ΒΙΒΛΙΟΘΗΚΗ
ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ
026000265469

2.7.5 ......200.4

Λογισμικό οδήγησης κάρτας PCI διαύλου I^2C και μετατροπέας ψηφιακών σημάτων TTL-ECL-NIM-LVDS για πειράματα Φυσικής Υψηλών Ενεργειών

171
ΜΠΛΕ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΑΣΧΑΛΗΣ Δ. ΒΗΧΟΥΔΗΣ
ΗΛΕΚΤΡΟΝΙΚΟΣ ΜΗΧΑΝΙΚΟΣ ΤΕ

ΕΠΙΒΛΕΠΩΝ: ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ Π.ΚΟΚΚΑΣ

ΕΡΓΑΣΤΗΡΙΟ ΦΥΣΙΚΗΣ ΥΨΗΛΩΝ ΕΝΕΡΓΕΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΣΤΙΣ ΣΥΓΧΡΟΝΕΣ ΗΛΕΚΤΡΟΝΙΚΕΣ ΤΕΧΝΟΛΟΓΙΕΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΚΕΝΤΡΙΚΗ ΒΙΒΛΙΟΘΗΚΗ

ΤΜΗΜΑ ΦΥΣΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΙΩΑΝΝΙΝΑ ΣΕΠΤΕΜΒΡΙΟΣ 2002

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Περίληψη

Η παρούσα μεταπτυχιακή διπλωματική εργασία, η οποία εκπονήθηκε στο Εργαστήριο Φυσικής Υψηλών Ενεργειών του Πανεπιστημίου Ιωαννίνων, αποτελείται από δύο μέρη.

Στο πρώτο μέρος αναλύεται το λογισμικό που αναπτύχθηκε για την οδήγηση μιας κάρτας PCI διαύλου I^2C. Το λογισμικό οδήγησης αναπτύχθηκε σε περιβάλλον Linux και για τη συγγραφή του χρησιμοποιήθηκε η γλώσσα προγραμματισμού C. Αναπτύχθηκαν επίσης απλές εφαρμογές σε γλώσσα προγραμματισμού C και σε περιβάλλον LabVIEW, με σκοπό να αποτελέσουν παραδείγματα χρήσης του εν λόγω λογισμικού οδήγησης.

Το αντικείμενο του δευτέρου μέρους της εργασίας είναι η ανάπτυξη ενός μετατροπέα ψηφιακών σημάτων TTL-ECL-NIM-LVDS. Σκοπός του μετατροπέα είναι η εύκολη διασύνδεση μεταξύ πειραματικών ηλεκτρονικών μονάδων ΦΥΕ που χρησιμοποιούν ψηφιακά σήματα TTL, ECL, NIM και LVDS. Ο μετατροπέας διαθέτει οκτώ ανεξάρτητα κανάλια και λειτουργεί ικανοποιητικά σε συχνότητες έως και 40 MHz.

Abstract

The present MSc thesis, which has been elaborated at the High Energy Physics Laboratory - University of Ioannina, consists of two parts

The first part includes the development of a Linux device driver for a PCI card which controls an I^2C bus. The device driver is based on C programming language. Furthermore, simple C and LabVIEW applications have been developed in order to set an example for the use of the device driver mentioned above.

The second part includes the development of a multi-converter for the TTL-ECL-NIM-LVDS standards. The purpose of this multi-converter is to be used as an interface among the various signal processing units that are used in High Energy Physics experiments. This multi-converter provides eight conversion channels while its maximum operating frequency is 40MHz.

Ευχαριστίες

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τους ανθρώπους που συνέβαλαν σημαντικά στην πραγματοποίηση αυτής της εργασίας.

Τον επιβλέποντα της διπλωματικής μου εργασίας Επίκουρο Καθηγητή Π.Κόκκα, μέλος του Εργαστηρίου Φυσικής Υψηλών Ενεργειών του Πανεπιστημίου Ιωαννίνων, για την καθοδήγησή του καθ' όλη τη διάρκεια εκπόνησης της εργασίας μου.

Τον Καθηγητή Φ.Τριάντη διευθυντή του ΕΦΥΕ, καθώς και τα άλλα μέλη του ΕΦΥΕ Επίκουρους Καθηγητές Ν.Μάνθο και Ι.Ευαγγέλου για τη βοήθεια που μου προσέφεραν και την εμπιστοσύνη που μου έδειξαν.

Τον Καθηγητή Π.Κωσταράκη διευθυντή του Δ-ΠΜΣ-ΣΗΤ για τις πολύτιμες συμβουλές του.

Τους φίλους και συναδέλφους κ Γ.Σιδηρόπουλο, για την άριστη συνεργασία σε όλο αυτό το διάστημα, και κ Α.Ασημίδη, για τη μύηση στα μυστικά του Linux και του PCI.

ΜΕΡΟΣ ΠΡΩΤΟ

ΛΟΓΙΣΜΙΚΟ ΟΔΗΓΗΣΗΣ ΚΑΡΤΑΣ PCI ΔΙΑΥΛΟΥ I^2C

Εισαγωγή

1.1 Βασικες εννοιες 2

1.1.1 Εισαγωγή στο I^2C 2

1.1.2 Εισαγωγή στο PCI 6

1.2.1 Ιστορία 6

1.2.2 Αρχιτεκτονική PCI 6

1.2.3 Πρωτόκολλο επικοινωνίας PCI 8

1.2.4 Αναγνώριση και προσπέλαση 10

1.2.5 Διευθυνσιοδότηση 11

1.2.6 Αρχικοποίηση 12

1.2 ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΚΑΡΤΑΣ PCI 16

1.3 Λογισμικο 20

1.3.1 Το λειτουργικό σύστημα Linux 21

1.3.2 Λογισμικό οδήγησης (Device Driver) για Linux 23

1.3.2.1 Linux Kernel 24

1.3.2.2 Κατηγορίες συσκευών και modules 27

1.3.3 Λογισμικό οδήγησης της κάρτας PCI 30

1.3.3.1 Οδήγηση του S5920 30

1.3.3.2 Οδήγηση του PCF8584 33

1.3.4 Εφαρμογές σε γλώσσα προγραμματισμού C 38

1.3.5 Εφαρμογές σε LabVIEW 5.1 για Linux 43

1.4 Συμπερασματα 50

1.5 Βιβλιογραφια 51

ΠΑΡΑΡΤΗΜΑ 53

1. i2cdrv.c 53

2. i2cdrv.h 60

3. i2c_functions.c 61

4. i2c_functions.h 70

5.i2c_app.c 72

6.ap6.c 73

7.makefile 74

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΜΕΡΟΣ ΠΡΩΤΟ

ΛΟΓΙΣΜΙΚΟ ΟΔΗΓΗΣΗΣ ΚΑΡΤΑΣ PCI ΔΙΑΥΛΟΥ I²C

Εισαγωγή

Στο πρώτο μέρος της παρούσας διπλωματικής εργασίας, η οποία εκπονήθηκε στο Εργαστήριο Υψηλών Ενεργειών (ΕΦΥΕ) του Πανεπιστημίου Ιωαννίνων, αναλύεται το λογισμικό (device driver) που αναπτύχθηκε σε γλώσσα προγραμματισμού C για την οδήγηση μιας κάρτας PCI σε περιβάλλον Linux. Η κάρτα PCI που οδηγείται από το εν λόγω λογισμικό αναπτύχθηκε επίσης στο ΕΦΥΕ από το συνάδελφο κ. Γ. Σιδηρόπουλο με σκοπό την οδήγηση ενός διαύλου I²C (I²C Controller). [1]

Στις παραγράφους που ακολουθούν εξηγούνται αρχικά κάποιοι από τους προαναφερθέντες όρους (I²C, PCI, Linux, device driver), η κατανόηση των οποίων είναι απαραίτητη για την μελέτη της εργασίας και στη συνέχεια ακολουθεί η ανάλυση του λογισμικού οδήγησης. Αναλύονται επίσης και κάποιες εφαρμογές που αναπτύχθηκαν σε γλώσσα προγραμματισμού C και περιβάλλον γραφικού προγραμματισμού LabVIEW, οι οποίες αποτελούν παραδείγματα χρήσης του εν λόγω λογισμικού οδήγησης.

1. Βασικές έννοιες

Στις επόμενες παραγράφους γίνεται μια εισαγωγή στο σειριακό πρωτόκολλο επικοινωνίας I²C και στην αρχιτεκτονική PCI.

1.1 Εισαγωγή στο I²C

Το I²C είναι ένα σειριακό πρωτόκολλο επικοινωνίας που αναπτύχθηκε από τη Philips Semiconductors στις αρχές τις δεκαετίας του 1980. Το όνομά του (I²C ή αλλιώς IIC) είναι το ακρωνύμιο του "Inter-Integrated Circuit" και εξηγεί κυριολεκτικά ποιος είναι ο σκοπός του: να παρέχει διασύνδεση μεταξύ ολοκληρωμένων κυκλωμάτων. Πλέον είναι ιδιαίτερα διαδεδομένο και έχει υιοθετηθεί από τις μεγαλύτερες εταιρίες ημιαγωγών.

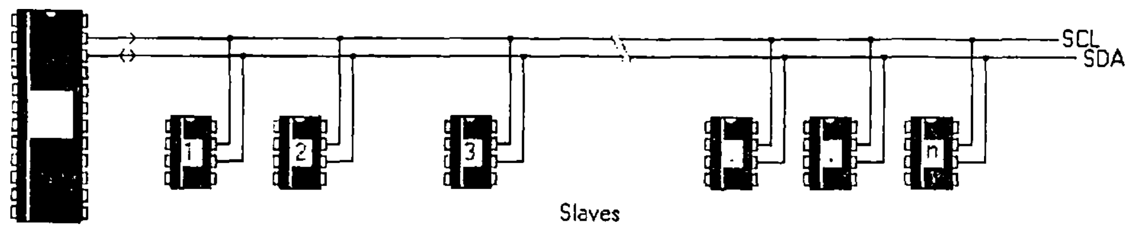
Ο δίαυλος I²C αποτελείται από δύο αμφίδρομους αγωγούς. Ο αγωγός SDA αποτελεί τη γραμμή σειριακής μετάδοσης των δεδομένων (**S**erial **Da**ta) ενώ ο αγωγός SCL τη γραμμή του χρονισμού της σειριακής επικοινωνίας (**S**erial **Cl**ock). Κάθε

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ολοκληρωμένο κύκλωμα που συνδέεται σε ένα δίαυλο έχει την δική του μοναδική διεύθυνση, ανεξαρτήτως τύπου ή πολυπλοκότητας (π.χ. CPU, EEPROM, ψηφιακό ποτενσιόμετρο κτλ). Τα ολοκληρωμένα κυκλώματα ενεργούν είτε ως πομποί είτε ως δέκτες σειριακών δεδομένων, ανάλογα με τη λειτουργία τους. Έτσι, π.χ., ένα ψηφιακό ποτενσιόμετρο λειτουργεί μόνο ως δέκτης σειριακών δεδομένων, σε αντίθεση με μία μνήμη ή μια CPU, οι οποίες μπορούν να λειτουργήσουν είτε σαν δέκτες είτε σαν πομποί. [2]

Μια από τις βασικές έννοιες στο I^2C, όπως και σε άλλα πρωτόκολλα επικοινωνίας, είναι η διάκριση των ολοκληρωμένων κυκλωμάτων σε "master" (κύριος του διαύλου) και "slave" (εξαρτώμενος του διαύλου). Στο πρωτόκολλο επικοινωνίας I^2C, ως "master" θεωρείται το ολοκληρωμένο κύκλωμα που δίνει τις εντολές για τη μετάδοση δεδομένων στο δίαυλο, ελέγχοντας παράλληλα τη γραμμή SCL (Σχ1.1). Άλλωστε, η κυριότητα ενός σειριακού διαύλου είναι στενά συνυφασμένη με τον έλεγχο του σήματος χρονισμού.

Ένα βασικό χαρακτηριστικό του I^2C είναι ότι πρόκειται για ένα "multi-master" δίαυλο επικοινωνίας, κάτι που σημαίνει ότι μπορούν να συνυπάρξουν σε αυτόν περισσότεροι τους ενός "masters". Συνήθως το ρόλο του "master" αναλαμβάνει ένας μικροελεγκτής.



Σχήμα 1.1 Συνήθης διάταξη διαύλου I^2C

Πρωτόκολλο επικοινωνίας

Η επικοινωνία γίνεται ως εξής: Ο "master" διευθυνσιοδοτεί τον "slave" με τον οποίο επιθυμεί να επικοινωνήσει. Οι "slaves" που είναι συνδεδεμένοι στο δίαυλο λαμβάνουν τη διεύθυνση και τη συγκρίνουν με τη δική τους. Αν αυτή συμπίπτει με τη δική τους, τότε ανταποκρίνονται, ενώ στην αντίθετη περίπτωση απλά αγνοούν τα εισερχόμενα δεδομένα και αναμένουν το τέλος της μετάδοσης τους.

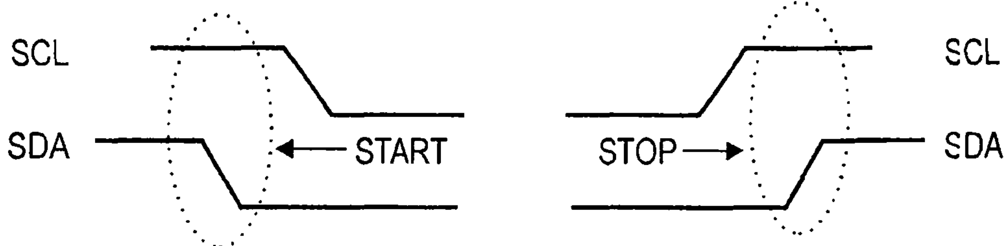
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Κάθε πληροφορία που μεταδίδεται μέσω του διαύλου έχει εύρος 8-bit. Η πληροφορία μεταδίδεται ξεκινώντας από το σημαντικότερο bit (MSB). Ο αριθμός των byte που μπορούν να αποσταλούν μέσω του διαύλου είναι απεριόριστος. [2]

➢ *ΕΝΑΡΞΗ - ΤΕΡΜΑΤΙΣΜΟΣ*

Πριν από κάθε μεταφορά δεδομένων στον δίαυλο I^2C προηγείται μια διαδικασία που ονομάζεται «Διαδικασία έναρξης» (START Condition). Το τέλος της επικοινωνίας μέσω του διαύλου δηλώνεται με μια αντίστοιχη διαδικασία που καλείται «διαδικασία τερματισμού» (STOP Condition).

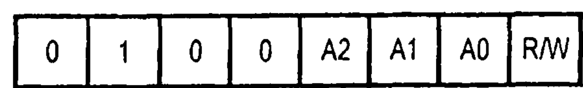
Ως διαδικασία έναρξης θεωρείται μια μετάβαση από υψηλή λογική κατάσταση σε χαμηλή στη γραμμή SDA, καθώς η γραμμή SCL βρίσκεται σε υψηλή λογική κατάσταση ενώ ,αντίστοιχα, ως διαδικασία τερματισμού θεωρείται μια μετάβαση, από χαμηλή λογική κατάσταση σε υψηλή στη γραμμή SDA, καθώς η γραμμή SCL βρίσκεται σε υψηλή λογική κατάσταση (Σχ.1.2).



Σχήμα 1.2 Σήματα έναρξης και τερματισμού

➢ *ΔΙΕΥΘΥΝΣΙΟΔΟΤΗΣΗ*

Η διευθυνσιοδότηση των "slaves" γίνεται με την αποστολή του πρώτου byte που ακολουθεί το σήμα έναρξης από τον "master". Η δομή του byte αυτού ποικίλει ανάλογα με τον τύπο του ολοκληρωμένου κυκλώματος. Σε όλες όμως τις περιπτώσεις, το λιγότερο σημαντικό bit (LSB), το οποίο ονομάζεται "R/W", δηλώνει τον τύπο της επικοινωνίας. Στο Σχ.1.3 απεικονίζεται ένα παράδειγμα byte διευθυνσιοδότησης (του PCF8574). Τα A2, A1 και A0 δηλώνουν τα λογικά επίπεδα στους αντίστοιχους ακροδέκτες του PCF8574.



Σχ.1.3 Byte διευθυνσιοδότησης του PCF8574

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

> *ΕΠΙΒΕΒΑΙΩΣΗ (ACK)*

Κάθε byte πληροφορίας που μεταδίδεται μέσω του διαύλου ακολουθείται από ένα bit επιβεβαίωσης (ACK), στον επόμενο παλμό ρολογιού. Κατά τη διάρκεια αυτού του παλμού, ο πομπός δεδομένων απελευθερώνει τη γραμμή SDA και ο δέκτης την οδηγεί σε χαμηλή λογική κατάσταση, δηλώνοντας την επιτυχία της μετάδοσης.

> *ΕΓΓΡΑΦΗ - ΑΝΑΓΝΩΣΗ*

Στα Σχ.1.4 και Σχ.1.5 απεικονίζονται απλοποιημένα παραδείγματα εγγραφής και ανάγνωσης ενός byte. Κατά τη διαδικασία εγγραφής αποστέλλεται αρχικά το σήμα έναρξης και στη συνέχεια ακολουθεί το byte διευθυνσιοδότησης με το LSB μηδενισμένο. Ακολουθεί το σήμα επιβεβαίωσης (ACK) από τον "slave" που διευθυνσιοδοτήθηκε και στη συνέχεια το byte προς εγγραφή. Ένα νέο σήμα ACK επιβεβαιώνει την εγγραφή και διαδικασία τελειώνει με το σήμα τερματισμού.

Σχ.1.4 Διαδικασία εγγραφής

Ομοίως, κατά τη διαδικασία ανάγνωσης αποστέλλεται αρχικά το σήμα έναρξης και στη συνέχεια ακολουθεί το byte διευθυνσιοδότησης, με το LSB όμως ίσο με 1. Ακολουθεί το σήμα επιβεβαίωσης (ACK) από τον "slave" που διευθυνσιοδοτήθηκε και στη συνέχεια γίνεται ανάγνωση του byte από το "master". Η διαδικασία τελειώνει με τη μη αποστολή ACK από το "master" (υψηλή λογική κατάσταση), ακολουθούμενη από το σήμα τερματισμού.

Σχ.1.5 Διαδικασία ανάγνωσης

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

1.2. Εισαγωγή στο PCI

1.2.1. Ιστορία του PCI

Η επικοινωνία και η μεταφορά των δεδομένων μεταξύ της κεντρικής μονάδας επεξεργασίας (CPU) και των περιφερειακών μονάδων σε έναν υπολογιστή, επιτυγχάνεται με τη βοήθεια ενός διαύλου δεδομένων (bus). Τα πρώτα PC χρησιμοποιούσαν ένα δίαυλο δεδομένων, του οποίου ο ρυθμός μετάδοσης των δεδομένων έφτανε ως και μερικές εκατοντάδες Kbytes/sec. Από τους παλαιότερους διαύλους, ο δίαυλος ISA χρησιμοποιείται ακόμη και σήμερα, αλλά το εύρος δεδομένων των 16bit, η μέγιστη συχνότητα ρολογιού των 8MHz, καθώς και ο μέγιστος ρυθμός μετάδοσης δεδομένων των 16Mbytes/sec, τον καθιστούν ανεπαρκή για τις νέες εφαρμογές.

Ο δίαυλος PCI (**P**eripheral **C**omponent **I**nterconnect local bus), αναπτύχθηκε από την Intel και η πρώτη έκδοση των προδιαγραφών του (PCI specification version 1.0) έγινε στις 22 Ιουνίου του 1992. Η νεότερη έκδοση του PCI Specification (version 2.2) έγινε το Φεβρουάριο του 1999. Ο δίαυλος PCI με εύρος δεδομένων 32bit και συχνότητα ρολογιού 33MHz επιτυγχάνει μέγιστο ρυθμό μετάδοσης δεδομένων 132Mbytes/sec και δίνει τη δυνατότητα στα PC να διαχειριστούν τις νεότερες και περισσότερο απαιτητικές εφαρμογές. Στη νεότερη έκδοση του PCI Specification, υποστηρίζεται εύρος δεδομένων έως και 64 bit, καθώς και συχνότητα ρολογιού 66MHz, κάτι που αυξάνει το ρυθμό μετάδοσης των δεδομένων σε 528 Mbytes/sec. [3]

Η αρχιτεκτονική PCI έχει υιοθετηθεί πλέον ως πρότυπο, υπό την επίβλεψη ενός οργανισμού που ονομάζεται PCI Special Interest Group (PCISIG). Ο δίαυλος PCI εμφανίστηκε για πρώτη φορά με το chipset "Saturn" της μητρικής κάρτας "Alfredo" που κατασκεύασε η Intel το 1994, για τους επεξεργαστές i486. Σκοπός της αρχιτεκτονικής PCI ήταν περισσότερο να συμπληρώσει παρά να αντικαταστήσει τους παραδοσιακούς διαύλους I/O (ISA, EISA, Micro Channel).

1.2.2 Αρχιτεκτονική PCI

Η επικοινωνία μεταξύ του διαύλου PCI και του τοπικού διαύλου της CPU (CPU local bus ή FSB), γίνεται μέσω ενός ολοκληρωμένου κυκλώματος που ονομάζεται ελεγκτής PCI ή γέφυρα PCI (PCI bridge/controller). Όταν η CPU επιθυμεί αποστολή

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

δεδομένων σε κάποιο περιφερειακό στο δίαυλο PCI, τα δεδομένα αυτά αποθηκεύονται στον ελεγκτή PCI, επιτρέποντας με αυτόν τον τρόπο στη CPU να κάνει κάποια άλλη λειτουργία, αντί να περιμένει πότε θα ολοκληρωθεί η μετάδοση δεδομένων. Ο ελεγκτής PCI αναλαμβάνει τη μετάδοση των δεδομένων προς το περιφερειακό με τη μεγαλύτερη δυνατή ταχύτητα. Η αρχιτεκτονική PCI υποστηρίζει και τη δυνατότητα ύπαρξης masters, που μπορούν να αποκτήσουν τον έλεγχο του και να εκτελέσουν διαδικασίες ανεξάρτητες της CPU.

Ή αρχιτεκτονική του διαύλου PCI είναι σύγχρονη (synchronous), δηλ. Ο χρονισμός της μεταφοράς των δεδομένων γίνεται με βάση το ρολόι του συστήματος (CLK). Στην πρώτη έκδοση των προδιαγραφών του PCI, η μέγιστη συχνότητα του CLK ήταν 33MHz. Αν και στις επόμενες εκδόσεις των προδιαγραφών του PCI υποστηρίζεται συχνότητα λειτουργίας μέχρι και 66MHz, στη συντριπτική πλειοψηφία των σημερινών PC χρησιμοποιείται αρχιτεκτονική PCI με μέγιστη συχνότητα λειτουργίας 33MHz.

Η αρχιτεκτονική PCI υλοποιεί έναν δίαυλο των 32-bit (AD[31:0]), στον οποίο πολυπλέκονται οι δίαυλοι διευθυνσιοδότησης (address bus) και δεδομένων (data bus) των 32-bit. Βέβαια, αν και σύμφωνα με την τελευταία έκδοση των προδιαγραφών του PCI (v2.2) υποστηρίζεται και εύρος δεδομένων 64-bit με τη χρήση ενός μεγαλύτερου connector, στα περισσότερα σημερινά PC εφαρμόζεται η 32-bit μετάδοση δεδομένων των 32-bit, με τη χρήση του βασικού connector του PCI.

Το κέρδος που προκύπτει από την πολυπλεξία των διαύλων διευθυνσιοδότησης και δεδομένων είναι η αισθητή μείωση του αριθμού των απαιτούμενων ακροδεκτών, κάτι που έχει ως αποτέλεσμα τη μείωση του κόστους και του μεγέθους των σχετικών εξαρτημάτων. Μια τυπική κάρτα PCI χρησιμοποιεί περίπου 50 ακροδέκτες, εκ των οποίων οι 32 αντιστοιχούν στον πολυπλεγμένο δίαυλο AD[31:0].

1.2.3 Πρωτόκολλο επικοινωνίας PCI

Η μεταφορά δεδομένων σύμφωνα με το πρωτόκολλο PCI γίνεται ως εξής: Ένας κύκλος μεταφοράς δεδομένων ξεκινά οδηγώντας την διεύθυνση των 32-bit στο δίαυλο AD[31:0], κατά τη διάρκεια του πρώτου παλμού του ρολογιού (φάση διευθυνσιοδότησης ή address phase). Η φάση της διευθυνσιοδότησης σηματοδοτείται με την ενεργοποίηση του σήματος FRAME#. Με τον επόμενο παλμό ακολουθεί η μετάδοση των δεδομένων μέσω του ίδιου διαύλου, η οποία γίνεται σε μία ή περισσότερες φάσεις (data phases), μια για κάθε παλμό ρολογιού.

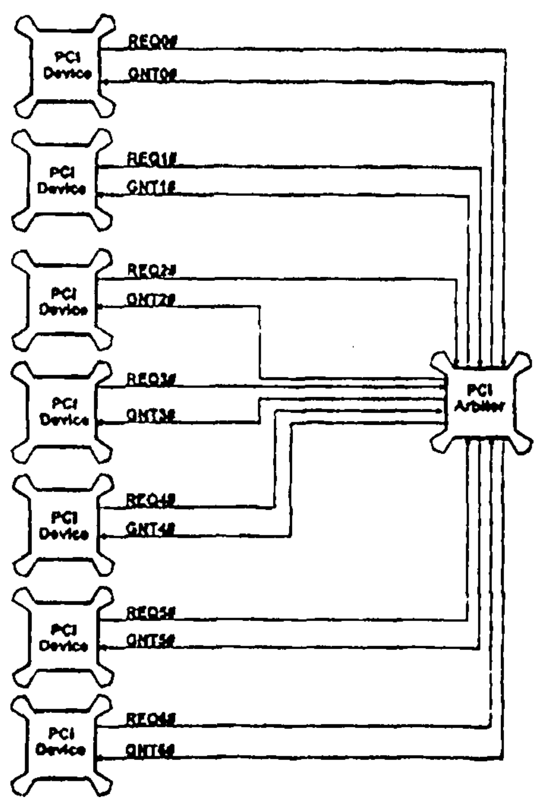
Η μεταφορά δεδομένων γίνεται μεταξύ ενός "initiator" (αρχικοποιητής) και ενός "target" (στόχος). Οι όροι "initiator" και "target" στην ορολογία του PCI αντιστοιχούν στους όρους "bus master" και "bus slave", που χρησιμοποιούνται γενικότερα στην ορολογία των πρωτοκόλλων επικοινωνίας. Κατά τη διάρκεια της φάσης διευθυνσιοδότησης ο "initiator" οδηγεί κατάλληλα τα σήματα C/BE[3:0], για να επιλέξει τον τύπο της μεταφοράς δεδομένων (εγγραφή ή ανάγνωση περιοχής μνήμης ή I/O). Κατά τη διάρκεια της φάσης μετάδοσης των δεδομένων τα σήματα C/BE[3:0], χρησιμοποιούνται ως "byte enable", δηλαδή υποδεικνύουν ποια από τα 4 bytes του διαύλου AD[31:0] χρησιμοποιούνται σε κάθε φάση μετάδοσης δεδομένων. Ο "initiator" αλλά και ο "target" μπορεί να εισάγει καταστάσεις αναμονής (wait states) κατά τη μεταφορά δεδομένων, απενεργοποιώντας τα σήματα TDRY# και IRDY#. Όταν τα σήματα αυτά είναι απενεργοποιημένα, οποιαδήποτε μεταφορά δεδομένων θεωρείται άκυρη, και τα δεδομένα αγνοούνται.

Όπως προαναφέρθηκε, η ανταλλαγή δεδομένων μέσω του διαύλου PCI αποτελείται από μία φάση διευθυνσιοδότησης και από μία ή περισσότερες φάσεις μεταφοράς δεδομένων. Στην περίπτωση που μεταφέρονται δεδομένα από ή προς κάποιον καταχωρητή του PCI "target" (λειτουργία I/O), συνήθως υπάρχει μόνο μία φάση μεταφοράς δεδομένων. Στην περίπτωση μεταφοράς δεδομένων από και προς τη μνήμη, εκτελούνται διαδοχικές φάσεις μεταφοράς δεδομένων , καθώς μεταφέρονται ολόκληρα πακέτα δεδομένων από ή προς διαδοχικές θέσεις μνήμης. Ο τερματισμός της μεταφοράς δεδομένων σηματοδοτείται είτε από τον "initiator" είτε από τον "target" με την ενεργοποίηση του σήματος STOP#. Αν το σήμα STOP# ενεργοποιηθεί χωρίς να έχει ολοκληρωθεί καμία φάση μεταφοράς δεδομένων, συνεπάγεται ότι ο "target" ζητά επανάληψη (retry) της διαδικασίας. Στην περίπτωση που το σήμα STOP# ενεργοποιηθεί μετά την ολοκλήρωση τουλάχιστο μιας φάσης

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

μεταφοράς δεδομένων, συνεπάγεται ότι ο "target" ζητά αποσύνδεση (disconnect). Οι "initiators" διεκδικούν την κυριότητα του διαύλου ενεργοποιώντας το σήμα REQ# που οδηγείται στο ολοκληρωμένο κύκλωμα που ρυθμίζει την απόδοση της κυριότητας του διαύλου και ονομάζεται PCI "arbiter" (διαιτητής) . O "arbiter" αποδίδει την κυριότητα του διαύλου σε κάποιον από τους "initiators" ενεργοποιώντας το σήμα GNT#. Για κάθε υποδοχή (slot) PCI, αντιστοιχεί ένα σήμα GNT# και ένα REQ#, για την εκτέλεση του αλγορίθμου απόδοσης της κυριότητας του διαύλου από τον "arbiter". O αλγόριθμος απόδοσης της κυριότητας του διαύλου είναι ασύγχρονος, δηλ δεν γίνεται σύμφωνα με το ρολόι του συστήματος, και εκτελείται παράλληλα με τη μεταφορά των δεδομένων που γίνεται την ίδια στιγμή, καθορίζοντας τον επόμενο κύριο του διαύλου. Η αρχιτεκτονική PCI υποστηρίζει έναν αυστηρό μηχανισμό αυτόματης ρύθμισης κατά την εκκίνηση του υπολογιστή, για την αναγνώριση του τύπου της κάρτας (SCSI, Ethernet κτλ), της εταιρίας καθώς και τη ρύθμιση άλλων πολύ σημαντικών παραμέτρων για την ορθή λειτουργία της. [4]

Η αρχιτεκτονική PCI υποστηρίζει λειτουργία με λογικά επίπεδα είτε 5V είτε 3.3V. Τα παλιότερα συστήματα υποστήριζαν αποκλειστικά λειτουργία με 5V και δεν παρείχαν τροφοδοσία 3.3V στους αντίστοιχους ακροδέκτες, αλλά σήμερα



Σχήμα 1.6 Απόδοση κυριότητας διαύλου

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

υποστηρίζονται κάρτες και των δύο τύπων. Για να αποφευχθεί η προσαρμογή μια κάρτας σε έναν PCI Connector με διαφορετική τάση λειτουργίας από αυτήν ποι απαιτεί η κάρτα, οι connectors έχουν διαφορετικό σχήμα ανάλογα με την τάσι λειτουργίας τους.

Στις προδιαγραφές του PCI συμπεριλαμβάνονται και τα χαρακτηριστικά των ολοκληρωμένων κυκλωμάτων που χρησιμοποιούνται, καθώς θα πρέπει να διασφαλιστεί η απαραίτητη ποιότητα των σημάτων που απαιτείται για τη λειτουργία στα 33 ή 66MHz. Τα τυπικά ολοκληρωμένα κυκλώματα τύπου TTL που χρησιμοποιούνταν σε παλαιότερους διαύλους, όπως οι δίαυλοι ISA και EISA, δεν καλύπτουν τις απαιτήσεις του PCI και για αυτό το λόγο τα PCI ολοκληρωμένα κυκλώματα υλοποιούνται σε ASICs.

Η υψηλή ταχύτητα του PCI περιορίζει τον αριθμό των υποδοχών σε έναν δίαυλο PCI σε 3 ή 4, σε αντίθεση με παλαιότερες αρχιτεκτονικές όπου ο αριθμός αυτός έφτανε τις 6 ή 7. Για την επίτευξη της επέκτασης του διαύλου σε περισσότερα των τεσσάρων slots, ο οργανισμός PCISIG προτείνει το μηχανισμό "PCI-to-PCI Bridge" (γεφύρωση διαύλων PCI). Τα "PCI-to-PCI Bridges" είναι ASICs που απομονώνουν ηλεκτρικά δύο διαύλους PCI, επιτρέποντας όμως τη μεταφορά δεδομένων από τον ένα δίαυλο στον άλλο. Κάθε ένα "PCI-to-PCI Bridge" έχει έναν "πρωτεύων" και ένα "δευτερεύων" δίαυλο PCI. Με αυτόν τον τρόπο, μπορούν να συνδεθούν διαδοχικά, δημιουργώντας ένα σύστημα με πολλούς διαύλους PCI.

1.2.4. Αναγνώριση και προσπέλαση (PCI access)

Σε αυτή την παράγραφο αναλύεται η διαδικασία αναγνώρισης και προσπέλασης του hardware από έναν PCI driver. Όπως έχει προαναφερθεί, η αρχιτεκτονική PCI σχεδιάστηκε για να αντικαταστήσει το δίαυλο ISA, με τρεις βασικούς στόχους:

- μεγαλύτερη ταχύτητα μεταφοράς δεδομένων
- συμβατότητα με όλες τις υπολογιστικές πλατφόρμες
- αυτοματοποίηση της διαδικασίας προσθήκης ή αφαίρεσης ενός περιφερειακού

Η αύξηση του ρυθμού μεταφοράς δεδομένων επιτεύχθηκε με την συχνότητα χρονισμού του διαύλου (33, 66 ή ακόμη και 133MHz) και το εύρος του διαύλου δεδομένων (32 ή 64 bit), όπως αναφέρεται και στην παράγραφο 1.2.2.

Η συμβατότητα σχεδόν με όλες τις πλατφόρμες επιτεύχθηκε επίσης, δίνοντας λύση σε ένα πρόβλημα που ταλάνιζε την υπολογιστική κοινότητα εδώ και χρόνια,

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

καθώς τα περισσότερα πρότυπα επικοινωνίας σχεδιάζονταν για συγκεκριμένες οικογένειες επεξεργαστών. Η αρχιτεκτονική PCI χρησιμοποιείται ευρέως και σε συστήματα με επεξεργαστές i386, IA-32, Alpha, PowerPC, SPARC64, IA-64 κτλ.

