ύπαρξη του προτύπου, αρκετοί κατασκευαστές επέλεξαν έναν ελαφρύτερο και μικρότερο συνδετήρα τεσσάρων ακροδεκτών για τις φορητές συσκευές τους, αφαιρώντας έτσι το ζεύγος της τροφοδοσίας. Η επιτροπή P1394 αναγνώρισε την de facto διάδοση του συγκεκριμένου συνδετήρα και τον ενσωμάτωσε στην τροπολογία IEEE Std 1394a-2000. Αντιστοίχως, για κόμβους που υλοποιούν το IEEE Std 1394b-2002 και υποστηρίζουν τα νέα χαρακτηριστικά προτείνεται ένας νέος συνδετήρας εννέα ακροδεκτών σε δυο παραλλαγές, μια για θύρες που υποστηρίζουν μόνο τις νέες προδιαγραφές (beta-mode ports) και μια για αυτές που μπορούν να συνδεθούν και με κόμβους που ακολουθούν τις παλαιότερες εκδόσεις του προτύπου (bilingual ports).

Η μεταφορά των δεδομένων στο δίαυλο γίνεται με τη χρήση της τεχνικής μη επιστροφής στο μηδέν (No Return to Zero, NRZ) μέσω δυο ζευγών καλωδίων. Η τεχνική αυτή ορίζει μια διαφορά δυναμικού σε ένα ζεύγος ως το λογικό 1 και την αντίστροφή της ως το λογικό 0 και παρουσιάζει πολύ καλή ανοχή στο θόρυβο. Από τα δυο ζεύγη καλωδίων, το ένα (Data) χρησιμοποιείται για τη μεταφορά των δεδομένων, ενώ το δεύτερο (Strobe) αλλάζει κατάσταση όποτε στο ζεύγος Data μεταδίδονται δυο ίδια διαδοχικά δυαδικά ψηφία. Αυτό επιτρέπει την ανάκτηση ενός σταθερού σήματος χρονισμού με μια απλή διαδικασία αποκωδικοποίησης της κατάστασης των δυο ζευγών. Οι υποστηριζόμενοι ρυθμοί μετάδοσης στο IEEE

Std 1394-1995 είναι 24.576 και 49.152 Mbit/sec για την τοπολογία αρτηρίας και 98.304, 196.608 και 393.216 Mbit/sec για την τοπολογία δένδρου. Οι ταχύτητες αυτές είναι πιο γνωστές με τις κωδικές ονομασίες S25, S50, S100, S200 και S400, ενώ οι ακριβείς τιμές τους επιλέχθηκαν για λόγους συμβατότητας με το πρωτόκολλο ISDN, καθώς αποτελούν ακέραια πολλαπλάσια του βασικού ρυθμού μεταφοράς δεδομένων του τελευταίου.

Με την τροπολογία του 2002 προστίθενται οι ταχύτητες S800, S1600 και S3200, αλλά για την υποστήριξη αυτών χρησιμοποιείται η κωδικοποίηση 8B/10B που συναντάται και στα δίκτυα υπολογιστών τύπου Ethernet και όχι η μέθοδος NRZ. Με την κωδικοποίηση 8B/10B (ρυθμός βήτα - beta mode) επιτρέπεται η χρήση των δυο ζευγών αγωγών TPA και TPB του διαύλου για τη δημιουργία μιας αμφίδρομης (full-duplex) σύνδεσης ανάμεσα στους κόμβους, με ένα ζεύγος ανά κατεύθυνση. Αυτή η τεχνική προσφέρει δραστικά καλύτερες επιδόσεις από την ημιαμφίδρομη (half-duplex) σύνδεση που επιτυγχάνεται με τη χρήση της κωδικοποίησης NRZ, ουσιαστικά διπλασιάζοντας το διαθέσιμο εύρος ζώνης, αν και οι κανόνες της διαιτησίας δεν επιτρέπουν την πλήρη αξιοποίησή του, παρά μόνο σε ορισμένες ειδικές περιπτώσεις. Για το λόγο αυτό στη δεύτερη τροπολογία IEEE Std 1394b-2002 η ταχύτητα S400 ορίζεται και για λειτουργία σε ρυθμό βήτα, γεφυρώνοντας έτσι το χάσμα ανάμεσα στην απλή S400 και την S800.

Τα ηλεκτρικά χαρακτηριστικά του διαύλου έχουν σχεδιαστεί έτσι ώστε το λογικό 1 να υπερτερεί αν δυο κόμβοι εκπέμψουν διαφορετικά δυαδικά ψηφία ταυτόχρονα. Αυτό χρησιμοποιείται κατά την περίοδο της διαιτησίας πριν από την αποστολή πακέτων, που είναι και το μόνο χρονικό διάστημα κατά το οποίο μπορεί να υπάρξει σύγκρουση μεταδόσεων. Το κύκλωμα οδήγησης του φυσικού μέσου μπορεί να βρίσκεται είτε σε κατάσταση λήψης (Z), είτε σε κατάσταση εκπομπής (0 ή 1). Τα κυκλώματα λήψης είναι πάντοτε ενεργά, κάτι που επιτρέπει τον εντοπισμό συγκρούσεων.

Χρησιμοποιούνται δυο διαφορετικές τεχνικές για το διαχωρισμό των δεδομένων από τα σήματα ελέγχου. Καταρχάς η διαφορά του μέσου όρου (κέντρου) των σημάτων στα ζεύγη TPA και TPB από την τάση αναφοράς υποδηλώνει τη μέγιστη υποστηριζόμενη ταχύτητα μετάδοσης και την παρουσία ή απουσία του κόμβου στην άλλη άκρη της σύνδεσης. Πέρα από αυτό, το φυσικό επίπεδο ενός κόμβου μπορεί να οδηγήσει τα ζεύγη TPA και TPB με συγκεκριμένες τιμές για διάστημα μεγαλύτερο της περιόδου χρονισμού, δηλαδή χωρίς να ακολουθεί τη σχέση Data/Strobe που ισχύει κατά τη μετάδοση δεδομένων. Με αυτό τον τρόπο ορίζονται τα όρια των πακέτων και μεταφέρονται σήματα ελέγχου, όπως αυτά της αρχικοποίησης του διαύλου και της διαιτησίας.

Η μέγιστη απόσταση ανάμεσα σε δυο κόμβους είναι 4.5m με τη χρήση χάλκινων καλωδίων για την επίτευξη μέγιστων επιδόσεων, αλλά με βάση το IEEE 1394b μπορούν επίσης να χρησιμοποιηθούν οπτικές ίνες σε αποστάσεις από δύο έως και εκατό μέτρα, καθώς και καλώδια UTP (Unshielded Twisted Pair) κατηγορίας 5 σε αποστάσεις έως 100m σε ταχύτητα S100. Σε τοπολογία δένδρου επιτρέπονται έως και 16 βήματα ανάμεσα σε δυο κόμβους, δίνοντας έτσι μέγιστες αποστάσεις 72m και 1600m αντιστοίχως.

Για την τοπολογία αρτηρίας δεν υπάρχει κάποια συγκεκριμένη προδιαγραφή για την φυσική και ηλεκτρική υλοποίηση του διαύλου. Η τοπολογία αυτή προορίζεται

κυρίως για να συμπληρώνει τη λειτουργία ενός παράλληλου διαύλου και για το λόγο αυτό το πρότυπο IEEE Std 1394-1995 προτείνει να ακολουθεί τα ηλεκτρικά χαρακτηριστικά του διαύλου που συνοδεύει για λόγους ομοιομορφίας. Ο δίαυλος αποτελείται από ένα μόνο ζεύγος αγωγών για τα σήματα Data και Strobe, με τα σήματα αναφοράς να προέρχονται από το υπόλοιπο σύστημα. Η φυσική διασύνδεση γίνεται μέσω των δεσμευμένων ακροδεκτών που προβλέπουν οι προδιαγραφές πολλών παράλληλων διαύλων για χρήση από βοηθητικούς σειριακούς διαύλους.

2.3.2 Τοπολογία δένδρου

Το φυσικό επίπεδο ενός κόμβου στην τοπολογία δένδρου αποτελείται από τέσσερα βασικά τμήματα:

- Μια ή περισσότερες θύρες, που αναλαμβάνουν τη διασύνδεση με το φυσικό μέσο που χρησιμοποιείται
- Κυκλώματα συγχρονισμού που φροντίζουν για τη διαφανή γεφύρωση ανάμεσα στο χρονισμό του διαύλου και τον τοπικό χρονιστή του κόμβου
- Κυκλώματα για την πραγματοποίηση των λειτουργιών διαιτησίας, συμπεριλαμβανομένης της βασικής αρχικοποίησης και αυτορρύθμισης του διαύλου
- Κωδικοποιητές και αποκωδικοποιητές για τη σηματοδοσία NRZ ή 8B/10B

Η εικονική επίπεδη τοπολογία αρτηρίας που παρουσιάζεται στα ανώτερα επίπεδα επιτυγχάνεται με κάθε κόμβο να αναμεταδίδει τα δεδομένα που λαμβάνει από κάθε θύρα σε όλες τις υπόλοιπες, αφού πρώτα τα συγχρονίσει με το σήμα χρονισμού της κάθε σύνδεσης σημείο-προς-σημείο. Ο συγχρονισμός αυτός επιτυγχάνεται σε δυο στάδια χρησιμοποιώντας τον τοπικό χρονιστή του κόμβου ως ενδιάμεσο βήμα. Τα καλώδια τροφοδοσίας επιτρέπουν στο φυσικό επίπεδο να παραμένει σε λειτουργία ακόμα και αν ο κόμβος στον οποίο ανήκει δεν τροφοδοτείται, διατηρώντας έτσι την ακεραιότητα του διαύλου. Για το λόγο αυτό ο συνδετήρας τεσσάρων ακροδεκτών θα πρέπει κανονικά να χρησιμοποιείται μόνο για τη σύνδεση κόμβων-φύλλων, ώστε να εξασφαλίζεται η παροχή ενέργειας σε όλους τους ενδιάμεσους κόμβους.

Μετά από κάθε προσθήκη ή αφαίρεση ενός κόμβου ένα σήμα επανεκκίνησης οδηγεί το δίαυλο σε μια περίοδο αυτόματης αναδιάταξης, κατά την οποία διαγράφονται τα στοιχεία της προηγούμενης τοπολογίας, εντοπίζονται οι διασυνδέσεις ανάμεσα σε όλους τους κόμβους, προσδιορίζονται οι οι κωδικοί αριθμοί ταυτοποίησής τους, και επιλέγεται η ρίζα του δένδρου και οι κόμβοι με ειδικές αρμοδιότητες στη διαχείριση του διαύλου. Η ολοκλήρωση της φάσης αυτορρύθμισης είναι απαραίτητη πριν καταστεί δυνατή η χρήση του συστήματος διαιτησίας για την αποστολή πακέτων από το υπόστρωμα διασύνδεσης στο δίαυλο. Η αυτορρύθμιση του διαύλου αποτελείται από τρία στάδια:

- Αρχικοποίηση της κατάστασης του διαύλου
- Προσδιορισμός της δενδρικής δομής και επιλογή της ρίζας
- Ταυτοποίηση των κόμβων

Κατά το στάδιο αρχικοποίησης του διαύλου παύουν οι λειτουργίες των ανώτερων επιπέδων και απορρίπτονται οι πληροφορίες και οι δεσμεύσεις πόρων από το προηγούμενο στιγμιότυπο του διαύλου. Κάθε κόμβος ανιχνεύει την κατάσταση όλων των θυρών του και διαπιστώνει αν αποτελεί φύλλο, με μόνο μια διασυνδεδεμένη θύρα, ή αν ανήκει στο εσωτερικό κάποιου κλάδου με περισσότερες από μία ενεργές θύρες.

Ο προσδιορισμός της δενδρικής δομής γίνεται με βάση τα στοιχεία που προέκυψαν κατά το προηγούμενο στάδιο. Η διαδικασία ξεκινά με κάθε κόμβο-φύλλο

να αποστέλλει στη μοναδική συνδεδεμένη θύρα του ένα σήμα ελέγχου με το οποίο ενημερώνει το φυσικό επίπεδο του άλλου άκρου εάν σε αυτή τη διασύνδεση βρίσκεται ένας κλάδος με απογόνους αυτού. Στη συνέχεια κάθε ενδιάμεσος κόμβος του δένδρου περιμένει για κάποιο χρονικό διάστημα προκειμένου ενημερωθεί πλήρως από όλους τους απογόνους του και στη συνέχεια αποστέλλει στην ενεργή θύρα στην οποία ο ίδιος δεν έχει απογόνους ένα σήμα που πληροφορεί για την παρουσία ενός κλάδου απογόνων. Ο κόμβος ρίζα είναι τελικά αυτός ο οποίος λαμβάνει από όλες τις θύρες του σήμα παρουσίας απογόνων και προφανώς ο ίδιος δεν έχει γονέα. Είναι πιθανόν δύο ή περισσότεροι κόμβοι να βρεθούν σε κατάσταση ανταγωνισμού για το ρόλο της ρίζας (root contention). Σε αυτή την περίπτωση υποχωρούν όλοι και ξαναπροσπαθούν μετά την πάροδο τυχαίου χρονικού διαστήματος.

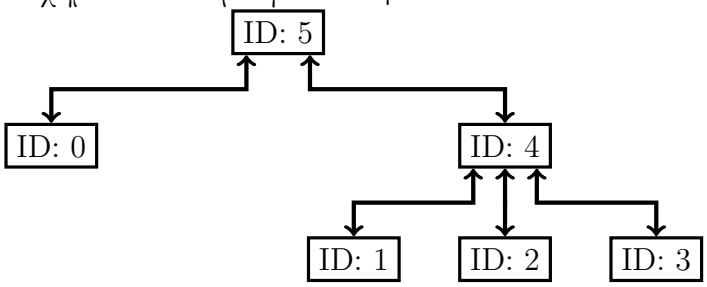
Συνήθως ο κόμβος ο οποίος περιμένει περισσότερο για τη λήψη σημάτων παρουσίας απογόνων στις θύρες του είναι αυτός που γίνεται τελικά η ρίζα, καθώς τα σήματα αυτά προωθούνται από τα φύλλα προς αυτόν. Αυτό επιτρέπει στο υπόστρωμα διαχείρισης του διαύλου να επιλέξει εκ των προτέρων τον κόμβο ρίζα με τη χρήση ενός ειδικού πακέτου ρύθμισης του φυσικού του επιπέδου που τον καθοδηγεί να αυξήσει σημαντικά το χρόνο αυτό αναμονής. Ο προκαθορισμός της ρίζας ενδέχεται να είναι σημαντικός για λόγους βελτιστοποίησης της τοπολογίας του διαύλου ή απλά για την επιλογή του ικανότερου κόμβου στο ρόλο της ρίζας. Με βάση το πρότυπο IEEE Std 1394-1995, για παράδειγμα, όταν υπάρχουν κόμβοι με δυνατότητα παροχής κύκλων χρονισμού για ισόχρονες συναλλαγές (Cycle Master, CM), τότε ένας από αυτούς πρέπει να γίνεται τελικά η ρίζα.

Το τελευταίο στάδιο για τη διαδικασία αυτοδιαμόρφωσης του διαύλου είναι η φάση ταυτοποίησης και αυτοπεριγραφής (self identification, Self-ID) των κόμβων. Σε αυτό το στάδιο κάθε κόμβος αποστέλλει στην πρώτη ευκαιρία ένα ή περισσότερα ειδικά πακέτα με τα οποία παρέχει πληροφορίες για τις δυνατότητές του και την κατάσταση των θυρών του, καθώς και τον αριθμό κόμβων που έστειλαν πακέτα Self-ID πριν καταφέρει να στείλει τα δικά του πακέτα. Ο αριθμός αυτός, που προφανώς βρίσκεται στο διάστημα [0-62], αποτελεί τον κωδικό αριθμό ταυτοποίησης του κόμβου αυτού.

Στην αρχή της διαδικασίας η ρίζα στέλνει διαδοχικά στις ενεργές θύρες της ένα σήμα εκκίνησης και κάθε κόμβος το επαναλαμβάνει διαδοχικά στις θύρες των απογόνων του. Όταν ένα φύλλο της δενδρικής τοπολογίας λάβει το σήμα αυτό αποκρίνεται με τα πακέτα Self-ID που το περιγράφουν, ενώ κάθε άλλος κόμβος στέλνει τα δικά του πακέτα όταν έχει λάβει τις αποκρίσεις Self-ID από όλους τους απογόνους του. Τα πακέτα Self-ID λαμβάνονται από όλους τους κόμβους και καταμετρώνται, ώστε να προσδιοριστούν οι αριθμοί ταυτοποίησής τους. Η ρίζα του δένδρου έχει πάντα τον υψηλότερο αριθμό ταυτοποίησης, καθώς τα πακέτα Self-ID που στέλνει ακολουθούν αυτά όλων των υπόλοιπων κόμβων.

Με την τοπολογία του διαύλου πλήρως προσδιορισμένη, μπορούν πλέον να χρησιμοποιηθούν οι μηχανισμοί διαιτησίας του φυσικού επιπέδου για την αποστολή πακέτων. Προκειμένου να λάβει ένας κόμβος τον έλεγχο του διαύλου, ώστε να μπορέσει να μεταδόσει δεδομένα, στέλνει μια αίτηση για τον έλεγχο του διαύλου στο γονέα του, ο οποίος, αφού αποκλείσει τους υπόλοιπους απογόνους του, αναμεταδίδει την αίτηση προς το δικό του γονέα. Μόλις η αίτηση φτάσει στη ρίζα, αυτή παραχωρεί το δίαυλο ενημερώνοντας ταυτόχρονα όλους τους κόμβους. Σε

Σχήμα 2: Δενδρική τοπολογία διαύλου IEEE 1394



περίπτωση που δύο ή περισσότεροι απόγονοι ενός κόμβου επιθυμούν ταυτόχρονα τον έλεγχο του διαύλου, τότε ο κόμβος με το μεγαλύτερο αριθμό ταυτοποίησης υπερτερεί. Εδώ προκύπτει άμεσα ο κίνδυνος κάποιοι κόμβοι να αποκλείουν συστηματικά άλλους με μικρότερο αριθμό ταυτοποίησης.

Μια ειδική περίπτωση είναι αυτή των πακέτων επιβεβαίωσης λήψης ACK, τα οποία αποστέλλονται χωρίς την πραγματοποίηση διαιτησίας, ουσιαστικά υπό την αιγίδα του κόμβου προέλευσης του πακέτου που επιβεβαιώνεται, αυξάνοντας έτσι την απόδοση του διαύλου. Αυτή η τεχνική ονομάζεται άμεση διαιτησία (immediate arbitration). Στην περίπτωση ασύγχρονων συναλλαγών που αποτελούνται από δυο στάδια (αίτηση-απόκριση) είναι δυνατόν το πακέτο ACK που αντιστοιχεί στην αίτηση να το ακολουθήσει άμεσα το πακέτο της απόκρισης (concatenated response), βελτιώνοντας έτσι το χρόνο καθυστέρησης για την ολοκλήρωση της συναλλαγής.

Συνεπώς για την αποστολή ενός πακέτου απαιτούνται τα ακόλουθα στάδια:

- Διαδικασία διαιτησίας για τον έλεγχο του διαύλου
- Αποστολή πακέτου
- Αναμονή για το πακέτο επιβεβαίωσης της λήψης
- Περίοδος αδράνειας

Τα τρία πρώτα στάδια συνοψίζονται στο κείμενο του προτύπου με τον όρο υποδράση (subaction). Η διάρκεια της περιόδου αδράνειας μπορεί να είναι ένα διάστημα είτε διαχωρισμού υποδράσεων (subaction gap), είτε έναρξης περιόδου αμεροληψίας (arbitration reset gap). Και τα δυο αυτά διαστήματα καθορίζονται από την τιμή της μεταβλητής gap_count, η οποία μπορεί να αλλάξει με ένα πακέτο ρυθμίσεων φυσικού επιπέδου (PHY configuration packet):

$$arb_reset_gap \approx (52 + gap_count * 32)/BASE_RATE$$
$$subaction_gap \approx (28 + gap_count * 16)/BASE_RATE$$

Η μεταβλητή gap_count έχει αρχικά την τιμή 63, που δίνει για τα παραπάνω διαστήματα χρόνους 21 μs και 10 μs περίπου. Σημαντική βελτίωση στις επιδόσεις του διαύλου μπορεί να προκύψει με τη μείωση της τιμής gap_count στο ελάχιστο απαραίτητο (gap count optimisation), ανάλογα με τη μέγιστη απόσταση ανάμεσα σε

δυο κόμβους, κατά τις συστάσεις του προτύπου IEEE Std 1394-1995. Η φυσική αναδιάταξη των κόμβων μπορεί να μειώσει αυτές τις αποστάσεις και να επιφέρει ακόμα μεγαλύτερη βελτίωση. Η επίδραση των περιόδων αδράνειας στις επιδόσεις του διαύλου γίνεται σημαντικά μεγαλύτερη όταν χρησιμοποιούνται μικρότερα πακέτα και εισάγονται αναλογικά περισσότερα διαστήματα.

Προκειμένου να εξασφαλιστεί η αμεροληψία κατά την παραχώρηση του διαύλου, κανένας κόμβος δεν επιχειρεί να λάβει τον έλεγχο του διαύλου πάνω από μια φορά κατά τη διάρκεια ενός χρονικού διαστήματος, που ονομάζεται διάστημα αμεροληψίας (fairness interval). Το τέλος του διαστήματος αυτού και η έναρξη του επόμενου προσδιορίζεται από μια περίοδο αδράνειας ίση ή μεγαλύτερη με το διάστημα έναρξης περιόδου αμεροληψίας, το οποίο είναι σημαντικά μεγαλύτερο από το χρόνο διαχωρισμού υποδράσεων. Κατά συνέπεια, όταν πολλαπλοί κόμβοι έχουν πακέτα προς μετάδοση επιχειρούν να στείλουν ο καθένας από ένα αμέσως μόλις παρέλθει μια περίοδος διαχωρισμού υποδράσεων. Προκειμένου να απουσιάσει κάθε δραστηριότητα για μεγαλύτερο χρόνο πρέπει όλοι οι κόμβοι να έχουν αποστείλει από ένα πακέτο και να αναμένουν πλέον το επόμενο διάστημα αμεροληψίας, εξασφαλίζοντας έτσι ότι κανένας από αυτούς δεν θα μονοπωλεί το δίαυλο.

Δυο ιδιαίτερες περιπτώσεις διαιτησίας είναι αυτές των ισόχρονων πακέτων και των πακέτων χρονισμού Cycle Start από την υπηρεσία Cycle Master της ρίζας. Η εκπομπή των ισόχρονων πακέτων πραγματοποιείται μετά από ένα διάστημα αδράνειας (isochronous gap) σημαντικά μικρότερο του ελάχιστου χρόνου ανάμεσα σε δυο υποδράσεις και συνεπώς έχει πάντα προτεραιότητα. Αντίστοιχα, τα πακέτα Cycle Start μεταδίδονται αμέσως μετά το πρώτο διάστημα υποδράσεων που παρουσιάζεται κάθε φορά, με τον κόμβο-ρίζα να τους δίνει προτεραιότητα απέναντι σε κάθε ασύγχρονο πακέτο.

Η διαδικασία της διαιτησίας δεν είναι απαραίτητη όταν ένας κόμβος έχει ήδη τον έλεγχο του διαύλου, ενώ δεν πραγματοποιείται επιβεβαίωση της λήψης για ασύγχρονα πακέτα που απευθύνονται σε όλους τους κόμβους (broadcast packets) ή κατά την τέλεση ισόχρονων συναλλαγών.

Η τροπολογία IEEE Std 1394a-2000 επιφέρει σημαντικές βελτιώσεις στον τομέα της διαιτησίας. Καταρχάς επιτρέπει σε έναν κόμβο να επιχειρήσει να λάβει τον έλεγχο του διαύλου πολλαπλές φορές (priority arbitration) εντός του ίδιου διαστήματος αμεροληψίας, αν πρέπει να μεταφέρει χρονικά κρίσιμα δεδομένα. Αυτής της μορφής η διαιτησία προτεραιότητας δεν παρουσιάζει προβλήματα συμβατότητας και προορίζεται κατά κύριο λόγο για τις αποκρίσεις απομακρυσμένων κόμβων στις συναλλαγές που ξεκινά το υπόστρωμα συναλλαγής, αλλά μπορεί να χρησιμοποιηθεί και σε άλλες περιπτώσεις.

Μια δεύτερη σημαντική βελτίωση είναι η δυνατότητα απαλοιφής του διαστήματος διαχωρισμού υποδράσεων μετά από ασύγχρονα πακέτα, με το πακέτο ACK που τα ακολουθεί να αποτελεί το σήμα για την έναρξη του επόμενου πακέτου (arbitration acceleration). Δυστυχώς η μέθοδος αυτή παρεμβάλλεται στη λειτουργία εκπομπής των πακέτων Cycle Start καθώς η ανυπαρξία διαστημάτων υποδράσεων δεν επιτρέπει την μετάδοσή τους την πρέπουσα χρονική στιγμή, παρότι θα έπρεπε να έχουν απόλυτη προτεραιότητα. Για την επίλυση αυτού του προβλήματος απαιτείται ένας σύνθετος μηχανισμός που απενεργοποιεί την επιτάχυνση διαιτησίας επιλεκτικά κατά τα χρονικά διαστήματα εκπομπής των πακέτων χρονισμού. Μια

παρόμοια συνέπεια είναι και ο αποκλεισμός από το δίαυλο των κόμβων που έχουν υλοποιηθεί με βάση την αρχική έκδοση του προτύπου IEEE Std 1394-1995 και που επίσης απαιτούν την ύπαρξη διαστημάτων αδράνειας πριν από την έναρξη της διαιτησίας.

Με την τροπολογία IEEE Std 1394b-2002 και την εισαγωγή των αμφίδρομων συνδέσεων ρυθμού βήτα (beta mode), προτάθηκε μια νέα μέθοδος διαιτησίας που εκμεταλλεύεται και τις δυο κατευθύνσεις των συνδέσεων. Κάθε κόμβος που στέλνει ένα πακέτο αποκτά θέση ελεγκτή στο δίαυλο (Bus Owner/Supervisor/Selector, BOSS) και αναμένει αιτήσεις διαιτησίας από την κατεύθυνση εισερχομένων των θυρών του. Αν δηλαδή θεωρήσουμε τον κόμβο BOSS ως τη ρίζα ενός δένδρου, τότε τα πακέτα δεδομένων διαδίδονται από τη ρίζα προς τα φύλλα και ταυτόχρονα οι αιτήσεις διαιτησίας ανέρχονται σταδιακά από τα φύλλα προς τη ρίζα. Με αυτό τον τρόπο παραλληλίζονται οι διαδικασίες διαιτησίας με την εκπομπή πακέτων, απαλείφοντας τα διαστήματα αδράνειας και αυξάνοντας δραστικά τις επιδόσεις του διαύλου. Εδώ πρέπει να σημειωθεί ότι, λόγω των περιορισμών της διαιτησίας και του ότι ο δίαυλος IEEE 1394 είναι ουσιαστικά ένα δίκτυο εκπομπής (broadcast network) και όχι μεταγωγής (switched network), δεν είναι δυνατή η παρουσία πακέτων δεδομένων και στις δυο κατευθύνσεις μιας σύνδεσης - σε κάθε περίπτωση ένα από τα δύο κανάλια θα υπολειτουργεί.

2.3.3 Πακέτα φυσικού επιπέδου

Τα πακέτα φυσικού επιπέδου (PHY packets) χρησιμοποιούνται για τη ρύθμιση των χαρακτηριστικών του διαύλου και στη λήψη και επεξεργασία τους δεν επεμβαίνουν τα ανώτερα επίπεδα των κόμβων. Αποτελούνται από δυο τετράδες (64 bit συνολικά), με τη δεύτερη να είναι η λογική αντιστροφή της πρώτης, ώστε να μπορούν να ανιχνευτούν τυχόν σφάλματα. Αποστέλλονται πάντοτε με το βασικό ρυθμό μετάδοσης δεδομένων (base rate), που για την τοπολογία δένδρου είναι ο S100. Αυτό επιτρέπει την επιτυχή μεταφορά αυτών των πακέτων πριν ακόμα οι κόμβοι αλληλοενημερωθούν για τις ταχύτητες που υποστηρίζουν. Η γενική δομή τους είναι η ακόλουθη:

Σχήμα 3: Μορφή πακέτου φυσικού επιπέδου

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
pid		Phy_ID						Λοιπά πεδία πακέτου																							
Λογική αντιστροφή πρώτης τετράδας																															

Πίνακας 1: Πεδία πακέτου φυσικού επιπέδου

Πεδίο	Μέγεθος	Περιγραφή
pid	2	Τύπος πακέτου φυσικού επιπέδου
Phy_ID	6	Φυσική διεύθυνση κόμβου στο δίαυλο

Με την τροπολογία IEEE Std 1394a-2000 υπάρχουν πλέον οι ακόλουθοι τύποι πακέτων PHY:

- Πακέτα αυτοπεριγραφής (Self-ID)
- Πακέτα ενεργοποίησης (Link-on)
- Πακέτα ρύθμισης (PHY Configuration)
- Εκτεταμένα πακέτα φυσικού επιπέδου (Extended PHY packets)

2.3.3.1 Πακέτα αυτοπεριγραφής (Self-ID)

Ακολουθώντας το πρότυπο IEEE Std 1394-1995, κάθε κόμβος μιας τοπολογίας δένδρου που λαμβάνει το κατάλληλο σήμα από το γονέα του κατά τη φάση της αυτοδιαμόρφωσης του διαύλου οφείλει να αποκριθεί με ένα ή περισσότερα πακέτα Self-ID. Το πρώτο πακέτο Self-ID από έναν κόμβο μπορεί να μεταφέρει πληροφορίες για τρεις θύρες, ενώ μπορούν να ακολουθήσουν έως και τρία συμπληρωματικά πακέτα που περιέχουν στοιχεία για οκτώ θύρες το καθένα, επιτρέποντας έτσι ένα μέγιστο αριθμό 27 θυρών ανά κόμβο. Η μορφή του πρώτου πακέτου Self-ID είναι η ακόλουθη:

Σχήμα 4: Μορφή πρώτου πακέτου Self-ID

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
10			Phy_ID					0	L	gap_count						sp		del		c	pwr			p0		p1		p2		i	m
Λογική αντιστροφή πρώτης τετράδας																															

Πίνακας 2: Πεδία πρώτου πακέτου Self-ID

Πεδίο	Μέγεθος	Περιγραφή
10	2	Χαρακτηριστικός κωδικός των πακέτων Self-ID
Phy_ID	6	Φυσική διεύθυνση του κόμβου στο δίαυλο
L	1	Ψηφίο που υποδεικνύει αν ο κόμβος έχει ενεργά επίπεδα διασύνδεσης και συναλλαγής
gap_count	6	Τρέχουσα τιμή βάσης για τα διαστήματα αδράνειας του διαύλου
sp	2	Μέγιστη υποστηριζόμενη ταχύτητα μετάδοσης δεδομένων
del	2	Μέγιστη καθυστέρηση αναμετάδοσης από θύρα σε θύρα
c	1	Ψηφίο που υποδεικνύει αν ο κόμβος είναι υποψήφιος διαχειριστής διαύλου ή ισόχρονων πόρων (contender)
pwr	3	Ενεργειακά χαρακτηριστικά του κόμβου
p0 ... p2	2	Κατάσταση θυρών
i	1	Ψηφίο που υποδεικνύει αν ο κόμβος αυτός προκάλεσε την τρέχουσα επανεκκίνηση του διαύλου
m	1	Ψηφίο που υποδηλώνει ότι ακολουθούν συμπληρωματικά πακέτα Self-ID

Τα συμπληρωματικά πακέτα Self-ID είναι δομημένα ως εξής:

Σχήμα 5: Μορφή συμπληρωματικού πακέτου Self-ID

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
10			Phy_ID					0	n			rsv		pa		pb		pc		pd		pe		pf		pg		ph		r	m
Λογική αντιστροφή πρώτης τετράδας																															

Πίνακας 3: Πεδία συμπληρωματικού πακέτου Self-ID

Πεδίο	Μέγεθος	Περιγραφή
n	3	Αριθμός συμπληρωματικού πακέτου Self-ID, στο διάστημα [0-2]
rsv	2	Δεσμευμένο για μελλοντική χρήση
pa ... pg	2	Κατάσταση θυρών που περιγράφονται από αυτό το πακέτο
r	1	Δεσμευμένο για μελλοντική χρήση

Στην τροπολογία IEEE Std 1394a-2000 η μορφή των πακέτων Self-ID αλλάζει σε κάποιο βαθμό, αν και προτείνεται οι υλοποιήσεις να γίνουν με τέτοιο τρόπο ώστε να μην υπάρχουν προβλήματα συμβατότητας. Ο μέγιστος αριθμός θυρών ενός κόμβου περιορίζεται στις 16, ενώ το πεδίο del είναι πλέον δεσμευμένο για μελλοντική χρήση, με την τροπολογία IEEE Std 1394b-2002 να το μετονομάζει τελικά σε brdg και να ορίζει πως προορίζεται ειδικά για υλοποιήσεις του προτύπου IEEE Std 1394.1-2004 IEEE Standard for High Performance Serial Bus Bridges, που οφείλουν να ακολουθούν οι γέφυρες (bridges) μεταξύ διαύλων IEEE 1394.

2.3.3.2 Πακέτα ενεργοποίησης (Link-on)

Τα πακέτα φυσικού επιπέδου Link-on ζητούν από τον κόμβο που τα λαμβάνει να ενεργοποιήσει τα ανώτερα επίπεδά του όταν αυτά είναι ανενεργά. Με αυτό τον τρόπο κόμβοι με υψηλή κατανάλωση ενέργειας μπορούν να τη μειώσουν, πέφτοντας σε κατάσταση νάρκης όταν δεν χρησιμοποιούνται. Τα πακέτα Link-on έχουν την ακόλουθη μορφή:

Σχήμα 6: Μορφή πακέτου Link-on

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
01		Phy_ID						0000 0000 0000 0000 0000 0000																							
Λογική αντιστροφή πρώτης τετράδας																															

Ένας κόμβος με ενεργό μόνο το φυσικό του επίπεδο, το οποίο μπορεί και να τροφοδοτείται από το δίαυλο, εξακολουθεί να λειτουργεί ως επαναλήπτης (repeater) δεδομένων και δεν δημιουργεί προβλήματα λειτουργίας. Στην αγορά υπάρχουν διαθέσιμες συσκευές (repeaters/hubs) με πολύ απλές υλοποιήσεις του προτύπου IEEE 1394 και μόνο σκοπό την παροχή περισσότερων θυρών ή και την επέκταση των μέγιστων αποστάσεων για τη σύνδεση κόμβων στο δίαυλο.

2.3.3.3 Πακέτα ρύθμισης (PHY Configuration)

Με τη χρήση των πακέτων ρύθμισης του φυσικού επιπέδου πραγματοποιούνται δυο σημαντικές διεργασίες βελτιστοποίησης του διαύλου:

1. Ρύθμιση της διάρκειας των διαστημάτων αδράνειας στη μικρότερη δυνατή τιμή (gap count optimisation)

2. Προεπιλογή του κόμβου που θα λάβει της θέση της ρίζας μετά την επόμενη αναδιαμόρφωση του διαύλου, έτσι ώστε ρίζα να είναι ο ικανότερος κόμβος. Συχνά αυτό απαιτείται για να επιλέγεται ρίζα που να μπορεί να λειτουργήσει ως παροχέας χρονισμού για ισόχρονες συναλλαγές (Cycle Master capable).

Τα πακέτα PHY Configuration έχουν την ακόλουθη μορφή:

Σχήμα 7: Μορφή πακέτου PHY Configuration

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00		root_ID						R	T	gap_count						0000 0000 0000 0000															
Λογική αντιστροφή πρώτης τετράδας																															

Πίνακας 4: Πεδία πακέτου PHY Configuration

Πεδίο	Μέγεθος	Περιγραφή
root_ID	6	Φυσική διεύθυνση του μελλοντικού κόμβου-ρίζα, χρησιμοποιείται μόνο όταν R = 1
R	1	Αν R = 1, τότε ενεργοποιείται ο μηχανισμός επιλογής του κόμβου-ρίζα
T	1	Αν T = 1, τότε σε κάθε κόμβο η μεταβλητή gap_count θα λάβει την τιμή του ομώνυμου πεδίου.
gap_count	6	Η τιμή για την ομώνυμη μεταβλητή του φυσικού επιπέδου

Όταν R = 1 ενεργοποιείται στον επιλεγμένο κόμβο το πεδίο force_root ενώ απενεργοποιείται σε κάθε άλλο κόμβο. Ο μηχανισμός επιλογής ρίζας επηρεάζει τη φάση προσδιορισμού της δενδρικής δομής κατά την επόμενη αναδιαμόρφωση του διαύλου (bus reset). Ο κόμβος που έχει το πεδίο force_root ενεργό καθυστερεί σκοπίμως κατά τη φάση αυτή του διαύλου, έτσι ώστε να λάβει τελικά τη θέση της ρίζας, όπως περιγράφεται αναλυτικά και στην παράγραφο 2.3.2.

2.3.3.4 Εκτεταμένα πακέτα φυσικού επιπέδου (Extended PHY packets)

Τα εκτεταμένα πακέτα φυσικού επιπέδου αποτελούν επέκταση που προτυποποιήθηκε με την τροπολογία IEEE Std 1394a-2000. Η δομή τους βασίζεται στα πακέτα ρύθμισης που είδαμε σε προηγούμενη παράγραφο, κάνοντας χρήση του συνδυασμού R = 0, T = 0 που ήταν δεσμευμένος στην αρχική έκδοση του προτύπου.

Σχήμα 8: Μορφή εκτεταμένου πακέτου PHY

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00	Phy_ID							00	type																						
Λογική αντιστροφή πρώτης τετράδας																															

Πίνακας 5: Πεδία εκτεταμένου πακέτου PHY

Πεδίο	Μέγεθος	Περιγραφή
type	4	Τύπος εκτεταμένου πακέτου PHY

2.3.3.4.1 Πακέτο επερώτησης (Ping)

Με την τροπολογία IEEE Std 1394a-2000 η αποστολή των πακέτων Self-ID ενός κόμβου μπορεί πλέον να ζητηθεί ανεξάρτητα από τη διαδικασία αυτορρύθμισης του διαύλου με τη χρήση ενός κατάλληλου πακέτου επερώτησης (Ping) προς το φυσικό του επίπεδο. Το πακέτο Ping έχει τη μορφή του σχήματος που ακολουθεί:

Σχήμα 9: Μορφή πακέτου Ping

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00	Phy_ID							00	0000					00 0000 0000 0000																	
Λογική αντιστροφή πρώτης τετράδας																															

Μια πρόσθετη χρήση του πακέτου επερώτησης είναι η ακριβής μέτρηση του χρόνου απόκρισης ενός κόμβου, ώστε να ρυθμιστούν με βέλτιστο τρόπο οι παράμετροι του διαύλου. Στην αρχική έκδοση του προτύπου διαδικασίες όπως η βελτιστοποίηση των διαστημάτων αδράνειας (gap count optimisation) λάμβαναν υπόψη μόνο την τοπολογία του διαύλου και το μέγιστο αριθμό συνδέσεων ανάμεσα σε δυο κόμβους, υποθέτοντας στη συνέχεια τη χειρότερη δυνατή περίπτωση για το μήκος των αγωγών επικοινωνίας. Με τη μέτρηση των πραγματικών χρόνων απόκρισης μπορούν πλέον να επιλεγούν οι απολύτως βέλτιστες παράμετροι λειτουργίας του διαύλου.

2.3.3.4.2 Πακέτο απομακρυσμένης πρόσβασης (Remote access)

Τα πακέτα απομακρυσμένης πρόσβασης (Remote access) επιτρέπουν την ανάγνωση των εσωτερικών καταχωρητών του φυσικού επιπέδου ενός κόμβου. Κάθε κόμβος που λαμβάνει ένα τέτοιο πακέτο απαντά με ένα αντίστοιχο πακέτο απόκρισης που φέρει τα ζητούμενα δεδομένα.

Σχήμα 10: Μορφή πακέτου Remote access

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00		Phy_ID						00		type				page			port				reg			Δεσμευμένο							
Λογική αντιστροφή πρώτης τετράδας																															

Πίνακας 6: Πεδία πακέτου Remote access

Πεδίο	Μέγεθος	Περιγραφή
Phy_ID	6	Φυσική διεύθυνση κόμβου προορισμού
type	4	1: Ανάγνωση βασικού καταχωρητή 5: Ανάγνωση σελιδοποιημένου καταχωρητή
page	3	Σελίδα καταχωρητών
port	4	Θύρα στην οποία αντιστοιχεί ο προς ανάγνωση καταχωρητής
reg	3	Καταχωρητής προς ανάγνωση

2.3.3.4.3 Πακέτο απόκρισης (Remote reply)

Τα πακέτα απόκρισης (Remote reply) μεταδίδονται ως απάντηση σε κάποιο πακέτο απομακρυσμένης πρόσβασης. Έχουν τη δομή του σχήματος 11.

Σχήμα 11: Μορφή πακέτου Remote reply

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00		Phy_ID						00		type				page			port				reg			data							
Λογική αντιστροφή πρώτης τετράδας																															

Πίνακας 7: Πεδία πακέτου Remote reply

Πεδίο	Μέγεθος	Περιγραφή
Phy_ID	6	Φυσική διεύθυνση κόμβου προέλευσης
data	8	Περιεχόμενα καταχωρητή

2.3.3.4.4 Πακέτο απομακρυσμένης εντολής (Remote command)

Τα πακέτα απομακρυσμένης εντολής (Remote command) ζητούν από τον κόμβο που τα λαμβάνει να τελέσει κάποια συγκεκριμένη λειτουργία. Μετά την εκτέλεση της εντολής ο κόμβος αποκρίνεται με ένα πακέτο επιβεβαίωσης.

Σχήμα 12: Μορφή πακέτου Remote command

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00		Phy_ID						00		1000				000			port				0000 0000								cmnd		
Λογική αντιστροφή πρώτης τετράδας																															

Πίνακας 8: Πεδία πακέτου Remote command

Πεδίο	Μέγεθος	Περιγραφή
Phy_ID	6	Φυσική διεύθυνση κόμβου προορισμού
port	4	Θύρα στην οποία αναφέρεται η εντολή
cmnd	3	Εντολή προς εκτέλεση: 0: Καμία 1: Απενεργοποίηση θύρας 2: Αναστολή λειτουργίας θύρας 4: Μηδενισμός ένδειξης σφάλματος 5: Ενεργοποίηση θύρας 6: Ανάνηψη θύρας

Η πιο σημαντική χρήση των πακέτων αυτών είναι στα πλαίσια των λειτουργιών εξοικονόμησης ενέργειας, καθώς με τη χρήση τους μπορεί να ανασταλεί επιλεκτικά η λειτουργία κάποιων θυρών και των κόμβων που είναι συνδεδεμένοι σε αυτές. Επιτρέπουν επίσης την πλήρη απενεργοποίηση κάποιων θυρών, ουσιαστικά αποσυνδέοντας κάθε συνδεδεμένο κόμβο.

2.3.3.4.5 Πακέτο επιβεβαίωσης (Remote confirmation)

Τα πακέτα επιβεβαίωσης (Remote confirmation) είναι η απόκριση ενός κόμβου στα πακέτα απομακρυσμένης εντολής που απευθύνονται σε αυτόν και έχουν την ακόλουθη μορφή:

Σχήμα 13: Μορφή πακέτου Remote confirmation

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00		Phy_ID						00		1010				000			port				000			f	c	b	d	ok	cmnd		
Λογική αντιστροφή πρώτης τετράδας																															

Πίνακας 9: Πεδία πακέτου Remote confirmation

Πεδίο	Μέγεθος	Περιγραφή
Phy_ID	6	Φυσική διεύθυνση κόμβου προέλευσης
f	1	Ένδειξη σφάλματος (fault)
c	1	Ένδειξη παρουσίας συνδεδεμένου κόμβου στη θύρα (connected)
b	1	Ένδειξη παρουσίας ενεργού κόμβου στη θύρα (bias)
d	1	Ένδειξη απενεργοποιημένης θύρας
ok	1	1 για επιτυχημένη εκτέλεση, 0 σε αντίθετη περίπτωση
cmnd	3	Εντολή που εκτελέστηκε

2.3.3.4.6 Πακέτο ανάνηψης (Resume)

Τα πακέτα ανάνηψης (Resume) αποστέλλονται για να επαναφέρουν σε πλήρη λειτουργία όλες τις συνδεδεμένες θύρες ενός απομακρυσμένου κόμβου που έχουν τεθεί σε καταστολή με προηγούμενες εντολές.

Σχήμα 14: Μορφή πακέτου Resume

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00		Phy_ID						00		1111				00 0000 0000 0000																	
Λογική αντιστροφή πρώτης τετράδας																															

2.3.4 Τοπολογία αρτηρίας

Το φυσικό επίπεδο για την τοπολογία αρτηρίας (backplane environment) παρέχει παρόμοιες υπηρεσίες με αυτό της δενδρικής, αλλά παρουσιάζει αρκετές διαφορές. Η πιο βασική διαφορά είναι ότι δεν υποστηρίζονται οι δυνατότητες αρχικοποίησης και αυτοδιαμόρφωσης της δενδρικής τοπολογίας, με τους κωδικούς ταυτοποίησης των κόμβων να προέρχονται από πηγή εξωτερική ως προς το δίαυλο IEEE 1394, όπως για παράδειγμα από έναν συνοδευόμενο παράλληλο δίαυλο. Η εισαγωγή ενός νέου κόμβου μπορεί και εδώ να γίνει με το σύστημα σε λειτουργία, χωρίς όμως να προκαλεί επανεκκίνηση του διαύλου (bus reset) ή την ευρεία ανταλλαγή πληροφοριών ανάμεσα τους κόμβους που βλέπουμε στο μηχανισμό Self-ID της τοπολογίας δένδρου. Γενικά οι κόμβοι που έχουν υλοποιηθεί για χρήση σε αρτηρία έχουν εξειδικευμένη εφαρμογή, συνήθως σε περιβάλλοντα με πολύ συγκεκριμένες προδιαγραφές, και δεν απαιτούν κάποιο σύνθετο μηχανισμό αρχικοποίησης και ελέγχου του διαύλου.

Ο μηχανισμός αμερόληπτης διαιτησίας είναι όμοιος με αυτόν της τοπολογίας δένδρου: όλοι οι κόμβοι που επιθυμούν να χρησιμοποιήσουν το δίαυλο περιμένουν για την έλευση μιας περιόδου αδράνειας και αποστέλλουν την παρακάτω ακολουθία διαιτησίας (arbitration sequence):

Σχήμα 15: Μορφή ακολουθίας διαιτησίας

0	1	2	3	4	5	6	7	8	9
priority				arbitration_number					

Πίνακας 10: Πεδία ακολουθίας διαιτησίας

Πεδίο	Μέγεθος	Περιγραφή
priority	4	Βαθμός προτεραιότητας
arbitration_number	6	Κωδικός αριθμός διατησίας, συνήθως ίσος με την φυσική διεύθυνση Phy_ID του κόμβου

Η εκπομπή της ακολουθίας πραγματοποιείται από όλους τους κόμβους συγχρονισμένα και κάθε κόμβος που ανιχνεύει στο δίαυλο λογικό επίπεδο 1, ενώ εκπέμπει λογικό 0, υποχωρεί. Κατά συνέπεια ο κόμβος με τη μεγαλύτερη τιμή για την ακολουθία διαιτησίας υπερτερεί πάντοτε των υπολοίπων. Η υψηλότερη προτεραιότητα είναι δεσμευμένη για τα πακέτα χρονισμού Cycle Start, εξασφαλίζοντας την άμεση μετάδοσή τους. Η ισότιμη πρόσβαση στο δίαυλο επιδιώκεται με τη χρήση διαστημάτων αμεροληψίας, αλλά κάθε κόμβος επιτρέπεται να τα αγνοήσει για χρονικά κρίσιμα πακέτα (urgent arbitration). Ωστόσο ο μηχανισμός της κρίσιμης διαιτησίας θέτει σαφή όρια, έτσι ώστε να αποφεύγεται ο αποκλεισμός των άλλων κόμβων από το δίαυλο. Μια παρόμοια δυνατότητα, αλλά σε απλούστερη μορφή (priority arbitration), προστέθηκε με την τροπολογία IEEE Std 1394a-2000 και στο φυσικό επίπεδο της δενδρικής τοπολογίας.

Οι διαφορές ανάμεσα στα φυσικά επίπεδα των δυο τοπολογιών σε κάποιες περιπτώσεις εμφανίζονται και στα υπόλοιπα επίπεδα, με κύρια διαφορά την απουσία του μηχανισμού αρχικοποίησης. Στο χώρο των προσωπικών υπολογιστών οι υλοποιήσεις με τοπολογία αρτηρίας είναι από σπάνιες έως ανύπαρκτες, με συνέπεια πολλά λειτουργικά συστήματα να μην διαθέτουν την αντίστοιχη υποστήριξη, παρότι υποστηρίζουν δενδρικούς διαύλους IEEE 1394. Στο ίδιο πνεύμα, οι τροπολογίες που ακολούθησαν την αρχική έκδοση του προτύπου δεν περιέχουν καμία ιδιαίτερη αναφορά ή βελτίωση για τους διαύλους με δομή αρτηρίας. Για τους λόγους αυτούς δεν θα αναφερθούμε περαιτέρω στην τοπολογία αρτηρίας, δεδομένου ότι και το σύνολο των μετρήσεων για την εργασία αυτή πραγματοποιήθηκε σε δίαυλο με δενδρική τοπολογία.

2.4 Το επίπεδο διασύνδεσης

Το επίπεδο διασύνδεσης (link layer) ευθύνεται για τη μετάδοση πακέτων και την απόκριση με το αντίστοιχο πακέτο επιβεβαίωσης της λήψης, όταν αυτό είναι απαραίτητο. Αναλαμβάνει, δηλαδή, την πραγματοποίηση μεμονωμένων υποδράσεων με χρήση του του φυσικού επιπέδου, υπό την καθοδήγηση του επιπέδου συναλλαγής. Για το σκοπό αυτό πραγματοποιεί τις ακόλουθες λειτουργίες:

- Προετοιμασία και αποστολή ασύγχρονων πακέτων με την πρώτη παραχώρηση του διαύλου
- Περιοδική αποστολή ισόχρονων πακέτων με εγγυημένη μέγιστη καθυστέρηση
- Λήψη και έλεγχος πακέτων
- Αποστολή πακέτων επιβεβαίωσης της λήψης
- Υπολογισμός και επιβεβαίωση του κώδικα ανίχνευσης σφαλμάτων για κάθε πακέτο
- Λειτουργία παροχέα χρονισμού ισόχρονων συναλλαγών (Cycle Master, CM) με τη συστηματική αποστολή πακέτων Cycle Start.

Το πρότυπο IEEE Std 1394-1995 ορίζει μια ιεραρχία πακέτων, όπου η δομή κάθε πακέτου αποτελεί εξειδίκευση μια απλούστερης μορφής με την προσθήκη επιπλέον πεδίων και την παρουσία συγκεκριμένων τιμών σε άλλα. Στην κορυφή αυτής της ιεραρχίας βρίσκονται τα πακέτα φυσικού επιπέδου PHY, τα πακέτα επιβεβαίωσης ACK και τα πρωτεύοντα πακέτα (primary packets) που αποτελούν την υπερκλάση όλων των υπόλοιπων πακέτων:

- Πρωτεύοντα πακέτα (primary packets)
  - Ασύγχρονα πακέτα
    - Ασύγχρονα πακέτα χωρίς φορτίο (no payload)
      - Αίτηση για ανάγνωση τετράδας (quadlet read request)
      - Απόκριση συναλλαγής εγγραφής (write transaction response)
    - Ασύγχρονα πακέτα με φορτίο μιας τετράδας (quadlet payload)
      - Αίτηση για ανάγνωση τμήματος δεδομένων (data block read request)
      - Αίτηση για εγγραφή τετράδας (quadlet write request)
      - Απόκριση συναλλαγής ανάγνωσης τετράδας (quadlet read transaction response)
      - Πακέτο χρονισμού ισόχρονων συναλλαγών Cycle Start
    - Ασύγχρονα πακέτα με φορτίο τμήματος δεδομένων (data block payload)

        - Αίτηση για εγγραφή τμήματος δεδομένων (data block write request)
        - Απόκριση συναλλαγής ανάγνωσης τμήματος δεδομένων (data block read transaction response)
        - Αίτηση συναλλαγής κλειδώματος (lock request)
        - Απόκριση συναλλαγής κλειδώματος (lock transaction response)
    - Ισόχρονα πακέτα
    - Πακέτα ασύγχρονης ροής (asynchronous stream packets)

- Πακέτα επιβεβαίωσης λήψης (ACK)
- Πακέτα φυσικού επιπέδου (PHY)

Κάθε πακέτο που μεταδίδεται μέσω ενός διαύλου IEEE 1394 ανήκει σε κάποια από τις παραπάνω κατηγορίες. Το υπόστρωμα διασύνδεσης επεξεργάζεται όλες τις κατηγορίες πακέτων, εκτός από τα πακέτα φυσικού επιπέδου, παρέχοντας μια σειρά σχετικών υπηρεσιών στα επίπεδα συναλλαγής και διαχείρισης διαύλου.

2.4.1 Πρωτεύοντα πακέτα

Όλα τα πακέτα δεδομένων που διακινούνται στο δίαυλο έχουν την ακόλουθη βασική δομή του πρωτεύοντος πακέτου:

Σχήμα 16: Μορφή πρωτεύοντος πακέτου

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Πρώτη τετράδα επικεφαλίδας																								tcode							
Λοιπές τετράδες επικεφαλίδας (αν υπάρχουν)																															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															
Πρώτη τετράδα τμήματος δεδομένων																															
Λοιπες τετράδες τμήματος δεδομένων																															
								Συμπληρωματικά bytes (padding)																							
Κωδικός ελέγχου σφαλμάτων τμήματος δεδομένων CRC																															

Πίνακας 11: Πεδία πρωτεύοντος πακέτου

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής (Transaction code)

Τα πακέτα αυτά αποτελούνται από μια επικεφαλίδα μιας ή περισσότερων τετράδων και προαιρετικά από ένα τμήμα δεδομένων, μαζί με τους ανάλογους κωδικούς ελέγχου λαθών CRC, οι οποίοι υπολογίζονται χωριστά για τις τετράδες της επικεφαλίδας και του τμήματος δεδομένων. Το επίπεδο συναλλαγής κάθε κόμβου οφείλει να απορρίπτει και να αγνοεί οποιοδήποτε πακέτο ελήφθη με σφάλμα.

Με τον προσδιορισμό συγκεκριμένων πεδίων και σταθερών τιμών στην παραπάνω δομή παράγονται οι μορφές όλων των ασύγχρονων και ισόχρονων πακέτων που ορίζει το πρότυπο IEEE 1394. Το ελάχιστο μέγεθος ενός τέτοιου πακέτου είναι δυο τετράδες, αν και για τις περισσότερες εφαρμογές προτιμώνται μεγαλύτερα πακέτα για την επίτευξη υψηλότερων επιδόσεων. Ο κωδικός συναλλαγής από την πρώτη τετράδα της επικεφαλίδας υποδηλώνει σε ποια υποκατηγορία ανήκει το κάθε πακέτο, επιτρέποντας έτσι την περαιτέρω επεξεργασία του με τον κατάλληλο τρόπο.

2.4.2 Ασύγχρονα πακέτα

Τα ασύγχρονα πακέτα έχουν μέγεθος τουλάχιστον τέσσερις τετράδες και την ακόλουθη εσωτερική δομή:

Σχήμα 17: Μορφή ασύγχρονου πακέτου

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">destination_ID</td><td colspan="4">tl</td><td>rt</td><td colspan="4">tcode</td><td colspan="4">pri</td></tr>
<tr><td colspan="16">source_ID</td><td colspan="16"></td></tr>
<tr><td colspan="32">Λοιπές πληροφορίες επικεφαλίδας</td></tr>
<tr><td colspan="32">Τετράδα δεδομένων</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC</td></tr>
<tr><td colspan="32">Πρώτη τετράδα τμήματος δεδομένων</td></tr>
<tr><td colspan="32">Δεύτερη τετράδα τμήματος δεδομένων</td></tr>
<tr><td colspan="32">Λοιπες τετράδες τμήματος δεδομένων</td></tr>
<tr><td colspan="16"></td><td colspan="16">Συμπληρωματικά bytes (padding)</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων τμήματος δεδομένων CRC</td></tr>
</table>

Πίνακας 12: Πεδία ασύγχρονου πακέτου

Πεδίο	Μέγεθος	Περιγραφή
destination_ID	16	Διεύθυνση κόμβου προορισμού
tl	6	Δείκτης συναλλαγής (transaction label)
rt	2	Κωδικός αναμετάδοσης (retry code)
pri	4	Κωδικός προτεραιότητας (priority)
source_ID	16	Διεύθυνση κόμβου προέλευσης

Το πεδίο του δείκτη συναλλαγής (transaction label, tl) χρησιμοποιείται για την ταυτοποίηση με μοναδικό τρόπο κάθε ημιτελούς συναλλαγής ανάμεσα σε δυο κόμβους. Η χρήση αυτού του πεδίου επιτρέπει την πολυνηματική επεξεργασία των συναλλαγών, παρέχοντας τη δυνατότητα τοποθέτησης πολλαπλών αιτήσεων σε ουρά αναμονής, χωρίς να χρειάζεται να έχει εξυπηρετηθεί η προηγούμενη αίτηση πριν ληφθεί η επόμενη. Αυτή η μορφή παράλληλης επεξεργασίας μπορεί δυνητικά να αυξήσει τις επιδόσεις του διαύλου, καθώς μειώνει την επίδραση των καθυστερήσεων σε αυτόν. Με βάση το πρότυπο IEEE 1394, όμως, η υποστήριξη πολυνηματικών συναλλαγών σε έναν κόμβο είναι προαιρετική.

Το πεδίο του κωδικού αναμετάδοσης πακέτου (retry code, rt) χρησιμοποιείται όταν κάποια αίτηση συναλλαγής δεν μπορεί να εξυπηρετηθεί άμεσα από τον κόμβο προορισμού της και το επίπεδο συναλλαγής κρίνει ότι πρέπει να αναμεταδοθεί, με βάση κάποιο κατάλληλο πρωτόκολλο. Το πεδίο rt επιτρέπει στον κόμβο που λαμβάνει το πακέτο να γνωρίζει αν αυτό αποτελεί επανάληψη κάποιας προηγούμενης αίτησης και ταυτόχρονα χρησιμοποιείται εσωτερικά ως δείκτης κατάστασης από τον αλγόριθμο αναμετάδοσης. Κάθε κόμβος IEEE 1394 οφείλει να υποστηρίζει ένα από τα δύο πρωτόκολλα επανάληψης που έχουν προδιαγραφεί στο πρότυπο,

έτσι ώστε το υπόστρωμα συναλλαγής να μπορεί να τελέσει τη διαδικασία αναμετάδοσης των αιτήσεων διαφανώς, χωρίς την επέμβαση του επιπέδου εφαρμογής.

Τέλος, το πεδίο κωδικού προτεραιότητας (priority, pri) αντιστοιχεί άμεσα στο ομώνυμο πεδίο της ακολουθίας διαιτησίας των διαύλων με τοπολογία αρτηρίας. Κατά συνέπεια δεν χρησιμοποιείται στην τοπολογία δένδρου, για την οποία δεν έχει προδιαγραφεί κάποιο σύστημα διαιτησίας με πολλαπλές κλάσεις προτεραιότητας.

Για την αποστολή ενός ασύγχρονου πακέτου, μετά από αίτηση του επιπέδου συναλλαγής, το υπόστρωμα διασύνδεσης κατασκευάζει το πακέτο ανάλογα με τον τύπο του και ζητά από το φυσικό επίπεδο να εκκινήσει τη διαδικασία διαιτησίας. Αν η διαδικασία αυτή δεν επιτύχει, τότε ο κόμβος προετοιμάζεται για πιθανό εισερχόμενο πακέτο και αναβάλλει την αποστολή για κάποια μεταγενέστερη χρονική στιγμή. Όταν πλέον του παραχωρηθεί ο έλεγχος του διαύλου, αποστέλλει το πακέτο και, αν είναι απαραίτητο, αναμένει το αντίστοιχο πακέτο για την επιβεβαίωση της λήψης, για την άφιξη της οποίας θα ενημερωθεί άμεσα το υπόστρωμα συναλλαγής. Αν το πακέτο ACK δεν φθάσει επιτυχώς εντός συγκεκριμένων χρονικών ορίων, το επίπεδο διασύνδεσης θεωρεί ότι η τρέχουσα υποδράση απέτυχε και πληροφορεί τα ανώτερα επίπεδα.

Όταν ένας κόμβος του διαύλου δεν μεταδίδει δεδομένα, τότε το φυσικό επίπεδο λαμβάνει οτιδήποτε εκπέμπεται στο δίαυλο. Το επίπεδο διασύνδεσης επεξεργάζεται τα δεδομένα και τη σηματοδοσία και ελέγχει κάθε ολοκληρωμένο πακέτο που λαμβάνει, απορρίπτοντας όσα πακέτα δεν απευθύνονται στον κόμβο ή έχουν φθάσει αλλοιωμένα. Με την επιτυχή λήψη ενός πακέτου το επίπεδο συναλλαγής ενημερώνεται και το υπόστρωμα διασύνδεσης αποκρίνεται με ένα πακέτο ACK, όταν αυτό είναι αναγκαίο. Σε κάποιες, περιπτώσεις όπου το επίπεδο συναλλαγής μπορεί να πραγματοποιήσει την υποδράση απόκρισης άμεσα, είναι δυνατόν το πακέτο απόκρισης να μεταδοθεί ενωμένο με το πακέτο ACK (concatenated response), χωρίς ο κόμβος να χρειαστεί νέο κύκλο διαιτησίας.

2.4.2.1 Ασύγχρονα πακέτα χωρίς φορτίο

Όταν ένα ασύγχρονο πακέτο δεν έχει φορτίο αποκτά την ακόλουθη γενική δομή:

Σχήμα 18: Μορφή ασύγχρονου πακέτου χωρίς φορτίο

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">destination_ID</td><td colspan="4">tl</td><td>rt</td><td colspan="3">tcode</td><td colspan="3">pri</td></tr>
<tr><td colspan="16">source_ID</td><td colspan="16"></td></tr>
<tr><td colspan="32">Λοιπές πληροφορίες επικεφαλίδας</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC</td></tr>
</table>

2.4.2.1.1 Αίτηση για ανάγνωση τετράδας

Οι αιτήσεις για ανάγνωση μιας τετράδας ακολουθούν τη γενική μορφή ασύγχρονου πακέτου χωρίς φορτίο, με την επικεφαλίδα να φέρει ένα πεδίο 48 δυαδικών ψηφίων που αποτελεί τη διεύθυνση της τετράδας που θα ανακτηθεί. Έχει, συνεπώς, τη δομή του σχήματος 19.

Σχήμα 19: Μορφή αίτησης ανάγνωσης τετράδας

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">destination_ID</td><td colspan="4">tl</td><td>rt</td><td colspan="3">tcode</td><td colspan="3">pri</td></tr>
<tr><td colspan="16">source_ID</td><td colspan="16"></td></tr>
<tr><td colspan="32">destination_offset</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC</td></tr>
</table>

Πίνακας 13: Πεδία αίτησης ανάγνωσης τετράδας

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 0100_2
destination_offset	48	Διεύθυνση ανάγνωσης στον κόμβο προορισμού

2.4.2.1.2 Απόκριση συναλλαγής εγγραφής

Τα πακέτα απόκρισης συναλλαγής εγγραφής αποστέλλονται μόνο ως απόκριση σε κάποια αίτηση για εγγραφή τετράδας ή τμήματος δεδομένων. Η μορφή τους παρουσιάζεται αναλυτικά στο σχήμα 20.

Σχήμα 20: Μορφή απόκρισης συναλλαγής εγγραφής

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
destination_ID																tl						rt		tcode				pri			
source_ID																rcode															
Δεσμευμένα ψηφία																															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															

Πίνακας 14: Πεδία απόκρισης συναλλαγής εγγραφής

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 0010_2
rcode	4	Κωδικός επιστροφής που υποδηλώνει αν η συναλλαγή ολοκληρώθηκε με επιτυχία

Το πεδίο rcode ενημερώνει τον κόμβο που έστειλε την αίτηση για την κατάσταση, επιτυχή ή όχι, ολοκλήρωσης της συναλλαγής. Οι τιμές που μπορεί να πάρει υποδηλώνουν επιτυχία ή κάποιο από τα ακόλουθα σφάλματα:

- Άκυρη διεύθυνση στην αίτηση της συναλλαγής, που βρίσκεται εκτός του πεδίου διευθύνσεων του κόμβου
- Άκυρο ή εσφαλμένο πεδίο στην αίτηση, όπως για παράδειγμα μια απόπειρα εγγραφής σε κάποια θέση μόνο ανάγνωσης
- Σύγκρουση με κάποια άλλη διεργασία στον κόμβο - η συναλλαγή μπορεί να επαναληφθεί σε μεταγενέστερη χρονική στιγμή
- Σφάλμα συστήματος στον κόμβο προορισμού της αίτησης που καθιστά αδύνατη την πραγματοποίηση της συναλλαγής

2.4.2.2 Ασύγχρονα πακέτα με φορτίο μιας τετράδας

Για δεδομένα με μέγεθος το πολύ μιας τετράδας χρησιμοποιείται μια μορφή ασύγχρονου πακέτου που περιέχει μια επιπλέον τετράδα στην επικεφαλίδα του. Αυτό επιτρέπει την αποφυγή της επιβάρυνσης που επιφέρει η ύπαρξη τμήματος δεδομένων, αφού δεν απαιτείται η επιπλέον τετράδα για το δεύτερο κώδικα CRC, ούτε επεξεργαστική ισχύς για τον υπολογισμό του.

Σχήμα 21: Μορφή ασύγχρονου πακέτου με φορτίο μιας τετράδας

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
destination_ID																tl						rt		tcode				pri			
source_ID																															
Λοιπές πληροφορίες επικεφαλίδας																															
Τετράδα δεδομένων																															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															

2.4.2.2.1 Αίτηση για ανάγνωση τμήματος δεδομένων

Σχήμα 22: Μορφή αίτησης για ανάγνωση τμήματος δεδομένων

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
destination_ID																tl						rt		tcode				pri			
source_ID																															
destination_offset																															
data_length																extended_tcode															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															

Πίνακας 15: Πεδία αίτησης για ανάγνωση τμήματος δεδομένων

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 0101_2
data_length	16	Μέγεθος τμήματος προς ανάγνωση
extended_tcode	16	Δεσμευμένο, πρέπει να είναι 0000_{16}

Για την ανάγνωση ενός τμήματος δεδομένων από κάποιον κόμβου του διαύλου αποστέλλεται ένα πακέτο με τη μορφή του σχήματος 22. Το μέγεθος του τμήματος που θα αναγνωσθεί προσδιορίζεται με ένα πεδίο 16 bit, με τιμές στο διάστημα [0-65535] bytes. Επειδή οι συναλλαγές ανάγνωσης τετράδων είναι πιο αποδοτικές και υποστηρίζονται υποχρεωτικά από όλους τους κόμβους, σπανίως θα δούμε τιμές

του πεδίου data_length μικρότερες ή ίσες του 4. Η μέγιστη τιμή του, όμως, εξαρτάται από τα χαρακτηριστικά του διαύλου. Με βάση την τροπολογία IEEE Std 1394b-2002 το μέγιστο μέγεθος του τμήματος δεδομένων εξαρτάται από το ρυθμό μεταφοράς δεδομένων του διαύλου και δίνεται από τον πίνακα 16. Παρατηρούμε ότι η επιλογή των μέγιστων τιμών έχει γίνει με σκοπό να παραμείνει σταθερός ο μέγιστος χρόνος διάρκειας της μετάδοσης ενός πακέτου, ανεξάρτητα από την ταχύτητα του διαύλου.