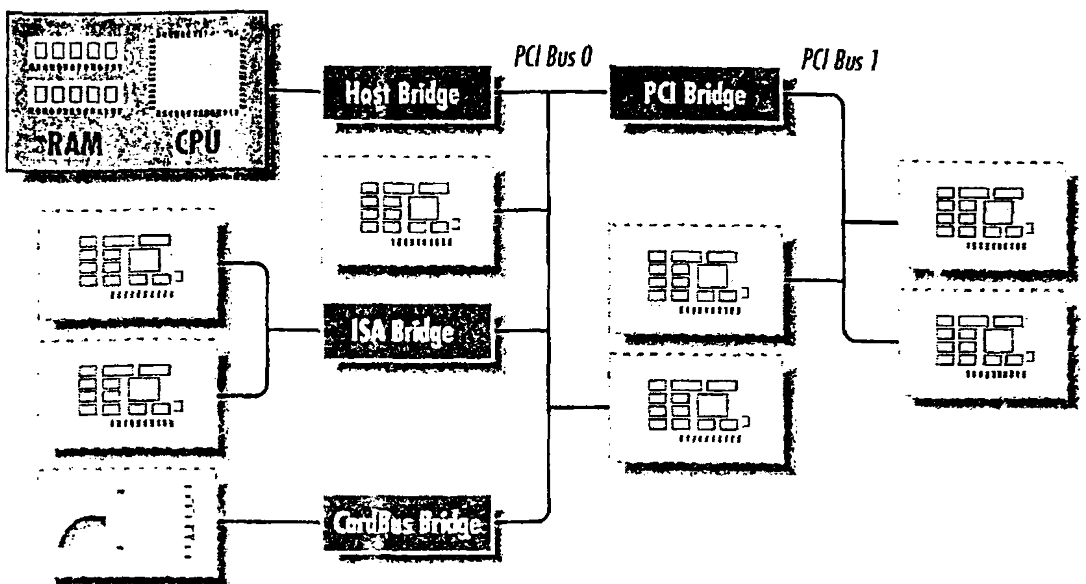
Ωστόσο, η δυνατότητα αυτόματης αναγνώρισης και ρύθμισης των PCI περιφερειακών ήταν αυτή που αποτέλεσε το βασικότερο πλεονέκτημα του PCI σε σχέση με τις παλαιότερες αρχιτεκτονικές διαύλων. Το ζήτημα της αυτόματης αναγνώρισης και ρύθμισης των περιφερειακών είναι αυτό που σχετίζεται περισσότερο με τη συγγραφή ενός PCI driver. Οι κάρτες PCI, σε αντίθεση με παλαιότερα περιφερειακά, ρυθμίζονται αυτόματα (auto-configuration) κατά την εκκίνηση του συστήματος (boot). Ο ρόλος του λογισμικού οδήγησης είναι να πάρει τις απαραίτητες πληροφορίες από το περιφερειακό, έτσι ώστε να επιτευχθεί η αρχικοποίησή του. Οι πληροφορίες για το λειτουργικό λαμβάνονται από έναν αριθμό καταχωρητών που ονομάζονται configuration registers (καταχωρητές ρυθμίσεων) και βρίσκονται σε μια περιοχή διευθύνσεων μεγέθους 256 bytes που υλοποιείται σε κάθε περιφερειακό PCI. Η περιοχή αυτή ονομάζεται configuration space (περιοχή ρυθμίσεων) και αναλύεται στη συνέχεια. [5]

1.2.5. Διευθυνσιοδότηση (PCI Addressing)

Κάθε περιφερειακό PCI αναγνωρίζεται από τον αριθμό διαύλου (bus number), συσκευής (device number) και λειτουργιών (function number). Σύμφωνα με τις προδιαγραφές του PCI, ένα σύστημα επιτρέπεται να φιλοξενεί μέχρι 256 διαύλους. Κάθε δίαυλος μπορεί να δεχτεί μέχρι και 32 συσκευές, η κάθε μία εκ των οποίων μπορεί να υλοποιεί μέχρι και 8 λειτουργίες (functions). Συνολικά, η κάθε λειτουργία μπορεί να αντιστοιχηθεί με ένα αριθμό των 16-bit.

Σε κάθε σύστημα φιλοξενούνται τουλάχιστο δυο δίαυλοι PCI. Η παρουσία περισσοτέρων του ενός διαύλου PCI επιτυγχάνεται, όπως έχει προαναφερθεί, με ολοκληρωμένα κυκλώματα "PCI-to-PCI bridge" που διασυνδέουν το βασικό δίαυλο PCI (bus 0), με τους υπολοίπους. Ένα τυπικό σύστημα PCI απεικονίζεται στο Σχ.1.7.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 1.7 Πρωτεύων [Bus 0] και δευτερεύων [Bus 1] δίαυλος PCI

Τα κυκλώματα των PCI περιφερειακών αλληλεπιδρούν με τρεις περιοχές διευθύνσεων:

- θέσεις μνήμης
- θύρες I/O
- περιοχή ρυθμίσεων.

Οι δύο πρώτες περιοχές διευθύνσεων είναι κοινές για όλες τις συσκευές που μοιράζονται το δίαυλο PCI, και έτσι η προσπέλαση μιας θέσης μνήμης γίνεται αντιληπτή από όλες τις συσκευές που είναι προσαρτημένες στο δίαυλο. Αντιθέτως, η προσπέλαση του configuration space γίνεται σε μία υποδοχή PCI κάθε φορά, αποφεύγοντας έτσι τυχόν διενέξεις (conflicts). [5]

1.2.6. Αρχικοποίηση

Καταρχήν, θα πρέπει να καταστεί σαφές ότι η μεγαλύτερη καινοτομία του προτύπου PCI σε σχέση με το ISA είναι η περιοχή ρυθμίσεων. Έτσι, στο λογισμικό οδήγησης θα πρέπει να περιλαμβάνεται και η δυνατότητα πρόσβασης σε αυτήν την περιοχή διευθύνσεων.

Κατά την εκκίνηση του συστήματος, εφαρμόζεται μεν η τάση τροφοδοσίας στα περιφερειακά PCI , αλλά το υλικό (hardware) παραμένει ανενεργό. Με την εφαρμογή της τάσης, δεν αντιστοιχείται (mapping) ακαριαία η περιοχή μνήμης και I/O που

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

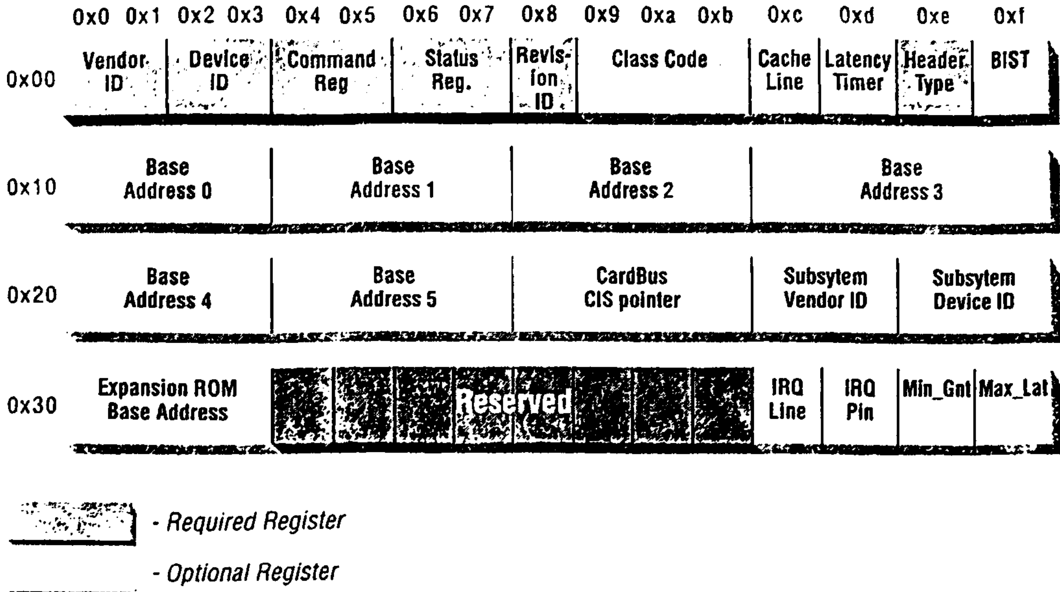
απαιτείται από την εκάστοτε συσκευή PCI στην περιοχή διευθύνσεων του υπολογιστή. Επίσης, οποιαδήποτε άλλη σχετική λειτουργία όπως π.χ. IRQ είναι επίσης απενεργοποιημένη. Το ρόλο αυτό αναλαμβάνει το BIOS, το οποίο δίνει τη δυνατότητα εγγραφής και ανάγνωσης των καταχωρητών του ελεγκτή PCI.

Το BIOS, αλληλεπιδρά διαδοχικά με κάθε περιφερειακό PCI με σκοπό την εύρεση και τη σωστή κατανομή των απαραίτητων πόρων που απαιτεί το καθένα για την ορθή λειτουργία του. Όταν το λογισμικό οδήγησης αποκτήσει πρόσβαση στο περιφερειακό, μετά από τη φόρτωση του λειτουργικού συστήματος, έχει ήδη ολοκληρωθεί η διαδικασία αντιστοίχησης στην περιοχή μνήμης και I/O του επεξεργαστή. Το λογισμικό οδήγησης μπορεί βέβαια να αλλάξει εκ των υστέρων τις αρχικές ρυθμίσεις του BIOS, αλλά δεν υπάρχει λόγος για να γίνει αυτό. [5]

Η ταυτότητα και οι απαιτήσεις του κάθε περιφερειακού, δηλώνονται από τον κατασκευαστή στους καταχωρητές ρυθμίσεων. Όπως προαναφέρθηκε, κάθε συσκευή PCI περιλαμβάνει ένα χώρο 256 bytes (PCI configuration space), στα πρώτα 64 byte του οποίου δηλώνονται οι τιμές των καταχωρητών ρυθμίσεων. Οπως φαίνεται και στο Σχ.1.21α, κάποιοι από αυτούς τους καταχωρητές είναι απαραίτητοι, ενώ η χρήση των υπολοίπων είναι προαιρετική. Τα προαιρετικά πεδία δεν λαμβάνονται υπόψη, εκτός και αν οι τιμές που αναγράφονται είναι έγκυρες. [3]

Η περιγραφή όλων των στοιχείων που δηλώνονται στην περιοχή ρυθμίσεων ξεφεύγει από τα πλαίσια αυτής της εργασίας. Ωστόσο, γίνεται μια σύντομη αναφορά στα σημεία που κρίνονται απαραίτητα.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 1.8 Καταχωρητές ρυθμίσεων (Configuration Registers)

Ταυτοποίηση

Τρεις, και κάποιες φορές πέντε, από τους καταχωρητές αυτούς χρησιμοποιούνται για την ταυτοποίηση του περιφερειακού.
Πιο συγκεκριμένα:

- *VendorID*

Πρόκειται για ένα καταχωρητή των 16-bit ο οποίος δηλώνει τον κατασκευαστή του περιφερειακού. Έτσι π.χ. τα περιφερειακά της εταιρείας Intel έχουν ως διακριτικό γνώρισμα την τιμή 8086. Οι αριθμοί αυτοί καθορίζονται από τον οργανισμό PCISIG και είναι μοναδικοί για κάθε κατασκευαστή.

- *DeviceID*

Πρόκειται για έναν άλλο καταχωρητή των 16-bit, του οποίου η τιμή αυτή τη φορά επιλέγεται από τον κατασκευαστή, χωρίς την παρέμβαση του PCISIG. Αυτή η τιμή μαζί με την τιμή του vendor id αναφέρεται συχνά και ως υπογραφή (signature) της κάθε συσκευής.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- *Class*

Κάθε PCI περιφερειακό PCI ανηκει σε μία κατηγορία (class), έτσι όπως έχουν οριστεί από τσ PCISIG. Ο αντίστοιχος καταχωρητής έχει εύρος 16-bit, εκ των οποίων τα πρώτα 8 bit δηλώνουν τη γενική κατηγορία (base class ή group). Έτσι π.χ. τα "ethernet" και "token ring" ανήκουν στο group των περιφερειακών δικτύου.

- *Subsystem vendorID & Subsystem deviceID*

Αυτά τα πεδία χρησιμοποιούνται για πιο λεπτομερή αναγνώριση του περιφερειακού και χρησιμοποιούνται κυρίως σε περιπτώσεις "on-board" συσκευών, που χρήζουν τελείως διαφορετικής αντιμετώπισης.

Απαιτήσεις

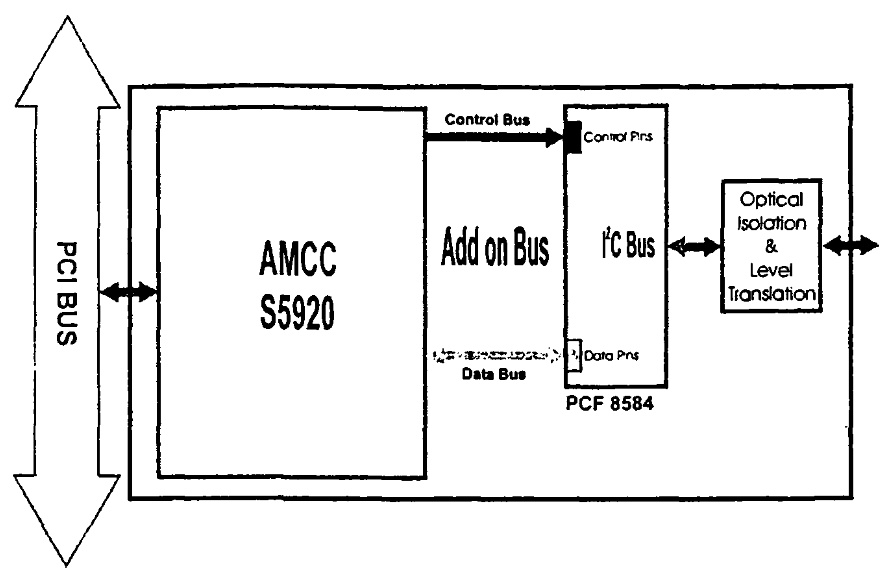
- *Base address registers*

Οι *Base address registers* (ή αλλιώς BARs) αποτελούν μια άλλη πολύ σημαντική κατηγορία καταχωρητών που σχετίζεται με τις απαιτήσεις του κάθε περιφερειακού σε περιοχή μνήμης και I/O. Αν και η υλοποίηση τους είναι προαιρετική χρησιμοποιείται σχεδόν πάντα τουλάχιστο ένας από αυτός (BAR0). Σε αυτούς δηλώνεται το μέγεθος της περιοχής μνήμης ή I/O που απαιτείται για την ορθή λειτουργία του κάθε περιφερειακού. Η κατανομή των απαραίτητων πόρων του συστήματος επιτυγχάνεται με την αλληλεπίδραση των BARs με το BIOS. [3]

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2. Περιγραφή της κάρτας PCI

Ο ρόλος της κάρτας PCI που αναπτύχθηκε, όπως έχει προαναφερθεί, είναι η οδήγηση ενός διαύλου I2C (I2C Controller). Πρόκειται για μία "target"[1] κάρτα των 32-bit, πλήρως συμβατή με τις προδιαγραφές PCI v2.2, με συχνότητα λειτουργίας τα 33MHz. Η συχνότητα του σήματος χρονισμού SCL του διαύλου I^2C είναι τα 90kHz. [1]

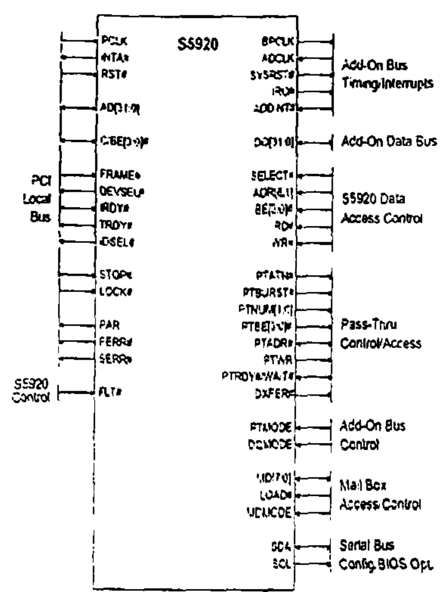


Σχήμα 1.9 Διάγραμμα βαθμίδων της κάρτας PCI

Η κάρτα αποτελείται από τρεις βαθμίδες:

Ο ρόλος της πρώτης είναι η μετατροπή των σημάτων του διαύλου PCI σε έναν παράλληλο δίαυλο των 32-bit. Με τη βοήθεια του διαύλου αυτού ελέγχεται η λειτουργία του ελεγκτή του διαύλου I^2C (I^2C master controller) της δεύτερης βαθμίδας του οποίου τα σήματα SDA και SCL οδηγούνται στην έξοδο μέσω της τρίτης βαθμίδας οπτικής απομόνωσης και προσαρμογής των επιπέδων τάσης. Στο Σχ.1.9 *απεικονίζεται το διάγραμμα βαθμίδων της κάρτας.*

Η πρώτη βαθμίδα αποτελείται από το PCI "bridge" (ή PCI bus target interface) S5920 της εταιρείας Applied Micro Circuits Corporation ή AMCC (Σχ.1.10).



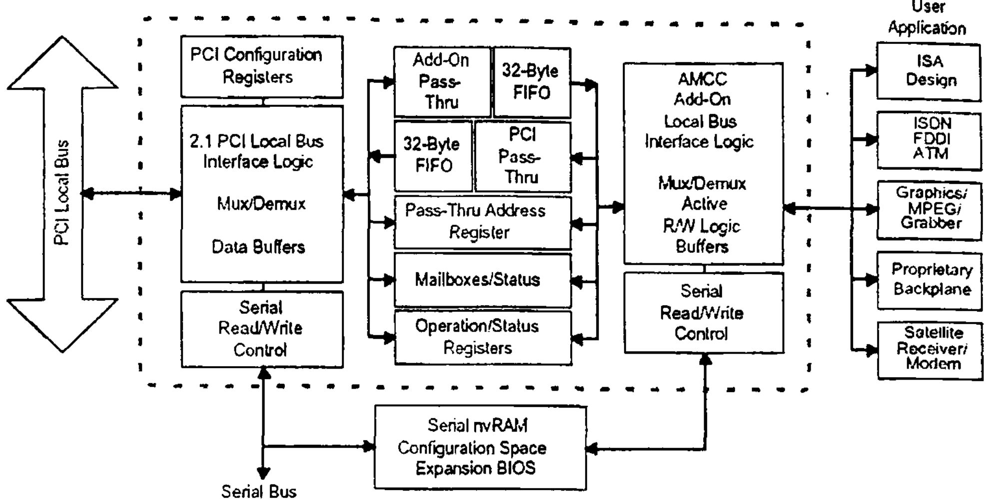
Σχήμα 1.10 AMCC S5920

Το ολοκληρωμένο S5920, το οποίο υποστηρίζει μέγιστη συχνότητα λειτουργίας 33MHz και μέγιστο ρυθμό μετάδοσης δεδομένων 132Mbytes/sec, αποτελεί μια ευέλικτη και εύχρηστη λύση για τους σχεδιαστές που θέλουν να προσαρμόσουν εφαρμογές στο δίαυλο PCI. Με τη χρήση του S5920 εξαλείφεται η αναγκαιότητα κατανόησης του χρονικού συσχετισμού μεταξύ των πολύπλοκων σημάτων του διαύλου PCI, καθώς και ο χρόνος που απαιτείται για την επίτευξη συμβατότητας με τις προδιαγραφές του PCI. Το S5920 μετατρέπει τα περίπλοκα σήματα του διαύλου PCI σε έναν εύχρηστο παράλληλο δίαυλο των 8, 16 ή 32bit που αναφέρεται συχνά και ως "Add-on bus". Τα σήματα του Add-on bus, που διακρίνονται και στο Σχ.1.9, συνθέτουν έναν κατά πολύ απλούστερο δίαυλο σε σχέση με αυτόν του PCI, διευκολύνοντας κατά πολύ με αυτόν τον τρόπο τον σχεδιαστή. [6]

Στην παράγραφο 1.3.3 περιγράφεται η μέθοδος με την οποία μεταφέρονται τα δεδομένα για την οδήγηση του ελεγκτή του διαύλου I^2C από τον δίαυλο PCI στο "Add-on bus". Στο Σχ.1.11 απεικονίζεται το διάγραμμα βαθμίδων του S5920.

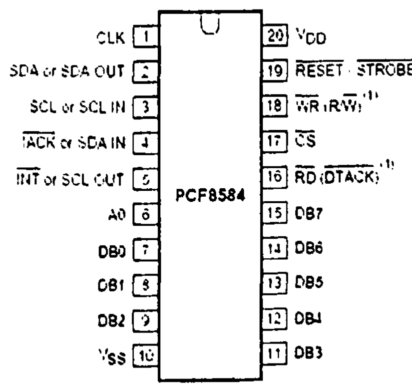
[1] Οι "target" κάρτες PCI δεν μπορούν να αποκτήσουν τον έλεγχο του διαύλου PCI

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

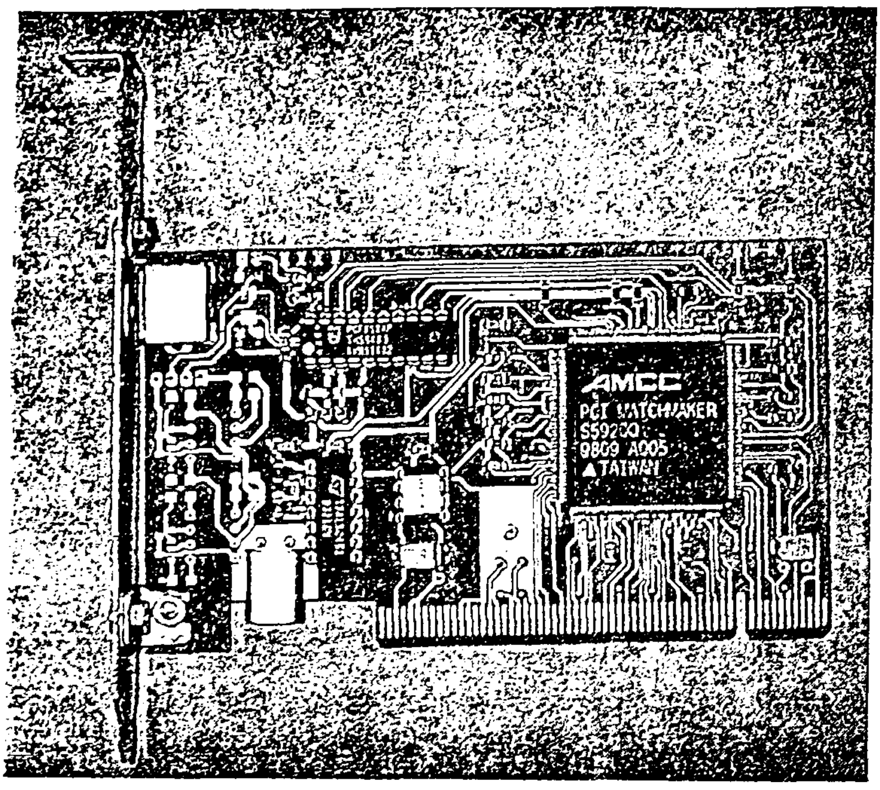


Σχήμα 1.11 Διάγραμμα βαθμίδων του S5920

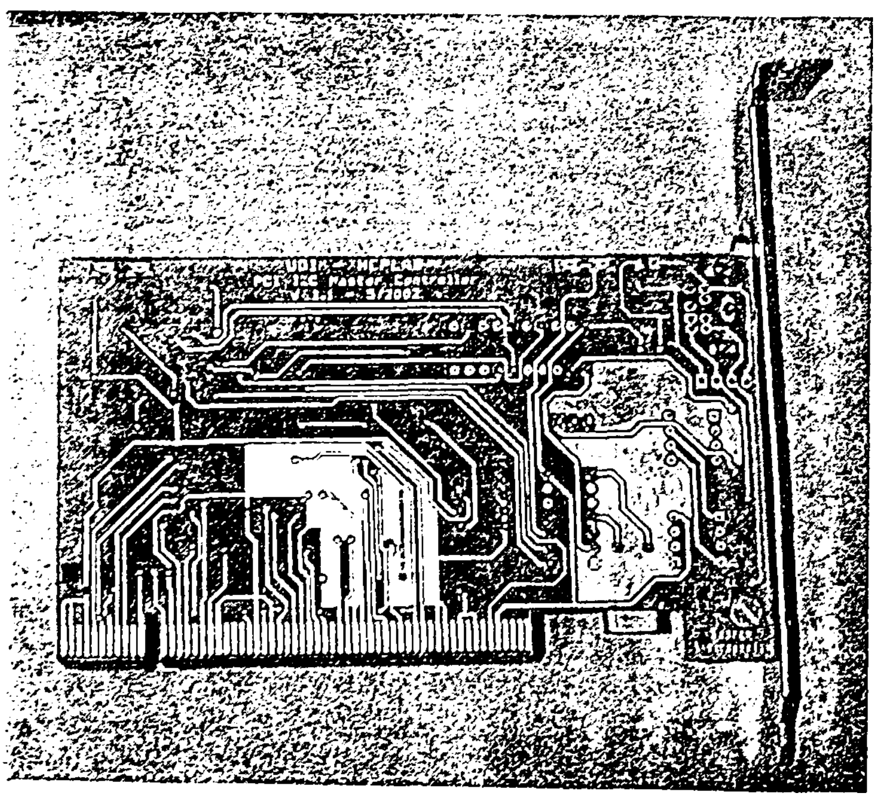
Η δεύτερη βαθμίδα αποτελείται από τον ελεγκτή διαύλου I²C PCF8584 της εταιρείας Philips Semiconductors (Σχ.1.12). Το PCF8584 λειτουργεί ως διεπαφή (interface) μεταξύ ενός παράλληλου διαύλου και του σειριακού διαύλου I²C. Η μεταφορά δεδομένων από και προς τον σειριακό δίαυλο βασίζεται σε εντολές που δίνονται στον αμφίδρομο δίαυλο δεδομένων των 8-bit (DB[7:0]) σε συνδυασμό με τα κατάλληλα σήματα στους 5 ακροδέκτες ελέγχου (CS, WR, RD, A0, RESET) [7]. Το PCF8584 χρονίζεται εξωτερικά από ένα κρύσταλλο 8MHz. Ο τρόπος με τον οποίο οδηγείται το PCF8584 περιγράφεται αναλυτικά στην παράγραφο 1.3.4.



Σχήμα 1.12 PCF8584



Σχήμα 1.13 Άποψη της κάρτας



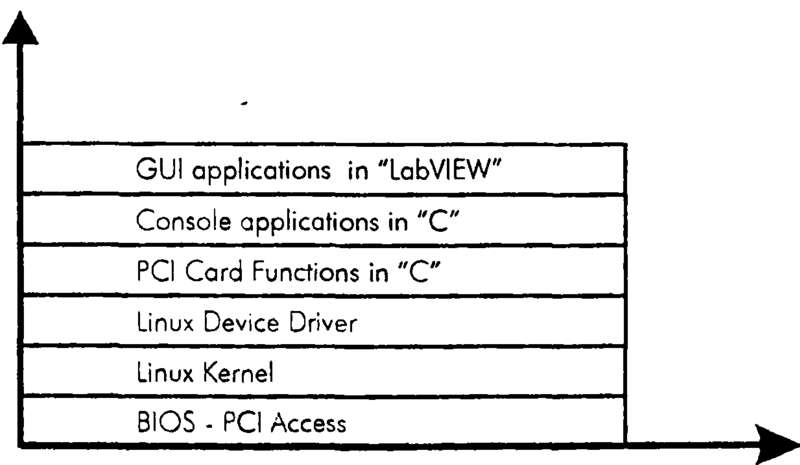
Σχήμα 1.14 Κάτοψη της κάρτας

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

3. Λογισμικό

Εισαγωγή

Στις παραγράφους που ακολουθούν αναλύεται η συγγραφή του λογισμικού οδήγησης (device driver) της εν λόγω κάρτας για το λειτουργικό σύστημα Linux καθώς και κάποιες εφαρμογές, που αποτελούν παράδειγμα για την χρήση της. Το λογισμικό οδήγησης της κάρτας έχει γραφεί σε γλώσσα προγραμματισμού C, όπως και κάποιες από τις εφαρμογές που δίνονται ως παραδείγματα χρήσης της. Εκτός αυτών, συμπεριλαμβάνονται επίσης και εφαρμογές σε γραφικό περιβάλλον LabVIEW. Πριν όμως από την ανάλυση του λογισμικού οδήγησης της κάρτας προηγούνται κάποια εισαγωγικά για το λειτουργικό σύστημα Linux, καθώς και κάποια για τους device drivers για Linux. Στο Σχ.1.15 απεικονίζεται η διαστρωμάτωση των επιπέδων από την χαμηλού επιπέδου προσπέλαση της κάρτας από το BIOS έως στον υψηλού επίπεδου προγραμματισμό σε περιβάλλον LabVIEW.



Σχήμα 1.15 διαστρωμάτωση επιπέδων προγραμματισμού

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

3.1 Το λειτουργικό σύστημα Linux

Το Linux είναι ένας κλώνος του λειτουργικού συστήματος Unix, το οποίο εμφανίστηκε το 1970. Τότε, οι υπολογιστές ήταν πολύ διαφορετικοί από ότι είναι τώρα, και ο μέσος πολίτης δεν είχε καν αριθμομηχανή, πόσο μάλλον ένα προσωπικό υπολογιστή (PC). Το Unix προοριζόταν για τη χρήση του στους κεντρικούς υπολογιστές (Mainframes) που χρησιμοποιούνταν κυρίως από τις επιχειρήσεις τηλεπικοινωνιών (όπως η Bell Labs). [8]

Ο Linus Torvalds είναι ο κύριος υπεύθυνος της ανάπτυξης του Linux στις αρχές της δεκαετίας του 90, κατά τη διάρκεια της φοίτησής του στο πανεπιστήμιο του Ελσίνκι στη Φινλανδία. Το όνομά του αποτέλεσε και τη βάση του ονόματος του λειτουργικού συστήματος (Linux από το Linus). Όσο για το πως προφέρεται, σύμφωνα με τη βιβλιογραφία, το όνομα "Linux" πρέπει να ομοιοκαταληκτήσει με τη λέξη "cynics".

Το Linux, λόγω της προέλευσής του, είναι ένα εντολο-οδηγούμενο (command-driven) λειτουργικό σύστημα, κάτι που σημαίνει ότι ο χρήστης θα πρέπει να πληκτρολογεί τις εντολές στην κονσόλα προκειμένου να κάνει αυτό που θέλει. Το γεγονός αυτό έρχεται σε αντίθεση με τα λειτουργικά συστήματα της δεκαετίας του '90, όπως τα Windows και το Macintosh OS. Η εντολο-οδηγούμενη φύση του Unix και του Linux μοιάζει περισσότερο με το DOS παρά με τα νεότερα GUI (Graphical User Interfaces) που συναντώνται στους περισσότερους υπολογιστές σήμερα.

Είναι πολύ σημαντικό για τους νέους χρήστες να γνωρίζουν την κληρονομιά του Unix που το Linux κατέχει. Αυτό θα τους βοηθήσει να κατανοήσουν ότι η σύγκρισή του με άλλα λειτουργικά συστήματα, όπως τα Windows 95 και 98, είναι ατυχής. Το Linux είναι ένα πολύ ισχυρό λειτουργικό σύστημα, και είναι ίσως το πιο κατάλληλο για δικτυακές και διαδικτυακές εφαρμογές (μαζί με το BSD). Πάντως, είναι γεγονός ότι η ισχύς που προσφέρει το Linux στις δικτυακές εφαρμογές δεν χρειάζεται στο μέσο χρήστη και γενικότερα δεν απευθύνεται σε αυτόν, αν και τα τελευταία χρόνια γίνεται μια προσπάθεια προσέγγισής του. [9]

Ένα από τα βασικά χαρακτηριστικά του είναι ότι πρόκειται για ένα ελεύθερο (open-source) λειτουργικό σύστημα. Αυτό σημαίνει ότι δεν κοστίζει τίποτα σε κάποιον που θέλει να το χρησιμοποιήσει, σε αντίθεση με τα Windows ή το Macintosh OS. Όλα τα "open-source" λειτουργικά συστήματα διέπονται από μια νομοθετημένη άδεια που καθιστά αποδεκτή και νόμιμη τη χρήση τους, και καλείται GNU General

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

Public License. Έτσι, όταν αγοράζετε το Linux από ένα κατάστημα υπολογιστών ή ακόμη μέσω διαδικτύου, δεν πληρώνετε για το λογισμικό, αλλά για τις δαπάνες συσκευασίας και διανομής. Βέβαια, αν κάποιος δεν θέλει να το αγοράσει, μπορεί να το ¨κατεβάσει¨ από το διαδίκτυο, εντελώς δωρεάν.

Ένα τελευταίο εισαγωγικό σημείο για το Linux είναι το γεγονός ότι υπάρχουν διαφορετικές εκδόσεις στην αγορά, που καλούνται αλλιώς και διανομές (distributions). Μερικές από τις εταιρίες που διανέμουν εκδόσεις του Linux είναι οι Red Hat, Debian, Caldera Systems, Corel και Slackware. Αυτές τις διανομές παρουσιάζουν μικρές διαφορές, όμως παρ' όλα αυτά ο βασικός πυρήνας είναι Linux. Η επιλογή μιας από αυτές τις διανομές είναι πραγματικά θέμα προσωπικής προτίμησης. Από αυτές, πάντως, η διανομή της Red Hat φαίνεται να είναι η δημοφιλέστερη.

3.2 Λογισμικό οδήγησης (Device Driver) για Linux

Εισαγωγή

Δεδομένου ότι η δημοτικότητα του λειτουργικού συστήματος Linux συνεχίζει να αυξάνεται, το ενδιαφέρον για την συγγραφή λογισμικού οδήγησης συσκευών (device driver) για Linux μεγαλώνει. Το Linux είναι σχεδόν ανεξάρτητο από το hardware, και οι περισσότεροι χρήστες μπορούν να είναι (ευτυχώς) απληροφόρητοι για τα hardware ζητήματα. Αλλά, για κάθε κομμάτι του hardware που υποστηρίζεται από Linux, κάποιος κάπου έχει γράψει έναν οδηγό για να το κάνει να συνεργαστεί με το λειτουργικό σύστημα. Χωρίς device drivers, δεν υπάρχει κανένα λειτουργικό σύστημα.