Πίνακας 16: Μέγιστη ποσότητα δεδομένων ασύγχρονου πακέτου

Ταχύτητα διαύλου	Μέγιστη ποσότητα δεδομένων ανά πακέτο (bytes)
S25	128
S50	256
S100	512
S200	1024
S400	2048
S800	4096
S1600	8192
S3200	16384

2.4.2.2.2 Αίτηση για εγγραφή τετράδας

Για την εγγραφή μιας τετράδας σε κάποια θέση μνήμης ενός κόμβου χρησιμοποιείται ένα πακέτο με την ακόλουθη δομή:

Σχήμα 23: Μορφή αίτησης για για εγγραφή τετράδας

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
destination_ID																tl						rt		tcode				pri			
source_ID																															
destination_offset																															
quadlet_data																															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															

Πίνακας 17: Πεδία αίτησης για εγγραφή τετράδας

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 0000_2
destination_offset	48	Διεύθυνση εγγραφής στον κόμβο προορισμού
quadlet_data	32	Δεδομένα τετράδας προς εγγραφή

2.4.2.2.3 Απόκριση συναλλαγής ανάγνωσης τετράδας

Όταν ένας κόμβος λάβει μια αίτηση για την ανάγνωση μιας τετράδας αποκρίνεται με ένα πακέτο με τη μορφή του σχήματος 24. Η τιμή τις τετράδας επιστρέφεται ως μέρος της επικεφαλίδας του πακέτου, ενώ το πεδίο rcode υποδηλώνει αν η συναλλαγή ολοκληρώθηκε επιτυχώς.

Σχήμα 24: Μορφή απόκρισης συναλλαγής ανάγνωσης τετράδας

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
destination_ID																tl						rt		tcode				pri			
source_ID																rcode															
Δεσμευμένα ψηφία																															
quadlet_data																															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															

Πίνακας 18: Πεδία απόκρισης συναλλαγής ανάγνωσης τετράδας

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 0110_2

2.4.2.2.4 Πακέτο χρονισμού ισόχρονων συναλλαγών Cycle Start

Σχήμα 25: Μορφή πακέτου Cycle Start

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
destination_ID																tl						rt		tcode				pri			
source_ID																															
destination_offset																															
cycle_time_data																															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															

Πίνακας 19: Πεδία πακέτου Cycle Start

Πεδίο	Μέγεθος	Περιγραφή
destination_ID	16	Διεύθυνση μαζικής αποστολής: FF_{16}
tl	6	Τιμή: 000000_2
rt	2	Τιμή: 00_2
tcode	4	Κωδικός συναλλαγής: 1000_2
pri	4	Μέγιστη προτεραιότητα: 1111_2
source_ID	16	Διεύθυνση κόμβου παροχής ισόχρονου χρονισμού CM
destination_offset	48	Διεύθυνση καταχωρητή CYCLE_TIME: $ffffff0000200_{16}$
cycle_time_data	32	Περιεχόμενα καταχωρητή CYCLE_TIME στον κόμβο CM

Το πακέτο χρονισμού με τη δομή του σχήματος 25 υποδεικνύει την έναρξη ενός νέου κύκλου ισόχρονων συναλλαγών στο δίαυλο. Αποστέλλεται μόνο από τον κόμβο παροχής χρονισμού (Cycle Master, CM) και το πρότυπο IEEE Std 1394-1995 θεωρεί ότι η αυτόματη περιοδική παραγωγή και η λήψη αυτού του πακέτου πραγματοποιείται από το επίπεδο διασύνδεσης, χωρίς την παρέμβαση των υψηλότερων επιπέδων, αν και σε μια πραγματική υλοποίηση μπορούν να εμπλέκονται και τα υπόλοιπα τμήματα του κόμβου. Το πακέτο αυτό μπορεί να ερμηνευθεί ως αίτηση εγγραφής στη θέση μνήμης του καταχωρητή CSR CYCLE_TIME, με την τετράδα δεδομένων να περιέχει την τιμή του καταχωρητή αυτού στον κόμβο CM. Μια ακόμα ιδιαιτερότητα του πακέτου αυτού είναι ότι αποστέλλεται ταυτόχρονα σε όλους τους κόμβους του διαύλου (broadcast), τυπικά με μια περίοδο 125 μs, που αντιστοιχεί στη συχνότητα έναρξης των κύκλων ισόχρονων συναλλαγών (8 kHz).

2.4.2.3 Ασύγχρονα πακέτα με φορτίο τμήματος δεδομένων

Για τη μεταφορά ποσοτήτων δεδομένων με μέγεθος μεγαλύτερο από 4 bytes ή σε περιπτώσεις όπου η τετράδα δεδομένων της επικεφαλίδας είναι ήδη δεσμευμένη χρησιμοποιούνται πακέτα με τη δομή του σχήματος 26. Αυτά τα πακέτα μεταφέρουν μηδέν ή περισσότερα bytes δεδομένων, τα οποία ακολουθούν την επικεφαλίδα και συνοδεύονται από έναν κώδικα ελέγχου λαθών CRC. Η μέγιστη ποσότητα δεδομένων ανά πακέτο δίνεται κατά κανόνα από τον πίνακα 16 που είδαμε σε προηγούμενη παράγραφο. Αν απαιτείται, τα δεδομένα προς μετάδοση συμπληρώνονται με μηδενικά bytes (padding), έτσι ώστε να μεταφέρονται ολοκληρωμένες τετράδες. Το τμήμα δεδομένων και ο αντίστοιχος κώδικας CRC δεν μεταδίδονται καθόλου αν η ποσότητα δεδομένων προς μεταφορά είναι μηδενική.

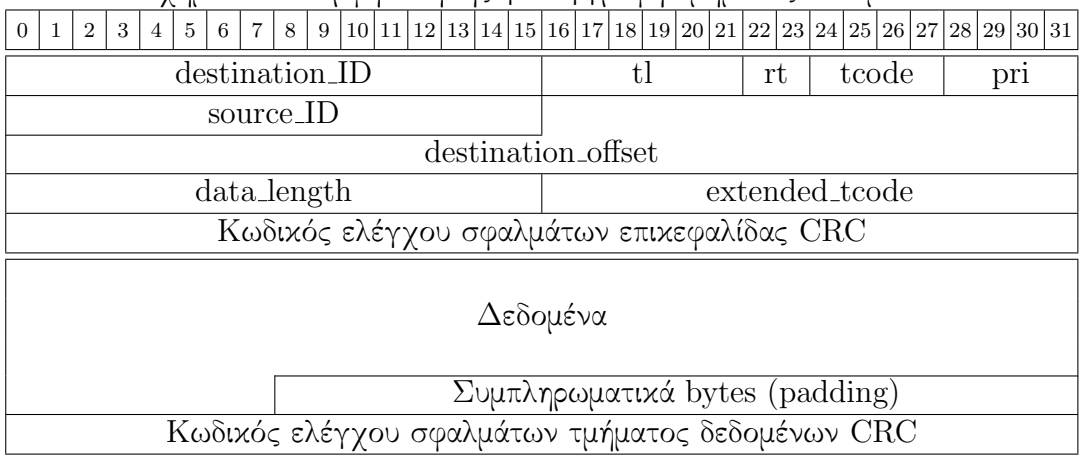
Σχήμα 26: Μορφή ασύγχρονου πακέτου με φορτίο τμήματος δεδομένων

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">destination_ID</td><td colspan="4">tl</td><td>rt</td><td colspan="3">tcode</td><td colspan="3">pri</td></tr>
<tr><td colspan="8">source_ID</td><td colspan="24"></td></tr>
<tr><td colspan="32">Λοιπές πληροφορίες επικεφαλίδας</td></tr>
<tr><td colspan="16">data_length</td><td colspan="16">extended_tcode</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC</td></tr>
<tr><td colspan="32">Δεδομένα</td></tr>
<tr><td colspan="8"></td><td colspan="24">Συμπληρωματικά bytes (padding)</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων τμήματος δεδομένων CRC</td></tr>
</table>

2.4.2.3.1 Αίτηση για εγγραφή τμήματος δεδομένων

Με το πακέτο αυτό ζητείται η εγγραφή του τμήματος δεδομένων που μεταφέρει στη διεύθυνση προορισμού που προσδιορίζεται:

Σχήμα 27: Μορφή αίτησης για εγγραφή τμήματος δεδομένων



Πίνακας 20: Πεδία αίτησης για εγγραφή τμήματος δεδομένων

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 0001_2
extended_tcode	16	Δεσμευμένο, πρέπει να είναι 0000_{16}

2.4.2.3.2 Απόκριση συναλλαγής ανάγνωσης τμήματος δεδομένων

Με το πακέτο αυτό μεταφέρεται ένα τμήμα δεδομένων που έχει ζητηθεί με προηγούμενη αίτηση στον κόμβο προέλευσής της. Σε περίπτωση που η συναλλαγή απέτυχε, το πεδίο data_length είναι μηδενικό και δεν μεταδίδεται το τμήμα ή ο αντίστοιχος κώδικας CRC. Η δομή του πακέτου αυτού αναλύεται στο σχήμα 28 και τον συνοδευτικό πίνακα.

Πίνακας 21: Πεδία απόκρισης συναλλαγής ανάγνωσης τμήματος δεδομένων

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 0111_2
extended_tcode	16	Δεσμευμένο, πρέπει να είναι 0000_{16}

Σχήμα 28: Μορφή απόκρισης συναλλαγής ανάγνωσης τμήματος δεδομένων

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
destination_ID																tl						rt		tcode				pri			
source_ID																															
Δεσμευμένα ψηφία																															
data_length																extended_tcode															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															
Δεδομένα																															
Συμπληρωματικά bytes (padding)																															
Κωδικός ελέγχου σφαλμάτων τμήματος δεδομένων CRC																															

2.4.2.3.3 Αίτηση συναλλαγής κλειδώματος

Οι συναλλαγές κλειδώματος προσφέρουν ατομική πρόσβαση (atomic access) στη διεύθυνση που προσδιορίζεται στην αίτηση. Παρέχουν ένα σύνολο από βασικές συναρτήσεις-τελεστές, που μπορούν να χρησιμοποιηθούν για τη δημιουργία πιο σύνθετων δομών ελέγχου και ταυτοχρονισμού. Μερικές από τις συναρτήσεις αυτές είναι οι τελεστές σύγκρισης-αντικατάστασης (compare-swap), ατομικής πρόσθεσης (atomic addition) και αντικατάστασης με χρήση μάσκας (mask swap).

Πολλά πρωτόκολλα βασισμένα στο IEEE 1394 ορίζουν κάποια μορφή συνεδρίας για τη χρήση των πόρων ενός κόμβου, όπως για παράδειγμα το login στο πρωτόκολλο SBP. Σε αυτές τις περιπτώσεις χρησιμοποιούνται εκτενώς συναλλαγές κλειδώματος για να αποφευχθούν οι δυσάρεστες παρενέργειες που μπορεί να επιφέρει η ανεξέλεγκτη πρόσβαση στο χώρο διευθύνσεων ενός κόμβου από πολλαπλές πηγές.

Σχήμα 29: Μορφή αίτησης συναλλαγής κλειδώματος

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
destination_ID																tl						rt		tcode				pri			
source_ID																															
destination_offset																															
data_length																extended_tcode															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															
arg_value																															
data_value																															
Κωδικός ελέγχου σφαλμάτων τμήματος δεδομένων CRC																															

Υποστηρίζονται συναρτήσεις 32 ή 64 bit με ένα ή δύο ορίσματα. Το πεδίο extended_tcode ορίζει τη συνάρτηση που θα χρησιμοποιηθεί, ενώ η τιμή data_length

Πίνακας 22: Πεδία αίτησης συναλλαγής κλειδώματος

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 1001_2
extended_tcode	16	Κωδικός συνάρτησης συναλλαγής κλειδώματος
arg_value	32/64	Δεύτερο όρισμα συνάρτησης
data_value	32/64	Πρώτο όρισμα συνάρτησης

επιτρέπει την επιλογή του σωστού εύρους (32 ή 64 bit) για τον τελεστή. Για συναρτήσεις με ένα μόνο όρισμα το πεδίο arg_value δεν περιλαμβάνεται στο πακέτο που μεταδίδεται.

2.4.2.3.4 Απόκριση συναλλαγής κλειδώματος

Για κάθε αίτηση συναλλαγής κλειδώματος ο κόμβος προορισμού της αποκρίνεται με το πακέτο του σχήματος 30. Η δομή του είναι παρόμοια με αυτήν της αίτησης, με το μήκος του να εξαρτάται από το εύρος της μεταβλητής πάνω στην οποία έδρασε η συναλλαγή. Σε περίπτωση σφάλματος το τμήμα δεδομένων δεν μεταδίδεται.

Σχήμα 30: Μορφή απόκρισης συναλλαγής κλειδώματος

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
destination_ID																tl						rt		tcode				pri			
source_ID																															
Δεσμευμένα ψηφία																															
data_length																extended_tcode															
Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC																															
old_value																															
Κωδικός ελέγχου σφαλμάτων τμήματος δεδομένων CRC																															

Πίνακας 23: Πεδία απόκρισης συναλλαγής κλειδώματος

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 1011_2
extended_tcode	16	Κωδικός συνάρτησης συναλλαγής κλειδώματος
old_value	32/64	Παλαιά τιμή της μεταβλητής ή του καταχωρητή που επηρεάστηκε από τη συναλλαγή κλειδώματος

2.4.3 Ισόχρονα πακέτα

Όλα τα ισόχρονα πακέτα έχουν συγκεκριμένη δομή, η οποία περιγράφεται στο ακόλουθο σχήμα:

Σχήμα 31: Μορφή ισόχρονου πακέτου

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">data_length</td><td colspan="2">tag</td><td colspan="4">channel</td><td colspan="2">tcode</td><td colspan="8">sy</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC</td></tr>
<tr><td colspan="32">Δεδομένα</td></tr>
<tr><td colspan="24"></td><td colspan="8">Συμπληρωματικά bytes (padding)</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων τμήματος δεδομένων CRC</td></tr>
</table>

Πίνακας 24: Πεδία ισόχρονου πακέτου

Πεδίο	Μέγεθος	Περιγραφή
tag	2	Μορφή δεδομένων
channel	6	Κανάλι ισόχρονης μετάδοσης
tcode	4	Κωδικός συναλλαγής: 1010_2
sy	4	Κωδικός συγχρονισμού (synchronization code)

Με βάση το πρότυπο IEEE Std 1394a-2000, η μέγιστη τιμή του πεδίου data_length εξαρτάται από το ρυθμό μεταφοράς δεδομένων του διαύλου και δίνεται από τον πίνακα 25. Σε κάθε περίπτωση, το μέγεθος των πακέτων θα πρέπει να είναι τέτοιο ώστε να κατά τη μετάδοσή τους να μην υπερβαίνουν το εύρος ζώνης (bandwidth) που έχει ζητηθεί και καταχωρηθεί στο διαχειριστή ισόχρονων πόρων (Isochronous Resource Manager, IRM) για το συγκεκριμένο κανάλι. Το πεδίο channel περιέχει τον αριθμό του καναλιού στο οποίο θα μεταδοθεί το πακέτο. Υπάρχουν συνολικά 64 κανάλια με αριθμούς στο διάστημα [0-63] και παραχωρούνται για χρήση μετά από αίτηση στον κόμβο IRM.

Τα πεδία tag και sy προορίζονται για χρήση από το επίπεδο της εφαρμογής. Το πεδίο tag υποδεικνύει αν τα δεδομένα έχουν κάποια συγκεκριμένη μορφή. Στην αρχική έκδοση IEEE Std 1394-1995 υποστηριζόταν μόνο η τιμή 00_2 , για δεδομένα χωρίς δομή, αλλά στην τροπολογία IEEE Std 1394a-2000 προστέθηκε και η τιμή 01_2 για εφαρμογές του προτύπου IEC 61883. Ομοίως, για το πεδίο sy δεν προβλέπεται κάτι συγκεκριμένο σε σχέση με τα περιεχόμενά του, αν και πρωτόκολλα όπως το IEC 61883 προσδιορίζουν τη χρήση του για την εκάστοτε εφαρμογή.

Η μετάδοση ισόχρονων πακέτων επιτρέπεται μόνο εντός του ισόχρονου κύκλου, ο οποίος αρχίζει με την εκπομπή ενός πακέτου Cycle Start από τον κόμβο CM και ολοκληρώνεται όταν δεν μεταδοθεί κάποιο πακέτο για χρόνο ίσο με το διάστημα υποδράσεων. Αντίστοιχα, ο μηχανισμός διαιτησίας δεν επιτρέπει ασύγχρονες μεταδόσεις μέχρι το πέρας του κύκλου. Η εξασφάλιση εύρους ζώνης για

Πίνακας 25: Μέγιστη ποσότητα δεδομένων ισόχρονου πακέτου

Ταχύτητα διαύλου	Μέγιστη ποσότητα δεδομένων ανά πακέτο (bytes)
S25	256
S50	512
S100	1024
S200	2048
S400	4096
S800	8192
S1600	16384
S3200	32768

ασύγχρονες συναλλαγές επαφίεται στον κόμβο IRM για τη σωστή καταχώρηση των πόρων και σε κάθε κόμβο χωριστά για την τήρηση των ορίων που έχουν δηλωθεί.

Κάθε κόμβος που χειρίζεται ισόχρονα πακέτα διαθέτει μια ουρά όπου τοποθετούνται πριν μεταδοθούν. Με την έλευση ενός ισόχρονου κύκλου πραγματοποιείται η διαδικασία της διαιτησίας και ο κόμβος που αποκτά τον έλεγχο του διαύλου στέλνει από ένα πακέτο για κάθε κανάλι εκπομπής του, αν υπάρχει τέτοιο στην ουρά. Τα πακέτα αυτά αποστέλλονται ενωμένα, ώστε να μην απαιτείται νέος κύκλος διαιτησίας. Η διαδικασία επαναλαμβάνεται για κάθε κόμβο που διαθέτει ισόχρονα πακέτα προς μετάδοση, ώσπου αυτά να εξαντληθούν και να ολοκληρωθεί ο κύκλος. Τα επίπεδα διασύνδεσης όλων των κόμβων μεταβαίνουν σε κατάσταση σφάλματος (CYCLE_TOO_LONG) αν η διάρκεια ενός ισόχρονου κύκλου υπερβεί το χρόνο που οριοθετείται από δυο πακέτα Cycle Start.

Για τη λήψη ισόχρονων πακέτων, το επίπεδο εφαρμογής οφείλει να έχει ρυθμίσει το υπόστρωμα διασύνδεσης, έτσι ώστε να δέχεται μεταδιδόμενα πακέτα από τα πρέποντα ισόχρονα κανάλια. Μετά από αυτή τη ρύθμιση το επίπεδο διασύνδεσης ενημερώνει τα ανώτερα επίπεδα για κάθε ισόχρονο πακέτο που ελήφθη με επιτυχία. Κατά την αρχική έκδοση του προτύπου ισόχρονα πακέτα εκτός του ισόχρονου κύκλου πρέπει να απορρίπτονται. Με την τροπολογία IEEE Std 1394a-2000, όμως, αυτός ο περιορισμός δεν υφίσταται πλέον, έτσι ώστε να υποστηριχθεί η λειτουργία των πακέτων ασύγχρονης ροής (asynchronous stream packets).

Ο χειρισμός των ισόχρονων συναλλαγών πραγματοποιείται ουσιαστικά από το υπόστρωμα διασύνδεσης, με το επίπεδο εφαρμογής να είναι υπεύθυνο για την τοποθέτηση δεδομένων στην ουρά αποστολής και την ανάκτησή τους από τους ενταμιευτές (buffers) λήψης. Το επίπεδο συναλλαγής δεν εμπλέκεται σε αυτή τη διαδικασία.

Κατά την περίοδο αυτορρύθμισης του διαύλου, η οποία τυπικά διαρκεί 200 μs, δεν πραγματοποιούνται ισόχρονες συναλλαγές. Το επίπεδο εφαρμογής θα πρέπει να μπορεί να χειριστεί αυτές τις διακοπές λειτουργίας και τυχόν χαμένα πακέτα, απαίτηση λογική δεδομένου ότι το σύστημα ισόχρονων συναλλαγών δεν έχει σχεδιαστεί με κύριο στόχο την αξιοπιστία, αλλά την έγκαιρη μετάδοση δεδομένων.

2.4.4 Πακέτα ασύγχρονης ροής

Τα πακέτα ασύγχρονης ροής (asynchronous stream packets) είναι μια συμβατή επέκταση του αρχικού προτύπου IEEE Std 1394-1995, η οποία περιλαμβάνεται στην τροπολογία IEEE Std 1394a-2000. Αποτελούν υβρίδιο ανάμεσα στα ήδη προδιαγραμμένα ισόχρονα και ασύγχρονα πακέτα. Συγκεκριμένα, όπως φαίνεται στο σχήμα 32, έχουν ακριβώς την ίδια δομή με τα ισόχρονα πακέτα, φέροντας και τον ίδιο κωδικό συναλλαγής. Αντίθετα όμως με τα ισόχρονα πακέτα, η μετάδοσή τους πραγματοποιείται εκτός του ισόχρονου κύκλου, μαζί με τα υπόλοιπα ασύγχρονα πακέτα και με τους κανόνες διαιτησίας που τα διέπουν. Το μέγιστο μέγεθός τους ακολουθεί τα όρια του πίνακα 16 (σελίδα 54), που ισχύουν και για τα ασύγχρονα πακέτα.

Σχήμα 32: Μορφή ασύγχρονου πακέτου ροής

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">data_length</td><td>tag</td><td colspan="6">channel</td><td colspan="4">tcode</td><td colspan="4">sy</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων επικεφαλίδας CRC</td></tr>
<tr><td colspan="32">Δεδομένα</td></tr>
<tr><td colspan="8"></td><td colspan="24">Συμπληρωματικά bytes (padding)</td></tr>
<tr><td colspan="32">Κωδικός ελέγχου σφαλμάτων τμήματος δεδομένων CRC</td></tr>
</table>

Πίνακας 26: Πεδία ασύγχρονου πακέτου ροής

Πεδίο	Μέγεθος	Περιγραφή
tcode	4	Κωδικός συναλλαγής: 1010_2

Για τη χρήση αυτής της επέκτασης απαιτείται απλά η δέσμευση ενός ισόχρονου καναλιού στον κόμβο IRM. Το πλεονέκτημα των πακέτων ασύγχρονης ροής είναι ότι δίνουν τη δυνατότητα αποστολής δεδομένων σε πολλαπλούς κόμβους χωρίς τη δέσμευση πόρων, όπως για παράδειγμα εύρους ζώνης, στον IRM και χωρίς να υπάρχει περίπτωση να επηρεάσουν τις ισόχρονες συναλλαγές, οι οποίες είναι χρονικά κρίσιμες. Αυτή η δυνατότητα δεν υπάρχει όταν χρησιμοποιούνται απλά ασύγχρονα πακέτα με διεύθυνση προορισμού $3f_{16}/63_{10}$ (broadcast), καθώς αυτά επηρεάζουν αδιακρίτως το χώρο διευθύνσεων όλων των κόμβων του διαύλου.

2.4.5 Πακέτα επιβεβαίωσης λήψης

Τα πακέτα πακέτα επιβεβαίωσης λήψης (acknowledgement packets, ACK) αποστέλλονται από έναν κόμβο ως άμεση απόκριση σε κάθε πακέτο που είχε ως μόνο προορισμό τον κόμβο αυτό. Κατά συνέπεια δεν παράγονται για ισόχρονα πακέτα ή για πακέτα που απευθύνονται σε όλους τους κόμβους του διαύλου (broadcast). Έχουν χαρακτηριστικό μήκος ενός μόνο byte - όλοι οι άλλοι τύποι πακέτων έχουν μεγέθη πολλαπλάσια της τετράδας - και δομή που παρουσιάζεται αναλυτικά στο σχήμα 33 και τον επεξηγηματικό πίνακα.

Σχήμα 33: Μορφή πακέτου ACK

0	1	2	3	4	5	6	7
ack_code				ack_parity			

Πίνακας 27: Πεδία πακέτου ACK

Πεδίο	Μέγεθος	Περιγραφή
ack_code	4	Κωδικός επιβεβαίωσης
ack_parity	4	Συμπλήρωμα του 1 των πρώτων τεσσάρων ψηφίων

Το πεδίο ack_code είναι το αποτέλεσμα της επεξεργασίας του πακέτου που ελήφθη από το υπόστρωμα διασύνδεσης. Από το πρότυπο IEEE Std 1394-1995 προβλέπονται τιμές για τις ακόλουθες περιπτώσεις:

- Η λήψη ήταν επιτυχής και η συναλλαγή έχει ολοκληρωθεί (ack_complete)
- Η λήψη ήταν επιτυχής και θα ακολουθήσει μια υποδράση απόκρισης για την ολοκλήρωση της συναλλαγής (ack_pending)
- Το πακέτο απερρίφθη, αλλά αν επαναληφθεί η μετάδοσή του σε κάποια μεταγενέστερη χρονική στιγμή ίσως να γίνει δεκτό (ack_busy_X, ack_busy_A, ack_busy_B)
- Η λήψη του πακέτου ήταν ανεπιτυχής επειδή ο έλεγχος σφαλμάτων με τον κώδικα CRC έδειξε ότι είχε αλλοιωθεί (ack_data_error)
- Το πακέτο απερρίφθη επειδή κάποιο πεδίο του είχε άκυρη τιμή (ack_type_error)

Το πεδίο ack_parity αποτελεί τον κώδικα ελέγχου λαθών του πακέτου ACK. Σε περίπτωση που αυτός ο έλεγχος αποτύχει ή το πεδίο ack_code έχει κάποια άκυρη τιμή τότε το πακέτο ACK αγνοείται από τον κόμβο που το έλαβε.

2.5 Το επίπεδο συναλλαγής

Το επίπεδο συναλλαγής προσφέρει τις εξής βασικές λειτουργίες για τη μεταφορά δεδομένων μέσω του διαύλου IEEE 1394:

- Συναλλαγές ανάγνωσης για τη μεταφορά δεδομένων από έναν κόμβο
- Συναλλαγές εγγραφής για τη μεταφορά δεδομένων προς έναν κόμβο
- Συναλλαγές κλειδώματος για την τέλεση ατομικών πράξεων σε έναν κόμβο

Κάθε συναλλαγή γίνεται κατόπιν εντολής από το υπόστρωμα διαχείρισης διαύλου ή το επίπεδο εφαρμογής και αποτελείται από μια υποδράση αίτησης, η οποία στις περισσότερες περιπτώσεις ακολουθείται από μια υποδράση απόκρισης. Για την πραγματοποίηση των υποδράσεων χρησιμοποιούνται κατά κόρον οι υπηρεσίες που παρέχει το επίπεδο διασύνδεσης. Το επίπεδο συναλλαγής μπορεί να έχει μονονηματική (singlethreaded) ή πολυνηματική (multithreaded) δομή. Η πολυνηματική δομή επιτρέπει την ταυτόχρονη επεξεργασία πολλαπλών συναλλαγών, βελτιώνοντας τις επιδόσεις του κόμβου, αλλά έχει αυξημένη δυσκολία και κόστος υλοποίησης.

Ένα εξίσου σημαντικό μέρος του επιπέδου συναλλαγής είναι και οι μηχανισμοί διαχείρισης λαθών και επανάληψης αιτήσεων προς απασχολημένους κόμβους. Με τις λειτουργίες αυτές βελτιώνεται σημαντικά η ανοχή του διαύλου σε πρόσκαιρα κωλύματα.

2.5.1 Συναλλαγές ανάγνωσης

Για την πραγματοποίηση μιας ασύγχρονης συναλλαγής ανάγνωσης το υπόστρωμα διαχείρισης διαύλου ή το επίπεδο εφαρμογής ενός κόμβου στέλνει μια εντολή στο επίπεδο συναλλαγής του, ώστε να τελεστεί η κατάλληλη υποδράση αίτησης. Στην εντολή αυτή συμπεριλαμβάνονται όλα τα απαραίτητα στοιχεία, όπως η διεύθυνση του απομακρυσμένου κόμβου (destination node), η θέση της περιοχής μνήμης που θα αναγνωσθεί και το μέγεθός της.

Όταν ο κόμβος προορισμού λάβει επιτυχώς το πακέτο της αίτησης, το επίπεδο διασύνδεσής του ενημερώνει το υπόστρωμα συναλλαγής, το οποίο επιστρέφει την τρέχουσα κατάσταση του συστήματος. Αν η επεξεργασία της αίτησης μπορεί να γίνει άμεσα, το επίπεδο διασύνδεσης αποκρίνεται με ένα πακέτο ACK που επιβεβαιώνει την επιτυχή λήψη της με τον κωδικό ack_pending. Σε αντίθετη περίπτωση το πακέτο ACK υποδεικνύει στον κόμβο που έστειλε την αίτηση να επιχειρήσει και πάλι τη συναλλαγή σε μεταγενέστερη χρονική στιγμή (retry). Αν η προς ανάγνωση περιοχή μνήμης αναφέρεται σε κάποιον καταχωρητή διαχείρισης του διαύλου, τότε το περιεχόμενό του ανακτάται από το αντίστοιχο υπόστρωμα, αλλιώς η αίτηση προωθείται στο επίπεδο εφαρμογής.

Εάν η ζητούμενη περιοχή μνήμης είναι προσβάσιμη, τότε ανακτώνται τα δεδομένα και το επίπεδο συναλλαγής εκκινεί μια υποδράση απόκρισης που τα μεταφέρει

στον κόμβο προέλευσης της αρχικής αίτησης. Στο πακέτο της απόκρισης περιλαμβάνονται επίσης οι διευθύνσεις των δυο κόμβων που συμμετέχουν στη διαδικασία, ο κωδικός επιστροφής (rcode), καθώς και ο δείκτης συναλλαγής (transaction label) από την αρχική αίτηση.

Αφού ληφθεί το πακέτο απόκρισης από το υπόστρωμα διασύνδεσης, το επίπεδο συναλλαγής του κόμβου αυτού επεξεργάζεται τα περιεχόμενα του πακέτου. Με το πέρας της επεξεργασίας αποστέλλεται η επιβεβαίωση ACK για την υποδράση της απόκρισης, φέροντας τον κατάλληλο κωδικό κατάστασης (συνήθως ack_complete), ενώ ταυτόχρονα ενημερώνεται και το επίπεδο που εκκίνησε τη συναλλαγή.

Μια συναλλαγή που αποτελείται από δυο πακέτα αίτησης και απόκρισης, για την αποστολή των οποίων πραγματοποιούνται ανεξάρτητοι κύκλοι διαιτησίας, αναφέρεται στο πρότυπο IEEE 1394 ως διαχωρισμένη συναλλαγή (split transaction). Οι συναλλαγές αυτού του τύπου είναι ίσως οι πιο συνηθισμένες, καθώς η τέλεση τους γίνεται κάτω από λιγότερο αυστηρά χρονικά περιθώρια, επιτρέποντας την χρήση κόμβων με σχετικά αργούς ρυθμούς επεξεργασίας.

Σε κάποιες περιπτώσεις όπου η επεξεργασία της αίτησης γίνεται άμεσα, είναι δυνατόν το πακέτο της απόκρισης να μεταδοθεί ενωμένο με το πακέτο ACK (concatenated response), χωρίς να πραγματοποιηθεί νέος κύκλος διαιτησίας. Αυτές οι συναλλαγές αποκαλούνται συνενωμένες συναλλαγές (concatenated transactions) και προσφέρουν βελτιωμένες επιδόσεις σε σχέση με τις ισοδύναμες διαχωρισμένες.

Στην περίπτωση που ο κόμβος προορισμού ανιχνεύσει κάποιο σφάλμα στο πακέτο της αίτησης, απαντά με ένα πακέτο ACK που φέρει την κατάλληλη ένδειξη λάθους, αλλά δεν προχωρά στην περαιτέρω επεξεργασία της συναλλαγής ή στην τέλεση της υποδράσης απόκρισης.

Σημειώνεται ότι επιτρέπεται το πολύ μία απόκριση ανά αίτηση. Συνεπώς δεν ορίζονται συναλλαγές ανάγνωσης με διεύθυνση προορισμού τη διεύθυνση μαζικής εκπομπής (broadcast) του διαύλου.

2.5.2 Συναλλαγές εγγραφής

Οι συναλλαγές εγγραφής τυπικά αποτελούνται από δύο υποδράσεις, με την ίδια βασική δομή και στάδια επεξεργασίας που έχουν και οι συναλλαγές ανάγνωσης. Το πακέτο της αίτησης προς τον απομακρυσμένο κόμβο περιέχει τα δεδομένα που θα εγγραφούν στην περιοχή μνήμης που προσδιορίζεται και η επιβεβαίωση ACK που εκπέμπεται ως απάντηση σε αυτό υποδεικνύει αν η εκτέλεση της συναλλαγής θα προχωρήσει απρόσκοπτα.

Με την ολοκλήρωση (ή την αποτυχία) της εγγραφής των δεδομένων, ο κόμβος προορισμού αποστέλλει ένα πακέτο απόκρισης που φέρει τα στοιχεία και τον κωδικό επιστροφής της συναλλαγής. Η απόκριση καταλήγει στο υπόστρωμα συναλλαγής του κόμβου προέλευσης της αίτησης, το οποίο ενημερώνει το επίπεδο που ξεκίνησε τη διαδικασία, ενώ ανταποδίδει με ένα πακέτο επιβεβαίωσης ACK.

Αν η εγγραφή των δεδομένων μπορεί να ολοκληρωθεί άμεσα με επιτυχία, τότε είναι δυνατόν η υποδράση αίτησης να μην ακολουθηθεί από μια υποδράση απόκρισης. Σε αυτή την περίπτωση ο κόμβος προορισμού ολοκληρώνει τη συναλλαγή και αποκρίνεται απευθείας με ένα πακέτο ACK με κωδικό ack_complete, έτσι ώστε ο κόμβος προέλευσης να μην αναμένει κάποιο πακέτο απόκρισης. Αυτή η διαδικασία ονομάζεται στο πρότυπο IEEE Std 1394-1995 ενοποιημένη συναλλαγή (unified transaction) και ορίζεται μόνο για την κατηγορία των συναλλαγών εγγραφής.

Ειδική περίπτωση αποτελούν οι συναλλαγές εγγραφής όταν ως διεύθυνση προορισμού χρησιμοποιείται η διεύθυνση μαζικής εκπομπής (broadcast) του διαύλου. Μια τέτοια συναλλαγή αποτελείται αποκλειστικά από το πακέτο της αρχικής αίτησης, χωρίς να έπονται πακέτα απόκρισης ή επιβεβαίωσης ACK.

2.5.3 Συναλλαγές κλειδώματος

Οι συναλλαγές κλειδώματος έχουν κοινά στοιχεία και με τους δυο τύπους συναλλαγών που προαναφέραμε. Συγκεκριμένα έχουν την ίδια βασική δομή με την ασύγχρονη ανάγνωση, με τη διαφορά ότι στην υποδράση αίτησης περιλαμβάνεται ένα τμήμα δεδομένων με τις παραμέτρους μιας ατομικής πράξης που θα τελεστεί στον κόμβο προορισμού, μεταβάλλοντας την προσδιορισμένη περιοχή μνήμης. Αντίστοιχα, το πακέτο απόκρισης περιέχει την προηγούμενη τιμή της θέσης μνήμης πάνω στην οποία έδρασε η συναλλαγή.

Το υπόστρωμα συναλλαγής οφείλει να εξασφαλίζει πως καμία συναλλαγή δεν μπορεί να εκτελεστεί παράλληλα με μια συναλλαγή κλειδώματος αν αναφέρονται και οι δύο σε κάποια κοινή περιοχή μνήμης. Αυτό επιτρέπει τη αξιοποίηση των συναλλαγών κλειδώματος για την εκτέλεση λειτουργιών συγχρονισμού όταν ένας κόμβος μπορεί να εξυπηρετήσει παράλληλα πολλαπλές αιτήσεις.

Σε περίπτωση σφάλματος στο πακέτο της αίτησης δεν πραγματοποιείται η υποδράση απόκρισης και ενημερώνεται ο κόμβος προέλευσης με τον ανάλογο κωδικό λάθους στο πακέτο επιβεβαίωσης ACK.

2.5.4 Διαχείριση λαθών και επανάληψη συναλλαγών

Το υπόστρωμα συναλλαγής ελέγχει για σφάλματα κάθε εισερχόμενη αίτηση συναλλαγής, όπως και κάθε πακέτο απόκρισης που λαμβάνει ως απάντηση σε προηγούμενη αίτησή του. Σκοπός αυτών των ελέγχων είναι να προστατευτεί όσο το δυνατόν περισσότερο το επίπεδο εφαρμογής από τη λήψη άκυρων δεδομένων. Έτσι είναι σε θέση να ανιχνεύει μια σειρά από προβληματικές καταστάσεις που ανήκουν σε δυο βασικές κατηγορίες:

- Πακέτα με αλλοιωμένα δεδομένα που ανιχνεύονται κατά την επιβεβαίωση των κωδικών εντοπισμού λαθών από το επίπεδο διασύνδεσης
- Πακέτα που καταφθάνουν με άκυρα πεδία ή χωρίς να αποτελούν μέρος κάποιας τρέχουσας συναλλαγής

Τα πακέτα αυτά απορρίπτονται άμεσα, αλλά το επίπεδο εφαρμογής υποχρεούται να ελέγχει αναλυτικά όλα τα υπόλοιπα εισερχόμενα πακέτα, καθώς και να επιβεβαιώνει την επιτυχή λήψη όλων των εξερχόμενων πακέτων όταν είναι δυνατόν. Το επίπεδο εφαρμογής θα πρέπει να διαθέτει κατάλληλο μηχανισμό ανάνηψης για την περίπτωση που εντοπίσει κάποιο λάθος, έτσι ώστε να μην επηρεάζει αρνητικά τη λειτουργία του διαύλου.

Μια ιδιαίτερη περίπτωση έχουμε όταν ο κόμβος προορισμού αποδέχεται την αίτηση, αλλά δεν μπορεί να την επεξεργαστεί άμεσα, συνήθως επειδή κάποιος απαιτούμενος πόρος είναι ήδη δεσμευμένος. Όταν συμβεί αυτό, οι δυο συμμετέχοντες κόμβοι μπορούν να χρησιμοποιήσουν ένα από δύο διαθέσιμα πρωτόκολλα επανάληψης της συναλλαγής (transaction retry protocol) που υλοποιούνται από το υπόστρωμα συναλλαγής.

Στο πρώτο και απλούστερο πρωτόκολλο επανάληψης (single-phase retry protocol) ο κόμβος προέλευσης χρησιμοποιεί έναν μετρητή για τις απόπειρες τέλεσης μιας συναλλαγής. Όταν ο μετρητής φθάσει ένα προκαθορισμένο όριο η συναλλαγή αποτυγχάνει οριστικά. Η μόνη συμμετοχή του κόμβου προορισμού σε αυτή τη διαδικασία είναι η απόρριψη κάθε συναλλαγής με τον κωδικό ack_busy_X όταν αδυνατεί να την τελέσει άμεσα.

Το δεύτερο πρωτόκολλο που περιγράφεται στο IEEE Std 1394-1995 διαχωρίζει τις εισερχόμενες συναλλαγές σε δύο ομάδες (dual-phase retry protocol), επιστρέφοντας κωδικό ack_busy_A ή ack_busy_B. Ο κόμβος προέλευσης επαναλαμβάνει την αίτηση της συναλλαγής κάθε τέσσερα διαστήματα αμεροληψίας, έως ότου εξαντληθεί ένα μέγιστο χρονικό όριο, με το πεδίο rt του πακέτου να υποδεικνύει σε ποια ομάδα ανήκει. Ταυτόχρονα, ο κόμβος προορισμού εκτελεί διαδοχικά τις συναλλαγές που είναι ήδη σε μια ομάδα, προωθώντας τις εισερχόμενες αιτήσεις στην άλλη. Η μετάβαση από τη μια ομάδα στην άλλη γίνεται όταν εξαντληθούν οι συναλλαγές στην τρέχουσα ομάδα, κάτι που διαπιστώνεται όταν δεν υπάρξουν επαναλήψεις αιτήσεων που ανήκουν σε αυτήν επί τέσσερα διαστήματα αμεροληψίας. Κόμβοι υψηλών επιδόσεων μπορούν να απαλείψουν το διάστημα αδράνειας που προκύπτει κατά τη μετάβαση χρησιμοποιώντας μετρητές για τον αριθμό των αιτήσεων κάθε ομάδας. Το πλεονέκτημα αυτού του πρωτοκόλλου είναι ότι, καθότι ανήκουν σε διαφορετικές ομάδες, οι παλαιότερες αιτήσεις δεν ανταγωνίζονται

ευθέως με τις νεότερες, αυξάνοντας έτσι δραστικά τις πιθανότητες να ολοκληρωθούν οι αντίστοιχες συναλλαγές σε λογικό χρονικό διάστημα και πριν παρέλθουν οι προκαθορισμένες προθεσμίες.

Ο μηχανισμός επανάληψης συναλλαγών υποχρεούται να λειτουργεί μέσα σε αυστηρά χρονικά όρια. Για το λόγο αυτό το πρότυπο IEEE 1394 συνιστά η υλοποίηση των πρωτοκόλλων επανάληψης να γίνεται στο υλικό και όχι σε λογισμικό, εν αντιθέσει με τις υπόλοιπες λειτουργίες του επιπέδου συναλλαγής.

2.6 Το επίπεδο διαχείρισης του διαύλου

Το επίπεδο διαχείρισης περιλαμβάνει μια σειρά από λειτουργίες που είναι απαραίτητες για την πλήρη αξιοποίηση του διαύλου IEEE 1394:

- Υλοποίηση των καταχωρητών της αρχιτεκτονικής CSR και της μνήμης ρυθμίσεων (Configuration ROM)
- Παροχή χρονισμού για ισόχρονες συναλλαγές (Cycle Master, CM)
- Διαχείριση ισόχρονων πόρων (Isochronous Resource Manager, IRM), με την παροχή μιας κεντρικής υπηρεσίας για την δέσμευση καναλιών και εύρους ζώνης
- Έκθεση σε κεντρική τοποθεσία σημαντικών στοιχείων, όπως η τοπολογία του διαύλου και οι υποστηριζόμενες ταχύτητες επικοινωνίας ανάμεσα στους κόμβους (Bus Manager - BM)
- Επιλογή της ρίζας σε τοπολογία δένδρου και των κόμβων CM, BM και IRM, καθώς και ενημέρωση των ενδιαφερόμενων κόμβων για την ταυτότητά τους
- Βελτιστοποίηση των παραμέτρων του διαύλου
- Μηχανισμοί διαχείρισης ενέργειας

Κάθε κόμβος IEEE 1394 μπορεί να ταξινομηθεί σύμφωνα με τις δυνατότητες που διαθέτει και, ανάλογα με την κατηγορία στην οποία ανήκει, μπορεί να είναι υποψήφιος για μία ή περισσότερες από τις θέσεις διαχείρισης του διαύλου. Οι δυνατότητες κάθε κόμβου μπορούν να διαπιστωθούν με την ανάγνωση της μνήμης ρυθμίσεών του και την ανάλυση των περιεχομένων της.

2.6.1 Η αρχιτεκτονική CSR

Η αρχιτεκτονική CSR, πάνω στην οποία βασίζεται ο δίαυλος IEEE 1394, προδιαγράφηκε αρχικά από το πρότυπο IEEE Std 1212-1994 IEEE Standard for a Control and Status Registers (CSR) Architecture for Microcomputer Buses (ISO/IEC 13213: 1994). Η χρήση της στους διαύλους IEEE 1394, SCI και FutureBus ανέδειξε μια σειρά από ελλείψεις και αδυναμίες, με αποτέλεσμα να συνταχθεί από την επιτροπή P1212 του IEEE η νεότερη έκδοση IEEE Std 1212-2001, με την οποία διευκρινίστηκαν και απλοποιήθηκαν αρκετά σημεία του προτύπου. Η σύνταξη του προτύπου IEEE Std 1394-1995, ωστόσο, έγινε με βάση την αρχική προδιαγραφή IEEE Std 1212-1994.

Ο δίαυλος IEEE 1394 κληρονομεί τέσσερα βασικά χαρακτηριστικά του από την αρχιτεκτονική CSR: μια ελάχιστη ομάδα υποστηριζόμενων συναλλαγών, τη μέθοδο διευθυνσιοδότησης, μια σειρά από καταχωρητές με το γενικό μηχανισμό υλοποίησής τους και τη δομή της μνήμης ρυθμίσεων (Configuration ROM). Από αυτά τα στοιχεία, οι καταχωρητές και η μνήμη ρυθμίσεων επηρεάζουν κυρίως το επίπεδο διαχείρισης του διαύλου, αλλά η μορφή των διευθύνσεων και οι παρεχόμενοι τύποι συναλλαγών έχουν καθοριστική επιρροή σε κάθε στοιχείο ενός κόμβου IEEE 1394.

2.6.1.1 Υποστήριξη συναλλαγών στην αρχιτεκτονική CSR

Η αρχιτεκτονική CSR απαιτεί την ύπαρξη συναλλαγών ανάγνωσης, εγγραφής και κλειδώματος. Αυτές υλοποιούνται στο δίαυλο IEEE 1394 με τις ασύγχρονες συναλλαγές ανάγνωσης, εγγραφής και κλειδώματος, τις οποίες περιγράψαμε αναλυτικά στις παραγράφους 2.5.1, 2.5.2 και 2.5.3 αντιστοίχως. Οι ισόχρονες συναλλαγές και οι συναλλαγές ασύγχρονης ροής αποτελούν επεκτάσεις ειδικές για το δίαυλο IEEE 1394 και δεν αποτελούν μέρος του προτύπου IEEE 1212.

2.6.1.2 Διευθυνσιοδότηση στην αρχιτεκτονική CSR

Όπως αναφέρθηκε και στην παράγραφο 2.2, κάθε δίαυλος που υλοποιεί την αρχιτεκτονική CSR χρησιμοποιεί διευθύνσεις 64 bit, με τα 16 πρώτα δυαδικά ψηφία να αποτελούν τη διεύθυνση ενός κόμβου και τα υπόλοιπα 48 να ορίζουν μια θέση στο χώρο διευθύνσεων του κόμβου αυτού. Στο δίαυλο IEEE 1394, από τα 16 bit της φυσικής διεύθυνσης του κόμβου τα 10 υποδεικνύουν έναν ακριβώς δίαυλο, ενώ τα υπόλοιπα 6 έναν συγκεκριμένο κόμβο συνδεδεμένο σε αυτόν. Όμοια ο χώρος διευθύνσεων του κόμβου έχει χωριστεί σε υποχώρους, με τη νοητή διαίρεση των 48 bit που τον καλύπτουν σε δυο πεδία των 20 και 28 δυαδικών ψηφίων. Σχηματικά μια διεύθυνση στο δίαυλο IEEE 1394 έχει την ακόλουθη μορφή:

Σχήμα 34: Δομή διεύθυνσης IEEE 1394

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62
bid					nid			Υποχώρος διευθύνσεων										Διεύθυνση													

Πίνακας 28: Πεδία διεύθυνσης IEEE 1394

Πεδίο	Μέγεθος	Περιγραφή
bid	10	Αριθμός ταυτοποίησης του διαύλου (bus ID)
nid	6	Αριθμός ταυτοποίησης του κόμβου (node ID)

Από τους διαθέσιμους υποχώρους, δύο είναι δεσμευμένοι από την αρχιτεκτονική CSR, ενώ οι υπόλοιποι αναφέρονται στο χώρο μνήμης του κόμβου. Συνοπτικά, ο χώρος διευθύνσεων ενός κόμβου κατανέμεται όπως φαίνεται στον πίνακα 29.

Πίνακας 29: Χώρος διευθύνσεων κόμβου IEEE 1394

Όνομα	Διεύθυνση	Μέγεθος (bytes)
Χώρος μνήμης	000000000000_{16}	256T - 512M
Ιδιωτικός χώρος	$ffffe0000000_{16}$	256M
Χώρος καταχωρητών	$fffff0000000_{16}$	256M
∘ Καταχωρητές CSR	$fffff0000000_{16}$	512
∘ Καταχωρητές διαύλου	$fffff0000200_{16}$	512
∘ Μνήμη ρυθμίσεων	$fffff0000400_{16}$	1024
∘ Χώρος μονάδας	$fffff0000800_{16}$	256M - 2048

Ο χώρος μνήμης (memory space) καταλαμβάνει σχεδόν το σύνολο του διαθέσιμου εύρους διευθύνσεων. Συνήθως η περιοχή αυτή ανταποκρίνεται άμεσα στη φυσική μνήμη του κόμβου και η πρόσβαση σε αυτήν θεωρείται ότι δεν έχει παρενέργειες (side-effects), ωστόσο αυτό είναι ζήτημα πρακτικής και όχι κάποια τυποποιημένη προδιαγραφή. Στην περίπτωση ενός υπολογιστή, τα σχεδόν 256 TiB που είναι διαθέσιμα σε αυτόν τον χώρο είναι σε θέση να καλύψουν το σύνολο της

φυσικής μνήμης χωρίς δυσκολία, τουλάχιστον στο προσεχές μέλλον. Παρ΄ όλα αυτά, όμως, η απευθείας αντιστοίχιση στο σύνολο των φυσικών διευθύνσεων του κόμβου μπορεί να είναι αδύνατη σε αρχιτεκτονικές υπολογιστών όπου η μνήμη δεν βρίσκεται σε μια συνεκτική (contiguous) περιοχή διευθύνσεων, όπως τα συστήματα NUMA (Non-Uniform Memory Access), ή σε περιπτώσεις όπου συσκευές εισόδου/εξόδου έχουν για κάποιο λόγο τμήματα μνήμης I/O με διευθύνσεις πέρα από το εύρος των 48 bit του σειριακού διαύλου. Η δεύτερη εκδοχή είναι σαφώς πιθανή σε συστήματα που διαθέτουν επεξεργαστή και διαύλους με εύρος διευθύνσεων 64 bit, τα οποία είναι αρκετά συνηθισμένα στην αγορά των δικτυακών διακομιστών.

Ο ιδιωτικός υποχώρος (private space) είναι δεσμευμένος για χρήσεις ειδικές στον τοπικό κόμβο. Μια τυπική χρήση αυτού του τύπου είναι η εξαγωγή ενός εσωτερικού ενταμιευτή προς το δίαυλο IEEE 1394.

Ο υποχώρος των καταχωρητών (register space) είναι δεσμευμένος για τους πόρους εκείνους που περιγράφονται από το IEEE 1212 και τα πρότυπα που βασίζονται σε αυτό. Σε αυτό το χώρο περιλαμβάνονται οι καταχωρητές που προβλέπονται από την αρχιτεκτονική CSR (CSR architecture registers), οι καταχωρητές που είναι ειδικοί στο δίαυλο (bus-dependent registers), η μνήμη ρυθμίσεων (Configuration ROM), καθώς και ένα εύρος διευθύνσεων μονάδας (units space) που προορίζεται για την προτυποποίηση της λειτουργίας του κόμβου. Το πρότυπο IEEE 1394 χρησιμοποιεί το αρχικό τμήμα του χώρου διευθύνσεων μονάδας (initial units space) για τους καταχωρητές που προβλέπει για τη διαχείριση του διαύλου.

2.6.1.3 Καταχωρητές στο δίαυλο IEEE 1394

Ο σειριακός δίαυλος IEEE 1394 χρησιμοποιεί μια σειρά από καταχωρητές, των οποίων η λειτουργία περιγράφεται στο πρότυπο IEEE 1212. Αυτό επιτρέπει την κοινή τέλεση κάποιων λειτουργιών, όπως για παράδειγμα η επανεκκίνηση του συστήματος, στην περίπτωση διασύνδεσής του με άλλους διαύλους που υποστηρίζουν την αρχιτεκτονική CSR μέσω γεφυρών.

Πίνακας 30: Βασικοί καταχωρητές αρχιτεκτονικήςCSR

Καταχωρητής	Διεύθυνση[1]	Μέγεθος (bytes)	Περιγραφή
STATE_CLEAR[2]	000_{16}	4	Γενικός καταχωρητής κατάστασης και ελέγχου του διαύλου
STATE_SET[2]	004_{16}	4	Χρησιμοποιείται για τη μεταβολή του καταχωρητή STATE_CLEAR
NODE_IDS[2]	008_{16}	4	Επιτρέπει τη μεταβολή της φυσικής διεύθυνσης του κόμβου
RESET_START[2]	$00c_{16}$	4	Επιτρέπει την επανεκκίνηση του κόμβου
SPLIT_TIMEOUT	$00c_{16}$	8	Μέγιστος χρόνος ανάμεσα στα δύο πακέτα μιας διαχωρισμένης συναλλαγής

[1]Διεύθυνση σχετική ως προς την αρχή του χώρου καταχωρητών

[2]Καταχωρητής που υλοποιείται υποχρεωτικά σε κάθε κόμβο

Επιπροσθέτως, το πρότυπο IEEE 1394 προδιαγράφει μια σειρά από καταχωρητές διαύλου και μονάδας, οι πιο βασικοί από τους οποίους περιγράφονται συνοπτικά στους πίνακες 31 και 32.

Η πρόσβαση στους καταχωρητές αυτούς γίνεται με απλές ασύγχρονες συναλλαγές στη διεύθυνση που ανταποκρίνεται σε αυτούς. Πολλοί κόμβοι, ανάλογα με το σχεδιασμό τους, δεν είναι σε θέση να χειριστούν συναλλαγές που αναφέρονται σε περισσότερους από έναν καταχωρητές ταυτόχρονα. Για το λόγο αυτό, συνίσταται να χρησιμοποιούνται πολλαπλές συναλλαγές, κάθε μια με ποσότητα δεδομένων που αντιστοιχεί ακριβώς στο μέγεθος του εκάστοτε καταχωρητή. Σε περιπτώσεις όπου πρέπει να εξασφαλιστεί η ατομικότητα των προσβάσεων είναι υποχρεωτική η τέλεση συναλλαγών κλειδώματος, εν αντιθέσει με τις απλές συναλλαγές εγγραφής με τις οποίες δεν μπορεί να εξασφαλιστεί η αποφυγή παθολογικών καταστάσεων ανταγωνισμού (race conditions) κατά την τέλεση ταυτόχρονων εγγραφών σε έναν καταχωρητή.

Πίνακας 31: Βασικοί καταχωρητές διαύλου IEEE 1394

Καταχωρητής	Διεύθυνση[1]	Μέγεθος (bytes)	Περιγραφή
CYCLE_TIME	200_{16}	4	Καταχωρητής χρονισμού για ισόχρονες συναλλαγές
BUS_TIME	204_{16}	4	Ρολόι διαύλου με ακρίβεια δευτερολέπτου
BUSY_TIMEOUT	208_{16}	4	Μέρος του πρωτοκόλλου επανάληψης συναλλαγών
BUS_MANAGER_ID[2]	$21c_{16}$	4	Χρησιμοποιείται για την επιλογή και τον εντοπισμό του διαχειριστή διαύλου
BANDWIDTH_AVAILABLE[2]	220_{16}	4	Δέσμευση εύρους ζώνης για ισόχρονες συναλλαγές
CHANNELS_AVAILABLE[2]	224_{16}	8	Δέσμευση ισόχρονων καναλιών
BROADCAST_CHANNEL[2]	234_{16}	8	Ισόχρονο κανάλι δεσμευμένο για ασύγχρονες συναλλαγές ροής

[1]Διεύθυνση σχετική ως προς την αρχή του χώρου καταχωρητών

[2]Η εγγραφή σε αυτόν τον καταχωρητή πρέπει να γίνεται με συναλλαγές κλειδώματος

Πίνακας 32: Καταχωρητές μονάδας IEEE 1394

Καταχωρητής	Διεύθυνση[1]	Μέγεθος (bytes)	Περιγραφή
TOPOLOGY_MAP	1000_{16}	1024 (max.)	Έγκυρος μόνο στο διαχειριστή διαύλου, περιέχει όλα τα πακέτα Self-ID που εκπέμφθηκαν, έτσι ώστε να μπορεί να σκιαγραφηθεί η τοπολογία του διαύλου
SPEED_MAP[2]	2000_{16}	4096 (max.)	Έγκυρος μόνο στο διαχειριστή διαύλου, παρέχει τη μέγιστη υποστηριζόμενη ταχύτητα μεταφοράς δεδομένων ανάμεσα σε κάθε δυνατό ζεύγος κόμβων στο δίαυλο

[1]Διεύθυνση σχετική ως προς την αρχή του χώρου καταχωρητών

[2]Καταχωρητής που υλοποιείται προαιρετικά

2.6.1.4 Η μνήμη ρυθμίσεων (Configuration ROM)

Η μνήμη ρυθμίσεων (Configuration ROM) αποτελεί ένα από τα πιο βασικά στοιχεία της αρχιτεκτονικής CSR και, κατά συνέπεια, του διαύλου IEEE 1394. Αποτελείται από μια συστοιχία συνδεδεμένων δομών δεδομένων οι οποίες περιγράφουν τα ιδιαίτερα χαρακτηριστικά του κόμβου. Η πρόσβαση στη μνήμη ρυθμίσεων γίνεται με τη χρήση ασύγχρονων συναλλαγών ανάγνωσης, ενώ τα περιεχόμενά της δεν μπορούν να μεταβληθούν από άλλους κόμβους.

Τα περιεχόμενα της μνήμης ρυθμίσεων είναι οργανωμένα σε μια δενδρική δομή, με τη ρίζα να βρίσκεται στην αρχή του χώρου διευθύνσεων Configuration ROM. Οι βασικές μονάδες της είναι ο κατάλογος (directory) και το φύλλο (leaf). Ο κατάλογος είναι μια δομή που περιέχει μια σειρά από εγγραφές, οι οποίες μπορεί να είναι σταθερές τιμές, καθώς και δείκτες προς τις θέσεις φύλλων ή άλλους καταλόγους. Ο κατάλογος είναι το βασικό δομικό στοιχείο με το οποίο κατασκευάζεται η δενδρική ιεραρχία της μνήμης και έχει την ακόλουθη μορφή:

Σχήμα 35: Μορφή καταλόγου μνήμης ρυθμίσεων

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">length</td><td colspan="16">crc</td></tr>
<tr><td colspan="8">key</td><td colspan="24">value</td></tr>
<tr><td colspan="8">key
...
key</td><td colspan="24">value
...
value</td></tr>
<tr><td colspan="8">key</td><td colspan="24">value</td></tr>
</table>

Πίνακας 33: Πεδία καταλόγου μνήμης ρυθμίσεων

Πεδίο	Μέγεθος	Περιγραφή
length	16	Αριθμός τετράδων του καταλόγου, χωρίς την αρχική
crc	16	Κωδικός ελέγχου σφαλμάτων CRC-16 των τετράδων που ακολουθούν
key	8	Τύπος εγγραφής
value	24	Περιεχόμενο εγγραφής

Ο ριζικός κατάλογος έχει επιπρόσθετα ένα πρόθεμα γενικής περιγραφής του κόμβου, οπότε έχει τη δομή του πίνακα 36. Τα πεδία node_vendor_id και chip_id του προθέματος συνιστούν έναν μοναδικό κωδικό αριθμό ταυτοποίησης (Unique ID, UID) που επιτρέπει την αναγνώριση της συσκευής από το λογισμικό ελέγχου.

Τα φύλλα της δενδρικής δομής μπορούν να παρέχουν μια ποικιλία πληροφοριών, όπως για παράδειγμα το φύλλο Node_Unique_Id, που παρέχει ένα αντίγραφο του κωδικού UID του κόμβου. Ο πιο κοινός τύπος φύλλου, όμως, είναι το φύλλο περιγραφής (descriptor) που μπορεί να φιλοξενήσει δομημένες πληροφορίες, όπως κείμενο (textual descriptor) ή ακόμα και εικονίδια (icon descriptors) για οπτική

Σχήμα 36: Μορφή ριζικού καταλόγου μνήμης ρυθμίσεων

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="8">info_length</td><td colspan="8">crc_length</td><td colspan="16">crc</td></tr>
<tr><td colspan="8">"1"</td><td colspan="8">"3"</td><td colspan="8">"9"</td><td colspan="8">"4"</td></tr>
<tr><td>irmc</td><td>cmc</td><td>isc</td><td>bmc</td><td colspan="4">r</td><td colspan="8">cyc_clk_acc</td><td colspan="4">max_rec</td><td colspan="12">r</td></tr>
<tr><td colspan="24">node_vendor_id</td><td colspan="8">chip_id_hi</td></tr>
<tr><td colspan="32">chip_id_lo</td></tr>
<tr><td colspan="16">root_length</td><td colspan="16">root_crc</td></tr>
<tr><td colspan="8">key</td><td colspan="24">value</td></tr>
<tr><td colspan="8">key
...
key</td><td colspan="24">value
...
value</td></tr>
<tr><td colspan="8">key</td><td colspan="24">value</td></tr>
</table>

Πίνακας 34: Πεδία ριζικού καταλόγου μνήμης ρυθμίσεων

Πεδίο	Μέγεθος	Περιγραφή
info_length	8	Αριθμός τετράδων προθέματος, χωρίς την αρχική
crc_length	8	Αριθμός τετράδων στον υπολογισμό του κωδικού CRC-16, χωρίς την αρχική
crc	16	Κωδικός ελέγχου σφαλμάτων CRC-16
irmc	1	1 αν ο κόμβος μπορεί να γίνει διαχειριστής ισόχρονων πόρων
cmc	1	1 αν ο κόμβος μπορεί να λειτουργήσει χρονιστής ισόχρονων συναλλαγών
isc	1	1 αν ο κόμβος έχει δυνατότητα ισόχρονων συναλλαγών
bmc	1	1 αν ο κόμβος μπορεί να γίνει διαχειριστής διαύλου
r		Δεσμευμένο
cyc_clk_acc	8	Ακρίβεια χρονιστή ισόχρονων συναλλαγών
max_rec	4	Μέγιστο φορτίο ασύγχρονης συναλλαγής εγγραφής προς τον κόμβο, ορίζεται ως 2^{max_rec+1}
node_vendor_id	24	Χαρακτηριστικός κωδικός αριθμός κατασκευαστή της συσκευής
chip_id_hi chip_id_lo	8 32	Χαρακτηριστικός κωδικός αριθμός μοντέλου της συσκευής
root_length	16	Αριθμός τετράδων του ριζικού καταλόγου, χωρίς την αρχική
root_crc	16	Κωδικός ελέγχου σφαλμάτων CRC-16 των τετράδων του ριζικού καταλόγου που ακολουθούν

αναγνώριση των συσκευών σε γραφικά περιβάλλοντα διαχείρισης του διαύλου. Η γενική δομή του φύλλου περιγραφής παρουσιάζεται στο σχήμα 37.

Σχήμα 37: Μορφή φύλλου περιγραφής

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

length	crc
type	specifier_ID
key ... key	value ... value
key	value

Πίνακας 35: Πεδία φύλλου περιγραφής

Πεδίο	Μέγεθος	Περιγραφή
length	16	Αριθμός τετράδων του φύλλου, χωρίς την αρχική
crc	16	Κωδικός ελέγχου σφαλμάτων CRC-16 των τετράδων που ακολουθούν
type	8	Γενική κατηγορία φύλλου περιγραφής: 0: Κείμενο 1: Εικόνα
specifier_ID	24	Κωδικός λειτουργίας του φύλλου, συνήθως είναι 0

Τα πιο αναγνωρίσιμα φύλλα περιγραφής είναι τα φύλλα κειμένου, καθώς συνήθως χρησιμοποιούνται για την αναγραφή του κατασκευαστή και του μοντέλου κάθε κόμβου, όπως επίσης και άλλων χρήσιμων πληροφοριών, σε μορφή φιλική προς τον άνθρωπο. Τα περιεχόμενά τους συχνά μπορούν να προβληθούν από όσους κόμβους διαθέτουν κάποιο εξελιγμένο περιβάλλον επιτήρησης του διαύλου από το χρήστη, όπως, λόγου χάρη, ένας υπολογιστής.

Το πρότυπο IEEE 1394 παρέχει ελάχιστες προσθήκες επί της δομής της μνήμης ρυθμίσεων, διευκρινίζοντας μόνο εκείνα τα τμήματα που στο IEEE 1212 είναι προκαθορισμένο ότι εξαρτώνται από την εκάστοτε υλοποίηση της αρχιτεκτονικής CSR. Ωστόσο, η προδιαγραφή της αρχιτεκτονικής παρέχει πλήθος δομικών μονάδων που προορίζονται για την επέκταση της μνήμης ρυθμίσεων από τα πρωτόκολλα που περιγράφουν τις υψηλότερες λειτουργίες των κόμβων του διαύλου.

2.6.2 Επαναλήπτες

Όλοι οι κόμβοι μιας δενδρικής τοπολογίας με περισσότερες από μια θύρες εκτελούν εξ ορισμού τη λειτουργία του επαναλήπτη (Repeater). Αυτό είναι το ελάχιστο επίπεδο λειτουργικότητας που απαιτείται από έναν κόμβο IEEE 1394 και υλοποιείται απλά με την παρουσία ενός ενεργού φυσικού υποστρώματος, το οποίο μπορεί και να τροφοδοτείται από το δίαυλο. Ένας κόμβος-επαναλήπτης υποχρεούται:

- Να φέρει ενεργό φυσικό επίπεδο
- Να αναμεταδίδει με ακρίβεια τα δεδομένα που λαμβάνει από μια θύρα του προς όσες από τις υπόλοιπες είναι συνδεδεμένες
- Να συμμετέχει στις διαδικασίες αυτοδιαμόρφωσης του διαύλου και διαιτησίας
- Να μπορεί να λειτουργήσει ως κόμβος-ρίζα
- Όταν λαμβάνει ένα πακέτο ρύθμισης του φυσικού επιπέδου, να αλλάζει τις παραμέτρους της λειτουργίας του αναλόγως

2.6.3 Κόμβοι με ικανότητα τέλεσης συναλλαγών

Όταν ένας κόμβος διαθέτει ενεργά επίπεδα διασύνδεσης και συναλλαγής, τότε έχει τη δυνατότητα να εκκινεί και να αποκρίνεται σε ασύγχρονες συναλλαγές (Transaction capable). Αν τα επίπεδα αυτά υφίστανται αλλά είναι ανενεργά, τότε ένα πακέτο PHY Link-on οδηγεί στον προβιβασμό του κόμβου από επαναλήπτη σε κόμβο ικανό συναλλαγών. Οι κόμβοι αυτής της κατηγορίας οφείλουν:

- Να έχουν ενεργό φυσικό επίπεδο
- Στην περίπτωση της δενδρικής τοπολογίας να έχουν όλες τις δυνατότητες ενός επαναλήπτη
- Να έχουν ενεργά επίπεδα διασύνδεσης και συναλλαγής
- Να υλοποιούν τους καταχωρητές STATE_CLEAR, STATE_SET, NODE_IDS, RESET_START και SPLIT_TIMEOUT

Οι κόμβοι με ικανότητα τέλεσης συναλλαγών περιέχουν ένα ακόμα ενεργό τμήμα, τον ελεγκτή κόμβου (node controller), ο οποίος υλοποιεί τους καταχωρητές CSR και συντονίζει τη λειτουργία των υπόλοιπων μερών του κόμβου. Σε πολλές συσκευές το επίπεδο εφαρμογής υλοποιείται στον ελεγκτή κόμβου και όχι σε κάποιο διακριτό σχηματισμό.

2.6.4 Κόμβοι με ικανότητα τέλεσης ισόχρονων συναλλαγών

Όποιος κόμβος μπορεί να λάβει ή να αποστείλει ισόχρονα πακέτα είναι ικανός ισόχρονων συναλλαγών (Isochronous capable), οπότε, με βάση το πρότυπο, πρέπει:

- Να έχει όλες τις δυνατότητες ενός κόμβου με ικανότητα τέλεσης συναλλαγών
- Να υλοποιεί τον καταχωρητή CYCLE_TIME και ένα ρολόι 24.576 MHz που τον ενημερώνει συνεχώς
- Να περιέχει μνήμη ρυθμίσεων (Configuration ROM) για την περιγραφή των δυνατοτήτων του στους άλλους κόμβους του διαύλου

2.6.5 Κόμβοι με δυνατότητα παροχής κύκλων χρονισμού ισόχρονων συναλλαγών

Για την τέλεση ισόχρονων συναλλαγών, είναι απολύτως απαραίτητος τουλάχιστον ένας κόμβος με δυνατότητα παροχής κύκλων χρονισμού (Cycle Master capable). Οι κόμβοι αυτοί υποχρεούνται:

- Να έχουν όλες τις δυνατότητες ενός κόμβου με ικανότητα τέλεσης ισόχρονων συναλλαγών
- Να μπορούν να εκπέμπουν στο δίαυλο πακέτα έναρξης ισόχρονου κύκλου (Cycle Start) με συχνότητα 8 kHz
- Να διαθέτουν τον καταχωρητή BUS_TIME

Με βάση το πρότυπο IEEE 1394, ο κόμβος που λαμβάνει τη θέση της ρίζας στην τοπολογία δένδρου πρέπει πάντοτε να είναι ο παροχέας κύκλων χρονισμού, έτσι ώστε να υποστηρίζονται σωστά οι ισόχρονες λειτουργίες. Αν μετά τη φάση της αυτοδιαμόρφωσης η ρίζα δεν έχει δυνατότητες CM, τότε πρέπει να επιλέγεται ένας άλλος κόμβος χωρίς αυτήν την αδυναμία και να επανεκκινείται ο δίαυλος. Η επιλογή αυτή γίνεται από το διαχειριστή διαύλου (BM) ή από το διαχειριστή ισόχρονων πόρων (IRM), όταν ο πρώτος δεν υπάρχει. Η διαδικασία ολοκληρώνεται με την ενεργοποίηση της γεννήτριας κύκλων χρονισμού από τον κόμβο διαχείρισης. Η ενεργοποίηση του τμήματος CM δεν είναι απαραίτητη αν με την επανεκκίνηση του διαύλου δεν άλλαξε ο κόμβος-ρίζα, οπότε και συνεχίζει κανονικά την εκπομπή πακέτων Cycle Start.