Οι device drivers παίζουν σημαντικό ρόλο και στον πυρήνα (Kernel) του Linux. Είναι διακριτά "μαύρα κουτιά" που κάνουν ένα συγκεκριμένο κομμάτι του hardware να αποκριθεί σε ένα καθορισμένο με σαφήνεια πρόγραμμα (programming interface), κρύβοντας εντελώς τις λεπτομέρειες για το πώς η συσκευή λειτουργεί.

Οι δραστηριότητες των χρηστών εκτελούνται μέσω ενός συνόλου τυποποιημένων κλήσεων (calls) που είναι ανεξάρτητες από το συγκεκριμένο driver. Ο ρόλος του device driver είναι η αντιστοίχηση (mapping) των κλήσεων αυτών στις διαδικασίες που ενεργούν στο συγκεκριμένο τμήμα του υλικού (hardware).

Ένα πολύ βασικό πλεονέκτημα είναι ότι το λογισμικό οδήγησης κάποιας συσκευής μπορεί να αναπτυχθεί χωριστά από το υπόλοιπο του πυρήνα και "να συνδεθεί" με τον πυρήνα όταν αυτό απαιτείται, διευκολύνοντας κατά πολύ τη συγγραφή.

Υπάρχουν αρκετοί και διάφοροι λόγοι να ενδιαφερθεί κανείς για τη συγγραφή λογισμικού οδήγησης για Linux. Ο ρυθμός με τον οποίο νέο hardware γίνεται διαθέσιμο (και ξεπερασμένο!), από μόνο του εγγυάται ότι αυτοί που γράφουν λογισμικό οδήγησης θα είναι απασχολημένοι για το εγγύς μέλλον. Άλλωστε, οι προμηθευτές hardware, με το να καταστήσουν τα προϊόντα τους συμβατά με το λειτουργικό σύστημα Linux, μπορούν να προσθέσουν τη μεγάλη και αυξανόμενη κοινότητα χρηστών Linux στις πιθανές αγορές τους. Επίσης, η "open-source" φύση του Linux δίνει τη δυνατότητα διάδοσης του πηγαίου κώδικα (source code) του λογισμικού οδήγησης κάποιας συσκευής σε εκατομμύρια χρηστών. [5]

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

3.2.1. Linux Kernel

Σε ένα σύστημα Unix, διάφορες διαδικασίες που εκτελούνται παράλληλα, αναλαμβάνουν ξεχωριστές εργασίες. Η κάθε μια από τις διαδικασίες απαιτεί πόρους από το σύστημα, όπως π.χ. υπολογιστική ισχύ, μνήμη, πρόσβαση σε δίκτυο κτλ. Ο kernel , ο πυρήνας του λειτουργικού συστήματος, αποτελεί ένα μεγάλο τμήμα εκτελέσιμου κώδικα που είναι επιφορτισμένο με τη διαχείριση αυτών των πόρων.
Ο διαχωρισμός μεταξύ των λειτουργιών που αναλαμβάνει ο Kernel δεν είναι πάντα σαφής, ωστόσο μπορούμε να διακρίνουμε το ρόλο του σε μερικές βασικές κατηγορίες, όπως απεικονίζεται στο Σχ.1.16.

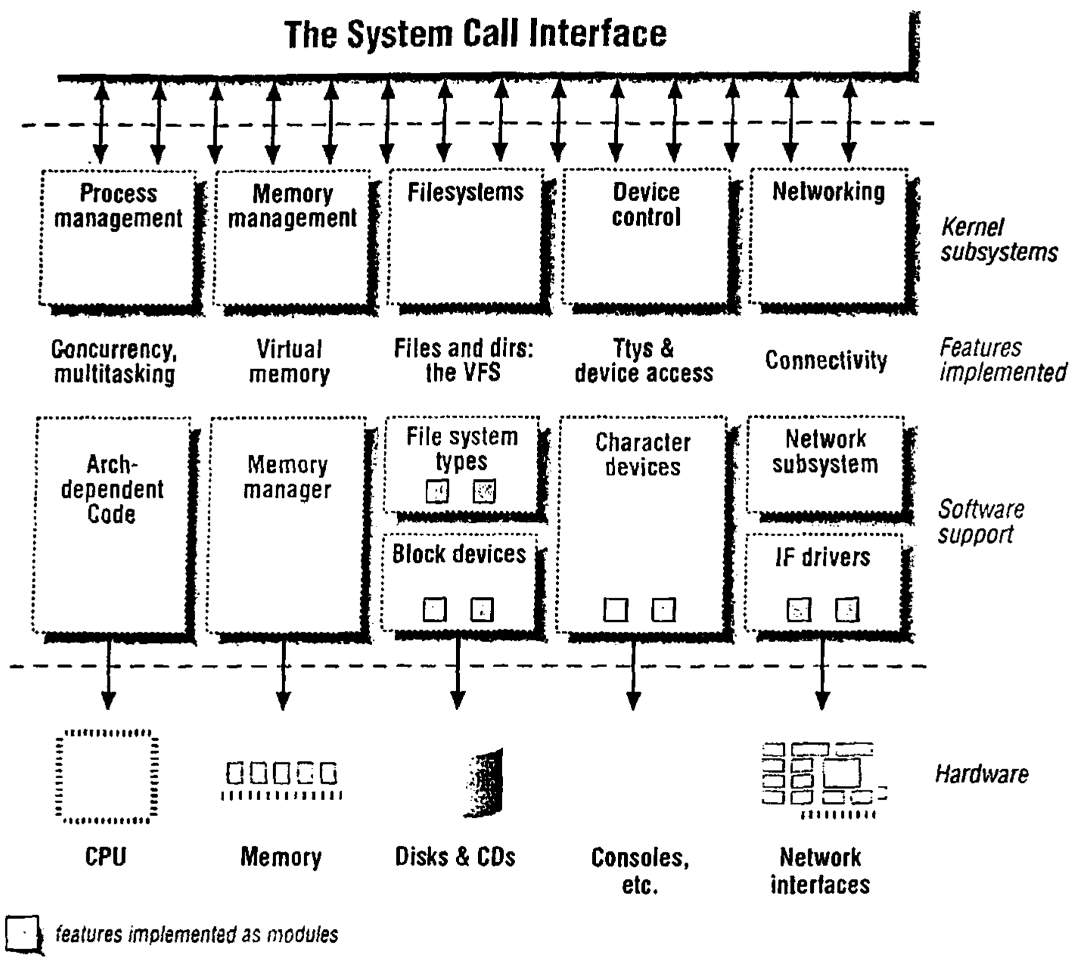
1.Διαχείρηση προγραμμάτων (Process management)

Ο Kernel αναλαμβάνει την εκκίνηση και τον τερματισμό προγραμμάτων (process), καθώς και την επικοινωνία τους με τον έξω κόσμο (I/O). Επίσης, εξασφαλίζει την επικοινωνία μεταξύ των διαδικασιών (processes) που εκτελούνται ταυτόχρονα, επιτυγχάνοντας με αυτόν τον τρόπο την ορθή λειτουργία του συστήματος. Επιπλέον, κατανέμει κατάλληλα την υπολογιστική ισχύ της CPU στις διαφορες διαδικασίες ή προγραμματα [scheduler].

2.Διαχείρηση μνήμης (Memory management)

Η μνήμη αποτελεί έναν άλλο πολύ σημαντικό πόρο, και η μέθοδος προσπέλασης της είναι ιδιαίτερα κρίσιμη για την αποδοση του συστήματος. Ο Kernel, λόγω του πεπερασμένου αριθμού διαθέσιμων πόρων του συστήματος (μνήμης), αναπτύσσει επιπλέον και μια εικονική περιοχή διευθύνσεων μνήμης για την εκτέλεση όλων των processes. Τα διάφορα λειτουργικά τμήματα του kernel, αλληλεπιδρούν με το υποσύστημα διαχείρισης της μνήμης με τη βοήθεια μιας ομάδας συναρτήσεων που είναι ενσωματωμένες στο λειτουργικό σύστημα (`malloc/free`).

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 1.16

3.Συστήματα αρχείων (Filesystems)

Το Unix είναι βασισμένο σε μεγάλο βαθμό στην ιδέα (concept) του συστήματος αρχείων (filesystem) - σχεδόν τα πάντα στο Unix μπορούν να θεωρηθούν ως αρχεία. Ως "filesystem" ορίζεται η μεθόδος οργάνωσης των δεδομένων στο φυσικό μέσο αποθήκευσης (π.χ. στο σκληρό δίσκο). Ο kernel δημιουργεί ένα δομημένο σύστημα αρχείων το οποίο επικάθεται στο κάθε άλλο παρά δομημένο hardware, και η οργάνωση αυτή των αρχείων κυριαρχεί σε όλο το σύστημα.Το Linux υποστηρίζει πολλούς τύπους filesystem, σε αντίθεση με άλλα λειτουργικά συστήματα. Ετσι λοιπόν εκτός του `ext2`, που είναι το βασικό filesystem που χρησιμοποιείται στο Linux, υποστηρίζονται και πολλά άλλα, συμπεριλαμβανομένου και του πολύ γνωστού FAT filesystem, που χρησιμοποιείται στα Windows.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

4.Διαχείρηση συσκευών (Device control)

Σχεδόν οποιαδήποτε λειτουργία του συστήματος αντιστοιχεί σε ένα φυσικό μέσο (device). Αν εξαιρεθεί ο επεξεργαστής, η μνήμη και λίγα ακόμη στοιχεία, ο έλεγχος της λειτουργίας των συσκευών (devices) επιτυγχάνεται με κώδικα, ο οποίος είναι συγκεκριμένος για την κάθε συσκευή και καλείται οδηγός συσκευής (device driver). Ο kernel θα πρέπει να έχει ενσωματωμένους τους οδηγούς για κάθε περιφερειακό του συστήματος, από το σκληρό δίσκο έως το πληκτρολόγιο. Αυτό το ζήτημα αποτελεί και το αντικείμενο της παρούσας εργασίας.

5.Δίκτυο (Networking)

Τα εισερχόμενα πακέτα δεδομένων είναι ασύγχρονα γεγονότα - δηλαδή συμβαίνουν σε τυχαία χρονική στιγμή. Τα πακέτα πρέπει να συλλεχθούν, αναγνωρισθούν, αποκωδικοποιηθούν πριν κάποια διαδικασία (process) αναλάβει τη διαχείρησή τους. Το σύστημα είναι επιφορτισμένο με τη διαδικασία διανομής των πακέτων διαμέσου προγραμμάτων και δικτυακών διεπαφών (interface), καθώς επίσης και με τον έλεγχο της εκτέλεσης των προγραμμάτων ανάλογα με την δικτυακή τους δραστηριότητα. Επιπρόσθετα, ο kernel αναλαμβάνει και τα θέματα της δρομολόγησης και διευθυνσιοδότησης των πακέτων.

Ένα από τα γνωρίσματα του Linux, είναι η ευχέρεια διεύρυνσης των δυνατοτήτων που παρέχονται από τον kernel, οποιαδήποτε στιγμή. Αυτό σημαίνει ότι μπορεί να προστεθεί κάποια νέα λειτουργία, ή ακόμη να αφαιρεθεί, στον kernel την ώρα που το σύστημα τρέχει. Τα τμήματα του κώδικα που μπορούν να ενσωματωθούν στον kernel εν ώρα λειτουργίας ονομάζονται "modules".

Ο kernel του Linux υποστηρίζει πολλές και διάφορες κατηγορίες (Classes) από "modules", συμπεριλαμβανομένων και των device drivers. Το κάθε "module" αποτελείται από κώδικα που δεν είναι πλήρως εκτελέσιμος και ονομάζεται "object code" συνδέεται δυναμικά με τον kernel με το πρόγραμμα "insmod", και αποσυνδέεται από τον kernel με το πρόγραμμα "rmmod".

Στο Σχ.1.16 απεικονίζονται διαφορες κατηγορίες "modules" που αναλαμβάνουν συγκεκριμένες εργασίες. Η διάταξη των "modules" στο Σχ.1.16 καλύπτει τις πιο

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

σημαντικές κατηγορίες, αλλά παρ'όλα αυτά δεν είναι πλήρης, καθώς μέρα με τη μέρα το Linux αναπτύσσεται και συνεχώς προστίθενται στον πυρήνα του νέα "modules". [5]

3.2.2. Κατηγορίες συσκευών και modules

Οι συσκευές του συστήματος (devices) διαχωρίζονται σε τρεις βασικές κατηγορίες (classes), τις οποίες το Unix αντιμετωπίζει με διαφορετικό τρόπο. Οι κατηγορίες αυτές είναι οι εξής:

- *Character Devices*
- *Block devices*
- *Network interfaces*

Το κάθε "module" συνήθως υπάγεται σε μια από αυτές τις κατηγορίες, και έτσι μπορεί να κατηγοριοποιηθεί σαν " char module", "block module" ή " network module". Η κατηγοριοποίηση αυτή δεν είναι απόλυτη - ένας προγραμματιστής μπορεί να αναπτύξει τεράστια "modules" που υλοποιούν διαφορετικούς device drivers σε ένα μεγάλο κομμάτι κώδικα. Παρ' όλα αυτά, ο δόκιμος τρόπος είναι η ανάπτυξη ξεχωριστού "modules" για κάθε μια λειτουργία, έτσι ώστε να είναι δυνατή η τμηματική αναβάθμιση του κώδικα.

1.Character Devices

Ως "character device" θεωρείται αυτή η συσκευή που προσπελαύνεται σαν μια ακολουθία (stream) από bytes, όπως ακριβώς προσπελαύνεται ένα αρχείο. Ο ρόλος του οδηγού μιας τέτοιας συσκευής είναι να υλοποιήσει αυτή τη μέθοδο προσπέλασης. Ένας τέτοιος οδηγός συμπεριλαμβάνει τουλάχιστο τις λειτουργίες (system calls) "open", "close", "read" και "write". Παραδείγματα "character device" είναι η κονσόλα κειμένου (`/dev/console`) και οι σειριακές θύρες (`/dev/ttyS0`) κτλ.

Η μόνη διαφορά στον τρόπο προσπέλασης μιας "character device" σε σχέση με αυτόν ενός αρχείου, είναι ότι στις "character devices" η προσπέλαση των δεδομένων

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

γίνεται μόνο ακολουθιακά, ενώ στα αρχεία είναι δυνατή και η τυχαία προσπέλαση των δεδομένων.

2.Block devices

Οι "block devices", οπως και οι "character devices", προσπελαύνονται σαν αρχεία που βρίσκονται επίσης στο φάκελο `/dev`. Στα περισσότερα συστήματα Unix μια "block device" προσπελαύνεται μόνο σαν μεγάλα τμήματα (blocks) δεδομένων, πολλάπλάσια (σε δυνάμεις του 2) του ενός kilobyte.

Το Linux δίνει τη δυνατότητα στην εκάστοτε εφαρμογή να διαβάζει και να γράφει σε μια "block device", όπως και σε μια "character device", επιτρέποντας την μεταφορά οποιουδήποτε αριθμού bytes κάθε φορά, σύμφωνα όμως με τις παραπάνω προϋποθέσεις.

Η μόνη διαφορά μεταξύ "character device" και "block device" έγκειται στον τρόπο που διαχειρίζονται τα δεδομένα από τον kernel, και συνεπώς και στον τρόπο λειτουργίας του αντίστοιχου οδηγού, και η διαφορά αυτή δεν γίνεται αντιληπτή από το χρήστη. Ένα παράδειγμα "block device" είναι ο σκληρός δίσκος (`/dev/hda`).

3.Network interfaces

Οποιαδήποτε μεταφορά δεδομένων μέσω δικτύου γίνεται με τη βοήθεια ενός interface (διεπαφή), δηλαδή μιας διάταξης (device) που παρέχει αυτή τη δυνατότητα. Το "network interface" είναι επιφορτισμένο με την αποστολή και τη λήψη πακέτων δεδομένων που οδηγούνται από το υποσύστημα δικτύου του kernel, χωρίς να ασχολείται με το πως αντιστοιχούν οι εμπλεκόμενες διαδικασίες με τα πραγματικά πακέτα που αποστέλλονται.

Το Unix παρέχει μεν ένα μοναδικό όνομα σε ένα "network interface" (οπως π.χ. `eth0`), αλλά χωρίς να αντιστοιχεί στο σύστημα αρχείων (όπως π.χ. η σειριακή θύρα `/dev/tty1`) και ο τρόπος επικοινωνίας με τον Kernel είναι τελείως διαφορετικός σε σχέση με αυτόν μεταξύ του kernel και των "character devices" ή "block devices". Έτσι αντί για "read" και "write", ο kernel χρησιμοποιεί συναρτήσεις σχετικές με μετάδοση πακέτων.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Βέβαια, στο Linux υπάρχουν και άλλες κατηγορίες από "modules" όπως π.χ. USB, Firewire κτλ οι οποίες όμως ανήκουν κατά κάποιο τρόπο σε μια από τις προαναφερθείσες γενικότερες κατηγορίες. Η πιο γνωστή κατηγορία συσκευών που παρουσιάζουν μεγάλη ιδιαιτερότητα και δεν ανήκουν σε μία από τις παραπάνω κατηγορίες είναι SCSI devices. Παρ' όλο που όσα περιφερειακά είναι συνδεδεμένα στο δίαυλο SCSI, εμφανίζονται στο φάκελο `/dev`, η εσωτερική οργάνωση του λογισμικού είναι τελείως διαφορετική σε σχέση με αυτήν ενός οδηγού για "character devices" ή "block devices". [5]

Ο οδηγός που αναπτύχθηκε στα πλαίσια αυτής της εργασίας ανήκει στην κατηγορία των "character drivers".

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

3.3 Λογισμικό οδήγησης της κάρτας PCI

Η οδήγηση της κάρτας PCI μπορεί να χωριστεί σε δύο μέρη: Το πρώτο από αυτά σχετίζεται με την οδήγηση του PCI Controller S5920 της εταιρείας AMCC και ενώ το δεύτερο με την οδήγηση του I^2C Controller PCF8584 της Philips. Αυτή η παράγραφος ασχολείται με την οδήγηση του S5920.

3.3.1. Οδήγηση του S5920

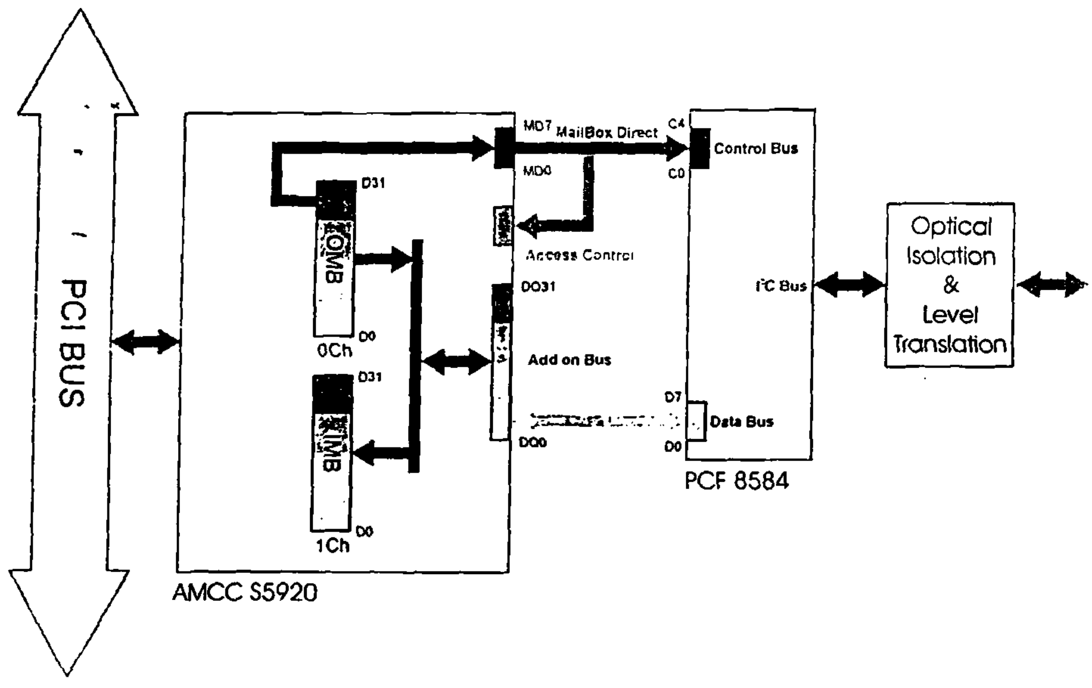
Ο σκοπός της κάρτας που υλοποιήθηκε είναι ο έλεγχος ενός διαύλου I^2C . Όπως προαναφέρθηκε, τον έλεγχο του διαύλου αναλαμβάνει το ολοκληρωμένο PCF8584 της Philips το οποίο απαιτεί για την οδήγηση του έναν αμφίδρομο δίαυλο δεδομένων των 8-bit, καθώς και ένα μονόδρομο δίαυλο ελέγχου των 5-bit. Το ολοκληρωμένο S5920 της AMCC είναι αυτό που παίζει το ρόλο του ενδιάμεσου κρίκου μεταξύ του διαύλου PCI και του διαύλου οδήγησης του PCF8584. Στο διάγραμμα βαθμίδων του Σχ.1.17 απεικονίζεται ο τρόπος με τον οποίο προκύπτουν τα απαραίτητα σήματα για την οδήγηση του PCF8584. Πιο συγκεκριμένα:

Με τις κατάλληλες ρυθμίσεις, επιλέχθηκε η λειτουργία "Mail-box " για το S5920. Κατά την λειτουργία "mailbox", χρησιμοποιούνται κυρίως δύο καταχωρητές των 32-bit, οι IMB (Incoming Mail-Box) και OMB (Outgoing Mail-Box). Στον IMB αποθηκεύονται τα δεδομένα εισόδου, ενώ στον OMB αποθηκεύονται τα δεδομένα εξόδου. Έτσι, όταν ο αμφίδρομος παράλληλος δίαυλος "Add-on" που παρέχει το S5920 λειτουργεί ως είσοδος δεδομένων, τα δεδομένα που διαβάζονται από το δίαυλο αποθηκεύονται στον IMB, ενώ όταν λειτουργεί ως έξοδος τα δεδομένα που είναι αποθηκευμένα στον OMB, οδηγούνται στο δίαυλο εξόδου. Η επιλογή λειτουργίας του διαύλου γίνεται με τα σήματα SEL, WR, RD. Επιπλέον, με κατάλληλη ρύθμιση, το πιο σημαντικό byte του OMB οδηγείται στην έξοδο μέσω ενός διαύλου που ονομάζεται "Mail-Box direct" και είναι ανεξάρτητος των σημάτων SEL, WR, RD . [6]

Με τη μέθοδο Mail-Box επιτυγχάνεται η παρουσία ενός διαύλου ελέγχου 8-bit, πέντε bit εκ των οποίων χρησιμοποιούνται για την οδήγηση του διαύλου ελέγχου του PCF8584 και τα υπόλοιπα τρία για την επιλογή της κατεύθυνσης (είσοδος ή έξοδος) του Add-on bus. Με αυτόν τον τρόπο το S5920 γίνεται κατά κάποιο τρόπο

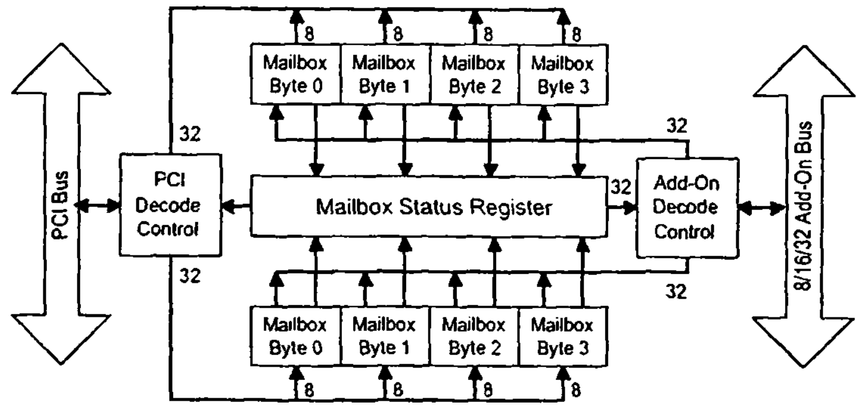
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

αυτοελεγχόμενο, χωρίς τη βοήθεια εξωτερικού κυκλώματος (συνήθως CPLD). Το λιγότερο σημαντικό byte του διαύλου Add-on χρησιμοποιείται ως ο 8-bit δίαυλος δεδομένων του PCF8584.



Σχήμα 1.17 Αναλυτικό διάγραμμα βαθμίδων της κάρτας PCI

Συνεπώς, ο σκοπός του λογισμικού οδήγησης του S5920 είναι η ανάγνωση και η εγγραφή δεδομένων των 32-bit στους καταχωρητές IMB και OMB, δηλαδή η μεταφορά δεδομένων σε περιοχή I/O. Το μέγεθος της περιοχής I/O που απαιτείται για την λειτουργία της κάρτας έχει δηλωθεί στον καταχωρητή Base Address 0, και είναι 128 bytes, εκ των 64Kbytes που διατίθενται συνολικά σε ένα PC. [4]



Σχήμα 1.18 Mail-box Operation

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Το λογισμικό οδήγησης του S5920 είναι βασισμένο στον "generic char driver" που περιλαμβάνεται στο βιβλίο του A.Rubini "*Linux Device Drivers*" και στην προκειμένη περίπτωση χωρίζεται σε πέντε βασικές ενότητες, τις παρακάτω: [5]

- Install Module

  Ο κώδικας αυτής της ενότητας εκτελείται κατά την εγκατάσταση του driver. Σε αυτήν την ενότητα καταχωρείται η συσκευή στον kernel καθώς και η περιοχή I/O που απαιτήθηκε από την κάρτα. Επίσης λαμβάνεται η βασική διεύθυνση της I/O περιοχής. Οι καταχωρητές IMB και OMB βρίσκονται μετατοπισμένοι από τη βασική διαύθυνση κατά 12 (0Ch) και 28 (1Ch) bytes, αντίστοιχα.

- Remove Module

  Ο κώδικας αυτής της ενότητας εκτελείται κατά την απεγκατάσταση του driver. Σε αυτό το τμήμα του κώδικα καταργείται η συσκευή από τον Kernel.

- Open Device

  Ο κώδικας αυτής της ενότητας εκτελείται κατά την εκκίνηση της προσπέλασης της συσκευής από κάποια εφαρμογή.

- Close Device

  Ο κώδικας αυτής της ενότητας εκτελείται κατά την αποδέσμευση της συσκευής μετά τον τερματισμό της προσπέλασής της από κάποια εφαρμογή.

- I/O Control

  Ο κώδικας αυτής της ενότητας εκτελείται κατά την εγγραφή ή ανάγνωση δεδομένων από την περιοχή μνήμης ή I/O και βασίζεται στη συνάρτηση "`ioctl`", που παίζει καταλυτικό ρόλο στην ανταλλαγή δεδομένων μέσω του διαύλου PCI. Η συνάρτηση "`ioctl`" καλεί με τη σειρά της τις κατάλληλες συναρτήσεις χαμηλού επιπέδου (low level), ανάλογα με τον τύπο της αλληλεπίδρασης (εγγραφή ή ανάγνωση) και την περιοχή προσπέλασης (μνήμης ή I/O). Στην προκειμένη περίπτωση γίνεται προσπέλαση μόνο της περιοχής I/O και όχι κάποιας περιοχής μνήμης, καθώς δεν υπάρχουν ανάλογες απαιτήσεις της δεδομένης κάρτας. Οι συναρτήσεις "`inl`" και "`outl`" είναι αυτές που χρησιμοποιούνται για την προσπέλαση της περιοχής I/O, η μεν "`inl`" για την ανάγνωση του IMB και η "`outl`" για την εγγραφή του OMB.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Η λεπτομερής ανάλυση του κώδικα είναι ιδιαίτερα περίπλοκη καθώς απαιτείται μεγάλο υπόβαθρο γνώσεων στη συγγραφή λογισμικού οδήγησης για Linux και δεν κρίνεται απαραίτητη. Ο κώδικας οδήγησης του S5920 περιλαμβάνεται στο παράρτημα (i2cdrv.c & i2cdrv.h). Στο Σχ.1.19 απεικονίζεται η δομή των αρχείων που χρησιμοποιήθηκαν από το μεταγλωττιστή (compiler) για την παραγωγή του λογισμικού οδήγησης. Το λογισμικο οδήγησης είναι σε μορφή *object*, και το αρχείο αυτό φορτώνεται στον kernel με την εντολή "`insmod`" π.χ. `insmod i2cdrv.o`

Για την απεγκατάσταση του λογισμικού οδήγησης χρησιμοποιείται η εντολή "`rmmod`" π.χ. `rmmod i2cdrv.o`

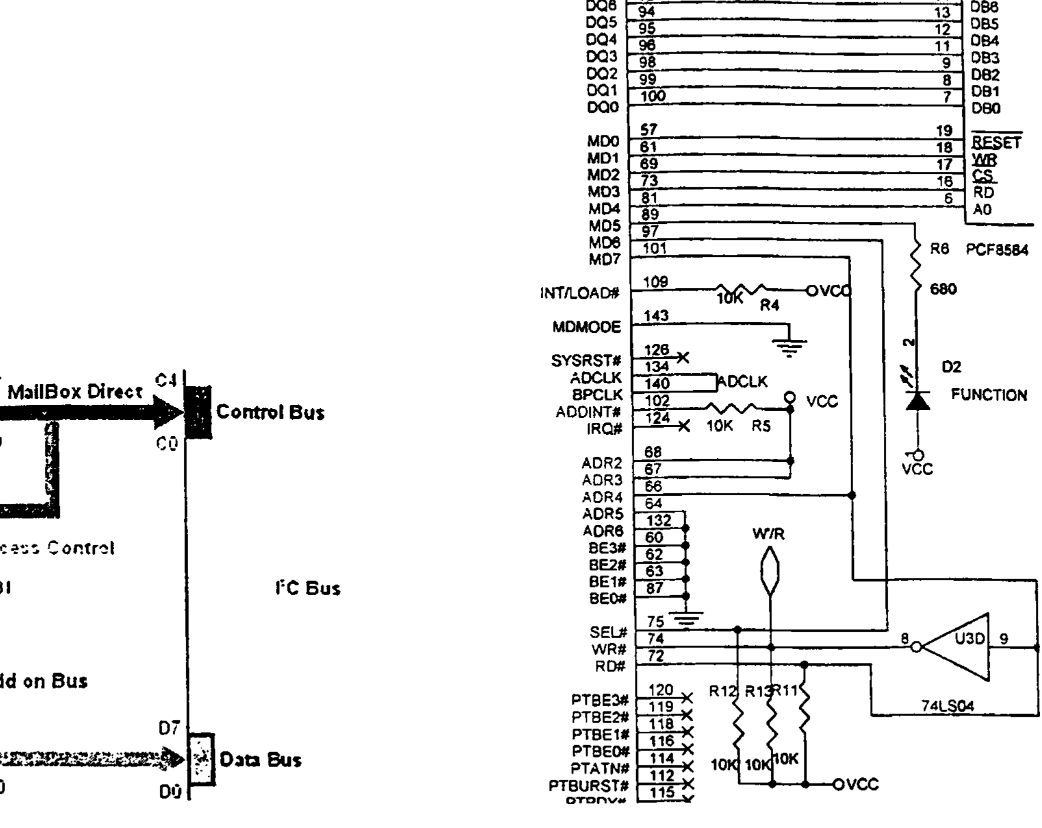
Σχήμα 1.19 Δομή λογισμικού οδήγησης του S5920

Οι παράμετροι της μεταγλώττισης περιλαμβάνονται στο αρχείο "makefile" που παρατίθεται επίσης στο παράρτημα.

3.3.2 Οδήγηση του PCF8584

Σε αυτή την παράγραφο αναλύεται η οδήγηση του PCF8584 από το δίαυλο που προέκυψε με τη βοήθεια του S5920. Στο Σχ.1.20 απεικονίζεται το τμήμα του διαγράμματος βαθμίδων της κάρτας καθώς και το αντίστοιχο τμήμα του σχηματικού κυκλώματος που σχετίζεται με την εν λόγω οδήγηση. Το λογισμικό οδήγησης, όπως είναι φυσικό, είναι στενά συνδεδεμένο με τα χαρακτηριστικά λειτουργίας (data sheet) του PCF8584. [7]

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 1.20 Κύκλωμα οδήγησης του PCF8584

Το λογισμικό οδήγησης (i2c_functions.c) του PCF8584 περιλαμβάνει τις παρακάτω συναρτήσεις:

- `writelong (int offset, long value);`

  Πρόκειται για συνάρτηση χαμηλού επιπέδου (low level function) που συνδέεται δυναμικά με τη συνάρτηση "`ioctl`" του λογισμικού οδήγησης του S5920 και ο σκοπός της είναι η εγγραφή μιας τιμής 32-bit σε έναν καταχωρητή I/O. Το όρισμα "`offset`" δίνει την απόκλιση της διεύθυνσης του καταχωρητή σε σχέση με τη βασική διεύθυνση περιοχής I/O που καθορίστηκε από το BIOS.

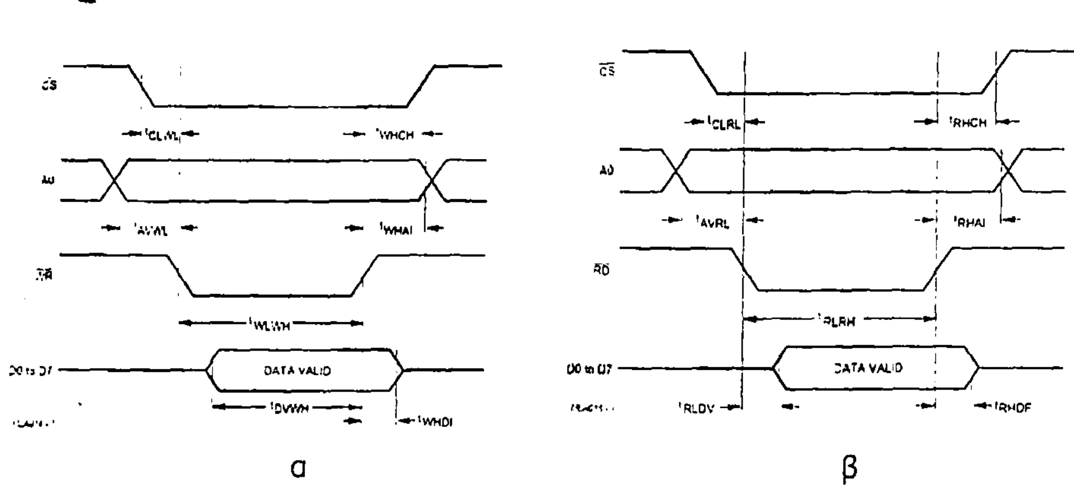
- `readlong(int offset,long* b);`

  Αντίστοιχη συνάρτηση της προηγούμενης, με τη μόνη διαφορά ότι σκοπός της είναι η ανάγνωση της τιμής των 32-bit που είναι αποθηκευμένη σε ένα καταχωρητή I/O.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- `writecycle(int val,int A0);`

Σκοπός της συνάρτησης αυτής είναι η αλληλουχία των σημάτων ελέγχου που παράγονται από το S5920 για την εγγραφή μιας τιμής των 8-bit στο δίαυλο δεδομένων του PCF8584. Βασίζεται στον κύκλο εγγραφής του Σχ.1.21α. [10] Το όρισμα "A0" σχετίζεται με τον καταχωρητή του PCF8584 που προσπελαύνεται.



Σχήμα 1.21 Κύκλοι εγγραφής και ανάγνωσης

- `readcycle(int A0);`

Αντίστοιχη συνάρτηση της προηγούμενης, με τη μόνη διαφορά ότι σκοπός της είναι η ανάγνωση μιας τιμής των 8-bit από το PCF8584 (Σχ.1.21β).

- `pcf8584_init(void);`

Πρόκειται για τη συνάρτηση αρχικοποίησης (initialization) του PCF8584. Η διαδικασία που ακολουθείται απεικονίζεται στο λογικό διάγραμμα του Σχ.1.22 και περιλαμβάνει τα εξής:

- *Αρχική απενεργοποίηση του κυκλώματος ελέγχου του σειριακού διαύλου*
- *Δήλωση συχνότητας του εξωτερικού κυκλώματος χρονισμού (π.χ. 8MHz)*
- *Επιλογή του τύπου οδήγησης των σημάτων ελέγχου (80XX ή 68XXX)*
- *Επιλογή της συχνότητας λειτουργίας του σειριακού διαύλου (π.χ. 90KHz).*
- *Έλεγχος για την κατάσταση του διαύλου (κατειλημμένος ή όχι)*

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

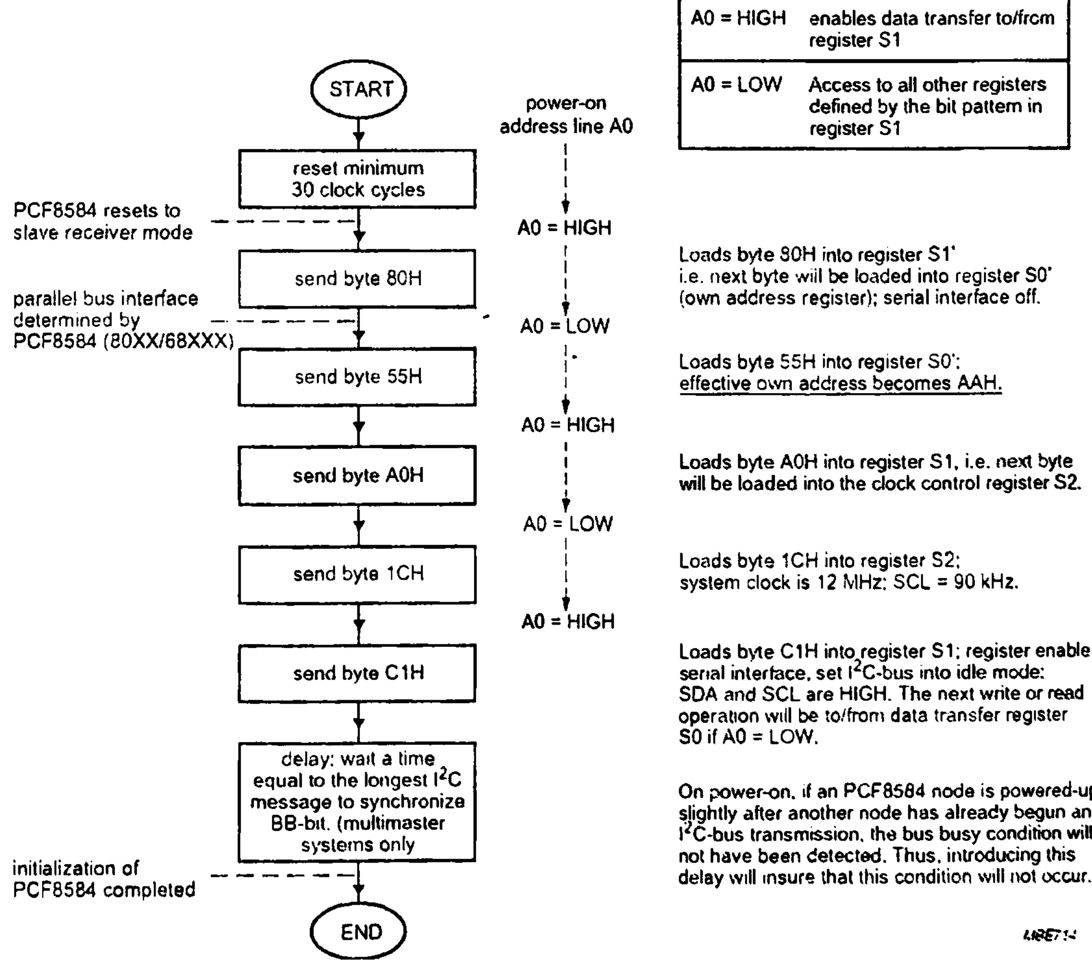
➢ *Ενεργοποίηση του κυκλώματος ελέγχου του σειριακού διαύλου*

- `send_slave_address(int val);`

Η συνάρτηση αποστέλλει το σήμα έναρξης της επικονωνίας μέσω του διαύλου I²C καθώς και τη διεύθυνση του I²C περιφερειακού με το οποίο θέλει να επικοινωνήσει το PCF8584. Η τιμή της διεύθυνσης δηλώνεται στο όρισμα "`val`".

- `get_acknowledgement(void);`

Η συνάρτηση αυτή ελέγχει αν η μετάδοση ενός byte μέσω του διαύλου I²C ολοκληρώθηκε με επιτυχία, με την ανάγνωση του κατάλληλου καταχωρητή του PCF8584



Σχήμα 1.22 Αρχικοποίηση

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- `single_byte_write(int val);`

  Η συνάρτηση αυτή αναλαμβάνει την μετάδοση ενός byte στο PCF8584 με τη βοήθεια των συναρτήσεων "`writelong`" και "`get_acknowledgement`".

- `single_byte_read(void);`

  Η συνάρτηση αυτή αναλαμβάνει την ανάγνωση ενός byte από το PCF8584 με τη βοήθεια της συνάρτησης "`readlong`".

- `check_if_bus_is_busy(void);`

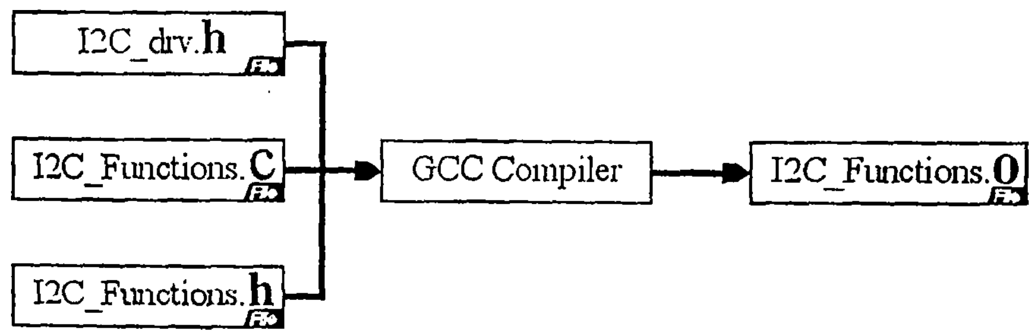
  Η συνάρτηση αυτή ελέγχει την κατάσταση του διαύλου, δηλ αν είναι κατειλημμένος η όχι, με την ανάγνωση του κατάλληλου καταχωρητή του PCF8584. Αυτό μπορεί να συμβαίνει μόνο σε συστήματα multimaster.

- `send_stop(void);`

  Η συνάρτηση αποστέλλει το σήμα τερματισμού της επικοινωνίας και απελευθερώνει το δίαυλο I²C.

Ο κώδικας οδήγησης του PCF8584 περιλαμβάνεται στο παράρτημα (i2c_functions.c & i2c_functions.h). Στο Σχ.1.23 απεικονίζεται η δομή των αρχείων που χρησιμοποιήθηκαν από το μεταγλωττιστή (compiler) για την παραγωγή του λογισμικού οδήγησης.

Οι παράμετροι της μεταγλώττισης περιλαμβάνονται στο αρχείο "makefile" που παρατίθεται επίσης στο παράρτημα.

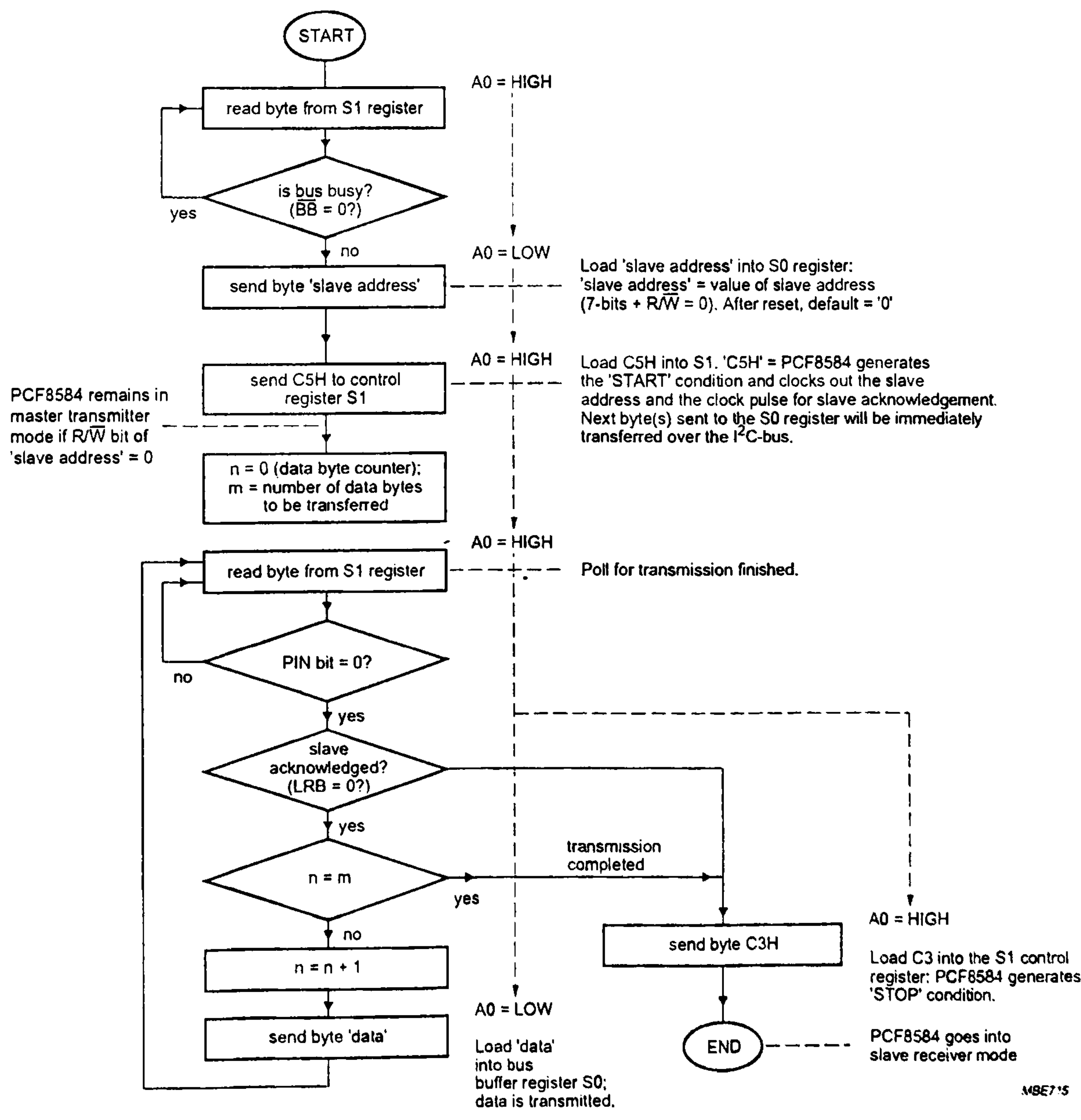


Σχήμα 1.23 Δομή λογισμικού οδήγησης του PCF8584

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

3.4 Εφαρμογές σε γλώσσα προγραμματισμού C

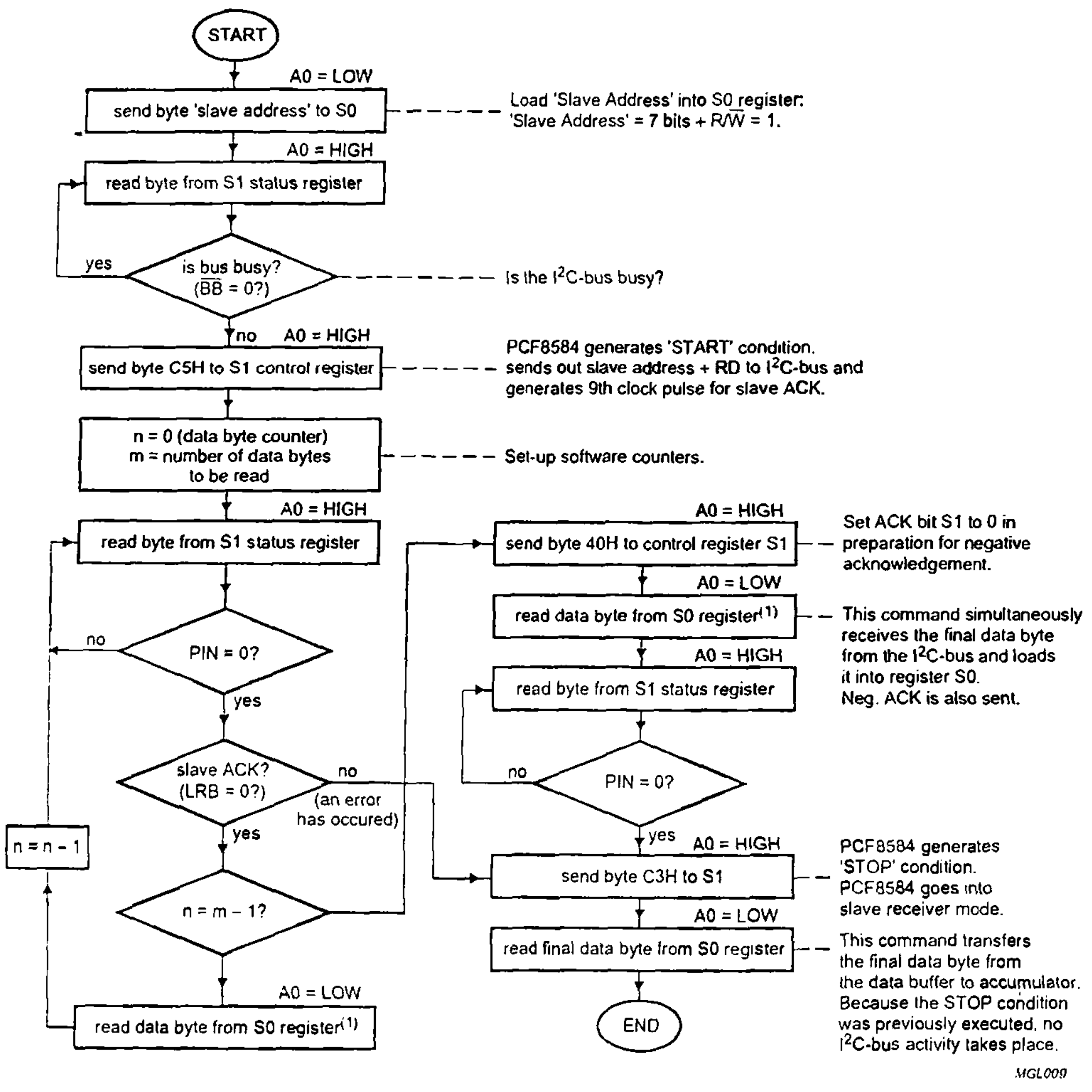
Σε αυτή την παράγραφο δίνονται δύο παραδείγματα εφαρμογών σε γλώσσα προγραμματισμού C για τη χρήση της κάρτας I²C. Το πρώτο από τα παραδείγματα αποτελεί μια απλή εφαρμογή επικοινωνίας της κάρτας με το ολοκληρωμένο κύκλωμα PCF8574. Το PCF8574 είναι ένας I²C expander της εταιρίας Philips, που χρησιμοποιείται συνήθως για τον έλεγχο της ορθής λειτουργίας του διαύλου. [12]



Σχήμα 1.24 Λειτουργία εγγραφής

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Η εφαρμογή αυτή βασίζεται στα διαγράμματα ροής εγγραφής και ανάγνωσης του PCF8584, τα οποία αποτελούν τμήμα των προδιαγραφών του και έχουν παρθεί αυτούσια από το datasheet (Σχ.1.24 & Σχ.1.25). [7]



Σχήμα 1.25 Λειτουργία ανάγνωσης

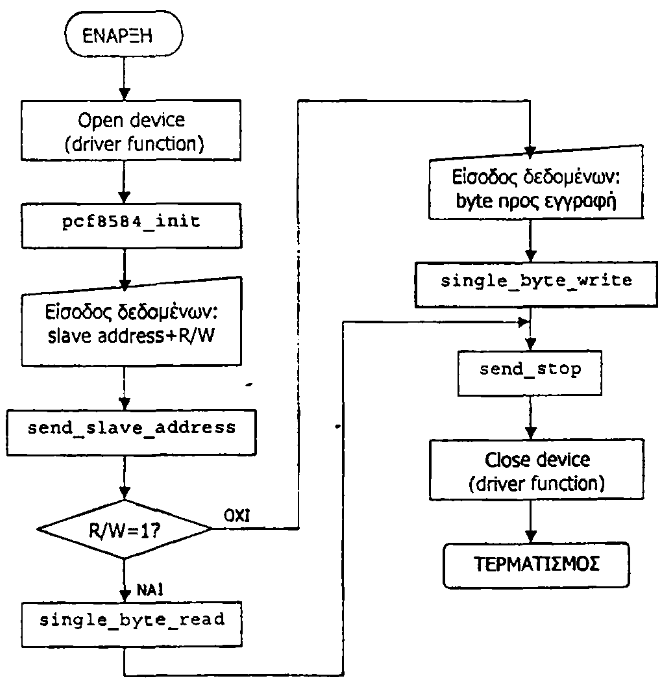
Για την υλοποίηση των προαναφερθέντων διαγραμμάτων ροής χρησιμοποιούνται οι συναρτήσεις της παραγράφου 3.3.2. Στο διάγραμμα ροής της εφαρμογής, το οποίο απεικονίζεται στο Σχ.1.26, εκτελούνται τα εξής βήματα:

Το πρώτο βήμα είναι η εκκίνηση προσπέλασης της κάρτας ή αλλιώς "open device" (βλ. § 3.3.1). Στη συνέχεια ζητείται από τον χρήστη το byte διευθυνσιοδότησης, το

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

οποίο αποστέλλεται με τη συνάρτηση "send_slave_address". Στην περίπτωση που το LSB του byte διευθυνσιοδότησης (R/W) είναι 1, ακολουθεί σειριακή ανάγνωση ενός byte με τη βοήθεια της συνάρτησης "single_byte_read" και η εμφάνιση του αποτελέσματος στην οθόνη. Στην αντίθετη περίπτωση, ζητείται από το χρήστη το byte προς εγγραφή, το οποίο αποστέλλεται με τη συνάρτηση "single_byte_write". Το επόμενο βήμα και για τις δύο περιπτώσεις είναι η αποστολή του σήματος τερματισμού με τη συνάρτηση "send_stop" και τέλος η αποδέσμευση της κάρτας με τη διαδικασία "close device" (βλ. § 3.3.1). Η ακολουθία των σημάτων για εγγραφή και ανάγνωση απεικονίζονται στα Σχ.1.4 και Σχ.1.5, αντίστοιχα.

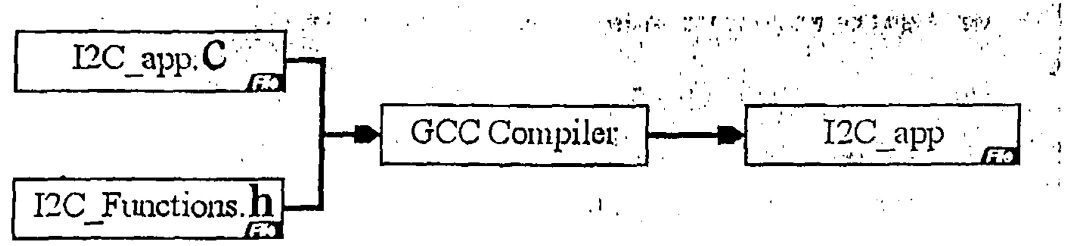
Η εφαρμογή περιλαμβάνεται στο παράρτημα (i2c_app.c).



Σχήμα 1.26 Διάγραμμα ροής εφαρμογής i2c_app

Στο Σχ.1.27 απεικονίζεται η δομή των αρχείων που χρησιμοποιήθηκαν από το μεταγλωττιστή για την παραγωγή του εκτελέσιμου αρχείου i2c_app.

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 1.27 Δομή αρχείων της εφαρμογής i2c_app

Η δεύτερη εφαρμογή σχετίζεται με την οδήγηση του APV6, ενός υβριδικού που χρησιμοποιείται για την επεξεργασία αναλογικών σημάτων από ανιχνευτές πυριτίου. Κατά την επικοινωνία με το APV6 απαιτείται εκτός απο τη διευθυνσιοδότηση του ολοκληρωμένου κυκλώματος και μια δεύτερη διευθυνσιοδότηση, που σχετίζεται με τον καταχωρητή του APV6 που προσπελαύνεται [13]

Συγκεκριμένα:

- Η εγγραφή μιας τιμής σε ένα καταχωρητή του APV6 επιτυγχάνεται με τη σειριακή μετάδοση τριών διαδοχικών αριθμών των 8-bit. Ο πρώτος καθορίζει τη διεύθυνση του APV6, ο δεύτερος τη διεύθυνση του καταχωρητή και ο τρίτος την τιμή που θα εγγραφεί (Σχ.1.28).
- Η ανάγνωση ενός καταχωρητή γίνεται σε δύο φάσεις. Στην πρώτη αποστέλλονται η διεύθυνση του APV6, η διεύθυνση του καταχωρητή, και το σήμα τερματισμού. Στη δεύτερη φάση αποστέλλεται η διεύθυνση του APV6 και στη συνέχεια γίνεται ανάγνωση του καταχωρητή που διευθυνσιοδοτήθηκε στην πρώτη φάση (Σχ.1.29).

Σχήμα 1.28 Εγγραφή ενός καταχωρητή του APV6

Σχήμα 1.29 Ανάγνωση ενός καταχωρητή του APV6

Στην εφαρμογή αυτή χρησιμοποιούνται οι τρεις παρακάτω συναρτήσεις εκ των οποίων οι δύο πρώτες υλοποιούν την εγγραφή και ανάγνωση κάποιου καταχωρητή, χρησιμοποιώντας τις συναρτήσεις της παραγράφου 3.3.2.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

- `write_to_apv6(int addr,int sub_addr,int data);`

  Πρόκειται για τη συνάρτηση εγγραφής καταχωρητή του APV6, κατά την οποία εκτελούνται διαδοχικά οι εξής συναρτήσεις (βλ. Σχ.1.28)

  - `send_slave_address(addr)`
  - `single_byte_write(sub_addr)`
  - `single_byte_write(data)`

- `read_from_apv6(int addr,int sub_addr);`

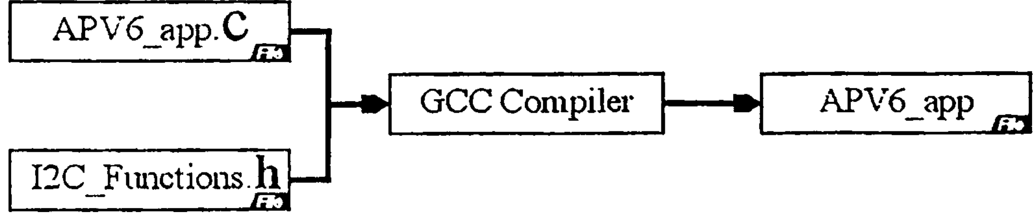
  Πρόκειται για τη συνάρτηση ανάγνωσης καταχωρητή του APV6, κατά την οποία εκτελούνται διαδοχικά οι εξής συναρτήσεις (βλ. Σχ.1.29)

  - `send_slave_address (addr)`
  - `single_byte_write(sub_addr)`
  - `send_stop(void)`
  - `send_slave_address (addr+1)`
  - `single_byte_read(void)`

- `apv6_register_select(char register_name[11]);`

  Πρόκειται για τη συνάρτηση μετάφρασης του ονόματος του εκάστοτε καταχωρητή στην αντίστοιχη 8-bit διεύθυνση [13].

Η εφαρμογή περιλαμβάνεται στο παράρτημα (APV6_app.c). Στο Σχ.1.30 απεικονίζεται η δομή των αρχείων που χρησιμοποιήθηκαν από το μεταγλωττιστή για την παραγωγή του εκτελέσιμου αρχείου APV6_app.



Σχήμα 1.30 Δομή αρχείων της εφαρμογής APV6_app

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

3.5 Εφαρμογές σε LabVIEW 5.1 για Linux

Εκτός των παραδειγμάτων εφαρμογών σε γλώσσα προγραμματισμού C, δίνονται και δύο παραδείγματα χρήσης της κάρτας μέσω του προγράμματος LabVIEW της εταιρείας National Instruments. Το LabVIEW (Laboratory Virtual Instruments Electronic Workbench) είναι ένα περιβάλλον γραφικού προγραμματισμού, ιδιαίτερα δημοφιλές στον χώρο των ηλεκτρικών μετρήσεων που βασίζεται στην γλώσσα γραφικού προγραμματισμού G. Έτσι, αντί για γραμμές κώδικα χρησιμοποιούνται έτοιμες συναρτήσεις υπό μορφή αντικειμένων και η διασύνδεση μεταξύ τους ορίζεται με τη βοήθεια αγωγών, με αποτέλεσμα το πρόγραμμα να μοιάζει περισσότερο σαν ένα διάγραμμα βαθμίδων. Τα προγράμματα καλούνται "εικονικά όργανα" (Virtual Instruments ή VI). [14]

Εφαρμογές

Οι δύο εφαρμογές που δίνονται είναι οι αντίστοιχες των προηγουμένων, με τη διαφορά ότι χρησιμοποιούνται σε γραφικό περιβάλλον. Μέσα από το LabVIEW καλούνται κάποιες συναρτήσεις που έχουν γραφεί σε γλώσσα προγραμματισμού C και έχουν μεταγλωττιστεί με ειδικές παραμέτρους. Με αυτήν τη μεταγλώττιση δημιουργείται η βιβλιοθήκη `lib.so` η οποία ενσωματώνεται στο λειτουργικό σύστημα, δίνοντας τη δυνατότητα κλήσης των παρακάτω συναρτήσεων. [15]

Οι συναρτήσεις είναι οι εξής:

- `i2c_app(int address,int data);`

  Πρόκειται για την εφαρμογή `i2c_app` υπο μορφή συνάρτησης (βλ. §3.4). Με αυτόν τον τρόπο αποφεύγονται οι αλλεπάλληλες κλήσεις συναρτήσεων μέσα στην εφαρμογή.

- `lv_write_apv6(int address,char register_name[11],int data);`

  Πρόκειται ουσιαστικά για τη συνάρτηση `write_apv6`, με κάποιες όμως μικροδιαφορές.

- `lv_read_apv6(int address,char register_name[11]);`

  Πρόκειται ουσιαστικά για τη συνάρτηση `write_apv6`, με κάποιες όμως μικροδιαφορές.

Τα διαγράμματα των προγραμμάτων απεικονίζονται στη συνέχεια.

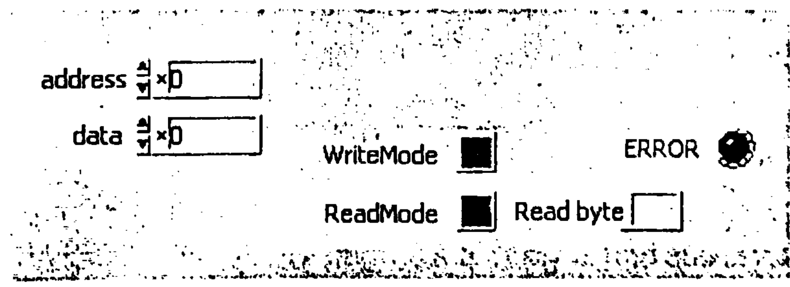
ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

I2C_APP.VI

Στο Σχ.1.31 απεικονίζεται η πρόσοψη (front panel) του εικονικού οργάνου της εφαρμογής οδήγησης του PCF8574 (I2C_APP.VI) στην οποία διακρίνονται τα πεδία εισαγωγής δεδομένων "address" και "data". Στο πεδίο "address" εισάγεται το byte διευθυνσιοδότησης του ολοκληρωμένου, ενώ στο πεδίο "data" , όταν πρόκειται για εγγραφή δεδομένων, εισάγεται το byte προς εγγραφή. Τα δεδομένα εισάγονται σε δεκαεξαδική μορφή, κάτι που δηλώνεται από το δείκτη "x" στα αριστερά του κάθε πεδίου. Στην πρόσοψη του οργάνου διακρίνονται επίσης οι ενδείκτες "WriteMode", "ReadMode" και "Error", καθώς και το πεδίο " Read Byte" στο οποίο εμφανίζεται, κατά την εκτέλεση της διαδικασίας ανάγνωσης, το byte που διαβάστηκε.

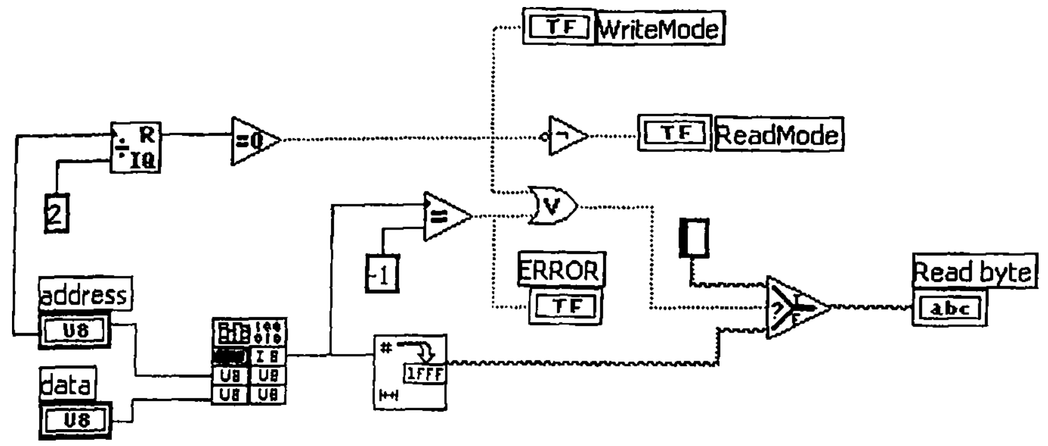
Το διάγραμμα λειτουργίας της εφαρμογής απεικονίζεται στο Σχ.1.32).

Η διαδικασία που εκτελείται είναι η εξής: Το εργαλείο κλήσης συνάρτησης (call library function) δέχεται ως ορίσματα τα δεδομένα των πεδίων "address" και "data", καλεί τη συνάρτηση `i2c_app` και επιστρέφει στην έξοδό του έναν αριθμό. Σε περίπτωση σφάλματος κατά την εκτέλεση της συνάρτησης, ο επιστρέφεται ο αριθμός -1 και ενεργοποιείται ο ενδείκτης σφάλματος "Error". Όταν πρόκειται για διαδικασία ανάγνωσης, δηλ όταν το LSB του byte διευθυνσιοδότησης είναι 1, ενεργοποιείται ο ενδείκτης "ReadMode", και εμφανίζεται ο αριθμός που επέστρεψε η συνάρτηση στο πεδίο "Read Byte". Στην αντίθετη περίπτωση ενεργοποιείται ο ενδείκτης "WriteMode" και στο πεδίο "Read Byte" δεν εμφανίζεται τίποτα.



Σχήμα 1.31 Front Panel του i2c_app.vi

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 1.32 Διάγραμμα του i2c_app.vi

Η ρύθμιση του εργαλείου "call library function", σε αυτήν την εφαρμογή, απεικονίζεται στο Σχ.1.33. Στο πεδίο "library name or path" δηλωνεται το όνομα της βιβλιοθήκης στην οποία περιλαμβάνεται η συνάρτηση που καλείται. Στο πεδίο "function name" δηλώνεται το όνομα της συνάρτησης ενώ στο πεδίο "calling conversions" η γλώσσα προγραμματισμού που χρησιμοποιήθηκε. Στο πεδίο "function prototype" εμφανίζονται οι παράμετροι της συνάρτησης, οι οποίες δηλώνονται με τη βοήθεια των πεδίων "parameter", "type" και "data type".

Library Name or Path: /usr/lib/lib.so — Browse...