2.6.6 Κόμβοι με δυνατότητα διαχείρισης ισόχρονων πόρων

Όπως και οι κόμβοι με δυνατότητα CM, έτσι και οι κόμβοι με ικανότητα διαχείρισης ισόχρονων πόρων (Isochronous Resource Manager capable) είναι απαραίτητοι για την τέλεση ισόχρονων συναλλαγών. Με βάση την τροπολογία IEEE Std 1394a-2000 θα πρέπει να πληρούν τις ακόλουθες προϋποθέσεις:

- Να έχουν όλες τις δυνατότητες ενός κόμβου με ικανότητα τέλεσης συναλλαγών
- Να υλοποιούν τους καταχωρητές BUS_MANAGER_ID, BANDWIDTH_AVAILABLE, CHANNELS_AVAILABLE και BROADCAST_CHANNEL
- Στην τοπολογία δένδρου να μπορούν να αναλύσουν τα πακέτα Self-ID που λαμβάνουν, ώστε να προσδιορίσουν τελικά τη φυσική διεύθυνση του κόμβου IRM
- Να περιέχουν μνήμη ρυθμίσεων (Configuration ROM) για την περιγραφή των δυνατοτήτων τους
- Σε περίπτωση απουσίας κόμβου με ικανότητα διαχείρισης διαύλου, να μπορούν να επιλέξουν τον κόμβο-ρίζα, ώστε να έχει δυνατότητες CM

Κατά τη διαδικασία αυτοδιαμόρφωσης του διαύλου ένας ακριβώς από τους κόμβους που διαθέτουν την δυνατότητα επιλέγεται ως κόμβος IRM και είναι υπεύθυνος για την κατανομή του διαθέσιμου εύρους ζώνης σε ισόχρονα κανάλια. Έως και 80% του συνολικού εύρους ζώνης μπορεί να κατανεμηθεί για την πραγματοποίηση ισόχρονων συναλλαγών. Εδώ πρέπει να σημειωθεί ότι ο κόμβος IRM, παρά την ονομασία του, έχει λειτουργία ρυθμιστική, αλλά δεν μπορεί να επιβάλλει τα όρια που έχουν τεθεί. Ουσιαστικά αποτελεί μια κεντρική τοποθεσία όπου οι κόμβοι του διαύλου μπορούν να κατανείμουν συλλογικά τους διαθέσιμους πόρους. Αυτή η εξάρτηση της καλής λειτουργίας από τη σωστή συμπεριφορά των επιμέρους κόμβων θα μπορούσε να θεωρηθεί τουλάχιστον προβληματική σε έναν δίαυλο περιφερειακών, αλλά είναι αποδεκτή στην περίπτωση του IEEE 1394, του οποίου ο σχεδιασμός προσανατολιζόταν εξ αρχής προς τους διαύλους επέκτασης συστήματος.

Για την επιλογή του κόμβου IRM κατά τη φάση αυτοδιαμόρφωσης, όλοι οι κόμβοι με την αντίστοιχη δυνατότητα παρατηρούν τα πακέτα Self-ID που αποστέλλονται στο δίαυλο. Τα πακέτα που προέρχονται από τους πιθανούς κόμβους IRM έχουν ενεργά τα πεδία l και i, επιτρέποντας τον εντοπισμό τους. Τη θέση του κόμβου διαχείρισης ισόχρονων πόρων λαμβάνει τελικά ο υποψήφιος κόμβος με τη μεγαλύτερη φυσική διεύθυνση.

Κατά την επανεκκίνηση του διαύλου όλες οι ασύγχρονες και ισόχρονες συναλλαγές παύουν. Οι ασύγχρονες συναλλαγές μπορούν να συνεχιστούν αμέσως μετά το τέλος της αρχικοποίησης του διαύλου, αλλά για την συνέχιση των ισόχρονων συναλλαγών απαιτείται από όλους τους εμπλεκόμενους κόμβους η επανάληψη της διαδικασίας δέσμευσης καναλιών και εύρους ζώνης από τον νέο IRM.

Ο καταχωρητής BROADCAST_CHANNEL αποτελεί προσθήκη της τοπολογίας IEEE Std 1394a-2000 και περιέχει τον αριθμό του καναλιού που θα χρησιμοποιείται για τις ασύγχρονες συναλλαγές ροής, ενώ η αρχική του τιμή είναι 31. Υπάρχει, συνεπώς, η περίπτωση επιλογής ενός κόμβου IRM που υποστηρίζει το αρχικό πρότυπο IEEE Std 1394-1995 και δε διαθέτει αυτόν τον καταχωρητή. Σε αυτήν την περίπτωση αν υπάρχει άλλος υποψήφιος IRM που υποστηρίζει πλήρως το IEEE Std 1394a-2000, τότε προκαλεί επανεκκίνηση του διαύλου και αναλαμβάνει τη θέση της ρίζας, εξασφαλίζοντας έτσι ότι θα λάβει και τη θέση του κόμβου IRM.

Με τη χρήση του καταχωρητή BUS_MANAGER_ID, ο κόμβος IRM αποτελεί επίσης την επίσημη πηγή για την φυσική διεύθυνση του διαχειριστή διαύλου BM, όταν αυτός υπάρχει. Σε περίπτωση που δεν υφίσταται κάποιος κόμβος με τις απαιτούμενες προδιαγραφές για τη θέση BM, ο κόμβος IRM μπορεί να επιφορτιστεί με κάποιες επιπλέον ευθύνες:

- Βελτιστοποίηση των διαστημάτων αδράνειας του διαύλου
- Περιορισμένης κλίμακας διαχείριση ενέργειας
- Επιλογή του κόμβου-ρίζα της δενδρικής τοπολογίας

Στην περίπτωση που σε έναν δίαυλο δεν υπάρχει κόμβος με δυνατότητες IRM, ο δίαυλος παραμένει χωρίς διαχειριστή (unmanaged) και μπορούν να πραγματοποιηθούν μόνο ασύγχρονες συναλλαγές.

2.6.7 Κόμβοι με ικανότητα διαχείρισης διαύλου

Οι κόμβοι με ικανότητα διαχείρισης διαύλου (Bus master capable) αποτελούν την πιο ικανή κατηγορία κόμβων, καθώς διαθέτουν τις ακόλουθες δυνατότητες:

- Έχουν όλες τις δυνατότητες ενός κόμβου με ικανότητα διαχείρισης ισόχρονων πόρων
- Υλοποιούν τους καταχωρητές TOPOLOGY_MAP για την περιγραφή της τοπολογίας του διαύλου
- Επιλογή του κόμβου-ρίζα, έτσι ώστε να έχει δυνατότητες CM
- Προαιρετικά:
  - Εκτελούν λειτουργίες βελτιστοποίησης του διαύλου, συνήθως με την επιλογή της ρίζας και την ρύθμιση των διαστημάτων αδράνειας (gap count optimisation)
  - Ελέγχουν τα χαρακτηριστικά διαχείρισης ενέργειας όσων κόμβων τα υποστηρίζουν

Όπως και ο κόμβος IRM, ο διαχειριστής του διαύλου γενικά δεν μπορεί να επιβάλλει την εφαρμογή των ορίων που έχουν τεθεί. Ωστόσο αρκετές από τις λειτουργίες του, όπως η βελτιστοποίηση των διαστημάτων αδράνειας, επηρεάζουν ευθέως τη λειτουργία των συνδεδεμένων κόμβων.

Για την επιλογή του διαχειριστή διαύλου χρησιμοποιείται ο καταχωρητής BUS_MANAGER_ID του IRM. Ο καταχωρητής αυτός περιέχει αρχικά την τιμή $3F_{16}$, που είναι προφανώς άκυρη καθώς δεν υποδεικνύει κάποιο συγκεκριμένο κόμβο. Με το πέρας της διαδικασίας Self-ID κατά τη διαμόρφωση του διαύλου, κάθε κόμβος με δυνατότητες IRM χρησιμοποιεί μια συναλλαγή κλειδώματος με την οποία επιχειρεί να αντικαταστήσει (compare and swap) την τιμή $3F_{16}$ με τη φυσική του διεύθυνση. Ο κόμβος που θα το επιτύχει αναλαμβάνει το ρόλο του διαχειριστή διαύλου.

Στην περίπτωση της επανεκκίνησης του διαύλου είναι επιθυμητό να παραμείνει ως διαχειριστής ο ίδιος κόμβος, έτσι ώστε να διατηρηθούν οι ίδιες ρυθμίσεις. Για το λόγο αυτό, ενώ ο κόμβος που είχε ήδη τη θέση BM επιχειρεί άμεσα να επικοινωνήσει με τον IRM, οι υπόλοιποι υποψήφιοι καθυστερούν για ένα χρονικό διάστημα τουλάχιστον 125 ms.

2.7 Θεωρητική απόδοση του διαύλου IEEE 1394

Με βάση τις προηγούμενες παραγράφους θα υπολογίσουμε το θεωρητικό μέγιστο ρυθμό μεταφοράς δεδομένων από και προς το επίπεδο εφαρμογής. Για να γίνει αυτός ο υπολογισμός πρέπει να ληφθούν υπόψη οι χρόνοι μετάδοσης δεδομένων που δεν αποτελούν μέρος του φορτίου των συναλλαγών, καθώς και τα διάφορα διαστήματα αδράνειας.

Θεωρούμε μια βέλτιστη διάταξη δύο ιδανικών κόμβων, οι οποίοι επικοινωνούν μεταξύ τους με συνδέσεις μηδενικής καθυστέρησης χωρίς εξωτερικές παρεμβολές και έχουν μηδενικό χρόνο επεξεργασίας και απόκρισης. Έτσι οι μόνες καθυστερήσεις που εισάγονται είναι αυτές που επιβάλλονται από το πρότυπο IEEE 1394. Χρησιμοποιώντας μια τοπολογία με μόνο δύο κόμβους ελαχιστοποιούνται οι χρόνοι διάδοσης πληροφορίας στο δίαυλο, ενώ τα αποτελέσματα των υπολογισμών μας καθίστανται συγκρίσιμα με τις πειραματικές μετρήσεις που πραγματοποιήθηκαν. Επιπροσθέτως, θα εστιάσουμε σε διαύλους με ταχύτητα μεταφοράς δεδομένων S400, καθώς αποτελούν πλέον την πιο συνηθισμένη περίπτωση.

2.7.1 Ασύγχρονες συναλλαγές

Από την παράγραφο 2.4.2 και τις υποπαραγράφους της προκύπτει ο συνοπτικός πίνακας 36 για τα μεγέθη των πακέτων που συνιστούν τις ασύγχρονες συναλλαγές ανάγνωσης και εγγραφής. Παρατηρούμε ότι κάθε πακέτο περιέχει έναν αριθμό τετράδων πλέον του φορτίου, ο οποίος όμως παραμένει σταθερός. Συνεπώς είναι αναμενόμενο ότι η απόδοση του διαύλου βελτιώνεται όταν αυξάνεται η ποσότητα δεδομένων που μεταφέρεται με κάθε πακέτο, καθώς μειώνεται έτσι η επίδραση των επιπλέον τετράδων.

Πίνακας 36: Μεγέθη ασύγχρονων πακέτων σε bytes

Συναλλαγή	Αίτηση	Απόκριση
Ανάγνωση τετράδας	16	20
Εγγραφή τετράδας	20	16
Ανάγνωση τμήματος	20	$24 + L$
Εγγραφή τμήματος	$24 + L$	16

L: Φορτίο πακέτου σε bytes, κανονικοποιημένο στην επόμενη τετράδα

Για τα θεωρητικά όρια στο χρονισμό των συναλλαγών ανατρέχουμε στον πίνακα 4.32 και την παράγραφο E.1.1 του IEEE Std 1394-1995, όπως επίσης και στην παράγραφο 8.7 της τροπολογίας IEEE Std 1394a-2000. Με αυτή τη βάση, ο ελάχιστος χρόνος που απαιτείται για την πραγματοποίηση της διαιτησίας στην τοπολογία που περιγράψαμε δίνεται από τον ακόλουθο τύπο:

$$arb_t = arb_delay + arb_signalling_t + MIN_DATA_PREFIX \Rightarrow$$
$$arb_t \approx gap_count * 4/S100 + 0.14\mu s \approx gap_count * 0.04069\mu s + 0.1\mu s \Rightarrow$$
$$arb_t \approx gap_count * 0.04069\mu s + 0.14\mu s$$

Για τη μετάδοση ενός πακέτου μεγέθους L bytes με ταχύτητα S χρειάζεται χρόνος $pkt_t = (8 * L)/S + DATA_END_TIME$, που για ρυθμό μετάδοσης δεδομένων S400 δίνει:

$$pkt_t(L) \approx (8 * L)/393.216\mu s + 0.24\mu s \Rightarrow$$
$$pkt_t(L) \approx L * 0.02035\mu s + 0.24\mu s$$

Τελικά, για τη μετάδοση του πακέτου επιβεβαίωσης απαιτείται ελάχιστος χρόνος

$$ack_t = RESPONSE_TIME + MIN_DATA_PREFIX + ack_pkt_t + DATA_END_TIME$$

που τελικά δίνει:

$$ack_t \approx 0.44035\mu s$$

Σχήμα 38: Τέλεση υποδράσης στο χρόνο

subaction gap	arb	DP	packet	DE	ack gap	DP	ack	DE	subaction gap

arb: Διαιτησία
DP: Σήμα Data Prefix
DE: Σήμα Data End

Συνεπώς, για την εκτέλεση μιας υποδράσης με ένα πακέτο μεγέθους L απαιτείται ο χρόνος που δίνεται από τους ακόλουθους υπολογισμούς:

$$subaction_t(L) = arb_t + pkt_t(L) + ack_t \Rightarrow$$
$$subaction_t(L) \approx gap_count * 0.04069\mu s + L * 0.02035\mu s + 0.82035\mu s$$

Οι διαχωρισμένες ασύγχρονες συναλλαγές (split transactions) αποτελούνται συνήθως από δύο ανεξάρτητες υποδράσεις για την αίτηση και την απόκριση της συναλλαγής, με τα αντίστοιχα διαστήματα αδράνειας. Σε ένα δίαυλο με δύο μόνο κόμβους όταν πραγματοποιηθεί μια διαχωρισμένη συναλλαγή και οι δύο κόμβοι έχουν εκπέμψει από ένα πακέτο, οπότε ολοκληρώνεται ένας κύκλος αμεροληψίας, γεγονός που σημαίνει ότι το επόμενο διάστημα αδράνειας που θα ακολουθήσει θα είναι ένα διάστημα έναρξης περιόδου αμεροληψίας. Με βάση τα παραπάνω και την παράγραφο 2.3.2, ο ελάχιστος χρόνος που απαιτείται συνολικά για μια τέτοια συναλλαγή με μεγέθη αίτησης και απόκρισης L1 και L2 bytes, αντιστοίχως, θα είναι:

$$split_transaction_t =$$
$$arb_reset_gap + subaction_t(L1) + subaction_gap + subaction_t(L2) \Rightarrow$$
$$split_transaction_t \approx (L1 + L2) * 0.02035\mu s + gap_count * 0.56966\mu s + 2.45449\mu s$$

Από τον ελάχιστο χρόνο τέλεσης προκύπτει η θεωρητική μέγιστη διαμεταγωγή (throughput) δεδομένων με συναλλαγές του εκάστοτε τύπου, αν κάθε συναλλαγή φέρει φορτίο L bytes:

$$throughput \approx (1000000/transaction_t) * L$$

Χρησιμοποιώντας τους παραπάνω τύπους προκύπτουν οι πίνακες 37-42 και το γράφημα του σχήματος 39. Διαπιστώνεται εύκολα η βελτίωση της διαμεταγωγής που προκύπτει με την αύξηση της ποσότητας δεδομένων που μεταφέρεται ανά πακέτο, όπως επίσης και η καθοριστική σημασία της βελτιστοποίησης των διαστημάτων αδράνειας, ειδικά σε μικρά μεγέθη πακέτων. Σε ένα μη ιδανικό σύστημα, οι επιπλέον καθυστερήσεις που εισάγονται οδηγούν σε μικρότερες επιδόσεις, ωστόσο η επίδραση των δύο αυτών μεταβλητών παραμένει ιδιαίτερα σημαντική, αν και ίσως ελαφρώς μετριασμένη.

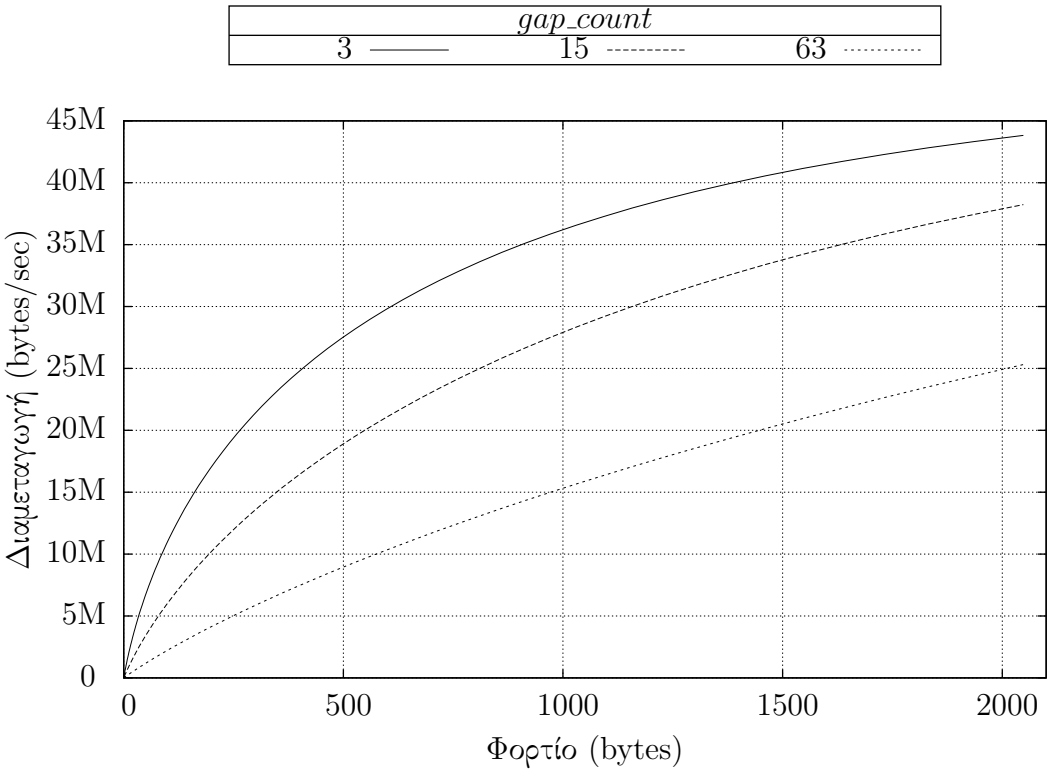
Πίνακας 37: Ελάχιστη καθυστέρηση (μs) για συναλλαγές ανάγνωσης ή εγγραφής τετράδας

gap_count	Καθυστέρηση (μs)
1	3.75675
3	4.89607
7	7.17471
15	11.73199
31	20.84655
63	39.07567

Πίνακας 38: Μέγιστη διαμεταγωγή (bytes/sec) με συναλλαγές ανάγνωσης ή εγγραφής τετράδας

gap_count	Διαμεταγωγή (bytes/sec)
1	1064750
3	816981
7	557513
15	340948
31	191878
63	102365

Σχήμα 39: Μέγιστη διαμεταγωγή με συναλλαγές ανάγνωσης τμήματος δεδομένων



Πίνακας 39: Ελάχιστη καθυστέρηση (μs) για συναλλαγές ανάγνωσης τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	4.00095	5.14027	7.41891	11.97619	21.09075	39.31987
2	4.00095	5.14027	7.41891	11.97619	21.09075	39.31987
4	4.00095	5.14027	7.41891	11.97619	21.09075	39.31987
8	4.08235	5.22167	7.50031	12.05759	21.17215	39.40127
16	4.24515	5.38447	7.66311	12.22039	21.33495	39.56407
32	4.57075	5.71007	7.98871	12.54599	21.66055	39.88967
64	5.22195	6.36127	8.63991	13.19719	22.31175	40.54087
128	6.52435	7.66367	9.94231	14.49959	23.61415	41.84327
256	9.12915	10.26847	12.54711	17.10439	26.21895	44.44807
512	14.33875	15.47807	17.75671	22.31399	31.42855	49.65767
1024	24.75795	25.89727	28.17591	32.73319	41.84775	60.07687
2048	45.59635	46.73567	49.01431	53.57159	62.68615	80.91527

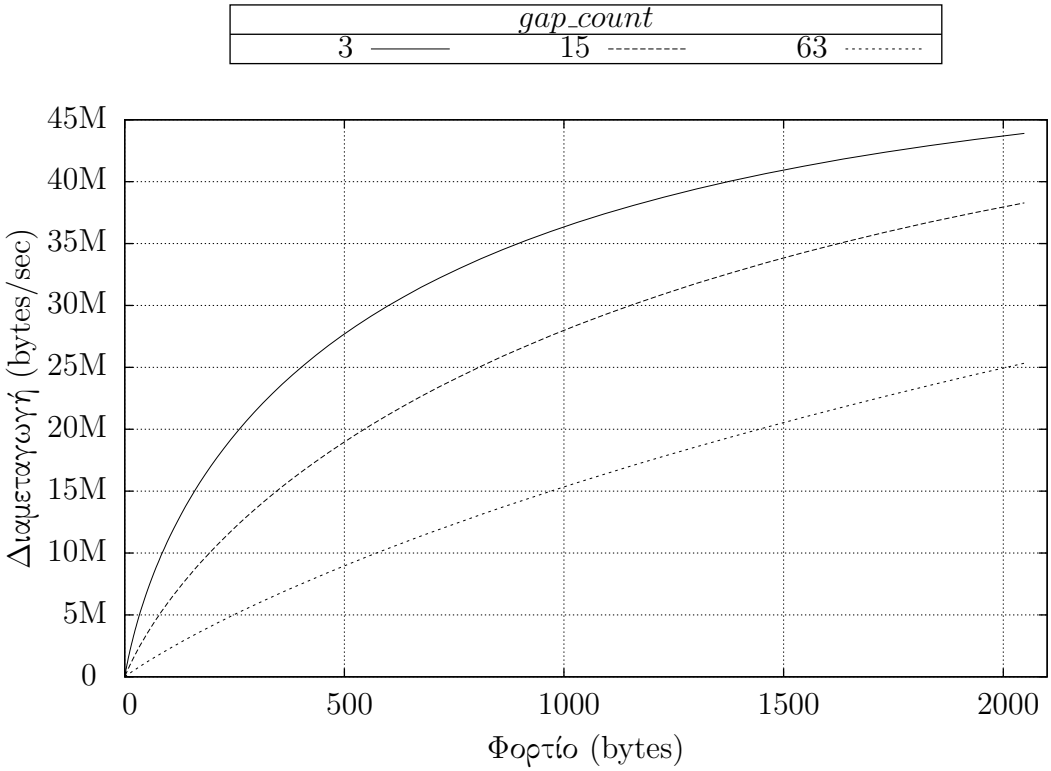
Πίνακας 40: Μέγιστη διαμεταγωγή (bytes/sec) με συναλλαγές ανάγνωσης τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	249940	194542	134790	83499	47414	25432
2	499881	389084	269581	166998	94828	50864
4	999762	778169	539162	333996	189656	101729
8	1959655	1532076	1066622	663482	377854	203039
16	3769006	2971508	2087925	1309287	749943	404407
32	7001039	5604134	4005652	2550615	1477340	802212
64	12255958	10060884	7407484	4849517	2868443	1578653
128	19618812	16702180	12874271	8827835	5420478	3059034
256	28042041	24930685	20403104	14966917	9763930	5759530
512	35707436	33079059	28834170	22945246	16290920	10310592
1024	41360451	39540847	36343103	31283232	24469654	17044829
2048	44915875	43820918	41783715	38229218	32670693	25310426

Πίνακας 41: Ελάχιστη καθυστέρηση (μs) για συναλλαγές εγγραφής τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	3.91955	5.05887	7.33751	11.89479	21.00935	39.23847
2	3.91955	5.05887	7.33751	11.89479	21.00935	39.23847
4	3.91955	5.05887	7.33751	11.89479	21.00935	39.23847
8	4.00095	5.14027	7.41891	11.97619	21.09075	39.31987
16	4.16375	5.30307	7.58171	12.13899	21.25355	39.48267
32	4.48935	5.62867	7.90731	12.46459	21.57915	39.80827
64	5.14055	6.27987	8.55851	13.11579	22.23035	40.45947
128	6.44295	7.58227	9.86091	14.41819	23.53275	41.76187
256	9.04775	10.18707	12.46571	17.02299	26.13755	44.36667
512	14.25735	15.39667	17.67531	22.23259	31.34715	49.57627
1024	24.67655	25.81587	28.09451	32.65179	41.76635	59.99547
2048	45.51495	46.65427	48.93291	53.49019	62.60475	80.83387

Σχήμα 40: Μέγιστη διαμεταγωγή με συναλλαγές εγγραφής τμήματος δεδομένων



Πίνακας 42: Μέγιστη διαμεταγωγή (bytes/sec) με συναλλαγές εγγραφής τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	255131	197672	136286	84070	47597	25485
2	510262	395345	272572	168140	95195	50970
4	1020525	790690	545144	336281	190391	101940
8	1999525	1556338	1078325	667992	379313	203459
16	3842689	3017120	2110341	1318066	752815	405241
32	7127980	5685179	4046888	2567272	1482912	803853
64	12450029	10191293	7477937	4879614	2878947	1581829
128	19866675	16881487	12980546	8877674	5439228	3064996
256	28294327	25129895	20536335	15038486	9794338	5770097
512	35911301	33253943	28966960	23029255	16333223	10327521
1024	41496886	39665523	36448402	31361220	24517344	17067955
2048	44996204	43897375	41853223	38287394	32713172	25335914

Στην περίπτωση που το πακέτο της απόκρισης μεταδοθεί ενωμένο με την επιβεβαίωση ACK της αίτησης (concatenated response) οι χρόνοι ολοκλήρωσης των ασύγχρονων συναλλαγών βελτιώνονται:

$$concat_transaction_t = arb_reset_gap + arb_t + pkt_t(L1) + RESPONSE_TIME + MIN_DATA_PREFIX + ack_pkt_t + MIN_PACKET_SEPARATION + pkt_t(L2) + ack_t \Rightarrow concat_transaction_t \approx (L1+L2)*0.02035\mu s + gap_count*0.36621\mu s + 1.99966\mu s$$

Για τις συνενωμένες συναλλαγές, συνεπώς, έχουμε τους πίνακες καθυστέρησης και διαμεταγωγής 43-48. Είναι εμφανής η βελτίωση που προκύπτει έναντι στις διαχωρισμένες συναλλαγές, ειδικά για μικρά μεγέθη πακέτων.

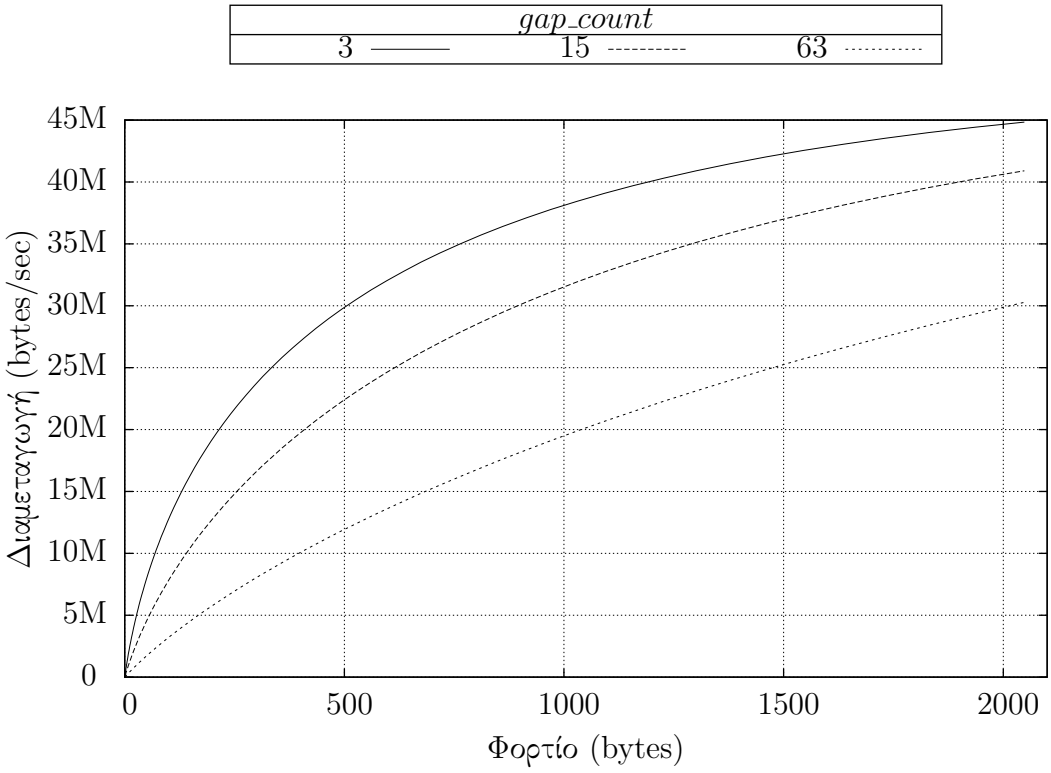
Πίνακας 43: Ελάχιστη καθυστέρηση (μs) για συνενωμένες συναλλαγές ανάγνωσης ή εγγραφής τετράδας

gap_count	Διαμεταγωγή (bytes/sec)
1	3.09847
3	3.83089
7	5.29573
15	8.22541
31	14.08477
63	25.80349

Πίνακας 44: Μέγιστη διαμεταγωγή (bytes/sec) με συνενωμένες συναλλαγές ανάγνωσης ή εγγραφής τετράδας

gap_count	Διαμεταγωγή (bytes/sec)
1	1290959
3	1044143
7	755325
15	486297
31	283994
63	155017

Σχήμα 41: Μέγιστη διαμεταγωγή με συνενωμένες συναλλαγές ανάγνωσης τμήματος δεδομένων



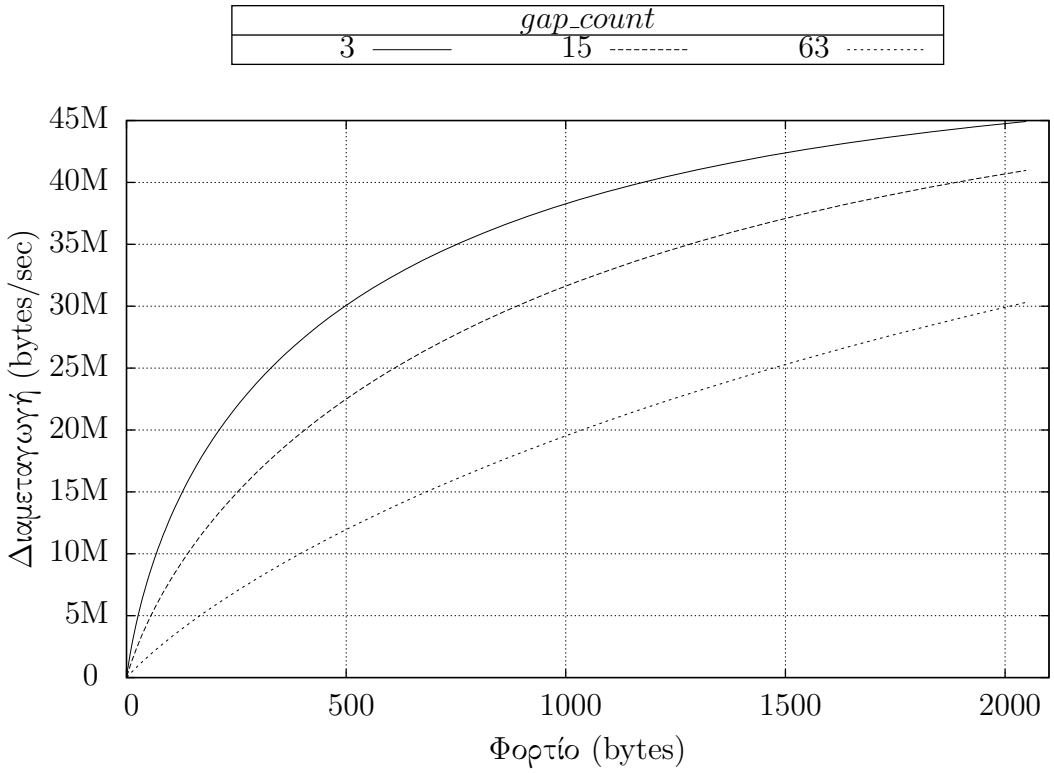
Πίνακας 45: Ελάχιστη καθυστέρηση (μs) για συνενωμένες συναλλαγές ανάγνωσης τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	3.34267	4.07509	5.53993	8.46961	14.32897	26.04769
2	3.34267	4.07509	5.53993	8.46961	14.32897	26.04769
4	3.34267	4.07509	5.53993	8.46961	14.32897	26.04769
8	3.42407	4.15649	5.62133	8.55101	14.41037	26.12909
16	3.58687	4.31929	5.78413	8.71381	14.57317	26.29189
32	3.91247	4.64489	6.10973	9.03941	14.89877	26.61749
64	4.56367	5.29609	6.76093	9.69061	15.54997	27.26869
128	5.86607	6.59849	8.06333	10.99301	16.85237	28.57109
256	8.47087	9.20329	10.66813	13.59781	19.45717	31.17589
512	13.68047	14.41289	15.87773	18.80741	24.66677	36.38549
1024	24.09967	24.83209	26.29693	29.22661	35.08597	46.80469
2048	44.93807	45.67049	47.13533	50.06501	55.92437	67.64309

Πίνακας 46: Μέγιστη διαμεταγωγή (bytes/sec) με συνενωμένες συναλλαγές ανάγνωσης τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	299162	245393	180507	118069	69788	38391
2	598324	490786	361015	236138	139577	76782
4	1196648	981573	722030	472276	279154	153564
8	2336400	1924700	1423150	935561	555155	306172
16	4460713	3704312	2766189	1836165	1097908	608552
32	8178976	6889291	5237547	3540054	2147828	1202217
64	14023801	12084386	9466153	6604331	4115763	2347014
128	21820401	19398377	15874334	11643762	7595370	4480053
256	30221216	27816139	23996707	18826561	13157103	8211473
512	37425614	35523756	32246423	27223312	20756669	14071543
1024	42490208	41236963	38939906	35036564	29185455	21878149
2048	45573830	44842960	43449361	40906812	36620886	30276558

Σχήμα 42: Μέγιστη διαμεταγωγή με συνενωμένες συναλλαγές εγγραφής τμήματος δεδομένων



Πίνακας 47: Ελάχιστη καθυστέρηση (μs) για συνενωμένες συναλλαγές εγγραφής τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	3.26127	3.99369	5.45853	8.38821	14.24757	25.96629
2	3.26127	3.99369	5.45853	8.38821	14.24757	25.96629
4	3.26127	3.99369	5.45853	8.38821	14.24757	25.96629
8	3.34267	4.07509	5.53993	8.46961	14.32897	26.04769
16	3.50547	4.23789	5.70273	8.63241	14.49177	26.21049
32	3.83107	4.56349	6.02833	8.95801	14.81737	26.53609
64	4.48227	5.21469	6.67953	9.60921	15.46857	27.18729
128	5.78467	6.51709	7.98193	10.91161	16.77097	28.48969
256	8.38947	9.12189	10.58673	13.51641	19.37577	31.09449
512	13.59907	14.33149	15.79633	18.72601	24.58537	36.30409
1024	24.01827	24.75069	26.21553	29.14521	35.00457	46.72329
2048	44.85667	45.58909	47.05393	49.98361	55.84297	67.56169

Πίνακας 48: Μέγιστη διαμεταγωγή (bytes/sec) με συνενωμένες συναλλαγές εγγραφής τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	306629	250394	183199	119214	70187	38511
2	613258	500789	366399	238429	140374	77022
4	1226516	1001579	732798	476859	280749	154045
8	2393296	1963146	1444061	944553	558309	307128
16	4564295	3775463	2805673	1853480	1104074	610442
32	8352757	7012177	5308269	3572221	2159627	1205904
64	14278479	12273021	9581512	6660276	4137421	2354041
128	22127450	19640667	16036221	11730624	7632235	4492853
256	30514442	28064359	24181215	18939940	13212378	8232969
512	37649633	35725524	32412592	27341649	20825393	14103094
1024	42634211	41372583	39060816	35134418	29253323	21916264
2048	45656532	44923028	43524526	40973431	36674267	30313036

Ειδικά για τις συναλλαγές εγγραφής έχουμε και την εκδοχή των ενοποιημένων συναλλαγών (unified transactions), όπου δεν υπάρχει πακέτο απόκρισης και ο κωδικός επιτυχίας της συναλλαγής επιστρέφεται με το πακέτο επιβεβαίωσης ACK. Πραγματοποιείται δηλαδή μόνο η υποδράση της αίτησης. Σε αυτή την περίπτωση παρατηρούμε ακόμα μικρότερους χρόνους εκτέλεσης της συναλλαγής και τις θεωρητικές βέλτιστες επιδόσεις των πινάκων 49 έως και 52.

$$unified_transaction_t = arb_reset_gap + subaction_t(L1) \Rightarrow$$
$$unified_transaction_t \approx L1 * 0.02035\mu s + gap_count * 0.36621\mu s + 1.34932\mu s$$

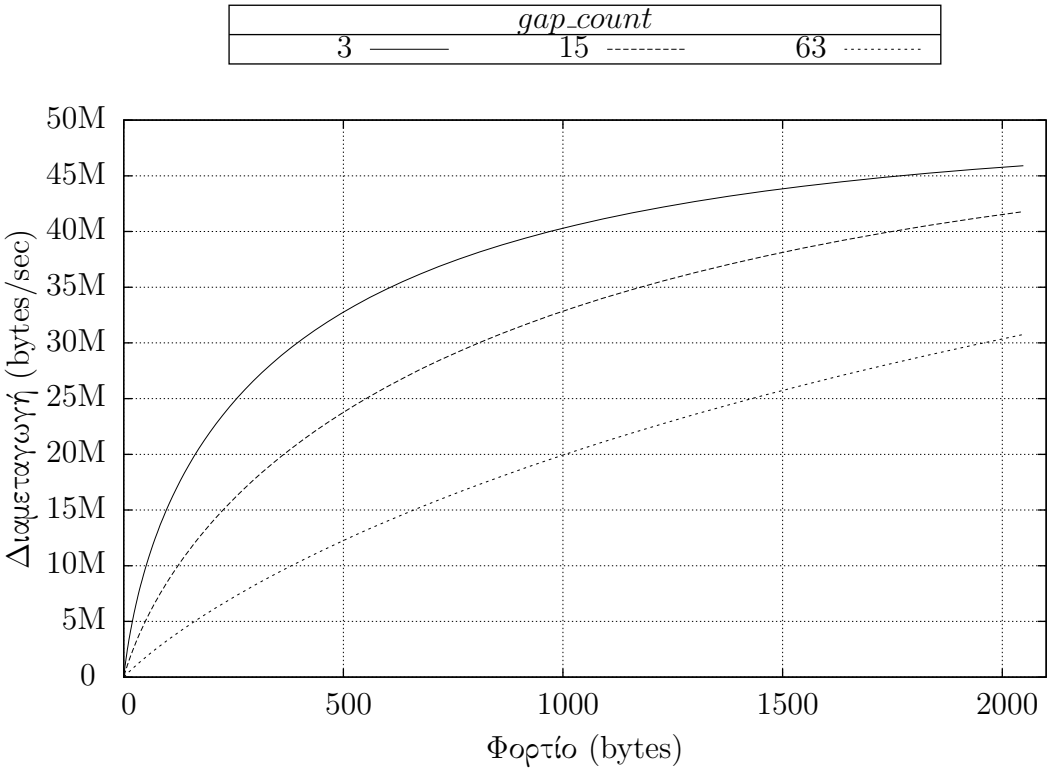
Πίνακας 49: Ελάχιστη καθυστέρηση (μs) για ενοποιημένες συναλλαγές εγγραφής τετράδας

gap_count	Διαμεταγωγή (bytes/sec)
1	2.12253
3	2.85495
7	4.31979
15	7.24947
31	13.10883
63	24.82755

Πίνακας 50: Μέγιστη διαμεταγωγή (bytes/sec) με ενοποιημένες συναλλαγές εγγραφής τετράδας

gap_count	Διαμεταγωγή (bytes/sec)
1	1884543
3	1401075
7	925970
15	551764
31	305137
63	161111

Σχήμα 43: Μέγιστη διαμεταγωγή με ενοποιημένες συναλλαγές εγγραφής τμήματος δεδομένων



Πίνακας 51: Ελάχιστη καθυστέρηση (μs) για ενοποιημένες συναλλαγές εγγραφής τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	2.28533	3.01775	4.48259	7.41227	13.27163	24.99035
2	2.28533	3.01775	4.48259	7.41227	13.27163	24.99035
4	2.28533	3.01775	4.48259	7.41227	13.27163	24.99035
8	2.36673	3.09915	4.56399	7.49367	13.35303	25.07175
16	2.52953	3.26195	4.72679	7.65647	13.51583	25.23455
32	2.85513	3.58755	5.05239	7.98207	13.84143	25.56015
64	3.50633	4.23875	5.70359	8.63327	14.49263	26.21135
128	4.80873	5.54115	7.00599	9.93567	15.79503	27.51375
256	7.41353	8.14595	9.61079	12.54047	18.39983	30.11855
512	12.62313	13.35555	14.82039	17.75007	23.60943	35.32815
1024	23.04233	23.77475	25.23959	28.16927	34.02863	45.74735
2048	43.88073	44.61315	46.07799	49.00767	54.86703	66.58575

Πίνακας 52: Μέγιστη διαμεταγωγή (bytes/sec) με ενοποιημένες συναλλαγές εγγραφής τμήματος δεδομένων

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	437573	331372	223085	134911	75348	40015
2	875147	662745	446170	269822	150697	80030
4	1750294	1325490	892341	539645	301394	160061
8	3380191	2581352	1752852	1067567	599114	319084
16	6325285	4905041	3384961	2089735	1183797	634051
32	11207895	8919736	6333636	4008985	2311899	1251948
64	18252702	15098790	11221002	7413181	4416037	2441690
128	26618254	23099898	18270080	12882875	8103814	4652219
256	34531458	31426659	26636728	20413907	13913172	8499745
512	40560463	38336122	34546999	28844956	21686249	14492692
1024	44439950	43070905	40571182	36351669	30092307	22383810
2048	46671967	45905747	44446383	41789377	37326605	30757331

Επειδή ο δίαυλος IEEE 1394 παρέχει τη δυνατότητα αμφίδρομης επικοινωνίας, μπορούμε να ορίσουμε την καθυστέρηση (latency) του συστήματος ως το χρόνο που απαιτείται για την αποστολή ενός πακέτου και τη λήψη μιας απόκρισης σε αυτό (round-trip delay). Στην περίπτωση των ασύγχρονων συναλλαγών, ο χρόνος ολοκλήρωσής τους μπορεί να δώσει ένα μέτρο της καθυστέρησης του διαύλου, καθώς αποτελούνται από δύο πακέτα δεδομένων και τις αντίστοιχες επιβεβαιώσεις ACK. Είναι προφανές ότι αναλόγως με τον τύπο της συναλλαγής ο χρόνος αυτός μεταβάλλεται, με ανάλογη επίδραση και στη συνολικότερη απόδοση του διαύλου.

Σε πραγματικές υλοποιήσεις του προτύπου IEEE 1394a-2000, λαμβάνοντας υπόψη το χρόνο διάδοσης των σημάτων στην καλωδίωση και τις καθυστερήσεις του φυσικού επιπέδου και του στρώματος διασύνδεσης και των δυο κόμβων, μπορούμε να αναμένουμε καθυστερήσεις ανά υποδράση ακόμα και 2 μs μεγαλύτερες από τις βέλτιστες που υπολογίσαμε παραπάνω, στις καλύτερες περιπτώσεις. Αυτό συνεπάγεται σημαντικά μειωμένη πραγματική απόδοση κατά την τέλεση συναλλαγών.

Στην περίπτωση που στο δίαυλο υπάρχει κάποιος ενεργός κόμβος παροχής χρονισμού ισόχρονων συναλλαγών (Cycle Master) τα παραπάνω θεωρητικά μέγιστα πρέπει να αναπροσαρμοστούν, λαμβάνοντας υπόψη το εύρος ζώνης που χάνεται λόγω των εκπεμπόμενων πακέτων Cycle Start. Τα πακέτα αυτά εκπέμπονται με ρυθμό 8 kHz και για το καθένα ισχύει:

$$cycle_start_t = subaction_gap + arb_t + pkt_t(20) \Rightarrow$$
$$cycle_start_t \approx gap_count * 0.20345\mu s + 1.07173\mu s$$

Πολλαπλασιάζοντας επί 8000 καταλήγουμε ότι για τη λειτουργία του χρονιστή κάθε δευτερόλεπτο απαιτούνται:

$$cycle_master_t \approx gap_count * 1627.6\mu s + 8573.84\mu s$$

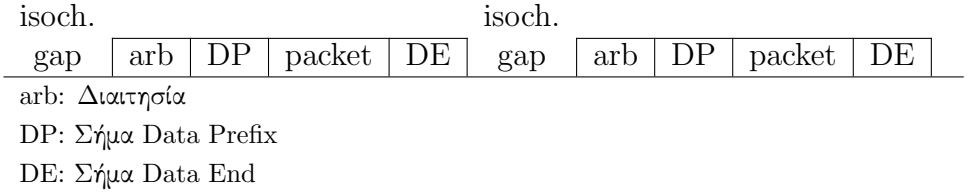
Με τη μεταβλητή *gap_count* να κινείται στο διάστημα 1-63, τα πακέτα Cycle Start καταλαμβάνουν από 1% έως και 11.1% του διαθέσιμου εύρους ζώνης, κάτι που μπορεί να επηρεάσει σημαντικά την εκτέλεση των ασύγχρονων συναλλαγών.

Διαπιστώνουμε, συνεπώς, ότι οι πραγματικές υλοποιήσεις απέχουν από τις επιδόσεις του ιδανικού μοντέλου. Για παράδειγμα αν στην τοπολογία δύο κόμβων που περιγράψαμε έχουμε επιπλέον καθυστέρηση ανά υποδράση 2 μs και ενεργό κόμβο Cycle Master, τότε για την ελάχιστη τιμή της μεταβλητής gap_count έχουμε μέγιστη διαμεταγωγή περίπου 43.6 MB/sec κατά την τέλεση ενοποιημένων συναλλαγών εγγραφής, έναντι των αρχικών 46.7 MB/sec. Ομοίως, για διαχωρισμένες συναλλαγής ανάγνωσης, οι οποίες αποτελούνται από δύο υποδράσεις, η μέγιστη διαμεταγωγή πέφτει από τα 44.9 MB/sec στα 40.5 MB/sec. Αν λάβουμε υπόψη ότι σε πολλές περιπτώσεις οι καθυστερήσεις που εισάγονται σε μια υποδράση μπορούν να είναι ακόμα μεγαλύτερες και πως συχνά θα έχουμε χαμένο εύρος ζώνης λόγω επαναλήψεων πακέτων και συγκρούσεων κατά τη διαιτησία, αντιλαμβανόμαστε ότι η μετρούμενη διαμεταγωγή σε ένα πραγματικό σύστημα θα είναι ακόμα μικρότερη.

2.7.2 Ισόχρονες συναλλαγές

Για τις ισόχρονες συναλλαγές σε ένα κανάλι ο υπολογισμός της μέγιστης διαμεταγωγής είναι πολύ απλούστερος. Όπως έχει προαναφερθεί στην παράγραφο 2.4.3, σε κάθε ισόχρονο κύκλο επιτρέπεται η αποστολή ενός μόνο πακέτου για κάθε κανάλι. Με 8000 κύκλους ανά δευτερόλεπτο και μέγιστο φορτίο ισόχρονου πακέτου σε ρυθμό μετάδοσης δεδομένων S400 τα 4096 bytes, λαμβάνουμε μέγιστη διαμεταγωγή για ένα κανάλι 32768000 bytes/sec.

Σχήμα 44: Τέλεση ισόχρονων συναλλαγών στο χρόνο



Για τη μέγιστη συνολική διαμεταγωγή δεδομένων μέσω ισόχρονων πακέτων με φορτίο L bytes σε όλα τα κανάλια έχουμε:

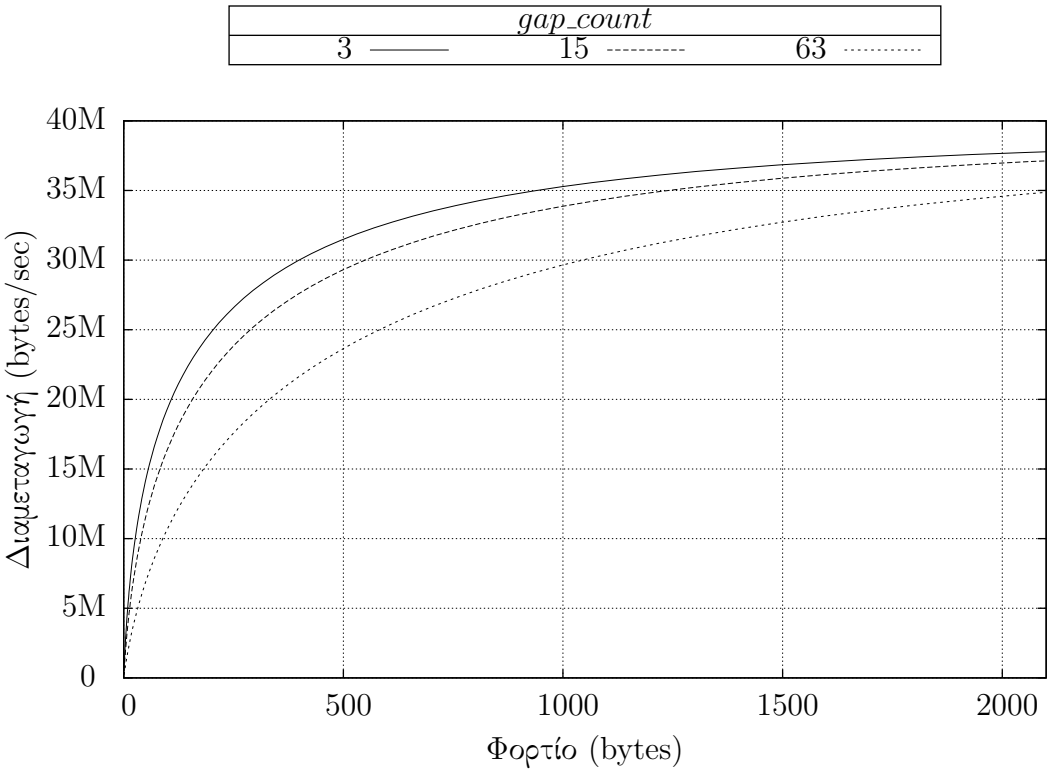
$$isoc_subaction_t = isochronous_gap + arb_t + pkt_t(L + 12) \Rightarrow$$
$$isoc_subaction_t \approx L * 0.02035\mu s + gap_count * 0.04069\mu s + 0.66414\mu s$$

Η τελευταία σχέση, λαμβάνοντας υπόψη ότι μόνο το 80% του συνολικού εύρους ζώνης του διαύλου είναι διαθέσιμο για ισόχρονες συναλλαγές, δίνει τελικά:

$$throughput \approx 0.8 * (1000000/isoc_transaction_t) * L$$

Με βάση τα παραπάνω προκύπτουν οι πίνακες 53 και 54. Παρατηρούμε ότι οι ισόχρονες συναλλαγές κάνουν καλύτερη χρήση του διαθέσιμου σε αυτές εύρους ζώνης από τις ασύγχρονες, πράγμα αναμενόμενο αφού εμπεριέχουν αναλογικά μικρότερα διαστήματα αδράνειας και λιγότερα δεδομένα εκτός του φορτίου των πακέτων. Εδώ πρέπει να σημειωθεί ότι ο περιορισμός στο 80% του συνολικού εύρους ζώνης του διαύλου δεν επιβάλλεται με κάποιον τρόπο, αλλά η τήρησή του επαφίεται στη σωστή υλοποίηση του προτύπου IEEE 1394 από κάθε κόμβο. Είναι λοιπόν δυνατόν σε ορισμένες ειδικές εφαρμογές να παρακαμφθεί εν μέρει το πρότυπο, προκειμένου να επιτευχθούν υψηλότερες επιδόσεις, όπως άλλωστε συμβαίνει σε εμπορικές εφαρμογές και με άλλους τυπικούς περιορισμούς που επιβάλλονται από τα διάφορα πρωτόκολλα.

Σχήμα 45: Μέγιστη συνολική διαμεταγωγή με ισόχρονες συναλλαγές



Πίνακας 53: Ελάχιστη καθυστέρηση (μs) για ισόχρονες συναλλαγές

Φορτίο (bytes)	gap_count 1	3	7	15	31	63
1	1.03043	1.11181	1.27457	1.60009	2.25113	3.55321
2	1.03043	1.11181	1.27457	1.60009	2.25113	3.55321
4	1.03043	1.11181	1.27457	1.60009	2.25113	3.55321
8	1.11183	1.19321	1.35597	1.68149	2.33253	3.63461
16	1.27463	1.35601	1.51877	1.84429	2.49533	3.79741
32	1.60023	1.68161	1.84437	2.16989	2.82093	4.12301
64	2.25143	2.33281	2.49557	2.82109	3.47213	4.77421
128	3.55383	3.63521	3.79797	4.12349	4.77453	6.07661
256	6.15863	6.24001	6.40277	6.72829	7.37933	8.68141
512	11.36823	11.44961	11.61237	11.93789	12.58893	13.89101
1024	21.78743	21.86881	22.03157	22.35709	23.00813	24.31021
2048	42.62583	42.70721	42.86997	43.19549	43.84653	45.14861
4096	84.30263	84.38401	84.54677	84.87229	85.52333	86.82541

Πίνακας 54: Μέγιστη διαμεταγωγή (bytes/sec) με ισόχρονες συναλλαγές

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	776374	719547	627662	499971	355377	225148
2	1552749	1439094	1255325	999943	710754	450297
4	3105499	2878189	2510650	1999887	1421508	900594
8	5756275	5363682	4719868	3806148	2743801	1760849
16	10042129	9439458	8427872	6940340	5129582	3370718
32	15997700	15223506	13880078	11797833	9075021	6209056
64	22741102	21947779	20516354	18149013	14745991	10724287
128	28813983	28168936	26961771	24833332	21447137	16851501
256	33254149	32820460	31986155	30438640	27753197	23590637
512	36030235	35774144	35272730	34310920	32536522	29486696
1024	37599661	37459742	37183006	36641620	35604805	33697775
2048	38436788	38363545	38217894	37929885	37366696	36289046
4096	38869487	38832001	38757246	38608596	38314691	37740103

3 Ο δίαυλος IEEE 1394 σε υπολογιστικά περιβάλλοντα

3.1 Γενικά

Η συντριπτική πλειοψηφία των υπολογιστών που έχει υποστήριξη για το πρότυπο IEEE 1394 χρησιμοποιεί ελεγκτές σε δίαυλο συστήματος είτε PCI (Peripheral Component Interconnect), είτε PCIe (PCI Express). Αυτό επιτρέπει στον ελεγκτή να επικοινωνεί με τον επεξεργαστή και τη φυσική μνήμη του συστήματος σε ταχύτητες που γενικά υπερβαίνουν το ρυθμό μεταφοράς δεδομένων της διασύνδεσης IEEE 1394, έτσι ώστε να επιτυγχάνονται οι μέγιστες δυνατές επιδόσεις σε αυτήν.

Αρχικά, μια σειρά από κατασκευαστές παρουσίασαν στην αγορά ελεγκτές που ακολουθούσαν το πρότυπο IEEE 1394, καθένας χρησιμοποιώντας τη δική του μέθοδο επικοινωνίας με το λογισμικό του υπολογιστή. Οι ελεγκτές PCI-Lynx της Texas Instruments αποτελούν χαρακτηριστικό παράδειγμα αυτής της αντίληψης, καθώς είχαν ευρεία διάδοση για αρκετά χρόνια.

Γρήγορα διαπιστώθηκε η πληθώρα δυσκολιών που προκύπτουν από την ύπαρξη πολλαπλών ασύμβατων υλοποιήσεων και έτσι τον Οκτώβριο του 1997 παρουσιάστηκε η πρώτη έκδοση του ανοικτού προτύπου 1394 Open Host Controller Interface (Open HCI, OHCI) 1.0, με την οποία επετράπη η χρήση κοινού λογισμικού ελέγχου για όσους ελεγκτές την υποστηρίζουν. Τον Ιανουάριο του 2000 δημοσιεύτηκε η βελτιωμένη έκδοση 1394 Open HCI 1.1, την οποία ακολουθεί και η συντριπτική πλειοψηφία ελεγκτών για υπολογιστικά συστήματα σήμερα.

Τα περισσότερα λειτουργικά συστήματα διαθέτουν υποστήριξη για το χειρισμό διαύλων IEEE 1394, καθώς και για κάποιες κατηγορίες συσκευών. Οι συνηθέστεροι τύποι συσκευών IEEE 1394 που χρησιμοποιούνται από έναν υπολογιστή είναι οι καταναλωτικές συσκευές κινούμενης εικόνας τεχνολογίας DV, καθώς και συστήματα αποθήκευσης δεδομένων που ακολουθούν το πρωτόκολλο Serial Bus Protocol (SBP). Ειδικά οι κόμβοι SBP, που συνήθως αποτελούνται από έναν ή περισσότερους εξωτερικούς σκληρούς δίσκους, έχουν ιδιαίτερο ενδιαφέρον σε κάποιες εφαρμογές, καθώς επιτρέπουν ταυτόχρονη πρόσβαση (multiple login) από περισσότερους του ενός υπολογιστές. Αυτό μπορεί να δώσει λύσεις σε αρκετά τεχνικά ζητήματα, τόσο σε εφαρμογές βάσεων δεδομένων (π.χ. Oracle), όσο και σε άλλες περιπτώσεις με τη χρήση διαμοιρασμένων συστημάτων αρχείων (shared filesystems), όπως για παράδειγμα το Global File System (GFS) που χρησιμοποιείται σε κατανεμημένα υπολογιστικά συστήματα Linux ή SGI IRIX.

Άλλες συσκευές με ευρεία υποστήριξη είναι οι μονάδες επεξεργασίας ήχου υψηλής πιστότητας που λειτουργούν με βάση το πρότυπο IEC 61883, καθώς και επιστημονικές και βιομηχανικές συσκευές σύλληψης κινούμενης εικόνας που χρησιμοποιούν το πρωτόκολλο Instrumentation & Industrial Digital Camera (IIDC). Αξιοσημείωτη είναι η χρήση υπολογιστών με κάμερες υψηλής ταχύτητας IEEE 1394b σε διαστημικά οχήματα της NASA και άλλων οργανισμών για την επιτήρηση της διαδικασίας εκτόξευσής τους.

3.2 Το πρότυπο 1394 Open Host Controller Interface (OHCI)

Το πρότυπο 1394 Open Host Controller Interface (Open HCI, OHCI) είναι μια ανοικτή προδιαγραφή για την υλοποίηση ελεγκτών IEEE 1394 για υπολογιστικά συστήματα γενικής χρήσης. Με βάση το Open HCI, το υλικό του ελεγκτή παρέχει το φυσικό επίπεδο και το υπόστρωμα διασύνδεσης του διαύλου, όπως επίσης και μια σειρά από λειτουργίες υποστήριξης για τα επίπεδα συναλλαγής και διαχείρισης διαύλου, τα οποία όμως κατά τα άλλα υλοποιούνται σε λογισμικό στον επεξεργαστή του συστήματος.

Οι ελεγκτές Open HCI έκδοσης 1.1 ακολουθούν επακριβώς το IEEE Std 1394a-2000, ενώ υπάρχουν προβλέψεις στην προδιαγραφή και για τις μεταγενέστερες εκδόσεις του προτύπου. Υποστηρίζονται πλήρως ασύγχρονες και ισόχρονες συναλλαγές, ενώ στο υλικό πρέπει να περιέχεται πάντα ένα ρολόι 8 kHz για την τέλεση της λειτουργίας παροχής κύκλων χρονισμού (Cycle Master, CM).

Η μεταφορά δεδομένων για τις ασύγχρονες συναλλαγές που εκκινεί ο τοπικός κόμβος πραγματοποιείται με χρήση μηχανισμών απευθείας πρόσβασης στη μνήμη (DMA engines) του υπολογιστή, έτσι ώστε να μεγιστοποιηθεί η απόδοση και να ελαχιστοποιηθούν οι απαιτήσεις σε επεξεργαστική ισχύ. Ο μηχανισμός DMA του ελεγκτή είναι δυνατόν να προγραμματιστεί έτσι ώστε να δρα ως γέφυρα ανάμεσα στο δίαυλο IEEE 1394 και το δίαυλο συστήματος του υπολογιστή, κάτι που επιτρέπει την άμεση εκτέλεση των εισερχόμενων ασύγχρονων συναλλαγών εγγραφής και ανάγνωσης απευθείας στο χώρο φυσικών διευθύνσεων του συστήματος. Αυτό, βεβαίως, απαιτεί ο ελεγκτής να μπορεί να λάβει τον έλεγχο του διαύλου συστήματος - για παράδειγμα να έχει δυνατότητες bus master σε δίαυλο PCI - έτσι ώστε να είναι σε θέση να πραγματοποιήσει διεργασίες ανάγνωσης και εγγραφής αυτόνομα από τον κεντρικό επεξεργαστή.

Για την πραγματοποίηση ισόχρονων συναλλαγών, πέρα από τον χρονιστή 8 kHz, κάθε ελεγκτής Open HCI διαθέτει από έναν ανεξάρτητο μηχανισμό DMA για την εκπομπή και λήψη δεδομένων. Κάθε μηχανή DMA μπορεί να υποστηρίζει ταυτόχρονα από 4 έως και 32 ισόχρονα κανάλια (isochronous contexts), αν και οι περισσότερες υλοποιήσεις υποστηρίζουν 4 ή 8 κανάλια, τα οποία είναι επαρκή για τις περισσότερες εφαρμογές. Ο ελεγκτής μπορεί να εκπέμψει ή να λάβει ένα πακέτο από κάθε κανάλι σε κάθε ισόχρονο κύκλο.

Για να μειωθούν οι πιθανότητας απώλειας πακέτων όταν ο κεντρικός επεξεργαστής είναι απασχολημένος, αλλά και για τη διεκόλυνση της διαχείρησης του ελεγκτή από το υποσύστημα IEEE 1394 του λειτουργικού συστήματος, κάθε ελεγκτής διαθέτει πολλαπλές ουρές (FIFO, First In First Out) και μηχανές DMA για την αποστολή και τη λήψη δεδομένων. Ένας μηχανισμός διακοπών (interrupts) ενημερώνει το σύστημα για κάθε γεγονός (event) στις ουρές αυτές, που είναι συνοπτικά οι ακόλουθες:

- Εξερχόμενες ασύγχρονες αιτήσεις (Asynchronous Transmit Request): Χρησιμοποιείται όταν ο τοπικός κόμβος εκκινεί την τέλεση μιας ασύγχρονης συναλλαγής, μεταφέροντας μέσω DMA και μιας ουράς την αίτηση από τη μνήμη του συστήματος στο φυσικό επίπεδο, έτσι ώστε να μεταδοθεί.

- Εξερχόμενες ασύγχρονες αποκρίσεις (Asynchronous Transmit Response): Όμοια με τον προηγούμενο μηχανισμό, χρησιμοποιείται για τη μετάδοση αποκρίσεων σε ασύγχρονες συναλλαγές που παράγονται από τον κεντρικό επεξεργαστή, όπως λόγου χάρη οι συναλλαγές που αναφέρονται σε κάποιον καταχωρητή.

- Εξερχόμενες ασύγχρονες αποκρίσεις φυσικών διευθύνσεων (Asynchronous Transmit Physical Response): Αυτή η μονάδα αναλαμβάνει την απευθείας ανάκτηση μέσω DMA δεδομένων από το εύρος φυσικών διευθύνσεων του υπολογιστή και τη μετάδοσή τους στο δίαυλο IEEE 1394.

- Εισερχόμενες ασύγχρονες αιτήσεις (Asynchronous Receive Request): Εισερχόμενες ασύγχρονες αιτήσεις που μεταφέρονται μέσω DMA στον κεντρικό επεξεργαστή για χειρισμό.

- Εισερχόμενες ασύγχρονες αποκρίσεις (Asynchronous Receive Response): Εισερχόμενες αποκρίσεις σε ασύγχρονες αιτήσεις που μεταφέρονται μέσω DMA για ανάλυση από τον κεντρικό επεξεργαστή.

- Εισερχόμενες ασύγχρονες αιτήσεις ανάγνωσης φυσικών διευθύνσεων (Physical Read Request): Αυτή η μονάδα προσδιορίζει πότε μια εισερχόμενη αίτηση ανάγνωσης αναφέρεται στο εύρος φυσικών διευθύνσεων του υπολογιστή και προωθεί την αίτηση στην μονάδα AT Physical Response.

- Εισερχόμενες ασύγχρονες αιτήσεις εγγραφής φυσικών διευθύνσεων (Physical Write Request): Το τμήμα αυτό αποφασίζει πότε μια εισερχόμενη αίτηση εγγραφής αναφέρεται στο εύρος φυσικών διευθύνσεων του υπολογιστή και μεταφέρει απευθείας τα συνημμένα δεδομένα στην καθορισμένη διεύθυνση μέσω DMA.

- Εξερχόμενα ισόχρονα πακέτα (Isochronous transmit): Χρησιμοποιείται για τη μεταφορά ισόχρονων πακέτων από τη μνήμη του συστήματος στο φυσικό επίπεδο για μετάδοση, μέσω DMA και μιας ουράς FIFO.

- Εισερχόμενα ισόχρονα πακέτα (Isochronous receive): Τα εισερχόμενα ισόχρονα πακέτα αποθηκεύονται σε μια ουρά και μεταφέρονται μέσω DMA στη μνήμη του συστήματος όταν ζητηθεί από τον κεντρικό επεξεργαστή.

- Εισερχόμενα πακέτα Self-ID (Self-ID receive): Αποθηκεύει μέσω DMA τα πακέτα Self-ID που λαμβάνει ο κόμβος κατά την αυτοδιαμόρφωση του διαύλου σε έναν ενταμιευτή μεγέθους 2 KiB της κεντρικής μνήμης.

Για την ταξινόμηση των εισερχόμενων αιτήσεων σε κατηγορίες, το εύρος διευθύνσεων του κόμβου διαχωρίζεται σε τέσσερα τμήματα:

- Φυσικές διευθύνσεις (Low Address Space): Είναι οι διευθύνσεις που βρίσκονται κάτω από ένα άνω όριο (physicalUpperBound) το οποίο προαιρετικά ρυθμίζεται μέσω ενός καταχωρητή του ελεγκτή. Στην περίπτωση που ο καταχωρητής δεν έχει υλοποιηθεί, τότε το όριο είναι η διεύθυνση

000100000000_{16} , που δίνει ένα εύρος φυσικών διευθύνσεων 4 GiB, καλύπτοντας τους περισσότερους υπολογιστές με λέξη 32 bit πλήρως. Ο ελεγκτής μπορεί προαιρετικά να ρυθμιστεί έτσι ώστε να αποκρίνεται άμεσα σε συναλλαγές εγγραφής με ένα πακέτο επιβεβαίωσης με κωδικό επιτυχίας ack_complete (posted writes). Αυτό οδηγεί σε σημαντική επιτάχυνση των συναλλαγών αυτών, όπως περιγράφεται παρακάτω.