Function Name: i2c_app — Run in UI Thread

Calling Conventions: C

Parameter: return type

Type: Numeric

Data Type: Unsigned 8-bit Integer

Add a Parameter Before

Add a Parameter After

Delete this Parameter

Function Prototype:

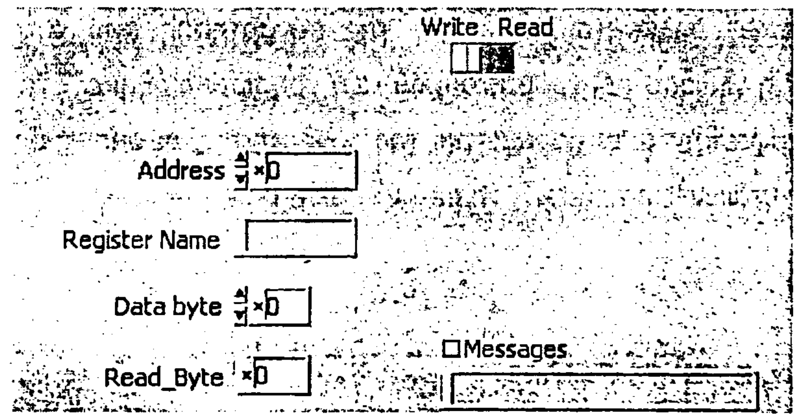
unsigned char i2c_app(unsigned char address, unsigned char data);

Σχήμα 1.33 Εργαλείο κλήσης συνάρτησης βιβλιοθήκης "i2c_app"

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

APV6.VI

Στο Σχ.1.34 απεικονίζεται η πρόσοψη (front panel) του εικονικού οργάνου της εφαρμογής οδήγησης του APV6 (APV6.VI) στην οποία διακρίνονται τα πεδία εισαγωγής δεδομένων "address", "Register Name" και "data". Στο πεδίο "address" εισάγεται η 7-bit διεύθυνση του APV6 (χωρίς το bit R/W), ενώ στο πεδίο "data", όταν πρόκειται για εγγραφή δεδομένων, εισάγεται το byte προς εγγραφή. Στο πεδίο "register Name" εισάγεται το όνομα του καταχωρητή που προσπελαύνεται από την εφαρμογή Τα αριθμητικά δεδομένα εισάγονται σε δεκαεξαδική μορφή, κάτι που δηλώνεται από το δείκτη "x" στα αριστερά του κάθε πεδίου. Στην πρόσοψη του οργάνου διακρίνεται επίσης ο επιλογέας "Write Read", από όπου επιλέγεται ο τύπος της επικοινωνίας με το APV6, το πεδίο "Read_byte", όπου εμφανίζεται το byte που διαβάστηκε κατά την εκτέλεση της διαδικασίας ανάγνωσης, και το πεδίο "Messages", όπου εμφανίζονται τα μηνύματα σφάλματος.

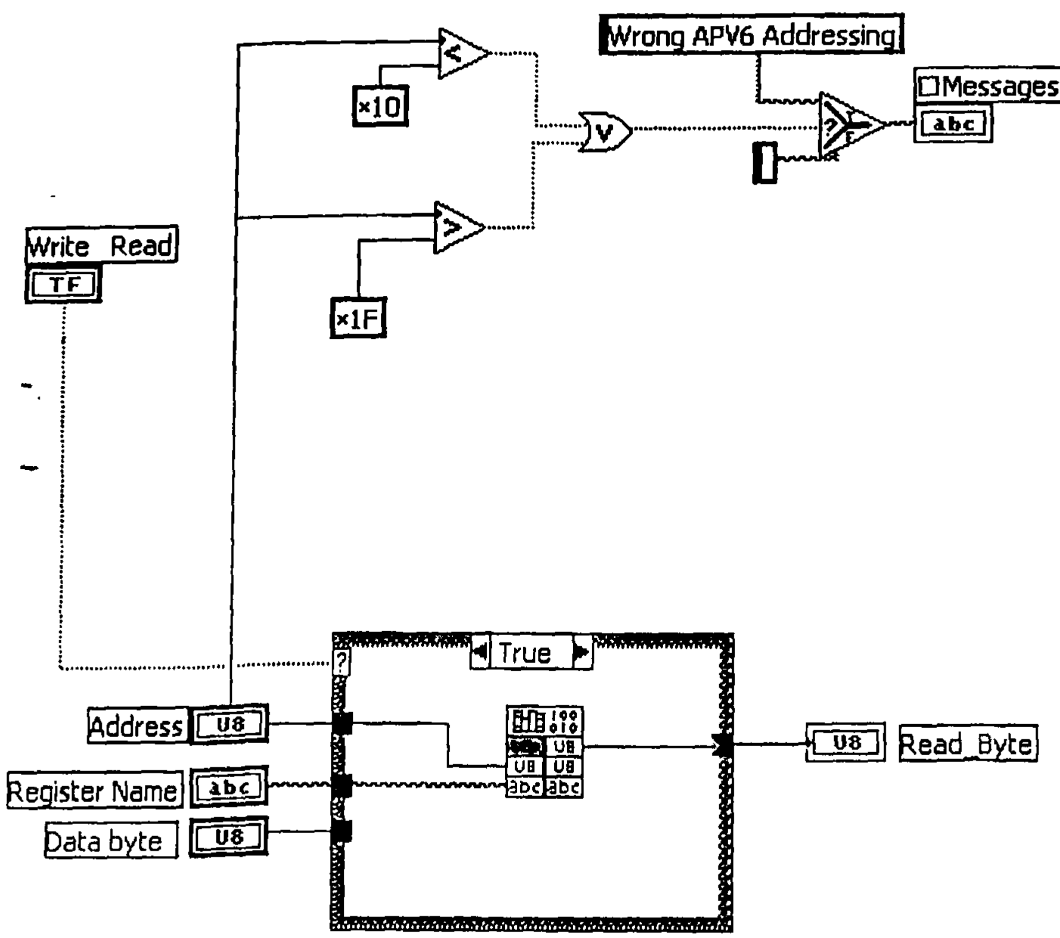


Σχήμα 1.34 Front Panel του apv6.vi

Το διάγραμμα για τη διαδικασία ανάγνωσης απεικονίζεται στο Σχ.1.35. Η διαδικασία ανάγνωσης εκτελείται ως εξής: Το εργαλείο κλήσης συνάρτησης (call library function) δέχεται ως ορίσματα τα δεδομένα των πεδίων "address", "register name", καλεί τη συνάρτηση `lv_read_apv6` και επιστρέφει στην έξοδό του το byte που διαβάστηκε, το οποίο εμφανίζεται στο πεδίο "Read_Byte". Σε περίπτωση λανθασμένης διευθυνσιοδότησης, εμφανίζεται στο πεδίο "Messages" το μήνυμα "wrong APV6 addressing".

Η ρύθμιση του εργαλείου "call library function", σε αυτήν την περίπτωση απεικονίζεται στο Σχ.1.36.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 1.35 Διάγραμμα του apv6.vi (Διαδικασία ανάγνωσης)

Library Name or Path: /usr/lib/lib.so — Browse...

Function Name: lv_read_apv6 — Run in UI Thread

Calling Conventions: C

Parameter: return type

Type: Numeric

Data Type: Unsigned 8-bit Integer

Add a Parameter Before

Add a Parameter After

Delete this Parameter

Function Prototype:

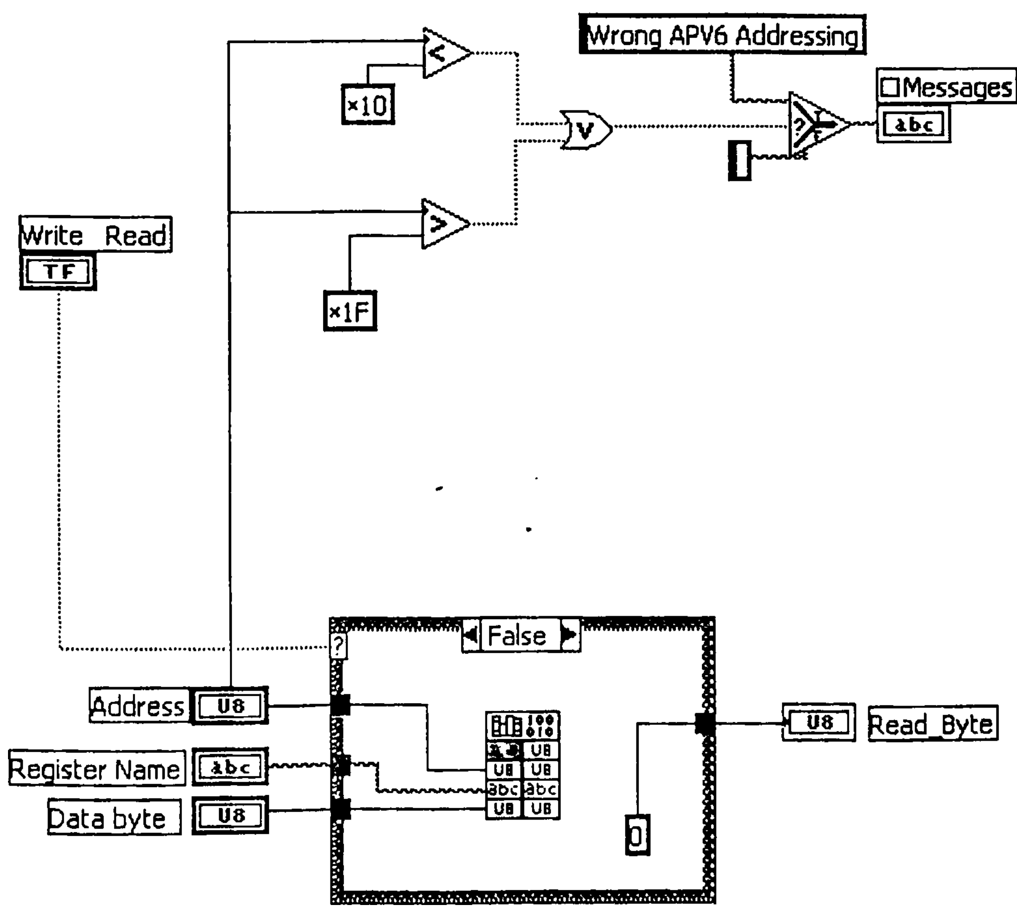
unsigned char lv_read_apv6(unsigned char address, CStr register_name);

Σχήμα 1.36 Εργαλείο κλήσης συνάρτησης βιβλιοθήκης "lv_read_apv6"

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Το διάγραμμα για τη διαδικασία εγγραφής απεικονίζεται στο Σχ.1.37. Η διαδικασία εγγραφής εκτελείται ως εξής: Το εργαλείο κλήσης συνάρτησης (call library function) δέχεται ως ορίσματα τα δεδομένα των πεδίων "address", "register name" και "data", και καλεί τη συνάρτηση `lv_write_apv6`. Σε περίπτωση λανθασμένης διευθυνσιοδότησης, εμφανίζεται στο πεδίο "Messages" το μήνυμα "wrong APV6 addressing".

Η ρύθμιση του εργαλείου "call library function", σε αυτήν την περίπτωση, απεικονίζεται στο Σχ.1.38.



Σχήμα 1.37 Διάγραμμα του apv6.vi (Διαδικασία εγγραφής)

Library Name or Path /usr/lib/lib.so
Browse...

Function Name lv_write_apv6
Run in UI Thread

Calling Conventions C

Parameter return type
Type Numeric
Data Type Unsigned 8-bit Integer

Add a Parameter Before
Add a Parameter After
Delete this Parameter

Function Prototype:
unsigned char lv_write_apv6(unsigned char address, CStr register_name, unsigned char data);

Σχήμα 1.38 Εργαλείο κλήσης συνάρτησης βιβλιοθήκης "lv_write_apv6"

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

4. Συμπεράσματα

Η συγγραφή του λογισμικού οδήγησης της κάρτας PCI αποτέλεσε το επιστέγασμα της πρώτης προσπάθειας ανάπτυξης ενός περιφερειακού PCI από το Εργαστήριο Φυσικής Υψηλών Ενεργειών του Πανεπιστημίου Ιωαννίνων. Η κατανόηση σε μεγάλο βαθμό του περίπλοκου πρωτοκόλλου επικοινωνίας PCI καθώς και η εμβάθυνση στα άδυτα της αρχιτεκτονικής του πυρήνα του λειτουργικού συστήματος Linux αποτέλεσαν το μεγαλύτερο κέρδος από αυτήν την προσπάθεια. Θα ήταν ευχής έργο η εργασία αυτή να μπορέσει να αποτελέσει βοήθημα σε όποιες μελλοντικές προσπάθειες ανάπτυξης ανάλογου λογισμικού.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Ανάπτυξη κάρτας PCI διαύλου I²C για τον έλεγχο μικρολωριδιακών αισθηρήτων πυριτίου – Γ.Σιδηρόπουλος
Πανεπιστήμιο Ιωαννίνων 2002

[2] The I²C-bus specification version 2.1
Philips Semiconductors 2001

[3] PCI System Architecture 4th edition - T.Shanley & D. Anderson
Addison Wesley 1999

[4] PCI Local bus specification revision 2.2
PCI Special Interest Group 1998

[5] Linux Device Drivers 2nd edition - A.Rubini
O'Reilly 2001

[6] S5920 PCI Target/Slave Interface Data Book 1st edition
Applied Micro Circuits Corp. 1997

[7] PCF8584 I²C bus Controller Product Specification
Philips Semiconductors 1997

[8] Unix - P.Abrahams & B.Larson
Addison Wesley 1992

[9] The Official Red Hat Linux 6.1 Reference Guide
Red Hat, Inc 1999

[10] AN425 Interfacing the PCF8584 I²C-bus controller to 80C51 family microcontrollers - Philips Semiconductors 1994

[11] C++ for Linux 1st edition - J.Liberty & D.Horvath
Sams 2000

[12] PCF8574A Remote 8-bit I/O expander for I²C-bus
Texas Instruments, Inc. 2001

[13] APV6 User Manual 2nd edition – M.French
Rutherford Appleton Laboratory 1997

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

[14] LabVIEW 5.1 User Manual
National Instruments Corp. 1998

[15] LabVIEW 5.1 Function & VI Reference Manual
National Instruments Corp. 1998

ΠΑΡΑΡΤΗΜΑ

1. i2cdrv.c

```
/***************** i2cdrv.c
*******************************************/

#define MODULE

/* retrieve the CONFIG_* macros */
#include <linux/autoconf.h>

#if defined(CONFIG_MODVERSIONS) && !defined(MODVERSIONS)
#define MODVERSIONS
#endif

#ifdef MODVERSIONS
#include <linux/modversions.h>
#endif

#include <linux/version.h>
#include <linux/module.h>
#include <linux/types.h>
#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/major.h>
#include <linux/mm.h>
#include <linux/timer.h>
#include <linux/ioport.h>
#include <linux/kernel.h>
#include <linux/malloc.h>
#include <linux/string.h>
#include <linux/signal.h>
#include <linux/config.h>       /* for CONFIG_PCI */
#include <linux/delay.h>
#include <asm/io.h>
#include <asm/byteorder.h>
#include <asm/segment.h>
#include <asm/uaccess.h>
#include <asm/system.h>
#include "i2cdrv.h"

#include <linux/signal.h>
#undef DMA_RUNNING

#ifdef DMA_RUNNING
#include "allocator.h"
#endif

#ifdef CONFIG_PCI
#include <linux/pci.h>
#endif
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/****** Prototype of public and private functions. ********/

static int i2cdrv_open(struct inode *iNode, struct file *filePtr);
static int i2cdrv_close(struct inode *iNode, struct file *filePtr);
static int i2cdrv_ioctl(struct inode *iNode, struct file *filePtr,
unsigned int cmd, LONG arg);
static int i2cdrv_mmap(struct file *filePtr, struct vm_area_struct
*vma);
static ssize_t i2cdrv_read(struct file *filePtr, char *buf,size_t
count, loff_t *off);
int init_module(void);
void cleanup_module(void) ;


/********** Global data. ********************/

typedef struct _i2cpmc5920 {

  unsigned long Base0;
  unsigned long Local0;
  unsigned long Local1;
  unsigned int open;
  unsigned char irq;
  struct task_struct* taskid;

  /* DMA */
  unsigned long PhysDmaAddress;
  unsigned long VirtDmaAddress;
  unsigned long BusDmaAddress;
} i2cpmc5920;

typedef struct _trigusr {
  int TriggerOn;
  struct task_struct* taskid;
} trigusr;
#define TEMPOIO 1000
#define HEPLABI2CDRV 2

static int NPmc=0;
static i2cpmc5920 Pmc[HEPLABI2CDRV];
static int MajorNumber = DEFAULT_MAJOR_DEV;

#define MAXI2CUSR 16
static trigusr User[MAXI2CUSR];

/*************file operations **********************/

static struct file_operations i2cdrv_fops = {
  NULL,             /* seek    */
  i2cdrv_read,     /* read    */
  NULL,            /* write   */
  NULL,              /* readdir */
  NULL,              /* select  */
  i2cdrv_ioctl,      /* ioctl   */
  i2cdrv_mmap,     /* mmap    */
  i2cdrv_open,
  NULL,
  i2cdrv_close,
  NULL               /* fsync   */ };
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/********************** insmod *********************/

int init_module(void)
{
  int pci_index,i;
  unsigned char pci_bus, pci_device_fn;
  unsigned int pci_ioaddr;
  WORD pci_status;

  int err;
  i2cpmc5920* p;

  struct pci_dev *dev = NULL;

#ifndef CONFIG_PCI
  /* this should not happen. */
  printk("%s pci not configured with kernel!\n", ADAPTER_ID);
  return -ENODEV;
#endif


  /* Register as a device with kernel.  */

  if ((err = register_chrdev(MajorNumber, "i2cdrv", &i2cdrv_fops))) {
    printk("%s: Failure to load module. error %d\n", ADAPTER_ID, -
err);
    return err;
  }


/* Register the region of IO space with kernel and get base address*/

  if(pci_present()) {
    for ( pci_index = 0; pci_index < HEPLABI2CDRV; pci_index++ ) {

      if( (dev = pci_find_device( I2CMCTRL_VENDOR_ID,
                                  I2CMCTRL_DEVICE_ID,
                                  dev)) == 0 ) break;

      pci_bus = dev->bus->number;
      pci_device_fn = dev->devfn;
      p =&Pmc[NPmc];

      /* Get PCI_BASE_ADDRESS_0 */
      pcibios_read_config_dword( pci_bus, pci_device_fn,
                                 PCI_BASE_ADDRESS_0, &pci_ioaddr );

      /* Strip the I/O address out of the returned value */
      pci_ioaddr &= PCI_BASE_ADDRESS_IO_MASK;
      printk("HEPLAB I2C CONTROLLER found at Base address 0
%x\n",pci_ioaddr);
      p->Base0 = pci_ioaddr;
      /* Now configure the PCI */

      pcibios_read_config_word( pci_bus, pci_device_fn,
                                PCI_STATUS, &pci_status );
      printk("HEPLAB I2C CONTROLLER :  PCI status %x \n",pci_status);
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
      printk("HEPLAB I2C CONTROLLER: PCI Interface is configured\n");

      NPmc++;
    }
  }
  for (i=0;i<MAXI2CUSR;i++)
    {
      User[i].taskid=0;
      User[i].TriggerOn=1;
    }
  return 0;
}


/********** rmmod *****************************/

void cleanup_module(void)
{
#ifdef DMA_RUNNING
  allocator_cleanup();
#endif
  if (MOD_IN_USE) {
    printk("%s: device busy, remove delayed.\n", ADAPTER_ID);
    return;
  }


  if (unregister_chrdev(MajorNumber, "i2cdrv") != 0) {
    printk("%s: cleanup_module failed.\n", ADAPTER_ID);
  } else {

#ifdef DEBUG
    printk("%s: module removed.\n", ADAPTER_ID);
#endif

  }
}

#ifdef __cplusplus
}
#endif


/****** open() service handler ********************/

static int i2cdrv_open(struct inode *iNode, struct file *filePtr)
{
int minor = MINOR(iNode->i_rdev);


/*
   check if device is already open: only one process may read from a
   port at a time.  There is still the possibility of two processes
   reading from two different channels messing things up. However,
   the overhead to check for this may not be worth it.
*/

if ( minor >= 0 && minor < HEPLABI2CDRV ) {
if ( Pmc[minor].open == 1 ) {
```


ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
return -1;

}
if (minor>NPmc-1) return -1;

//MOD_INC_USE_COUNT;

Pmc[minor].taskid = current;

Pmc[minor].open = 1;
}
 if (minor>=16 && minor <31)
   {
    User[minor-16].taskid = current;
   }
 return 0;
}

static ssize_t i2cdrv_read(struct file *filePtr, char *buf,size_t
count, loff_t *off)
{
  int minor;
  /* Get the board index */
  struct inode *iNode = filePtr->f_dentry->d_inode;
  minor = MINOR(iNode->i_rdev);
  /*
    printk("%d count asked \n",count);
  */
  if ( copy_to_user(buf, (void*) Pmc[minor].VirtDmaAddress, count))
    return -1;
  return (count);

}

/********  close() service handler  *********************/

static int i2cdrv_close(struct inode *iNode, struct file *filePtr)
{
  int minor = MINOR(iNode->i_rdev);


  //  MOD_DEC_USE_COUNT;

  if ( minor >= 0 && minor < HEPLABI2CDRV )
      {
    Pmc[minor].open=0;
    iounmap((void*) Pmc[minor].Local0);
    iounmap((void*) Pmc[minor].Local1);
      }

 if (minor>=16 && minor <31)
   {
     User[minor-16].taskid =0;
     User[minor-16].TriggerOn =1;
   }

  return 0;
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/****  ioctl() service handler   ***********************/

/* Note:
    Remember that FIOCLEX, FIONCLEX, FIONBIO, and FIOASYN are
reserved ioctl cmd numbers
*/

static int i2cdrv_ioctl(struct inode *iNode, struct file *filePtr,
unsigned int cmd, LONG arg)
{
  int minor = MINOR(iNode->i_rdev);
  LONG offset,rdval,argb[3];
  WORD rdvas;
  BYTE rdvac;
  int err = 0;
  int size = _IOC_SIZE(cmd);       /* the size bitfield in cmd */

  /*
   * extract the type and number bitfields, and don't decode
   * wrong cmds;  return EINVAL before verify_area()
   */

  if (_IOC_TYPE(cmd) != IOCTL_MAGIC) return -EINVAL;
  if (_IOC_NR(cmd) > IOCTL_MAXNB)  return -EINVAL;

  if (_IOC_DIR(cmd) & _IOC_READ) {
    err = verify_area(VERIFY_WRITE, (void *)arg, size);
  } else if (_IOC_DIR(cmd) & _IOC_WRITE) {
    err = verify_area(VERIFY_READ, (void *)arg, size);
  }
  if (err) return err;
  /*  if ( minor >= 0 && minor < HEPLABI2CDRV ) { */
  if ( minor >= 0 && minor < 32 ) {
    switch (cmd) {
    case GET_BASE0_ADDRESS:
      put_user(Pmc[minor].Base0,(long*) arg);
      break;

     case READ_BASE0_LONG:
      copy_from_user(&offset,(LONG*) arg,4);
      rdval = inl(Pmc[minor].Base0+offset);
      copy_to_user((LONG*) arg,&rdval,4);
      break;

   case WRITE_BASE0_LONG:
      err= copy_from_user(argb,(LONG*) arg,8);
      offset =argb[0];
      rdval = argb[2];
      outl(rdval,Pmc[minor].Base0+offset);
      break;

    case REENABLE_TRIGGER:
      if (minor>=16)
        {
          int ich,trg=1;
          User[minor-16].TriggerOn=1;

          for (ich=0;ich<MAXI2CUSR;ich++)
            trg *= User[ich].TriggerOn;
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
          /*        printk("i2c Driver Trigger reenable %d
\n",trg);*/
          if (trg>0)
            {
              /* Reenable all trigger */
              for (ich=0;ich<NPmc;ich++)
            writel(1,Pmc[ich].Local1+0x2c);
            //                *(long*)(Pmc[ich].Local1+0x2c)=1;
            }
        }
        break;
      default:
        return(-EINVAL);
        break;
      } /* end switch */
    }  /* end if */

  return 0;
}

static int i2cdrv_mmap(struct file *filePtr, struct vm_area_struct
*vma)
{
  /*
    struct inode *iNode = filePtr->f_dentry->d_inode;
  */
  LONG off = vma->vm_offset;
  LONG len = vma->vm_end - vma->vm_start;
  printk("Remapping %lx %lx %lx \n", vma->vm_start,vma->vm_end,
(long) vma->vm_offset);
  if (remap_page_range(vma->vm_start, off,len, vma->vm_page_prot))
    return -EAGAIN;
  /*
    vma->vm_inode = inode;
    inode->i_count++;
  */
  return 0;

}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2. i2cdrv.h

```
/*
 *              i2cdrv.h
 */

#define I2CMCTRL_VENDOR_ID 0x10e8
#define I2CMCTRL_DEVICE_ID 0x5920

#define REG_LOCAL_PLXID  0x70
#define REG_LOCAL_BASE1  0x00
#define REG_LOCAL_BASE3  0xF0

#include <linux/ioctl.h>
#define IOCTL_MAGIC 'w'

/*  ioctl() values */

#define GET_BASE0_ADDRESS     _IOR(IOCTL_MAGIC, 1,4)
#define GET_BASE1_ADDRESS     _IOR(IOCTL_MAGIC, 2,4)
#define GET_BASE3_ADDRESS     _IOR(IOCTL_MAGIC, 3,4)
#define GET_DMABUS_ADDRESS     _IOR(IOCTL_MAGIC, 4,4)
#define GET_DMAVIRT_ADDRESS     _IOR(IOCTL_MAGIC, 5,4)
#define GET_DMAPHYS_ADDRESS     _IOR(IOCTL_MAGIC, 6,4)

#define READ_BASE0_BYTE         _IOR(IOCTL_MAGIC,  7,4)
#define READ_BASE0_SHORT        _IOR(IOCTL_MAGIC,  8,4)
#define READ_BASE0_LONG         _IOR(IOCTL_MAGIC,  9,4)
#define READ_BASE1_BYTE         _IOR(IOCTL_MAGIC,  0,4)
#define READ_BASE1_SHORT        _IOR(IOCTL_MAGIC, 11,4)
#define READ_BASE1_LONG         _IOR(IOCTL_MAGIC, 12,4)
#define READ_BASE3_BYTE         _IOR(IOCTL_MAGIC, 13,4)
#define READ_BASE3_SHORT        _IOR(IOCTL_MAGIC, 14,4)
#define READ_BASE3_LONG         _IOR(IOCTL_MAGIC, 15,4)

#define WRITE_BASE0_BYTE         _IO(IOCTL_MAGIC, 16)
#define WRITE_BASE0_SHORT         _IO(IOCTL_MAGIC, 17)
#define WRITE_BASE0_LONG         _IO(IOCTL_MAGIC, 18)
#define WRITE_BASE1_BYTE         _IO(IOCTL_MAGIC, 19)
#define WRITE_BASE1_SHORT         _IO(IOCTL_MAGIC, 20)
#define WRITE_BASE1_LONG         _IO(IOCTL_MAGIC, 21)
#define WRITE_BASE3_BYTE         _IO(IOCTL_MAGIC, 22)
#define WRITE_BASE3_SHORT         _IO(IOCTL_MAGIC, 23)
#define WRITE_BASE3_LONG         _IO(IOCTL_MAGIC, 24)
#define ENABLE_INTERRUPT         _IO(IOCTL_MAGIC, 25)
#define DISABLE_INTERRUPT         _IO(IOCTL_MAGIC, 26)
#define REENABLE_TRIGGER          _IO(IOCTL_MAGIC, 27)
#define TOGGLE_TRIGGER            _IOR(IOCTL_MAGIC, 28,4)
#define IOCTL_MAXNB 28          /* maxinum ordinal number */


#define BYTE unsigned char
#define WORD unsigned short
#define LONG unsigned long
```

3. i2c_functions.c

```
#include "i2cdrv.h"
#include "i2c_functions.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>


/************************ full functions
****************************/

/*********** full i2c_application for labview **************/

int i2c_app(int address,int data)
{
int typed_address,typed_byte,RW;
int ret=0;

  fd= open("/dev/i2cdrv00",6);
  if (fd==0)
    { printf ("INFO=>An error occured while trying to open i2c
controller\n");
      printf ("INFO=>Exiting...\n");
      exit (1);
    }
  pcf8584_init();
  usleep(10);

  printf("\n************* HEP LAB I2c Controller *************");
  RW=send_slave_address(address);

  if (RW==0)
  {
  single_byte_write(data);
  }
  if (RW==1)
  {
  ret=single_byte_read();
  printf("\nRead byte:%2x",ret);
  }
  if (RW==-1)
  {
  ret=RW;
  }
  printf("\n**************** End of Operation ****************\n\n");

  close(fd);
  return ret;
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/************* labview write APV6 ************/

int lv_write_apv6(int address,char register_name[11],int data)
{
long int read_byte;
int reg;

  fd= open("/dev/i2cdrv00",6);
  if (fd==0)
    { printf ("INFO=>An error occured while trying to open i2c
controller\n");
      printf ("INFO=>Exiting...\n");
      exit (1);
    }
  pcf8584_init();
  usleep(1000);
  printf("\n***** HEP LAB I2c Controller for APV6 - Write Mode
***********\n");
  reg=apv6_register_select(register_name);
  printf("\n* reg=%x *\n",reg);
  write_to_apv6(address,reg,data);
  printf("\n**************** End of Operation ****************\n\n");
  close(fd);
  return 0;
}



/************ labview read appv6 *****************/

int lv_read_apv6(int address,char register_name[11])
{
long int read_byte;
int reg;

  fd= open("/dev/i2cdrv00",6);
  if (fd==0)
    { printf ("INFO=>An error occured while trying to open i2c
controller\n");
      printf ("INFO=>Exiting...\n");
      exit (1);
    }
  pcf8584_init();
  usleep(1000);
  printf("\n***** HEP LAB I2c Controller for APV6 - Read Mode
************\n");
  reg=apv6_register_select(register_name);
  read_byte=read_from_apv6(address,reg);
  printf("\nRead byte:%2x",read_byte);
  printf("\n**************** End of Operation ****************\n\n");

  close(fd);
  return read_byte;
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/**************** APV6 Functions *****************************/
/********* write to APV6 ***********/
int write_to_apv6(int addr,int sub_addr,int data)
{

addr=(addr<<1);
send_slave_address(0x40);/*(addr);*/
printf("\n#trick! write to 0x40");
usleep(10000);
single_byte_write(sub_addr);
usleep(10000);
single_byte_write(data);
send_stop();

return 0;
}

/******* read from APV6 ***********/
long int read_from_apv6(int addr,int sub_addr)
{
long int read_byte;


addr=addr<<1;
send_slave_address(0x40);/*(addr);*/
printf("\n#trick! write to 0x40");
usleep(10000);
single_byte_write(sub_addr+1);
usleep(10000);
send_stop();
send_slave_address(0x45); /*(addr+1);*/
printf("\n#trick! read from 0x45");
usleep(10000);
read_byte=single_byte_read();

return read_byte;
}


/****** APV6 Register Select ******/

int apv6_register_select(char register_name[11])
{
int reg=0;

if (!strcmp(register_name,"ERROR"))
{
printf("\a\nINFO=>ERROR Register is Read-only\n");

}

if (!strcmp(register_name,"MODE"))    reg=MODE;
if (!strcmp(register_name,"LATENCY")) reg=LATENCY;
if (!strcmp(register_name,"IPRE"))    reg=IPRE;
if (!strcmp(register_name,"ISHA"))    reg=ISHA;
if (!strcmp(register_name,"IPSP"))    reg=IPSP;
if (!strcmp(register_name,"ISFB"))    reg=ISFB;
if (!strcmp(register_name,"VPRE"))    reg=VPRE;
if (!strcmp(register_name,"VSHA"))    reg=VSHA;
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
if (!strcmp(register_name,"VADJ"))      reg=VADJ;
if (!strcmp(register_name,"VCAS"))      reg=VCAS;
if (!strcmp(register_name,"CLVL"))      reg=CLVL;
if (!strcmp(register_name,"CSKW"))      reg=CSKW;
if (!strcmp(register_name,"CDRV"))      reg=CDRV;
if (!reg)
{
printf("\nERROR:Wrong Register Name\n");
reg=-1;

}
return reg;
}




/*****************PCF functions *****************************/

/************ PCF init ************/
int pcf8584_init(void)
{
writelong (0x0c, 0x5E000000); /* reset PCF8584 */
writecycle(0x80,1);
writecycle(0x55,0);
writecycle(0xA0,1);
writecycle(0x18,0);
writecycle(0xC1,1);
readcycle(1);
return 0;
}

/******* Send Slave Address *******/

int send_slave_address(int val)
{
int ret;
int RW=val%2;
int slave_address=val;
int read_byte;

writecycle(slave_address,0);
writecycle(start_condition,1);

if (!RW)
printf("\nINFO=>WriteMode");
else
printf("\nINFO=>ReadMode");
/*usleep(1);*/


ret=get_acknowledgement();
if (ret==-1)
RW=ret;

return RW;
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/**** single byte write *************/

int single_byte_write(int val)
{
int ret=0;

writecycle(val,0);
/*usleep(1);*/
ret=get_acknowledgement();

return ret;
}


/**** single byte read ************/
long int single_byte_read(void)
{
long int read_byte;

writecycle(0x40,1);
/*usleep(1);*/
readcycle(0);
/*usleep(1);*/
readcycle(1);
/*usleep(1);*/
writecycle(0xC3,1);
/*usleep(1);*/
read_byte=readcycle(0);
/*usleep(1);*/
return read_byte;
}



/***** Get acknowledgement ********/
int get_acknowledgement(void)
{
long int read_byte;
int ret=0;
read_byte=readcycle(1);
/*usleep(1);*/
read_byte &=ack_mask;

if (read_byte==0x08)
      {
      printf("\nINFO=>Error:No acknowledgement");
      /* printf("\nINFO=>End of Operation\n\n"); */
      writecycle(release_Bus,1);
      ret=-1;

      }
else
      {
      printf("\nINFO=>Success!");
      }/*end if*/

return ret;
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/****** Check If Bus Is Busy ******/

int check_if_bus_is_busy(void)
{
long int read_byte;
int ret=0;

writecycle(bus_is_busy,1);
usleep(1);
read_byte=readcycle(1);
read_byte &=busbusy_mask;
if (!read_byte)
{
printf("\nINFO=>Error:Bus is Busy");
ret=-1;

}

return ret;
}

/********** send_stop *************/
int send_stop(void)

{
writecycle(release_Bus,1);
return 0;
}


/***************** PCF read & write cycle ***********************/
/***************** WriteCycle *********************/


int writecycle(int val,int A0)
{
long int OMB_value;

int byte_to_send=val;

readlong (0x0c, &OMB_value);/* read current value of OMB */
usleep(1);

if (A0==0)
{
OMB_value &=clr_A0_mask_AND;   /* Clear A0*/
writelong (0x0c, OMB_value);
}
else
{
OMB_value |=set_A0_mask_OR;    /* Set A0*/
writelong (0x0c, OMB_value);
}



OMB_value &=enable_AND; /* enable s5920 internal logic*/
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
OMB_value &=write_mask_AND;        /* write enable */
OMB_value &=data_mask;             /* set data */
OMB_value |=byte_to_send;
writelong (0x0c, OMB_value);

/*usleep(1);*/


/*usleep(1);*/

OMB_value &=rw_cycle_mask;
OMB_value |=deasserted;
writelong (0x0c, OMB_value);
/*usleep(1);*/


/*usleep(1);*/

OMB_value &=rw_cycle_mask;
OMB_value |=cs_on;
writelong (0x0c, OMB_value);
/*usleep(1);*/


/*usleep(1);*/

OMB_value &=rw_cycle_mask;
OMB_value |=cs_wr_on;
writelong (0x0c, OMB_value);
/*usleep(1);*/


/*usleep(1);*/

OMB_value &=rw_cycle_mask;
OMB_value |=cs_on;
writelong (0x0c, OMB_value);
/*usleep(1);*/


/*usleep(1);*/

OMB_value &=rw_cycle_mask;
OMB_value |=deasserted;
writelong (0x0c, OMB_value);
/*usleep(1);*/


/*usleep(1);*/

return 0;
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/**************** ReadCycle *********************/


long int readcycle(int A0)

{
long int OMB_value, IMB_value;



readlong (0x0c, &OMB_value);/* read current value of OMB */
usleep(1);

if (A0==0)
{
OMB_value &=clr_A0_mask_AND;   /* Clear A0*/
writelong (0x0c, OMB_value);
}
else
{
OMB_value |=set_A0_mask_OR;    /* Set A0*/
writelong (0x0c, OMB_value);
}

OMB_value &=enable_AND; /* enable s5920 internal logic*/
OMB_value |=read_mask_OR;      /* read enable */
OMB_value &=rw_cycle_mask;      /* read  cycle */
OMB_value |=deasserted;
writelong (0x0c, OMB_value);
usleep(1);


OMB_value &=rw_cycle_mask;
OMB_value |=cs_on;
writelong (0x0c, OMB_value);

usleep(1);

OMB_value &=rw_cycle_mask;
OMB_value |=cs_rd_on;
writelong (0x0c, OMB_value);

usleep(1);

readlong (0x1c, &IMB_value);   /* read IMB */


OMB_value &=rw_cycle_mask;
OMB_value |=cs_on;
writelong (0x0c, OMB_value);
usleep(1);

OMB_value &=rw_cycle_mask;
OMB_value |=deasserted;
writelong (0x0c, OMB_value);
usleep(1);

return IMB_value;
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/********************** driver Functions ************************/
/****************** writelong **********************/
int writelong (int offset, int value)
{
long ret=-1, arg[3];

  arg[0]=offset;
  arg[2]=value;
  ret =ioctl(fd,WRITE_BASE0_LONG,arg);
  return ret;
}


/****************** readlong **********************/

int readlong(int offset,long* b)
{
  long ret =-1;
  long arg = offset;
  ret =ioctl(fd,READ_BASE0_LONG,&arg);
  *b = arg;
  return ret;
}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

4. i2c_functions.h