- Μέσος χώρος διευθύνσεων (Middle Address Space): Είναι οι διευθύνσεις από το άνω άκρο του προηγούμενου υποχώρου (physicalUpperBound) έως και τη διεύθυνση $fffeffffffff_{16}$. Ο ελεγκτής αποκρίνεται αυτόματα σε συναλλαγές εγγραφής με ένα πακέτο επιβεβαίωσης ACK που έχει κωδικό ack_complete (unified transactions) και στη συνέχεια προωθεί την αίτηση στον κεντρικό επεξεργαστή. Αυτό αυξάνει την απόδοση του διαύλου IEEE 1394, αλλά είναι δυνατόν μια συναλλαγή να αποτύχει σιωπηλά. Για το λόγο αυτό, ο μέσος χώρος συνίσταται να χρησιμοποιείται σε συνδυασμό με εφαρμογές που έχουν δικές τους μεθόδους ανίχνευσης χαμένων και αλλοιωμένων πακέτων, όπως είναι το πρωτόκολλο TCP/IP. Η συμπεριφορά αυτή σκοπίμως παραβιάζει το πρότυπο IEEE 1394, με βάση το οποίο οι ασύγχρονες συναλλαγές πρέπει να είναι πλήρως αξιόπιστες

- Ανώτερος χώρος διευθύνσεων (Upper Address Space): Περιλαμβάνει τις διευθύνσεις από $ffff00000000_{16}$ έως και $ffffeffffffff_{16}$ και έχει, συνεπώς, μέγεθος 3840 MiB. Σε αυτόν το χώρο οι εγγραφές πραγματοποιούνται με διαχωρισμένες συναλλαγές (split transactions), εξασφαλίζοντας έτσι την αξιοπιστία της μεταφοράς δεδομένων σε περίπτωση λαθών. Η περιοχή αυτή εμπεριέχει το άνω τμήμα του χώρου μνήμης (memory space) και ολόκληρο τον ιδιωτικό υποχώρο διευθύνσεων (private space), όπως αυτοί ορίζονται από την αρχιτεκτονική CSR.

- Χώρος CSR: Είναι η περιοχή που δεσμεύεται από την αρχιτεκτονική CSR για τους καταχωρητές. Οι περισσότερες συναλλαγές σε αυτήν την περιοχή προωθούνται στον κεντρικό επεξεργαστή, αλλά κάποιες πραγματοποιούνται στο υλικό του ελεγκτή για την επιτάχυνση συγκεκριμένων καταχωρητών.

Συνολικά υπάρχουν οι εξής κατηγορίες λειτουργιών DMA, με τις οποίες επιταχύνεται δραστικά η λειτουργία των επιπέδων συναλλαγής και διαχείρισης τα οποία υλοποιούνται με λογισμικό στον κεντρικό επεξεργαστή:

- DMA γενικής χρήσης για τα πακέτα των συναλλαγών που διαχειρίζεται ο επεξεργαστής

- Πρόσβαση μέσω DMA απευθείας από το δίαυλο IEEE 1394 στις φυσικές διευθύνσεις του κόμβου (Physical DMA)

- DMA στον ενταμιευτή των πακέτων Self-ID

- DMA σε έναν ενταμιευτή μεγέθους 1 KiB, στον οποίο αποθηκεύεται η μνήμη ρυθμίσεων (Configuration ROM) του κόμβου

Προκειμένου να αποφευχθούν προβλήματα με συσκευές που δυσλειτουργούν και για την προστασία ευαίσθητων δεδομένων, όλοι οι ελεγκτές που ακολουθούν το πρότυπο Open HCI διαθέτουν μια σειρά από προγραμματιζόμενα φίλτρα. Τα φίλτρα αυτά επιτρέπουν στο λειτουργικό σύστημα του υπολογιστή να ελέγξει από ποιους κόμβους του διαύλου γίνονται δεκτές αιτήσεις πρόσβασης στη φυσική μνήμη και από ποιους οι λοιπές αιτήσεις για ασύγχρονες συναλλαγές.

Για τις ασύγχρονες συναλλαγές πρέπει να υποστηρίζεται κάποιο από τα δυο πρωτόκολλα επανάληψης που προβλέπει το πρότυπο IEEE 1394. Για τις εισερχόμενες αιτήσεις, ωστόσο, συνίσταται από την προδιαγραφή Open HCI να υλοποιείται το πρωτόκολλο επανάληψης δύο φάσεων. Ο μηχανισμός επανάληψης λειτουργεί απολύτως αυτόνομα, με τον κεντρικό επεξεργαστή να μην ενημερώνεται για συναλλαγές που έχουν αναβληθεί. Για βελτιστοποίηση της χρήσης του διαύλου, επιτρέπεται η εκτέλεση συναλλαγών εκτός σειράς (out of order), έτσι ώστε να συνεχίζει η επεξεργασία της εκάστοτε ουράς, ακόμα και όταν κάποια συναλλαγή της πρέπει να επαναληφθεί.

Το πρότυπο Open HCI προβλέπει την υλοποίηση των βασικών καταχωρητών CSR του διαύλου IEEE 1394 στο υλικό του ελεγκτή, για την επιτάχυνση της πρόσβασης σε αυτούς. Τόσο οι καταχωρητές αυτοί, όσο και μια σειρά άλλοι καταχωρητές ρυθμίσεων, είναι προσβάσιμοι από το δίαυλο συστήματος του υπολογιστή με τη χρήση κατάλληλων διευθύνσεων I/O. Με τη χρήση τους επιτυγχάνεται τόσο η λεπτομερής καθοδήγηση της λειτουργίας των μηχανισμών μεταφοράς δεδομένων, όσο και η μεταβολή πιο στατικών ρυθμίσεων όπως είναι οι παράμετροι του φυσικού επιπέδου του κόμβου.

3.3 Ο δίαυλος IEEE 1394 σε προσωπικούς υπολογιστές αρχιτεκτονικής PC

Η τυπική υλοποίηση ενός σημερινού προσωπικού υπολογιστή αρχιτεκτονικής PC (Personal Computer) με υποστήριξη για το πρότυπο IEEE 1394 περιλαμβάνει έναν ελεγκτή Open HCI συνδεδεμένο στο δίαυλο συστήματος, ο οποίος μπορεί να είναι είτε PCI (Peripheral Component Interconnect), είτε PCIe (PCI Express). Το λειτουργικό σύστημα που εκτελείται στον κεντρικό επεξεργαστή πρέπει να περιλαμβάνει τα επίπεδα συναλλαγής και διαχείρησης του διαύλου, καθώς και να παρέχει όλες τις απαραίτητες υπηρεσίες στο επίπεδο εφαρμογής του κόμβου.

Ο δίαυλος PCI προτυποποιήθηκε στα μέσα του 1992 με την προδιαγραφή PCI Local Bus Specification Revision 1.0, η οποία βελτιώθηκε και επεκτάθηκε με πολλαπλές μεταγενέστερες εκδόσεις, φθάνοντας στην έκδοση 3.0 το 2004. Είναι ένας παράλληλος δίαυλος με τοπολογία αρτηρίας, πολυπλεξία δεδομένων και διευθύνσεων και εύρος 32 ή 64 bit. Ένας δίαυλος PCI μπορεί να συνδεθεί με τον επεξεργαστή μέσω μιας γέφυρας (host bus bridge), αλλά και με έναν άλλο δίαυλο PCI μέσω μιας γέφυρας PCI-PCI (PCI-to-PCI bridge).

Αντίστοιχα, το αρχικό πρότυπο PCI Express Base Specification Revision 1.0 για το δίαυλο PCIe δημοσιεύτηκε το 2002 και ήδη στα τέλη του 2007 έφτασε την έκδοση 2.0. Στη φυσική του υλοποίηση ο δίαυλος PCIe είναι ριζικά διαφορετικός, καθώς αποτελείται από συνδέσεις σημείο-προς-σημείο που συνδέουν κάθε συσκευή με μια μονάδα μεταγωγής (switch). Κάθε σύνδεση μπορεί να αποτελείται από μία ή περισσότερες γραμμές, στις οποίες η μετάδοση των δεδομένων γίνεται σειριακά, με τη μονάδα μεταγωγής να αναμεταδίδει τα δεδομένα ανάλογα με τον προορισμό τους. Το μοντέλο διευθυνσιοδότησης του PCIe κληρονομείται από τον PCI, ενώ η λογική δομή της μονάδας μεταγωγής του PCIe είναι ένα σύνθετο δίκτυο από γέφυρες PCI-PCI, επιτρέποντας έτσι την απρόσκοπτη χρήση της από παλαιότερα λειτουργικά συστήματα.

Κατά συνέπεια, οι δίαυλοι συστήματος PCI και PCIe είναι παραπλήσιοι από τη σκοπιά του λογισμικού, αν και η φυσική τους διαμόρφωση δεν έχει καμία απολύτως ομοιότητα. Έτσι για λόγους απλότητας, όπου δεν υφίσταται λόγος διαφοροποίησης, θα αναφερόμαστε και στους δυο διαύλους με το γενικό ακρωνύμιο PCI.

Αρκετά διαφορετικός είναι ο δίαυλος AGP (Accelerated Graphics Port), ο οποίος είναι ουσιαστικά μια σημείο-προς-σημείο διασύνδεση ανάμεσα στο δίαυλο του επεξεργαστή και την κάρτα γραφικών. Παρέχει τη δυνατότητα μεταφοράς δεδομένων με ρυθμούς μεταφοράς δεδομένων πολλαπλάσιους του PCI, με την τελευταία έκδοσή του να υπερβαίνει τα 2 GiB/sec. Επιπλέον διαθέτει ανεξάρτητες γραμμές διευθυνσιοδότησης, καθώς και μια εξελιγμένη μονάδα πρόσβασης στην κύρια μνήμη του υπολογιστή (Graphics Address Remapping Table, GART). Προγραμματιστικά ο δίαυλος AGP αποτελεί υπερσύνολο του PCI και υποστηρίζει όλα τα χαρακτηριστικά του τελευταίου.

Για τους σκοπούς αυτής της εργασίας, ιδιαίτερο ενδιαφέρον έχει μόνο το μοντέλο διευθυνσιοδότησης των προαναφερθέντων διαύλων, καθώς και η αλληλεπίδραση μιας συσκευής συνδεδεμένης σε αυτούς με τα υπόλοιπα τμήματα ενός υπολογιστή. Οι δίαυλοι αυτοί χρησιμοποιούν κατά κόρον ένα επίπεδο (flat) μοντέλο διευθυνσιοδότησης, στο οποίο οι συσκευές προβάλλουν τα απαραίτητα τμήματα της μνήμης

τους ως μέρος του χώρου διευθύνσεων του υπολογιστή (I/O memory mapping). Το πλάτος των διευθύνσεων μπορεί να είναι 32 ή 64 bit, ανάλογα με το μέγεθος λέξης του κεντρικού επεξεργαστή. Ταυτόχρονα υπάρχει και ένα ανεξάρτητο εύρος διευθύνσεων εισόδου/εξόδου (I/O address range), του οποίου η χρήση, όμως, αποθαρρύνεται από την προδιαγραφή του PCI, καθώς είναι ένας μάλλον περιορισμένος πόρος. Το πρώτο μοντέλο που αναφέραμε έχει ιδιαίτερο ενδιαφέρον για τις υλοποιήσεις IEEE 1394 για δυο λόγους:

- Η φυσική μνήμη RAM του συστήματος είναι, υπό προϋποθέσεις, προσβάσιμη από απομακρυσμένους κόμβους μέσω της λειτουργίας Physical DMA των ελεγκτών Open HCI
- Είναι δυνατή η πρόσβαση στις διευθύνσεις μνήμης I/O των συσκευών του διαύλου PCI

Η λειτουργία πρόσβασης στη RAM μέσω DMA προϋποθέτει η γέφυρα που συνδέει το δίαυλο PCI με αυτόν του επεξεργαστή (host bus bridge) να επιτρέπει την πραγματοποίηση αυτών των συναλλαγών. Σε παλαιότερα συστήματα αυτό συνέβαινε πάντοτε, καθώς η εν λόγω γέφυρα ήταν απολύτως διαφανής. Σε πιο πρόσφατους υπολογιστές χρησιμοποιούνται μονάδες IOMMU (Input Output Memory Management Unit) που ελέγχουν τις διευθύνσεις που διέρχονται από αυτές και απαγορεύουν οποιαδήποτε συναλλαγή εκτός συγκεκριμένων περιοχών μνήμης. Με τον τρόπο αυτό προστατεύεται η ακεραιότητα του συστήματος από συσκευές που δυσλειτουργούν, χαρτογραφούνται οι διευθύνσεις I/O σε πιο κατάλληλες θέσεις και επιτρέπεται η απομόνωση μιας συσκευής για χρήση σε εφαρμογές εικονικών μηχανών (virtualisation). Οι μονάδες IOMMU, συνεπώς, απαιτούν προγραμματισμό προκειμένου να επιτρέψουν στις συσκευές του διαύλου να έχουν πρόσβαση στο σύνολο της μνήμης RAM.

Η πρόσβαση στη φυσική μνήμη του συστήματος από τον ελεγκτή IEEE 1394 έχει ως κύρια χρήση εφαρμογές επιτήρησης (monitoring) και αποσφαλμάτωσης (debugging) ενός υπολογιστή που βρίσκεται σε λειτουργία, χωρίς την ύπαρξη επιπλέον λογισμικού που μπορεί να επηρεάσει τον κώδικα που εκτελείται από τον επεξεργαστή. Με τη χρήση κατάλληλου λογισμικού συγχρονισμού, όμως, είναι δυνατή και η μεταφορά δεδομένων απευθείας από τη μνήμη με τον επεξεργαστή να χρησιμοποιείται μόνο κατά την ενημέρωση των απομακρυσμένων κόμβων για τις διευθύνσεις και τα μεγέθη των τοπικών ενταμιευτών.

Με τη λειτουργία Physical DMA προκύπτει και το πιο σημαντικό πρόβλημα ασφάλειας δεδομένων που επιφέρει η ύπαρξη θυρών IEEE 1394 σε έναν υπολογιστή, καθώς με την ίδια μέθοδο μπορούν να αναγνωσθούν ευαίσθητα δεδομένα κατευθείαν από τη μνήμη του. Ο περιορισμός της φυσικής πρόσβασης στο σύστημα και τις θύρες του σε έμπιστα άτομα, η απενεργοποίηση των θυρών που δεν χρησιμοποιούνται και ο προγραμματισμός του ελεγκτή Open HCI έτσι ώστε η πρόσβαση στη μνήμη να είναι επιλεκτική ή ακόμα και εντελώς απενεργοποιημένη μπορεί να μειώσει δραστικά αυτόν τον κίνδυνο.

Παρομοίως, η πρόσβαση στις διευθύνσεις μνήμης I/O άλλων συσκευών του υπολογιστή είναι επίσης δυνατή αλλά και χρήσιμη, υπό προϋποθέσεις. Εντός του ίδιου διαύλου PCI όλες οι συσκευές επικοινωνούν μεταξύ τους χωρίς περιορισμούς, αλλά η επικοινωνία ανάμεσα σε διαύλους PCI που συνδέονται μέσω κάποιας

γέφυρας (PCI-to-PCI bridge) ενδέχεται να μην είναι εξίσου απρόσκοπτη - το ίδιο προφανώς ισχύει και για τη μονάδα μεταγωγής του PCIe. Με βάση την προδιαγραφή PCI-to-PCI Bridge Architecture Specification Revision 1.1 που περιγράφει τις γέφυρες PCI-PCI, όμως, κάθε γέφυρα υποχρεούται να προωθεί όσες συναλλαγές προέρχονται από έναν δίαυλο με διευθύνσεις που δεν ανήκουν σε κάποια από τις συσκευές του διαύλου. Συμπεραίνουμε λοιπόν ότι ένας ελεγκτής IEEE 1394 μπορεί υπό φυσιολογικές συνθήκες να προσπελάσει οποιαδήποτε άλλη συσκευή που ανήκει στην ίδια ιεραρχία διαύλων PCI.

Πέρα από την εφικτότητα της επικοινωνίας με άλλες συσκευές, πρέπει παράλληλα να εξεταστεί και η χρησιμότητά της. Καθώς ο δίαυλος IEEE 1394 δεν μπορεί να χρησιμοποιηθεί για την αποστολή διακοπών ή για την πρόσβαση στο χώρο διευθύνσεων I/O του PCI, δεν είναι συνήθως δυνατή η απομακρυσμένη χρήση μιας συσκευής χωρίς ειδικό λογισμικό στον τοπικό κόμβο. Ωστόσο, κάποιες κατηγορίες συσκευών, όπως για παράδειγμα κάποιες κάρτες συλλογής δεδομένων (Data Acquisition, DAQ) και πολλές κάρτες γραφικών εξάγουν το σύνολο της μνήμης τους στο χώρο διευθύνσεων του συστήματος, προσφέροντας μια σειρά από ενδιαφέρουσες προοπτικές. Η απευθείας ανάκτηση των δεδομένων που έχουν συλλεγεί από μια κάρτα DAQ επιτρέπει την άμεση μεταφορά τους σε κάποια άλλη συσκευή IEEE 1394 με την ελάχιστη δυνατή καθυστέρηση. Αντιστοίχως, στην περίπτωση των καρτών γραφικών είναι συχνά προσβάσιμο το περιεχόμενο της οθόνης, επιτρέποντας λόγου χάρη σε έναν απομακρυσμένο κόμβο να την χρησιμοποιήσει απευθείας, χωρίς σύνθετα πρωτόκολλα δικτύωσης.

3.4 Υποστήριξη του διαύλου IEEE 1394 στο λειτουργικό σύστημα Linux

Ο πυρήνας του ανοικτού λειτουργικού συστήματος Linux απέκτησε προκαταρκτική υποστήριξη για ελεγκτές IEEE 1394 με τη δοκιμαστική έκδοση 2.3.40, η οποία ανακοινώθηκε στις αρχές 2000, ενώ η πρώτη σταθερή έκδοση με το υποσύστημα αυτό ήταν η 2.4.0 που κυκλοφόρησε τον Ιανουάριο του 2001. Οι πρόσφατες εκδόσεις του πυρήνα διαθέτουν σχεδόν πλήρη υποστήριξη για διαύλους IEEE 1394, με οδηγούς για πληθώρα συσκευών.

Από την έκδοση 2.6.22 (Ιούλιος 2007) και έπειτα ο πυρήνας του Linux διαθέτει δυο εναλλακτικές υλοποιήσεις του προτύπου IEEE 1394. Το υποσύστημα ieee1394 είναι η πλέον σταθερή εκδοχή, καθώς έχει φθάσει σε ένα καλό στάδιο ωριμότητας. Παρέχει υποστήριξη για ασύγχρονες και ισόχρονες συναλλαγές και μπορεί να λειτουργήσει ως διαχειριστής ισόχρονων πόρων (IRM). Υπάρχουν επίσης οδηγοί για τη συντριπτική πλειοψηφία των ελεγκτών που κυκλοφόρησαν στην αγορά, για συσκευές SBP-2, καθώς και για δικτύωση IP μέσω του διαύλου IEEE 1394.

Τα βασικά μειονεκτήματα αυτής της υλοποίησης είναι η απουσία δυνατοτήτων διαχειριστή διαύλου (BM) και μεθόδων βελτιστοποίησης της λειτουργίας του, ενώ δεν υποστηρίζονται οι μέθοδοι επιτάχυνσης του διαύλου που εισήγαγε η τροπολογία IEEE Std 1394a-2000, όπως η επιτάχυνση της διαιτησίας με πακέτα επιβεβαίωσης ACK (arbitration acceleration). Επιπλέον η υποστήριξη συναλλαγών ασύγχρονης ροής είναι ελλιπής τόσο στην υλοποίηση, όσο και στην τεκμηρίωσή της.

Αντίθετα, το πειραματικό υποσύστημα firewire έχει σχεδιαστεί έτσι ώστε, όταν ολοκληρωθεί, να υποστηρίζει πλήρως το IEEE Std 1394a-2000. Εμφανίζει σταθερά καλύτερες επιδόσεις, διαθέτει δυνατότητες Bus Master και συναρτήσεις για τη βελτιστοποίηση των διαστημάτων αδράνειας του διαύλου (gap count optimisation), ενώ η αρχιτεκτονική του είναι ορθότερα δομημένη και πιο επεκτάσιμη. Από την άλλη έχει περιορισμένη υποστήριξη για ελεγκτές IEEE 1394, παρουσιάζει ασταθή λειτουργία σε κάποιες περιπτώσεις, ενώ κάποια τμήματα του προτύπου έχουν ατελή ή προβληματική υλοποίηση. Για τους λόγους αυτούς κατά την πραγματοποίηση των μετρήσεων σε αυτή την εργασία αποφεύχθηκε η χρήση του υποσυστήματος firewire και χρησιμοποιήθηκε μόνο το ieee1394.

Ανεξάρτητα από το υποσύστημα IEEE 1394 που χρησιμοποιεί ο πυρήνας, παρέχονται μια ομάδα κλήσεων συστήματος (raw1394) και μια συνοδευτική βιβλιοθήκη λογισμικού (libraw1394) που επιτρέπουν την πραγματοποίηση συναλλαγών και τη διαχείριση του διαύλου από προγράμματα στο χώρο χρήστη (user space). Αυτό καθιστά δυνατό τον έλεγχο συσκευών IEEE 1394 χωρίς τη συγγραφή κώδικα στο χώρο πυρήνα (kernel space). Μόνη προϋπόθεση είναι το λογισμικό ελέγχου να εκτελείται με επαρκή δικαιώματα, καθώς η πρόσβαση στο υποσύστημα IEEE 1394 από τους χρήστες είναι αυστηρά ελεγχόμενη λόγω των εγγενών ζητημάτων ασφαλείας που συνεπάγεται. Τυπικά απαιτείται η χρήση του λογαριασμού υπερχρήστη (root) προκειμένου να εξασφαλιστεί το απαραίτητο επίπεδο πρόσβασης στο σύστημα.

Η βιβλιοθήκη libraw1394 έχει γραφεί στη γλώσσα προγραμματισμού C και αποτελεί ουσιαστικά ένα επίπεδο αφαίρεσης (abstraction layer) που ελαχιστοποιεί την επίδραση τυχόν αλλαγών του πυρήνα στη λειτουργία των εφαρμογών. Παρέχει

μια σειρά από ευκολίες για τη χρήση του διαύλου, ενώ επιτρέπει και την τροποποίηση της τοπικής μνήμης ρυθμίσεων (Configuration ROM), για εφαρμογές όπου ο τοπικός υπολογιστής προσφέρει κάποια υπηρεσία στους υπόλοιπους κόμβους.

Συνοπτικά, κατά την τέλεση μιας ασύγχρονης συναλλαγής μέσω του συστήματος raw1394 ακολουθείται η παρακάτω διαδικασία:

- Το λογισμικό ελέγχου υποβάλλει μια αίτηση στον πυρήνα του λειτουργικού με μια κλήση συστήματος
- Ο πυρήνας του λειτουργικού χρησιμοποιεί τον ελεγκτή IEEE 1394 για την αποστολή της αίτησης στον απομακρυσμένο κόμβο
- Ο απομακρυσμένος κόμβος επεξεργάζεται την αίτηση και αποκρίνεται
- Ο πυρήνας του τοπικού κόμβου λαμβάνει την απόκριση της συναλλαγής μέσω του ελεγκτή
- Το λογισμικό ελέγχου λαμβάνει την απόκριση μέσω μιας δεύτερης κλήσης συστήματος

Οι δυο κλήσεις συστήματος που απαιτούνται είναι και η βασική αιτία για την οποία οι μέγιστες επιδόσεις που επιτυγχάνονται μέσω του μηχανισμού raw1394 είναι περιορισμένες. Κάθε κλήση συστήματος οδηγεί σε μια μεταγωγή περιεχομένου (context switch), διαδικασία η οποία στο Linux, όπως και σε όλα τα λειτουργικά συστήματα, είναι ιδιαίτερα χρονοβόρα. Με στόχο τη μείωση της επίδρασης αυτών των καθυστερήσεων, η βιβλιοθήκη libraw1394 επιτρέπει των παραλληλισμό των συναλλαγών και συνεπώς την ταυτόχρονη εκτέλεση κάποιων από τα στάδιά τους, αυξάνοντας σε κάποιο βαθμό την απόδοση, ιδίως όταν ο κόμβος προορισμού έχει τη δυνατότητα εξυπηρέτησης πολλαπλών συναλλαγών.

Εδώ πρέπει να σημειωθεί ότι η libraw1394 δεν διαχωρίζει τις ασύγχρονες συναλλαγές με φορτίο τετράδας από αυτές με φορτίο τμήματος δεδομένων. Απλά, επειδή οι συναλλαγές με φορτίο τετράδας κάνουν καλύτερη χρήση του διαύλου, τις χρησιμοποιεί αυτόματα όποτε πρόκειται να μεταφερθούν 4 bytes δεδομένων.

Ιδιαίτερη προσοχή έχει δοθεί στην υποστήριξη ισόχρονων συναλλαγών, καθώς αυτές διέπονται από αυστηρούς χρονικούς περιορισμούς. Αξιοποιώντας το γεγονός ότι αυτές οι συναλλαγές δεν είναι αμφίδρομες, ο μηχανισμός raw1394 επιτρέπει την ομαδική μεταφορά πακέτων από το χώρο χρήστη στο χώρο του πυρήνα με μόνο μια κλήση συστήματος. Αυτό επιτρέπει την απρόσκοπτη πραγματοποίηση ισόχρονων συναλλαγών, αποφεύγοντας ταυτόχρονα τον τεράστιο επεξεργαστικό φόρτο που προκύπτει όταν απαιτούνται μία η περισσότερες κλήσεις συστήματος ανά πακέτο, όπως συμβαίνει με τις ασύγχρονες συναλλαγές.

Μια ακόμα σημαντική λειτουργία υψηλού επιπέδου που παρέχει ο πυρήνας του Linux είναι η αντιστοίχιση ενός συνεκτικού ενταμιευτή από τη φυσική μνήμη του συστήματος σε ένα εύρος διευθύνσεων του διαύλου IEEE 1394 (address range mapping - ARM). Όταν ο ελεγκτής του κόμβου λάβει μια αίτηση για κάποια συναλλαγή σε αυτό το εύρος, ενημερώνει με μια διακοπή τον κεντρικό επεξεργαστή και ο πυρήνας του λειτουργικού αναλαμβάνει τον προγραμματισμό των μονάδων DMA του ελεγκτή για την εξυπηρέτησή της. Αυτή η μέθοδος είναι πιο αργή από

την επεξεργασία των συναλλαγών σε εξειδικευμένο υλικό, αλλά είναι πολύ ευέλικτη και επιτρέπει σε έναν υπολογιστή Linux να δράσει ως μια πλήρης συσκευή IEEE 1394.

Για την αξιοποίηση αυτής της δυνατότητας, έχει υλοποιηθεί ένα σύνολο από κλήσεις συστήματος για την πρόσβαση στον ενταμιευτή και την ενημέρωση του χώρου χρήστη για όσες συναλλαγές πραγματοποιούνται σε αυτόν από άλλους κόμβους. Αυτές οι υπηρεσίες επιτρέπουν, ανάμεσα στα άλλα, την έμμεση δημιουργία περιοχών μνήμης που λειτουργούν ως καταχωρητές, με την εκάστοτε εφαρμογή ελέγχου να αντιλαμβάνεται κάθε μεταβολή τους. Υπάρχει επίσης η δυνατότητα χειρισμού των εισερχόμενων αιτήσεων εξ ολοκλήρου από λογισμικό στο χώρο χρήστη, αν και αυτό εισάγει σημαντικές καθυστερήσεις στην αποστολή των απο-κρίσεων.

Όλες οι εφαρμογές και οι βιβλιοθήκες που εκτελούνται στο περιβάλλον GNU/Linux και υποστηρίζουν πρωτόκολλα όπως το DV για καταναλωτικές συσκευές video, το IIDC και το IEC 61883 έχουν υλοποιηθεί με αυτήν ακριβώς τη μέθοδο. Μια εξίσου ενδιαφέρουσα εφαρμογή είναι η προσομοίωση μιας μονάδας αποθήκευσης δεδομένων SBP-2 με τη χρήση ενός υπολογιστή γενικού σκοπού, λειτουργία που συνήθως γίνεται με τη χρήση αφοσιωμένου υλικού.

Η μόνη αδυναμία της προσέγγισης που θέλει το λογισμικό ελέγχου μιας συσκευής να εκτελείται στο χώρο χρήστη είναι οι χρόνοι καθυστέρησης (latency) που προκύπτουν. Στο χώρο πυρήνα, για παράδειγμα, μια διακοπή από το υλικό, όπως αυτές που παράγονται από τον ελεγκτή IEEE 1394 για κάθε εισερχόμενο πακέτο, προκαλεί την άμεση εκτέλεση του κώδικα που τη χειρίζεται από τον κεντρικό επεξεργαστή. Αντιθέτως, όταν ο κώδικας αυτός βρίσκεται στο χώρο χρήστη προκύπτει μια αναπόφευκτη καθυστέρηση μέχρι ο χρονοδρομολογητής του λειτουργικού συστήματος (scheduler) να πραγματοποιήσει τη μεταγωγή περιεχομένου (context switch) στην αντίστοιχη διεργασία, κάτι που μπορεί να αργήσει ακόμα περισσότερο αν κάποιες άλλες διεργασίες επιλεγούν για εκτέλεση στο ενδιάμεσο. Συνολικά ένας κύκλος αίτησης/απάντησης, όπως λόγου χάρη μια ασύγχρονη συναλλαγή ανάγνωσης, συνεπάγεται δύο κλήσεις συστήματος και έναν τουλάχιστον κύκλο χρονοδρομολόγησης διεργασιών, κάτι που αθροιστικά μπορεί να ισοδυναμεί με έναν χρόνο καθυστέρησης μη αποδεκτό για χρονικά κρίσιμες εφαρμογές.

4 Συμπεριφορά του διαύλου IEEE 1394 σε εφαρμογές κινούμενης εικόνας

4.1 Ανάπτυξη λογισμικού

Για τη μελέτη των ιδιαίτερων χαρακτηριστικών και της συμπεριφοράς του διαύλου IEEE 1394 αναπτύχθηκε η πειραματική εφαρμογή v1394 και μια σειρά από βοηθητικά εργαλεία για το περιβάλλον του λειτουργικού συστήματος Linux. Το λογισμικό v1394 επιτρέπει τη λήψη και την αποστολή κινούμενης εικόνας από και προς έναν απομακρυσμένο κόμβο, ενώ ταυτόχρονα παρέχει μια σειρά από χρήσιμα στατιστικά στοιχεία για τη λειτουργία του. Αναπτύχθηκε στη γλώσσα προγραμματισμού C, με τη βιβλιοθήκη libraw1394 να παρέχει τις υπηρεσίες πρόσβασης στο υποσύστημα IEEE 1394 του πυρήνα και τη βιβλιοθήκη GNU ncurses τα βασικά δομικά στοιχεία του περιβάλλοντος χρήστη.

Κατά την ανάπτυξη του v1394 επιλέχθηκε μια αρθρωτή δομή, έτσι ώστε να μεγιστοποιηθεί η ευελιξία του με την ελάχιστη δυνατή λειτουργική επικάλυψη των διαφόρων τμημάτων κώδικα. Επιπλέον κρίθηκε απαραίτητη η χρήση μιας πολυνηματικής (multithreaded) αρχιτεκτονικής, έτσι ώστε να είναι δυνατόν να παραλληλιστούν οι διαδικασίες εισόδου και εξόδου δεδομένων που πραγματοποιεί το πρόγραμμα, αποφεύγοντας τις περιόδους αδράνειας του κεντρικού επεξεργαστή. Κατά την εκτέλεσή του, η διεργασία του προγράμματος αποτελείται συνολικά από τρία νήματα (threads):

- Νήμα εισόδου: Χρησιμοποιεί μια από τις διαθέσιμες συναρτήσεις εισόδου για την τροφοδοσία του προγράμματος με δεδομένα κινούμενης εικόνας.
- Νήμα εξόδου: Αναλαμβάνει την έξοδο δεδομένων με τη χρήση της συνάρτησης που έχει επιλέξει ο χρήστης.
- Νήμα ελέγχου: Φροντίζει για την περιοδική συλλογή στατιστικών στοιχείων από τα δυο άλλα νήματα, αλλά και για τη διακοπή της λειτουργίας του προγράμματος όταν το ζητήσει ο χρήστης.

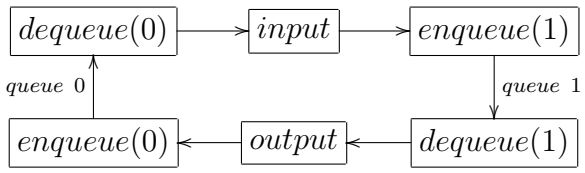
Η βασική μονάδα δεδομένων είναι το πλαίσιο (frame), που ορίζεται ως ένα από μια ακολουθία στιγμιοτύπων που συνιστούν μια ροή κινούμενης εικόνας. Προγραμματιστικά είναι ένας απλός ενταμιευτής με το κατάλληλο μέγεθος - ένα πλαίσιο με πλάτος X και ύψος Y εικονοστοιχεία (pixels) με B bytes για την αναπαράσταση κάθε εικονοστοιχείου απαιτεί μια περιοχή μνήμης με μέγεθος τουλάχιστον $X * Y * B$ bytes. Ο χώρος που απαιτείται για κάθε εικονοστοιχείο εξαρτάται από το βάθος χρώματος (color depth) του πλαισίου. Το λογισμικό που αναπτύξαμε επιτρέπει την ρύθμιση και των τριών βασικών παραμέτρων που περιγράφουν ένα πλαίσιο. Το v1394 διαθέτει δύο διακριτές ουρές (queues), έστω 0 και 1, που επιτρέπουν στα πλαίσια να μετακινούνται από το νήμα εισόδου στο νήμα εξόδου και αντιστρόφως.

Τα νήματα εισόδου και εξόδου έχουν την ίδια ακριβώς δομή, με μόνη διαφορά τη συνάρτηση χειρισμού πλαισίων που καλείται κάθε φορά και τις ουρές που χρησιμοποιούνται. Κάθε νήμα αποτελείται από ένα βρόχο, ο οποίος σε κάθε επανάληψη ανακτά ένα πλαίσιο από την ουρά εισόδου (dequeue) του νήματος, καλεί

μια συνάρτηση για την επεξεργασία του πλαισίου και στη συνέχεια το τοποθετεί στο τέλος της ουράς εξόδου (enqueue). Έτσι κάθε πλαίσιο διανύει τον ακόλουθο κύκλο ζωής:

- dequeue(0): Το νήμα εισόδου ανακτά ένα πλαίσιο από την ουρά 0.
- input: Το ίδιο νήμα χρησιμοποιεί την επιλεγμένη συνάρτηση εισόδου και καταγράφει το επόμενο στιγμιότυπο στον ενταμιευτή του πλαισίου.
- enqueue(1): Το πλαίσιο τοποθετείται στο τέλος της ουράς 1.
- dequeue(1): Το πλαίσιο μετακινείται στην ουρά έως ότου ανακτηθεί από το νήμα εξόδου.
- output: Το νήμα εξόδου τροφοδοτεί με τα δεδομένα του πλαισίου τη συνάρτηση εξόδου.
- enqueue(0): Το πλαίσιο τοποθετείται στο τέλος της ουράς 0, έτσι ώστε μελλοντικά να ανακτηθεί και πάλι από το νήμα εισόδου.

Σχήμα 46: Κύκλος ζωής πλαισίου



Οι συναρτήσεις εισόδου και εξόδου λαμβάνουν ως κύρια ορίσματα τους ένα πλαίσιο προς επεξεργασία, μια δομή (struct) με τις παραμέτρους της λειτουργίας τους και μια δομή για την αποθήκευση στατιστικών στοιχείων. Εξωτερικά δε διαφέρουν μεταξύ τους, με ορισμένες από αυτές να μπορούν να χρησιμοποιηθούν τόσο ως συναρτήσεις εξόδου, όσο και εισόδου. Οι μέθοδοι εισόδου δεδομένων που προσφέρονται είναι οι ακόλουθες:

- Ανάγνωση δεδομένων από απομακρυσμένο κόμβο με ασύγχρονες συναλλαγές
- Λήψη δεδομένων με ισόχρονες συναλλαγές
- Εγγραφή δεδομένων από απομακρυσμένο κόμβο σε τοπικό ενταμιευτή
- Ανάκτηση πλαισίων από αρχείο
- Παραγωγή δεδομένων από μια γεννήτρια προτύπων πλαισίων με οριζόντιες και κάθετες γραμμές (pattern generator)

- Παραγωγή δεδομένων από μια γεννήτρια μονόχρωμων πλαισίων (zero)
- Απλή συνάρτηση που δεν επηρεάζει τα περιεχόμενα των πλαισίων που διέρχονται από αυτήν (null)

Αντιστοίχως, οι εναλλακτικές συναρτήσεις εξόδου του v1394 είναι οι εξής:

- Εμφάνιση των διαδοχικών πλαισίων στην οθόνη του υπολογιστή (monitor)
- Εγγραφή δεδομένων σε απομακρυσμένο κόμβο με ασύγχρονες συναλλαγές
- Εκπομπή δεδομένων με ισόχρονες συναλλαγές
- Εγγραφή δεδομένων σε τοπικό ενταμιευτή προσβάσιμο από το δίαυλο IEEE 1394
- Εγγραφή πλαισίων σε αρχείο
- Απλή συνάρτηση που αγνοεί τα περιεχόμενα των πλαισίων που διέρχονται από αυτήν (null)

Οι συναρτήσεις που δρουν τελώντας ασύγχρονες συναλλαγές μέσω του διαύλου IEEE 1394 έχουν μια σειρά από παραμέτρους που ελέγχουν τη λειτουργία τους. Τα πιο βασικά ορίσματα αυτών των συναρτήσεων είναι ο χαρακτηριστικός αριθμός UID που περιέχεται στη μνήμη ρυθμίσεων του απομακρυσμένου κόμβου στον οποίο αποστέλλονται οι αιτήσεις των συναλλαγών και η διεύθυνση (offset) στην οποία αναφέρονται. Επιτρέπουν επίσης τη ρύθμιση του μέγιστου φορτίου ενός πακέτου και της μεταβλητής gap_count του φυσικού επιπέδου, που όπως είδαμε στην παράγραφο 2.7.1 έχουν καθοριστική επίδραση στην απόδοση του διαύλου.

Κατά τη λήψη στατιστικών στοιχείων από την τέλεση ασύγχρονων συναλλαγών εγγραφής, το λογισμικό v1394 είναι σε θέση να διαχωρίσει τις ενοποιημένες συναλλαγές, οι οποίες δεν εμπεριέχουν πακέτο απόκρισης, από τις διαχωρισμένες ή συνενωμένες. Με αυτό τον τρόπο επιτρέπεται η καλύτερη αξιολόγηση των αποτελεσμάτων που προκύπτουν, καθώς κάθε τύπος έχει διαφορετικά χαρακτηριστικά.

Παρομοίως, κατά την τέλεση ισόχρονων συναλλαγών είναι δυνατή η επιλογή του καναλιού εκπομπής και λήψης, καθώς και του μέγιστου φορτίου κάθε πακέτου. Παρέχεται επίσης η δυνατότητα προαποθήκευσης ενός αριθμού πακέτων προκειμένου να αντιμετωπιστούν επιτυχώς ανωμαλίες κατά την περιοδική αποστολή ή λήψη δεδομένων.

Εδώ πρέπει να σημειωθεί ότι η ανυπαρξία εγγενών μηχανισμών αξιοπιστίας στις ισόχρονες συναλλαγές δημιουργεί μια σημαντική δυσκολία στο συγχρονισμό και την οριοθέτηση των πλαισίων κατά τη μετάδοσή τους. Για το λόγο αυτό, τα τέσσερα πρώτα bytes του φορτίου κάθε ισόχρονου πακέτου χρησιμοποιούνται για τον προσδιορισμό της θέσης μέσα στο πλαίσιο όπου αναφέρονται τα υπόλοιπα δεδομένα. Αυτό οδηγεί σε μια μετρήσιμη απώλεια επιδόσεων, ιδίως σε μικρά μεγέθη πακέτων, αλλά απλοποιεί σημαντικά τη δομή του λογισμικού.

Για τη μεγιστοποίηση της απόδοσης του διαύλου το λογισμικό v1394 περιλαμβάνει μια απλή συνάρτηση βελτιστοποίησης των διαστημάτων αδράνειας (gap count optimisation). Όπως αναφέραμε στην παράγραφο 3.4, το υποσύστημα ieee1394

του πυρήνα Linux δε διαθέτει κάποια αντίστοιχη λειτουργία, αν και με βάση την τροπολογία IEEE Std 1394a-2000 η παρουσία της είναι επιτρεπτή σε κόμβους με δυνατότητες IRM. Το αποτέλεσμα είναι η μεταβλητή gap_count του φυσικού επιπέδου να λαμβάνει εξ ορισμού τη μέγιστή της τιμή 63_{10} , μειώνοντας δραστικά τις επιδόσεις. Το πρότυπο IEEE 1394 συνιστά να χαρτογραφείται η τοπολογία του διαύλου και στη συνέχεια η μεταβλητή αυτή να τίθεται στην ελάχιστη δυνατή τιμή, ανάλογα με τη μέγιστη απόσταση ανάμεσα σε δύο κόμβους.

Επειδή η σκιαγράφηση της τοπολογίας του διαύλου είναι μια ιδιαίτερα σύνθετη διαδικασία, απλά χρησιμοποιούμε το συνολικό αριθμό κόμβων N για τον υπολογισμό ενός άνω ορίου για τη μέγιστη απόστασή τους, το οποίο προφανώς θα είναι $N - 1$. Στη συνέχεια χρησιμοποιούμε τον πίνακα που παρέχει το πρότυπο IEEE 1394 για την εύρεση μιας κατάλληλης τιμής gap_count. Η μέθοδος αυτή δεν προσφέρει πάντα το απολύτως βέλτιστο αποτέλεσμα, αλλά φθάνει πολύ κοντά σε αυτό σε διαύλους με μικρό αριθμό κόμβων, όπως αυτοί που συναντώνται συνήθως σε προσωπικούς υπολογιστές.

Τόσο για τις ασύγχρονες, όσο και για τις ισόχρονες συναλλαγές, έχει ιδιαίτερη σημασία η μέριμνα για το χειρισμό της διαδικασίας επανεκκίνησης και αυτοδιαμόρφωσης του διαύλου, η οποία μπορεί να προκληθεί λόγω κάποιας σκόπιμης μεταβολής της τοπολογίας του διαύλου από το χρήστη, μετά από απαίτηση του επιπέδου διαχείρισης ή ακόμα και από κάποια τυχαία ηλεκτρική ανωμαλία. Η διαδικασία αυτή διακόπτει όλες τις συναλλαγές, δυνητικά μεταβάλλει τις φυσικές διευθύνσεις των κόμβων, ενώ χάνονται όλες οι δεσμεύσεις πόρων στο διαχειριστή ισόχρονων συναλλαγών.

Κατά την ανάπτυξη του λογισμικού v1394, έγινε συγκεκριμένη προσπάθεια έτσι ώστε να καταστεί ανθεκτικό σε αυτού του τύπου τη διαταραχή. Για τις ασύγχρονες συναλλαγές, ο κόμβος προορισμού ορίζεται όχι με τη φυσική του διεύθυνση, αλλά χρησιμοποιώντας τον χαρακτηριστικό κωδικό αριθμό UID που φέρει. Στην περίπτωση επανεκκίνησης του διαύλου, ανακτώνται οι αριθμοί UID από όλους τους κόμβους και έτσι προσδιορίζεται η νέα φυσική διεύθυνση του επιλεγμένου κόμβου. Αντίστοιχα, όταν τελούνται ισόχρονες συναλλαγές, το v1394 δεσμεύει εκ νέου τους απαραίτητους ισόχρονους πόρους, επιτρέποντας την απρόσκοπτη συνέχιση της μεταφοράς δεδομένων.

Οι συναρτήσεις που χρησιμοποιούν έναν εξωτερικά προσβάσιμο τοπικό ενταμιευτή επιτρέπουν την επιλογή της διεύθυνσης, στην οποία θα γίνει η αντιστοίχισή του (address range mapping). Έχουν, επίσης, καταχωρημένη και μια μέθοδο ειδοποίησης (notifier) που ενεργοποιείται όταν υπάρξει εξωτερική πρόσβαση από κάποιον κόμβο στην τελευταία τετράδα του ενταμιευτή. Αυτό είναι μια απλή, πλην όμως αποτελεσματική, μέθοδος για να ειδοποιείται το λογισμικό όταν ολοκληρωθεί η ανάγνωση ή ανανέωση του ενταμιευτή από κάποιον εξωτερικό κόμβο.

Από τις υπόλοιπες μεθόδους χειρισμού πλαισίων, άξια μνείας είναι η συνάρτηση προβολής κινούμενης εικόνας στην οθόνη του υπολογιστή, μέσω της διαδοχικής εμφάνισης των εισερχόμενων πλαισίων. Για να επιτευχθεί αυτή η λειτουργία χρησιμοποιείται η βιβλιοθήκη Simple Directmedia Layer (SDL), η οποία παρέχει πληθώρα υπηρεσιών που είναι συχνά απαραίτητες σε εφαρμογές πολυμέσων και παιχνίδια. Με τη βιβλιοθήκη αυτή μπορεί να γραφεί φορητός (portable) κώδικας, χωρίς να χρειάζεται γνώση του υλικού πάνω στο οποίο να εκτελεστεί, καθώς

η SDL εμπεριέχει το απαραίτητο υπόστρωμα αφαίρεσης (abstraction layer). Ο κύριος λόγος που επιλέχθηκε αυτή η βιβλιοθήκη ήταν η ευκολία συγγραφής του ζητούμενου κώδικα, η οποία αντισταθμίζει το γεγονός ότι λόγω της έντονης χρήσης του επεξεργαστή οι επιδόσεις της δεν είναι ιδιαίτερα υψηλές.

Σημαντικό χαρακτηριστικό της ομάδας προγραμμάτων που αναπτύχθηκαν είναι και η δυνατότητα μεταβολής της μνήμης ρυθμίσεων του τοπικού κόμβου με την προσθήκη επιπλέον φύλλων κειμένου. Χρησιμοποιώντας αυτή την τεχνική αναρτώνται σημαντικά στοιχεία, όπως λόγου χάρη οι παράμετροι του πλαισίου στο v1394 ή η διεύθυνση ενός εξωτερικά προσβάσιμου ενταμιευτή. Το v1394 μπορεί να αξιοποιήσει αυτά τα φύλλα κειμένου σε έναν απομακρυσμένο κόμβο προκειμένου να θέσει αυτόματα κάποιες από τις παραμέτρους του, μειώνοντας τον αριθμό των ρυθμίσεων που πρέπει να εκτελέσει χειροκίνητα ο χρήστης.

4.2 Πειραματικές δοκιμές και αποτελέσματα

Για την πραγματοποίηση των πειραματικών μετρήσεων χρησιμοποιήθηκαν δύο ηλεκτρονικοί υπολογιστές με το λειτουργικό σύστημα Linux, οι οποίοι φέρουν ελεγκτές IEEE 1394 σε δίαυλο συστήματος PCI, με μέγιστο υποστηριζόμενο ρυθμό μεταφοράς δεδομένων την ταχύτητα S400. Οι δύο αυτοί κόμβοι ήταν συνδεδεμένοι μεταξύ τους μέσω ενός διαύλου IEEE 1394, επιτρέποντας την μεταξύ τους επικοινωνία.

Ο πρώτος κόμβος (A), στον οποίο εκτελέστηκε το λογισμικό v1394 στις περισσότερες περιπτώσεις έχει τα ακόλουθα βασικά τεχνικά χαρακτηριστικά:

- Κεντρικός επεξεργαστής: Intel Pentium M 1.70GHz
- Μνήμη: 1024 MiB με μέγιστη ταχύτητα μεταφοράς δεδομένων περίπου 2100 MB/sec
- Ελεγκτής IEEE 1394: Texas Instruments OHCI-Lynx PCI
- Κάρτα γραφικών: nVidia GeForce FX Go5200 64MiB AGP

Ο δεύτερος υπολογιστής (B), που είχε συνήθως το ρόλο του κόμβου προορισμού, είναι διαφορετικός σε όλα τα σημεία:

- Κεντρικός επεξεργαστής: AMD Athlon 900 MHz
- Μνήμη: 256 MiB με μέγιστη ταχύτητα μεταφοράς δεδομένων περίπου 533 MB/sec
- Ελεγκτής IEEE 1394: VIA Technologies Inc. VT6306 PCI
- Κάρτα γραφικών: S3 Virge 2MiB PCI

Ο δεύτερος κόμβος, συνεπώς, έχει σαφώς υποδεέστερα τεχνικά χαρακτηριστικά, αν και εξακολουθεί να υπερκαλύπτει τις ανάγκες του ελεγκτή IEEE 1394 που διαθέτει. Επιπροσθέτως, οι περισσότερες συναλλαγές είναι συμμετρικές ως προς τη χρήση των διαθέσιμων επεξεργαστικών πόρων και στους δυο υπολογιστές, οπότε όσες αποκλίσεις προκύπτουν μπορούν με σχετική βεβαιότητα να αποδοθούν στους διαφορετικούς ελεγκτές IEEE 1394 που χρησιμοποιούνται: διαπιστώσαμε σχετικά γρήγορα ότι το ολοκληρωμένο κύκλωμα της Texas Instruments ήταν σαφώς ανώτερο από αυτό της VIA Technologies.

Κατά τις αρχικές δοκιμές με τη διάταξη αυτή, χρησιμοποιήθηκαν και τα δύο εναλλακτικά υποσυστήματα IEEE 1394 που διαθέτει ο πυρήνας του Linux στις πλέον πρόσφατες εκδόσεις του. Η υπό ανάπτυξη εκδοχή firewire έδωσε αρχικά καλύτερες επιδόσεις από το ieee1394 στις ασύγχρονες συναλλαγές, κυρίως λόγω των υπηρεσιών βελτιστοποίησης του διαύλου που παρέχει. Παράλληλα, όμως, παρουσίασε πληθώρα προβλημάτων σταθερότητας τόσο στο χειρισμό του διαύλου όσο και στη συνολικότερη λειτουργία του πυρήνα, ενώ η υποστήριξη ισόχρονων συναλλαγών ήταν ουσιαστικά ανύπαρκτη. Επιπλέον, διαπιστώθηκαν κάποια προβλήματα συμβατότητας με τη βιβλιοθήκη libraw1394, όπως λόγου χάρη στο χειρισμό της μνήμης ρυθμίσεων του τοπικού κόμβου. Για τους λόγους αυτούς προτιμήθηκε η

χρήση της εναλλακτικής ieee1394, με το λογισμικό v1394, το οποίο εκτελείται στο χώρο χρήστη, να παρέχει κάποιες στοιχειώδεις υπηρεσίες βελτιστοποίησης του διαύλου.

Με τη βοήθεια του λογισμικού v1394, το οποίο περιγράφηκε στην παράγραφο 4.1, εκτελέστηκε σειρά δοκιμών με διαφορετικές παραμέτρους, προκειμένου να προσδιοριστεί η επίδρασή τους στη λειτουργία του συστήματος. Πέρα από το είδος των συναλλαγών, οι σημαντικότερες παράμετροι ήταν το μέγεθος των πακέτων και η τιμή της μεταβλητής gap_count του φυσικού επιπέδου, καθώς επηρεάζουν απευθείας τη μορφή και το χρονισμό των πακέτων που μεταδίδονται στο δίαυλο, και όχι μόνο τη συμπεριφορά του υποσυστήματος ieee1394. Σε όλες τις δοκιμές τα πλαίσια κινούμενης εικόνας είχαν ανάλυση $640 * 480$, βάθος χρώματος 16 bit και συνολικό μέγεθος 600 KiB ακριβώς. Για την επεξεργασία των πλαισίων στον τοπικό κόμβο χρησιμοποιήθηκε η συνάρτηση null, η οποία δεν τροποποιεί τα περιεχόμενά τους, ελαχιστοποιώντας έτσι την επεξεργαστική ισχύ που διατίθεται σε λειτουργίες που δεν αφορούν άμεσα το δίαυλο IEEE 1394.

Οι χαρακτηριστικές ποσότητες, των οποίων η μέτρηση έχει ιδιαίτερο ενδιαφέρον σε κάθε σύστημα, είναι η διαμεταγωγή (throughput) και η καθυστέρηση (latency), με το λογισμικό να υπολογίζει τις μέσες τιμές τους κατά τη διάρκεια κάθε κύκλου. Κατά τη διάρκεια των δοκιμών, διαπιστώθηκε ότι δεν ήταν δυνατόν να επιτευχθούν ταυτόχρονα οι βέλτιστες τιμές και για τα δυο μεγέθη. Για τη μεγιστοποίηση της διαμεταγωγής απαιτείται υψηλός βαθμός πλήρωσης τόσο των ουρών του v1394 για τα πλαίσια, όσο και των ουρών του πυρήνα για τα πακέτα IEEE 1394. Αυτό όμως έχει το αποτέλεσμα να αυξάνονται οι μετρούμενοι χρόνοι απόκρισης, οπότε για τη μέτρηση της καθυστέρησης οι ουρές του συστήματος διατηρούνται στον ελάχιστο δυνατό βαθμό πλήρωσης.

4.2.1 Ασύγχρονες συναλλαγές

Κατά την εκτέλεση ασύγχρονων συναλλαγών η απόδοση του συστήματος παρουσιάστηκε σημαντικά χαμηλότερη από τα θεωρητικά μέγιστα της παραγράφου 2.7.1. Ο κύριος λόγος για αυτό το αποτέλεσμα είναι οι χρόνοι καθυστέρησης που υπεισέρχονται στη διαδικασία τέλεσης των συναλλαγών, οι οποίοι είναι σε πολλές περιπτώσεις πολλαπλάσιοι των χρόνων μετάδοσης δεδομένων στο δίαυλο IEEE 1394. Για το λόγο αυτό, το λογισμικό v1394 έχει τη δυνατότητα παραλληλισμού των συναλλαγών, στέλνοντας έναν αριθμό αιτήσεων πριν ακόμα λάβει τις αντίστοιχες αποκρίσεις. Αυτό επιτρέπει την ύπαρξη δραστηριότητας στο δίαυλο ακόμα και όταν αναμένεται κάποια μεταγωγή ρυθμού λειτουργίας (context switch) για την επικοινωνία του πυρήνα με το λογισμικό ελέγχου που βρίσκεται στο χώρο χρήστη. Αυτή η τεχνική, ωστόσο, έχει περιορισμένη αποτελεσματικότητα με πολύ μικρά μεγέθη πακέτων, καθώς τότε το κύριο σημείο στένωσης είναι ο κεντρικός επεξεργαστής που αδυνατεί να ανταπεξέλθει στο φόρτο παραγωγής πακέτων με τόσο μεγάλη συχνότητα.

Απαραίτητη προϋπόθεση για τον παραλληλισμό των συναλλαγών είναι η ύπαρξη αντίστοιχης υποστήριξης στον κόμβο προορισμού. Η ύπαρξη μιας απλής ουράς επιτρέπει την υποβολή πολλαπλών αιτήσεων, κάτι που στην περίπτωση μας

είναι απαραίτητο για την απαλοιφή των καθυστερήσεων που εισάγονται από τις κλήσεις συστήματος. Η πλήρωση αυτής της ουράς μεγιστοποιεί τη διαμεταγωγή δεδομένων (throughput), αλλά ταυτόχρονα αυξάνει την καθυστέρηση ανά συναλλαγή (latency). Σε περίπτωση υπερχείλισης της ουράς, ζητείται από τον κόμβο προέλευσης της αίτησης η αναβολή της συναλλαγής για μεταγενέστερη χρονική στιγμή.

Για την ουσιαστική επιτάχυνση των συναλλαγών, ωστόσο, απαιτείται η παράλληλη εξυπηρέτησή τους σε όλα τα στάδια. Στις μετρήσεις που πραγματοποιήθηκαν, όμως, χρησιμοποιήθηκε πάντοτε ένας μοναδικός χώρος μνήμης εντός της ίδιας υπομονάδας του απομακρυσμένου υπολογιστή. Κατά συνέπεια δεν μπορούσε να υπάρξει ταυτοχρονισμός κατά τη μεταφορά δεδομένων από και προς τον ελεγκτή IEEE 1394, οδηγώντας το δίαυλο σε εκτενείς περιόδους αδράνειας.

Στο σύστημα που χρησιμοποιήθηκε για τις δοκιμές ο αριθμός ταυτόχρονων συναλλαγών που υποστηρίζεται εξαρτάται από τη διεύθυνση προορισμού στον κόμβο και την αντίστοιχη μέθοδο εξυπηρέτησης που εφαρμόζεται. Κατά τις συναλλαγές που αναφέρονται σε φυσικές διευθύνσεις οι περιορισμοί επιβάλλονται από το υλικό του ελεγκτή IEEE 1394, καθώς πραγματοποιούνται αυτόνομα, χωρίς την επέμβαση του κεντρικού επεξεργαστή. Στην περίπτωση που εξετάστηκε, διαπιστώθηκε ότι ο ελεγκτής του κόμβου προορισμού έχει μια ουρά τεσσάρων αιτήσεων, ενώ η καθυστέρηση απόκρισης μεταβάλλεται ανάλογα με την υπομονάδα που εμπεριέχει την προσδιοριζόμενη διεύθυνση.

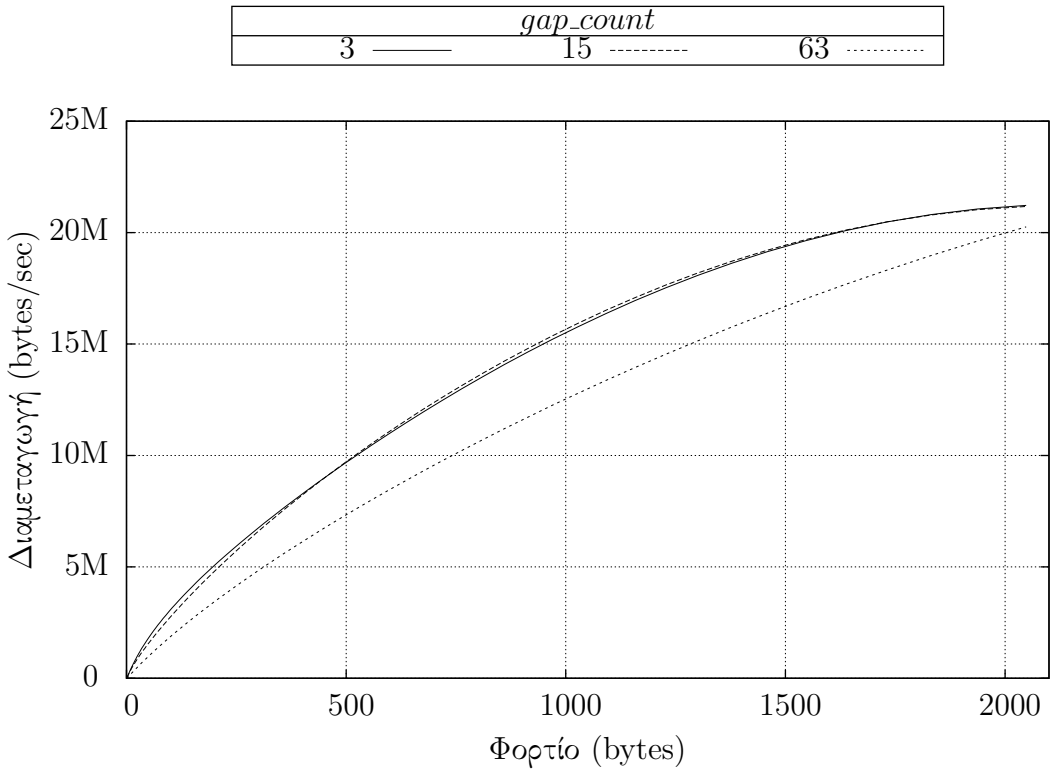
Όταν η εισερχόμενη αίτηση δεν αναφέρεται σε κάποια φυσική διεύθυνση, ο χειρισμός της μεταβιβάζεται στον κεντρικό επεξεργαστή. Το υποσύστημα ieee1394 δεν επιβάλλει κάποιον περιορισμό στο μέγεθος της ουράς αιτήσεων, ωστόσο η συσσώρευση ενός υπερβολικού αριθμού από αυτές οδηγεί στην αποτυχία συναλλαγών λόγω υπέρβασης των χρονικών ορίων για την ολοκλήρωσή τους. Για τις μετρήσεις που ακολουθούν χρησιμοποιήθηκαν οκτώ παράλληλες αιτήσεις, που δίνουν την καλύτερη δυνατή διαμεταγωγή σε μεσαία και μεγάλα μεγέθη πακέτων, χωρίς να οδηγούν σε υπερβολικούς χρόνους καθυστέρησης.

4.2.1.1 Ανάγνωση δεδομένων από την κύρια μνήμη απομακρυσμένου υπολογιστή

Για τη μέτρηση αυτή ήταν ιδιαίτερα χρήσιμο το γεγονός ότι το λογισμικό v1394 ουσιαστικά αδιαφορεί για το περιεχόμενο των πακέτων που διαχειρίζεται. Οποιαδήποτε συνεκτική περιοχή μνήμης με το σωστό μέγεθος μπορεί να θεωρηθεί ότι περιέχει ένα πλαίσιο κινούμενης εικόνας. Συνεπώς είναι δυνατή η πραγματοποίηση μετρήσεων κατά την ανάγνωση οποιασδήποτε περιοχής της κύριας μνήμης ενός απομακρυσμένου υπολογιστή, αν και δεν έχει νόημα η προβολή των πλαισίων που ανακτώνται, των οποίων τα περιεχόμενα είναι τυχαία. Μόνη προϋπόθεση για αυτή τη διαδικασία είναι η ενεργοποίηση στον απομακρυσμένο ελεγκτή IEEE 1394 του μηχανισμού απευθείας πρόσβασης μέσω DMA στις φυσικές διευθύνσεις του κόμβου (Physical DMA), κάτι που το υποσύστημα ieee1394 του Linux κάνει εξ ορισμού κατά την έναρξη της λειτουργίας του.

Για την πραγματοποίηση των πειραματικών δοκιμών επιλέξαμε τη διεύθυνση 0x000000100000 στον κόμβο B, η οποία ορίζει μια περιοχή που ξεκινά ακριβώς μετά το πρώτο MiB του εύρους φυσικών διευθύνσεων, περιοχή που βρίσκεται εντός της φυσικής μνήμης σε όλους τους σημερινούς υπολογιστές. Πολλαπλές επαναλήψεις των μετρήσεων αυτών με άλλες διευθύνσεις της κύριας μνήμης έδωσαν παρόμοια αποτελέσματα:

Σχήμα 47: Διαμεταγωγή κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου B



Πίνακας 55: Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	48.21	48.21	48.27	48.22	45.26	53.75
2	48.46	48.25	48.25	48.39	48.27	51.10
4	48.26	47.96	48.57	48.05	48.20	53.01
8	48.48	48.45	48.53	48.21	48.39	53.88
16	48.27	48.50	48.33	48.43	48.57	54.19
32	48.78	48.73	48.51	48.86	48.77	54.59
64	47.02	49.14	49.11	48.87	48.80	55.73
128	48.93	49.85	49.83	49.35	49.63	56.08
256	50.48	47.25	50.54	50.14	51.14	56.80
512	67.84	74.93	68.74	73.86	72.01	79.00
1024	100.33	101.38	101.16	101.35	101.54	102.61
2048	148.45	152.12	149.59	148.94	149.75	152.28

Πίνακας 56: Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	23375	23885	22922	20590	6770	5531
2	70670	70464	70106	64555	37461	21069
4	165527	165771	164081	152681	86629	64854
8	351301	336290	347236	329123	215514	151293
16	723632	722616	715943	681543	455383	322539
32	1461513	1457609	1439946	1373121	932156	670082
64	2767259	2907860	2844869	1908840	1851290	1340030
128	3892763	4229940	3830794	3750056	3628626	2610904
256	7152639	7082032	6946145	7097803	6536971	4919387
512	10774608	9375877	10764070	10021588	12513266	7919935
1024	20466457	20273900	20228464	20328066	19163130	14521227
2048	21485293	21220584	21355742	21172072	20988571	20259020

Παρατηρούμε ότι οι επιδόσεις του συστήματος σε αυτή τη δοκιμή είναι σημαντικά χαμηλότερες από τα θεωρητικά βέλτιστα της παραγράφου 2.7.1, με τη μέγιστη διαμεταγωγή να βρίσκεται στα 21.5 MB/sec. Το αίτιο είναι οι σημαντικά μεγαλύτεροι χρόνοι που απαιτούνται για την ολοκλήρωση μιας συναλλαγής,

τόσο λόγω της εκτέλεσης λογισμικού στο χώρο χρήστη, όσο και λόγω των ορίων των φυσικών εξαρτημάτων του συστήματος. Αυτή η εκτίμηση επιβεβαιώνεται από τη μεταβολή που προκύπτει όταν αλλάζει η τιμή gap_count. Διαπιστώνουμε ότι στο διάστημα 1-15 έχουμε αμελητέα μείωση στην απόδοση του διαύλου IEEE 1394, ενώ για υψηλότερες τιμές η πτώση στις επιδόσεις είναι σαφώς μικρότερη από την αναμενόμενη στο ιδανικό μοντέλο, κάτι που υποδεικνύει την ύπαρξη ενός σταθερού σημείου στένωσης.

Μπορούμε να λάβουμε μια ενδεικτική τιμή για την τάξη μεγέθους της καθυστέρησης που προκαλείται από το λογισμικό από τον πίνακα 55 για πολύ μικρά φορτία πακέτων, όπου διαπιστώνουμε μια αύξηση περίπου 45 μs σε σχέση με τους θεωρητικούς υπολογισμούς. Η διαφορά αυτή όμως δεν παραμένει σταθερή, αλλά αυξάνεται παράλληλα με το μέγεθος των πακέτων, κάτι που υποδηλώνει έλλειψη παραλληλισμού κατά την επεξεργασία των συναλλαγών, με τις διάφορες καθυστερήσεις να αθροίζονται αντί να επικαλύπτονται.

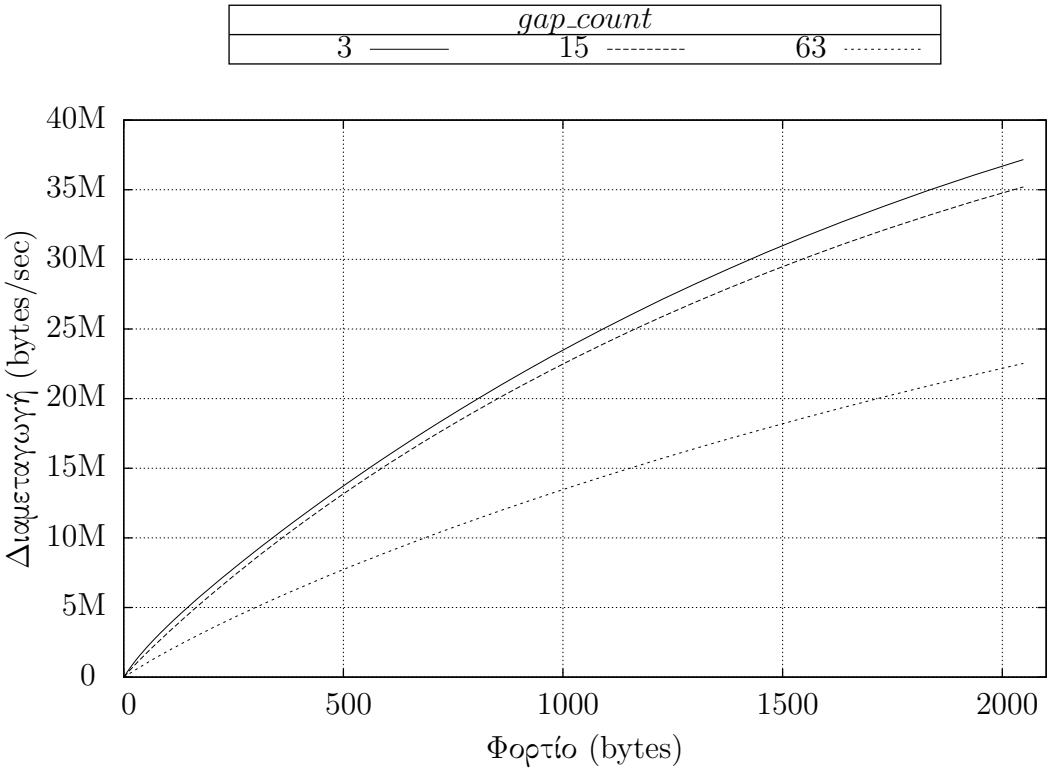
Οι συναλλαγές ανάγνωσης που πραγματοποιούνται σε αυτή την περίπτωση είναι πάντοτε διαχωρισμένες συναλλαγές και η επεξεργασία τους γίνεται από τη μονάδα φυσικών αποκρίσεων (Physical Response Unit) του απομακρυσμένου ελεγκτή IEEE 1394. Για την εξυπηρέτηση της κάθε συναλλαγής απαιτείται ο προγραμματισμός των μονάδων DMA του ελεγκτή και η ανάκτηση των ζητούμενων δεδομένων από την κύρια μνήμη του υπολογιστή μέσω του διαύλου PCI. Στην περίπτωση που ο απομακρυσμένος ελεγκτής δεν επιτρέπει την ταυτόχρονη εξυπηρέτηση πολλαπλών συναλλαγών πρόσβασης σε φυσικές διευθύνσεις δεν είναι δυνατός ο παραλληλισμός των προσβάσεων στο δίαυλο PCI με τη μετάδοση πακέτων IEEE 1394, με αποτέλεσμα να εισάγονται σημαντικοί χρόνοι αναμονής στη διαδικασία.