```
#define data_mask          0xFFFFFF00
#define enable_AND         0xBFFFFFFF /* clear EN# bit (DQ30) Logical
AND*/
#define disable_OR         0x40000000 /* set EN# bit (DQ30) Logical
OR*/
#define write_mask_AND     0x7FFFFFFF /* clear RD/WR# bit (DQ31)
Logical AND*/
#define read_mask_OR       0x80000000 /* setRD/WR# bit (DQ31) Logical
OR*/
#define set_A0_mask_OR     0x10000000 /* set A0 bit (DQ28) Logical OR*/
#define clr_A0_mask_AND    0xEFFFFFFF /* clear A0 bit (DQ28) Logical
AND*/

#define rw_cycle_mask      0xF00000FF
#define deasserted         0x0F000000
#define cs_on              0x0B000000
#define cs_wr_on           0x09000000
#define cs_rd_on           0x03000000

#define ack_mask           0x08
#define busbusy_mask       0x01
#define start_condition    0xC5
#define release_Bus         0xC3
#define bus_is_busy         0xC1

/* APV6 REGISTERS */

#define ERROR                0x01
#define MODE                0x02
#define LATENCY             0X04
#define IPRE                0x40
#define ISHA                0x42
#define IPSP                0x44
#define ISFB                0x46
#define VPRE                0x48
#define VSHA                0x4A
#define VADJ                0x4C
#define VCAS                0x4E
#define CLVL                0x50
#define CSKW                0x52
#define CDRV                0x54

int fd;


/*************** extras ****************************/
int lv_read_apv6(int address,char register_name[11]);
int lv_write_apv6(int address,char register_name[11],int data);
int i2c_app(int address,int data);



/*************** Driver functions ******************/
int writelong (int offset, int value);
int readlong(int offset,long* b);
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
/*************** PCF read&write cycles *************/
int writecycle(int val,int A0);
long int readcycle(int A0);



/*************** PCF main functions ****************/
int pcf8584_init(void);
int send_slave_address(int val);
int single_byte_write(int val);
long int single_byte_read(void);
int get_acknowledgement(void);
int check_if_bus_is_busy(void);
int send_stop(void);
/*************** APV6 functions *******************/
int write_to_apv6(int addr,int sub_addr,int data);
long int read_from_apv6(int addr,int sub_addr);
int apv6_register_select(char register_name[11]);
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

5. i2c_app.c

```
#include "i2c_functions.h"
/********************** main program *****************************/
int main ()
{
int typed_address,typed_byte,RW;
long int read_byte;
int ret=0;

  fd= open("/dev/i2cdrv00",6);
  if (fd==0)
    { printf ("INFO=>An error occured while trying to open i2c
controller\n");
      printf ("INFO=>Exiting...\n");
      exit (1);
    }

  pcf8584_init();
  usleep(10);

  printf("\n************* HEP LAB I2c Controller *************");
  printf("\nType slave address (HEX):");
  scanf("%x",&typed_address);
  RW=send_slave_address(typed_address);

  if (RW==0)
  {
  printf("\nType byte to write (HEX):");
  scanf("%x",&typed_byte);
  single_byte_write(typed_byte);
  }

  if (RW==1)
  {
  read_byte=single_byte_read();
  printf("\nRead byte:%#2x\n",read_byte);
  }

  if (RW==-1)
  {
  ret=RW;
  }

  send_stop(void);

  printf("\nINFO=>End of Operation\n\n");
  close(fd);
  return ret;
}
```

6. apv6.c

```
#include "i2c_functions.h"
/********************** main program *****************************/
int main ()
{
long int read_byte;
int APV6_address,reg,data;
char register_name[11];
char RW;

  fd= open("/dev/i2cdrv00",6);
  if (fd==0)
    { printf ("INFO=>An error occured while trying to open i2c
controller\n");
      printf ("INFO=>Exiting...\n");
      exit (1);
    }

  pcf8584_init();
  usleep(1000);

  printf("\n************* HEP LAB I2c Controller for APV6
*************\n");
  printf("\nPress 'W' to write to APV6 or 'R'  to read from APV6:");
  RW=getchar();
  if (RW=='W' || RW=='w' || RW=='R' || RW=='r')
      {
      printf("\nType APV6 7-bit address [HEX]:");
      scanf("%x",&APV6_address);
      if (APV6_address<0x10 || APV6_address>0x1F)
             {
             printf("\nERROR:Wrong APV6 Addressing\n");
             exit(1);
             }
      printf("\nType Register name[caps]:");
      scanf("%s",&register_name);
      reg=apv6_register_select(register_name);
      }
 else
      {
      printf("\nERROR[wrong input]\n");
      exit(1);
      }
 if (RW=='W' || RW=='w')
      {
      printf("\nType data byte [HEX]:");
      scanf("%x",&data);
      write_to_apv6(APV6_address,reg,data);
         }
 if (RW=='R' || RW=='r')
      {
      read_byte=read_from_apv6(APV6_address,reg);
      printf("\nRead byte:%2x\n",read_byte);
      }
 printf("\n**************** End of Operation ****************\n\n");
 close(fd);
 return 0;}
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

7. makefile

```
#
#       Makefile for building:
#

VERSION=1.1
ID=PCI-i2c
DIST_NAME=$(ID).$(VERSION).tgz
MAJOR_DEV=147
SRCS   = i2cdrv.c simple.c i2c_app.c i2c_functions.c apv6.c
#testi2cdrv.c
HEADERS = i2cdrv.h i2c_functions.h
OBJS   = i2cdrv.o i2c_functions.o i2c_app.o apv6.o
CKFLAGS       = -DADAPTER_ID=\"$(ID)\" -D__KERNEL__ -
DDEFAULT_MAJOR_DEV=$(MAJOR_DEV) -Wall -O6 -fomit-frame-pointer -m486
-I/usr/src/linux/include -I/usr/include/asm
LDKFLAGS = -s -N
CC=gcc
BINDIR=/sbin
TARGETS=i2cdrv.o simple i2c_functions.o i2c_app i2c_app.o apv6.o apv6
DIST_FILES={i2cdrv.h,i2cdrv.c,simple.c,i2c_app.c,apv6.c,i2c_functions
.c,i2c_functions.h,Makefile}

all: $(TARGETS)

simple:            simple.c
      $(CC) -Wall -g -o $@ $@.c -lm


i2cdrv.o:i2cdrv.c i2cdrv.h $(KERNEL_VERSION)
      $(CC) $(CKFLAGS) -c i2cdrv.c -o i2cdrv_compile.o
      $(LD) -r -o $@ i2cdrv_compile.o


CFLAG = -c -g -O4
SPECIAL_CFLAGS = -fPIC -c
#SPECIAL_CFLAGS =
#LIB      = -llynx

apv6: apv6.c i2c_functions.o
      $(CC) apv6.c i2c_functions.o -o apv6 $(LIB)

app: i2c_app.c i2c_functions.o
      $(CC) i2c_app.c i2c_functions.o -o i2c_app $(LIB)

functions:i2c_functions.c i2c_functions.h
      $(CC) $(CFLAG) i2c_functions.c -o i2c_functions.o

shared_lib: i2c_functions.c  i2c_functions.h
      $(CC) $(SPECIAL_CFLAGS) i2c_functions.c
      $(CC) -shared -Wl,-soname,libi2c_functions.so.1 -o
libi2c_functions.so i2c_functions.o -lc
      cp libi2c_functions.so /usr/lib
      ldconfig -n /usr/lib
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
clean:
	rm -f *.o \#* *~ $(TARGETS)

dist:
	- make clean
	cd ..; tar -zcvf $(DIST_NAME) $(DIST_FILES);

install:
	-/sbin/rmmod i2cdrv
	-/sbin/insmod -f i2cdrv.o
	-/bin/cp ./i2cdrv.h /usr/local/include/i2cdrv.h
	-/bin/chmod 644 /usr/local/include/i2cdrv.h
	-install ./i2cdrv.o /lib/modules/preferred/misc/i2cdrv.o
# for non Red-Hat distributions comment the above line and uncomment
the one below.
#	-install ./i2cdrv.o /lib/modules/`uname -r`/misc/i2cdrv.o


devices:
	-/bin/rm  /dev/i2c*
	-/bin/mknod /dev/i2cdrv00 c $(MAJOR_DEV) 0
	-/bin/mknod /dev/i2cdrv01 c $(MAJOR_DEV) 1
	-/bin/mknod /dev/i2cusr00 c $(MAJOR_DEV) 16
	-/bin/mknod /dev/i2cusr01 c $(MAJOR_DEV) 17
	-/bin/mknod /dev/i2cusr02 c $(MAJOR_DEV) 18
	-/bin/mknod /dev/i2cusr03 c $(MAJOR_DEV) 19
	-/bin/mknod /dev/i2cusr04 c $(MAJOR_DEV) 20
	-/bin/mknod /dev/i2cusr05 c $(MAJOR_DEV) 21
	-/bin/chmod 666 /dev/i2cdrv*
	-/bin/chmod 666 /dev/i2cusr*

depend:
	-/bin/cp Makefile.orig Makefile

# DO NOT DELETE

i2cdrv.o: /usr/include/linux/autoconf.h /usr/include/linux/version.h
i2cdrv.o: /usr/include/linux/module.h /usr/include/linux/config.h
i2cdrv.o: /usr/include/asm/atomic.h /usr/include/linux/types.h
i2cdrv.o: /usr/include/linux/posix_types.h
/usr/include/linux/stddef.h
i2cdrv.o: /usr/include/asm/posix_types.h /usr/include/asm/types.h
i2cdrv.o: /usr/include/linux/errno.h /usr/include/asm/errno.h
i2cdrv.o: /usr/include/linux/fs.h /usr/include/linux/linkage.h
i2cdrv.o: /usr/include/linux/limits.h /usr/include/linux/wait.h
i2cdrv.o: /usr/include/linux/vfs.h /usr/include/asm/statfs.h
i2cdrv.o: /usr/include/linux/net.h /usr/include/linux/socket.h
i2cdrv.o: /usr/include/asm/socket.h /usr/include/asm/sockios.h
i2cdrv.o: /usr/include/linux/sockios.h /usr/include/linux/uio.h
i2cdrv.o: /usr/include/linux/kdev_t.h /usr/include/linux/ioctl.h
i2cdrv.o: /usr/include/asm/ioctl.h /usr/include/linux/list.h
i2cdrv.o: /usr/include/linux/dcache.h /usr/include/linux/stat.h
i2cdrv.o: /usr/include/linux/bitops.h /usr/include/asm/bitops.h
i2cdrv.o: /usr/include/asm/cache.h /usr/include/linux/major.h
i2cdrv.o: /usr/include/linux/mm.h /usr/include/linux/sched.h
i2cdrv.o: /usr/include/asm/param.h /usr/include/linux/binfmts.h
i2cdrv.o: /usr/include/linux/ptrace.h /usr/include/asm/ptrace.h
i2cdrv.o: /usr/include/linux/capability.h
/usr/include/linux/personality.h
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

```
i2cdrv.o: /usr/include/linux/tasks.h /usr/include/linux/kernel.h
i2cdrv.o: /usr/include/linux/times.h /usr/include/linux/timex.h
i2cdrv.o: /usr/include/asm/timex.h /usr/include/asm/msr.h
i2cdrv.o: /usr/include/asm/system.h /usr/include/asm/segment.h
i2cdrv.o: /usr/include/asm/semaphore.h /usr/include/asm/spinlock.h
i2cdrv.o: /usr/include/asm/page.h /usr/include/linux/smp.h
i2cdrv.o: /usr/include/linux/tty.h /usr/include/linux/sem.h
i2cdrv.o: /usr/include/linux/ipc.h /usr/include/linux/signal.h
i2cdrv.o: /usr/include/asm/signal.h /usr/include/asm/siginfo.h
i2cdrv.o: /usr/include/linux/securebits.h /usr/include/linux/time.h
i2cdrv.o: /usr/include/linux/param.h /usr/include/linux/resource.h
i2cdrv.o: /usr/include/asm/resource.h /usr/include/linux/timer.h
i2cdrv.o: /usr/include/asm/processor.h /usr/include/asm/vm86.h
i2cdrv.o: /usr/include/asm/math_emu.h /usr/include/asm/sigcontext.h
i2cdrv.o: /usr/include/linux/ioport.h /usr/include/linux/malloc.h
i2cdrv.o: /usr/include/linux/slab.h /usr/include/linux/string.h
i2cdrv.o: /usr/include/asm/string.h /usr/include/linux/delay.h
i2cdrv.o: /usr/include/asm/delay.h /usr/include/asm/io.h
i2cdrv.o: /usr/include/asm/uaccess.h i2cdrv.h
i2cdrv.o: /usr/include/linux/pci.h
```

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΜΕΡΟΣ ΔΕΥΤΕΡΟ

ΜΕΤΑΤΡΟΠΕΑΣ ΨΗΦΙΑΚΩΝ ΣΗΜΑΤΩΝ TTL-ECL-NIM-LVDS ΓΙΑ ΠΕΙΡΑΜΑΤΑ ΦΥΣΙΚΗΣ ΥΨΗΛΩΝ ΕΝΕΡΓΕΙΩΝ

Εισαγωγή

2.1 Ψηφιακά σήματα 78

2.1.1 TTL 78
2.1.2 NIM 79
2.1.3 ECL 79
2.1.4 LVDS 80

2.2 Μετατροπή ψηφιακών σημάτων 82

2.2.1 TTL σε LVDS 85
2.2.2 LVDS σε TTL 85
2.2.3 TTL σε ECL 86
2.2.4 ECL σε TTL 86
2.2.5 TTL σε NIM 87
2.2.6 NIM σε TTL 90

2.3 Σχεδιασμός – Υλοποίηση 91

2.3.1 Υπομονάδα μετατροπής TTL σε ECL-NIM-LVDS 91
2.3.2 Υπομονάδα μετατροπής ECL-NIM-LVDS σε TTL 95
2.3.3 Τελική υλοποίηση του μετατροπέα ψηφιακών σημάτων 101

2.4 Μετρήσεις 103

2.5 Αποτίμηση της μονάδας – Συμπεράσματα 103

2.6 Βιβλιογραφία 112

ΠΑΡΑΡΤΗΜΑ 113

Α. Φωτογραφίες 113
Β. NIM Connector 115
Γ. Εσωτερική λογική του XC9536 116
Δ. Τροφοδοτικό 117
Ε. Τυπωμένα κυκλώματα 118
1. Τροφοδοτικό – BOTTOM LAYER 118
2. Τροφοδοτικό – COMPONENTS 118
3. Υπομονάδα TTL ΣΕ ECL – NIM – LVDS – TOP LAYER 119
4. Υπομονάδα TTL ΣΕ ECL – NIM – LVDS – BOTTOM LAYER 120
5. Υπομονάδα TTL ΣΕ ECL – NIM – LVDS – COMPONENTS 121
6. Υπομονάδα ECL – NIM – LVDS ΣΕ TTL – TOP LAYER 122
7. Υπομονάδα ECL– NIM – LVDS ΣΕ TTL – BOTTOM LAYER 123
8. Υπομονάδα ECL–NIM–LVDS ΣΕ TTL - COMPONENT 124

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ΜΕΡΟΣ ΔΕΥΤΕΡΟ

ΜΕΤΑΤΡΟΠΕΑΣ ΨΗΦΙΑΚΩΝ ΣΗΜΑΤΩΝ TTL-ECL-NIM-LVDS ΓΙΑ ΠΕΙΡΑΜΑΤΑ ΦΥΣΙΚΗΣ ΥΨΗΛΩΝ ΕΝΕΡΓΕΙΩΝ

Εισαγωγή

Οι ηλεκτρονικές μονάδες που χρησιμοποιούνται σε πειράματα φυσικής υψηλών ενεργειών (ΦΥΕ), παράγουν, εκτός των άλλων, και ψηφιακά σήματα εξόδου. Τα σήματα αυτά διαφέρουν από μονάδα σε μονάδα και η επιλογή του τύπου που χρησιμοποιείται εξαρτάται από την εφαρμογή, τον κατασκευαστή και από διάφορους άλλους παράγοντες.

Σκοπός του μετατροπέα που υλοποιήθηκε είναι η εύκολη διασύνδεση μεταξύ πειραματικών ηλεκτρονικών μονάδων ΦΥΕ που χρησιμοποιούν σήματα TTL, ECL, NIM και LVDS. Ο μετατροπέας σχεδιάστηκε έτσι ώστε να λειτουργεί ικανοποιητικά σε συχνότητες έως και 40 MHz δίνοντας τη δυνατότητα μετατροπής οκτώ καναλιών.

2.1 Ψηφιακά σήματα

Στις επόμενες παραγράφους γίνεται μια σύντομη περιγραφή των σημάτων TTL, ECL, NIM και LVDS που υποστηρίζονται από τον εν λόγω μετατροπέα ψηφιακών σημάτων.

2.1.1 TTL

Για περισσότερες από δύο δεκαετίες η λογική TTL (**T**ransistor **T**ransistor **L**ogic) υπήρξε εξαιρετικά δημοφιλής. Μάλιστα για το μεγαλύτερο όγκο των ψηφιακών εφαρμογών που χρησιμοποιούν κυκλώματα με μικρή ή μεσαία κλίμακα ολοκλήρωσης, την λογική TTL ανταγωνίζεται μόνο η CMOS. Η λογική TTL βασίζεται σε διπολικά transistor που λειτουργούν είτε στον κόρο είτε στην αποκοπή[1] (saturating logic) [1]. Τα σήματα TTL είναι θετικής λογικής και τα λογικά τους επίπεδα είναι τα εξής: λογικό

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

σε γενικές γραμμές ο χρόνος ανόδου (rise time) αλλά και η καθυστέρηση μετάδοσης (propagation delay), δύο από τα σημαντικότερα χαρακτηριστικά των λογικών κυκλωμάτων, είναι της τάξης των 10ns. Ένα άλλο μειονέκτημα των TTL είναι μεγάλη τους ευαισθησία στο θόρυβο. Τα χαρακτηριστικά τους αυτά τα καθιστούν ακατάλληλα για μετάδοση δεδομένων σε υψηλές συχνότητες (πάνω των 100MHz) και σε μεγάλες αποστάσεις.

2.1.2 NIM

Το πρότυπο NIM (DOE/ER-0457), ακρωνύμιο για το **N**uclear **I**nstrumentation **M**ethods, εμφανίστηκε και καθιερώθηκε στον χώρο της πυρηνικής φυσικής και φυσικής υψηλών ενεργειών το 1964 [2]. Ο σκοπός του NIM ήταν να προσφέρει μεγαλύτερη ευελιξία στην χρήση των μόναδων που χρησιμοποιούνται στα πειράματα των προαναφερθέντων τομέων της φυσικής, κάτι που επιτεύχθηκε, με αποτέλεσμα το πρότυπο αυτό να χρησιμοποιείται ευρέως ακόμη και σήμερα. Το πρότυπο NIM καθόρισε επίσης τρεις κατηγορίες λογικών επιπέδων για ψηφιακά σήματα. Απο αυτές, η πλέον διαδεδομένη είναι η ταχεία αρνητική λογική (fast-negative logic), που αναφέρεται συχνά και ως λογική NIM (NIM logic). Στη λογική αυτή τα επίπεδα καθορίζονται από τιμές ρεύματος και όχι τάσης, όπως συνηθίζεται (λογικό 0 στα 0mA και λογικό 1 στα -16mA). Καθώς όμως το NIM απαιτεί αντίσταση τερματισμού της τάξης των 50Ω, οι τιμές του ρεύματος που καθορίζουν τα λογικά επίπεδα αντιστοιχούν σε τάσεις 0V και -0,8V για λογικό 0 και 1 αντίστοιχα (Σχ.2.1). Τα κυκλώματα αυτής της λογικής μπορούν να επιτύχουν χρόνο ανόδου της τάξης του 1nsec.

2.1.3 ECL

Η λογική ζεύξης εκπομπού ECL (**E**mitter **C**oupled **L**ogic) είναι μια από τις γρηγορότερες οικογένειες ολοκληρωμένων κυκλωμάτων. Η ταχύτητα τους οφείλεται στο γεγονός ότι όλα τα transistor που τα αποτελούν δεν λειτουργούν στον κόρο (saturation), σε αντίθεση με τα TTL. Με αυτόν τον τρόπο αποφεύγονται οι καθυστερήσεις λόγω του χρόνου αποθήκευσης [1]. Μια από τις ιδιαιτερότητες των

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

ECL είναι οτι έχουν αρνητική τάση τροφοδοσίας[2] (-5.2V). Παρά το γεγονός ότι τροφοδοτούνται αρνητικά, τα σήματα ECL είναι θετικής λογικής, δηλαδή η τάση που αντιστοιχεί στο λογικό 1 είναι μεγαλύτερη από αυτήν του λογικού 0 (-0.8V για λογικό 1 και –1.6V για λογικό 0), οπως απεικονίζεται και στο Σχ.2.1

Το μικρό αυτό πλάτος του σήματος, και συνεπώς της διακύμανσης μεταξύ λογικού 0 και 1, το οποίο είναι της τάξης των 800mV [-0.8V-(-1.6V)], έχει ως αποτέλεσμα μικρούς χρόνους ανόδου και καθόδου (rise και fall time) της τάξης του 1ns. Ένα άλλο σημαντικό χαρακτηριστικό τους είναι ότι μεταδίδονται διαφορικά (με συνεστραμμένα ζεύγη καλωδίων), με αποτέλεσμα ο θόρυβος να τα επηρεάζει ελάχιστα [3].

Αυτά τα χαρακτηριστικά των ECL επιτρέπουν μετάδοση δεδομένων σε μεγάλες ταχύτητες (φτάνουν έως και Gbps) και σε μεγάλες αποστάσεις[3]. Ένα σημαντικό όμως μειονέκτημα που παρουσιάζουν τα ECL είναι η πολύ υψηλή κατανάλωση ισχύος, η οποία εξαρτάται και από τη συχνότητα των σημάτων.

2.1.4 LVDS

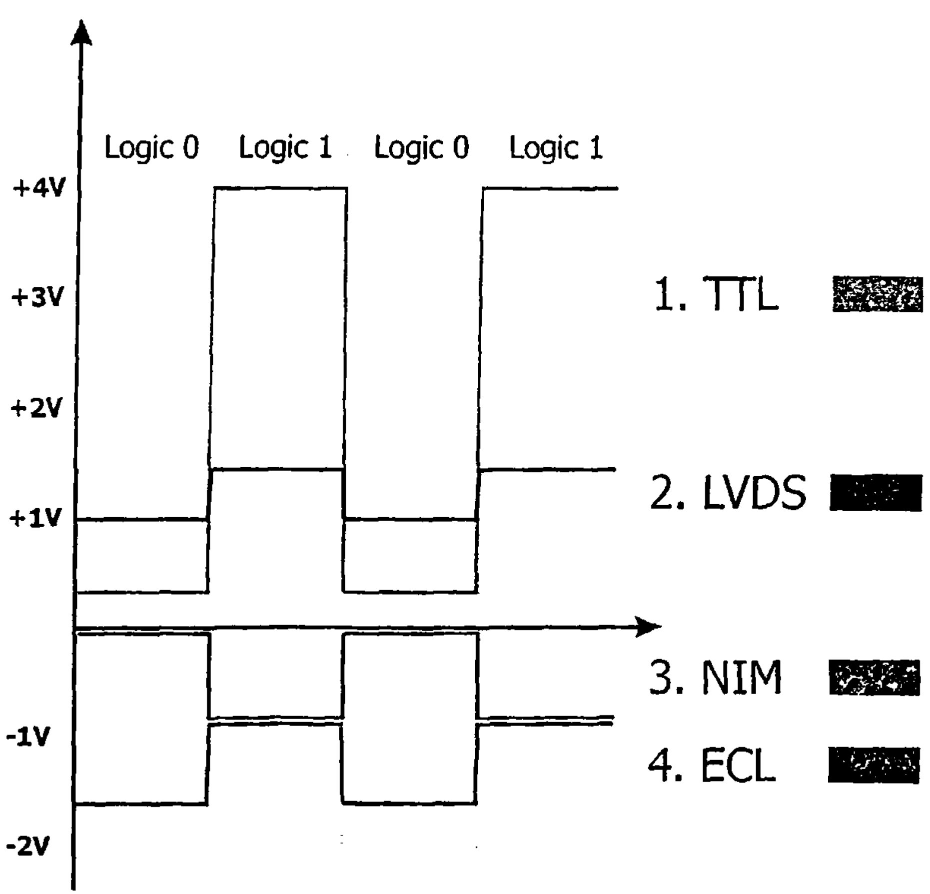
Η τεχνολογία LVDS (**L**ow **V**oltage **D**ifferential **S**ignaling), είναι ένα νέο πρότυπο ψηφιακού σήματος που καλύπτει τις σημερινές ανάγκες για μετάδοση δεδομένων με πολύ υψηλή ταχύτητα. Είναι σχεδιασμένη επίσης έτσι ώστε να είναι συμβατή με τις ανάγκες μελλοντικών εφαρμογών χαμηλής τάσης, καθώς η τάση τροφοδοσίας που απαιτείται μπορεί να φτάσει μέχρι και τα 2V [4]. Η τεχνολογία LVDS βασίζεται στο πρότυπο ANSI/ΤΙΑ/ΕΙΑ-644. Τα κύρια χαρακτηριστικά της είναι το διαφορικό σήμα χαμηλής τάσης το οποίου το πλάτος είναι της τάξης των 330mV (250mV ελάχιστη και 450mV μέγιστη τιμή) και οι χρόνοι μετάβασης (χρόνοι ανόδου και καθόδου) οι οποίοι είναι της τάξης του 1ns. Τα λογικά επίπεδα είναι περίπου 1V και 1,4V για λογικό 0 και 1, αντίστοιχα. Αυτά τα χαρακτηριστικά των LVDS επιτρέπουν μετάδοση δεδομένων σε μεγάλες ταχύτητες (φτάνουν έως και Gbps) και σε μεγάλες αποστάσεις. Επιπλέον, το πολύ μικρό πλάτος του σήματος, ελαχιστοποιεί

[2] *Υπάρχουν βέβαια και ολοκληρωμένα κυκλώματα ECL με θετική τάση τροφοδοσίας (PECL – Positive ECL), τα οποία όμως δεν υποστηρίζονται από την εν λόγω μονάδα μετατροπής.*

[3] *Μια από τις εφαρμογές των σημάτων ECL είναι οι κάρτες Ethernet*

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

την κατανάλωση ισχύος, σε αντίθεση με την τεχνολογία ECL, παρέχοντας επίσης και όλα τα προτερήματα της διαφορικής μετάδοσης σήματος (παρ' όλα αυτά η τεχνολογία LVDS δεν είναι τόσο διαδεδομένη όσο αυτή των ECL). Μία από τις πιο συνήθεις εφαρμογές είναι η μετατροπή σημάτων TTL σε LVDS, η υψηλής ταχύτητας μετάδοσή τους μέσω καλωδιώσεων και η μετατροπή τους ξανά σε TTL[4]. Η τεχνολογία αυτή, εκτός από την υψηλή ταχύτητα, χαμηλή κατανάλωση και χαμηλό θόρυβο, μειώνει δραματικά και το κόστος, καθώς δεν απαιτούνται ιδιαίτερα καλής ποιότητας καλωδιώσεις και συνδετήρες (connectors).



Σχήμα 2.1 Τα λογικά επίπεδα των σημάτων LVDS, NIM και ECL

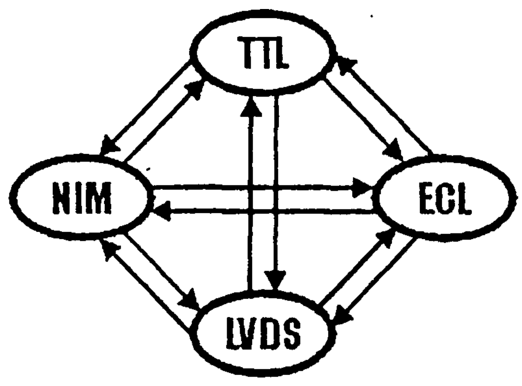
[4] *Μια από τις εφαρμογές της τεχνολογίας LVDS είναι οι σκληροί δίσκοι Ultra 2 SCSI*

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.2 Μετατροπή ψηφιακών σημάτων

Στις παρακάτω παραγράφους αναλύονται λεπτομερώς οι μετατροπές μεταξύ των ψηφιακών σημάτων, καθώς και η λογική με την οποία αναπτύχθηκε ο μετατροπέας.

Η ιδανική υλοποίηση του μετατροπέα ψηφιακών σημάτων είναι η απευθείας μετατροπή του καθενός σήματος σε όλα τα άλλα, όπως φαίνεται στο διάγραμμα του Σχ.2.2. Στην πράξη, αυτή η περίπτωση αποδείχθηκε αρκετά σύνθετη, καθώς το κύκλωμα περιπλέκεται αρκετά και η υλοποίησή του σε τυπωμένο κύκλωμα δύο επιπέδων γίνεται σχεδόν αδύνατη. Επίσης, ο αριθμός των συνδετήρων (connectors), αυξάνεται υπερβολικά έχοντας ως αποτέλεσμα την αύξηση του όγκου της κατασκευής, καθώς απαιτείται διπλάσια επιφάνεια στήριξης των συνδετήρων. Τέλος διπλασιάζεται και το κόστος της κατασκευής.



Σχήμα 2.2 Απευθείας μετατροπή των ψηφιακών σημάτων

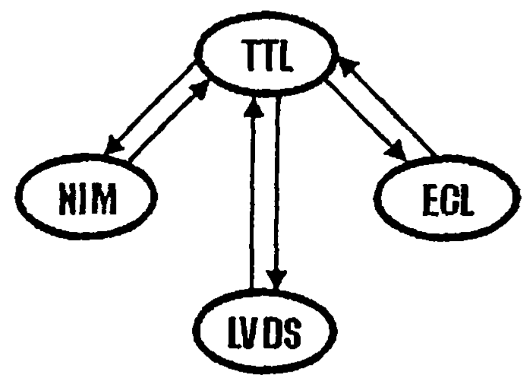
Πιο συγκεκριμένα, απαιτούνται δώδεκα κυκλώματα μετατροπής, ένα για κάθε μετατροπή, και δεκαέξι συνδετήρες για κάθε κανάλι μετατροπής, όπως φαίνεται στον Πίνακα 2.1. Εφόσον πρόκειται για μετατροπέα οκτώ καναλιών, οι connectors που απαιτούνται φτάνουν τους 16x8=128.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

	Απαιτούμενοι connectors	
Μετατροπή	Ποσότητα	Περιγραφή
TTL to ECL,NIM,LVDS	4	TTL input, ECL-NIM-LVDS output
ECL to TTL,NIM,LVDS	4	ECL input, TTL-NIM-LVDS output
NIM to TTL,ECL,LVDS	4	NIM input, TTL-ECL-LVDS output
LVDS to TTL,ECL,NIM	4	LVDS input, TTL-ECL-NIM output

Πίνακας 2.1 Απαιτούμενοι connectors για άμεση μετατροπή

Η ανάπτυξη του εν λόγω μετατροπέα ψηφιακών σημάτων βασίστηκε στην ιδέα ότι η βαθμίδα TTL αποτελεί τον ενδιάμεσο κρίκο όλων των μετατροπών, όπως απεικονίζεται στο Σχ.2.3. Έτσι, η μετατροπή απο LVDS σε ECL γίνεται έμμεσα, μετατρέποντας πρώτα το LVDS σε TTL, και στη συνέχεια το TTL σε ECL.



Σχήμα 2.3 Έμμεση μετατροπή των ψηφιακών σημάτων

Με αυτήν την σχεδίαση, μειώνονται κατά 50% τα κυκλώματα μετατροπής (6 κυκλώματα αντί για 12), καθώς και οι απαιτούμενοι connectors (64 connectors αντί για 128), όπως φαίνεται και στον Πίνακα 2.2

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

	Απαιτούμενοι connectors	
Μετατροπή	[illegible]	Περιγραφή
TTL to ECL,NIM,LVDS	4	TTL input, ECL-NIM-LVDS output
ECL,NIM,LVDS to TTL	4	ECL-NIM-LVDS input, TTL output

Πίνακας 2.2 Απαιτούμενοι connectors για έμμεση μετατροπή

Εκτός αυτού, υπάρχει ένας μόνο connector εξόδου ψηφιακού σήματος για κάθε κανάλι, σε αντίθεση με την ιδανική περίπτωση της άμεσης μετατροπής, όπου απαιτούνται παραπάνω του ενός connector εξόδου . Έτσι, για παράδειγμα, στην πρώτη περίπτωση υπάρχει ένας κοινός connector εξόδου LVDS για το δεύτερο κανάλι μετατροπής, ενώ στην αντίθετη περίπτωση θα έπρεπε υπάρχουν τρεις (ένας για LVDS που προέρχεται από NIM, ένας για LVDS που προέρχεται από TTL και ένας για LVDS που προέρχεται από ECL). Συνεπώς, με αυτήν τη σχεδίαση επιτεύχθηκε και απλούστευση της χρήσης του μετατροπέα.