Προκειμένου να επιβεβαιώσουμε αυτή την εκτίμηση αντιστρέψαμε τη φορά των συναλλαγών, με τον κόμβο A να είναι πλέον ο κόμβος προορισμού. Οι επιδόσεις που παρατηρήθηκαν στις μετρήσεις διαμεταγωγής ήταν καθαρά αυξημένες, στην ίδια τάξη μεγέθους με τις θεωρητικά αναμενόμενες, ενώ και οι καθυστερήσεις ήταν βελτιωμένες στα μεγάλα μεγέθη πακέτων.

Επιπρόσθετες δοκιμές που πραγματοποιήθηκαν στον κόμβο A με μια εξωτερική μονάδα σκληρού δίσκου SBP-2 επιβεβαιώνουν ότι ο ελεγκτής IEEE 1394 που φέρει δεν αντιμετωπίζει κάποια δυσκολία κατά τη μετάδοση δεδομένων με ρυθμούς της τάξης των 40 MiB/sec. Κατά συνέπεια, συμπεραίνουμε ότι ο ελεγκτής του κόμβου B διαθέτει σαφώς υποδεέστερους μηχανισμούς για την εξυπηρέτηση εισερχόμενων συναλλαγών ανάγνωσης, τουλάχιστον στην περιοχή των φυσικών διευθύνσεων.

Ένα σημείο με ενδιαφέρον είναι ότι το μέγιστο των 37.2 MB/sec της διαμεταγωγής παρουσιάζεται όταν η μεταβλητή gap_count έχει τιμή 7. Αυτό υποδηλώνει ότι ο κόμβος δεν μπορεί να ανταπεξέλθει στον υψηλότερο ρυθμό άφιξης αιτήσεων όταν τα διαστήματα αδράνειας είναι μικρότερα, με αποτέλεσμα να χάνεται μέρος του διαθέσιμου εύρους ζώνης με την επανάληψη των αιτήσεων που απορρίπτονται.

Σχήμα 48: Διαμεταγωγή κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου Α



Πίνακας 57: Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου Α

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	50.70	50.85	55.42	56.36	56.62	61.24
2	50.63	51.01	54.90	56.06	56.25	61.15
4	50.48	51.61	55.59	55.66	55.57	61.16
8	51.02	51.17	55.71	56.38	55.87	61.13
16	51.18	51.02	55.76	56.20	55.84	61.26
32	51.57	51.74	56.13	56.26	56.70	61.25
64	51.35	51.33	56.91	57.21	57.13	61.36
128	53.26	53.39	58.29	58.18	59.95	61.69
256	55.77	60.05	60.09	60.35	61.17	83.35
512	62.57	65.20	64.62	66.06	66.59	88.27
1024	88.49	95.00	95.14	95.29	95.24	96.70
2048	118.61	123.05	123.40	123.21	123.12	128.98

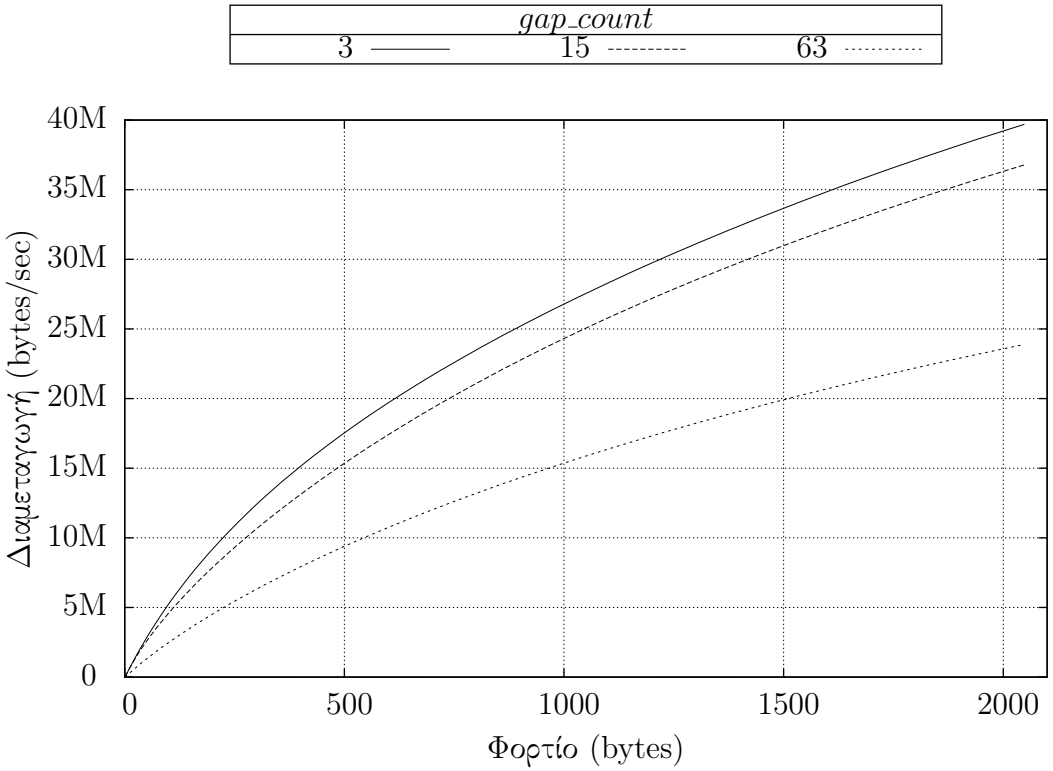
Πίνακας 58: Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από την κύρια μνήμη του κόμβου Α

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	24647	25628	21855	18108	8797	5296
2	76476	73298	67329	54010	40088	19393
4	178071	176244	158491	149189	107262	61767
8	368553	366932	334075	283062	230633	144375
16	755315	764305	683391	584113	476616	312768
32	1532517	1547777	1381912	1171307	1797723	646547
64	3055894	3017525	2729784	2273102	2248672	1299676
128	5593169	5463440	4621886	4027153	3735691	2539067
256	9445917	7379946	8374884	8038261	7089684	4834296
512	17085121	16122013	16017804	15577498	13519730	8700872
1024	28311985	27289352	26459426	26083411	22120505	15157670
2048	36553302	37167275	37247430	35204796	29482761	22526165

4.2.1.2 Εγγραφή δεδομένων στην κύρια μνήμη απομακρυσμένου υπολογιστή

Απαραίτητη προϋπόθεση κατά την εγγραφή δεδομένων στη φυσική μνήμη ενός απομακρυσμένου κόμβου είναι η περιοχή προορισμού να είναι δεσμευμένη και προετοιμασμένη για αυτή τη χρήση. Η ανεξέλεγκτη αλλοίωση των περιεχομένων της μνήμης μπορεί να οδηγήσει σε ποικίλα προβλήματα, όπως για παράδειγμα στην κατάρρευση του λειτουργικού συστήματος ενός υπολογιστή. Για αυτό το λόγο κατά την εκκίνηση του κόμβου B δόθηκε στον πυρήνα του Linux η παράμετρος mem=128M, η οποία εξασφαλίζει ότι μόνο τα πρώτα 128 MiB της φυσικής μνήμης θα χρησιμοποιηθούν από το λογισμικό. Στη συνέχεια επιλέχθηκε η διεύθυνση 0x00000c000000, που ορίζει μια περιοχή μνήμης που ξεκινά αμέσως μετά τα πρώτα 192 MiB, για την πραγματοποίηση ασύγχρονων συναλλαγών εγγραφής.

Σχήμα 49: Διαμεταγωγή κατά την εγγραφή δεδομένων στην κύρια μνήμη του κόμβου B



Πίνακας 59: Καθυστέρηση (μs) κατά την εγγραφή δεδομένων στην κύρια μνήμη του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	48.36	48.21	46.46	48.28	48.72	48.17
2	48.32	48.22	48.84	48.61	48.72	50.22
4	46.47	46.76	46.76	47.66	47.31	49.02
8	48.91	48.52	48.53	48.44	46.55	49.77
16	49.35	48.47	48.82	48.69	49.03	50.34
32	50.06	48.94	49.72	49.75	49.57	52.28
64	50.65	49.73	50.15	50.20	51.22	54.27
128	53.01	53.28	53.61	54.53	57.05	54.78
256	57.67	58.62	57.56	58.21	60.26	61.81
512	68.34	68.44	67.62	68.14	68.30	76.68
1024	85.69	85.52	85.72	86.15	88.53	89.96
2048	115.65	117.06	115.57	116.53	120.95	125.20

Πίνακας 60: Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων στην κύρια μνήμη του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	37201	37084	33011	37921	19413	7244
2	102785	102357	86246	102494	61863	34870
4	173575	170296	171173	172046	163675	76198
8	469872	477676	476512	475789	278088	208266
16	818775	970666	962094	963064	539309	437817
32	1805675	1886172	1947562	1949742	1309455	889429
64	3641484	3892491	3702724	3779743	2443974	1784693
128	6877087	7660167	7633648	6815270	4535232	3473837
256	14800879	14626200	13320932	10290123	8753290	6468902
512	19242256	19938943	20002943	18637147	15876898	11103635
1024	29310475	29631348	28993462	26974399	23432991	17227627
2048	40513184	39696394	39166488	36781704	30549404	23891171

Το υποσύστημα ieee1394 ενεργοποιεί εξ ορισμού τη δυνατότητα ενοποιημένων συναλλαγών εγγραφής (posted writes) στη περιοχή φυσικών διευθύνσεων του κόμβου, με σκοπό τη βελτίωση των επιδόσεων. Αυτό επιτρέπει τον παραλληλισμό της εξυπηρέτησης των συναλλαγών, με ένα πακέτο ACK με κωδικό ack_complete

να αποστέλλεται πριν ακόμα ολοκληρωθεί η εγγραφή των δεδομένων στη διεύθυνση προορισμού. Αυτό μπορεί να έχει ως συνέπεια την αποσιώπηση σφαλμάτων, αλλά προσφέρει τη βέλτιστη δυνατή απόδοση.

Αν και οι ενοποιημένες συναλλαγές αποκρύπτουν τυχόν σφάλματα από τον κόμβο προέλευσης, στον κόμβο προορισμού εντοπίζονται κανονικά. Κατά την προδιαγραφή Open HCI, όταν ο ελεγκτής είναι ρυθμισμένος με αυτόν τον τρόπο ενημερώνει τον κεντρικό επεξεργαστή με διακοπή μόνο σε περίπτωση σφάλματος. Ο πυρήνας του Linux εξάγει στο αρχείο /proc/interrupts των αριθμό διακοπών που έλαβε από κάθε συσκευή, οπότε με επίβλεψη του αρχείου αυτού είναι δυνατόν να διαπιστωθεί αν ο αριθμός των λαθών που εντοπίζει ο ελεγκτής IEEE 1394 βρίσκεται μέσα στα αποδεκτά όρια.

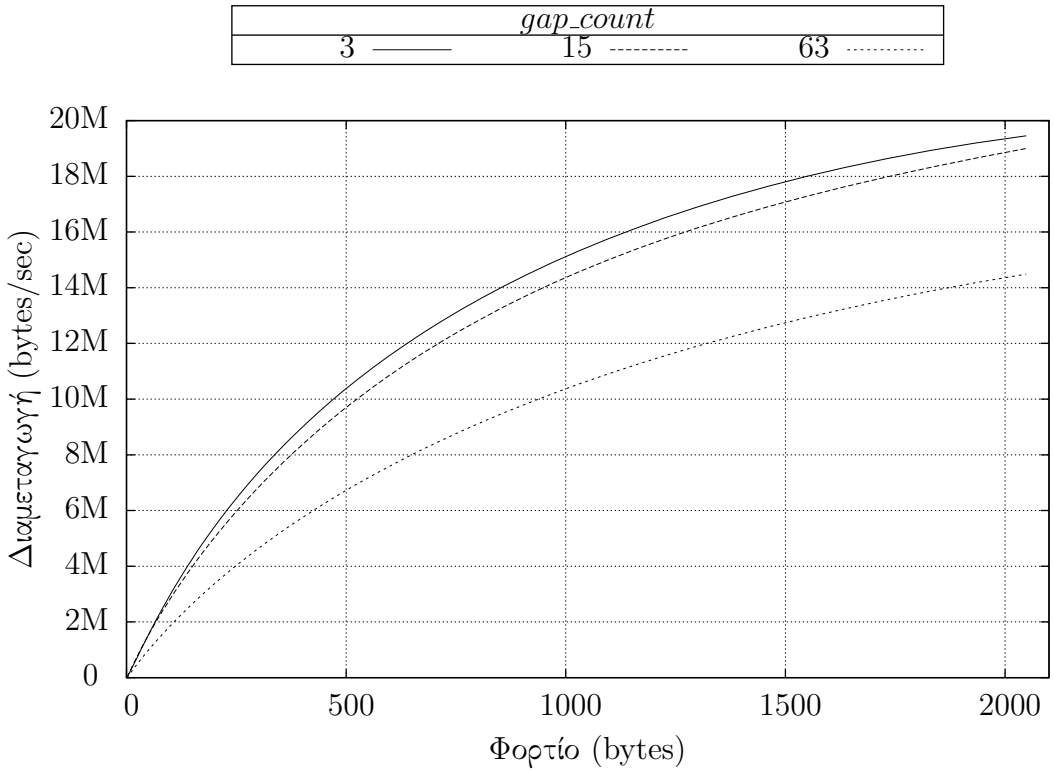
Με βάση αυτά τα δεδομένα, παρατηρούμε ότι οι μετρήσεις που πραγματοποιήθηκαν έδωσαν αποτελέσματα κοντά στα αναμενόμενα, με τη μέγιστη διαμεταγωγή να ανέρχεται στα 40.5 MB/sec. Η τιμή αυτή είναι κατά 7% μικρότερη από την μάλλον αισιόδοξη πρόβλεψη των 43.6 MB/s της παραγράφου 2.7.1, αλλά δεν μπορεί να θεωρηθεί ότι αποκλίνει σημαντικά.

Σύμφωνα με τα τεχνικά χαρακτηριστικά του ελεγκτή VIA VT6306, το εν λόγω ολοκληρωμένο κύκλωμα διαθέτει μια ουρά τριών θέσεων για την εξυπηρέτηση συναλλαγών εγγραφής αυτού του τύπου. Διαπιστώσαμε ότι για την μεγιστοποίηση της διαμεταγωγής ήταν απαραίτητος ο παραλληλισμός τεσσάρων συναλλαγών από το λογισμικό v1394. Η τιμή αυτή εξασφαλίζει τον υψηλό βαθμό πλήρωσης της ουράς, αλλά μπορεί περιοδικά να οδηγεί σε απόρριψη κάποιων αιτήσεων, που σε συνδυασμό με τις καθυστερήσεις που εισάγονται στη συνολική λειτουργία του συστήματος εξηγεί τη μειωμένη απόδοση σε σχέση με τις θεωρητικές προβλέψεις.

4.2.1.3 Ανάγνωση δεδομένων από απομακρυσμένο ενταμιευτή

Όπως αναφέρθηκε στην παράγραφο 3.4, το υποσύστημα IEEE 1394 του Linux παρέχει τη δυνατότητα αντιστοίχισης ενός συνεκτικού ενταμιευτή από τη φυσική μνήμη του υπολογιστή σε ένα εύρος διευθύνσεων του διαύλου (address range mapping - ARM). Κάθε συναλλαγή σε αυτή την περιοχή διευθύνσεων παράγει μια διακοπή προς τον κεντρικό επεξεργαστή και ο πυρήνας του λειτουργικού συστήματος αναλαμβάνει το χειρισμό της. Η διαδικασία αυτή είναι κατανοητά πιο αργή από το μηχανισμό Physical DMA, ο οποίος είναι υλοποιημένος στο υλικό του ελεγκτή IEEE 1394, οπότε είναι πιθανές αυξημένες καθυστερήσεις και μειωμένες τιμές για τη διαμεταγωγή. Η διεύθυνση προορισμού που επιλέξαμε είναι η 0xffffe0000000, που είναι η αρχή του ιδιωτικού χώρου διευθύνσεων (private space), σύμφωνα με την αρχιτεκτονική CSR.

Σχήμα 50: Διαμεταγωγή κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου B



Πίνακας 61: Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	66.26	65.91	66.21	66.70	67.88	77.89
2	67.02	67.48	66.82	66.05	68.80	77.68
4	56.86	56.94	56.85	58.34	59.95	64.15
8	66.87	67.74	65.58	67.68	68.45	77.56
16	68.37	67.99	66.61	68.77	70.00	81.35
32	68.91	69.48	68.66	72.39	70.67	80.73
64	78.42	78.51	79.00	79.13	78.67	83.36
128	81.40	82.06	81.69	81.78	80.32	84.82
256	86.81	87.67	86.04	86.86	87.77	89.41
512	103.10	99.69	99.07	99.78	101.05	104.33
1024	126.49	127.02	126.27	126.66	129.50	132.41
2048	179.83	178.19	180.72	181.01	181.49	183.89

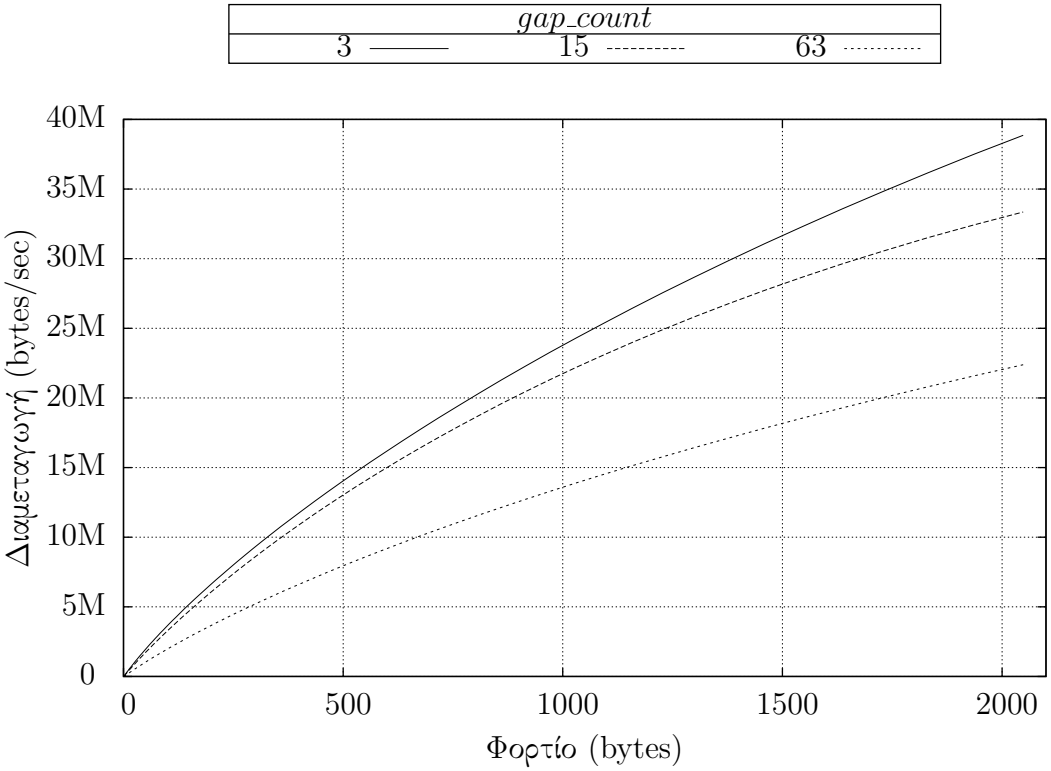
Πίνακας 62: Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	3733	3977	3786	6398	3593	2724
2	33731	34511	34057	41771	31554	17533
4	92932	96053	95953	104744	91919	59761
8	216008	218223	203629	245428	206028	149835
16	431474	420724	468328	513760	438132	323074
32	917926	942926	963822	1022999	891578	672527
64	1913303	1945955	2051516	2139155	1741005	1348244
128	4467959	4554227	4646919	4296873	2928830	2624684
256	9118150	8640621	8371491	6833327	6632610	4960339
512	12748489	12787001	13115178	13011050	10560856	8332195
1024	17629928	17331934	16966160	15954703	14539938	11836136
2048	19591402	19451774	19332338	19000090	17835161	14489667

Συναντούμε και πάλι ιδιαίτερα χαμηλές επιδόσεις, με μέγιστο ρυθμό μεταφοράς δεδομένων 19.6 MB/sec. Συγκριτικά με τις συναλλαγές ανάγνωσης από τη φυσική μνήμη έχουμε διαμεταγωγή μειωμένη κατά 9% και καθυστερήσεις αυξημένες από 12 έως και 31 μs περίπου. Έχοντας ήδη διαπιστώσει μια αδυναμία του ελεγκτή

VIA VT6306 κατά την αποστολή δεδομένων στην παράγραφο 4.2.1.1, εξετάζουμε τη συμπεριφορά του διαύλου αν οι συναλλαγές πραγματοποιούνται από τον κόμβο B, με τον ενταμιευτή να βρίσκεται στον κόμβο A.

Σχήμα 51: Διαμεταγωγή κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου A



Πίνακας 63: Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου A

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	85.65	87.64	89.79	89.50	89.43	91.22
2	84.23	86.70	88.74	86.65	89.26	87.73
4	82.90	83.57	86.99	86.82	87.54	89.67
8	84.69	88.14	87.57	87.87	90.40	91.15
16	86.72	87.28	89.00	87.06	90.81	90.57
32	88.18	89.94	89.10	88.37	91.46	91.25
64	86.97	89.81	92.13	88.29	93.34	95.23
128	90.59	96.17	94.09	93.03	95.49	99.47
256	97.51	97.17	98.49	102.26	100.53	104.05
512	112.30	109.23	112.51	111.69	112.24	116.27
1024	134.56	134.96	134.84	135.57	135.29	137.33
2048	174.24	174.99	175.40	175.70	176.02	179.17

Πίνακας 64: Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από ενταμιευτή ARM του κόμβου A

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	19601	18113	12995	11251	6949	2837
2	61873	60448	45777	50434	40346	18079
4	162407	164018	136790	124242	106563	64104
8	320375	330771	293727	281398	238130	154360
16	679665	660074	608196	577433	488245	332475
32	1359569	1357058	1236530	1172386	1063868	689755
64	2642060	2660818	2478023	2306826	2158829	1381717
128	5262386	5186655	4943253	4449918	4225730	2708356
256	9035452	9080469	8950585	8453939	8274492	5117570
512	16156228	16139396	16144301	15335133	13759130	9176147
1024	25828696	26535680	26378567	25205761	22089782	15130288
2048	35519567	38863980	35835703	33351538	29755991	22388926

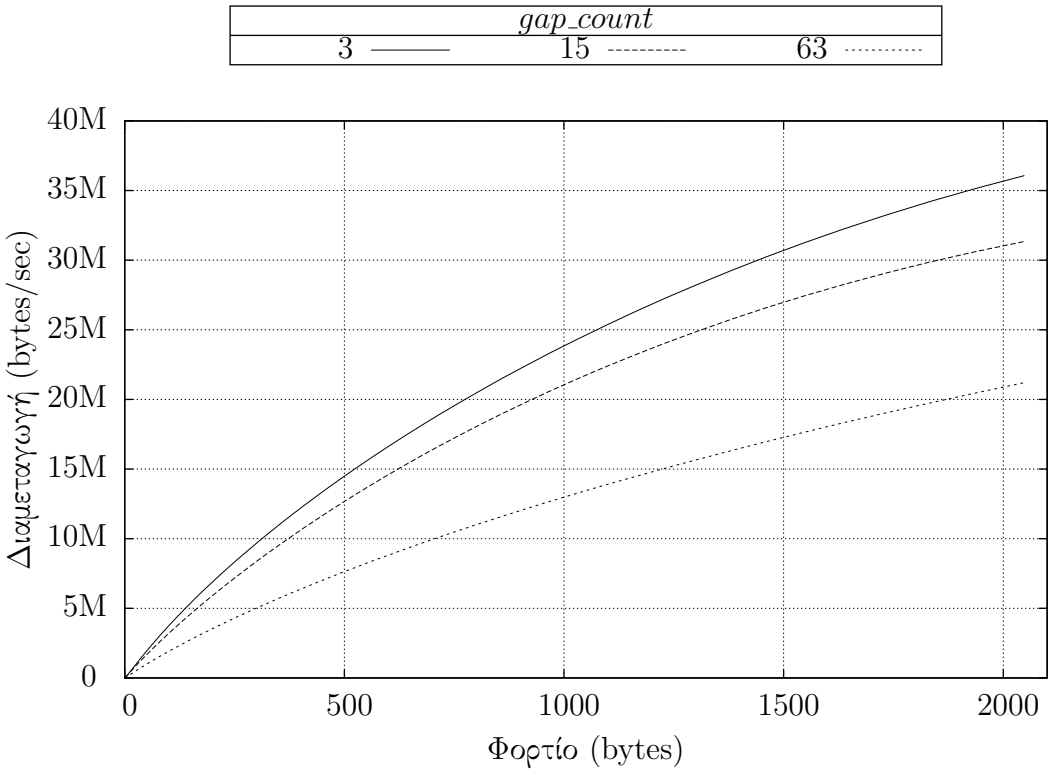
Από τις μετρήσεις αυτές επιβεβαιώνεται η αδυναμία του κόμβου B να μεταδόσει δεδομένα με υψηλότερες ταχύτητες. Παρατηρούμε ότι εδώ η μέγιστη διαμεταγωγή έχει βελτιωθεί σε σχέση με την ανάγνωση δεδομένων από τη φυσική μνήμη του υπολογιστή και φτάνει τα 38.9 MB/sec. Από την άλλη πλευρά, ο χρόνος ολο-

κλήρωσης των συναλλαγών αυξάνεται δραστικά, από 26 έως και 56 μs περίπου, αν και εξακολουθεί να είναι καλύτερος από τους χρόνους του κόμβου B στην ίδια δοκιμή. Συμπεραίνουμε ότι ο ελεγκτής Texas Instruments OHCI-Lynx που φέρει ο κόμβος A διαθέτει ουρές με μεγαλύτερο βάθος και βελτιωμένες μηχανές DMA για την εξυπηρέτηση αυτού του τύπου συναλλαγών. Οι μεγαλύτερες ουρές επιτρέπουν τη διατήρηση υψηλών τιμών για τη διαμεταγωγή, με αντίτιμο τις υψηλότερες καθυστερήσεις απόκρισης που προκύπτουν, κάτι που συνδέεται και με τη μεγιστοποίηση της απόδοσης σε ελαφρώς μεγαλύτερες τιμές της μεταβλητής gap_count που παρατηρείται και εδώ.

4.2.1.4 Εγγραφή δεδομένων σε απομακρυσμένο ενταμιευτή

Για τη μέτρηση αυτή χρησιμοποιήθηκε ένας ενταμιευτής ARM στη διεύθυνση 0xffffe8000000, που αντιστοιχεί στο μέσο του ιδιωτικού χώρου διευθύνσεων. Οι συναλλαγές εγγραφής που πραγματοποιούνται σε αυτή την περιοχή δεν μπορούν πλέον να λάβουν τη μορφή ενοποιημένων συναλλαγών, καθώς οι αποκρίσεις παράγονται από τον κεντρικό επεξεργαστή. Κατά συνέπεια οι επιδόσεις που αναμένουμε είναι μειωμένες σε σχέση με αυτές που είδαμε στις προηγούμενες παραγράφους.

Σχήμα 52: Διαμεταγωγή κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου B



Πίνακας 65: Καθυστέρηση (μs) κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	63.75	64.53	65.15	66.33	66.76	71.70
2	64.15	64.98	64.08	65.11	66.58	70.53
4	63.60	63.76	64.20	65.10	64.72	69.94
8	63.51	64.50	65.24	65.74	67.21	71.10
16	64.78	70.93	65.46	67.46	68.89	70.60
32	65.42	66.21	68.72	69.06	67.67	74.06
64	71.18	70.51	72.49	71.90	73.15	79.26
128	78.03	79.15	80.27	82.28	83.86	92.43
256	88.70	87.59	88.63	88.61	90.91	95.99
512	98.96	101.96	103.14	101.34	100.73	110.00
1024	138.27	126.39	125.50	138.57	140.67	139.54
2048	179.54	177.08	154.65	181.52	182.29	184.19

Πίνακας 66: Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου B

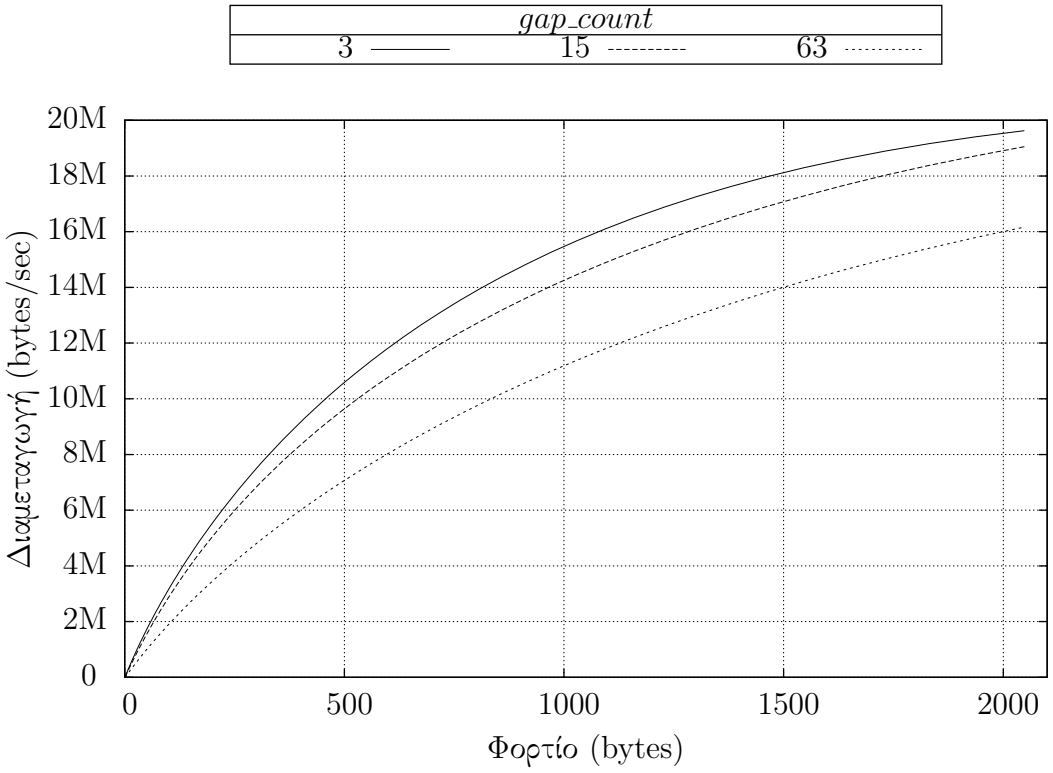
	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	11584	12612	11293	7908	3790	2710
2	53774	52823	55222	46171	35347	17070
4	97577	93656	98016	97463	90937	60911
8	296855	294586	299001	265828	221783	148906
16	618612	587329	629079	556589	468511	323383
32	1266600	1160819	1292600	1144329	959376	668910
64	2543854	2772945	2541428	2255918	1868977	1323414
128	5071166	5244117	5086858	4399881	3490870	2629446
256	10116424	9960930	9392672	8112547	6776280	4965934
512	16673393	16552899	16004741	14868699	13153027	8841216
1024	23571577	27891271	26328251	24994368	21480461	14483279
2048	25640440	36072463	30902563	31339313	28488016	21202683

Πράγματι, η καθυστέρηση εμφανίζεται από 11 έως και 37 μs μεγαλύτερη συγκριτικά με τις ενοποιημένες συναλλαγές εγγραφής, ενώ η μέγιστη διαμεταγωγή έχει πέσει από τα 40.5 MB/sec στα 36.1 MB/sec και επιτυγχάνεται και εδώ για gap_count = 7, υποδηλώνοντας αδυναμία του συστήματος να ανταπεξέλθει σε

υψηλότερους ρυθμούς αιτήσεων. Για την τιμή αυτή του gap_count, έχουμε μέγιστη θεωρητική διαμεταγωγή τα 41.8 MB/sec, τιμή που πέφτει στα 37.9 MB/sec σε ένα πραγματικό σύστημα με καθυστέρηση ανά υποδράση 2 μs μεγαλύτερη από τη βέλτιστη. Διαπιστώνουμε, συνεπώς, ότι η μετρούμενη διαμεταγωγή είναι αισθητά μικρότερη από το αναμενόμενο, κάτι που οφείλεται στο βαθμό χρήσης του κεντρικού επεξεργαστή του κόμβου B κατά την εξυπηρέτηση των συναλλαγών, ο οποίος στην περίπτωση αυτή φτάνει το 100% και αποτελεί το σημείο στένωσης του συστήματος.

Για το σχηματισμό μιας πλήρους εικόνας, αντιστράφηκαν και εδώ οι ρόλοι των δυο κόμβων. Συναντήσαμε και πάλι ένα άνω φράγμα στα 20 MB/sec περίπου, όπως και σε όλες τις άλλες περιπτώσεις μετάδοσης δεδομένων από τον κόμβο B προς τον κόμβο A που είδαμε στις προηγούμενες παραγράφους.

Σχήμα 53: Διαμεταγωγή κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου A



Πίνακας 67: Καθυστέρηση (μs) κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου A

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	88.43	88.54	83.70	88.94	88.41	93.19
2	90.94	87.80	86.84	89.11	90.47	94.51
4	86.05	85.33	86.55	87.11	87.92	87.20
8	89.75	88.89	90.02	89.38	89.21	90.60
16	90.03	91.99	92.22	89.81	91.80	90.50
32	92.67	93.19	92.14	90.49	90.99	91.05
64	89.66	93.84	90.55	93.36	95.45	97.20
128	96.88	94.65	98.25	97.43	98.45	99.27
256	100.09	104.04	103.33	105.49	105.60	107.26
512	118.56	118.23	114.22	119.45	122.70	124.37
1024	145.33	140.61	145.00	145.66	146.09	147.44
2048	199.10	199.01	199.58	199.45	199.43	201.02

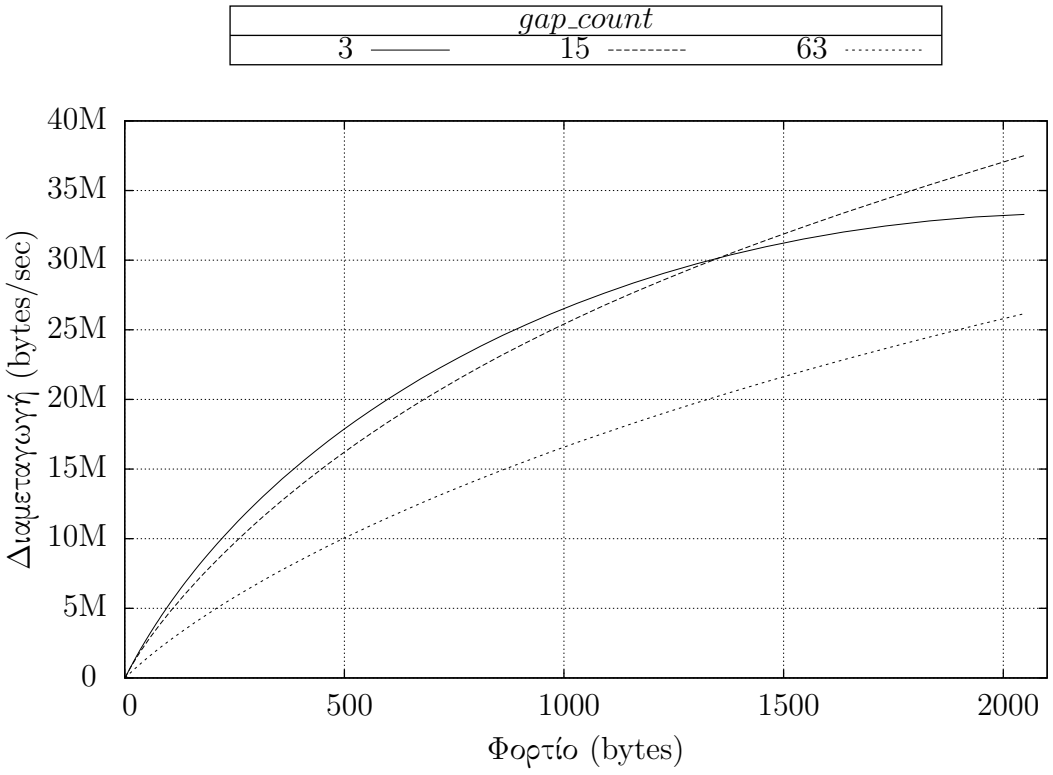
Πίνακας 68: Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων σε ενταμιευτή ARM του κόμβου A

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	13173	12368	11497	9815	6624	2829
2	38848	51874	45229	39198	39010	18001
4	161519	163452	135319	124469	109300	63825
8	294040	287077	280978	261686	237307	152220
16	528807	601354	528818	507490	489917	325146
32	1162490	1176179	1174801	1025129	1008880	679612
64	2350586	2400528	2291177	2154646	1968527	1343493
128	4424076	4556260	4193164	4147935	3707714	2638999
256	8189705	8065637	7771523	7745925	6490704	5052821
512	13413450	13666663	12886024	11858598	10734082	8578677
1024	18083176	17741890	17138295	16168293	15307555	12844485
2048	19789767	19628812	19419745	19050455	18219126	16161609

4.2.1.5 Εγγραφή δεδομένων στο μέσο χώρο διευθύνσεων απομακρυσμένου υπολογιστή

Με βάση την προδιαγραφή Open HCI όλες οι συναλλαγές εγγραφής που πραγματοποιούνται στο μέσο χώρο διευθύνσεων ενός απομακρυσμένου υπολογιστή εκτελούνται ως ενοποιημένες συναλλαγές, παρέχοντας έτσι καλύτερες επιδόσεις. Προκειμένου να δοκιμαστεί αυτό το ιδιαίτερο χαρακτηριστικό χρησιμοποιήθηκε μια περιοχή μνήμης ARM στη διεύθυνση 0x000100000000, η οποία βρίσκεται στον προαναφερθέντα χώρο.

Σχήμα 54: Διαμεταγωγή κατά την εγγραφή δεδομένων στο μέσο χώρο διευθύνσεων του κόμβου B



Πίνακας 69: Καθυστέρηση (μs) κατά την εγγραφή δεδομένων στο μέσο χώρο διευθύνσεων του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	48.73	48.67	48.39	48.62	48.93	49.52
2	48.61	48.07	45.03	48.91	49.13	49.91
4	46.80	47.31	47.30	47.75	47.60	47.99
8	48.49	48.27	48.60	48.94	48.91	48.95
16	49.03	48.91	48.90	48.93	49.32	50.16
32	49.30	49.46	49.39	49.67	49.74	51.45
64	49.87	50.06	49.79	50.16	50.68	52.57
128	54.25	53.89	54.26	53.96	54.96	56.15
256	57.65	57.64	57.68	58.28	58.70	60.25
512	67.66	67.87	68.62	68.71	69.71	70.14
1024	86.51	85.37	87.96	85.83	85.68	86.20
2048	115.11	115.13	114.91	115.63	116.50	118.29

Πίνακας 70: Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων στο μέσο χώρο διευθύνσεων του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	36394	34727	36583	35231	16812	3976
2	97383	96737	97635	90487	60218	34369
4	170883	171530	170480	169585	147616	96301
8	323942	479896	465281	474459	318644	217773
16	952757	967982	967910	959918	662078	465351
32	1737725	1918840	1954913	1878281	1348291	953917
64	3880303	3733782	3915841	3561696	2622025	1888037
128	7379654	7621780	7571188	6695663	5431672	3700774
256	14165674	14128712	12599620	10622770	10642570	6844888
512	21887962	21383155	19901231	21074773	17869452	11839539
1024	30910878	32060798	34080913	27722237	25429010	18463534
2048	31498438	33287118	40217082	37511652	32990064	26163126

Οι βέλτιστες τιμές για τα χαρακτηριστικά μεγέθη που μετρήθηκαν είναι αντίστοιχες με αυτές της παραγράφου 4.2.1.2 για τις συναλλαγές εγγραφής στη φυσική μνήμη του κόμβου. Αυτό το αποτέλεσμα ήταν αναμενόμενο, καθώς η χρήση ενοποιημένων συναλλαγών επιτρέπει τον παραλληλισμό της επεξεργασίας τους,

βελτιώνοντας τις επιδόσεις.

Ο ελεγκτής IEEE 1394 προκαλεί μια διακοπή στον κεντρικό επεξεργαστή για κάθε συναλλαγή σε κάθε περιοχή μνήμης ARM, αν και στην περίπτωση του μέσου χώρου διευθύνσεων αποστέλλει άμεσα την απόκριση σε κάθε συναλλαγή εγγραφής. Αυτή η λειτουργία, σε συνδυασμό με τον πιο σύνθετο χειρισμό του μηχανισμού DMA που απαιτείται για την πρόσβαση στους ενταμιευτές ARM, αυξάνει σε κάποιο βαθμό την πολυπλοκότητα της εξυπηρέτησης μιας συναλλαγής. Αυτό έγινε αισθητό με μια μετατόπιση των υψηλότερων τιμών της διαμεταγωγής για μεγάλα φορτία πακέτων προς μεγαλύτερες τιμές της μεταβλητής gap_count. Με αυτό τον τρόπο εξασφαλίζεται ένας ελαφρώς μικρότερος ρυθμός αποστολής αιτήσεων, αποτρέποντας την απόρριψη και επανάληψή τους.

4.2.1.6 Ανάγνωση δεδομένων από τη μνήμη γραφικών απομακρυσμένου κόμβου

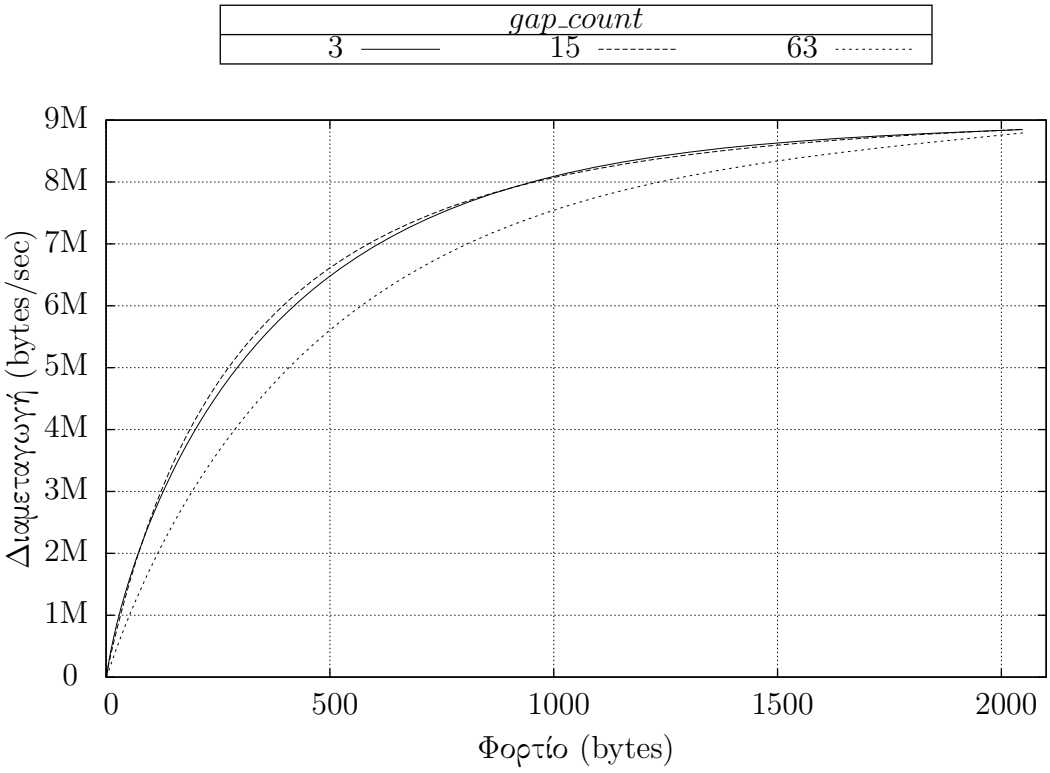
Για τη λειτουργία της κάρτας γραφικών ενός υπολογιστή είναι απαραίτητη η ύπαρξη μιας περιοχής μνήμης για την αποθήκευση τόσο του πλαισίου που εμφανίζεται στην οθόνη την τρέχουσα χρονική στιγμή, όσο και και την υποστήριξη των υπόλοιπων λειτουργιών της. Αν και υπάρχουν συστήματα που χρησιμοποιούν την κεντρική μνήμη του υπολογιστή, η συντριπτική πλειοψηφία των καρτών που έχουν κατασκευαστεί περιλαμβάνει αφοσιωμένα ολοκληρωμένα κυκλώματα μνήμης RAM, τουλάχιστον για την απεικόνιση του περιεχομένου της οθόνης.

Οι περισσότερες κάρτες γραφικών εκθέτουν τη μνήμη που διαθέτουν σε κάποιο εύρος διευθύνσεων του διαύλου συστήματος του υπολογιστή, επιτρέποντας στο λογισμικό να τις τροφοδοτήσει άμεσα με τη μέγιστη δυνατή ταχύτητα. Παράλληλα καθίσταται εφικτή η απευθείας μεταφορά δεδομένων από άλλες συσκευές του διαύλου συστήματος προς την κάρτα γραφικών. Συνήθως αυτή η δυνατότητα αξιοποιείται από συσκευές σύλληψης και ψηφιοποίησης κινούμενης εικόνας για την προβολή του εισερχόμενου σήματος στην οθόνη του υπολογιστή χωρίς να δεσμεύονται πόροι του κεντρικού επεξεργαστή. Ωστόσο η ίδια μέθοδος μπορεί να χρησιμοποιηθεί από έναν ελεγκτή IEEE 1394 για την πρόσβαση στη μνήμη γραφικών, όπως και σε κάθε άλλη περιοχή μνήμης που είναι προσβάσιμη από το χώρο διευθύνσεων του διαύλου.

Ο σχεδιασμός της κάρτας γραφικών που χρησιμοποιείται επηρεάζει δραστικά την αλληλεπίδρασή της με τις άλλες συσκευές. Το μέγιστο μέγεθος συναλλαγής PCI που υποστηρίζει συνδέεται ευθέως με το φορτίο των ασύγχρονων συναλλαγών που μπορεί να χρησιμοποιηθεί, ενώ ο τρόπος οργάνωσης της μνήμης που φέρει μπορεί να είναι ιδιαίτερα σύνθετος, καθώς υπάρχουν περιπτώσεις όπου δεν χρησιμοποιείται απλά ένας γραμμικός συνεκτικός ενταμιευτής για την αποθήκευση του τρέχοντος πλαισίου.

Κατά την πραγματοποίηση αυτής της δοκιμής διαπιστώθηκε ότι σε κάποιες περιπτώσεις απαιτείται η χρήση ειδικού λογισμικού ελέγχου έτσι ώστε η κάρτα γραφικών να εκθέσει τον ενταμιευτή του τρέχοντος πλαισίου (frame buffer) στο εύρος διευθύνσεων του διαύλου συστήματος. Το λειτουργικό σύστημα Linux, ωστόσο, διαθέτει τους απαραίτητους οδηγούς για τις περισσότερες κάρτες γραφικών, οπότε αυτή η απαίτηση δεν δημιούργησε κάποιο ιδιαίτερο πρόβλημα.

Σχήμα 55: Διαμεταγωγή κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου B



Πίνακας 71: Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	48.53	45.36	48.38	48.45	48.27	53.47
2	48.32	50.93	48.22	48.23	48.30	53.74
4	48.19	48.61	47.64	48.10	48.25	52.60
8	48.48	48.49	48.46	48.63	48.43	53.93
16	49.04	49.11	48.51	48.35	48.31	54.32
32	48.76	48.71	48.70	48.50	45.79	54.73
64	48.44	48.74	48.84	48.34	49.10	56.27
128	49.20	49.68	49.15	49.82	50.32	58.76
256	84.15	84.24	84.69	84.40	84.75	86.42
512	112.14	112.35	112.84	112.62	113.41	118.88
1024	171.01	171.44	171.73	171.75	171.45	173.31
2048	284.80	288.57	287.94	289.76	286.66	287.94

Πίνακας 72: Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου B

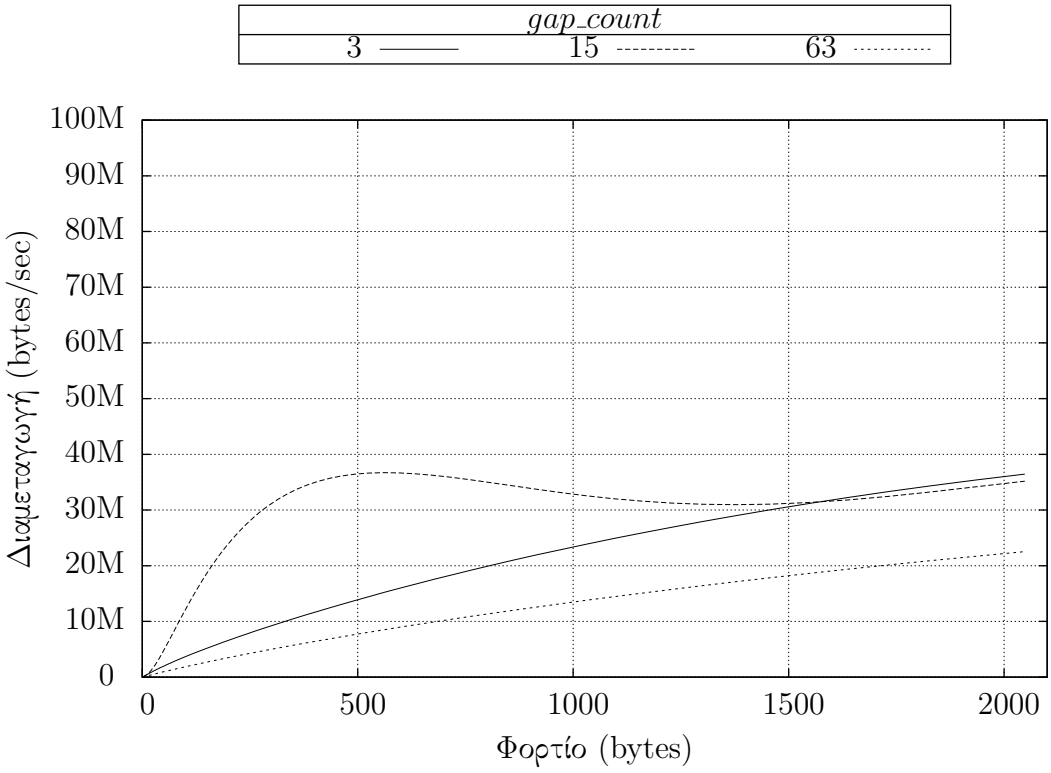
	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	23387	23678	23341	20729	7533	5516
2	70130	62065	70122	64466	37062	21128
4	163838	162774	136533	150087	97775	66286
8	351661	345246	346475	325989	214683	151974
16	714236	711210	710106	597611	455958	325214
32	1446540	1433020	1405923	947870	930180	672372
64	1909195	1910030	1914938	1904607	1671986	1340005
128	4688518	3712501	4689578	4440344	3387866	2445792
256	6874009	6440710	6713582	6900240	6419473	4928637
512	8203004	8210639	8197279	7909856	7898640	7714174
1024	8564980	8574330	8550932	8527185	8542341	8049691
2048	8792545	8846315	8808403	8850618	8790824	8792560

Κατά την πρώτη φάση χρησιμοποιήθηκαν ασύγχρονες συναλλαγές ανάγνωσης για την πρόσβαση στη μνήμη γραφικών του κόμβου B, η οποία βρισκόταν στη διεύθυνση 0xd8000000. Οι ταχύτητες ανάγνωσης που επιτεύχθηκαν ήταν κατά πολύ υποδεέστερες από τις αναμενόμενες, με τη μέγιστη διαμεταγωγή να φτάνει τα 8.5 MB/sec. Αυτή η τιμή είναι πολύ χαμηλή, ακόμα και αν ληφθεί υπόψη το άνω φράγμα των 20 MB/sec που φαίνεται να χαρακτηρίζει τον κόμβο B κατά την αποστολή δεδομένων.

Σημειώνεται ότι οι κάρτες γραφικών αποτελούν την κύρια μονάδα εξόδου ενός υπολογιστή και κατά συνέπεια έχουν σχεδιαστεί με κύριο προσανατολισμό τη μεταφορά δεδομένων προς αυτές και όχι αντίστροφα, με ορισμένα μοντέλα να μην επιτρέπουν την ανάγνωση της μνήμης τους. Αυτή η διαφοροποίηση είναι συχνά πιο αισθητή στις παλαιότερες μονάδες, όπως η κάρτα με το ολοκληρωμένο κύκλωμα S3 Virge με την οποία δοκιμάστηκε ο κόμβος B. Σε αυτές τις περιπτώσεις η αμφίδρομη επικοινωνία με τον επεξεργαστή είναι περιορισμένη ακόμα και κατά τις πιο σύνθετες λειτουργίες τους, με αποτέλεσμα η δυνατότητα για ταχεία αποστολή δεδομένων στο δίαυλο συστήματος να μην είναι απαραίτητη.

Όπως είναι εμφανές από τις τεχνικές προδιαγραφές, ο κόμβος A διαθέτει μια κάρτα γραφικών σαφώς ανώτερη από τον κόμβο B, με τη μνήμη της στη διεύθυνση 0xd0000000. Για το λόγο αυτό κρίθηκε απαραίτητη η επανάληψη των δοκιμών με το λογισμικό v1394 να εκτελείται στον κόμβο B. Οι επιδόσεις που επιτεύχθηκαν ήταν αντίστοιχες με αυτές που παρατηρήσαμε κατά την ανάγνωση δεδομένων από την κύρια μνήμη ενός κόμβου στην παράγραφο 4.2.1.1, κάτι που οφείλεται τόσο στην επίδραση του ταχύτερου διαύλου AGP, με τον οποίο συνδέεται η κάρτα γραφικών, όσο και στη νεότερη αρχιτεκτονική της ίδιας της κάρτας.

Σχήμα 56: Διαμεταγωγή κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου Α



Πίνακας 73: Καθυστέρηση (μs) κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου Α

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	55.72	56.54	56.27	56.18	56.57	61.28
2	55.94	56.10	55.98	56.34	56.09	61.28
4	55.42	55.56	56.14	55.68	55.43	61.02
8	55.49	55.39	55.94	56.10	56.74	61.25
16	56.25	56.13	55.88	56.14	55.95	61.31
32	56.52	56.56	56.17	57.20	56.11	61.25
64	56.79	56.97	57.24	57.18	57.14	61.34
128	57.47	57.80	57.53	57.73	59.62	61.69
256	60.08	59.67	60.46	61.55	61.51	82.33
512	64.63	65.32	65.57	65.71	70.68	87.91
1024	95.44	95.41	95.92	94.99	94.79	97.21
2048	124.00	125.03	124.38	124.64	124.68	129.68

Πίνακας 74: Διαμεταγωγή (bytes/sec) κατά την ανάγνωση δεδομένων από τη μνήμη γραφικών του κόμβου Α

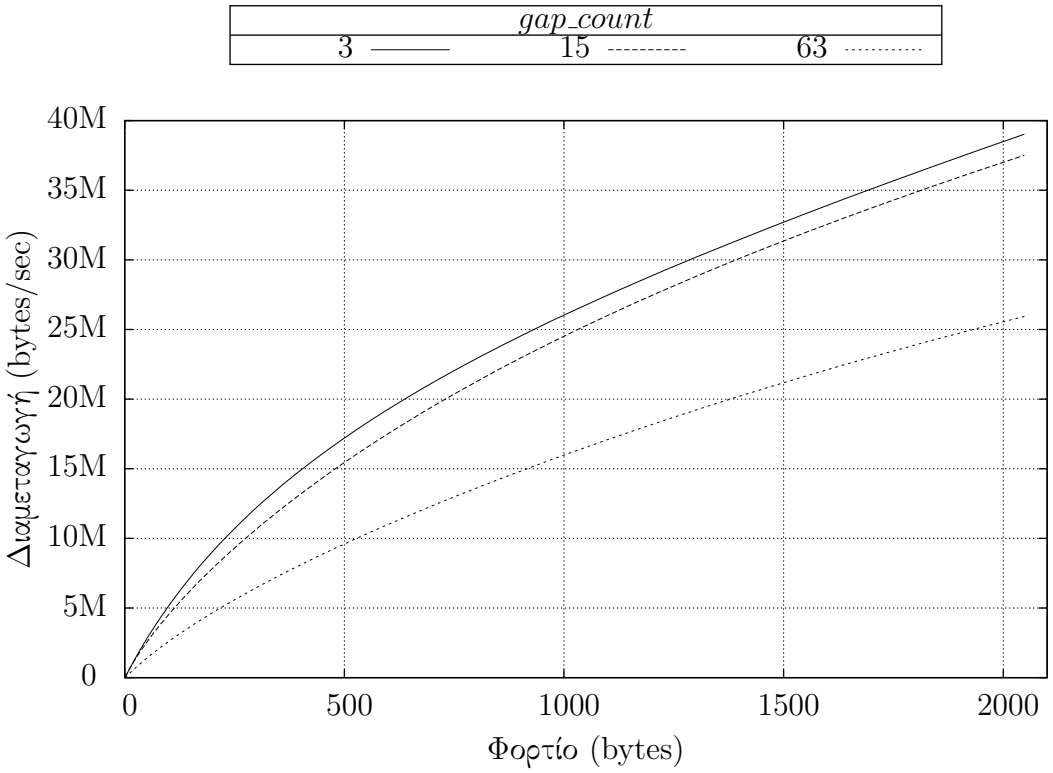
	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	23126	21987	21368	19802	8823	5240
2	66264	67923	64903	63606	40932	18700
4	160935	159627	155541	150132	107028	61800
8	343764	342258	332201	323484	232300	146703
16	705495	700657	678077	661419	7969528	313547
32	1405657	1417686	1382504	1180230	943857	644089
64	2866318	2776915	2712117	2218534	46469973	1300350
128	5252815	5292241	4577479	4020538	3746477	2542194
256	8117806	8738754	8387699	99910822	7188087	4834238
512	16093746	15798527	15583899	15437874	13985691	8700104
1024	27121593	27255533	26786863	26246104	22229707	15194813
2048	36058080	36449700	37222832	35172548	29573935	22529822

Μια από τις πιθανές εφαρμογές της τεχνικής που αναπτύχθηκε είναι η ανάκτηση των περιεχομένων της οθόνης ενός απομακρυσμένου κόμβου μέσω του διαύλου IEEE 1394, διαδικασία που μπορεί να γίνει ακόμα πιο αποτελεσματική με τη χρήση ενός διαύλου IEEE 1394b, ο οποίος παρέχει σαφώς υψηλότερες ταχύτητες μετάδοσης δεδομένων. Πέρα όμως από τις κάρτες γραφικών, η ίδια μέθοδος μπορεί να χρησιμοποιηθεί με οποιαδήποτε συσκευή η οποία επιτρέπει τη διευθυνσιοδότηση ενός ή περισσότερων από τους ενταμιευτές της μέσω του διαύλου συστήματος, προσφέροντας ποικίλες δυνατότητες.

4.2.1.7 Εγγραφή δεδομένων στη μνήμη γραφικών απομακρυσμένου κόμβου

Με βάση τα δεδομένα που προαναφέρθησαν στην παράγραφο 4.2.1.6, πραγματοποιήθηκαν μια σειρά από δοκιμές με τη χρήση ασύγχρονων συναλλαγών για την εγγραφή δεδομένων στον ενταμιευτή πλαισίου της κάρτας γραφικών του κόμβου B. Η διεύθυνση 0xd8000000 στην οποία βρίσκεται ανήκει στην περιοχή φυσικών διευθύνσεων, οπότε οι αιτήσεις εγγραφής εξυπηρετούνται ως ενοποιημένες συναλλαγές. Τα αποτελέσματα των μετρήσεων ήταν αναμενόμενα, παραπλήσια με αυτά της παραγράφου 4.2.1.2 για την εγγραφή δεδομένων στην κύρια μνήμη ενός απομακρυσμένου κόμβου. Αυτό οφείλεται τόσο στην υψηλότερη απόδοση των ενοποιημένων συναλλαγών, όσο και στο γεγονός ότι οι κάρτες γραφικών σχεδιάζονται πάντα έτσι ώστε να υποστηρίζουν υψηλούς ρυθμούς λήψης δεδομένων από το δίαυλο συστήματος.

Σχήμα 57: Διαμεταγωγή κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου B



Πίνακας 75: Καθυστέρηση (μs) κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου B

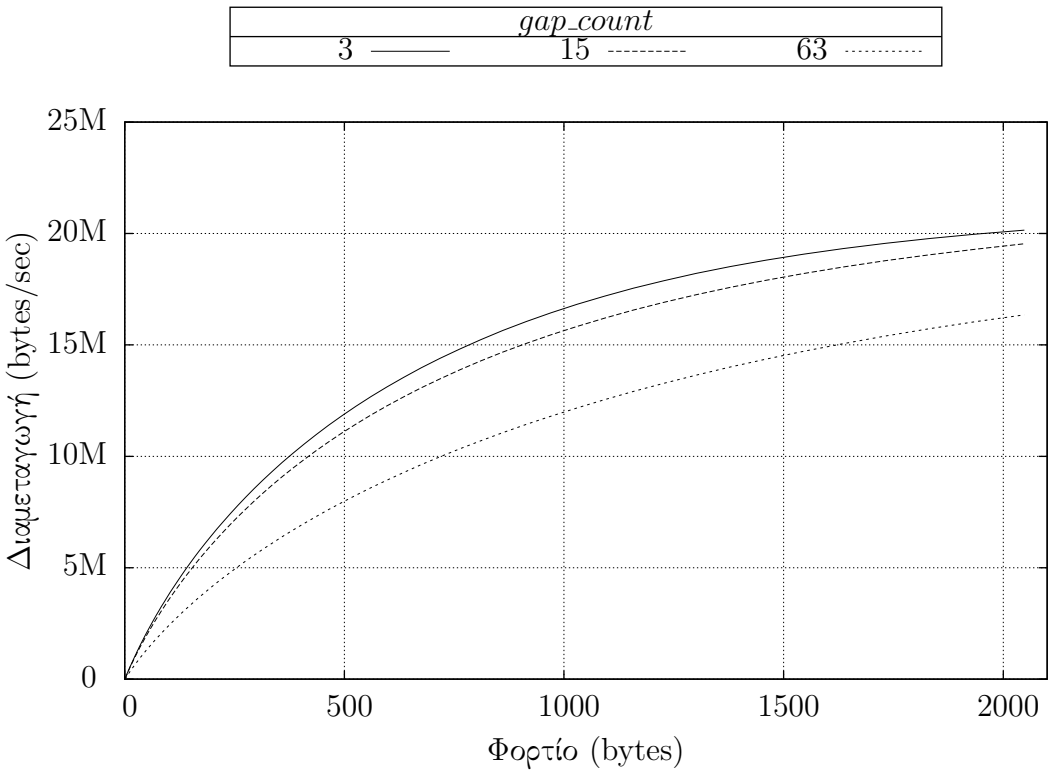
	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	48.37	48.96	48.94	48.57	48.89	52.11
2	48.45	48.80	48.63	48.52	48.63	49.47
4	47.11	47.65	47.12	47.40	44.97	47.77
8	48.63	48.67	48.15	48.60	47.66	49.48
16	48.94	48.74	48.80	48.83	48.53	50.11
32	49.25	46.96	49.27	49.50	49.24	50.65
64	50.02	49.97	50.26	50.19	50.18	51.78
128	53.78	53.93	53.88	53.62	54.25	55.35
256	57.92	57.78	57.87	58.96	58.48	59.34
512	67.87	67.86	67.98	67.92	69.19	69.91
1024	85.30	85.49	85.28	85.27	85.23	86.11
2048	114.77	115.44	115.67	116.69	115.44	119.07

Πίνακας 76: Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου B

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	39917	40297	39890	40361	20674	7811
2	101636	102056	99371	103802	61409	38631
4	173899	175210	175618	172813	152602	100523
8	480481	472858	471252	482704	324579	224075
16	972965	976753	975694	973998	626190	470440
32	1965121	1969876	1969892	1941764	1249768	961409
64	3889429	3416568	3588682	3554887	2654988	1913213
128	7700287	7731907	7691909	6495346	5461035	3660719
256	14921407	14480054	12771731	10718432	11561426	6888343
512	19150444	20199630	20025260	18767850	16478150	10552270
1024	30402665	27817976	28637455	26882558	25958646	17881872
2048	40225968	39025897	39094548	37505935	32780674	25935790

Κατά την αποστολή δεδομένων από τον κόμβο B στη μνήμη γραφικών του A έγινε πάλι αισθητό το όριο των 20 MB/sec που είδαμε και σε προηγούμενες παραγράφους. Οι χρόνοι ολοκλήρωσης των συναλλαγών για μικρά μεγέθη πακέτων, ωστόσο, φαίνεται να παραμένουν ανεπηρέαστοι από αυτόν τον περιορισμό.

Σχήμα 58: Διαμεταγωγή κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου Α



Πίνακας 77: Καθυστέρηση (μs) κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου Α

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	48.93	51.85	51.46	51.27	53.08	52.68
2	49.00	52.77	52.16	52.02	48.58	50.93
4	50.92	52.63	51.08	50.83	49.94	50.78
8	47.53	52.28	50.97	51.19	48.12	51.24
16	52.33	51.49	51.57	52.47	48.88	50.53
32	48.72	52.18	51.83	51.52	50.05	50.05
64	50.70	53.37	53.02	50.58	51.69	50.88
128	49.01	54.34	53.82	53.58	53.62	54.83
256	54.26	57.99	57.76	57.96	58.29	57.22
512	66.28	69.99	68.60	69.84	69.44	69.52
1024	90.69	93.85	93.80	94.44	94.04	94.08
2048	140.27	142.93	142.81	142.54	143.39	142.37

Πίνακας 78: Διαμεταγωγή (bytes/sec) κατά την εγγραφή δεδομένων στη μνήμη γραφικών του κόμβου Α

	gap_count					
Φορτίο (bytes)	1	3	7	15	31	63
1	24201	24572	23187	24575	18959	8909
2	73240	72119	69772	71251	61423	40875
4	222487	218605	174765	170346	151995	111221
8	356788	358116	353915	353388	310901	232444
16	717745	729476	701833	739413	638994	466094
32	1453290	1464767	1448789	1456563	1274355	941929
64	2881371	2907959	2901849	2695168	2684764	1852373
128	5650384	5706570	5447269	5131650	5197268	3457019
256	9533643	9485182	9443183	9310643	8510236	6108807
512	15616767	15714620	15243006	14221282	12230625	9817093
1024	18919800	18606740	18185736	17435666	16293665	13643577
2048	20340989	20148445	19942032	19537555	18758688	16349112

Με τη μέθοδο αυτή είναι προφανώς δυνατόν να μεταβληθεί το περιεχόμενο της οθόνης ενός απομακρυσμένου κόμβου. Ωστόσο η ίδια διαδικασία επιτρέπει την εξ αποστάσεως επέμβαση σε κάθε συσκευή η οποία εκθέτει τους καταχωρητές και τους εσωτερικούς ενταμιευτές της μέσω του διαύλου συστήματος, παρέχοντας πλήθος διαφορετικών δυνατοτήτων.