Τα μειονεκτήματα αυτής της μεθόδου είναι ότι αυξάνεται ο χρόνος μετάδοσης του σήματος (propagation delay), λόγω της έμμεσης μετατροπής, καθώς επίσης και το ότι μειώνεται η συχνότητα μέγιστης λειτουργίας, λόγω της αδυναμίας του TTL να λειτουργήσει σε πολύ υψηλές συχνότητες (πάνω από 100MHz) [5]. Το γεγονός αυτό δεν αποτελεί ανασταλτικό παράγοντα, καθώς οι μεν απαιτήσεις σε ταχύτητα των πειραμάτων ΦΥΕ είναι μικρότερες, και εκτός των άλλων είναι αρκετά δύσκολο από κατασκευαστικής άποψης μια τέτοια μονάδα να φτάσει πολύ ψηλότερα σε συχνότητα, λόγω των υπάρχουσων καλωδιώσεων. Έναν άλλο λόγο που το TTL επιλέχθηκε ως ενδιάμεσος κρίκος αποτελεί και το γεγονός ότι στην αγορά υπάρχει μεγάλη γκάμα διαθέσιμων προϊόντων για μετατροπές από και σε TTL σε προσιτές τιμές.

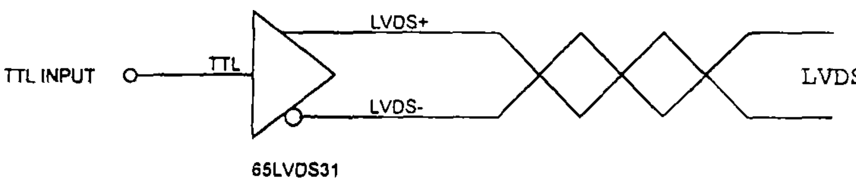
Πάντως, θα πρέπει να τονιστεί ιδιαίτερα ότι η λύση με τα καλύτερα αποτελέσματα στο θέμα της μετατροπής ψηφιακών σημάτων είναι η προσθήκη του απαραίτητου κυκλώματος μετατροπής στη βαθμίδα εισόδου της κάθε μονάδας, που χρησιμοποιεί αυτά τα σήματα (on-board).

Στις επόμενες παραγράφους αναλύονται τα κυκλώματα μετατροπής σημάτων που συμπεριλαμβάνονται στο μετατροπέα.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.2.1 TTL σε LVDS

Για την μετατροπή από TTL σε LVDS, υλοποιήθηκε το κύκλωμα που απεικονίζεται στο Σχ.2.4. Για τη μετατροπή αυτή χρησιμοποιήθηκε το ολοκληρωμένο κύκλωμα 65LVDS31 της Texas Instruments. Πρόκειται για έναν οδηγό διαφορικού σήματος (line driver) τεσσάρων καναλιών, ειδικά σχεδιασμένο για εφαρμογές που απαιτούν ιδιαίτερα χαμηλή κατανάλωση ισχύος και υψηλούς ρυθμούς μετάδοσης δεδομένων. Η κατανάλωση ισχύος είναι της τάξης των 25 mW ανά κανάλι στη μέγιστη συχνότητα λειτουργίας 200MHz . Οι τυπικοί χρόνοι ανόδου και μετάδοσης είναι 0.5ns και 1,7ns αντίστοιχα. Δέχεται στην είσοδό του σήματα επιπέδου TTL και τα μετατρέπει σε διαφορικό σήμα LVDS πλάτους 400mV. Τα σήματα εξόδου που παράγονται, μεταδίδονται με συνεστραμμένα ζεύγη καλωδίων (twisted pairs), και απαιτούν φορτίο 100Ω, το οποίο συνδέεται μεταξύ των εισόδων του δέκτη LVDS που τα λαμβάνει [4].

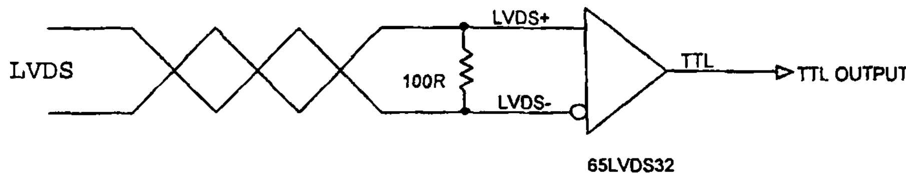


Σχήμα 2.4 Μετατροπή από TTL σε LVDS

2.2.2 LVDS σε TTL

Για την μετατροπή από LVDS σε TTL, υλοποιήθηκε το κύκλωμα που απεικονίζεται στο Σχ.2.5. Για τη μετατροπή αυτή χρησιμοποιήθηκε το ολοκληρωμένο κύκλωμα 65LVDS32 της Texas Instruments. Πρόκειται για ένα δέκτη διαφορικού σήματος (line receiver) τεσσάρων καναλιών, ειδικά σχεδιασμένο για εφαρμογές που απαιτούν ιδιαίτερα χαμηλή κατανάλωση ισχύος και υψηλούς ρυθμούς μετάδοσης δεδομένων. Οι τυπικοί χρόνοι ανόδου και μετάδοσης είναι 0.8ns και 4ns αντίστοιχα. Δέχεται στην είσοδό του διαφορικά σήματα LVDS πλάτους 400mV και τα μετατρέπει σε σήματα TTL. Τα σήματα LVDS που λαμβάνονται, προσαρμόζονται με φορτίο 100Ω μεταξύ των δύο εισόδων [4].

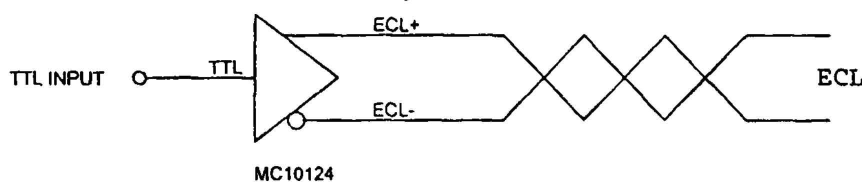
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 2.5 Μετατροπή από LVDS σε TTL

2.2.3 TTL σε ECL

Για την μετατροπή από TTL σε ECL, υλοποιήθηκε το κύκλωμα που απεικονίζεται στο Σχ.2.6. Για τη μετατροπή αυτή χρησιμοποιήθηκε ο line driver τεσσάρων καναλιών MC10124 της Motorola. Η τυπική κατανάλωση ισχύος είναι της τάξης των 400 mW (χωρίς φορτίο). Οι τυπικοί χρόνοι ανόδου και μετάδοσης είναι 2,5ns και 3,5ns αντίστοιχα. Δέχεται στην είσοδό του σήματα επιπέδου TTL και τα μετατρέπει σε διαφορικό σήμα ECL πλάτους 800mV. Τα σήματα εξόδου που παράγονται, μεταδίδονται με συνεστραμμένα ζεύγη καλωδίων (twisted pairs), και απαιτούν φορτίο 100Ω, το οποίο συνδέεται μεταξύ των εισόδων του δέκτη ECL που τα λαμβάνει. Με αυτόν τον τρόπο μια πληροφορία TTL μπορεί να μεταδοθεί διαφορικά μέσω συνεστραμμένων καλωδίων, χωρίς να επηρεάζεται από το θόρυβο [6].



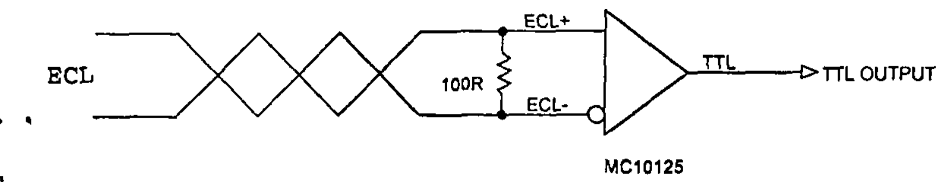
Σχήμα 2.6 Μετατροπή από TTL σε ECL

2.2.4 ECL σε TTL

Για την μετατροπή από ECL σε TTL, υλοποιήθηκε το κύκλωμα που απεικονίζεται στο Σχ.2.7. Για τη μετατροπή αυτή χρησιμοποιήθηκε ο line receiver τεσσάρων καναλιών MC10125 της Motorola. Η τυπική κατανάλωση ισχύος είναι της τάξης των 400 mW (χωρίς φορτίο). Οι τυπικοί χρόνοι ανόδου και μετάδοσης είναι 2.5ns και 4,5ns αντίστοιχα. Δέχεται στην είσοδό του διαφορικά σήματα ECL πλάτους

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

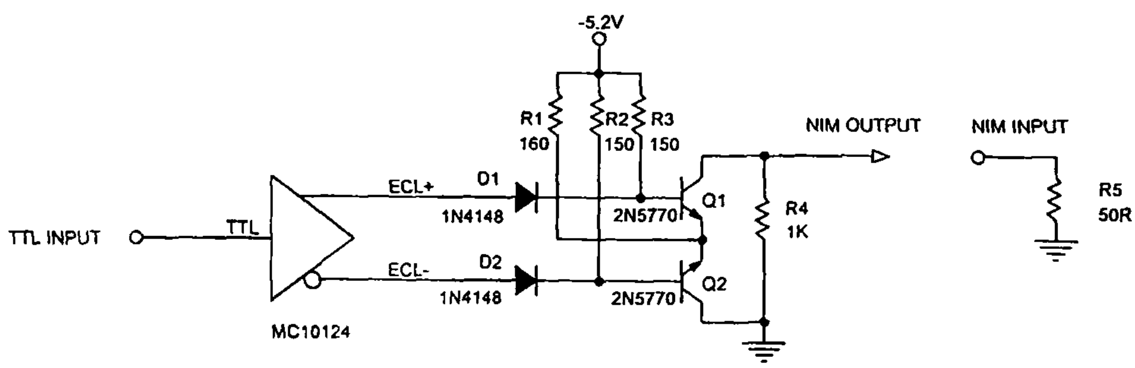
800mV και τα μετατρέπει σε σήματα TTL. Τα σήματα ECL που λαμβάνονται, προσαρμόζονται με φορτίο 100Ω μεταξύ των δύο εισόδων [6].



Σχήμα 2.7 Μετατροπή από ECL σε TTL

2.2.5 TTL σε NIM

Για την μετατροπή από TTL σε NIM, υλοποιήθηκε το κύκλωμα που απεικονίζεται στο Σχ.2.8. Για τη μετατροπή αυτή χρησιμοποιήθηκαν ο line driver MC10124 της Motorola και τα RF transistors 2N5770 της Fairchild Semiconductors. Σε αυτό το σημείο υπενθυμίζεται ότι τα σήματα NIM είναι αρνητικής λογικής και τα λογικά επίπεδα για 0 και 1 είναι 0V και –0.8V, αντίστοιχα. Έτσι, στα άκρα της αντίστασης τερματισμού 50Ω του σήματος NIM η διαφορά δυναμικού θα πρέπει να είναι ίση με –0.8V στην περίπτωση που το σήμα TTL βρίσκεται σε λογικό 1, ενώ στην περίπτωση που το TTL βρίσκεται σε λογικό 0, η διαφορά δυναμικού στα άκρα της αντίστασης τερματισμού θα πρέπει να είναι 0V.



Σχήμα 2.8 Μετατροπή από TTL σε NIM

Η λειτουργία του κυκλώματος είναι η εξής:

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

1η περίπτωση (TTL σε λογικό 0):

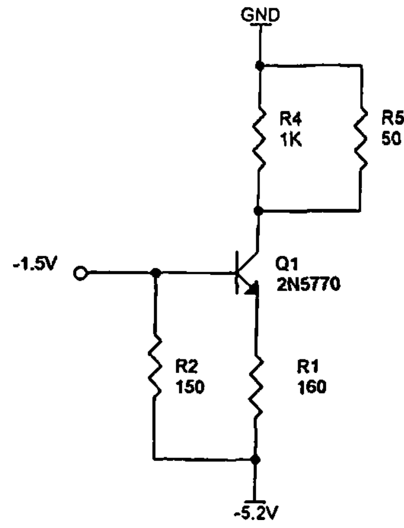
Όταν το TTL βρίσκεται σε λογικό 1, η ECL (ECL+) έξοδος του MC10124 οδηγείται στα -1.6V (λογικό 0 για επίπεδα ECL), ενώ η ανάστροφη έξοδος ECL(ECL-) οδηγείται στα -0.8V (λογικό 1 για επίπεδα ECL). Η τάση στη βάση του transistor Q2 γίνεται -1,5V μετά την πτώση τάσης στη δίοδο D2 και η τάση στον εκπομπό του ίδιου transistor φτάνει τα -2.2V, λόγω της πτώσης τάσης στη δίοδο βάσης-εκπομπού. Η τάση στον εκπομπό του transistor Q1 είναι επίσης -2.2V, καθώς τα δύο σημεία είναι ισοδυναμικά και εφόσον η τάση στη βάση του transistor Q1 είναι ίση με -2.3V, λόγω της πτώσης τάσης στη δίοδο D1, το transistor Q1 δεν άγει και οδηγείται στην αποκοπή. Αυτό έχει σαν αποτέλεσμα να μην διαρρέεται απο ρεύμα η αντίσταση R4, και συνεπώς η έξοδος NIM να οδηγείται μέσω της R4 στη γείωση.

Τελικά, για TTL σε λογικό 0, η έξοδος NIM οδηγείται στα 0V, μια τιμή που αντιστοιχει σε λογικό 0, για το πρότυπο NIM.

2η περίπτωση (TTL σε λογικό 1):

Όταν το TTL βρίσκεται σε λογικό 1, η ECL (ECL+) έξοδος του MC10124 οδηγείται στα
-0.8V (λογικό 1 για επίπεδα ECL), ενώ η ανάστροφη έξοδος ECL (ECL-) οδηγείται στα -1,6V (λογικό 0 για επίπεδα ECL). Η τάση στη βάση του transistor Q1 γίνεται -1,5V μετά την πτώση τάσης στη δίοδο D1 και η τάση στον εκπομπό του ίδιου transistor φτάνει τα -2.2V, λόγω της πτώσης τάσης στη δίοδο βάσης-εκπομπού. Η τάση στον εκπομπό του transistor Q2 είναι επίσης 2.2V, καθώς τα δύο σημεία είναι ισοδυναμικά και εφόσον η τάση στη βάση του transistor Q2 είναι ίση με –2.3V, λόγω της πτώσης τάσης στη δίοδο D2, το transistor Q2 δεν άγει και οδηγείται στην αποκοπή. Το ισοδύναμο κύκλωμα που προκύπτει σε αυτήν την περίπτωση απεικονίζεται στο Σχ.2.9

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 2.9 Ισοδύναμο κύκλωμα για TTL σε λογικό 1

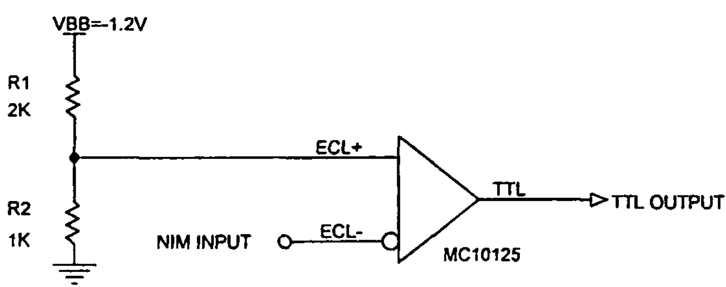
Το transistor πολώνεται κατάλληλα και η τάση V_{CE} φτάνει τα 1,7V. Η τάση του συλλέκτη του transistor Q1, που είναι ουσιαστικά η έξοδος NIM του κυκλώματος υπολογίζεται ως εξής:

$$V_{NIM} = V_{CQ1} - V_{CE} = (-5.2V + 1,7) \cdot \frac{(R4 // R5)}{(R4 // R5) + R1} = -3,5 \cdot \frac{(1000 // 50)}{(1000 // 50) + 160} = -0.802V$$

Τελικά, για TTL σε λογικό 1, η έξοδος NIM οδηγείται στα –0.8V, μια τιμή που αντιστοιχει σε λογικό 1, για το πρότυπο NIM.

2.2.6 NIM σε TTL

Για την μετατροπή από NIM σε TTL, υλοποιήθηκε το κύκλωμα που απεικονίζεται στο Σχ.2.10. Για τη μετατροπή αυτή χρησιμοποιήθηκαν ο line receiver MC10125 της Motorola. Το σήμα NIM εισάγεται στην συμπληρωματική είσοδο του MC10125, ενώ στην άλλη είσοδο εφαρμόζεται τάση της τάξης των -0.4V. Η τάση αυτή προκύπτει με τη βοήθεια ενός διαιρέτη τάσης (αντιστάσεις R1 και R2) και του pin V_{BB} του MC10125, το οποίο παρέχει τάση -1,2V.



Σχήμα 2.10 Μετατροπή από NIM σε TTL

Η λειτουργία του κυκλώματος βασίζεται στη λογική ότι το MC10125 δίνει λογικό 1 στην έξοδο όταν η τάση στη συμπληρωματική είσοδό του (ECL-) είναι μικρότερη της άλλης εισόδου (ECL+), και λογικό 0 όταν συμβαίνει το αντίστροφο. Τα λογικά επίπεδα της εξόδου του MC10125 υπενθυμίζεται ότι είναι TTL.

Έτσι λοιπόν, όταν το σήμα NIM βρίσκεται σε λογικό 1, στην συμπληρωματική είσοδο του MC10125 εφαρμόζεται τάση -0,8V. Επειδή όμως η τάση στην άλλη είσοδο του MC10125 είναι υψηλότερη (-0,4V), η έξοδος οδηγείται σε λογικό 1. Στην αντίθετη περίπτωση, όταν το σήμα NIM βρίσκεται σε λογικό 0, στην συμπληρωματική είσοδο του MC10125 εφαρμόζεται τάση 0V. Επειδή όμως η τάση στην άλλη είσοδο του MC10125 είναι χαμηλότερη (-0,4V), η έξοδος οδηγείται σε λογικό 0.

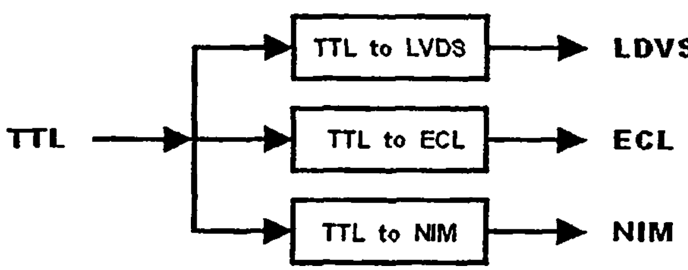
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.3 Σχεδιασμός – Υλοποίηση

Στις παρακάτω παραγράφους δίνονται τα κυκλωματικά σχέδια του μετατροπέα ψηφιακών σημάτων. Ο μετατροπέας χωρίζεται σε δύο βασικές υπομονάδες. Αυτές είναι η υπομονάδα μετατροπής των σημάτων TTL σε ECL, NIM και LVDS, και η υπομονάδα μετατροπής των σημάτων ECL, NIM και LVDS σε TTL. Σημειώνεται ότι η δύο αυτές υπομονάδες είναι τελείως αυτόνομες, και μπορούν να χρησιμοποιηθούν και ξεχωριστά. Στην παράγραφο 2.3.3 εξηγείται η λογική με την οποία συνεργάζονται αυτές οι δύο αυτόνομες μονάδες.

2.3.1 Υπομονάδα μετατροπής TTL σε ECL-NIM-LVDS

Η υπομονάδα μετατροπής TTL σε ECL-NIM-LVDS, είναι η πρώτη από τις δύο αυτόνομες υπομονάδες, που αναφέρθηκαν παραπάνω. Αποτελείται από τρία τμήματα, ένα για κάθε μετατροπή, τα οποία έχουν την TTL είσοδό τους κοινή, κάτι που σημαίνει ότι γίνεται ταυτόχρονη μετατροπή του TTL στα τρία άλλα σήματα. Το διάγραμμα της υπομονάδας απεικονίζεται στο Σχ.2.11

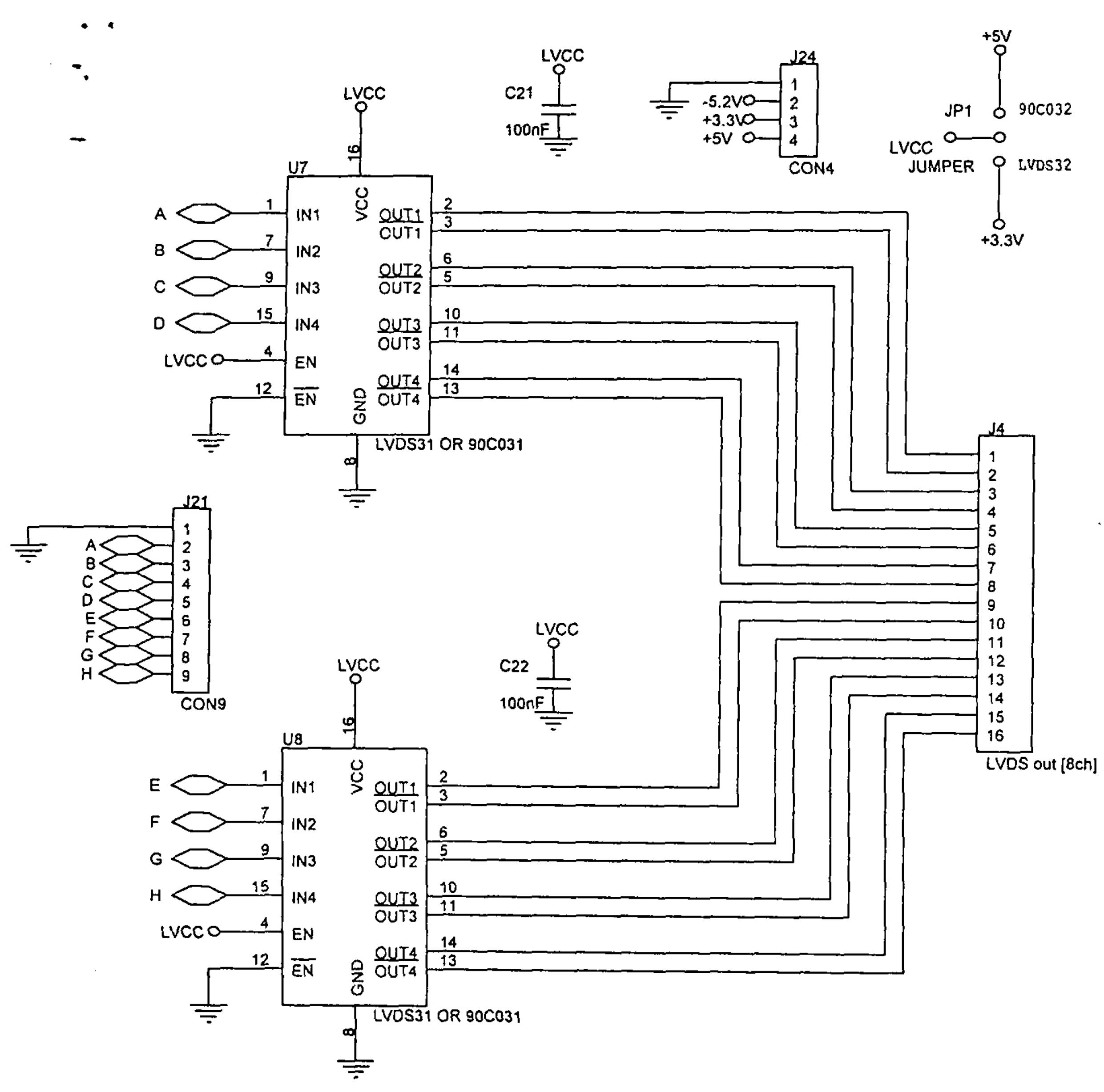


Σχήμα 2.11 Υπομονάδα μετατροπής TTL σε ECL-NIM-LVDS

Στα σχήματα που ακολουθούν (Σχ.2.12, Σχ.2.13 και Σχ.2.14), δίνονται τα πλήρη κυκλωματικά σχέδια της εν λόγω υπομονάδας μετατροπής.

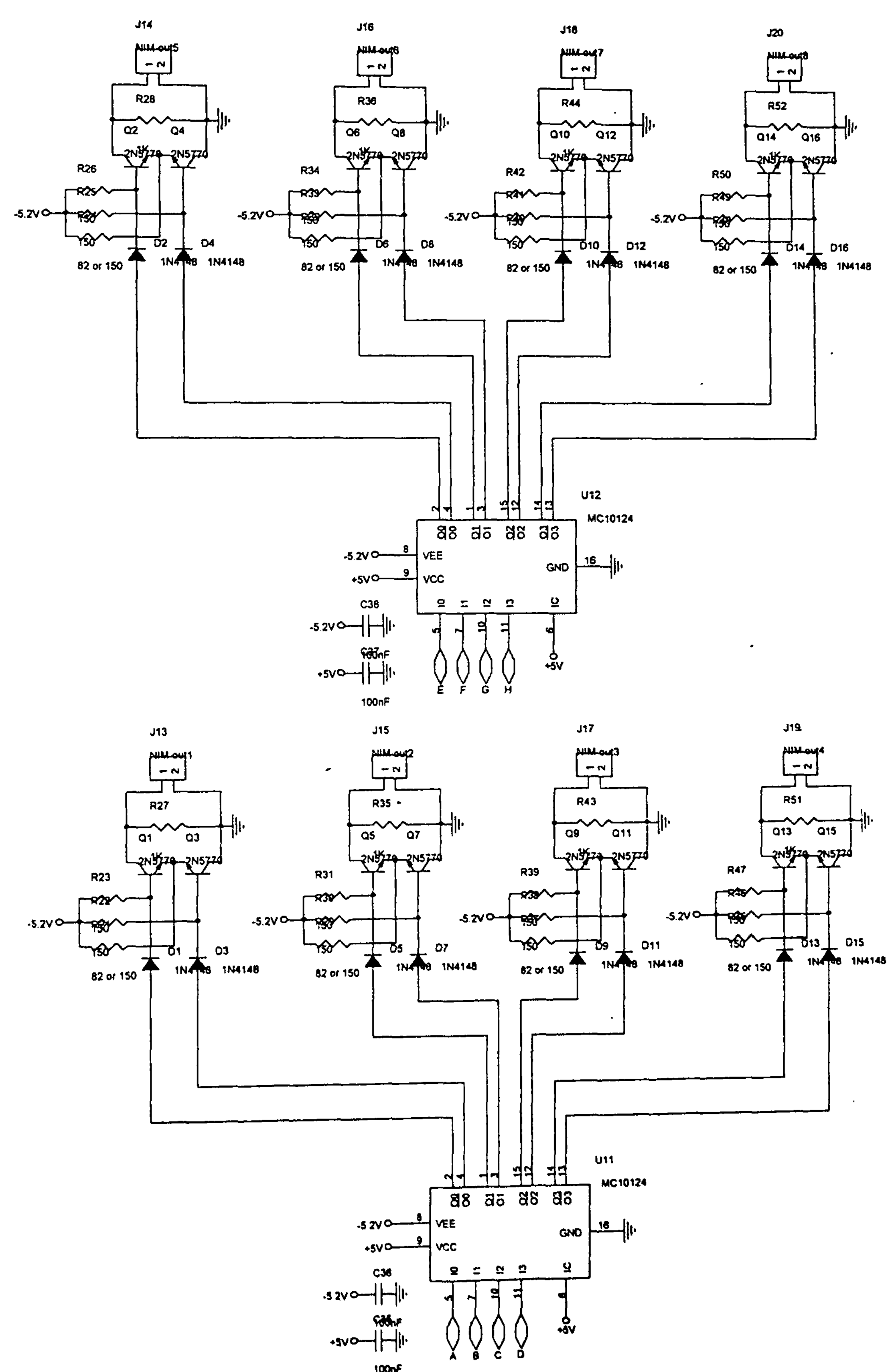
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Σχήμα 2.12 Μετατροπή οκτώ καναλιών TTL σε ECL



Σχήμα 2.13 Μετατροπή οκτώ καναλιών TTL σε LVDS

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

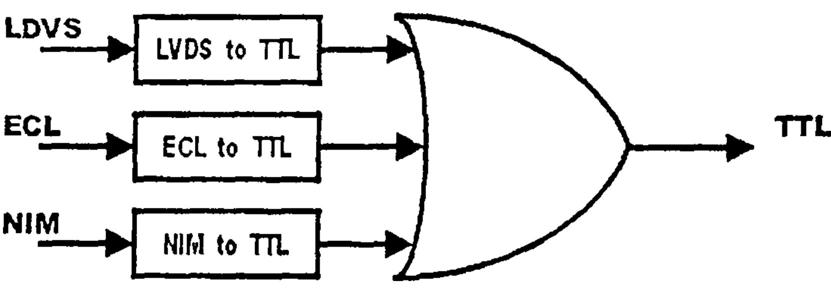


Σχήμα 2.14 Μετατροπή οκτώ καναλιών TTL σε NIM

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.3.2 Μονάδα μετατροπής ECL-NIM-LVDS σε TTL

Η υπομονάδα μετατροπής TTL σε ECL-NIM-LVDS, είναι η δεύτερη από τις αυτόνομες υπομονάδες, που αναφέρθηκαν παραπάνω. Αποτελείται από τέσσερα τμήματα, τρία για τις αντίστοιχες μετατροπές, και ένα τέταρτο για την TTL βαθμίδα εξόδου. Το διάγραμμα της υπομονάδας απεικονίζεται στο Σχ.2.15



Σχήμα 2.15 Υπομονάδα μετατροπής ECL-NIM-LVDS σε TTL

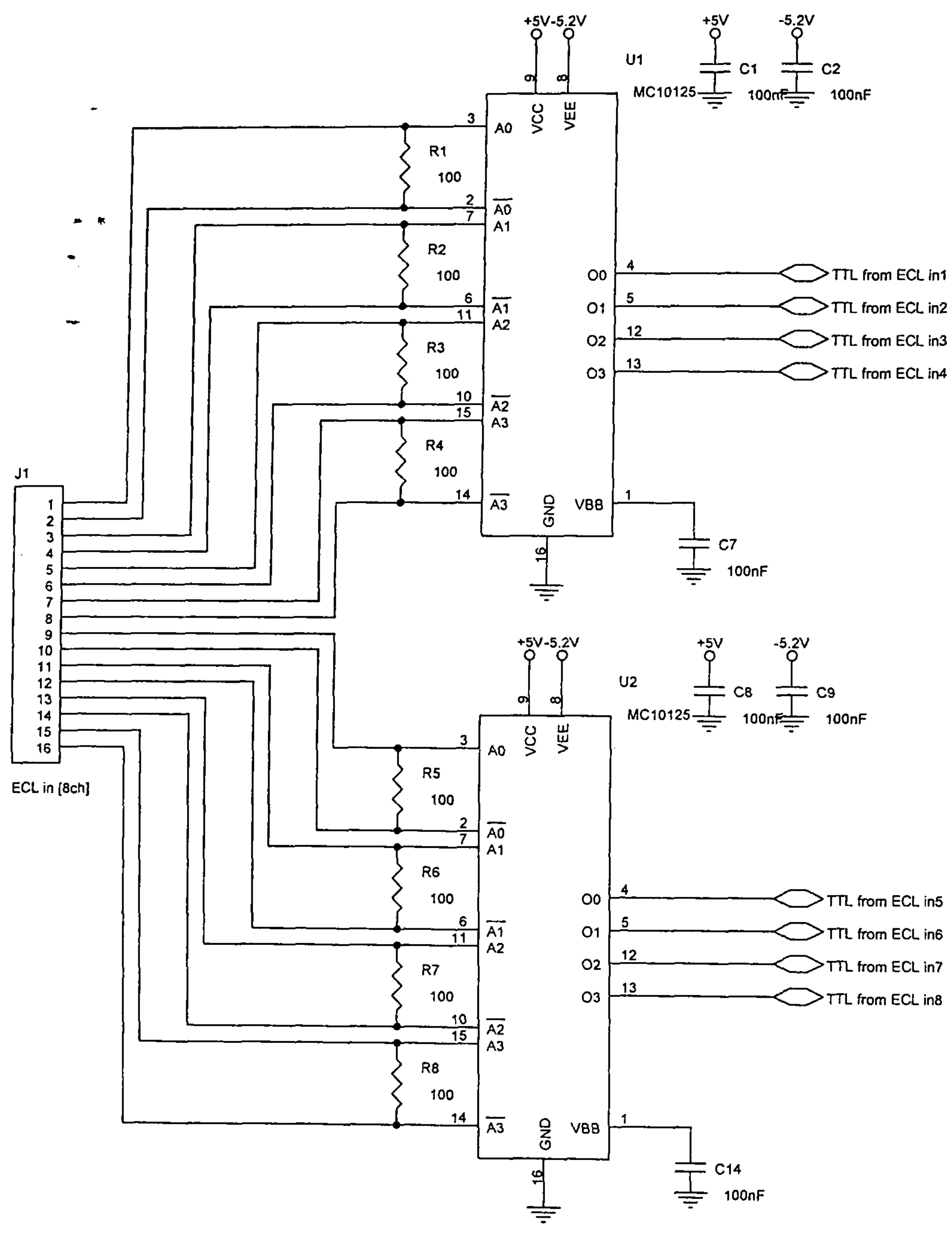
Όπως φαίνεται και στο διάγραμμα, μετά τη βαθμίδα μετατροπής, ακολουθεί η βαθμίδα εξόδου, στην οποία γίνεται λογικό OR μεταξύ των TTL σημάτων που προήλθαν από τα αντίστοιχα σήματα, σε κάθε ένα απο τα κανάλια του μετατροπέα. Αυτό γίνεται για να υπάρχει μία κοινή TTL έξοδος στο σύστημα, και όχι τρεις ξεχωριστές, μία για κάθε σήμα εισόδου. Έτσι, αντί να υπάρχει μία έξοδος TTL που προέρχεται από ECL, μία από NIM και μία από LVDS, υπάρχει μία και μόνο TTL έξοδος. Βέβαια, για να γίνει αυτό υπάρχουν δύο προϋποθέσεις: η πρώτη είναι να συνδέεται ένα μόνο σήμα σε κάθε καθε κανάλι εισόδου (μόνο LVDS ή μόνο ECL ή μόνο NIM), και η δεύτερη είναι να έχουν ως έξοδο λογικό 0 όλες οι ασύνδετες είσοδοι LVDS, ECL και NIM. Η πρώτη προϋπόθεση έγκειται στο χρήστη, ενώ η δεύτερη καλύπτεται από τη σχεδίαση του κυκλώματος. Συνεπώς, εάν εφαρμοστεί ένα σήμα TTL από την έξοδο κάποιας βαθμίδας μετατροπής στη μια είσοδο της πύλης OR, η οποία αντιστοιχεί σε κάθε κανάλι, ενώ οι αλλες δύο είσοδοί της βρίσκονται σε λογικό 0, τότε το σήμα εισόδου οδηγείται στην έξοδο, με μια όμως χρονική καθυστέρηση ίση με το propagation delay του λογικού κυκλώματος. Σε περίπτωση που συνδεθούν

ΒΙΒΛΙΟΘΗΚΗ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΙΩΑΝΝΙΝΩΝ

από λάθος δύο ή και τρία σήματα εισόδου στο ίδιο κανάλι, προφανώς το TTL σήμα εξόδου δεν θα είναι το αναμενόμενο. Ένα άλλο πλεονέκτημα αυτής της μεθόδου, εκτός των προαναφερθέντων είναι και η ομοιομορφία στα TTL σήματα εξόδου, ανεξάρτητα από την προέλευσή τους. Έτσι το TTL σήμα εξόδου έχει ίδια χαρακτηριστικά, ανεξάρτητα άν προήλθε από LVDS ή από ECL ή από NIM[5]. Τα χαρακτηριστικά του σήματος εξόδου εξαρτώνται μόνο από την υλοποίηση της αθμίδας εξόδου. Για αυτόν τον λόγο επιλέχθηκε η χρήση του CPLD XC9536-1 της εταιρίας Xilinx, το οποίο παρουσιάζει εξαιρετικούς χρόνους ανόδου σε σχέση με τις οικογένειες ολοκληρωμένων κυκλωμάτων TTL. Στα σχήματα που ακολουθούν (Σχ.2.16, Σχ.2.17, Σχ.2.18 και Σχ.2.19), δίνονται τα πλήρη κυκλωματικά σχέδια της εν λόγω υπομονάδας μετατροπής.