4.2.2 Ισόχρονες συναλλαγές

Στα πλαίσια των πειραματικών δοκιμών επιδιώχθηκε η σκιαγράφηση των ορίων της υλοποίησης και η σύγκριση με τα αντίστοιχα θεωρητικά όρια, όπως αυτά προκύπτουν από το πρότυπο IEEE 1394 και τις τροπολογίες του. Στον τομέα των ισόχρονων συναλλαγών οι επιδόσεις ήταν οι αναμενόμενες από τη θεωρητική μελέτη, με πολύ μικρές αποκλίσεις. Διαπιστώθηκε, ωστόσο, ότι το υποσύστημα ieee1394 επιβάλλει κάποιους περιορισμούς στο μέγεθος του φορτίου των πακέτων, οι οποίοι, αν και δεν επηρεάζουν τη συντριπτική πλειοψηφία των εμπορικών εφαρμογών, αποτελούν δυνητικά πηγή προβλημάτων.

Συγκεκριμένα, κατά τη λήψη ισόχρονων πακέτων μέσω ενός ελεγκτή OHCI το συνολικό μέγεθος του πακέτου δεν πρέπει να υπερβαίνει το μέγεθος μιας σελίδας μνήμης του υπολογιστή. Στην πλέον διαδεδομένη αρχιτεκτονική επεξεργαστών IA32, η οποία περιλαμβάνει όλους τους επεξεργαστές 32-bit της οικογένειας x86, το μέγεθος μιας σελίδας μνήμης είναι ακριβώς 4096 bytes. Αυτό σημαίνει ότι, αφαιρώντας την επικεφαλίδα των 8 bytes, το μέγιστο φορτίο ενός εισερχόμενου ισόχρονου πακέτου είναι 4088 bytes. Αυτό δεν θα αποτελούσε πρόβλημα για την αρχική προδιαγραφή του 1995, η οποία προέβλεπε μέγιστο φορτίο 2048 bytes, αλλά η τροπολογία του 2000 ανέβασε την τιμή αυτή στα 4096 bytes για την ταχύτητα S400, ενώ οι μετέπειτα τροπολογίες προβλέπουν ακόμα μεγαλύτερες τιμές όταν χρησιμοποιούνται υψηλότερες ταχύτητες μετάδοσης. Αυτός ο περιορισμός δεν υφίσταται κατά την αποστολή ισόχρονων πακέτων.

Πίνακας 79: Διαμεταγωγή (bytes/sec) με ισόχρονες συναλλαγές

Φορτίο (bytes)	Διαμεταγωγή (bytes/sec)
8	32200
16	96600
32	225398
64	483000
128	998168
256	2027849
512	4087540
1024	8202199
2048	16431642
4088	32687944
4096	32693958

Μια ιδιαιτερότητα κατά τη τέλεση ισόχρονων συναλλαγών προκύπτει όταν ζητείται η δημιουργία μιας ροής δεδομένων με σταθερό ρυθμό μετάδοσης. Με τις ασύγχρονες συναλλαγές η απλούστερη μέθοδος είναι να ρυθμιστεί η συχνότητα με την οποία εκπέμπονται τα πακέτα. Ο μηχανισμός με τον οποίο πραγματοποιούνται οι ισόχρονες συναλλαγές στο Linux, όμως, καθιστά αυτή τη μέθοδο προβληματική, καθώς έρχεται σε σύγκρουση με τους περιορισμούς που επιβάλλει η σταθερή συχνότητα των κύκλων χρονισμού. Κατά συνέπεια, είναι προτιμότερο να μεταβάλλεται το μέγεθος του φορτίου κάθε πακέτου, με τη συχνότητα εκπομπής πακέτων

να διατηρείται σταθερή στα 8 kHz. Αυτή η τεχνική μειώνει την απόδοση του διαύλου, καθώς ελαττώνεται το μέγεθος των πακέτων που χρησιμοποιούνται, αλλά εξασφαλίζει μια πραγματικά σταθερή ροή δεδομένων.

Μια αξιοσημείωτη παρατήρηση που προέκυψε κατά τις δοκιμές αυτές είναι ότι ο κόμβος B δεν είχε καμία δυσκολία κατά τη μετάδοση ισόχρονων πακέτων. Αυτό μας επιτρέπει να συμπεράνουμε με σχετική βεβαιότητα ότι τα προβλήματα που διαπιστώσαμε κατά τις δοκιμές με ασύγχρονες συναλλαγές οφείλονται στον ελεγκτή IEEE 1394 που φέρει, καθότι τα υπόλοιπα εξαρτήματα του υπολογιστή συμμετέχουν στην πραγματοποίηση και των δύο τύπων συναλλαγών, ενώ το λογισμικό είναι κοινό και για τους δύο κόμβους.

4.2.3 Μετάδοση κινούμενης εικόνας

Τα αποτελέσματα των πειραματικών μετρήσεων που παρατέθηκαν στις προηγούμενες παραγράφους συνιστούν μια αντικειμενική αξιολόγηση των δυνατοτήτων του διαύλου IEEE 1394. Σε εφαρμογές κινούμενης εικόνας, ωστόσο, είναι ιδιαίτερα σημαντικός ο ανθρώπινος παράγοντας, καθώς η υποκειμενική του αντίληψη είναι τελικά αυτή που κρίνει το βαθμό επιτυχίας κάθε τεχνικής. Για το λόγο αυτό το λογισμικό v1394 χρησιμοποιήθηκε για την τέλεση μιας σειράς δοκιμών με διαφορετικές μεθόδους μεταφοράς και απεικόνισης κινούμενης εικόνας.

Σημειώνεται ότι δεν έγιναν μετρήσεις της καθυστέρησης ειδικά για τη μετάδοση κινούμενης εικόνας. Το λογισμικό που αναπτύχθηκε δεν έχει τη δυνατότητα επεξεργασίας ενός πλαισίου πριν ολοκληρωθεί η εισαγωγή του στη μνήμη, οπότε οι συνολικές καθυστερήσεις που προκύπτουν δεν είναι αντιπροσωπευτικές των ιδιοτήτων του διαύλου IEEE 1394. Για μια σχετικά ακριβή εκτίμηση των χρόνων καθυστέρησης που θα συναντήσει μια πλήρως βελτιστοποιημένη εφαρμογή μπορούν να χρησιμοποιηθούν τα μεγέθη που μετρήθηκαν σε επίπεδο πακέτου στις προηγούμενες παραγράφους.

4.2.3.1 Λήψη κινούμενης εικόνας από απομακρυσμένο ενταμιευτή

Για τη δοκιμή αυτή επιλέχθηκε ένα αρχείο κινούμενης εικόνας διάρκειας 60 δευτερολέπτων με 30 πλαίσια ανά δευτερόλεπτο, το οποίο αναπαράγεται σε ανάλυση $640 * 480$ και βάθος χρώματος 16 bit. Ο ισοδύναμος ρυθμός μετάδοσης δεδομένων για αυτό το αρχείο θα είναι 18000 KiB/sec, αρκετά χαμηλός έτσι ώστε να μην δημιουργηθεί πρόβλημα ακόμα και με τον ελεγκτή του κόμβου B, ο οποίος παρουσίασε ένα άνω φράγμα των 20 MB/sec περίπου κατά την εκπομπή δεδομένων σε ασύγχρονες συναλλαγές. Στη συνέχεια το αρχείο αυτό τροφοδοτείται στο λογισμικό v1394 έτσι ώστε να μεταδοθεί μέσω ενός ενταμιευτή ARM στον κόμβο B, με τον κόμβο A να χρησιμοποιεί ασύγχρονες συναλλαγές για την ανάγνωση των πλαισίων τα οποία εμφανίζει στην οθόνη του υπολογιστή.

Με τις συγκεκριμένες ρυθμίσεις δεν παρατηρήθηκαν ατέλειες κατά την προβολή της ροής δεδομένων από το δίαυλο IEEE 1394, συγκριτικά με το αρχικό αρχείο. Διαπιστώθηκε ωστόσο ότι αν η συχνότητα λήψης πλαισίων ανέβει σε σημείο που να οδηγεί τον επεξεργαστή του κόμβου B σε κορεσμό, τότε ο ενταμιευτής ARM δεν ανανεώνεται πλήρως πριν αναγνωσθεί το επόμενο πλαίσιο, με αποτέλεσμα σε κάποιες χρονικές στιγμές η εικόνα που προβάλλεται στην οθόνη να αποτελείται από τμήματα δύο ή περισσότερων πλαισίων του αρχείου προέλευσης.

Το λογισμικό v1394 δεν περιέχει κάποιον αλγόριθμο για την αντιμετώπιση της επικάλυψης πλαισίων που περιγράφηκε παραπάνω, καθώς κρίθηκε ότι θα ανέβαζε υπέρμετρα την πολυπλοκότητα του κώδικα. Αυτό το πρόβλημα θα μπορούσε να αντιμετωπιστεί σε κάποιο βαθμό με τη βοήθεια ενός διπλού ενταμιευτή πλαισίου (double buffering), αν και αυτή η μέθοδος απαιτεί τη δέσμευση της διπλάσιας ποσότητας μνήμης στο χώρο του πυρήνα και δεν εξασφαλίζει την επίλυση του ζητήματος σε ακραίες περιπτώσεις. Μια εναλλακτική λύση με εξασφαλισμένη αποτε-

λεσματικότητα θα ήταν η χρήση μιας αμφίδρομης μεθόδου συγχρονισμού ανάμεσα στους δύο κόμβους, αν και σε αυτήν την περίπτωση παρουσιάζονται ελαφρώς μειωμένες επιδόσεις και απαιτείται ειδική μέριμνα για το ενδεχόμενο απώλειας πακέτων ελέγχου.

4.2.3.2 Εγγραφή κινούμενης εικόνας σε απομακρυσμένο ενταμιευτή

Στο πείραμα αυτό χρησιμοποιήθηκε το ίδιο αρχείο κινούμενης εικόνας με την παράγραφο 4.2.3.1, με τον κόμβο B να αποτελεί την πηγή της ροής δεδομένων στο δίαυλο IEEE 1394. Η οπτική αξιολόγηση του αποτελέσματος δεν αποκάλυψε ατέλειες στη συγκεκριμένη συχνότητα πλαισίου, αν και πάλι αυτό αλλάζει αν αυτή ανέβει σημαντικά, λόγω της απουσίας αμφίδρομου συγχρονισμού ανάμεσα στους δύο κόμβους.

4.2.3.3 Λήψη κινούμενης εικόνας από τον ενταμιευτή πλαισίου απομακρυσμένου κόμβου

Για την πραγματοποίηση αυτού του πειράματος η κάρτα γραφικών του κόμβου B τέθηκε σε ανάλυση 640∗480 και βάθος χρώματος 16 bit. Στη συνέχεια χρησιμοποιήθηκε το λογισμικό v1394 στον κόμβο A για την εμφάνιση του περιεχομένου του ενταμιευτή πλαισίου του κόμβου B στην οθόνη. Η περιορισμένη ταχύτητα ανάγνωσης από την κάρτα γραφικών S3 Virge περιόρισε την συχνότητα ανανέωσης στα 12.5 πλαίσια ανά δευτερόλεπτο, αλλά δεν υπήρχαν σημαντικές ατέλειες κατά την προβολή απλού σχετικά περιεχομένου όπως λόγου χάρη η κονσόλα του λειτουργικού συστήματος Linux. Δεν επιχειρήθηκε η αντιστροφή των ρόλων των δύο κόμβων, επειδή ο κόμβος B δεν μπόρεσε να ανταπεξέλθει ικανοποιητικά στις απαιτήσεις που θέτει η έξοδος κινούμενης εικόνας με χρήση της βιβλιοθήκης SDL στην ισχύ τόσο του κεντρικού επεξεργαστή, όσο και της κάρτας γραφικών.

4.2.3.4 Εγγραφή κινούμενης εικόνας στον ενταμιευτή πλαισίου απομακρυσμένου κόμβου

Σε αυτή τη δοκιμή το v1394 χρησιμοποιήθηκε στον κόμβο A για την αποστολή του αρχείου κινούμενης εικόνας της παραγράφου 4.2.3.1 στην κάρτα γραφικών του κόμβου B. Το αποτέλεσμα ήταν η προβολή του περιεχομένου του αρχείου στην οθόνη του απομακρυσμένου υπολογιστή. Διαπιστώθηκε ότι η αναπαραγωγή του αρχείου γίνεται χωρίς ατέλειες, με την προϋπόθεση το λογισμικό του κόμβου B να μην ανανεώνει την οθόνη κατά τη διάρκεια της δοκιμής, κάτι που μπορεί να

εξασφαλιστεί με σχετική ευκολία.

4.2.3.5 Μετάδοση κινούμενης εικόνας με ισόχρονες συναλλαγές

Με βάση τα στοιχεία των ποσοτικών μετρήσεων οι ισόχρονες συναλλαγές αποτελούν την πλέον προβλέψιμη μέθοδο μετάδοσης κινούμενης εικόνας, με μόνη ιδιαιτερότητα την απαίτηση να προσαρμοστεί το μέγεθος των πακέτων έτσι ώστε να επιτευχθεί ο επιθυμητός ρυθμός ανανέωσης πλαισίου. Πράγματι, τα αποτελέσματα ήταν ικανοποιητικά, χωρίς απώλειες συγχρονισμού ή άλλα προβλήματα κατά τη μετάδοση της κινούμενης εικόνας. Ταυτόχρονα, χάρη στον τρόπο με τον οποίο γίνεται ο χρονισμός των ισόχρονων πακέτων στο δίαυλο IEEE 1394 και η διαχείρισή τους από το λειτουργικό σύστημα Linux και το λογισμικό v1394, αποφεύγονται τα προβλήματα που προκύπτουν κατά την τέλεση ασύγχρονων συναλλαγών όταν ένας κόμβος δρα ταχύτερα από τον άλλον.

Επιπλέον, όπως προαναφέρθηκε στην παράγραφο 4.2.2, η τέλεση ισόχρονων συναλλαγών ήταν η μόνη διαθέσιμη μέθοδος για την επίτευξη ενός αποδεκτού βαθμού χρήσης του διαθέσιμου εύρους ζώνης κατά τη μετάδοση κινούμενης εικόνας από τον κόμβο B. Κατά τη μετάδοση του αρχείου που προαναφέραμε με τη μέγιστη δυνατή ταχύτητα παρουσιάστηκε ένας μέγιστος ρυθμός ανανέωσης της τάξης των 45 πλαισίων ανά δευτερόλεπτο, με τη διαθέσιμη επεξεργαστική ισχύ να αποτελεί τον περιοριστικό παράγοντα.

4.2.4 Συμπεράσματα

Με το σύνολο των πειραματικών δοκιμών που πραγματοποιήθηκαν σκιαγραφήθηκαν με σχετική πληρότητα οι δυνατότητες και τα ιδιαίτερα χαρακτηριστικά του διαύλου IEEE 1394. Η μεγαλύτερη έμφαση δόθηκε στη σύγκριση των θεωρητικών βέλτιστων τιμών, όπως αυτές προκύπτουν από το πρότυπο του διαύλου και τις τροπολογίες του, με τις επιδόσεις ενός πραγματικού συστήματος που αποτελείται από δύο υπολογιστές.

Μια σημαντική παρατήρηση ήταν η επίπτωση των εγγενών καθυστερήσεων που προκύπτουν από τη χρήση λογισμικού σε έναν επεξεργαστή γενικού σκοπού για τη διαχείριση του διαύλου. Η αύξηση που επιφέρουν στο χρόνο ολοκλήρωσης κάθε διεργασίας ευνοεί δραστικά τη χρήση πακέτων με μεγάλα φορτία κατά την τέλεση ασύγχρονων συναλλαγών, σε βαθμό πολύ υψηλότερο από τον θεωρητικά αναμενόμενο.

Η πιο αξιοσημείωτη διαπίστωση, ωστόσο, ήταν η διαφοροποίηση που αναπόφευκτα προκύπτει ανάμεσα σε μια προδιαγραφή και την υλοποίησή της. Εν γένει οι θεωρητικές προδιαγραφές είναι αισιόδοξες, με την πρακτική εφαρμογή τους να είναι ελαφρώς υποδεέστερη. Επιπλέον, μόνο ένα μικρό υποσύνολο ενός προτύπου είναι συνήθως υποχρεωτικό, με κάποια πιο προηγμένα στοιχεία να είναι προαιρετικά. Χαρακτηριστικό παράδειγμα αποτελούν κάποιοι ελεγκτές IEEE 1394 οι οποίοι δεν υποστηρίζουν το μέγιστο φορτίο πακέτου που προβλέπεται ή απαιτούν πιο χαλαρό χρονισμό κατά την τέλεση συναλλαγών προκειμένου να ανταπεξέλθουν. Στην περίπτωση που μελετήθηκε έγινε ιδιαίτερα εμφανής η πτώση στις επιδόσεις που μπορεί να επιφέρει μια τεχνικά ελλιπής, αν και τυπικά συμβατή, υλοποίηση.

5 Επίλογος

Στα πλαίσια αυτής της εργασίας πραγματοποιήσαμε μια εκτενή θεωρητική ανάλυση των χαρακτηριστικών του διαύλου IEEE 1394, με έμφαση σε αυτά που έχουν ενδιαφέρον κατά τη χρήση του σε εφαρμογές πολυμέσων. Στη συνέχεια αναπτύχθηκε το λογισμικό v1394 για τη μετάδοση κινούμενης εικόνας χωρίς συμπίεση από έναν κόμβο σε έναν άλλο, κάτι που επέτρεψε την εκτέλεση δοκιμών σε ένα πραγματικό σύστημα.

Με τη χρήση του λογισμικού αυτού, το οποίο βασίζεται στην υποδομή που παρέχει το λειτουργικό σύστημα Linux, σκιαγραφήθηκε η συμπεριφορά δύο υπολογιστών που ήταν συνδεδεμένοι μέσω ενός διαύλου IEEE 1394. Με αυτή τη βάση αξιολογήθηκαν τόσο τα τεχνικά χαρακτηριστικά του διαύλου, όσο και η παρεχόμενη υποστήριξη για τη χρήση του στο περιβάλλον ενός ηλεκτρονικού υπολογιστή.

Κατά τη μελέτη των πειραματικών αποτελεσμάτων κατέστη εφικτός ο προσεγγιστικός προσδιορισμός των τεχνικών προδιαγραφών των δύο διαφορετικών ελεγκτών IEEE 1394 που χρησιμοποιήθηκαν. Έγινε εμφανές ότι μια σειρά από παρατηρούμενες συμπεριφορές συνδέονται άμεσα με το σχεδιασμό του υλικού των κόμβων, σε συνδυασμό με τη δομή και τους περιορισμούς του λογισμικού υποστήριξης.

Σε γενικές γραμμές ο δίαυλος IEEE 1394 έδωσε αποτελέσματα κοντά στα θεωρητικά αναμενόμενα, αλλά έγινε εμφανής η διαφορά που μπορεί να υπάρξει ανάμεσα σε δύο υλοποιήσεις του προτύπου όταν μία από αυτές παρουσιάζει τεχνικές και σχεδιαστικές αδυναμίες. Ανάλογα με την εκάστοτε εφαρμογή, η ύπαρξη μιας υλοποίησης που υστερεί μπορεί να αποβεί ιδιαίτερα προβληματική, καθώς δεν επιτρέπει την επίτευξη των βέλτιστων προβλεπόμενων επιδόσεων.

Η χρήση του πειραματικού συστήματος για τη μετάδοση κινούμενης εικόνας έγινε γενικά με επιτυχία. Στην περίπτωση της κινούμενης εικόνας χωρίς συμπίεση με χαμηλή έως και μέση ανάλυση ο δίαυλος IEEE 1394 είναι σε θέση να παρέχει μετάδοση σε πραγματικό χρόνο και χωρίς αισθητά προβλήματα. Αυτό δεν ισχύει όταν η ανάλυση της εικόνας ανεβαίνει, καθώς ο ρυθμός ανανέωσης πλαισίου πέφτει δραστικά, σε σημείο που να γίνεται αισθητό από τον θεατή. Ωστόσο αυτό είναι ένα πρόβλημα που μπορεί να λυθεί με τη χρήση τεχνικών συμπίεσης, όπως συμβαίνει για παράδειγμα στις καταναλωτικές συσκευές τεχνολογίας DV.

A Βιβλιογραφία

Εκδόσεις:

- 1394 Open Host Controller Interface Specification - Release 1.1
  January 6, 2000

- AGP V3.0 Interface Specification - Revision 1.0
  Intel Corporation
  September 2002

- IEEE Std 1212^{TM}-2001, IEEE Standard for a Control and Status Registers (CSR) Architecture for Microcomputer Buses

- IEEE Std 1394^{TM}-1995, IEEE Standard for a High Performance Serial Bus

- IEEE Std 1394a^{TM}-2000, IEEE Standard for a High-Performance Serial Bus - Amendment

- IEEE Std 1394b^{TM}-2002, IEEE Standard for a High-Performance Serial Bus - Amendment 2

- IEEE Std 1394c^{TM}-2006, IEEE Standard for a High-Performance Serial Bus - Amendment 3

- IEEE Std 1394.1^{TM}-2004, IEEE Standard for High Performance Serial Bus Bridges

- PCI Local Bus Specification - Revision 2.2
  PCI Special Interest Group
  December 18, 1998

- PCI Local Bus Specification - Revision 3.0
  PCI Special Interest Group
  February 3, 2004

- PCI-to-PCI Bridge Architecture Specification - Revision 1.1
  PCI Special Interest Group
  December 18, 1998

- PCI Express Base Specification - Revision 1.0
  PCI Special Interest Group
  April 29, 2002

- TSB12LV26-EP OHCI Lynxt PCI Based IEEE 1394 Host Controller Data Manual
  Texas Instruments Inc.
  February 9, 2005

- VT6306 PCI 1394a Integrated Host Controller - Revision 1.16
  VIA Technologies Inc.
  July 19, 2002

- Δίκτυα Υπολογιστών - 3η Έκδοση
  Andrew S. Tanenbaum
  Μετάφραση: Ε. Συκάς, Μ. Θεολόγου, Β. Στυλιανάκης
  Αθήνα 2000

- Λειτουργικά Συστήματα
  Γ. Κ. Παπακωνσταντίνου, Ν. Α. Μπιλάλης, Π. Δ. Τσανάκας
  Αθήνα 1999

- Οργάνωση & Αρχιτεκτονική Υπολογιστών - 6η Έκδοση
  William Stallings
  Μετάφραση: Γ. Κ. Αδάμ, Μ. Δ. Τζιόλας
  Θεσσαλονίκη 2003

- Το Λειτουργικό Σύστημα EMPIX
  Γ. Παπακωνσταντίνου, Χ. Δελλαρόκας, Π. Τσανάκας
  Αθήνα 1993

Ιστοσελίδες:

- Accelerated Graphics Port
  URL: http://en.wikipedia.org/wiki/Accelerated_Graphics_Port
  Wikipedia, the free encyclopedia
  Ανακτήθηκε: 29 Σεπτεμβρίου 2008

- Fundamentals of Firewire
  URL: http://www.embedded.com/1999/9906/9906feat2.htm
  John Canosa
  Ανακτήθηκε: 12 Αυγούστου 2008

- Host controller interface
  URL: http://en.wikipedia.org/wiki/Host_controller_interface
  Wikipedia, the free encyclopedia
  Ανακτήθηκε: 20 Σεπτεμβρίου 2008

- IEEE 1394 interface
  URL: http://en.wikipedia.org/wiki/IEEE_1394_interface
  Wikipedia, the free encyclopedia
  Ανακτήθηκε: 17 Σεπτεμβρίου 2008

- PCI Express
  URL: http://en.wikipedia.org/wiki/PCI_Express
  Wikipedia, the free encyclopedia
  Ανακτήθηκε: 28 Σεπτεμβρίου 2008

- PCI Local Bus
  URL: http://en.wikipedia.org/wiki/Peripheral_Component_Interconnect
  Wikipedia, the free encyclopedia
  Ανακτήθηκε: 28 Σεπτεμβρίου 2008

B Τεχνικά χαρακτηριστικά διάταξης δοκιμών

B.1 Κόμβος A

Ο κόμβος A είναι ένας φορητός υπολογιστής Dell Latitude D800 με τα ακόλουθα τεχνικά χαρακτηριστικά:

Υλικό:

- Κεντρικός επεξεργαστής: Intel Pentium M 1.70GHz
- Περιφερειακά κυκλώματα υποστήριξης: Intel 82855PM
- Μνήμη: 1024 MiB με μέγιστη ταχύτητα μεταφοράς δεδομένων περίπου 2100 MB/sec
- Κάρτα γραφικών: nVidia GeForce FX Go5200 64MiB AGP
- Ελεγκτής IEEE 1394: Texas Instruments OHCI-Lynx PCI

Λογισμικό:

- Λειτουργικό σύστημα GNU/Linux 2.6.26.3
- Πρότυπη βιβλιοθήκη C: GNU glibc-2.7
- Μεταγλωττιστής: GNU gcc-4.2.4
- Βιβλιοθήκη IEEE 1394: libraw1394-2.0.0
- Βιβλιοθήκη χειρισμού τερματικού: GNU ncurses-5.6
- Βιβλιοθήκη πολυμέσων: SDL-1.2.13

B.2 Κόμβος B

Ο κόμβος B είναι ένας επιτραπέζιος υπολογιστής με τα ακόλουθα τεχνικά χαρακτηριστικά:

Υλικό:

- Κεντρικός επεξεργαστής: AMD Athlon 900 MHz
- Περιφερειακά κυκλώματα υποστήριξης: VIA Technologies Inc. KT133A
- Μνήμη: 256 MiB με μέγιστη ταχύτητα μεταφοράς δεδομένων περίπου 533 MB/sec
- Κάρτα γραφικών: S3 Virge 2MiB PCI
- Ελεγκτής IEEE 1394: VIA Technologies Inc. VT6306 PCI

Λογισμικό:

- Λειτουργικό σύστημα GNU/Linux 2.6.26.1
- Πρότυπη βιβλιοθήκη C: GNU glibc-2.3.2
- Μεταγλωττιστής: GNU gcc-3.2.2
- Βιβλιοθήκη IEEE 1394: libraw1394-2.0.0
- Βιβλιοθήκη χειρισμού τερματικού: GNU ncurses-5.3
- Βιβλιοθήκη πολυμέσων: SDL-1.2.7

C Πηγαίος κώδικας

Παρακάτω παρατίθεται ο πηγαίος κώδικας του λογισμικού v1394 και των συνοδευτικών εργαλείων του. Στον κώδικα συμπεριλαμβάνονται εκτενή σχόλια, που έχουν γραφεί στην Αγγλική για λόγους ευκολίας κατά την ανάπτυξή του.

C.1 cbase.c

Περιγραφή: Συναρτήσεις για τη δημιουργία συστατικών στοιχείων του περιβάλλοντος χρήστη του v1394

```
#include "cbase.h"


/* A basic XMENU menu control keyboard loop */
int cbase_xmkbd(MENU *m, void **p) {
        FUNCINIT;

        i0 = KEY_HOME;
        while ((i0 != 'q') && (i0 != 'r') && (i0 != '\n')) {
                cterm_xmenu_kdrv(m, i0, p);
                cterm_xmenu_detail(m, &s0);
                OUT("%s", s0);

                i0 = GETCH;
        }
        CNDJMP(i0 == 'q', end, 1);
        CNDJMP(i0 == 'r', end, 2);

end:
        return R;
}


/* Select an IEEE 1394 node */
int cbase_n1394(CLIST **t, int *n, int *p, U32 *v, U64 *c, P1394 **d, int o) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND((p == NULL) || (v == NULL) || (c == NULL), e00, -1,
                "invalid pointer");

        /* List pointers */
        CLIST *l = NULL, *ii = NULL, *jj = NULL, *kk = NULL;

        /* Get the node list */
        if ((t == NULL) || (*t == NULL)) {
                do {
                        r = u1394_blist(NULL, &l, NULL, &i0);
                } while (r == RST);
                CNDJMP(r != 0, e00, -1);
        } else {
                ERRCND(n == NULL, e00, -1, "invalid node count pointer");
                l = *t;
                i0 = *n;
        }
        ERRCND((t != NULL) && (n == NULL), e00, -1,
                "invalid node count pointer");

        /* The menu definition root pointer */
        void ****s = NULL;

        /* The menu definition root array */
```

```
CALLOC(s, i0 + 1, SZP(void), e01, FAIL, "");

/* The menu strings */
i = 0;
CLISTFOR(l, ii) {
        B1394 *b = ii->d;

        CLISTFOR(b->n, jj) {
                P1394 *p = jj->d;

                if ((o == 0) && (PHY(p->b->l) != p->n))
                        continue;

                CALLOC(s[i], 2, SZP(void), e02, FAIL, "");
                CALLOC(s[i][0], 5, SZP(void), e02, FAIL, "");

                /* The menu item name */
                MKSTR(s[i][0][0], e02, FAIL, 0, " %02d:%02d %c  ",
                        p->p, p->n, (p->n == PHY(p->b->l))?'*':' ');

                /* The menu item description */
                s[i][0][1] = strdup(IFPTR(p->txt, p->txt->d, ""));

                /* The menu item details */
                MKSTR(s[i][0][2], e02, FAIL, p->ltxt + p->ntxt * 3,
                        "Port: %02d    [%-48s]\n\n"
                        "\t\tLocal node                    : %02d\n"
                        "\t\tIsochronous Resourse Manager : %02d\n"
                        "\t\tBus Master                    : %02d%s\n\n"
                        "Node: %02d    [%-48s]\n\n"
                        "\tVendor:  0x%06x\n"
                        "\tProduct: 0x%010llx\n"
                        "\tCapabilities:\n"
                        "\t\tCycle Master                  : %d\n"
                        "\t\tIsochronous Resourse Manager : %d\n"
                        "\t\tBus Master                    : %d\n"
                        "\t\tIsochronous operations        : %d\n"
                        "\t\tMaximum write size            : %4d\n\n"
                        "\tText descriptors:\n",
                        p->p, p->b->d, PHY(p->b->l), PHY(p->b->i),
                        PHY(p->b->b), (PHY(p->b->b) == 63)?" (none)":"",
                        p->n, (char *)IFPTR(p->txt, p->txt->d, ""),
                        p->vid, p->cid, p->cmc, p->irmc, p->bmc, p->isc,
                        p->maxwr
                );
                CLISTFOR(p->txt, kk) {
                        strcat(s[i][0][2], "\t\t");
                        strcat(s[i][0][2], (char *)(kk->d));
                        strcat(s[i][0][2], "\n");
                }

                /* The user data pointer */
                MKPTR(s[i][0][3], p, e02, FAIL, "");

                ++i;
        }
}

/* Page footer */
STS("Enter:Accept    R:Restart    Q:Quit");

/* Menu creation */
MKXMENU(m, s, &p0, e02, e03);

/* Return the selected node properties */
P1394 **u = p0;
*p = (*u)->p;
*v = (*u)->vid;
*c = (*u)->cid;
if ((t != NULL) && (d != NULL))
        *d = *u;
```

```
e03:
        cterm_xmenu_free(&m, 0);
e02:
        shlib_ptr_free((void *)(&s), 4, 4);
e01:
        if ((t == NULL) || (r < 0)) {
                u1394_blist_free((void *)(&l));
        } else {
                *t = l;
                *n = i0;
        }
e00:
        return R;
}


/* Read an IEEE 1394 address */
int cbase_a1394(A1394 s, A1394 *a) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND(a == NULL, e00, -1, "invalid pointer");

        MKSTR(p0, e00, -1, 0, "0x%012llx", s);

        STS("Enter:Accept");
        do {
                r = cterm_inbox(wi, (void *)(&p1), 18, p0);
                CNDJMP(r < 0, e01, -1);

                r = sscanf(p1, "0x%012llx", a);
                CNDJMP(r < 0, e02, -1);
        } while (r < 1);

e02:
        GFREE(p1);
e01:
        GFREE(p0);
e00:
        return R;
}


/* Select isochronous transfer channel */
int cbase_ichan(int s, int *c) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND(c == NULL, e00, -1, "invalid pointer");

        /* Menu definition */
        void *d = PAR(
                XITEMINT("Channel number: ", "%2.2i", "", IAR(s), IAR(0),
                        IAR(63))
        );

        /* Page footer */
        STS("Enter:Accept    R:Restart    Q:Quit");

        /* Menu creation */
        MKXMENU(m, d, &p0, e00, e01);

        /* Act on the user selection */
        *c = XITEMINT_CUR(m, 0);

e01:
        cterm_xmenu_free(&m, 0);
e00:
        return R;
}
```

```
/* Read a file name */
int cbase_fname(char *s, char **f) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND(f == NULL, end, -1, "invalid pointer");

        STS("Enter:Accept");
        do {
                GFREE(*f);

                r = cterm_inbox(wi, f, 0, s);
                CNDJMP(r < 0, end, -1);

                if (strlen(*f) > 0) {
                        break;
                } else {
                        GFREE(*f);
                        OUT("Invalid filename!");
                }
        } while (1);

end:
        return R;
}
```

C.2 cbase.h

```
#ifndef __CBASE_H__
#define __CBASE_H__


/*
 * Local headers
 */

#include "cterm.h"
#include "prcdp.h"
#include "prcfm.h"
#include "prcfw.h"


/*
 * Local macros
 */

/* Create a standard XMENU menu */
#define MKXMENU(m, d, p, l0, l1) \
        /* Menu creation */ \
        MENU *m = NULL; \
        r = cterm_xmenu(wi, &m, d); \
        CNDJMP(r < 0, l0, -1); \
        \
        /* The keyboard input loop */ \
        r = cbase_xmkbd(m, p); \
        CNDJMP(r < 0, l0, -1); \
        CNDJMP(r > 0, l1, r);

/* Check menu return value */
#define XMCHK(r, l) \
        CNDJMP(r < 0, l, -1); \
        CNDJMP(r > 0, l, r);


/*
 * Local functions
 */

int cbase_xmkbd(MENU *, void **);
int cbase_n1394(CLIST **, int *, int *, U32 *, U64 *, P1394 **, int);
int cbase_a1394(A1394, A1394 *);
int cbase_ichan(int, int *);
int cbase_fname(char *, char **);


#endif /* __CBASE_H__ */
```

C.3 clist.c

Περιγραφή: Υλοποίηση μιας διπλά συνδεδεμένης κυκλικής λίστας

```
#include "clist.h"


/*
 * Circular list specifics:
 *
 * - A list is referenced using a pointer to its first node. An empty list is
 *   referenced by a NULL pointer. A node is referenced by a pointer to it.
 * - Each node contains a pointer to the previous and to the next node, as well
 *   as a data pointer. For the first node, the pointer to the previous node in
 *   fact points to the last node of the list. If there is only one node, then
 *   both its node pointers point to itself.
 */


/* Insert a new node before a reference node */
int clist_insrt(CLIST **r, CLIST *n) {
        if (n == NULL) {
                ERRMSG("invalid list node pointer");
                return -1;
        }

        /* Adjust the node pointers */
        if (*r == NULL) {
                /*
                 * If the reference node does not exist, make both
                 * node pointers point to the new node itself.
                 */
                n->p = n;
                n->n = n;
        } else {
                /* Point to the previous and next nodes */
                n->p = (*r)->p;
                n->n = *r;

                /* Adjust the neighbour node pointers */
                (*r)->p->n = n;
                (*r)->p = n;
        }

        /* Alter the reference node pointer to point to the new node */
        *r = n;

        return 0;
}


/* Create a new node before a reference node */
int clist_mknod(CLIST **r, void *d) {
        /* First allocate a new node */
        CLIST *n = calloc(1, SZO(CLIST));
        if (n == NULL) {
                ERRMSG("unable to create list node: %s", strerror(errno));
                return -1;
        }

        /* Set the data pointer */
        n->d = d;

        return clist_insrt(r, n);
}


/* Extract a node from a list */
int clist_xtrct(CLIST **n, CLIST **x) {
        /* Return immediately on a NULL pointer */
```

```
        if (*n == NULL)
                return -1;

        /* Optionally keep the node pointer */
        if (x != NULL)
                *x = *n;

        if ((*n)->n == *n) {
                /*
                 * If this is the only node in the list, just
                 * free it and set the list pointer to NULL.
                 */
                if (x == NULL)
                        GFREE(*n);
                *n = NULL;
        } else {
                CLIST *t = *n;

                /* Adjust the neighbour node pointers */
                t->n->p = t->p;
                t->p->n = t->n;

                /* Return a pointer to the next node */
                *n = t->n;

                /* Free the node */
                if (x == NULL)
                        GFREE(t);
        }

        return 0;
}


/*
 * Delete a node, optionally freeing the pointed-to data as well, and
 * return a pointer to the next node of the list, or NULL if the list
 * is now empty. A destroyer for the data can also be specified.
 */
int clist_rmnod(CLIST **n, int d, void (*f)(void **)) {
        if (d) {
                if (f != NULL) {
                        (*f)(&((*n)->d));
                } else {
                        GFREE((*n)->d);
                }
        }

        return clist_xtrct(n, NULL);
}


/* Create a node at the end of the list */
int clist_mkend(CLIST **l, void *d) {
        /* Add a node to the start of the list */
        int r = clist_mknod(l, d);
        if (r < 0)
                return r;

        /* Move the list pointer back */
        *l = (*l)->n;

        return 0;
}


/* Empty a list, optionally freeing the pointed-to data as well. */
int clist_rmall(CLIST **l, int d, void (*f)(void **)) {
        while (*l != NULL)
                clist_rmnod(l, d, f);
```

```
        return 0;
}
```

C.4 clist.h

```
#ifndef __CLIST_H__
#define __CLIST_H__


/*
 * Local headers
 */

#include "shlib.h"


/*
 * Local definitions
 */

/* Circular list node definition */
typedef struct clist {
        struct clist *p;        /* Previous node */
        struct clist *n;        /* Next node */
        void *d;                /* Data pointer */
} CLIST;


/*
 * Local macros
 */

/* Struct member access macros */
#define CLD(l)                                  ((CLIST *)(l))->d
#define CLN(l)                                  ((CLIST *)(l))->n
#define CLP(l)                                  ((CLIST *)(l))->p

/* A loop helper macro */
#define CLISTFOR(l, i) \
        for ((i) = (l); (i) != NULL; (i) = (CLN(i) == (l))?NULL:CLN(i))

/* A wrapper around clist_mknod() */
#define CLSNOD(c, d, l, e, m, ...)              { \
        r = clist_mknod((CLIST **)(void *)(c), (d)); \
        ERRCND(r != 0, l, e, m, ##__VA_ARGS__); \
}

/* A wrapper around clist_mkend() */
#define CLSEND(c, d, l, e, m, ...)              { \
        r = clist_mkend((CLIST **)(void *)(c), (d)); \
        ERRCND(r != 0, l, e, m, ##__VA_ARGS__); \
}

/* A wrapper around clist_rmall() */
#define CLSDEL(c, d) \
        clist_rmall((CLIST **)(void *)(c), (d), NULL)

/*
 * A CALLOC() variant that records the resulting buffer addresses. All the
 * allocated buffers are recorded in reverse order, so that any references
 * will not be broken while freeing.
 */
#define RALLOC(p, n, s, r, l, e, m, ...)        { \
        CALLOC(p, n, s, l, e, m, ##__VA_ARGS__); \
        CLSNOD(r, p, l, e, m, ##__VA_ARGS__); \
}

/* Record-keeping macro variants */
#define MKROBJ(p, t, v, r, l, e, m, ...)        { \
        MKOBJ(p, t, v, l, e, m, ##__VA_ARGS__); \
        CLSNOD(r, p, l, e, m, ##__VA_ARGS__); \
}
```

```
#define MKRPTR(p, o, r, l, e, m, ...) \
        MKROBJ(p, void *, o, r, l, e, m, ##__VA_ARGS__)
#define MKRSTR(p, r, l, e, n, f, ...)   { \
        MKSTR(p, l, e, n, f, ##__VA_ARGS__); \
        CLSNOD(r, p, l, e, "unable to register string buffer"); \
}

/*
 * Local functions
 */

int clist_insrt(CLIST **, CLIST *);
int clist_mknod(CLIST **, void *);
int clist_xtrct(CLIST **, CLIST **);
int clist_rmnod(CLIST **, int, void (*)(void **));
int clist_mkend(CLIST **, void *);
int clist_rmall(CLIST **, int, void (*)(void **));


#endif /* __CLIST_H__ */
```

C.5 cmenu.c

Περιγραφή: Συναρτήσεις για τη δημιουργία του περιβάλλοντος χρήστη του v1394

```
#include "cmenu.h"


/* Input/Output selection screen */
int cmenu_iosel(FPDLD *o) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND(o == NULL, e00, -1, "invalid pointer");

        /* Menu definition */
        void *d = PAR(
                PAR(
                        XITEMDEF("Input:  ", "IEEE 1394 Asynchronous reader  ",
                                "A video source that draws data from a remote\n"
                                "IEEE 1394 node using asynchronous operations",
                                fproc_fwrd),
                        XITEMDEF("Input:  ", "IEEE 1394 Isochronous receiver ",
                                "A video source that receives data broadcast\n"
                                "to an IEEE 1394 isochronous channel",
                                fproc_fwir),
                        XITEMDEF("Input:  ", "IEEE 1394 Local buffer         ",
                                "A video source that draws data from a local\n"
                                "buffer that is remotely accessed through an\n"
                                "IEEE 1394 bus",
                                fproc_fwar),
                        XITEMDEF("Input:  ", "File reader                    ",
                                "A video source that draws data from a file",
                                fproc_fsrd),
                        XITEMDEF("Input:  ", "Pattern generator              ",
                                "A simple pattern generator for testing",
                                fproc_patt),
                        XITEMDEF("Input:  ", "Zero                           ",
                                "A video source that generates black frames",
                                fproc_zero),
                        XITEMDEF("Input:  ", "NULL                           ",
                                "A video source that produces no data",
                                fproc_null)
                ),
                PAR(
                        XITEMDEF("Output: ", "Monitor                        ",
                                "A video sink that sends data to a visible\n"
                                "window using the SDL library facilities",
                                fproc_disp),
                        XITEMDEF("Output: ", "IEEE 1394 Asynchronous writer  ",
                                "A video output that sends data to a remote\n"
                                "IEEE 1394 node using asynchronous operations",
                                fproc_fwwr),
                        XITEMDEF("Output: ", "IEEE 1394 Isochronous broadcast",
                                "A video output that broadcasts data using an\n"
                                "IEEE 1394 isochronous channel",
                                fproc_fwiw),
                        XITEMDEF("Output: ", "IEEE 1394 Local buffer         ",
                                "A video sink that writes all data to a local\n"
                                "buffer that is remotely accessed through an\n"
                                "IEEE 1394 bus",
                                fproc_fwaw),
                        XITEMDEF("Output: ", "File writer                    ",
                                "A video sink that writes all data to a file",
                                fproc_fswr),
                        XITEMDEF("Output: ", "NULL                           ",
                                "A video sink that ignores all data",
                                fproc_null)
                )
```

```
        );

        /* Page header and footer */
        HDR("Select video Input/Output");
        STS("Enter:Accept    R:Restart    Q:Quit");

        /* Menu creation */
        MENU *m = NULL;
        r = cterm_xmenu(wi, &m, d);
        CNDJMP(r < 0, e00, -1);

        /* The keyboard input loop */
        i0 = KEY_HOME;
        while ((i0 != 'q') && (i0 != 'r') && (i0 != '\n')) {
                cterm_xmenu_kdrv(m, i0, &p0);
                cterm_xmenu_detail(m, &s0);
                OUT("%s", s0);

                i0 = GETCH;
        }
        CNDJMP(i0 == 'q', e01, 1);
        CNDJMP(i0 == 'r', e01, 2);

        /* Act on the user selection */
        XMENU *x = menu_userptr(m);
        o->d0.f = item_userptr(x->i[0].m[x->i[0].i]);
        o->d1.f = item_userptr(x->i[1].m[x->i[1].i]);

e01:
        cterm_xmenu_free(&m, 0);
e00:
        return R;
}


/* Get IEEE 1394 asynchronous transaction parameters */
int cmenu_fwaxp(int is, int in, int *s, int *n, int *o, int *g) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND((s == NULL) || (n == NULL), e00, -1, "invalid pointer");

        /* Menu definition */
        void *d = PAR(
                XITEMINT("Maximum packet size: ", "%4i", "", IAR(is), IAR(4),
                        IAR(MAXBLK), NULL, IAR(4)),
                PAR(
                        XITEMDEF("Set gap count:         ", "Yes", "",
                                NULL),
                        XITEMDEF("Set gap count:         ", "No ", "",
                                NULL)
                ),
                XITEMINT("            gap_count: ", "%2i ",
                        "   0: perform automatic gap count optimisation\n"
                        "1-63: Set the gap count accordingly",
                        IAR(0), IAR(0), IAR(63)),
                XITEMINT("Concurrent packets : ", "%4i", "", IAR(in), IAR(1),
                        IAR(128), NULL, IAR(1))
        );

        /* Page footer */
        STS("Enter:Accept    R:Restart    Q:Quit");

        /* Menu creation */
        MKXMENU(m, d, &p0, e00, e01);

        /* Act on the user selection */
        XMENU *x = menu_userptr(m);
        *s = XITEMINT_CUR(m, 0);
        *o = (x->i[1].i == 0);
        *g = XITEMINT_CUR(m, 2);
```

```
        *n = XITEMINT_CUR(m, 3);

e01:
        cterm_xmenu_free(&m, 0);
e00:
        return R;
}


/* Get IEEE 1394 isochronous transaction parameters */
int cmenu_fwixp(int is, int in, int *s, int *n, int *o, int *g) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND((s == NULL) || (n == NULL), e00, -1, "invalid pointer");

        /* Menu definition */
        void *d = PAR(
                XITEMINT("Maximum packet size: ", "%4i", "", IAR(is), IAR(8),
                        IAR(MAXISO), NULL, IAR(4)),
                PAR(
                        XITEMDEF("Set gap count:         ", "Yes", "",
                                NULL),
                        XITEMDEF("Set gap count:         ", "No ", "",
                                NULL)
                ),
                XITEMINT("            gap_count: ", "%2i ",
                        "   0: perform automatic gap count optimisation\n"
                        "1-63: Set the gap count accordingly",
                        IAR(0), IAR(0), IAR(63)),
                XITEMINT("Buffered packets   : ", "%4i",
                        "0: automatic selection",
                        IAR(in), IAR(1), IAR(16000), NULL, IAR(1))
        );

        /* Page footer */
        STS("Enter:Accept    R:Restart    Q:Quit");

        /* Menu creation */
        MKXMENU(m, d, &p0, e00, e01);

        /* Act on the user selection */
        XMENU *x = menu_userptr(m);
        *s = XITEMINT_CUR(m, 0);
        *o = (x->i[1].i == 0);
        *g = XITEMINT_CUR(m, 2);
        *n = XITEMINT_CUR(m, 3);

e01:
        cterm_xmenu_free(&m, 0);
e00:
        return R;
}


/* Get input parameters */
int cmenu_iprop(FPDLD *o) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND(o == NULL, end, -1, "invalid pointer");

        /* Act on the input processor selection */
        if (o->d0.f == fproc_fwrd) {
                HDR("Select IEEE 1394 input node:");
                OUT("");
                r = cbase_n1394(NULL, NULL, &i0, &u32, &u64, NULL, 1);
                XMCHK(r, end);

                o->d0.o.u0 = i0;
                o->d0.o.u1 = u32;
```

```
        o->d0.o.u2 = u64;

        /* Attempt automatic parameter detection */
        r = fproc_fwxx_cfrom_get(i0, u32, u64, 1, 0, &(o->d0.o.x),
                 &(o->d0.o.y), &(o->d0.o.b), &(o->d0.i), &(o->d0.o.u3));
        CNDJMP(r < 0, end, -1);
        if (r == 1) {
                o->d0.o.x = 0;
                o->d0.o.y = 0;
                o->d0.o.b = 0;
                o->d0.i = 0;
                o->d0.o.u3 = 0;

                HDR("Remote IEEE 1394 input buffer address:");
                OUT("");
                r = cbase_a1394(DFL_AMR_R, &(o->d0.o.u3));
                CNDJMP(r < 0, end, -1);
        }

        /* Transaction parameters */
        HDR("Select IEEE 1394 asynchronous read parameters:");
        OUT("");
        r = cmenu_fwaxp(MAXBLK, 4, &i0, &i1, &i2, &i3);
        XMCHK(r, end);
        o->d0.o.u4 = i0;
        o->d0.o.u5 = i1;
        o->d0.o.u6 = i2;
        o->d0.o.u7 = i3;
} else if (o->d0.f == fproc_fwir) {
        HDR("Select IEEE 1394 input:");
        OUT("");
        r = cbase_n1394(NULL, NULL, &i0, &u32, &u64, NULL, 0);
        XMCHK(r, end);

        o->d0.o.u0 = i0;

        /* Attempt automatic parameter detection */
        r = fproc_fwxx_cfrom_get(i0, u32, u64, 1, 1, &(o->d0.o.x),
                 &(o->d0.o.y), &(o->d0.o.b), &(o->d0.i), &(o->d0.o.u1));
        CNDJMP(r < 0, end, -1);
        if (r == 1) {
                o->d0.o.x = 0;
                o->d0.o.y = 0;
                o->d0.o.b = 0;
                o->d0.i = 0;
                o->d0.o.u1 = 0;

                HDR("Select isochronous reception channel:");
                OUT("");
                r = cbase_ichan(0, &i0);
                XMCHK(r, end);

                o->d0.o.u1 = i0;
        }

        /* Transaction parameters */
        HDR("Select IEEE 1394 isochronous reception parameters:");
        OUT("");
        r = cmenu_fwixp(MAXISO, 200, &i0, &i1, &i2, &i3);
        XMCHK(r, end);
        o->d0.o.u2 = i0;
        o->d0.o.u3 = i1;
        o->d0.o.u4 = i2;
        o->d0.o.u5 = i3;
} else if (o->d0.f == fproc_fwar) {
        HDR("Select IEEE 1394 input:");
        OUT("");
        r = cbase_n1394(NULL, NULL, &i0, &u32, &u64, NULL, 0);
        XMCHK(r, end);

        o->d0.o.u0 = i0;
```

```
                HDR("IEEE 1394 ARM input buffer address:");
                OUT("");
                r = cbase_a1394(DFL_AMR_W, &(o->d0.o.u1));
                CNDJMP(r < 0, end, -1);
        } else if (o->d0.f == fproc_fsrd) {
                HDR("Enter input filename");
                OUT("");
                r = cbase_fname("", &(o->d0.o.f));
                CNDJMP(r < 0, end, -1);
        } else if ((o->d0.f == fproc_patt) || (o->d0.f == fproc_zero) ||
                        (o->d0.f == fproc_null)) {
                goto end;
        }

end:
        return R;
}


/* Get output parameters */
int cmenu_oprop(FPDLD *o) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND(o == NULL, end, -1, "invalid pointer");

        /* Act on the output processor selection */
        if (o->d1.f == fproc_fwwr) {
                HDR("Select IEEE 1394 output node:");
                OUT("");
                r = cbase_n1394(NULL, NULL, &i0, &u32, &u64, NULL, 1);
                XMCHK(r, end);

                o->d1.o.u0 = i0;
                o->d1.o.u1 = u32;
                o->d1.o.u2 = u64;

                /* Attempt automatic parameter detection */
                r = fproc_fwxx_cfrom_get(i0, u32, u64, 0, 0, &(o->d1.o.x),
                         &(o->d1.o.y), &(o->d1.o.b), &(o->d1.i), &(o->d1.o.u3));
                CNDJMP(r < 0, end, -1);
                if (r == 1) {
                        o->d1.o.x = 0;
                        o->d1.o.y = 0;
                        o->d1.o.b = 0;
                        o->d1.i = 0;
                        o->d1.o.u3 = 0;

                        HDR("Remote IEEE 1394 output buffer address:");
                        OUT("");
                        r = cbase_a1394(DFL_AMR_W, &(o->d1.o.u3));
                        CNDJMP(r < 0, end, -1);
                }

                /* Transaction parameters */
                HDR("Select IEEE 1394 asynchronous write parameters:");
                OUT("");
                r = cmenu_fwaxp(MAXBLK, 4, &i0, &i1, &i2, &i3);
                XMCHK(r, end);
                o->d1.o.u4 = i0;
                o->d1.o.u5 = i1;
                o->d1.o.u6 = i2;
                o->d1.o.u7 = i3;
        } else if (o->d1.f == fproc_fwiw) {
                HDR("Select IEEE 1394 output:");
                OUT("");
                r = cbase_n1394(NULL, NULL, &i0, &u32, &u64, NULL, 0);
                XMCHK(r, end);

                o->d1.o.u0 = i0;
```

```
                /* Attempt automatic parameter detection */
                r = fproc_fwxx_cfrom_get(i0, u32, u64, 0, 1, &(o->d1.o.x),
                         &(o->d1.o.y), &(o->d1.o.b), &(o->d1.i), &(o->d1.o.u1));
                CNDJMP(r < 0, end, -1);
                if (r == 1) {
                        o->d1.o.x = 0;
                        o->d1.o.y = 0;
                        o->d1.o.b = 0;
                        o->d1.i = 0;
                        o->d1.o.u1 = 0;

                        HDR("Select isochronous transmission channel:");
                        OUT("");
                        r = cbase_ichan(0, &i0);
                        XMCHK(r, end);

                        o->d1.o.u1 = i0;
                }

                /* Transaction parameters */
                HDR("Select IEEE 1394 isochronous transmission parameters:");
                OUT("");
                r = cmenu_fwixp(MAXISO, 200, &i0, &i1, &i2, &i3);
                XMCHK(r, end);
                o->d1.o.u2 = i0;
                o->d1.o.u3 = i1;
                o->d1.o.u4 = i2;
                o->d1.o.u5 = i3;
        } else if (o->d1.f == fproc_fwaw) {
                HDR("Select IEEE 1394 output:");
                OUT("");
                r = cbase_n1394(NULL, NULL, &i0, &u32, &u64, NULL, 0);
                XMCHK(r, end);

                o->d1.o.u0 = i0;

                HDR("IEEE 1394 ARM output buffer address:");
                OUT("");
                r = cbase_a1394(DFL_AMR_R, &(o->d1.o.u1));
                CNDJMP(r < 0, end, -1);
        } else if (o->d1.f == fproc_fswr) {
                HDR("Enter output filename");
                OUT("");
                r = cbase_fname("", &(o->d1.o.f));
                CNDJMP(r < 0, end, -1);
        } else if (o->d1.f == fproc_null) {
                goto end;
        }

end:
        return R;
}


/* Get frame parameters */
int cmenu_frpar(FPDLD *o) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND(o == NULL, e00, -1, "invalid pointer");

        /* Pre-defined resolution array */
        int **t = TAR(int *,
                IAR(320, 240),
                IAR(640, 480),
                IAR(720, 576),
                IAR(800, 600),
                IAR(1024, 768),
                IAR(0, 0)
        );
```

```
        /* Menu definition */
        void *d = PAR(
                PAR(
                        XITEMDEF("Resolution:            ", "320x240  ", "",
                                NULL),
                        XITEMDEF("Resolution:            ", "640x480  ", "",
                                NULL),
                        XITEMDEF("Resolution:            ", "720x576  ", "",
                                NULL),
                        XITEMDEF("Resolution:            ", "800x600  ", "",
                                NULL),
                        XITEMDEF("Resolution:            ", "1024x768 ", "",
                                NULL),
                        XITEMDEF("Resolution:            ", "Custom   ", "",
                                NULL)
                ),
                XITEMINT("                       ", "X: %4i  ", "", IAR(640),
                        IAR(320), IAR(3200)),
                XITEMINT("                       ", "Y: %4i  ", "", IAR(480),
                        IAR(240), IAR(2400)),
                PAR(
                        XITEMDEF("Bits per pixel:        ", "  15     ",
                                "5:5:5 pixel format", IAR(15)),
                        XITEMDEF("Bits per pixel:        ", "  16     ",
                                "5:6:5 pixel format", IAR(16)),
                        XITEMDEF("Bits per pixel:        ", "  24     ",
                                "8:8:8 pixel format", IAR(24)),
                        XITEMDEF("Bits per pixel:        ", "  32     ",
                                "8:8:8:8 pixel format", IAR(32))
                ),
                XITEMINT(        "Frame interval (usec): ", "%7i  ",
                        "Minimum time interval between consecutive frames.\n"
                        "It is specified in microseconds - a zero value will\n"
                        "disable the interval timing mechanism.",
                        IAR(0), IAR(0), IAR(2000000), NULL, IAR(10))
        );

        /* Page header and footer */
        HDR("Select frame parameters:");
        OUT("");
        STS("Enter:Accept    R:Restart    Q:Quit");

        /* Menu creation */
        MKXMENU(m, d, &p0, e00, e01);

        /* Act on the user selection */
        XMENU *x = menu_userptr(m);
        o->d0.o.x = t[x->i[0].i][0];
        o->d0.o.y = t[x->i[0].i][1];
        if ((o->d0.o.x == 0) || (o->d0.o.y == 0)) {
                o->d0.o.x = XITEMINT_CUR(m, 1);
                o->d0.o.y = XITEMINT_CUR(m, 2);
        }
        o->d0.o.b = OBJ(int, item_userptr(x->i[3].m[x->i[3].i]));
        o->d0.i = XITEMINT_CUR(m, 4);

        /* Processor options should be in sync */
        o->d1.o.x = o->d0.o.x;
        o->d1.o.y = o->d0.o.y;
        o->d1.o.b = o->d0.o.b;
        o->d1.i = o->d0.i;

e01:
        cterm_xmenu_free(&m, 0);
e00:
        return R;
}


/* Get operating parameters */
```

```
int cmenu_fppar(FPDLD *o) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND(o == NULL, e00, -1, "invalid pointer");

        /* Option matching */
        CMPEQ(o->d0.o.x, o->d1.o.x, opt);
        CMPEQ(o->d0.o.y, o->d1.o.y, opt);
        CMPEQ(o->d0.o.b, o->d1.o.b, opt);
        CMPEQ(o->d0.o.c, o->d1.o.b, opt);
        CMPEQ(o->d0.i, o->d1.i, opt);

        /* Request missing parameters */
        if ((o->d0.o.x == 0) || (o->d0.o.y == 0)) {
                r = cmenu_frpar(o);
                XMCHK(r, e00);
        }

        /* Menu definition */
        void *d = PAR(
                XITEMINT("Frame buffers:       ", "            %2.2i", "",
                        IAR(5), IAR(1), IAR(100)),
                XITEMINT("Maximum frame count: ", "%14i",
                        "The maximum number of frames to process.\n"
                        "A zero value disables the limit.",
                        IAR(0), IAR(0), IAR(INT_MAX))
        );

        /* Frame loop parameters */
        HDR("Select frame processing parameters:");
        OUT("");
        STS("Enter:Accept    R:Restart    Q:Quit");

        /* Menu creation */
        MKXMENU(m, d, NULL, e00, e01);

        /* Act on the user selection */
        o->n = XITEMINT_CUR(m, 0);
        o->d0.m = XITEMINT_CUR(m, 1);
        o->d1.m = o->d0.m;

e01:
        cterm_xmenu_free(&m, 0);
e00:
        return R;

opt:
        HDR("");
        OUT("ERROR: Input/Output processor option mismatch!");
        STS("");
        getch();
        return -1;
}
```

C.6 cmenu.h

```
#ifndef __CMENU_H__
#define __CMENU_H__


/*
 * Local headers
 */

#include "cbase.h"


/*
 * Local constants
 */

/* Default readable ARM buffer address within the private CSR space */
#define DFL_AMR_R       0xffffe0000000ULL

/* Default writable ARM buffer address within the private CSR space */
#define DFL_AMR_W       0xffffe8000000ULL


/*
 * Local functions
 */

int cmenu_iosel(FPDLD *);
int cmenu_iprop(FPDLD *);
int cmenu_oprop(FPDLD *);
int cmenu_fppar(FPDLD *);


#endif /* __CMENU_H__ */
```

C.7 cterm.c

Περιγραφή: Συναρτήσεις χειρισμού του τερματικού, βασισμένες στη βιβλιοθήκη ncurses

```
#include "cterm.h"


/* The areas of the terminal interface */
WINDOW *wh = NULL, *wi = NULL, *ww = NULL, *wo = NULL, *ws = NULL;


/* Procure a blank line */
int cterm_lncr(WINDOW *w, int l) {
        FUNCINIT;

        /* Move the cursor to the start of the line */
        r = wmove(w, l, 0);
        NCHECK(end, "unable to move cursor to line %i", l);

        /* Clear the line */
        r = wclrtoeol(w);
        NCHECK(end, "unable to clear line %i", l);

        /* Refresh window */
        wrefresh(w);

end:
        return R;
}


/* Shut down the terminal facilities */
int cterm_stop() {
        FUNCINIT;

        WFREE(wh);
        WFREE(wi);
        WFREE(ww);
        WFREE(wo);
        WFREE(ws);

        r = endwin();
        NCHECK(end, "error leaving visual mode");

end:
        return R;
}


/* Initialize the terminal */
int cterm_init() {
        FUNCINIT;

        /* Initialize the curses library */
        initscr();

        /* Make the cursor invisible */
        curs_set(0);

        /* Put the terminal in raw input mode */
        cbreak();

        /* Set 8-bit terminal mode */
        meta(stdscr, TRUE);

        /* Do not echo keystrokes */
        noecho();
```

```
/* Allow for function keys */
keypad(stdscr, TRUE);

/* Color support */
start_color();

/* Color definitions */
init_pair(C_HDR, COLOR_WHITE, COLOR_RED);
init_pair(C_INP, COLOR_GREEN, COLOR_BLUE);
init_pair(C_WSP, COLOR_WHITE, COLOR_RED);
init_pair(C_OUT, COLOR_GREEN, COLOR_BLUE);
init_pair(C_STS, COLOR_WHITE, COLOR_RED);

/* The window separator line */
u32 = LINES / 3 - 2;

/* Create the interface windows */
wh = newwin(1, COLS, 0, 0);
MCHECK(wh, end, -1, "unable to create header window");
wi = newwin(u32 - 3, COLS, 2, 0);
MCHECK(wi, end, -1, "unable to create input window");
ww = newwin(1, COLS, u32, 0);
MCHECK(ww, end, -1, "unable to create separator window");
wo = newwin(LINES - u32 - 2, COLS, u32 + 1, 0);
MCHECK(wo, end, -1, "unable to create output window");
ws = newwin(1, COLS, LINES - 1, 0);
MCHECK(ws, end, -1, "unable to create status window");

/* Window settings */
keypad(wh, TRUE);
scrollok(wh, TRUE);
keypad(wi, TRUE);
scrollok(wi, TRUE);
keypad(ww, TRUE);
scrollok(ww, TRUE);
keypad(wo, TRUE);
scrollok(wo, TRUE);
keypad(ws, TRUE);
scrollok(ws, TRUE);

/* Color support */
r = wattron(wh, COLOR_PAIR(C_HDR));
NCHECK(end, "unable to set color for header window");
wbkgd(wh, COLOR_PAIR(C_HDR));
r = wattron(wi, COLOR_PAIR(C_INP));
NCHECK(end, "unable to set color for input window");
wbkgd(wi, COLOR_PAIR(C_INP));
r = wattron(ww, COLOR_PAIR(C_WSP));
NCHECK(end, "unable to set color for separator window");
wbkgd(ww, COLOR_PAIR(C_WSP));
r = wattron(wo, COLOR_PAIR(C_OUT));
NCHECK(end, "unable to set color for output window");
wbkgd(wo, COLOR_PAIR(C_OUT));
r = wattron(ws, COLOR_PAIR(C_STS));
NCHECK(end, "unable to set color for status window");
wbkgd(ws, COLOR_PAIR(C_STS));

/* Paint the lines that do not actually belong to a window */
bkgdset(COLOR_PAIR(C_INP));
cterm_lncr(stdscr, 1);
cterm_lncr(stdscr, u32 - 1);
bkgdset(COLOR_PAIR(C_DFL));

/* Initial refresh */
refresh();
wrefresh(wh);
wrefresh(wi);
wrefresh(ww);
wrefresh(wo);
wrefresh(ws);
```

```
end:
        if (R != 0)
                cterm_stop();
        return R;
}


/*
 * Destroy a menu, optionally freeing the items array as well.
 *
 * - If f == 0, only free the menu.
 * - If f == 1, free the menu items as well.
 * - If f == 2, free the menu item strings as well.
 */
int cterm_menu_free(MENU **m, int f) {
        FUNCINIT;

        /* Sanity checking */
        if (m == NULL)
                FATAL;
        if (*m == NULL)
                goto end;

        /* Remove the menu from the screen */
        r = unpost_menu(*m);
        NCHECK(end, "unable to unpost menu");
        wrefresh(menu_win(*m));

        /* Keep the menu item array pointer */
        ITEM **o = menu_items(*m);

        /* Free the menu itself */
        free_menu(*m);
        *m = NULL;

        /* Free the menu items */
        if (f > 0) {
                for (i = 0; o[i] != NULL; ++i) {
                        if (f > 1) {
                                free((void *)item_name(o[i]));
                                free((void *)item_description(o[i]));
                        }

                        free_item(o[i]);
                        o[i] = NULL;
                }

                GFREE(o);
        }

end:
        return R;
}


/* Create a menu in a window */
int cterm_menu(WINDOW *w, MENU **m, ITEM **o) {
        FUNCINIT;

        /* Sanity checking */
        if (o == NULL)
                FATAL;

        /* Allocate the new menu */
        *m = new_menu(o);
        MCHECK(*m, end, -1, "unable to create menu object");

        /* Assign the menu to the window */
        set_menu_win(*m, w);

        /*
```

```
         * Allow the menu to use the whole window height,
         * but create only a single item column on screen.
         */
        getmaxyx(w, i, j);
        set_menu_format(*m, i, 1);

        /* Set menu colors */
        set_menu_fore(*m, (getbkgd(w) & A_COLOR) | A_REVERSE);
        set_menu_back(*m, (getbkgd(w) & A_COLOR));

        /* Draw the menu */
        r = post_menu(*m);
        NCHECK(end, "menu creation failed");
        wrefresh(w);

end:
        if (R != 0)
                cterm_menu_free(m, 0);
        return R;
}


/* Operate a menu using the keyboard */
int cterm_menu_kdrv(MENU *m, int c, void **p) {
        FUNCINIT;

        switch (c) {
                /*
                 * KEY_FIND and KEY_SELECT are returned by several
                 * terminal emulators for the PC keyboard Home and
                 * End keys respectively.
                 */
                case KEY_HOME:
                case KEY_FIND:
                        r = menu_driver(m, REQ_FIRST_ITEM);
                        break;
                case KEY_UP:
                        r = menu_driver(m, REQ_UP_ITEM);
                        break;
                case KEY_DOWN:
                        r = menu_driver(m, REQ_DOWN_ITEM);
                        break;
                case KEY_END:
                case KEY_SELECT:
                        r = menu_driver(m, REQ_LAST_ITEM);
                        break;
                default:
                        goto end;
                        break;
        }

        /* Avoid out-of-range errors */
        if (r == E_REQUEST_DENIED)
                r = 0;

        /* Error checking */
        NCHECK(end, "menu command failed");

        /* Return the user data pointer */
        if (p != NULL)
                *p = item_userptr(current_item(m));

        /* Menu refresh */
        wrefresh(menu_win(m));

end:
        return R;
}


/* Reserved pointer values for marking extended menu items */
```

```
void *XITEM_INT = &XITEM_INT;


/* Destroy an extended menu, optionally freeing the definition objects */
int cterm_xmenu_free(MENU **m, int f) {
        FUNCINIT;

        XMENU *x = NULL;
        ITEM **c = NULL;
        void ****d = NULL;

        /* Sanity checking */
        if (m == NULL)
                FATAL;
        if (*m == NULL)
                goto end;

        x = menu_userptr(*m);
        c = menu_items(*m);
        d = x->d;

        /* More sanity checking */
        if ((x == NULL) || (x->i == NULL) || (c == NULL) || (d == NULL))
                FATAL;

        /* Destroy the menu itself... */
        cterm_menu_free(m, 0);

        /* ...and its support structure */
        GFREE(c);
        for (i = 0; x->i[i].m != NULL; ++i) {
                for(j = 0; x->i[i].m[j] != NULL; ++j)
                        free_item(x->i[i].m[j]);
                GFREE(x->i[i].m);
                if (x->i[i].s[0] == XITEM_INT) {
                        for (j = 0; j < 6; ++j)
                                GFREE(((void **)x->i[i].v)[j]);
                        GFREE(x->i[i].v);
                } else {
                        GFREE(x->i[i].d);
                }
        }
        GFREE(x->i);
        GFREE(x);

        /* Free the definition arrays */
        shlib_ptr_free((void *)(&d), 4, f);

end:
        return R;
}


/* Create an extended functionality menu */
int cterm_xmenu(WINDOW *w, MENU **m, void ****d) {
        FUNCINIT;

        XMENU *x = NULL;
        ITEM **c = NULL;

        /* Sanity checking */
        if (d == NULL)
                FATAL;
        if (d[0] == NULL)
                FATAL;

        /* Count the menu items */
        i0 = 0;
        while (d[i0] != NULL) {
                if (d[i0][0] == NULL)
                        FATAL;
```

```
        ++i0;
}

/* Error message */
s0 = "extended menu creation failed";

/*
 * Build an XMENU construct for the actual menu items
 */

/* Allocate the XMENU object */
RALLOC(x, 1, SZO(XMENU), &p0, end, -1, "%s", s0);

/* Allocate the top-level XITEM array */
RALLOC(x->i, i0 + 1, SZO(XITEM), &p0, end, -1, "%s", s0);

x->d = d;

/* Fill in the array */
for (i = 0; i < i0; ++i) {
        x->i[i].i = 0;
        x->i[i].s = d[i];

        /* Handle active menu items */
        if (d[i][0] == XITEM_INT) {
                if ((d[i][1] == NULL) || (d[i][2] == NULL) ||
                                (d[i][3] == NULL))
                        FATAL;

                /* Allocate the ITEM object array */
                RALLOC(x->i[i].m, 2, SZP(ITEM), &p0, end, -1, "%s", s0);

                /* Allocate the pointer table */
                RALLOC(pp0, 9, SZP(void), &p0, end, -1, "%s", s0);
                x->i[i].v = pp0;

                /*
                 * The menu strings - INT_MIN usually has the maximum
                 * number of digits to print, including the minus sign.
                 */
                MKRSTR(pp0[0], &p0, end, -1, 64, (char *)d[i][1],
                        INT_MIN);
                MKRSTR(pp0[1], &p0, end, -1, 64, (char *)d[i][2],
                        INT_MIN);
                MKRSTR(pp0[2], &p0, end, -1, 64, (char *)d[i][3],
                        INT_MIN);

                /* The integer value and its initial setting */
                i1 = (d[i][4] == NULL)?0:OBJ(int, d[i][4]);
                MKROBJ(pp0[3], int, i1, &p0, end, -1, "%s", s0);

                /* The minimum and maximum values */
                i1 = (d[i][5] == NULL)?INT_MIN:OBJ(int, d[i][5]);
                MKROBJ(pp0[4], int, i1, &p0, end, -1, "%s", s0);
                i1 = (d[i][6] == NULL)?INT_MAX:OBJ(int, d[i][6]);
                MKROBJ(pp0[5], int, i1, &p0, end, -1, "%s", s0);

                /* Sanity checking */
                if ((OBJ(int, pp0[4]) > OBJ(int, pp0[5])) ||
                                (OBJ(int, pp0[3]) < OBJ(int, pp0[4])) ||
                                (OBJ(int, pp0[3]) > OBJ(int, pp0[5])))
                        FATAL;

                /* The update function */
                pp0[6] = IFPTR(d[i][7], d[i][7],
                        (void *)cterm_xitem_int_kdrv);
                pp0[7] = d[i][8];

                /* Run the update function */
                r =((int (*)(int, XITEM *))(pp0[6]))(0, &(x->i[i]));
                NCHECK(end, "%s", s0);
```

```
                        /* The ITEM object */
                        x->i[i].m[0] = new_item(pp0[0], pp0[1]);
                        MCHECK(x->i[i].m[0], end, -1, "%s", s0);
                        CLSNOD(&p1, x->i[i].m[0], end, -1, "%s", s0);

                        x->i[i].d = (void *)(&(pp0[2]));

                        continue;
                }

                /* Count the available alternatives for this menu item */
                i1 = 0;
                while (d[i][i1] != NULL) {
                        if (d[i][i1][0] == NULL)
                                FATAL;
                        ++i1;
                }

                /* Allocate the ITEM object array */
                RALLOC(x->i[i].m, i1 + 1, SZP(ITEM), &p0, end, -1, "%s", s0);

                /* Allocate the detail string pointer array */
                RALLOC(x->i[i].d, i1 + 1, SZP(char), &p0, end, -1, "%s", s0);

                /* Fill in the arrays */
                for (j = 0; j < i1; ++j) {
                        /* The ITEM object */
                        x->i[i].m[j] = new_item(d[i][j][0], d[i][j][1]);
                        MCHECK(x->i[i].m[j], end, -1, "%s", s0);
                        CLSNOD(&p1, x->i[i].m[j], end, -1, "%s", s0);
                        set_item_userptr(x->i[i].m[j], d[i][j][3]);

                        /* The details string */
                        x->i[i].d[j] = d[i][j][2];
                }
                x->i[i].m[i1] = NULL;
        }

        /* The ITEM array for the menu */
        RALLOC(c, i0 + 1, SZP(ITEM), &p0, end, -1, "%s", s0);

        /* Populate the ITEM array */
        for (i = 0; i < i0; ++i)
                c[i] = x->i[i].m[0];
        c[i0] = NULL;

        /* Create the menu */
        r = cterm_menu(w, m, c);
        NCHECK(end, "%s", s0);

        /* Set the menu user data pointer */
        set_menu_userptr(*m, x);

end:
        if (R != 0) {
                CLSDEL(&p0, 1);
                CLISTFOR(p1, p2)
                        free_item(((CLIST *)p2)->d);
        } else {
                CLSDEL(&p0, 0);
        }
        CLSDEL(&p1, 0);
        return R;
}


/* Extended menu keyboard operation */
int cterm_xmenu_kdrv(MENU *m, int c, void **p) {
        FUNCINIT;
```

```
        XMENU *x = NULL;
        ITEM **d = NULL;

        /* Sanity checking */
        if (m == NULL)
                FATAL;

        x = menu_userptr(m);
        d = menu_items(m);

        /* Error message */
        s0 = "extended menu operation failed";

        /* Use the left/right arrow to alter the current item */
        switch (c) {
                case KEY_LEFT:
                case KEY_RIGHT:
                case '-':
                case '+':
                case '=':
                case '[':
                case ']':
                case '{':
                case '}':
                        i0 = item_index(current_item(m));

                        if (x->i[i0].s[0] == XITEM_INT) {
                                pp0 = x->i[i0].v;

                                r =((int (*)(int, XITEM *))(pp0[6]))
                                        (c, &(x->i[i0]));
                                NCHECK(end, "%s", s0);
                        } else if (c == KEY_LEFT) {
                                if (x->i[i0].i == 0)
                                        goto end;

                                --x->i[i0].i;
                        } else if (c == KEY_RIGHT) {
                                if (x->i[i0].m[x->i[i0].i + 1] == NULL)
                                        goto end;

                                ++x->i[i0].i;
                        }

                        d[i0] = x->i[i0].m[x->i[i0].i];

                        r = unpost_menu(m);
                        NCHECK(end, "%s", s0);

                        r = set_menu_items(m, d);
                        NCHECK(end, "%s", s0);

                        r = post_menu(m);
                        NCHECK(end, "%s", s0);

                        r = set_current_item(m, d[i0]);
                        NCHECK(end, "%s", s0);

                        /* Menu refresh */
                        wrefresh(menu_win(m));

                        if (p != NULL)
                                *p = item_userptr(current_item(m));

                        break;
                default:
                        cterm_menu_kdrv(m, c, p);
                        break;
        }

end:
```

```
        return R;
}


/* Get the detail string for the current extended menu item */
int cterm_xmenu_detail(MENU *m, char **d) {
        FUNCINIT;

        XMENU *x = NULL;

        /* Sanity checking */
        if (m == NULL)
                FATAL;

        x = menu_userptr(m);
        i = CURIDX(m);
        *d = x->i[i].d[x->i[i].i];

        return R;
}


/* The XITEM_INT string update function */
int cterm_xitem_int_ustr(XITEM *x, int d, int a) {
        FUNCINIT;

        pp0 = x->v;

        if (pp0[7] != NULL)
                i = OBJ(int, pp0[7]);
        else
                i = 1;

        if (a)
                i0 = d;
        else
                i0 = OBJ(int, pp0[3]) + (d * i);

        i1 = OBJ(int, pp0[4]);
        i2 = OBJ(int, pp0[5]);

        if ((i0 < i1) || (i0 > i2))
                goto end;

        OBJ(int, pp0[3]) = i0;

        sprintf(pp0[0], (char *)x->s[1], i0);
        sprintf(pp0[1], (char *)x->s[2], i0);
        sprintf(pp0[2], (char *)x->s[3], i0);

end:
        return R;
}


/* The default XITEM_INT update function */
int cterm_xitem_int_kdrv(int c, XITEM *x) {
        FUNCINIT;

        switch (c) {
                case 0:
                        cterm_xitem_int_ustr(x, 0, 0);
                        break;
                case KEY_LEFT:
                        cterm_xitem_int_ustr(x, -1, 0);
                        break;
                case KEY_RIGHT:
                        cterm_xitem_int_ustr(x, +1, 0);
                        break;
                case '-':
                        cterm_xitem_int_ustr(x, -10, 0);
```

```
                        break;
                case '+':
                case '=':
                        cterm_xitem_int_ustr(x, +10, 0);
                        break;
                case '[':
                        cterm_xitem_int_ustr(x, -100, 0);
                        break;
                case ']':
                        cterm_xitem_int_ustr(x, +100, 0);
                        break;
                case '{':
                        cterm_xitem_int_ustr(x, -1000, 0);
                        break;
                case '}':
                        cterm_xitem_int_ustr(x, +1000, 0);
                        break;
                default:
                        goto end;
                        break;
        }

end:
        return R;
}


/* A text input box that returns the entered value */
int cterm_inbox(WINDOW *w, char **c, int n, char *s) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND((w == NULL) || (c == NULL), e00, -1, "invalid pointer");

        FIELD *b[2] = { NULL, NULL };
        FORM *f = NULL;

        b[0] = new_field(1, (n <= 0)?(COLS - 8):n, 2, 4, 0, 0);
        ERRCND(b[0] == NULL, e00, -1, "unable to create input field");

        set_field_fore(b[0], (getbkgd(w) & A_COLOR) | A_REVERSE | A_UNDERLINE);
        set_field_back(b[0], (getbkgd(w) & A_COLOR) | A_REVERSE | A_UNDERLINE);
        field_opts_off(b[0], O_AUTOSKIP);
        if (s != NULL) {
                r = set_field_buffer(b[0], 0, s);
                NCHECK(e01, "unable to initialise input field");
        }

        /* Create the form */
        f = new_form(b);
        ERRCND(f == NULL, e01, -1, "unable to create form");
        set_form_win(f, w);

        /* Make the cursor visible */
        curs_set(1);

        /* Display the input box */
        r = post_form(f);
        NCHECK(e02, "unable to display form");
        wrefresh(w);

        /* Configure editing mode */
        form_driver(f, REQ_INS_MODE);

        /* Wait for the Enter key */
        while ((i0 = getch()) != '\n') {
                switch (i0) {
                        case KEY_LEFT:
                                i0 = REQ_PREV_CHAR;
                                break;
                        case KEY_RIGHT:
```

```
                                i0 = REQ_NEXT_CHAR;
                                break;
                        /* The Delete Character key */
                        case KEY_DC:
                                i0 = REQ_DEL_CHAR;
                                break;
                        /* Backspace may return the ASCII DEL code */
                        case 127:
                        case KEY_BACKSPACE:
                                i0 = REQ_DEL_PREV;
                                break;
                        default:
                                break;
                }

                form_driver(f, i0);
                wrefresh(w);
        }

        /* Return the input box content */
        form_driver(f, REQ_VALIDATION);
        *c = strdup(field_buffer(b[0], 0));
        ERRCND(*c == NULL, e03, -1, "unable to copy text input box string");
        shlib_trim(*c);

e03:
        unpost_form(f);
        wrefresh(w);
e02:
        curs_set(0);
        free_form(f);
e01:
        free_field(b[0]);
e00:
        return R;
}
```

C.8 cterm.h

```
#ifndef __CTERM_H__
#define __CTERM_H__


/*
 * Local headers
 */

#include "shlib.h"
#include "clist.h"


/*
 * Sub-system headers
 */

/* ncurses headers */
#include <curses.h>
#include <form.h>
#include <menu.h>


/*
 * Local definitions
 */

/* Color pair numbers */
enum {
        C_DFL,          /* Default color */
        C_HDR,          /* Header color */
        C_INP,          /* User input window color */
        C_WSP,          /* Window separator line color */
        C_OUT,          /* Output window color */
        C_STS           /* Status line color */
};

/* Extended functionality menu item */
typedef struct xitem {
        int i;          /* Current alternative for this menu entry */
        void ***s;      /* Pointer to the second level menu definition array */
        ITEM **m;       /* Menu item alternatives for this menu entry */
        char **d;       /* Detailed information for each alternative */
        void *v;        /* Variable data pointer for internal use */
} XITEM;

/* Extended functionality menu */
typedef struct xmenu {
        XITEM *i;       /* The XITEM array */
        void ****d;     /* Pointer to the menu definition root array */
} XMENU;


/*
 * Local macros
 */

/* Set window content */
#define CNT(w, m, ...)          { \
        int x, y; \
        \
        werase(w); \
        r = wprintw(w, m, ##__VA_ARGS__); \
        \
        getmaxyx(w, y, x); \
        wmove(w, y, x); \
        wrefresh(w); \
}
```

```
/* A conveniency wrapper aroung getch() */
#define GETCH                   (tolower(getch()))

/* Set the header bar content */
#define HDR(m, ...)             CNT(wh, m, ##__VA_ARGS__)

/* Get the index of the current MENU menu item */
#define CURIDX(m)               item_index(current_item((MENU *)m))

/* Error checking for ncurses functions */
#define NCHECK(l, m, ...)       { \
        ERRCND(r != OK, l, -1, m, ##__VA_ARGS__); \
}

/* Set the output window content */
#define OUT(m, ...)             CNT(wo, m, ##__VA_ARGS__)

/* Set the status bar content */
#define STS(m, ...)             CNT(ws, m, ##__VA_ARGS__)

/* Free a window object */
#define WFREE(p)                { \
        if ((p) != NULL) { \
                r = delwin(p); \
                if (r != E_OK) \
                        FATAL; \
        } \
        p = NULL; \
}

/* Set the window separator line content */
#define WSP(m, ...)             CNT(ww, m, ##__VA_ARGS__)

/* XITEM object definition helper macro */
#define XITEMDEF(n, d, l, u)    PAR(n, d, l, u)

/*
 * An XITEMINT active menu item
 *
 * Parameters:
 *
 * 1. name string format
 * 2. description string format
 * 3. detailed description string format
 * 4. initial value (optional)
 * 5. minimum value (optional)
 * 6. maximum value (optional)
 * 7. update function (optional)
 * 8. update function parameter (optional)
 */
#define XITEMINT(n, d, l, ...)  \
        PAR(XITEM_INT, n, d, l, ##__VA_ARGS__, \
                NULL, NULL, NULL, NULL, NULL, NULL)

/* The current value of an XITEMINT active menu item */
#define XITEMINT_CUR(m, n) \
        OBJ(int, ((void **)(((XMENU *)menu_userptr(m))->i[n].v))[3])


/*
 * Local declarations
 */

/*
 * The areas of the terminal interface:
 *
 * - Header line
 * - User input
 * - Separator line
 * - Program output
```

```
 * - Status and help line
 */
extern WINDOW *wh, *wi, *ww, *wo, *ws;

/* Reserved pointer values for marking extended menu items */
extern void *XITEM_INT;


/*
 * Local functions
 */

int cterm_lncr(WINDOW *, int);
int cterm_stop();
int cterm_init();
int cterm_menu_free(MENU **, int);
int cterm_menu(WINDOW *, MENU **, ITEM **);
int cterm_menu_kdrv(MENU *, int, void **);
int cterm_xmenu_free(MENU **, int);
int cterm_xmenu(WINDOW *, MENU **, void ****);
int cterm_xitem_int_ustr(XITEM *, int, int);
int cterm_xmenu_kdrv(MENU *, int, void **);
int cterm_xmenu_detail(MENU *, char **);
int cterm_xitem_int_kdrv(int, XITEM *);
int cterm_inbox(WINDOW *, char **, int, char *);


#endif /* __CTERM_H__ */
```

C.9 fproc.c

Περιγραφή: Ο πυρήνας του κώδικα επεξεργασίας πλαισίων κινούμενης εικόνας

```
#include "fproc.h"


/* The core frame processor routine */
void *fproc_worker(void *p) {
        FUNCINIT;

        FPDAT *d = p;
        FRAME *f = NULL;
        CLIST *n = NULL;

        d->s.fs = d->o.x * d->o.y * OCTS(d->o.b);

        r = (d->f)(FPROC_INIT, NULL, &(d->o), &(d->s));
        CNDJMP(r < 0, e00, -1);

        d->x = 0;

        if (d->i > 0) {
                r = gettimeofday(&tv0, NULL);
                ERRCND(r < 0, e00, -1, "cannot get timestamp");
        }

        while (1) {
                if (d->i > 0) {
                        r = timer_wait(tv0);
                        CNDJMP(r < 0, e01, -1);

                        r = gettimeofday(&tv0, NULL);
                        ERRCND(r < 0, e01, -1, "cannot get timestamp");

                        USECTV(U64, tv0, TVUSEC(U64, tv0) + d->i);
                }

                if (d->x)
                        goto e03;

                r = frame_dequeue(d->qi, &n, &f);
                CNDJMP(r < 0, e01, -1);

                if (f == NULL)
                        goto e02;

                d->s.tq += TVDIFF(U64, FRD(f), FRE(f));

                r = (d->f)(FPROC_PROC, f, &(d->o), &(d->s));
                CNDJMP(r < 0, e02, -1);
                ++(d->s.nf);

                d->s.tf += TVDIFF(U64, FRU(f), FRD(f));

                /* Handle the End-Of-Processing condition */
                if (((d->m > 0) && (d->s.nf >= d->m)) || (r == 1))
                        i0 = 1;

                r = frame_enqueue(d->qo, n, f);
                CNDJMP(r < 0, e01, -1);

                if (i0)
                        goto e03;
        };

        goto e01;

e03:
        f = NULL;
```

```
        n = NULL;
e02:
        r = queue_enqueue(d->qo, n, f);
        CNDJMP(r < 0, e01, -1);
e01:
        r = (d->f)(FPROC_FREE, NULL, &(d->o), &(d->s));
        CNDJMP(r < 0, e00, -1);
e00:
        d->r = R;
        d->x = 1;
        return d;
}


/* Initialise a dual processor loop */
int fproc_dloop(FPDLD *d) {
        FUNCINIT;

        /* Sanity enforcement */
        CNDJMP(d == NULL, opt, -1);
        CMPEQ(d->d0.o.x, d->d1.o.x, opt);
        CMPEQ(d->d0.o.y, d->d1.o.y, opt);
        CMPEQ(d->d0.o.b, d->d1.o.b, opt);
        CMPEQ(d->d0.o.c, d->d1.o.c, opt);

        r = queue_init(&(d->q0));
        CNDJMP(r < 0, e00, -1);
        r = queue_init(&(d->q1));
        CNDJMP(r < 0, e01, -1);

        CALLOC(d->f, d->n, SZP(FRAME), e01, -1, "cannot allocate frame array");

        for (i = 0; i < d->n; ++i) {
                r = frame_init(&(d->f[i]), d->d0.o.x, d->d0.o.y, d->d0.o.b,
                        d->d0.o.c);
                CNDJMP(r < 0, e02, -1);

                r = frame_enqueue(&(d->q0), NULL, d->f[i]);
                CNDJMP(r < 0, e02, -1);
        }

        /* Processor setup */
        d->d0.qi = &(d->q0);
        d->d0.qo = &(d->q1);
        d->d1.qi = &(d->q1);
        d->d1.qo = &(d->q0);

        /* Create joinable threads */
        r = pthread_attr_init(&(d->a));
        ERRCND(r < 0, e02, -1, "cannot initialise thread attributes");
        r = pthread_attr_setdetachstate(&(d->a), PTHREAD_CREATE_JOINABLE);
        ERRCND(r < 0, e03, -1, "cannot initialise thread attributes");

        /* Start processor threads */
        r = pthread_create(&(d->t0), &(d->a), fproc_worker, &(d->d0));
        ERRCND(r < 0, e03, -1, "cannot launch processor thread");
        r = pthread_create(&(d->t1), &(d->a), fproc_worker, &(d->d1));
        ERRCND(r < 0, e04, -1, "cannot launch processor thread");

        return R;

e04:
        d->d0.x = 1;
        queue_enqueue(&(d->q0), NULL, NULL);
        pthread_join(d->t0, &p0);
e03:
        pthread_attr_destroy(&(d->a));
e02:
        for (i = 0; i < d->n; ++i)
                frame_free(&(d->f[i]));
        GFREE(d->f);
```

```
e01:
        queue_free(&(d->q1), 0);
e00:
        queue_free(&(d->q0), 0);
        return R;

opt:
        ERRMSG("processor configuration error");
        return -1;
}


/* Terminate the dual processor loop */
int fproc_dloop_stop(FPDLD *d, int f) {
        FUNCINIT;

        d->d0.x = 1;
        d->d1.x = 1;

        queue_enqueue(&(d->q0), NULL, NULL);
        queue_enqueue(&(d->q1), NULL, NULL);

        (d->d0.f)(FPROC_STOP, NULL, &(d->d0.o), &(d->d0.s));
        (d->d1.f)(FPROC_STOP, NULL, &(d->d1.o), &(d->d1.s));

        pthread_join(d->t0, &p0);
        pthread_join(d->t1, &p0);

        pthread_attr_destroy(&(d->a));

        for (i = 0; i < d->n; ++i)
                frame_free(&(d->f[i]));
        GFREE(d->f);

        queue_free(&(d->q1), 0);
        queue_free(&(d->q0), 0);

        if (f) {
                GFREE(d->d0.o.f);
                GFREE(d->d1.o.f);
        }

        return R;
}
```

C.10 fproc.h

```
#ifndef __FPROC_H__
#define __FPROC_H__


/*
 * Local headers
 */

#include "shlib.h"
#include "frame.h"
#include "timer.h"


/*
 * Local definitions
 */

/* Frame processor function type */
#define FPFUN(I) \
        int I(int, FRAME *, FPOPT *, FPSTT *)

/* Frame processor function modes */
enum {
        FPROC_INIT,
        FPROC_PROC,
        FPROC_FREE,
        FPROC_STOP
};

/* Frame processor options struct type */
typedef struct fpopt {
        char *f;                /* Filename for the fs* processors */
        int x;                  /* Pixels per line */
        int y;                  /* Pixels per column */
        int b;                  /* Bits per pixel */
        int c;                  /* Color format */
        U64 u0;                 /* General purpose unsigned integer */
        U64 u1;                 /* General purpose unsigned integer */
        U64 u2;                 /* General purpose unsigned integer */
        U64 u3;                 /* General purpose unsigned integer */
        U64 u4;                 /* General purpose unsigned integer */
        U64 u5;                 /* General purpose unsigned integer */
        U64 u6;                 /* General purpose unsigned integer */
        U64 u7;                 /* General purpose unsigned integer */
} FPOPT;

/* Frame processor state struct type */
typedef struct fpstt {
        int fs;                 /* Frame size in bytes */
        U64 tb;                 /* Transferred byte count */
        U64 it;                 /* Immediate response transaction */
        U64 dt;                 /* Delayed response transaction */
        U64 pt;                 /* Postponed transaction */
        U64 et;                 /* Erroneous transaction */
        U64 nf;                 /* Number of processed frames */
        U64 tf;                 /* Total frame processing time (usec) */
        U64 tq;                 /* Total frame queueing time (usec) */
        U64 tp;                 /* Total packet processing time (usec) */
        int i;                  /* General purpose integer */
        struct timeval t;       /* General purpose timestamp */
        void *p0;               /* General purpose pointer */
        void *p1;               /* General purpose pointer */
        U16 u0;                 /* General purpose unsigned integer */
        U16 u1;                 /* General purpose unsigned integer */
        U64 u2;                 /* General purpose unsigned integer */
        U64 u3;                 /* General purpose unsigned integer */
} FPSTT;
```

```
/* Frame processor data compound */
typedef struct fpdat {
        FPFUN((*f));            /* The actual frame processor function */
        QUEUE *qi;              /* Incoming frame queue */
        QUEUE *qo;              /* Outgoing frame queue */
        FPOPT o;
        FPSTT s;
        int r;                  /* Return value */
        int x;                  /* Processing loop exit flag */
        U64 i;                  /* Minimum interval between frames (usec) */
        U64 m;                  /* Maximum number of processed frames */
} FPDAT;

/* Dual processor loop data compound */
typedef struct fpdld {
        QUEUE q0;
        QUEUE q1;
        pthread_attr_t a;       /* Thread attributes */
        pthread_t t0;           /* Worker thread */
        pthread_t t1;           /* Worker thread */
        FRAME **f;
        FPDAT d0;
        FPDAT d1;
        int n;                  /* Number of frames in the loop */
} FPDLD;


/*
 * Local macros
 */

/* Packet performance counter update */
#define UPCNT(s, f)             { \
        ++((s)->it); \
        (s)->tb += (s)->fs; \
        (s)->tp += TVDIFF(U64, FRU(f), FRD(f)); \
}

/* Format string for statistics output */
#define STFMT(P, S) \
        P "               |   Input    |   Output\n" S \
        P "Throughput:    |            |\n" S \
        P "  Bytes/sec    | %10llu | %10llu\n" S \
        P "  Packets/sec  | %10.2f | %10.2f\n" S \
        P "    Immediate  | %10.2f | %10.2f\n" S \
        P "    Delayed    | %10.2f | %10.2f\n" S \
        P "    Postponed  | %10.2f | %10.2f\n" S \
        P "  Frames/sec   | %10.2f | %10.2f\n" S \
        P "Latency:       |            |\n" S \
        P "  Packet (us)  | %10.2f | %10.2f\n" S \
        P "  Frame  (us)  | %10.2f | %10.2f\n" S \
        P "  Queue  (us)  | %10.2f | %10.2f\n" S \
        P "  Total  (us)  | %10.2f | %10.2f\n" S \
        P "Errors:        |            |\n" S \
        P "  Per frame    | %10.2f | %10.2f\n" S \
        P "  Per second   | %10.2f | %10.2f\n" S \
        P "  Total        | %10llu | %10llu\n" S

/* Arguments for statistics output */
#define STARG(D, U, T) \
        (1000000 * (D).d0.s.tb) / (U), \
        (1000000 * (D).d1.s.tb) / (U), \
        \
        (1000000 * (double)((D).d0.s.it + (D).d0.s.dt + (D).d0.s.pt)) / (T), \
        (1000000 * (double)((D).d1.s.it + (D).d1.s.dt + (D).d1.s.pt)) / (T), \
        \
        (1000000 * (double)(D).d0.s.it) / (T), \
        (1000000 * (double)(D).d1.s.it) / (T), \
        \
        (1000000 * (double)(D).d0.s.dt) / (T), \
```

```
        (1000000 * (double)(D).d1.s.dt) / (T), \
        \
        (1000000 * (double)(D).d0.s.pt) / (T), \
        (1000000 * (double)(D).d1.s.pt) / (T), \
        \
        (1000000 * (double)(D).d0.s.nf) / (T), \
        (1000000 * (double)(D).d1.s.nf) / (T), \
        \
        DFP((D).d0.s.tp, ((D).d0.s.it + (D).d0.s.dt + (D).d0.s.pt)), \
        DFP((D).d1.s.tp, ((D).d1.s.it + (D).d1.s.dt + (D).d1.s.pt)), \
        \
        DFP((D).d0.s.tf, (D).d0.s.nf), \
        DFP((D).d1.s.tf, (D).d1.s.nf), \
        \
        DFP((D).d0.s.tq, (D).d0.s.nf), \
        DFP((D).d1.s.tq, (D).d1.s.nf), \
        \
        DFP(((D).d0.s.tf + (D).d0.s.tq), (D).d0.s.nf), \
        DFP(((D).d1.s.tf + (D).d1.s.tq), (D).d1.s.nf), \
        \
        DFP((D).d0.s.et, (D).d0.s.nf), \
        DFP((D).d1.s.et, (D).d1.s.nf), \
        \
        (1000000 * (double)(D).d0.s.et) / (T), \
        (1000000 * (double)(D).d1.s.et) / (T), \
        \
        (D).d0.s.et, (D).d1.s.et


/*
 * Local functions
 */

#ifndef __FPROC_D__
int fproc_dloop(FPDLD *);
int fproc_dloop_stop(FPDLD *, int);
#else /* __FPROC_D__ */
NULLP(fproc_dloop);
NULLP(fproc_dloop_stop);
#endif /* __FPROC_D__ */


#endif /* __FPROC_H__ */
```

C.11 frame.c

Περιγραφή: Βασικές συναρτήσεις χειρισμού πλαισίων κινούμενης εικόνας

```
#include "frame.h"


/* Create a video frame */
int frame_init(FRAME **f, int x, int y, int b, int c) {
        FUNCINIT;

        /* Sanity checking */
        if (f == NULL)
                ERRJMP(e00, -1, "invalid video frame pointer");

        /* Allocate memory */
        CALLOC(*f, 1, SZO(FRAME), e00, -1, "");
        CALLOC(FRF(*f), 1, x * y * b, e01, -1, "");

        /* Set the frame information fields */
        FRX(*f) = x;
        FRY(*f) = y;
        FRB(*f) = b;
        FRC(*f) = c;

        goto e00;

e01:
        GFREE(*f);
e00:
        return R;
}


/* Destroy a video frame */
int frame_free(FRAME **f) {
        FUNCINIT;

        /* Sanity checking */
        if (f == NULL)
                ERRJMP(end, -1, "invalid video frame pointer");

        if (*f == NULL)
                goto end;

        /* Free allocated memory */
        GFREE(FRF(*f));
        GFREE(*f);

end:
        return R;
}


/* Enqueue a video frame */
int frame_enqueue(QUEUE *q, CLIST *n, FRAME *f) {
        FUNCINIT;

        /* Sanity checking */
        if (q == NULL)
                FATAL;

        if (n != NULL)
                f = CLD(n);

        /* Set the appropriate timestamp */
        if (f != NULL) {
                r = gettimeofday(&(FRE(f)), NULL);
                ERRCND(r != 0, end, -1, "unable to set timestamp");
        }
```

```
        /* Enqueue the video frame */
        r = queue_enqueue(q, n, f);
        ERRCND(r != 0, end, -1, "unable to enqueue video frame");

end:
        return R;
}


/* Dequeue a video frame */
int frame_dequeue(QUEUE *q, CLIST **n, FRAME **f) {
        FUNCINIT;

        /* Sanity checking */
        if (q == NULL)
                FATAL;

        /* Dequeue a video frame */
        r = queue_dequeue(q, n , (void **)f);
        ERRCND(r != 0, end, -1, "unable to dequeue video frame");

        /* Set the appropriate timestamp */
        if (*f != NULL) {
                r = gettimeofday(&(FRD(*f)), NULL);
                ERRCND(r != 0, end, -1, "unable to set timestamp");
        }

end:
        return R;
}
```

C.12 frame.h

```
#ifndef __FRAME_H__
#define __FRAME_H__


/*
 * Local headers
 */

#include "shlib.h"
#include "queue.h"


/*
 * Local definitions
 */

/* The base video frame struct */
typedef struct frame {
        int x;                  /* Pixels per line */
        int y;                  /* Pixels per column */
        int b;                  /* Bits per pixel */
        int c;                  /* Color format */
        char *f;                /* The frame data */
        struct timeval e;       /* Frame enqueueing time */
        struct timeval d;       /* Frame dequeueing time */
        struct timeval u;       /* Frame use end time */
} FRAME;


/*
 * Local macros
 */

/* Struct member access macros */
#define FRX(F)                  ((FRAME *)(F))->x
#define FRY(F)                  ((FRAME *)(F))->y
#define FRB(F)                  ((FRAME *)(F))->b
#define FRC(F)                  ((FRAME *)(F))->c
#define FRF(F)                  ((FRAME *)(F))->f
#define FRE(F)                  ((FRAME *)(F))->e
#define FRD(F)                  ((FRAME *)(F))->d
#define FRU(F)                  ((FRAME *)(F))->u


/*
 * Local functions
 */

int frame_init(FRAME **, int, int, int, int);
int frame_free(FRAME **);
int frame_enqueue(QUEUE *, CLIST *, FRAME *);
int frame_dequeue(QUEUE *, CLIST **, FRAME **);


#endif /* __FRAME_H__ */
```

C.13 l1394.c

Περιγραφή: Πρόγραμμα το οποίο ερευνά όλους τους διαθέσιμους διαύλους IEEE 1394 και εμφανίζει πληροφορίες για τους κόμβους που τους αποτελούν

```
#include "u1394.h"


MAIN {
        MAININIT;

        /* The raw1394 kernel subsystem handle */
        H1394 h = NULL;

        /* The bus list pointers */
        CLIST *l = NULL, *ii = NULL, *jj = NULL, *kk = NULL;

restart:
        /* Collect information on all buses */
        r = u1394_blist(&h, &l, NULL, NULL);
        ERRCHK(r, restart, end);

        /* Display port information */
        MSG("Available IEEE 1394 nodes:");
        CLISTFOR(l, ii) {
                B1394 *b = ii->d;

                /* Print port information */
                MSG("(P%02d) \"%s\"\n\t<N:%02d - L:%02d - BM:%02d - IRM:%02d>",
                        b->p, b->d, b->c, PHY(b->l), PHY(b->b), PHY(b->i));

                /* Print node information */
                CLISTFOR(b->n, jj) {
                        P1394 *p = jj->d;

                        if (p->m == 1) {
                                MSG("\t(N%02d) ID: 0x%06llx", p->n, p->cid);
                                continue;
                        }

                        /* Print general information about the node */
                        MSG("\t(N%02d)\n\t\t<irmc:%d cmc:%d isc:%d bmc:%d "
                                "maxwr:%4d vid:0x%06x cid:0x%010llx>", p->n,
                                p->irmc, p->cmc, p->isc, p->bmc, p->maxwr,
                                p->vid, p->cid);

                        /* Print the textual descriptors of the node */
                        CLISTFOR(p->txt, kk)
                                MSG("\t\t\"%s\"", (char *)(kk->d));
                }
        }

end:
        /* Free memory */
        u1394_blist_free((void *)(&l));
        HFREE(h);
        return R;
}
```

C.14 prcdp.c

Περιγραφή: Κώδικας για την προβολή κινούμενης εικόνας στην οθόνη του υπολογιστή

```
#include "prcdp.h"


/*
 * The display output video frame processor
 *
 * Output the frame on a display
 */
int fproc_disp(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
#ifndef __SDLIB_D__
                        /* Initialise the SDL library */
                        r = SDL_Init(SDL_INIT_VIDEO);
                        ERRCND(r < 0, end, -1, "unable to initialise display");

                        /* Prepare the video output */
                        s->p0 = SDL_SetVideoMode(o->x, o->y, o->b,
                                SDL_SWSURFACE);
                        if (s->p0 == NULL)
                                ERRJMP(end, -1, "unable to initialise display");
#endif

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

#ifndef __SDLIB_D__
                        /* Screen locking */
                        if (SDL_MUSTLOCK(((SDL_Surface *)(s->p0)))) {
                                r = SDL_LockSurface(s->p0);
                                ERRCND(r < 0, end, -1, "failed display lock");
                        }

                        /* Update the video buffer */
                        memcpy(((SDL_Surface *)(s->p0))->pixels, FRF(f), s->fs);

                        /* Unlock the screen */
                        if (SDL_MUSTLOCK(((SDL_Surface *)(s->p0)))) {
                                SDL_UnlockSurface(s->p0);
                        }

                        /* Perform a screen update */
                        SDL_UpdateRect(s->p0, 0, 0, FRX(f), FRY(f));
#endif

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

#ifndef __SDLIB_D__
                        /* Allow the user to terminate the processing */
                        SDL_Event e;
                        while (SDL_PollEvent(&e)) {
                                if (e.type != SDL_KEYDOWN)
                                        continue;
                                switch (e.key.keysym.sym) {
```

```
                                        case SDLK_ESCAPE:
                                        case SDLK_q:
                                                R = 1;
                                                break;
                                        default:
                                                break;
                                }
                        }
#endif

                        UPCNT(s, f);

                        goto end;
                        break;
                case FPROC_FREE:
#ifndef __SDLIB_D__
                        /* Free the screen buffer */
                        SDL_FreeSurface(s->p0);

                        /* Shut down SDL */
                        SDL_Quit();
#endif
                case FPROC_STOP:
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}
```

C.15 prcdp.h

```
#ifndef __PRCDP_H__
#define __PRCDP_H__


/*
 * Local headers
 */

#include "fproc.h"


/*
 * Sub-system headers
 */

/* SDL headers */
#ifndef __SDLIB_D__
#include <SDL.h>
#endif


/*
 * Local functions
 */

#ifndef __FPROC_D__
FPFUN(fproc_disp);
#else /* __FPROC_D__ */
NULLP(fproc_disp);
#endif /* __FPROC_D__ */


#endif /* __PRCDP_H__ */
```

C.16 prcfm.c

Περιγραφή: Κώδικας για την είσοδο και έξοδο κινούμενης εικόνας σε αρχείο, καθώς και για την παραγωγή και κατανάλωση πλαισίων

```
#include "prcfm.h"


/*
 * The NULL video frame processor
 *
 * Do nothing but update the performance timestamps
 */
int fproc_null(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        UPCNT(s, f);

                        goto end;
                        break;
                case FPROC_FREE:
                case FPROC_STOP:
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}


/*
 * The zero video frame processor
 *
 * Produce a zeroed-out frame - in most cases this is an all-black image
 */
int fproc_zero(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Zero-out the frame */
```

```
                        memset(FRF(f), 0, s->fs);

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        UPCNT(s, f);

                        goto end;
                        break;
                case FPROC_FREE:
                case FPROC_STOP:
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}


/*
 * The test pattern video frame generator
 *
 * Generate test frames with easily recognisable patterns
 */
int fproc_patt(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        /* Black and white pixel values */
        s0 = (char []){ 0x00, 0x00, 0x00, 0x00 };
        s1 = (char []){ 0xff, 0xff, 0xff, 0x00 };

        switch (m) {
                case FPROC_INIT:
                        s->i = 0;

                        /* Set the initial timestamp */
                        r = gettimeofday(&(s->t), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Get the current time */
                        r = gettimeofday(&tv0, NULL);
                        ERRCND(r != 0, end, -1, "unable to get timestamp");

                        /* See if we have to switch the pattern */
                        if (TVDIFF(U64, tv0, s->t) > 2000000) {
                                s->i = (s->i + 1) % 4;
                                s->t = tv0;
                        }

                        /* Frame properties */
                        i0 = OCTS(FRB(f));
                        i1 = FRX(f) * i0;
                        i2 = FRY(f) * i1;

                        /* Set the pattern */
                        if (s->i == 0) {
                                /* An all-white frame */
```

```
                                for (i = 0; i < i1; i += i0)
                                        memcpy(FRF(f) + i, s1, i0);
                                for (i = i1; i < i2; i += i1)
                                        memcpy(FRF(f) + i, FRF(f), i1);
                        } else if (s->i == 1) {
                                /* Vertical stripes */
                                for (i = 0; i < FRX(f); ++i) {
                                        p0 = ((i & 0x20) == 0)?s0:s1;
                                        memcpy(FRF(f) + i * i0, p0, i0);
                                }
                                for (i = i1; i < i2; i += i1)
                                        memcpy(FRF(f) + i, FRF(f), i1);
                        } else if (s->i == 2) {
                                /* Horizontal stripes */
                                for (i = 0; i < FRY(f); ++i) {
                                        p0 = FRF(f) + i * i1;
                                        if ((i & 0x1f) == 0) {
                                                p1 = ((i & 0x20) == 0)?s0:s1;
                                                p2 = p0;
                                                for (j = 0; j < FRX(f); ++j)
                                                        memcpy(p2 + j * i0, p1,
                                                                i0);
                                        } else {
                                                memcpy(p0, p2, i1);
                                        }
                                }
                        } else if (s->i == 3) {
                                /* An all-black frame */
                                for (i = 0; i < i1; i += i0)
                                        memcpy(FRF(f) + i, s0, i0);
                                for (i = i1; i < i2; i += i1)
                                        memcpy(FRF(f) + i, FRF(f), i1);
                        } else {
                                FATAL;
                        }

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        UPCNT(s, f);

                        goto end;
                        break;
                case FPROC_FREE:
                case FPROC_STOP:
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}


/*
 * The filesystem writer video frame processor
 *
 * Output the raw video frame data to a file
 */
int fproc_fswr(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
```

```
                        if ((o->f == NULL) || (strlen(o->f) == 0))
                                ERRJMP(end, -1, "invalid filename");

                        s->p0 = fopen(o->f, "w");
                        if (s->p0 == NULL)
                                ERRJMP(end, -1, "cannot open output file");

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Write to the file */
                        r = fwrite(FRF(f), s->fs, 1, s->p0);
                        ERRCND(r < 1, end, -1, "file write failed");

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        UPCNT(s, f);

                        goto end;
                        break;
                case FPROC_FREE:
                        r = fclose(s->p0);
                        if (r != 0)
                                FATAL;
                        s->p0 = NULL;
                case FPROC_STOP:
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}


/*
 * The filesystem reader video frame processor
 *
 * Obtain the raw video frame data from a file
 */
int fproc_fsrd(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
                        if ((o->f == NULL) || (strlen(o->f) == 0))
                                ERRJMP(end, -1, "invalid filename");

                        s->p0 = fopen(o->f, "r");
                        if (s->p0 == NULL)
                                ERRJMP(end, -1, "cannot open input file");

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Read from the file */
                        r = fread(FRF(f), s->fs, 1, s->p0);
```

```
                        CNDJMP(feof(s->p0), end, 1);
                        ERRCND(r < 1, end, -1, "file read failed");

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        UPCNT(s, f);

                        goto end;
                        break;
                case FPROC_FREE:
                        r = fclose(s->p0);
                        if (r != 0)
                                FATAL;
                        s->p0 = NULL;
                case FPROC_STOP:
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}
```

C.17 prcfm.h

```
#ifndef __PRCFM_H__
#define __PRCFM_H__


/*
 * Local headers
 */

#include "fproc.h"


/*
 * Local functions
 */

#ifndef __FPROC_D__
FPFUN(fproc_null);
FPFUN(fproc_zero);
FPFUN(fproc_patt);
FPFUN(fproc_fswr);
FPFUN(fproc_fsrd);
#else /* __FPROC_D__ */
NULLP(fproc_null);
NULLP(fproc_zero);
NULLP(fproc_patt);
NULLP(fproc_fswr);
NULLP(fproc_fsrd);
#endif /* __FPROC_D__ */


#endif /* __PRCFM_H__ */
```

C.18 prcfw.c

Περιγραφή: Συναρτήσεις για τη λήψη και εκπομπή κινούμενης εικόνας μέσω του διαύλου IEEE 1394

```
#include "prcfw.h"


/*
 * The IEEE 1394 video frame output function
 *
 * Output the raw video frame to an IEEE 1394 node using quadlet or block
 * transactions
 *
 * FPOPT fields:
 *
 * - u0: Port
 * - u1: Vendor ID
 * - u2: Chip ID
 * - u3: Address
 * - u4: Maximum packet size
 * - u5: Maximum number of concurrent packets
 * - u6: Set gap count ?
 * - u7: gap_count value
 *
 * FPSTT fields:
 *
 * - p0:  raw1394 handle
 * - u0: Node ID
 */
int fproc_fwwr(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
                        r = u1394_new_handle((void *)(&(s->p0)));
                        ERRCHK(r, end, end);

                        r = u1394_find_id((H1394)(s->p0), o->u0, 1, o->u1,
                                o->u2, &(s->u0));
                        ERRCHK(r, end, end);

                        if (o->u6 != 0) {
                                r = u1394_set_gap_count((H1394)(s->p0), o->u7);
                                ERRCHK(r, end, end);
                        }

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Write to the remote node */
                        r = u1394_write(s->p0, &(s->u0), o->u1, o->u2, o->u3,
                                o->u4, o->u5, s->fs, FRF(f), &(s->tb), &(s->it),
                                &(s->dt), &(s->pt), &(s->et), &(s->tp));
                        ERRCHK(r, end, end);

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        goto end;
                        break;
                case FPROC_FREE:
```

```
                        HFREE(s->p0);
                case FPROC_STOP:
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}


/*
 * The IEEE 1394 video frame input function
 *
 * Obtain the raw video frame from an IEEE 1394 node using quadlet or block
 * transactions
 */
int fproc_fwrd(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
                        r = u1394_new_handle((void *)(&(s->p0)));
                        ERRCHK(r, end, end);

                        r = u1394_find_id((H1394)(s->p0), o->u0, 1, o->u1,
                                o->u2, &(s->u0));
                        ERRCHK(r, end, end);

                        if (o->u6 != 0) {
                                r = u1394_set_gap_count((H1394)(s->p0), o->u7);
                                ERRCHK(r, end, end);
                        }

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Read from the remote node */
                        r = u1394_read(s->p0, &(s->u0), o->u1, o->u2, o->u3,
                                o->u4, o->u5, s->fs, FRF(f), &(s->tb), &(s->it),
                                &(s->dt), &(s->pt), &(s->et), &(s->tp));
                        ERRCHK(r, end, end);

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        goto end;
                        break;
                case FPROC_FREE:
                        HFREE(s->p0);
                case FPROC_STOP:
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}
```

```
/*
 * The ARM-based IEEE 1394 video frame output function
 *
 * Output the raw video frame to an ARM buffer, for other nodes to pick-up
 *
 * FPOPT fields:
 *
 * - u0: Port
 * - u1: ARM address
 * - u2: Set gap count ?
 * - u3: gap_count value
 *
 * FPSTT fields:
 *
 * - p0: ARM struct pointer
 */
int fproc_fwaw(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
                        r = u1394_hport((void *)(&p0), o->u0);
                        ERRCHK(r, end, end);

                        if (o->u2 != 0) {
                                r = u1394_set_gap_count((H1394)(p0), o->u3);
                                ERRCHK(r, end, end);
                        }

                        r = u1394_arm_register(p0, o->u1, s->fs, NULL,
                                (void *)(&(s->p0)), RAW1394_ARM_READ);
                        ERRCHK(r, end, end);

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Write to the ARM buffer */
                        r = u1394_armwr(s->p0, s->fs, FRF(f));
                        ERRCHK(r, end, end);

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        UPCNT(s, f);

                        goto end;
                        break;
                case FPROC_FREE:
                        p0 = ((ARM *)(s->p0))->h;

                        r = u1394_arm_unregister((void *)(&(s->p0)));
                        ERRCHK(r, end, end);

                        HFREE(p0);

                        goto end;
                        break;
                case FPROC_STOP:
                        r = raw1394_echo_request(((ARM *)(s->p0))->h, 1);
                        ERRCND(r != 0, end, -1,
                                "unable to wake up raw1394_loop_iterate()");
```

```
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}


/*
 * The ARM-based IEEE 1394 video frame input function
 *
 * Input the raw video frame from an ARM buffer, after it is written
 * by another node
 */
int fproc_fwar(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
                        r = u1394_hport((void *)(&p0), o->u0);
                        ERRCHK(r, end, end);

                        if (o->u2 != 0) {
                                r = u1394_set_gap_count((H1394)(p0), o->u3);
                                ERRCHK(r, end, end);
                        }

                        r = u1394_arm_register(p0, o->u1, s->fs, NULL,
                                (void *)(&(s->p0)), RAW1394_ARM_WRITE);
                        ERRCHK(r, end, end);

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Write to the ARM buffer */
                        r = u1394_armrd(s->p0, s->fs, FRF(f));
                        ERRCHK(r, end, end);

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        UPCNT(s, f);

                        goto end;
                        break;
                case FPROC_FREE:
                        p0 = ((ARM *)(s->p0))->h;

                        r = u1394_arm_unregister((void *)(&(s->p0)));
                        ERRCHK(r, end, end);

                        HFREE(p0);

                        goto end;
                        break;
                case FPROC_STOP:
                        r = raw1394_echo_request(((ARM *)(s->p0))->h, 1);
                        ERRCND(r != 0, end, -1,
                                "unable to wake up raw1394_loop_iterate()");
```

```
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}


/*
 * The isochronous IEEE 1394 video frame output function
 *
 * Output the raw video frame to an isochronous IEEE 1394 channel
 *
 * FPOPT fields:
 *
 * - u0: Port
 * - u1: Channel
 * - u2: Maximum packet size
 * - u3: Number of ring-buffer packets
 * - u4: Set gap count ?
 * - u5: gap_count value
 *
 * FPSTT fields:
 *
 * - p0:  raw1394 handle
 */
int fproc_fwiw(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
                        r = u1394_hport((void *)(&(s->p0)), o->u0);
                        ERRCHK(r, end, end);

                        if (o->u4 != 0) {
                                r = u1394_set_gap_count((H1394)(s->p0), o->u5);
                                ERRCHK(r, end, end);
                        }

                        r = u1394_isotx_init((H1394)(s->p0), o->u3, o->u2,
                                o->u1);
                        ERRCHK(r, end, end);

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Send to the channel */
                        r = u1394_isoxx(s->p0, s->fs, FRF(f), &(s->tb),
                                &(s->it), &(s->et));
                        ERRCHK(r, end, end);

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        (s)->tp += TVDIFF(U64, FRU(f), FRD(f));

                        goto end;
                        break;
                case FPROC_FREE:
                        u1394_isoxx_stop(s->p0);
```

```
                        HFREE(s->p0);
                case FPROC_STOP:
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}


/*
 * The isochronous IEEE 1394 video frame input function
 *
 * Receive the raw video frame from an isochronous IEEE 1394 channel
 */
int fproc_fwir(int m, FRAME *f, FPOPT *o, FPSTT *s) {
        FUNCINIT;

        if ((o == NULL) || (s == NULL))
                ERRJMP(end, -1, "invalid object pointer");

        switch (m) {
                case FPROC_INIT:
                        r = u1394_hport((void *)(&(s->p0)), o->u0);
                        ERRCHK(r, end, end);

                        if (o->u4 != 0) {
                                r = u1394_set_gap_count((H1394)(s->p0), o->u5);
                                ERRCHK(r, end, end);
                        }

                        r = u1394_isorx_init((H1394)(s->p0), o->u3, o->u2,
                                o->u1);
                        ERRCHK(r, end, end);

                        goto end;
                        break;
                case FPROC_PROC:
                        if (f == NULL)
                                ERRJMP(end, -1, "invalid video frame pointer");

                        /* Receive from the channel */
                        r = u1394_isoxx(s->p0, s->fs, FRF(f), &(s->tb),
                                &(s->it), &(s->et));
                        ERRCHK(r, end, end);

                        /* Update the use end timestamp */
                        r = gettimeofday(&(FRU(f)), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");

                        (s)->tp += TVDIFF(U64, FRU(f), FRD(f));

                        goto end;
                        break;
                case FPROC_FREE:
                        u1394_isoxx_stop(s->p0);

                        HFREE(s->p0);

                        goto end;
                        break;
                case FPROC_STOP:
                        r = raw1394_echo_request((H1394)(s->p0), 1);
                        ERRCND(r != 0, end, -1,
                                "unable to wake up raw1394_loop_iterate()");
```

```
                        goto end;
                        break;
                default:
                        FATAL;
                        break;
        }

end:
        return R;
}


/* Update the Configuration ROM to export important information to the bus */
int fproc_fwxx_cfrom_upd(FPDAT *d, int e) {
        FUNCINIT;

        /* Sanity checking */
        if (d == NULL)
                FATAL;

        /* Avoid unnecessary operations */
        CNDJMP((d->f == fproc_fwwr) || (d->f == fproc_fwrd), e00, 0);

        /* Determine the exported options */
        enum { IN, OUT };
        enum { ASYNC, ISO };

        if (d->f == fproc_fwaw) {
                i = OUT;
                j = ASYNC;
        } else if (d->f == fproc_fwar) {
                i = IN;
                j = ASYNC;
        } else if (d->f == fproc_fwiw) {
                i = OUT;
                j = ISO;
        } else if (d->f == fproc_fwir) {
                i = IN;
                j = ISO;
        } else {
                if (e) {
                        ERRJMP(e00, -1, "unsupported frame processor");
                } else {
                        goto e00;
                }
        }

        /* The new descriptor string */
        MKSTR(p0, e00, -1, 0,
                "IEEE-1394:VIDEO:%c:%ix%i-%i/%llu:%c-0x%016llx",
                (i == IN)?'I':'O', d->o.x, d->o.y, d->o.b, d->i,
                (j == ASYNC)?'A':'I', d->o.u1);

        /* Update the ROM accordingly */
        r = u1394_cfrom_updtxt(d->o.u0, p0, &(d->s.p1), &u0);
        CNDJMP(r < 0, e01, -1);
        d->s.u2 = u0;

e01:
        GFREE(p0);
e00:
        return R;
}


/* Restore the host Configuration ROM */
int fproc_fwxx_cfrom_rst(FPDAT *d, int e) {
        FUNCINIT;

        /* Sanity checking */
        if (d == NULL)
```

```
                FATAL;

        /* Avoid unnecessary operations */
        CNDJMP((d->f == fproc_fwwr) || (d->f == fproc_fwrd), end, 0);

        /* Restore the original ROM image */
        if ((d->f == fproc_fwaw) || (d->f == fproc_fwar) ||
                        (d->f == fproc_fwiw) || (d->f == fproc_fwir)) {
                r = u1394_cfrom_update(d->o.u0, d->s.p1, d->s.u2, NULL, NULL);
                CNDJMP(r < 0, end, -1);

                GFREE(d->s.p1);
                d->s.u2 = 0;
        } else {
                if (e)
                        ERRJMP(end, -1, "unsupported frame processor");
        }

end:
        return R;
}


/*
 * Go over the Configuration ROM of a remote node, looking
 * for a correctly formatted textual descriptor to parse.
 *
 * ‘p’ is the IEEE 1394 port
 * ‘v’ is the IEEE 1394 vendor ID
 * ‘c’ is the IEEE 1394 product ID
 * ‘d’ is the direction (0 = in, 1 = out)
 * ‘m’ is the transaction type (0 = asynchronous, 1 = isochronous)
 */
int fproc_fwxx_cfrom_get(int p, U32 v, U64 c, int d, int m, int *x, int *y,
                int *b, U64 *t, U64 *u) {
        FUNCINIT;

        /* Construct the format string */
        char *f = NULL;
        MKSTR(f, e00, -1, 0,
                "IEEE-1394:VIDEO:%c:%%ix%%i-%%i/%%llu:%c-0x%%016llx",
                (d == 0)?’I’:’O’, (m == 0)?’A’:’I’);

        /* Get a raw1394 handle */
        H1394 h = NULL;
        r = u1394_hport(&h, p);
        CNDJMP(r < 0, e01, -1);

        /* Variable declaration */
        N1394 n = 0;
        P1394 *s = NULL;
        CLIST *ii = NULL;

rst:
        /* Find the indicated node address */
        r = u1394_find_id(h, p, 0, v, c, &n);
        CNDJMP(r != 0, e02, r);

        /* Query the node */
        CALLOC(s, 1, SZO(P1394), e02, -1, "unable to query node %d.%d",
                p, PHY(n));
        r = u1394_cfrom(h, n, s);
        ERRCHK(r, rst, e03);

        /* Go over the textual descriptors */
        CLISTFOR(s->txt, ii) {
                r = sscanf(ii->d, f, x, y, b, t, u);
                if (r >= 5)
                        goto e04;
        }
```

```
        R = 1;

e04:
        u1394_cfrom_free((void *)&s);
e03:
        GFREE(s);
e02:
        HFREE(h);
e01:
        GFREE(f);
e00:
        return R;
}
```

C.19 prcfw.h

```
#ifndef __PRCFW_H__
#define __PRCFW_H__


/*
 * Local headers
 */

#include "fproc.h"
#include "u1394.h"


/*
 * Local functions
 */

#ifndef __FPROC_D__
FPFUN(fproc_fwwr);
FPFUN(fproc_fwrd);
FPFUN(fproc_fwaw);
FPFUN(fproc_fwar);
FPFUN(fproc_fwiw);
FPFUN(fproc_fwir);
int fproc_fwxx_cfrom_upd(FPDAT *, int);
int fproc_fwxx_cfrom_rst(FPDAT *, int);
int fproc_fwxx_cfrom_get(int, U32, U64, int, int, int *, int *, int *, U64 *,
        U64 *);
#else /* __FPROC_D__ */
NULLP(fproc_fwwr);
NULLP(fproc_fwrd);
NULLP(fproc_fwaw);
NULLP(fproc_fwar);
NULLP(fproc_fwiw);
NULLP(fproc_fwir);
static int UNUSED fproc_fwxx_cfrom_upd(void *p, int e) { return 0; }
static int UNUSED fproc_fwxx_cfrom_rst(void *p, int e) { return 0; }
static int UNUSED fproc_fwxx_cfrom_get(int p, U32 v, U64 c, int d, int m,
        int *x, int *y, int *b, U64 *t, U64 *u) { return 0; }
#endif /* __FPROC_D__ */


#endif /* __PRCFW_H__ */
```

C.20 queue.c

Περιγραφή: Γενική υλοποίηση μιας ουράς

```
#include "queue.h"


/* Initialise a queue */
int queue_init(QUEUE *q) {
        FUNCINIT;

        /* The queue is initially empty */
        q->q = NULL;

        /* Initialise the mutex attributes object */
        r = pthread_mutexattr_init(&(q->a));
        if (r != 0)
                ERRJMP(e00, -1, "unable to initialise mutex attributes object");

        /* Allow the mutexes to be operated by all threads */
        r = pthread_mutexattr_setpshared(&(q->a), PTHREAD_PROCESS_SHARED);
        if (r != 0) {
#if (__GLIBC__ == 2) && (__GLIBC_MINOR__ <= 6)
                ERRMSG("unable to configure mutex attributes object");
#else
                ERRJMP(e01, -1, "unable to configure mutex attributes object");
#endif
        }

        /* Initialise the mutexes */
        r = pthread_mutex_init(&(q->l), &(q->a));
        ERRCND(r != 0, e01, -1, "unable to initialise queue lock");
        r = pthread_mutex_init(&(q->n), &(q->a));
        ERRCND(r != 0, e02, -1, "unable to initialise queue lock");

        /* The queue is empty */
        r = pthread_mutex_lock(&(q->n));
        ERRCND(r != 0, e03, -1, "unable to initialise queue lock");

        goto e00;

e03:
        r = pthread_mutex_destroy(&(q->n));
        if (r != 0)
                FATAL;
e02:
        r = pthread_mutex_destroy(&(q->l));
        if (r != 0)
                FATAL;
e01:
        r = pthread_mutexattr_destroy(&(q->a));
        if (r != 0)
                FATAL;
e00:
        return R;
}


/* Insert a data pointer in the queue */
int queue_enqueue(QUEUE *q, CLIST *n, void *d) {
        FUNCINIT;

        /* Lock the queue */
        r = pthread_mutex_lock(&(q->l));
        ERRCND(r != 0, e00, -1, "unable to acquire queue lock");

        /* Check if the queue is non-empty */
        i0 = (q->q != NULL);

        /* Update the queue backend */
```

```
        if (n != NULL)
                r = clist_insrt(&(q->q), n);
        else
                r = clist_mkend(&(q->q), d);
        CNDJMP(r != 0, e01, -1);

        /* If the queue was not empty, skip the unblocking code */
        if (i0)
                goto e01;

        /* Allow any waiting threads to continue */
        r = pthread_mutex_trylock(&(q->n));
        if (r != 0) {
                if ((errno != 0) && (errno != EAGAIN) && (errno != EBUSY)) {
                        ERRMSG("pthread_mutex_trylock: %s", strerror(errno));
                        R = -1;
                        PANIC(e01);
                }
        }
        r = pthread_mutex_unlock(&(q->n));
        if (r != 0)
                FATAL;

e01:
        r = pthread_mutex_unlock(&(q->l));
        if (r != 0)
                FATAL;
e00:
        return R;
}


/* Retrieve a data pointer from the queue */
int queue_dequeue(QUEUE *q, CLIST **n, void **d) {
        FUNCINIT;

        /* Block on an empty queue */
        r = pthread_mutex_lock(&(q->n));
        ERRCND(r != 0, e00, -1, "unable to acquire queue lock");

        /* Lock the queue */
        r = pthread_mutex_lock(&(q->l));
        ERRCND(r != 0, e00, -1, "unable to acquire queue lock");

        /* Retrieve the pointer from the head of the queue */
        if (d != NULL)
                *d = CLD(q->q);

        /* Remove the queue head */
        if (n != NULL)
                r = clist_xtrct(&(q->q), n);
        else
                r = clist_rmnod(&(q->q), 0, NULL);
        CNDJMP(r != 0, e02, -1);

        /* Subsequent calls to this function should block on an empty queue */
        if (q->q == NULL)
                goto e01;

e02:
        r = pthread_mutex_unlock(&(q->n));
        if (r != 0)
                FATAL;
e01:
        r = pthread_mutex_unlock(&(q->l));
        if (r != 0)
                FATAL;
e00:
        return R;
}
```

```
/* Destroy a queue */
int queue_free(QUEUE *q, int d) {
        FUNCINIT;

        /* Empty the queue */
        if (q->q != NULL)
                clist_rmall(&(q->q), d, NULL);

        /* Destroy the mutex-related objects */
        r = pthread_mutex_destroy(&(q->l));
        if (r != 0)
                FATAL;
        r = pthread_mutex_unlock(&(q->n));
        if (r != 0)
                FATAL;
        r = pthread_mutexattr_destroy(&(q->a));
        if (r != 0)
                FATAL;

        return R;
}
```

C.21 queue.h

```
#ifndef __QUEUE_H__
#define __QUEUE_H__


/*
 * Local headers
 */

#include "shlib.h"
#include "clist.h"


/*
 * Local definitions
 */

/* The base queue struct */
typedef struct queue {
        CLIST *q;               /* A CLIST for the actual queue storage */
        pthread_mutex_t l;      /* Access control mutex */
        pthread_mutex_t n;      /* Mutex to be locked if the queue is empty */
        pthread_mutexattr_t a;  /* Mutex attributes to use */
} QUEUE;


/*
 * Local macros
 */

/* Check for an empty queue - WARNING: no locking performed */
#define QEMPTY(q)               (q.q == NULL)


/*
 * Local functions
 */

int queue_init(QUEUE *);
int queue_enqueue(QUEUE *q, CLIST *, void *);
int queue_dequeue(QUEUE *, CLIST **, void **);
int queue_free(QUEUE *, int);


#endif /* __QUEUE_H__ */
```

C.22 r1394.c

Περιγραφή: Πρόγραμμα για τη διαχείριση της μνήμης ρυθμίσεων (Configuration ROM) του τοπικού κόμβου

```
#include "u1394.h"


/* The usage string */
#define USAGE   "usage: %s (get|set|add) <port> <file> [<text>]"


MAIN {
        MAININIT;

        enum { GET, SET, ADD };

        /* Sanity checking and option handling */
        ERRCND(argc < 4, e00, 1, USAGE, argv0);
        if (strncmp(argv[1], "get", 3) == 0) {
                i0 = GET;
                s0 = "w";
        } else if (strncmp(argv[1], "set", 3) == 0) {
                i0 = SET;
                s0 = "r";
        } else if (strncmp(argv[1], "add", 3) == 0) {
                ERRCND(argc < 5, e00, 1, USAGE, argv0);
                i0 = ADD;
                s0 = "w";
        } else {
                ERRJMP(e00, 1, USAGE, argv0);
        }
        r = sscanf(argv[2], "%i", &i1);
        ERRCND((r < 0) || (r == EOF), e00, 1, USAGE, argv0);

        /* Open the file */
        FILE *f = fopen(argv[3], s0);
        ERRCND(f == NULL, e00, 1, "unable to open file `%s'", argv[3]);

        /* Get an initial image size */
        if (i0 == SET) {
                struct stat t;

                r = fstat(fileno(f), &t);
                ERRCND(r == EOF, e01, 1, "unable to stat file `%s'", argv[3]);

                u0 = t.st_size;
                ERRCND(u0 > MAXCRZ, e01, 1, "image file too large");
                ERRCND((u0 & 0x3) > 0, e01, 1, "invalid image file");
        } else {
                u0 = MAXCRZ;
        }

        /* Handle the textual descriptor add operation */
        if (i0 == ADD) {
                r = u1394_cfrom_updtxt(i1, argv[4], &p0, &u1);
                CNDJMP(r < 0, e01, 1);

                r = fwrite(p0, u1, 1, f);
                ERRCND(r < 1, e01, 1, "unable to write image to file `%s'",
                        argv[3]);

                goto e01;
        }

        /* The raw1394 kernel subsystem handle */
        H1394 h = NULL;

        r = u1394_hport(&h, i1);
```

```
        CNDJMP(r < 0, e01, 1);

        /* Prepare the buffer */
        UCH v;
        UCH b[MAXCRZ];
        memset(b, 0, MAXCRZ);

        /* Get the current image */
        r = u1394_cfrom_get(h, b, MAXCRZ, &u1, &v, 0);
        CNDJMP(r < 0, e02, 1);

        /* Act as requested */
        if (i0 == GET) {
                r = fwrite(b, u1, 1, f);
                ERRCND(r < 1, e02, 1, "unable to write image to file `%s'",
                        argv[3]);
        } else {
                r = fread(b, u0, 1, f);
                ERRCND(r < 1, e02, 1, "unable to read image from file `%s'",
                        argv[3]);

                r = u1394_cfrom_set(h, b, u0, v, 0);
                CNDJMP(r < 0, e02, 1);
        }

e02:
        HFREE(h);
e01:
        GFREE(p0);
        fclose(f);
e00:
        return R;
}
```

C.23 shlib.c

Περιγραφή: Συναρτήσεις γενικής χρήσης

```
#include "shlib.h"


/*
 * A recursive function to free multi-level pointer arrays:
 *
 * 'p' is the address of the top-level array pointer
 * 'l' is the number of pointer levels, level 1 being the "leaves"
 * 'f' controls which array levels will be freed. If it is positive,
 *     then only levels lower or equal to that number will be freed.
 *     If it is negative, then only levels higher or equal to the
 *     absolute value of that number will be freed.
 */
int shlib_ptr_free(void **p, int l, int f) {
        FUNCINIT;

        if ((p == NULL) || (*p == NULL) || (f == 0) || (l == 0))
                goto end;

        if (l > 1) {
                for (i = 0; ((void **)(*p))[i] != NULL; ++i)
                        shlib_ptr_free(&(((void **)(*p))[i]), l - 1, f);
        }

        if (((f > 0) && (l <= f)) || ((f < 0) && (l >= -f)))
                GFREE(*p);

end:
        return R;
}


/* Trim whitespace from the beggining and the end of a string */
int shlib_trim(char *c) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND(c == NULL, end, -1, "invalid pointer");

        /* Find the first non-whitspace character */
        for (i = 0; ((c[i] != 0) && (isspace(c[i]))); ++i) {}
        if (c[i] == 0) {
                c[0] = 0;
                goto end;
        }

        /* Trim the ending whitespace characters */
        for (j = (strlen(c) - 1); ((j > i) && (isspace(c[j]))); --j) {}

        /* Modify the string */
        memmove(c, c + i, j - i + 1);
        c[j - i + 1] = 0;

end:
        return R;
}
```

C.24 shlib.h

Περιγραφή: Μακροεντολές και ορισμοί γενικής χρήσης

```
#ifndef __SHLIB_H__
#define __SHLIB_H__


/*
 * System headers
 */

#include <features.h>

#if (__GLIBC__ == 2) && (__GLIBC_MINOR__ <= 6)
#define __USE_XOPEN2K                   1
#endif

#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <limits.h>
#include <pthread.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

/* Endianess conversions */
#include <arpa/inet.h>


/*
 * Local definitions
 */

/* Type abbreviations */
#define I16                             int16_t
#define I32                             int32_t
#define I64                             int64_t
#define INT                             int
#define U16                             uint16_t
#define U32                             uint32_t
#define U64                             uint64_t
#define UCH                             unsigned char
#define UNT                             unsigned int
#define VPT                             void *


/* Data units */
#define KB                              1024ULL
#define MB                              (KB * KB)
#define GB                              (KB * MB)


/*
 * Local macros
 */

/* Abbreviations */
#define SZA(t)                          (sizeof(t) / sizeof(t[0]))
#define SZO(t)                          sizeof(t)
#define SZP(t)                          sizeof(t *)

/* Attributes */
#define UNUSED                          __attribute__((unused))
```

```
/* Octets to bits */
#define BITS(n)                         (n * 8)

/* Zero-out an object */
#define CLEAR(o)                        memset(&(o), 0, SZO(o))

/* Compare two integers and equalise if one is zero */
#define CMPEQ(i0, i1, l)                { \
        if ((i0) != (i1)) { \
                if ((i0) == 0) \
                        (i0) = (i1); \
                else if ((i1) == 0) \
                        (i1) = (i0); \
                else \
                        goto l; \
        } \
}

/* Jump on condition and set the return value */
#define CNDJMP(c, l, e)                 { \
        if (c) { \
                R = e; \
                goto l; \
        } \
}

/* Pointer to a constant */
#define CPTR(t, v)                      ((t []){ v })

/* Pointer dereference operations */
#define DI16(p)                         (*((I16 *)(p)))
#define DI32(p)                         (*((I32 *)(p)))
#define DI64(p)                         (*((I64 *)(p)))
#define DINT(p)                         (*((INT *)(p)))
#define DU16(p)                         (*((U16 *)(p)))
#define DU32(p)                         (*((U32 *)(p)))
#define DU64(p)                         (*((U64 *)(p)))
#define DUCH(p)                         (*((UCH *)(p)))
#define DUNT(p)                         (*((UNT *)(p)))

/* Error message macro */
#define ERROUT(M, ...)                  { \
        fprintf(stderr, "%s: " M "\n", argv0, ##__VA_ARGS__); \
        fflush(stderr); \
}

/* An error message with a source code reference */
#define ERRSRC(M, ...)                  { \
        ERROUT(M " %s:%i", ##__VA_ARGS__, __FILE__, __LINE__); \
}

/* Output an error message */
#ifdef DEBUG
#define ERRMSG(M, ...)                  ERRSRC(M, ##__VA_ARGS__)
#else
#define ERRMSG(M, ...)                  ERROUT(M, ##__VA_ARGS__)
#endif

/* Division helper macros */
#define DIV(n, d)                       (((d) != 0)?((n) / (d)):0)
#define DFP(n, d) \
        (((d) != 0)?(double)((double)(n) / (double)(d)):0)

#define ERRJMP(l, e, m, ...)            { \
        R = e; \
        if (errno != 0) { \
                ERRMSG(m ": %s", ##__VA_ARGS__, strerror(errno)); \
        } else { \
                ERRMSG(m, ##__VA_ARGS__); \
        } \
```

```
        goto l; \
}

#define ERRCND(c, l, e, m, ...)                 { \
        if (c) \
                ERRJMP(l, e, m, ##__VA_ARGS__); \
}

/* Fatal error message - no recovery possible */
#define FATAL                                   { \
        ERRMSG("internal error in %s:%i", __FILE__, __LINE__); \
        exit(0); \
}

/* Common function declarations */
#define FUNCINIT \
        /* Return code for the current function */ \
        int R UNUSED = 0; \
        /* Return code for any called functions */ \
        int r UNUSED = 0; \
        /* Loop iterators */ \
        int i UNUSED = 0; \
        int j UNUSED = 0; \
        int k UNUSED = 0; \
        /* Temporary variables */ \
        I16 i16 UNUSED = 0; \
        I32 i32 UNUSED = 0; \
        I64 i64 UNUSED = 0; \
        U16 u16 UNUSED = 0; \
        U32 u32 UNUSED = 0; \
        U64 u64 UNUSED = 0; \
        /* Integers */ \
        int i0 UNUSED = 0; \
        int i1 UNUSED = 0; \
        int i2 UNUSED = 0; \
        int i3 UNUSED = 0; \
        /* Pointers */ \
        void *p0 UNUSED = NULL; \
        void *p1 UNUSED = NULL; \
        void *p2 UNUSED = NULL; \
        void *p3 UNUSED = NULL; \
        /* Pointers to a pointer */ \
        void **pp0 UNUSED = NULL; \
        void **pp1 UNUSED = NULL; \
        /* Strings */ \
        char *s0 UNUSED = NULL; \
        char *s1 UNUSED = NULL; \
        char *s2 UNUSED = NULL; \
        char *s3 UNUSED = NULL; \
        /* Time values */ \
        struct timeval tv0 UNUSED = { 0, 0 }; \
        struct timeval tv1 UNUSED = { 0, 0 }; \
        struct timespec ts0 UNUSED = { 0, 0 }; \
        struct timespec ts1 UNUSED = { 0, 0 }; \
        /* Floating point numbers */ \
        double f0 UNUSED = 0; \
        double f1 UNUSED = 0; \
        double f2 UNUSED = 0; \
        double f3 UNUSED = 0; \
        /* Unsigned integers */ \
        UNT u0 UNUSED = 0; \
        UNT u1 UNUSED = 0; \
        UNT u2 UNUSED = 0; \
        UNT u3 UNUSED = 0; \
        /* Unsigned character */ \
        UCH uch UNUSED = 0;

/* Network->Host integer byte order conversion */
#define H32(i)                                  ntohl(*((U32 *)(i)))

/* main() call */
```

```
#define MAIN \
        /* The global program name pointer */ \
        char *argv0 = NULL; \
        int main(int argc, char **argv)

/* Initial statements for main() */
#define MAININIT \
        FUNCINIT; \
        /* Set the global program name pointer */ \
        argv0 = argv[0];

/* Free allocated memory using the generic free() function */
#define GFREE(p)                        { \
        if ((p) != NULL) \
                free(p); \
        p = NULL; \
}

/* A conditional pointer value */
#define IFPTR(p, n, z)                  ((void *)(((p) == NULL)?(z):(n)))

/* A conditional string */
#define IFSTR(c, s)                     ((char *)(((c) != 0)?s:""))

/* Error checking for malloc() */
#define MCHECK(p, l, e, m, ...)         { \
        if ((p) == NULL) \
                ERRJMP(l, e, m, ##__VA_ARGS__); \
}

/* A wrapper around calloc() */
#define CALLOC(p, n, s, l, e, m, ...)   { \
        (p) = calloc((n), (s)); \
        if ((m == NULL) || (strlen(m) == 0)) { \
                MCHECK((p), l, (e), "unable to allocate buffer"); \
        } else { \
                MCHECK((p), l, (e), m, ##__VA_ARGS__); \
        } \
}

/* Create an object */
#define MKOBJ(p, t, v, l, e, m, ...)    { \
        CALLOC(p, 1, SZO(t), l, e, m, ##__VA_ARGS__); \
        *((t *)p) = v; \
}

/* Create a pointer object */
#define MKPTR(p, o, l, e, m, ...) \
        MKOBJ(p, void *, o, l, e, m, ##__VA_ARGS__)

/* Allocate and fill in a string */
#define MKSTR(p, l, e, n, f, ...)       { \
        int __l__ =  snprintf(NULL, 0, (f), ##__VA_ARGS__) + (n) + 1; \
        (p) = calloc(1, __l__); \
        MCHECK((p), l, (e), "unable to allocate string buffer"); \
        snprintf((p), __l__, (f), ##__VA_ARGS__); \
}

/* A simple message */
#define MSG(M, ...)                     { \
        printf(M "\n", ##__VA_ARGS__); \
}

/* Host->Network integer byte order conversion */
#define N32(i)                                  htonl(*((U32 *)(i)))

/* Initialise a timestamp string environment */
#define TSTAMP(T, l, e, C)              { \
        char T[40]; \
        time_t t0 = time(NULL); \
        ERRCND(t0 < 0, l, e, "unable to get current time"); \
```

```
        struct tm t1; \
        gmtime_r(&t0, &t1); \
        strftime(T, sizeof(T), "%F %T", &t1); \
        C; \
}

/* Redirect an output stream to a file */
#define NEWOSF(s, p, l, e)                      { \
        FILE *f = freopen((p), "a", s); \
        ERRCND(f == NULL, l, e, "unable to redirect " #s); \
        TSTAMP(T, l, e, fprintf(f, "%s: %s\n", argv0, T)); \
}

/* Redirect the standard error stream */
#define NEWERR(p, l, e)                 NEWOSF(stderr, p, l, e)

/* Redirect the standard output stream */
#define NEWOUT(p, l, e)                 NEWOSF(stdout, p, l, e)

/* Convert to n-lets */
#define NLET(n, i)                      (i / n + ((i % n) > 0))

/* A NULL pointer declaration */
#define NULLP(p)                        void *p = NULL

/* Access an object through a pointer */
#define OBJ(t, p)                       (*((t *)(p)))

/* Convert to octets */
#define OCTS(n)                         (((n) >> 3) + (((n) & 0x7) > 0))

/* Internal error message */
#define PANIC(l)                        { \
        ERRSRC("internal error in"); \
        goto l; \
}

/* Convert to quadlets */
#define QUAD(n)                         (((n) >> 2) + (((n) & 0x3) > 0))

/* Print system information */
#define SYSINFO                         { \
        ERRMSG("System information:\n\nPage size: %li\n", \
                sysconf(_SC_PAGESIZE)); \
}

/* An object array */
#define TAR(t, ...)                     (t []){ __VA_ARGS__ }
#define IAR(...)                        TAR(int, __VA_ARGS__)
#define PAR(...)                        TAR(void *, __VA_ARGS__, NULL)

/* Program execution tracer */
#define TRACE                           ERRSRC("TRACE");

/* struct timeval difference in microseconds */
#define TVDIFF(c, t1, t0) \
        ((c)((t1).tv_sec - (t0).tv_sec) * (c)1000000 + \
        (c)((t1).tv_usec - (t0).tv_usec))

/* Convert a struct timeval to microseconds */
#define TVUSEC(c, t) \
        ((c)(t).tv_sec * (c)1000000 + (c)(t).tv_usec)

/* Convert a number of microseconds to a struct timeval */
#define USECTV(c, tv, t)                { \
        (tv).tv_sec = (time_t)(c)((c)(t) / (c)1000000); \
        (tv).tv_usec = (suseconds_t)(c)((c)(t) % (c)1000000); \
}


/*
```

```
 * Local declarations
 */

/* A global pointer to the program name */
extern char *argv0;


/*
 * Local functions
 */

int shlib_ptr_free(void **, int, int);
int shlib_trim(char *);


#endif /* __SHLIB_H__ */
```

C.25 timer.c

Περιγραφή: Συναρτήσεις χρονισμού με υποστήριξη για πολυνηματικές εφαρμογές

```
#include "timer.h"


/* Timer system ready flag */
static int              timer_ready = 0;

/* The mutex to use for the timer operations */
static pthread_mutex_t  timer_mutex = PTHREAD_MUTEX_INITIALIZER;


/* Prepare the timer system */
int timer_init() {
        if (timer_ready)
                return 0;

        int r = pthread_mutex_lock(&timer_mutex);
        if (r < 0) {
                r = -1;
                ERRMSG("unable to initialise timer mutex");
        }

        timer_ready = 1;

        return r;
}


/* Wait until the specified time */
int timer_wait(struct timeval tv) {
        FUNCINIT;

        r = timer_init();
        CNDJMP(r < 0, end, -1);

        struct timespec ts = {
                .tv_sec = tv.tv_sec,
                .tv_nsec = tv.tv_usec * 1000,
        };

        r = pthread_mutex_timedlock(&timer_mutex, &ts);
        ERRCND ((r < 0) && (errno != ETIMEDOUT), end, -1,
                "timer operation failed");

end:
        return R;
}


/* A crude usleep() implementation */
int timer_usleep(U64 t) {
        FUNCINIT;

        r = gettimeofday(&tv0, NULL);
        ERRCND(r < 0, end, -1, "cannot get timestamp");

        USECTV(U64, tv0, TVUSEC(U64, tv0) + t);
        r = timer_wait(tv0);
        CNDJMP(r < 0, end, -1);

end:
        return R;
}
```

C.26 timer.h

```
#ifndef __TIMER_H__
#define __TIMER_H__


/*
 * Local headers
 */

#include "shlib.h"


/*
 * Local functions
 */

int timer_init();
int timer_wait(struct timeval);
int timer_usleep(U64);


#endif /* __TIMER_H__ */
```

C.27 u1394.c

Περιγραφή: Συναρτήσεις για το χειρισμό του διαύλου IEEE 1394

```
#include "u1394.h"


#define QUERY_RETRY_LIMIT       3


/* The mutex to use for Configuration ROM get/set operations */
static pthread_mutex_t          cfrom_mutex = PTHREAD_MUTEX_INITIALIZER;


/* Create a new raw1394 handle */
int u1394_new_handle(H1394 *h) {
        *h = raw1394_new_handle();;
        if (*h == NULL) {
                if (errno == 0) {
                        ERRMSG("unable to access the IEEE 1394 subsystem");
                } else {
                        ERRMSG("unable to access the IEEE 1394 subsystem: %s",
                                strerror(errno));
                }
                return SYSERR;
        }

        int r = raw1394_get_port_info(*h, NULL, 0);
        if (r < 0) {
                ERRMSG("unable to retrieve port information");
                return FAIL;
        }

        return SUCCESS;
}


/* Attach the handle to a port */
int u1394_set_port(H1394 h, int p) {
        FUNCINIT;

        r = raw1394_set_port(h, p);
        if (r != 0) {
                if ((errno == ESTALE) || (errno == EAGAIN)) {
                        R = RST;
                } else {;
                        ERRMSG("unable to access port %d: %s", p,
                                strerror(errno));
                        R = SYSERR;
                }
        }

        return R;
}


/* Create a new raw1394 handle and attach to a port */
int u1394_hport(H1394 *h, int p) {
        FUNCINIT;

        r = u1394_new_handle(h);
        if (r != SUCCESS)
                goto end;

        r = u1394_set_port(*h, p);

end:
        return r;
}
```

```
/* Set the gap count, optionally performing gap count optimisation */
int u1394_set_gap_count(H1394 h, UCH g) {
        FUNCINIT;

        /* Sanity checking */
        if (g > 63)
                g = 63;

        /*
         * Perform gap count optimisation using a fixed hops/gap_count
         * function value table. Instead of going over the bus topology,
         * we use the number of nodes to produce a maximum number of
         * transmission hops. This method is sufficient for buses with
         * only a few nodes and it is still better than an unoptimised
         * bus, even with a high number of nodes.
         */
        if (g == 0) {
                UCH t[] = IEEE1394A_GCO_T;

                /* Maximum number of hops */
                i0 = raw1394_get_nodecount(h) - 1;
                if (i0 < 1)
                        i0 = 1;
                if (i0 > SZA(t))
                        i0 =  SZA(t);

                /* Access gap_count table */
                g = t[i0 - 1];
        }

        /* Build the PHY configuration packet data quadlet */
        u32 = (0x40 | g) << 16;

        /* Send a PHY configuration packet to set the gap count */
        r = raw1394_phy_packet_write(h, htonl(u32));
        ERRCND(r < 0, end, -1, "unable to set the gap count");

        /* Reset the bus */
        r = raw1394_reset_bus(h);
        ERRCND(r < 0, end, -1, "unable to reset the bus");

end:
        return r;
}


/* Retrieve the properties of all available ports */
int u1394_get_port_info(H1394 h, S1394(portinfo) **p, int *n) {
        int i = 0, r = 0;
        while (1) {
                /* Free any memory allocated by previous failed attempts */
                GFREE(*p);

                /* Retrieve the number of available ports */
                *n = raw1394_get_port_info(h, NULL, 0);
                if (*n == 0) {
                        ERRMSG("no IEEE 1394 ports available");
                        return NOPORT;
                }

                /* Allocate memory for the port properties */
                *p = calloc(*n, SZO(S1394(portinfo)));
                if (*p == NULL) {
                        ERRMSG("unable to retrieve the port properties: %s",
                                strerror(errno));
                        return FAIL;
                }

                /* Fill in the port properties array */
                r = raw1394_get_port_info(h, *p, *n);
```

```
                if (r == *n)
                        return SUCCESS;

                /* Enforce the retry limit */
                ++i;
                if (i >= QUERY_RETRY_LIMIT) {
                        GFREE(*p);
                        *n = 0;
                        ERRMSG("unable to retrieve the port properties: %s",
                                "retry limit exceeded");
                        return SYSERR;
                }
        }
}


/*
 * Read a quadlet and perform byte order conversion
 * for storage in an unsigned 32-bit variable.
 */
int u1394_rq_u32(H1394 h, N1394 n, A1394 a, U32 *b) {
        int r = raw1394_read(h, n, a, 4, b);
        if (r != 0) {
                if ((errno == ESTALE) || (errno == EAGAIN)) {
                        return RST;
                } else {
                        ERRMSG("unable to read quadlet at " NAPF ": %s",
                                NAPV(n, a), strerror(errno));
                        return RDERR;
                }
        }

        /* Byte order conversion */
        *b = H32(b);

        return SUCCESS;
}


/*
 * Read a quadlet for storage in a buffer
 */
int u1394_rq_str(H1394 h, N1394 n, A1394 a, void *b) {
        int r = raw1394_read(h, n, a, 4, b);
        if (r != 0) {
                if ((errno == ESTALE) || (errno == EAGAIN)) {
                        return RST;
                } else {
                        ERRMSG("unable to read quadlet at " NAPF ": %s",
                                NAPV(n, a), strerror(errno));
                        return RDERR;
                }
        }

        return SUCCESS;
}


/*
 * Compute the CRC-16 checksum for a quadlet.
 * Code adapted from IEEE Std 1212-2001/7.3.
 */
void u1394_crc16(U16 *c, U32 q) {
        FUNCINIT;

        U32 t = *c;

        for (i = 28; i >= 0; i -= 4) {
                u32 = ((t >> 12) ^ (q >> i)) & 0xf;
                t = (t << 4) ^ (u32 << 12) ^ (u32 << 5) ^ u32;
        }
```

```
        *c = t & 0xffff;
}


/* Compute the CRC-16 checksum for a number of quadlets */
void u1394_crc16_loop(U16 *c, void *q, UNT n) {
        FUNCINIT;

        *c = 0;
        for (i = 0; i < n; ++i)
                u1394_crc16(c, H32(q + (i << 2)));
}


/* Parse a Configuration ROM descriptor (IEEE Std 1212-2001/7.5.4) */
int u1394_cfdsc(H1394 h, N1394 n, A1394 a, P1394 *p) {
        FUNCINIT;

        /* Descriptor CRC-16, length and type */
        U16 c = 0;
        U32 l = 0, t = 0;

        /* A memory buffer pointer */
        NULLP(m);

        /* Get the length of the descriptor and the CRC code */
        r = u1394_rq_u32(h, n, a, &u32);
        ERRCHK(r, end, end);
        l = (u32 >> 16) << 2;
        c = u32 & 0xffff;

        /* Get the type of the descriptor */
        r = u1394_rq_u32(h, n, a + 4, &u32);
        ERRCHK(r, end, end);
        t = (u32 >> 24);
        u1394_crc16(&u16, u32);

        /* Act on the descriptor type field */
        switch (t) {
                case 0: /* Textual descriptor (IEEE Std 1212-2001/7.5.4.1) */
                        /*
                         * Allocate the necessary memory to store the string.
                         * Two of the descriptor quadlets are the metadata, so
                         * we do not need to allocate space for them. On the
                         * other hand we do need to allocate space for the
                         * terminating zero byte of the string, which is not
                         * necessarily included in the read quadlets.
                         */
                        CALLOC(m, 1, l - 7, end, FAIL,
                                "unable to parse textual descriptor " NAPF,
                                NAPV(n, a));

                        /* Retrieve the metadata quadlets for the CRC check */
                        r = u1394_rq_u32(h, n, a + 8, &u32);
                        ERRCHK(r, end, end);
                        u1394_crc16(&u16, u32);

                        /*
                         * Retrieve the textual descriptor content, assuming
                         * that it is an ASCII string, as seems to be the case
                         * with most IEEE 1394 devices.
                         */
                        for (i = 0; i < (l - 8); i += 4) {
                                r = u1394_rq_str(h, n, a + 12 + i, m + i);
                                ERRCHK(r, end, end);
                                u1394_crc16(&u16, H32(m + i));
                        }

                        /* CRC checksum check */
                        ERRCND(c != u16, end, CKSUM,
```

```
                                "CRC error for descriptor " NAPF, NAPV(n, a));

                        /* Store the string */
                        r = clist_mkend(&(p->txt), m);
                        CNDJMP(r != 0, end, FAIL);
                        p->ntxt += 1;
                        p->ltxt += l - 8;
                        m = NULL;

                        break;
                default:
                        break;
        }

end:
        GFREE(m);
        return R;
}


/*
 * Parse a Configuration ROM directory, as defined generally for the
 * CSR architecture in IEEE Std 1212-2001/7.5-7.7 and specifically for
 * IEEE 1394 in IEEE Std 1394-1995/8.3.2.5.5-8.3.2.5.7.1.
 */
int u1394_cfdir(H1394 h, N1394 n, A1394 a, P1394 *p) {
        FUNCINIT;

        /* Directory CRC-16 and length */
        U16 c = 0;
        U32 l = 0;

        /* Get the length of the directory and the CRC code */
        r = u1394_rq_u32(h, n, a, &u32);
        ERRCHK(r, end, end);
        l = (u32 >> 16) * 4;
        c = u32 & 0xffff;

        /* Go over the directory contents */
        for (i = 0; i < l; i += 4) {
                /* Retrieve a key */
                r = u1394_rq_u32(h, n, a + 4 + i, &u32);
                ERRCHK(r, end, end);
                u1394_crc16(&u16, u32);

                /* The address offset */
                A1394 o = (u32 & 0xffffff) * 4;

                /* Parse the type key field */
                switch (u32 >> 30) {
                        case 0: /* Immediate value */
                                break;
                        case 1: /* CSR offset */
                                /* Parse the key_ID value */
                                switch ((u32 >> 24) & 0x3f) {
                                        case 1: /* Descriptor leaf */
                                                r = u1394_cfdsc(h, n,
                                                        ADDR_CSR_BASE + o, p);
                                                ERRCHK(r, end, end);
                                                break;
                                        default:
                                                break;
                                }
                                break;
                        case 2: /* Leaf */
                                /* Parse the key_ID value */
                                switch ((u32 >> 24) & 0x3f) {
                                        case 1: /* Descriptor leaf */
                                                r = u1394_cfdsc(h, n,
                                                        a + 4 + i + o, p);
                                                ERRCHK(r, end, end);
```

```
                                                break;
                                        default:
                                                break;
                                }
                                break;
                        case 3: /* Directory */
                                /* Parse the key_ID value */
                                switch ((u32 >> 24) & 0x3f) {
                                        case 1: /* Descriptor directory */
                                                r = u1394_cfdir(h, n,
                                                        a + 4 + i + o, p);
                                                ERRCHK(r, end, end);
                                                break;
                                        default:
                                                break;
                                }
                                break;
                        default:
                                R = INTERR;
                                PANIC(end);
                }
        }

        /* CRC checksum check */
        ERRCND(c != u16, end, CKSUM, "CRC error for directory " NAPF,
                NAPV(n, a));

end:
        return R;
}


/* Free a node properties struct */
void u1394_cfrom_free(void **p) {
        clist_rmall(&(((P1394 *)(*p))->txt), 1, NULL);
        GFREE(*p);
}


/*
 * Parse the Configuration ROM of a node, storing a few
 * interesting values for later use.
 */
int u1394_cfrom(H1394 h, N1394 n, P1394 *p) {
        FUNCINIT;

        /* Block length */
        int l = 0;

        /*
         * Get the Bus_Info_Block length field from the first Configuration
         * ROM quadlet (IEEE Std 1212-2001/7.2).
         */
        r = u1394_rq_u32(h, n, ADDR_CSR_CFG_ROM_BASE, &u32);
        ERRCHK(r, end, end);
        l = u32 >> 24;

        /*
         * If the node Configuration ROM is in the deprecated
         * minimal ROM format (IEEE Std 1212-2001/7.2), store
         * the ROM ID (rid) field value only.
         */
        if (l == 1) {
                p->n = PHY(n);
                p->m = 1;
                p->cid = u32 & 0x00ffffff;

                goto end;
        } else {
                l *= 4;
        }
```

```
        /*
         * The second quadlet is always the string "1394", therefore we
         * can ignore it. The third quadlet, though, contains the node
         * capabilitities.
         */
        r = u1394_rq_u32(h, n, ADDR_CSR_CFG_ROM_BASE + 8, &u32);
        ERRCHK(r, end, end);
        p->n = PHY(n);
        p->irmc = (u32 >> 31) & 0x1;
        p->cmc = (u32 >> 30) & 0x1;
        p->isc = (u32 >> 29) & 0x1;
        p->bmc = (u32 >> 28) & 0x1;
        p->maxwr = 1 << (((u32 >> 12) & 0xf) + 1);

        /*
         * The fourth and fifth quadlets are the Vendor and Chip IDs, as
         * specified in IEEE Std 1394-1995/8.3.2.5.4.
         */
        r = u1394_rq_u32(h, n, ADDR_CSR_CFG_ROM_BASE + 12, &u32);
        ERRCHK(r, end, end);
        p->vid = (u32 >> 8);
        p->cid = ((U64)u32 & 0xff) << 32;
        r = u1394_rq_u32(h, n, ADDR_CSR_CFG_ROM_BASE + 16, &u32);
        ERRCHK(r, end, end);
        p->cid += u32;

        /* Query the root directory */
        r = u1394_cfdir(h, n, ADDR_CSR_CFG_ROM_BASE + 1 + 4, p);
        ERRCHK(r, end, end);

end:
        return R;
}


/*
 * Get the Bus Manager node ID. According to IEEE Std 1394-1995/8.3.2.3.6
 * the proper way to do that is to query the BUS_MANAGER_ID CSR register
 * of the Isochronous Resource Manager. A value of 0x3f (or 63 in decimal)
 * means that no Bus Manager has been selected, in which case the IRM will
 * perform some of the bus management tasks.
 */
int u1394_getbm(H1394 h, N1394 *n) {
        FUNCINIT;

        r = u1394_rq_u32(h, raw1394_get_irm_id(h), ADDR_CSR_BUS_MANAGER_ID,
                &u32);
        ERRCHK(r, end, end);

        /* Convert to the format used by libraw1394 */
        *n = NID(u32 & 0x3f);

end:
        return R;
}


/*
 * Get the IDs of several important nodes, i.e. the local node, the Isochronous
 * Resource Manager and the  Bus Manager.
 */
int u1394_getid(H1394 h, N1394 *nl, N1394 *ni, N1394 *nb) {
        FUNCINIT;

        /*
         * Get the local node ID - this accesses the raw1394handle_t
         * struct internals, therefore no error checking is needed.
         */
        *nl = raw1394_get_local_id(h);
```

```
        /* Get the IRM node ID */
        *ni = raw1394_get_irm_id(h);

        /* Get the BM node ID */
        r = u1394_getbm(h, nb);
        ERRCHK(r, end, end);

end:
        return R;
}


/* Free a node list */
void u1394_nlist_free(void **l) {
        clist_rmall((CLIST **)l, 1, u1394_cfrom_free);
}


/*
 * Query all nodes accessible via a handle, optionally selecting a port.
 * For each node store the result of the query in a circular list.
 */
int u1394_nlist(H1394 h, B1394 *b, int p, int s, CLIST **l) {
        FUNCINIT;

        /* If requested, attach the handle to a port */
        if (s) {
                r = u1394_set_port(h, p);
                ERRCHK(r, end, end);
        }

        /* Query all nodes */
        for (i = 0; i < raw1394_get_nodecount(h); ++i) {
                P1394 *q = calloc(1, SZO(P1394));
                MCHECK(q, end, FAIL, "unable to query node %d.%d", p, i);

                /* Populate the property struct */
                q->p = p;
                q->b = b;
                r = u1394_cfrom(h, NID(i), q);
                if (r != SUCCESS) {
                        R = r;
                        GFREE(q);
                        goto end;
                }

                /* Add to the node list */
                clist_mkend(l, q);
        }

end:
        return R;
}


/* Free an IEEE 1394 bus property struct */
void u1394_binfo_free(void **b) {
        B1394 *t = *b;

        HFREE(t->h);
        GFREE(t->d);

        u1394_nlist_free((void *)(&(t->n)));

        GFREE(*b);
}


/* Examine an IEEE 1394 bus ("port" in libraw1394 terminology) */
int u1394_binfo(B1394 **b, int p, char *d) {
        FUNCINIT;
```

```
        /* Allocate memory if necessary */
        if (*b == NULL)
                CALLOC(*b, 1, SZO(B1394), end, FAIL,
                        "unable to query port %i", p);

        /* Create a new raw1394 handle if necessary */
        if ((*b)->h == NULL) {
                r = u1394_new_handle(&((*b)->h));
                ERRCHK(r, end, end);
        }

        /* Query all nodes */
        r = u1394_nlist((*b)->h, *b, p, 1, &((*b)->n));
        ERRCHK(r, end, end);

        /* Get the important node IDs */
        r = u1394_getid((*b)->h, &((*b)->l), &((*b)->i), &((*b)->b));
        ERRCHK(r, end, end);

        /* Fill in the bus properties */
        (*b)->p = p;
        (*b)->c = raw1394_get_nodecount((*b)->h);
        (*b)->d = strdup(d);
        ERRCND((*b)->d == NULL, end, FAIL, "unable to query port %i", p);

end:
        if (R != SUCCESS)
                u1394_binfo_free((void *)b);
        return R;
}


/* Free an IEEE 1394 bus list */
void u1394_blist_free(void **l) {
        clist_rmall((CLIST **)l, 1, u1394_binfo_free);
}


/* Examine all IEEE 1394 buses */
int u1394_blist(H1394 *d, CLIST **l, int *c, int *n) {
        FUNCINIT;

        /* The port properties array */
        S1394(portinfo) *p = NULL;

        /*
         * Allow for a null pointer as the port
         * count and node count arguments.
         */
        if (c == NULL)
                c = &j;
        if (n == NULL)
                n = &k;

        /*
         * Create an access handle to the raw1394 kernel
         * subsystem, if necessary.
         */
        H1394 h = NULL;
        if ((d == NULL) || (*d == NULL)) {
                r = u1394_new_handle(&h);
                ERRCHK(r, end, end);
        } else {
                h = *d;
        }

        /* Retrieve the properties of all available ports */
        r = u1394_get_port_info(h, &p, c);
        ERRCHK(r, end, end);
```

```
        /* Examine all buses */
        *n = 0;
        for (i = 0; i < *c; ++i) {
                B1394 *b = NULL;

                /* Get the bus information */
                r = u1394_binfo(&b, i, p[i].name);
                ERRCHK(r, end, end);

                /* We should at least have the local node */
                if (b->n == NULL) {
                        R = INTERR;
                        PANIC(end);
                }

                /* Add to the list */
                r = clist_mkend(l, b);
                CNDJMP(r != 0, end, FAIL);

                *n += b->c;
        }

end:
        GFREE(p);
        if (R != SUCCESS)
                u1394_blist_free((void *)l);
        if (d == NULL) {
                HFREE(h);
        } else {
                *d = h;
        }
        return R;
}


/* Find a node in a specific bus using its UID */
int u1394_find_id(H1394 h, int p, int s, U64 v, U64 c, N1394 *N) {
        FUNCINIT;

rst:
        u1394_nlist_free(&p0);
        r = u1394_nlist(h, NULL, p, s, (void *)(&p0));
        ERRCHK(r, rst, end);

        /* Go through the list */
        CLISTFOR(p0, p1) {
                if ((((P1394 *)((CLIST *)p1)->d)->vid == v) &&
                                (((P1394 *)((CLIST *)p1)->d)->cid == c)) {
                        *N = NID(((P1394 *)((CLIST *)p1)->d)->n);
                        goto end;
                }
        }

        R = NONODE;

end:
        u1394_nlist_free(&p0);
        return R;
}


/* Generic request handler for raw1394_loop_iterate() */
int u1394_rqh(H1394 h, void *d, raw1394_errcode_t e) {
        int ii = 0, dd = 0, pp = 0, ee = 0;

        /* Error decoding */
        if (raw1394_internal_err(e)) {
                ee = 1;
                ((RQH *)d)->tb = 0;
        } else if (raw1394_get_ack(e) == 1) {
                ii = 1;
```

```
        } else if (raw1394_errcode_to_errno(e) == 0) {
                dd = 1;
        } else if (raw1394_errcode_to_errno(e) == EAGAIN) {
                pp = 1;
                ((RQH *)d)->tb = 0;
        } else {
                ee = 1;
                ((RQH *)d)->tb = 0;
        }

        RQHOP(d, 1, ii, dd, pp, ee, d);

        return 0;
}


/*
 * The core read/write transaction function. This function issues one or more
 * asynchronous requests and listens for responses afterwards, thus improving
 * the use of the bus bandwidth.
 *
 * 'v'  is the Vendor ID
 * 'c'  is the Chip ID
 * 'm'  is the maximum request size
 * 'q'  is the maximum number of incomplete requests
 * 's'  is the total number of data bytes to transfer
 * 'b'  is the data buffer
 * 'it' is the number of immediate responses (i.e. unified transactions)
 * 'dt' is the number of delayed responses received
 * 'pt' is the number of postponed transactions
 * 'et' is the erroneous transaction counter
 * 't'  is the sum of the transaction latencies in microseconds
 * 'f'  is the asynchronous raw1394 function to call
 *
 * If the node address was not provided, or in the case of a bus reset, this
 * function can use the UID of the node to detect its current IEEE 1394 ID.
 * However, any transactions interrupted by errors will generally not be
 * repeated.
 *
 * Please note that this function is not thread-safe - libraw1394 as a whole
 * is not thread-safe either. Each raw1394 handle should be accessed by one
 * thread only.
 */
int u1394_rdwr(H1394 h, N1394 *n, U64 v, U64 c, A1394 a, int m, int q, int s,
                VPT b, U64 *tb, U64 *it, U64 *dt, U64 *pt, U64 *et, U64 *t,
                F1394(*f)) {
        FUNCINIT;

        /* Sanity checking */
        if (m > MAXBLK)
                m = MAXBLK;

        /* Allocate the request handle array */
        CALLOC(p0, q, SZO(RQH), end, FAIL, "unable to allocate handle array");

        /* Setup the request handle array */
        for (i = 0; i < q; ++i) {
                ((RQH *)p0)[i].r.callback = u1394_rqh;
                ((RQH *)p0)[i].r.data = &(((RQH *)p0)[i]);
                ((RQH *)p0)[i].u = &p1;
        }

        if (PHY(*n) != 0x3f)
                goto r01;

r00:
        r = u1394_find_id(h, 0, 0, v, c, n);
        ERRCHK(r, end, end);

r01:
        /* Submit transactions asynchronously */
```

```
        while ((i0 < s) && (i1 < q)) {
                i2 = s - i0;
                if (i2 > m)
                        i2 = m;

                if ((n != NULL) && (t != NULL)) {
                        r = gettimeofday(&(((RQH *)p0)[i1].t), NULL);
                        ERRCND(r != 0, end, -1, "unable to set timestamp");
                }
                ((RQH *)p0)[i1].tb = i2;

                r = f(h, *n, a + i0, i2, b + i0,
                        (unsigned long)(&(((RQH *)p0)[i1].r)));
                E1394CK(r, r00, end, "transaction submission failed");

                i0 += i2;
                ++i1;
        }

        goto r03;

r02:
        r = u1394_find_id(h, 0, 0, v, c, n);
        ERRCHK(r, end, end);

r03:
        /* Complete the data transfers */
        while (i0 < s) {
                if (p1 == NULL) {
                        r = raw1394_loop_iterate(h);
                        if (p1 != NULL) {
                                --i1;
                                if (tb != NULL)
                                        *tb += ((RQH *)(p1))->tb;
                                if (it != NULL)
                                        *it += ((RQH *)(p1))->it;
                                if (dt != NULL)
                                        *dt += ((RQH *)(p1))->dt;
                                if (pt != NULL)
                                        *pt += ((RQH *)(p1))->pt;
                                if (et != NULL)
                                        *et += ((RQH *)(p1))->et;
                                if ((n != NULL) && (t != NULL)) {
                                        r = gettimeofday(&tv0, NULL);
                                        ERRCND(r != 0, end, -1,
                                                "unable to get timestamp");

                                        *t += TVDIFF(U64, tv0, ((RQH *)p1)->t);
                                }
                        }
                        E1394CK(r, r02, end, "transaction completion failed");
                }

                if (p1 != NULL) {
                        i2 = s - i0;
                        if (i2 > m)
                                i2 = m;

                        RQHOP(p1, 0, 0, 0, 0, 0, p1);

                        if ((n != NULL) && (t != NULL)) {
                                r = gettimeofday(&(((RQH *)p0)[i1].t), NULL);
                                ERRCND(r != 0, end, -1,
                                        "unable to set timestamp");
                        }
                        ((RQH *)p0)[i1].tb = i2;

                        r = f(h, *n, a + i0, i2, b + i0,
                                (unsigned long)(&(((RQH *)p1)->r)));
                        E1394CK(r, r02, end, "transaction submission failed");
```

```
                        i0 += i2;
                        ++i1;
                        p1 = NULL;
                }
        }

        /* Collect the remaining responses */
        while (i1 > 0) {
                r = raw1394_loop_iterate(h);
                if (p1 != NULL) {
                        --i1;
                        if (tb != NULL)
                                *tb += ((RQH *)(p1))->tb;
                        if (it != NULL)
                                *it += ((RQH *)(p1))->it;
                        if (dt != NULL)
                                *dt += ((RQH *)(p1))->dt;
                        if (pt != NULL)
                                *pt += ((RQH *)(p1))->pt;
                        if (et != NULL)
                                *et += ((RQH *)(p1))->et;
                        if ((n != NULL) && (t != NULL)) {
                                r = gettimeofday(&tv0, NULL);
                                ERRCND(r != 0, end, -1,
                                        "unable to get timestamp");

                                *t += TVDIFF(U64, tv0, ((RQH *)(p1))->t);
                        }
                }
                E1394CK(r, r02, end, "transaction completion failed");
        }

end:
        GFREE(p0);
        return R;
}


/* Read data from a node */
int u1394_read(H1394 h, N1394 *n, U64 v, U64 c, A1394 a, int m, int q, int s,
                VPT b, U64 *tb, U64 *it, U64 *dt, U64 *pt, U64 *et, U64 *t) {
        return u1394_rdwr(h, n, v, c, a, m, q, s, b, tb, it, dt, pt, et, t,
                raw1394_start_read);
}


/* Write data to a node */
int u1394_write(H1394 h, N1394 *n, U64 v, U64 c, A1394 a, int m, int q, int s,
                VPT b, U64 *tb, U64 *it, U64 *dt, U64 *pt, U64 *et, U64 *t) {
        return u1394_rdwr(h, n, v, c, a, m, q, s, b, tb, it, dt, pt, et, t,
                raw1394_start_write);
}


/* Check if the last byte of an ARM buffer has been accessed */
int u1394_arm_rqh(H1394 h, S1394(arm_request_response) *q, UNT l, void *c,
                byte_t t) {
        ARM *s = c;

        /* ARM request starting and ending addresses */
        U64 rs = q->request->destination_offset;
        U64 re = rs + q->request->buffer_length + q->response->buffer_length;

        /* ARM area ending address */
        U64 ae = s->a + s->l;

        /*
         * If the last byte of the ARM buffer is accessed, set the
         * operation completion flag for notification purposes.
         */
        if (((t & (s->m)) != 0) && (rs <= ae) && (ae <= re))
```

```
                s->c = 1;

        return 0;
}


/* Register an Address Range Mapping */
int u1394_arm_register(H1394 h, A1394 a, int l, void *b, ARM **s, int m) {
        FUNCINIT;

        CALLOC(*s, 1, SZO(ARM), e00, -1, "unable to allocate ARM struct");

        (*s)->h = h;
        (*s)->a = a;
        (*s)->l = l;
        (*s)->m = m;
        (*s)->r.arm_callback = u1394_arm_rqh;
        (*s)->r.pcontext = *s;
        (*s)->c = 0;

        u64 = (U64)(unsigned long)(&((*s)->r));
        r = raw1394_arm_register(h, a, l, b, u64, m, m, 0);
        ERRCND(r < 0, e01, -1, "ARM registration failed");

        goto e00;

e01:
        GFREE(*s);
e00:
        return R;
}


/* Unregister an Address Range Mapping */
int u1394_arm_unregister(ARM **s) {
        FUNCINIT;

        r = raw1394_arm_unregister((*s)->h, (*s)->a);
        ERRCND(r < 0, end, -1, "ARM unregistration failed");

        GFREE(*s);

end:
        return R;
}


/*
 * Copy data to an ARM buffer and wait for it to be retrieved by another node.
 * We assume that the data has been read whenever the last byte of the buffer
 * is read - not a bullet-proof way to do it, but it works for our purposes.
 */
int u1394_armwr(ARM *s, int l, void *b) {
        FUNCINIT;

        s->c = 0;

        r = raw1394_arm_set_buf(s->h, s->a, l, b);
        ERRCND(r < 0, end, -1, "unable to write to ARM buffer");

        while (!(s->c)) {
                r = raw1394_loop_iterate(s->h);
                ERRCND(r < 0, end, -1, "ARM operation error");
                if (r == 1)
                        goto end;
        }

        s->c = 0;

end:
        return R;
```

```
}


/* Wait for data to be written to an ARM buffer and then retrieve it. */
int u1394_armrd(ARM *s, int l, void *b) {
        FUNCINIT;

        s->c = 0;

        while (!(s->c)) {
                r = raw1394_loop_iterate(s->h);
                ERRCND(r < 0, end, -1, "ARM operation error");
                if (r == 1)
                        goto end;
        }

        r = raw1394_arm_get_buf(s->h, s->a, l, b);
        ERRCND(r < 0, end, -1, "unable to write to ARM buffer");

        s->c = 0;

end:
        return R;
}


/*
 * Bus reset handler for isochronous operations - needed to reallocate
 * the various isochronous resources at the IRM after a bus reset.
 */
int u1394_isobrh(H1394 h, UNT g) {
        FUNCINIT;

        IOP *s = raw1394_get_userdata(h);

        /* Execute the previous handler */
        r = (s->h)(h, g);
        CNDJMP(r < 0, e00, r);

        /* Re-allocate resources */
        r = raw1394_channel_modify(h, s->r.c, RAW1394_MODIFY_ALLOC);
        ERRCND(r < 0, e01, 0, "unable to allocate isochronous channel");
e01:
        r = raw1394_bandwidth_modify(h, s->r.b, RAW1394_MODIFY_ALLOC);
        ERRCND(r < 0, e00, 0, "unable to allocate isochronous bandwidth");
e00:
        return R;
}


/* The outgoing isochronous packet handler */
enum raw1394_iso_disposition u1394_isotxh(H1394 h, UCH *b, UNT *l, UCH *t,
                UCH *s, int z, UNT e) {
        enum raw1394_iso_disposition R = RAW1394_ISO_OK;

        IOP *o = raw1394_get_userdata(h);

        if ((o->l - o->i) > (o->m - SZO(o->i)))
                *l = o->m - SZO(o->i);
        else
                *l = o->l - o->i;

        /*
         * Isochronous packets have no address field. Transmit the
         * buffer offset so that the receiving node will know what
         * to do with the data.
         */
        DU32(b) = htonl(o->i);

        /* Copy the data */
        memcpy(b + SZO(o->i), o->b + o->i, *l);
```

```
        /* Update the IOP struct */
        o->i += *l;
        if (o->t != NULL)
                *(o->t) += *l;
        if (o->p != NULL)
                *(o->p) += 1;
        if (o->e != NULL)
                *(o->e) += (e != 0);
        if (o->i >= o->l) {
                o->c = 1;
                R = RAW1394_ISO_DEFER;
        }

        *l += SZO(o->i);
        *t = 0;
        *s = 0;

        return R;
}


/* The incoming isochronous packet handler */
enum raw1394_iso_disposition u1394_isorxh(H1394 h, UCH *b, UNT l, UCH c,
                UCH t, UCH s, UNT z, UNT e) {
        enum raw1394_iso_disposition R = RAW1394_ISO_OK;

        IOP *o = raw1394_get_userdata(h);

        U32 n = H32(b);
        l -= SZO(o->i);

        /* Copy the data */
        memcpy(o->b + n, b + SZO(o->i), l);

        /* Update the IOP struct */
        if (o->t != NULL)
                *(o->t) += l;
        if (o->p != NULL)
                *(o->p) += 1;
        if (o->e != NULL)
                *(o->e) += (e != 0);
        if (((n + l) >= o->l) || (n < o->i)) {
                o->c = 1;
                R = RAW1394_ISO_DEFER;
        }
        o->i = n;

        return R;
}


/* Start isochronous packet transmission */
int u1394_isotx_init(H1394 h, int n, int m, int c) {
        FUNCINIT;

        IOP *o = NULL;
        CALLOC(o, 1, SZO(IOP), e00, -1, "unable to allocate IOP struct");

        o->b = NULL;
        o->l = 0;
        o->i = 0;
        o->e = NULL;
        o->m = m;
        o->c = 0;
        o->x = 1;
        o->r.b = 6144;
        o->r.c = c;

        raw1394_set_userdata(h, o);
```

```
        /* Channel allocation (IEEE Std 1394-1995/8.3.2.3.8) */
        r = raw1394_channel_modify(h, c, RAW1394_MODIFY_ALLOC);
        ERRCND(r < 0, l00, 0, "unable to allocate isochronous channel");

l00:
        /* Allocate all available bandwidth (IEEE Std 1394-1995/8.3.2.3.7) */
        r = raw1394_bandwidth_modify(h, o->r.b, RAW1394_MODIFY_ALLOC);
        ERRCND(r < 0, l01, 0, "unable to allocate isochronous bandwidth");

l01:
        /* Set the bus reset handler */
        p0 = raw1394_set_bus_reset_handler(h, u1394_isobrh);
        ERRCND(p0 == NULL, e01, -1, "unable to set the bus reset handler");
        if (p0 != u1394_isobrh)
                o->h = p0;

        r = raw1394_iso_xmit_init(h, u1394_isotxh, n, m, c,
                RAW1394_ISO_SPEED_400, -1);
        ERRCND(r < 0, e01, -1, "unable to initialise isochronous transmission");

        r = raw1394_iso_xmit_start(h, -1, -1);
        ERRCND(r < 0, e03, -1, "unable to start isochronous transmission");

        goto e00;

e03:
        r = raw1394_bandwidth_modify(h, o->r.b, RAW1394_MODIFY_FREE);
        ERRCND(r < 0, e02, -1, "unable to free isochronous bandwidth");
e02:
        r = raw1394_channel_modify(h, c, RAW1394_MODIFY_FREE);
        ERRCND(r < 0, e01, -1, "unable to free isochronous channel");
e01:
        GFREE(o);
e00:
        return R;
}


/* Start isochronous packet reception */
int u1394_isorx_init(H1394 h, int n, int m, int c) {
        FUNCINIT;

        IOP *s = NULL;
        CALLOC(s, 1, SZO(IOP), e00, -1, "unable to allocate IOP struct");

        s->b = NULL;
        s->l = 0;
        s->i = 0;
        s->e = NULL;
        s->m = m;
        s->c = 0;
        s->x = 0;
        s->r.b = 0;
        s->r.c = c;

        raw1394_set_userdata(h, s);

        r = raw1394_iso_recv_init(h, u1394_isorxh, n, m, c,
                RAW1394_DMA_DEFAULT, -1);
        ERRCND(r < 0, e01, -1, "unable to initialise isochronous reception");

        r = raw1394_iso_recv_start(h, -1, -1, 0);
        ERRCND(r < 0, e01, -1, "unable to start isochronous reception");

        goto e00;

e01:
        GFREE(s);
e00:
        return R;
}
```

```
/* Stop isochronous operations */
int u1394_isoxx_stop(H1394 h) {
        FUNCINIT;

        raw1394_iso_stop(h);
        raw1394_iso_shutdown(h);

        IOP *s = raw1394_get_userdata(h);

        /* Free isochronous resources at the IRM */
        if (s->x) {
                r = raw1394_bandwidth_modify(h, s->r.b, RAW1394_MODIFY_FREE);
                ERRCND(r < 0, l00, 0, "unable to free isochronous bandwidth");

l00:
                r = raw1394_channel_modify(h, s->r.c, RAW1394_MODIFY_FREE);
                ERRCND(r < 0, l01, 0, "unable to free isochronous channel");

l01:
                /* Restore the original bus reset handler */
                p0 = raw1394_set_bus_reset_handler(h, s->h);
                ERRCND(p0 == NULL, end, -1,
                        "unable to restore the bus reset handler");
        }

end:
        GFREE(s);
        raw1394_set_userdata(h, NULL);
        return R;
}


/* Transmit/Receive data using isochronous operations */
int u1394_isoxx(H1394 h, int l, void *b, U64 *t, U64 *p, U64 *e) {
        FUNCINIT;

        IOP *o = raw1394_get_userdata(h);

        o->b = b;
        o->l = l;
        o->i = 0;
        o->t = t;
        o->p = p;
        o->e = e;
        o->c = 0;

        while (!(o->c)) {
                r = raw1394_loop_iterate(h);
                ERRCND(r < 0, end, -1, "isochronous operation error");
                if (r == 1)
                        goto end;
        }

        o->b = NULL;
        o->l = 0;
        o->i = 0;
        o->c = 0;

end:
        return R;
}


/*
 * Change all Configuration ROM directory addresses by an offset
 *
 * 'd' is the directory address
 * 'o' is the offset in quadlets
 * 'l' is the number of directory levels for which the relative entries
```

```
 *     should be adjusted according to the offset
 */
int u1394_cfdir_offset(char *d, int o, int l) {
        FUNCINIT;

        /* Directory length */
        u0 = (H32(d) >> 16) << 2;

        /* Go over the directory contents */
        for (i = 0; i < u0; i += 4) {
                u32 = H32(d + 4 + i);

                /* The value field */
                U32 v = u32 & 0xffffff;

                /* Parse the type key field */
                switch (u32 >> 30) {
                        case 3: /* Directory */
                                /* Recurse to the sub-directory */
                                r = u1394_cfdir_offset(d + 4 + i + (v << 2), o,
                                        (l > 0)?(l - 1):0);
                                CNDJMP(r < 0, end, r);
                        case 2: /* Leaf */
                                if (l <= 0)
                                        break;
                        case 1: /* CSR offset */
                                ERRCND(v > (0xffffff - o), end,
                                        -1, "CSR address range overflow");

                                u32 += o;
                                DU32(d + 4 + i) = htonl(u32);

                                break;
                        case 0: /* Immediate value */
                                break;
                        default:
                                FATAL;
                                break;
                }

                u1394_crc16(&u16, u32);
        }

        /* Update the CRC checksum */
        DU32(d) = htonl((((u0 >> 2) << 16) & 0xffff0000) + u16);

end:
        return R;
}


/*
 * Add a textual descriptor to a Configuration ROM image
 *
 * 'o' is the original image
 * 's' is the image size in quadlets
 * 't' is the textual descriptor string
 * 'n' is the resulting image
 * 'l' is the resulting image size in quadlets
 */
int u1394_cfrom_addtxt(char *o, UNT s, char *t, char **n, UNT *l) {
        FUNCINIT;

        /* Sanity checking */
        if ((o == NULL) || (t == NULL) || (n == NULL) || (l == NULL))
                FATAL;

        s <<= 2;
        *l = 0;

        /* Get the Bus_Info_Block length field */
```

```
        u0 = H32(o) >> 24;

        /* If the ROM is in the minimal format, return an error */
        ERRCND(u0 <= 1, e00, -1, "unsupported Configuration ROM format");

        u0 <<= 2;

        /* The string length */
        i0 = strlen(t);
        u1 = QUAD(i0) << 2;

        /* Allocate the space for the new image */
        CALLOC(*n, 1, s + u1 + 16, e00, -1, "");
        memcpy(*n, o, s);

        /* Root directory address */
        p0 = *n + u0 + 4;

        /* Update the Configuration ROM offsets */
        r = u1394_cfdir_offset(p0, 1, 1);
        CNDJMP(r < 0, e01, r);

        /* Root directory length */
        u2 = (H32(p0) >> 16) << 2;

        /* Make space for the new directory entry */
        for (i = s - u2 - u0 - 4; i > 0; --i)
                DUCH(p0 + 4 + u2 + i + 4) = DUCH(p0 + 4 + u2 + i);

        /* Create the textual descriptor by setting any non-zero fields */
        p1 = *n + s + 4;
        DU32(p1) = htonl(((u1 >> 2) + 2) << 16);
        memcpy(p1 + 12, t, i0);

        /* Textual descriptor CRC checksum */
        u1394_crc16_loop(&u16, p1 + 4, H32(p1) >> 16);
        DU32(p1) = htonl(H32(p1) + u16);

        /* Root directory entry for the textual descriptor */
        DU32(p0 + 4 + u2) = htonl(0x81000000 + (U32)((p1 - p0 - u2 - 4) >> 2));

        /* Root directory CRC checksum and length */
        u1394_crc16_loop(&u16, p0 + 4, (u2 >> 2) + 1);
        DU32(p0) = htonl((((u2 >> 2) + 1) << 16) + u16);

        *l = (s + u1 + 16) >> 2;

        goto e00;

e01:
        GFREE(*n);
e00:
        return R;
}


/*
 * Get the local host Configuration ROM, adjusting
 * the data buffer byte order if requested.
 */
int u1394_cfrom_get(H1394 h, void *b, UNT m, UNT *s, UCH *v, int e) {
        FUNCINIT;

        r = raw1394_get_config_rom(h, b, m, s, v);
        ERRCND(r < 0, end, -1, "unable to get Configuration ROM image");

        /* Adjust endianess if requested */
        if (e)
                for (i = 0; i < *s; ++i)
                        DU32(b + (i << 2)) = ntohl(DU32(b + (i << 2)));
```

```
end:
        return R;
}


/*
 * Set the local host Configuration ROM, adjusting
 * the data buffer byte order if requested.
 */
int u1394_cfrom_set(H1394 h, void *b, UNT s, UCH v, int e) {
        FUNCINIT;

        /* Adjust endianess if requested */
        if (e)
                for (i = 0; i < s; ++i)
                        DU32(b + (i << 2)) = ntohl(DU32(b + (i << 2)));

        r = raw1394_update_config_rom(h, b, s, v);
        ERRCND(r < 0, end, -1, "unable to set Configuration ROM image");

        /* Restore buffer endianess */
        if (e)
                for (i = 0; i < s; ++i)
                        DU32(b + (i << 2)) = htonl(DU32(b + (i << 2)));

end:
        return R;
}


/* Add a textual descriptor to the local host Configuration ROM */
int u1394_cfrom_updtxt(int p, char *d, void **o, UNT *c) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND((p < 0) || (d == NULL), e00, -1, "invalid argument");

        /* Access the IEEE 1394 bus */
        H1394 h = NULL;
        r = u1394_hport(&h, p);
        CNDJMP(r < 0, e00, -1);

        /* Lock the mutex */
        r = pthread_mutex_lock(&cfrom_mutex);
        ERRCND(r < 0, e01, -1, "unable to lock Configuration ROM mutex");

        /* Get the current ROM */
        CALLOC(p0, 1, MAXCRZ, e02, -1, "");
        r = u1394_cfrom_get(h, p0, MAXCRZ, &u32, &uch, 0);
        CNDJMP(r < 0, e03, -1);

        /* Create the new image */
        r = u1394_cfrom_addtxt(p0, u32 >> 2, d, (void *)(&p1), &u0);
        CNDJMP(r < 0, e03, -1);

        /* Update the host ROM */
        r = u1394_cfrom_set(h, p1, u0 << 2, uch, 0);
        GFREE(p1);
        CNDJMP(r < 0, e03, -1);

        /* Optionally return the original ROM image */
        if (o != NULL)
                *o = p0;
        else
                GFREE(p0);
        if (c != NULL)
                *c = u32;

        goto e02;

e03:
```

```
        GFREE(p0);
e02:
        r = pthread_mutex_unlock(&cfrom_mutex);
        if (r != 0)
                FATAL;
e01:
        HFREE(h);
e00:
        return R;
}


/* Replace the host Configuration ROM */
int u1394_cfrom_update(int p, void *b, UNT s, void **o, UNT *c) {
        FUNCINIT;

        /* Sanity checking */
        ERRCND((p < 0) || (b == NULL) || (s == 0), e00, -1, "invalid argument");

        /* Access the IEEE 1394 bus */
        H1394 h = NULL;
        r = u1394_hport(&h, p);
        CNDJMP(r < 0, e00, -1);

        /* Lock the mutex */
        r = pthread_mutex_lock(&cfrom_mutex);
        ERRCND(r < 0, e01, -1, "unable to lock Configuration ROM mutex");

        /* Get the current ROM */
        CALLOC(p0, 1, MAXCRZ, e02, -1, "");
        r = u1394_cfrom_get(h, p0, MAXCRZ, &u32, &uch, 0);
        CNDJMP(r < 0, e03, -1);

        /* Update the host ROM */
        r = u1394_cfrom_set(h, b, s, uch, 0);
        GFREE(p1);
        CNDJMP(r < 0, e03, -1);

        /* Optionally return the original ROM image */
        if (o != NULL)
                *o = p0;
        else
                GFREE(p0);
        if (c != NULL)
                *c = u32;

        goto e02;

e03:
        GFREE(p0);
e02:
        r = pthread_mutex_unlock(&cfrom_mutex);
        if (r != 0)
                FATAL;
e01:
        HFREE(h);
e00:
        return R;
}
```

C.28 u1394.h

```
#ifndef __U1394_H__
#define __U1394_H__


/*
 * Local headers
 */

#include "shlib.h"
#include "clist.h"


/*
 * Sub-system headers
 */

/* libraw1394 headers */
#include <libraw1394/csr.h>
#include <libraw1394/raw1394.h>


/*
 * Local definitions
 */

/* Return codes */
enum {
        SUCCESS,                /* Success */
        FAIL,                   /* Generic failure code */
        INVAL,                  /* Invalid argument */
        INTERR,                 /* Internal error */
        RST,                    /* Bus reset */
        SYSERR,                 /* Subsystem error */
        NOPORT,                 /* No ports available */
        NONODE,                 /* No node detected */
        RDERR,                  /* Read error */
        WRERR,                  /* Write error */
        CKSUM                   /* Checksum error */
};

/* Type abbreviations */
#define A1394                   nodeaddr_t
#define H1394                   raw1394handle_t
#define N1394                   nodeid_t
#define S1394(s)                struct raw1394_##s
#define F1394(...) \
        int (__VA_ARGS__) \
        (H1394, N1394, A1394, size_t , quadlet_t *, unsigned long)

/* IEEE 1394 bus properties */
typedef struct b1394 {
        H1394 h;                /* raw1394 handle */
        U32 p;                  /* IEEE 1394 port */

        char *d;                /* An identifying string for this bus */

        N1394 l;                /* The local node */
        N1394 i;                /* The Isochronous Resource Manager node */
        N1394 b;                /* The Bus Manager node */

        U32 c : 16;             /* Node count */
        CLIST *n;               /* List of nodes in this bus */
} B1394;

/* Node properties */
typedef struct p1394 {
        U32 p;                  /* IEEE 1394 port */
```

```
        U16 n;                  /* Node address */

        U32 m : 1;              /* m = 1 if the minimal ROM format is used */
        U32 : 15;               /* Explicit struct padding */

        U32 vid : 24;           /* Vendor ID */
        U32 : 8;                /* Explicit struct padding */
        U64 cid;                /* Chip ID */

        U32 irmc : 1;           /* Isochronous Resource Manager Capable */
        U32 cmc : 1;            /* Cycle Master Capable */
        U32 isc : 1;            /* Isochronous operations supported */
        U32 bmc : 1;            /* Bus Master Capable */
        U32 : 28;               /* Explicit struct padding */
        U32 maxwr;              /* Maximum block size for writes to the node */

        B1394 *b;               /* A pointer to this node's bus struct */

        CLIST *txt;             /* A list of textual descriptor strings */
        int ntxt;               /* The number of textual descriptors */
        int ltxt;               /* Total length of the textual descriptors */
} P1394;

/* Request handler data */
typedef struct rqh {
        int c;                  /* Request complete flag */
        int it;                 /* Immediate response transaction */
        int dt;                 /* Delayed response transaction */
        int pt;                 /* Postponed transaction */
        int et;                 /* Erroneous transaction */
        int tb;                 /* Transaction bytes */
        S1394(reqhandle) r;     /* Request handle for the default handler */
        void **u;               /* Struct reuse pointer */
        struct timeval t;       /* Submission time */
} RQH;


/* ARM information struct */
typedef struct arm {
        H1394 h;                /* IEEE 1394 handle */
        A1394 a;                /* Start address */
        int l;                  /* Mapped area length */
        int m;                  /* Area access mode */
        S1394(arm_reqhandle) r; /* ARM request handler */
        int c;                  /* Operation complete flag */
} ARM;


/* Isochronous operation information struct */
typedef struct iop {
        U32 m;                  /* Maximum packet size */
        int x;                  /* 1 if transmitting, 0 otherwise */
        struct {
                int b;          /* Bandwidth units */
                int c;          /* Channel */
        } r;                    /* Resources */
        void *b;                /* Data buffer */
        U32 l;                  /* Buffer length */
        U32 i;                  /* Current buffer offset */
        U64 *t;                 /* Transferred bytes */
        U64 *p;                 /* Packet counter */
        U64 *e;                 /* Error count */
        int c;                  /* Operation complete flag */
        bus_reset_handler_t h;  /* The original bus reset handler */
} IOP;


/*
 * Local constants
 */
```

```
/* CSR address space offset */
#define ADDR_CSR_BASE           0xfffff0000000ULL
/* BUS_MANAGER_ID register offset */
#define ADDR_CSR_BUS_MANAGER_ID (ADDR_CSR_BASE + 0x21c)
/* Configuration ROM offset */
#define ADDR_CSR_CFG_ROM_BASE   (ADDR_CSR_BASE + 0x400)

/* Maximum block transaction payload size */
#define MAXBLK                  2048

/* Maximum isochronous packet payload size */
#define MAXISO                  4096

/* Maximum Configuration ROM size */
#define MAXCRZ                  1024

/*
 * Gap count optimisation tables
 *
 * IEEE Std 1394-1995 has a fixed table for the gap_count value, since
 * it had no way to measure the node-to-node round-trip times.
 *
 * IEEE Std 1394a-2000 contains a revised version of that table for
 * bus managers that are not able to measure the round-trip times.
 * The new version offers lower performance, but also more reliable
 * operation.
 */
#define IEEE1394_GCO_T          \
        { 1, 4, 6, 9, 12, 14, 17, 20, 23, 25, 28, 31, 33, 36, 39, 42 }
#define IEEE1394A_GCO_T         \
        { 5, 7, 8, 10, 13, 16, 18, 21, 24, 26, 29, 32, 35, 37, 40, 43, 46, 48, \
                51, 54, 57, 59, 62 }


/*
 * Local macros
 */

/* Check a libraw1394 function return value for errors */
#define E1394CK(r, rst, end, m, ...) { \
        if (r < 0) { \
                if ((errno == ESTALE) || (errno == EAGAIN)) { \
                        goto rst; \
                } else { \
                        ERRJMP(end, FAIL, m, ##__VA_ARGS__); \
                } \
        } \
}

/* Check a function return value for errors */
#define ERRCHK(r, rst, end)     { \
        if ((r) == RST) { \
                goto rst; \
        } else if ((r) != SUCCESS) { \
                R = r; \
                goto end; \
        } \
}

/* Destroy a raw1394 handle */
#define HFREE(h)                { \
        raw1394_destroy_handle(h); \
        h = NULL; \
}

/* Create a libraw1394 node ID out of a physical ID */
#define NID(i)                  (0xffc0 | (i))

/* Extract the physical ID from a libraw1394 node ID */
#define PHY(i)                  ((i) & 0x3f)
```

```
/* Node/Address pair macros */
#define NAPF                    "%d/0x%016llx"
#define NAPV(n, a)              PHY(n), (unsigned long long)(U64)(a)

/* Operate on the variable fields of the RQH type */
#define RQHOP(rr, cc, ii, dd, pp, ee, uu) \
        ((RQH *)(rr))->it = ii; \
        ((RQH *)(rr))->dt = dd; \
        ((RQH *)(rr))->pt = pp; \
        ((RQH *)(rr))->et = ee; \
        ((RQH *)(rr))->c = cc; \
        *(((RQH *)(rr))->u) = uu;


/*
 * Local functions
 */

int u1394_new_handle(H1394 *);
int u1394_set_port(H1394, int);
int u1394_hport(H1394 *, int);
int u1394_set_gap_count(H1394, UCH);
int u1394_get_port_info(H1394, struct raw1394_portinfo **, int *);
int u1394_rq_u32(H1394, N1394, A1394, U32 *);
int u1394_rq_str(H1394, N1394, A1394, void *);
void u1394_crc16(U16 *, U32);
void u1394_crc16_loop(U16 *, void *, UNT);
int u1394_cfdsc(H1394, N1394, A1394, P1394 *);
int u1394_cfdir(H1394, N1394, A1394, P1394 *);
void u1394_cfrom_free(void **);
int u1394_cfrom(H1394, N1394, P1394 *);
int u1394_getbm(H1394, N1394 *);
int u1394_getid(H1394, N1394 *, N1394 *, N1394 *);
void u1394_nlist_free(void **);
int u1394_nlist(H1394, B1394 *, int, int, CLIST **);
void u1394_binfo_free(void **);
int u1394_binfo(B1394 **, int, char *);
void u1394_blist_free(void **);
int u1394_blist(H1394 *, CLIST **, int *, int *);
int u1394_find_id(H1394, int, int, U64, U64, N1394 *);
int u1394_rdwr(H1394, N1394 *, U64, U64, A1394, int, int, int, void *, U64 *,
        U64 *, U64 *, U64 *, U64 *, U64 *, F1394(*));
int u1394_read(H1394, N1394 *, U64, U64, A1394, int, int, int, void *, U64 *,
        U64 *, U64 *, U64 *, U64 *, U64 *);
int u1394_write(H1394, N1394 *, U64, U64, A1394, int, int, int, void *, U64 *,
        U64 *, U64 *, U64 *, U64 *, U64 *);
int u1394_arm_register(H1394, A1394, int, void *, ARM **, int);
int u1394_arm_unregister(ARM **);
int u1394_armwr(ARM *, int, void *);
int u1394_armrd(ARM *, int, void *);
int u1394_isotx_init(H1394, int, int, int);
int u1394_isorx_init(H1394, int, int, int);
int u1394_isoxx_stop(H1394);
int u1394_isoxx(H1394, int, void *, U64 *, U64 *, U64 *);
int u1394_cfdir_offset(char *, int, int);
int u1394_cfrom_addtxt(char *, UNT, char *, char **, UNT *);
int u1394_cfrom_get(H1394, void *, UNT, UNT *, UCH *, int);
int u1394_cfrom_set(H1394, void *, UNT, UCH, int);
int u1394_cfrom_updtxt(int, char *, void **, UNT *);
int u1394_cfrom_update(int, void *, UNT, void **, UNT *);


#endif /* __U1394_H__ */
```

C.29 v1394.c

Περιγραφή: Το κυρίως πρόγραμμα v1394

```
/* Local headers */
#include "cmenu.h"


MAIN {
        MAININIT;

        NEWERR("v1394.err", e00, 1);

        SYSINFO;

        /* Terminal initialization */
        r = cterm_init();
        CNDJMP(r < 0, e00, 1);

        FPDLD o;

rst:
        R = 0;
        memset(&o, 0, SZO(o));

        /* The menu system */
        r = cmenu_iosel(&o);
        XMCHK(r, e01);
        r = cmenu_iprop(&o);
        XMCHK(r, e01);
        r = cmenu_oprop(&o);
        XMCHK(r, e01);
        r = cmenu_fppar(&o);
        XMCHK(r, e01);

        /* IEEE 1394 Configuration ROM handling */
        r = fproc_fwxx_cfrom_upd(&(o.d0), 0);
        CNDJMP(r < 0, e01, -1);
        r = fproc_fwxx_cfrom_upd(&(o.d1), 0);
        CNDJMP(r < 0, e02, -1);

        /* Start the worker threads */
        r = fproc_dloop(&o);
        CNDJMP(r < 0, e03, -1);

        /*
         * The main thread processing loop. It is only useful for the
         * display of statistics, while waiting for the user to stop
         * the processing.
         */
        HDR("Processing...");
        OUT("");
        STS("Enter:Stop    R:Restart    Q:Quit");
        nodelay(stdscr, 1);

        i0 = KEY_HOME;

        r = gettimeofday(&tv0, NULL);
        ERRCND(r < 0, e04, -1, "cannot get timestamp");
        while ((!o.d0.x) && (!o.d1.x) && (i0 != 'q') && (i0 != 'r') &&
                        (i0 != '\n')) {
                r = timer_usleep(500000);
                CNDJMP(r < 0, e04, -1);

                r = gettimeofday(&tv1, NULL);
                ERRCND(r < 0, e04, -1, "cannot get timestamp");
                u64 = TVDIFF(U64, tv1, tv0);
                f0 = u64;

                OUT(STFMT("", ""), STARG(o, u64, f0));
```

```
                i0 = GETCH;
        }
        nodelay(stdscr, 0);
        CNDJMP(i0 == 'q', e04, 1);
        if (i0 == 'r')
                R = 2;

        ERRMSG("Statistics:\n\n" STFMT("", ""), STARG(o, u64, f0));

e04:
        /* Stop the worker threads */
        fproc_dloop_stop(&o, 1);
e03:
        fproc_fwxx_cfrom_rst(&(o.d1), 0);
e02:
        fproc_fwxx_cfrom_rst(&(o.d0), 0);
e01:
        if (R == 2)
                goto rst;

        endwin();
e00:
        return R;
}
```