[5] *Σημειώνεται ότι οι TTL δεν πρέπει να τερματιστούν στα 50Ω.*

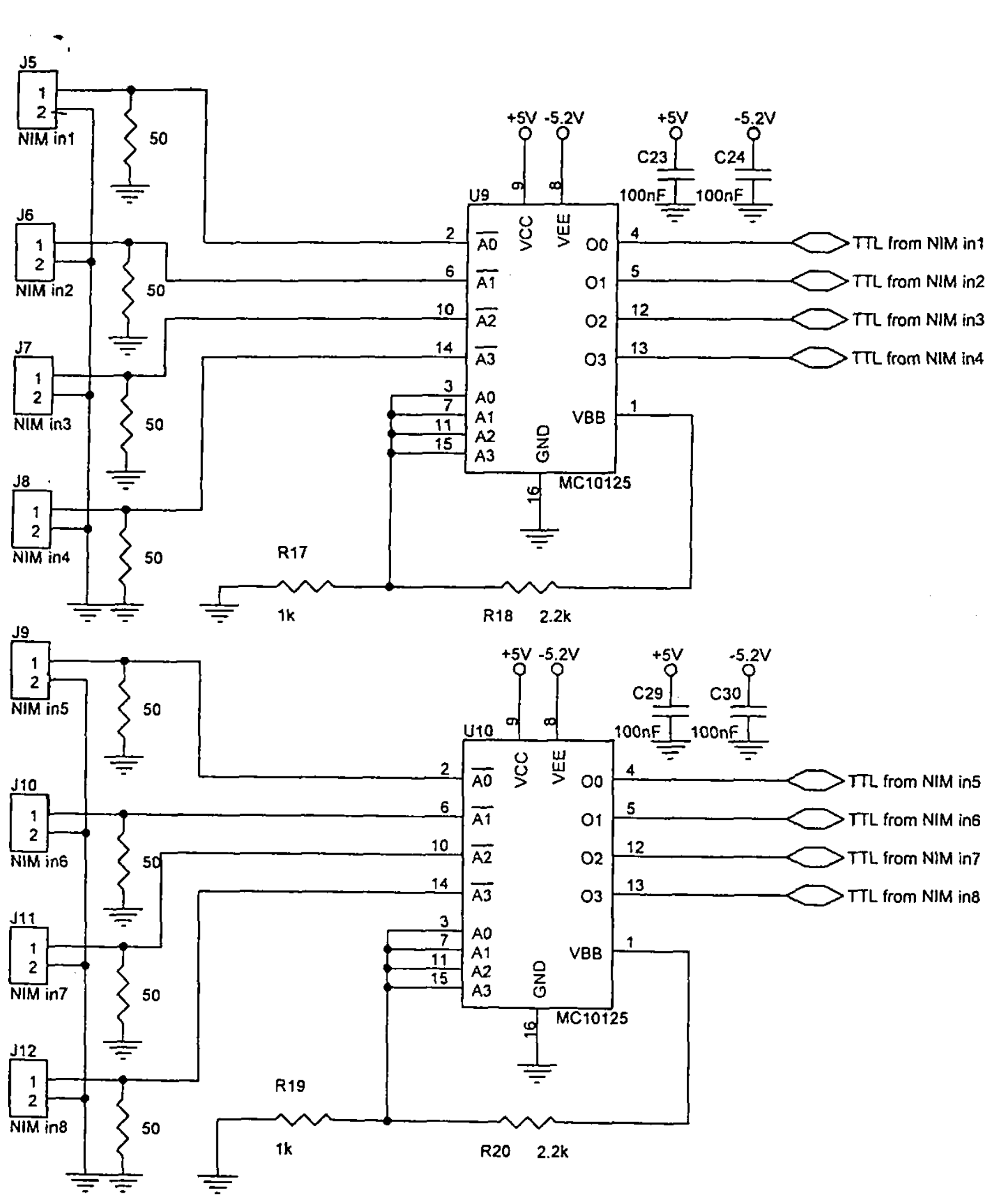
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 2.16 Μετατροπή οκτώ καναλιών ECL σε TTL

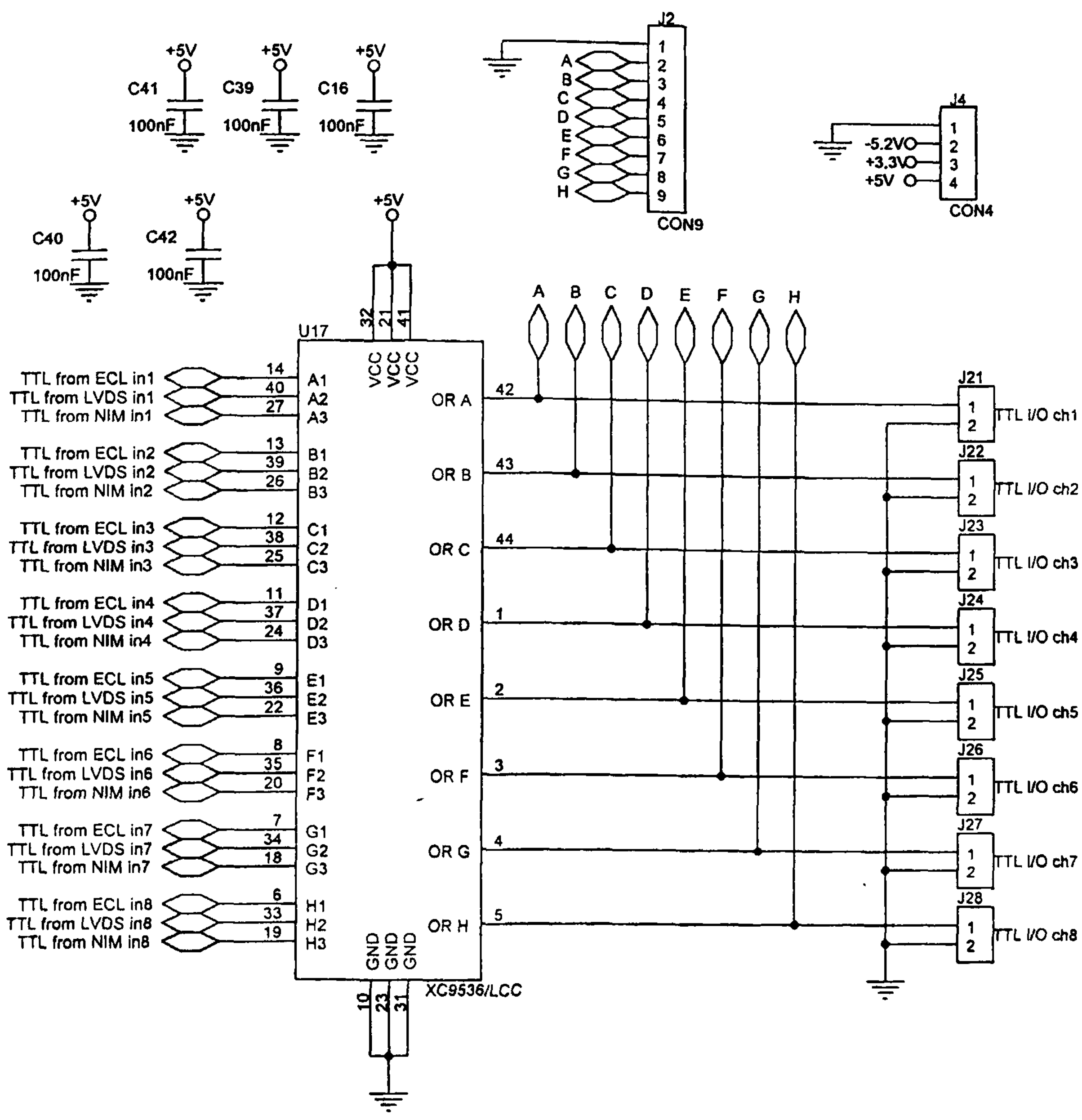
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Σχήμα 2.17 Μετατροπή οκτώ καναλιών LVDS σε TTL



Σχήμα 2.18 Μετατροπή οκτώ καναλιών NIM σε TTL

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



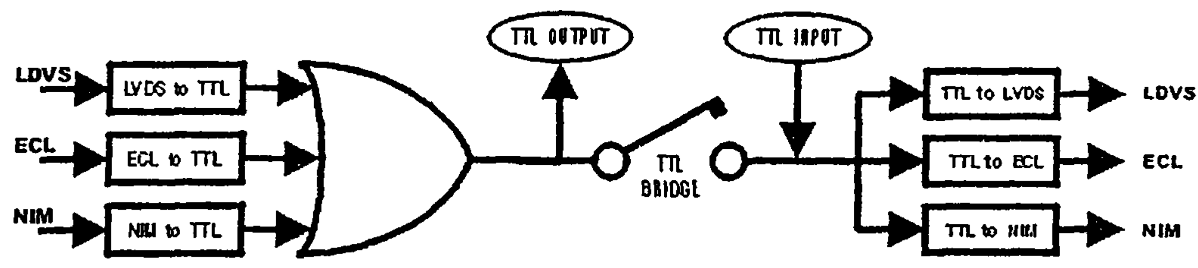
Σχήμα 2.19 Βαθμίδα εξόδου TTL[6]

[6] *Χρησιμοποιήθηκαν pull down αντιστάσεις 10K στα –5.2V, στις μη συμπληρωματικές LVDS εισόδους, οι οποίες δε φαίνονται στο σχέδιο, έτσι ώστε οι ασύνδετες LVDS είσοδοι να δίνουν λογικό 0 στις αντίστοιχες TTL εξόδους τους.*

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.3.3 Τελική υλοποίηση του μετατροπέα ψηφιακών σηματων

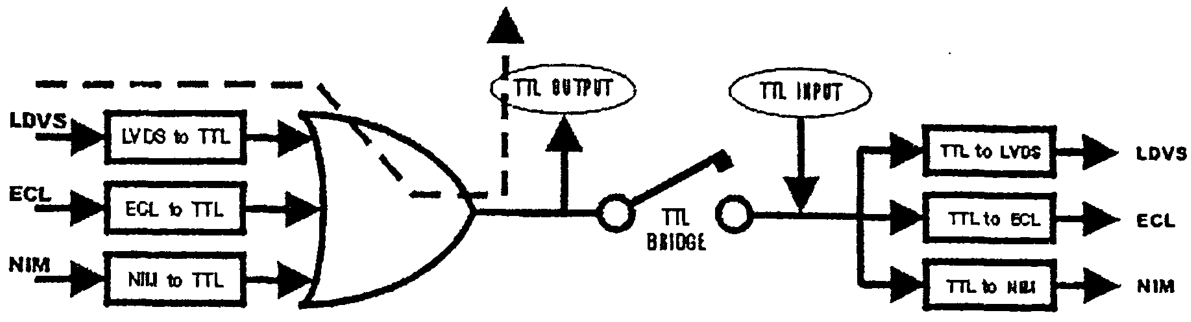
Η υλοποίηση του μετατροπέα βασίζεται στο διάγραμμα του Σχ. 2.20, στο οποίο φαίνεται ο τρόπος που συνεργάζονται οι δύο υπομονάδες μετατροπής που παρουσιάστηκαν παραπάνω. Η TTL έξοδος της μονάδας μετατροπής από ECL-NIM-LVDS οδηγείται μέσω ενός μεταγωγού διακόπτη, όταν αυτός είναι κλειστός, στην TTL είσοδο της μονάδας μετατροπής σε ECL-NIM-LVDS, ενώ στη αντίθετη περίπτωση οι TTL είσοδοι και έξοδοι είναι τελείως απομονωμένες, και δύο μονάδες μετατροπής λειτουργούν τελείως αυτόνομα.



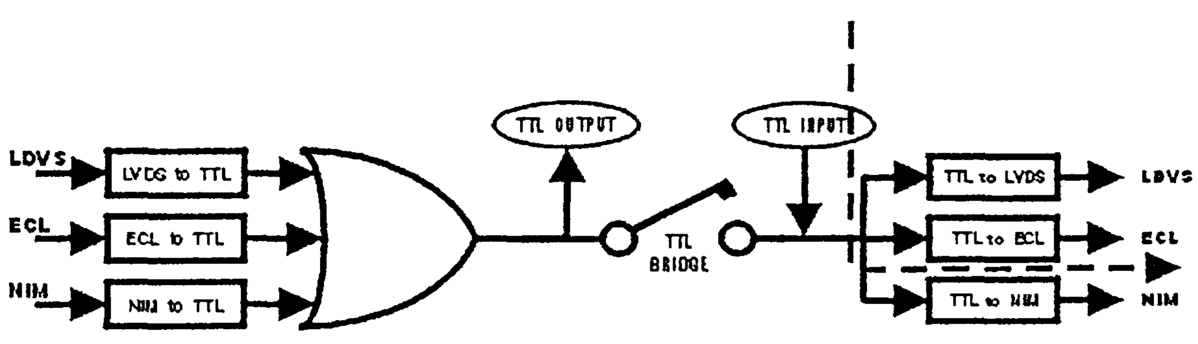
Σχήμα 2.20 Τελική υλοποίηση του μετατροπέα

Στα Σχ.2.21, Σχ.2.22 και Σχ.2.23 που ακολουθούν δίνονται τρία παραδείγματα λειτουργίας του μετατροπέα, στα οποία διακρίνεται η διαδρομή που ακολουθεί το σήμα, καθώς και η θέση του μεταγωγού διακόπτη. Στο Σχ.2.21 απεικονίζεται η μετατροπή από LVDS σε TTL, στο Σχ.2.22 η μετατροπή από TTL σε ECL, και στο Σχ.2.23 η μετατροπή από LVDS σε NIM. Στην τρίτη περίπτωση οι δύο υπομονάδες συνεργάζονται ενώ στις αλλες δύο λειτουργούν τελείως αυτόνομα.

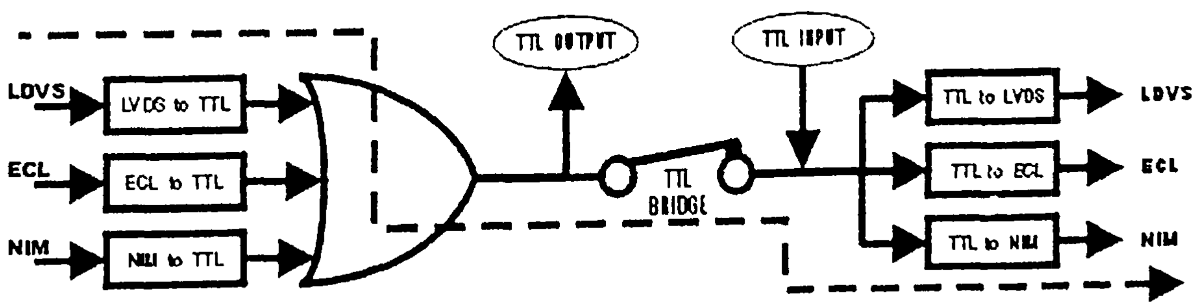
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Σχήμα 2.21 LVDS σε TTL



Σχήμα 2.22 TTL σε ECL



Σχήμα 2.23 LVDS σε NIM

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.4 Μετρήσεις

Στις παρακάτω εικόνες 1 έως και 16 φαίνονται οι μετρήσεις (plots) που πάρθηκαν με τον παλμογράφο TDS684B της εταιρίας Tektronix. Στις πρώτες δώδεκα εικόνες καλύπτονται όλοι οι δυνατοί συνδυασμοί λειτουργίας του μετατροπέα σε συχνότητα 10MHz, ενώ στις επόμενες τέσσερις δίνονται τα σήματα εξόδου στα 40MHz, με τους αντίστοιχους χρόνους ανόδου.

2.5 Αποτίμηση της μονάδας - Συμπεράσματα

Η παρούσα μονάδα μετατροπής ψηφιακών σημάτων TTL, ECL, NIM και LVDS καλύπτει απόλυτα τις απαιτήσεις για πειράματα ΦΥΕ. Πιο συγκεκριμένα, οι προδιαγραφές της μονάδας είναι οι παρακάτω

ΠΡΟΔΙΑΓΡΑΦΕΣ

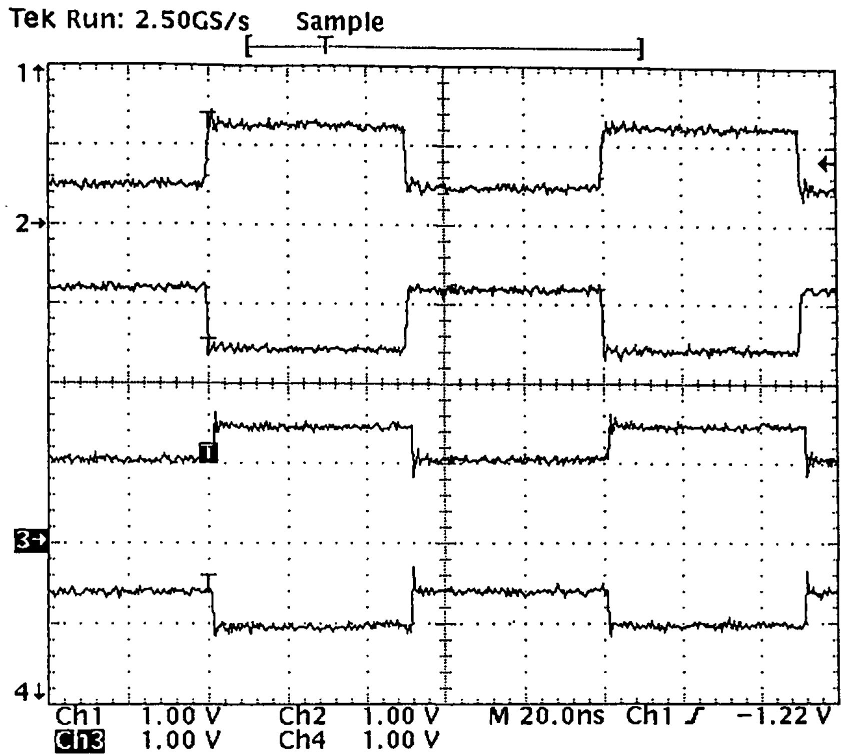
- Η μέγιστη προτεινόμενη συχνότητα λειτουργίας του μετατροπέα είναι τα 60MHz.
- Η μέγιστη καθυστέρηση μετάδοσης (propagation delay) είναι ίση με 25ns
- Τυπικές τιμές χρόνων ανόδου για σήματα εξόδου TTL, ECL, NIM και LVDS είναι αντίστοιχα 3ns, 1.5ns, 1.5ns και 1ns.
- Τα ψηφιακά σήματα εισόδου θα πρέπει να συμφωνούν με τα αντίστοιχα πρότυπα
- Οι TTL έξοδοι δεν πρέπει να τερματίζονται στα 50Ω

Ο Μετατροπέας σχεδιάστηκε σύμφωνα με το πρότυπο NIM [2], έτσι ώστε να προσαρμόζεται σε οποιοδήποτε NIM Crate. Μπορεί όμως να χρησιμοποιηθεί και αυτόνομα, με τη χρήση εξωτερικού τροφοδοτικού. Οι ακροδέκτες του NIM connector που χρησιμοποιούνται είναι τα +12V, -12V και η γείωση.

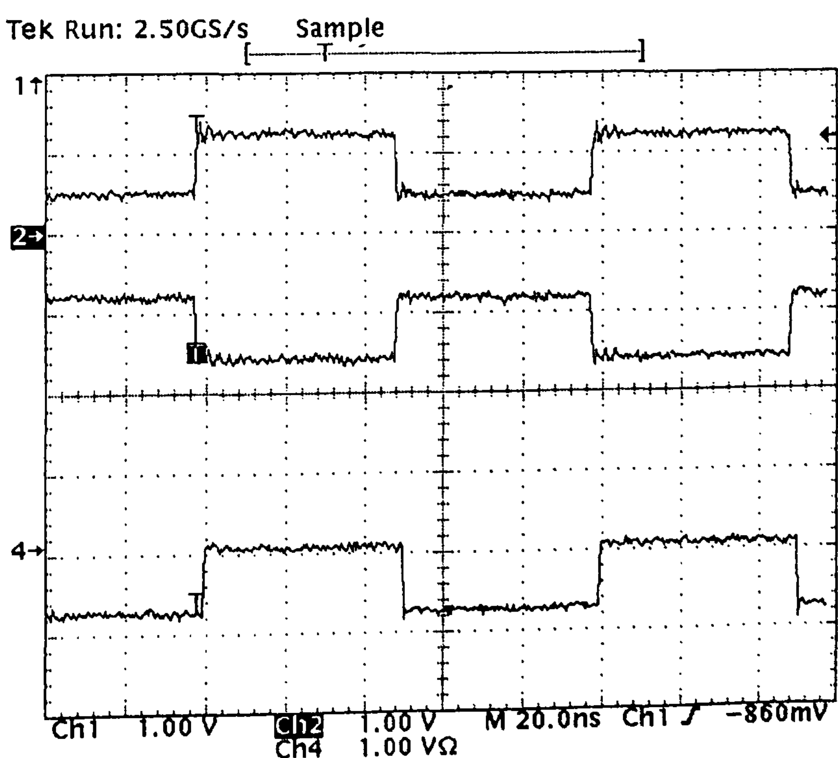
Σημειώνεται ότι η μονάδα κατασκευάστηκε εξ ολοκλήρου στο Εργαστήριο Φυσικής Υψηλών Ενεργειών του πανεπιστημίου Ιωαννίνων κάτι που σημαίνει ότι τα τυπωμένα κυκλώματα καθώς και το περίβλημα υλοποιήθηκαν αποκλειστικά με τον εξοπλισμό του εργαστηρίου.

Μια πιθανή μελλοντική αναβάθμιση της μονάδας μετατροπής θα ήταν η ανάπτυξη της με ενδιάμεσο κρίκο αυτή τη φορά τη βαθμίδα ECL, με σκοπό την επίτευξη υψηλότερης συχνότητας λειτουργίας.

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

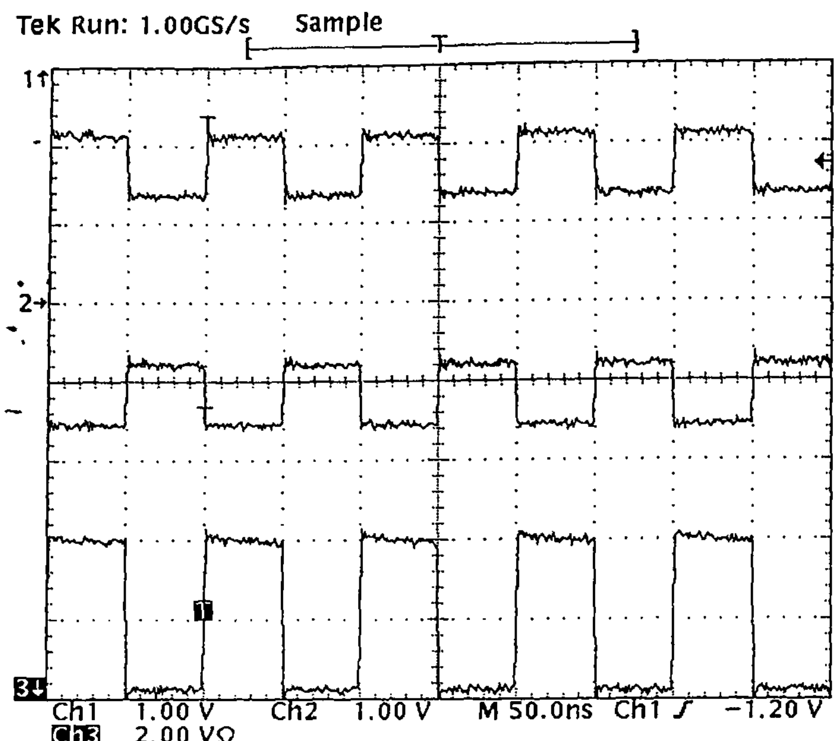


Εικόνα 1. ECL→LVDS @10MHz

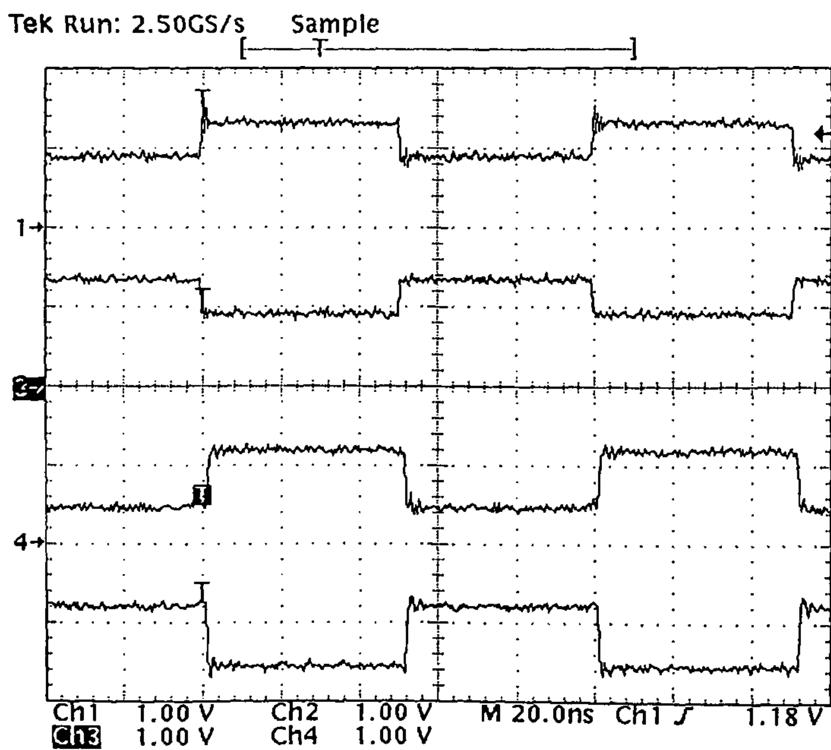


Εικόνα 2. ECL→NIM @10MHz

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

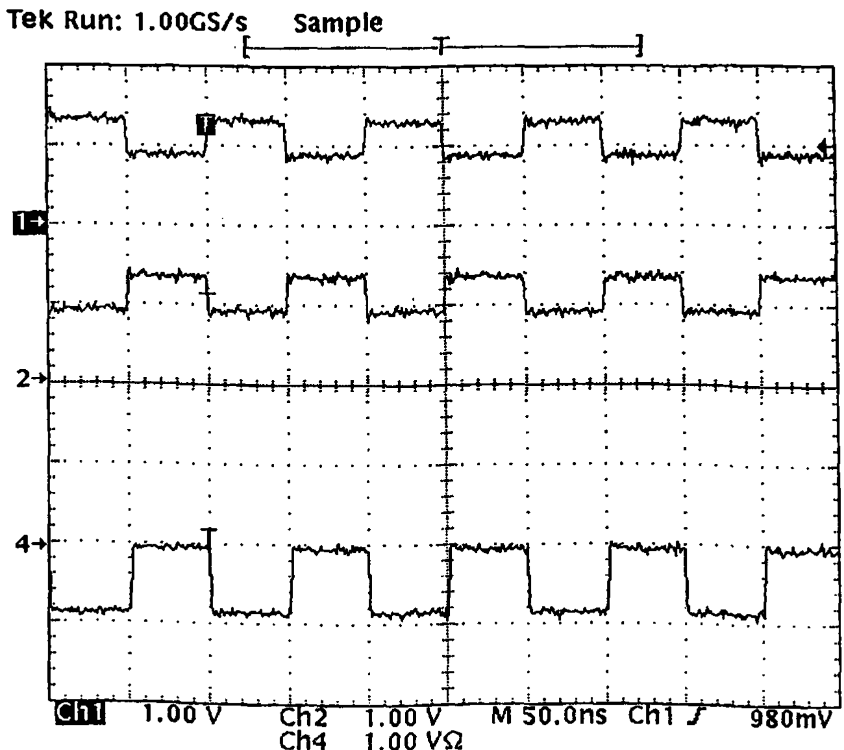


Εικόνα 3. ECL→TTL @10MHz

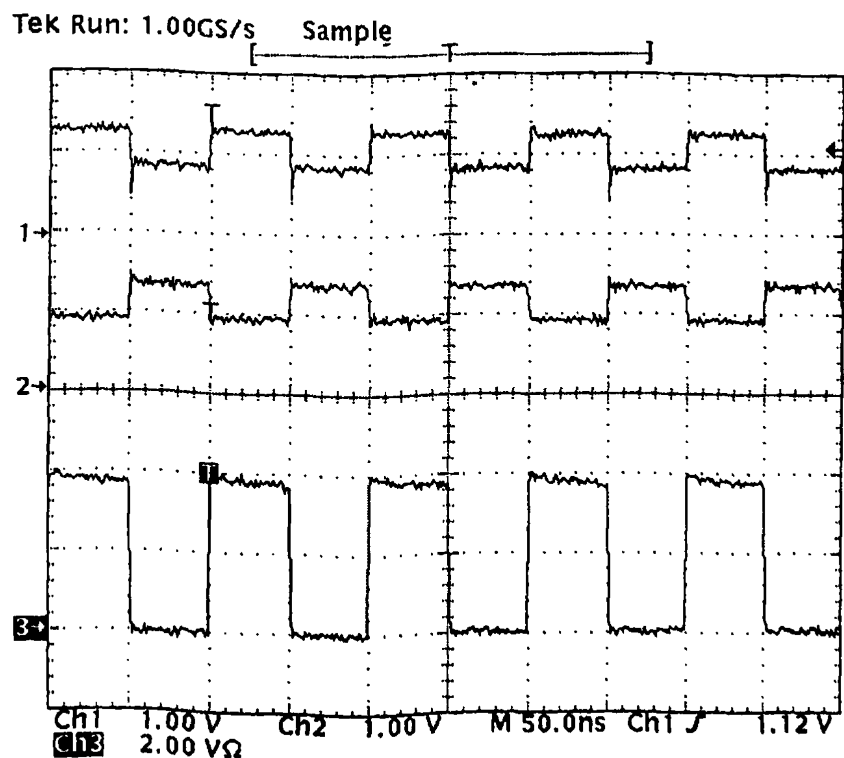


Εικόνα 4. LVDS→ECL @10MHz

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

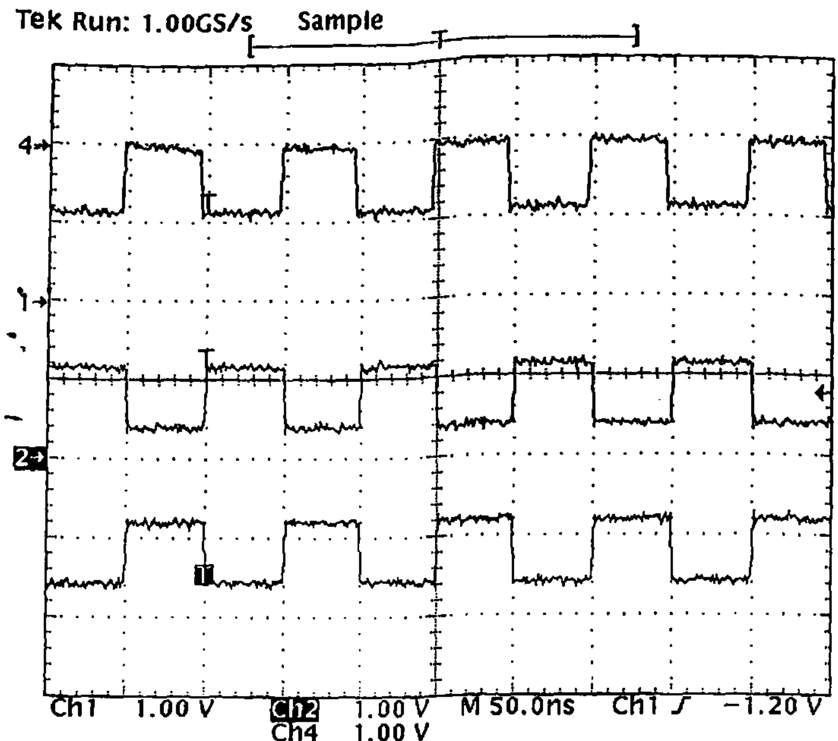


Εικόνα 5. LVDS→NIM @10MHz

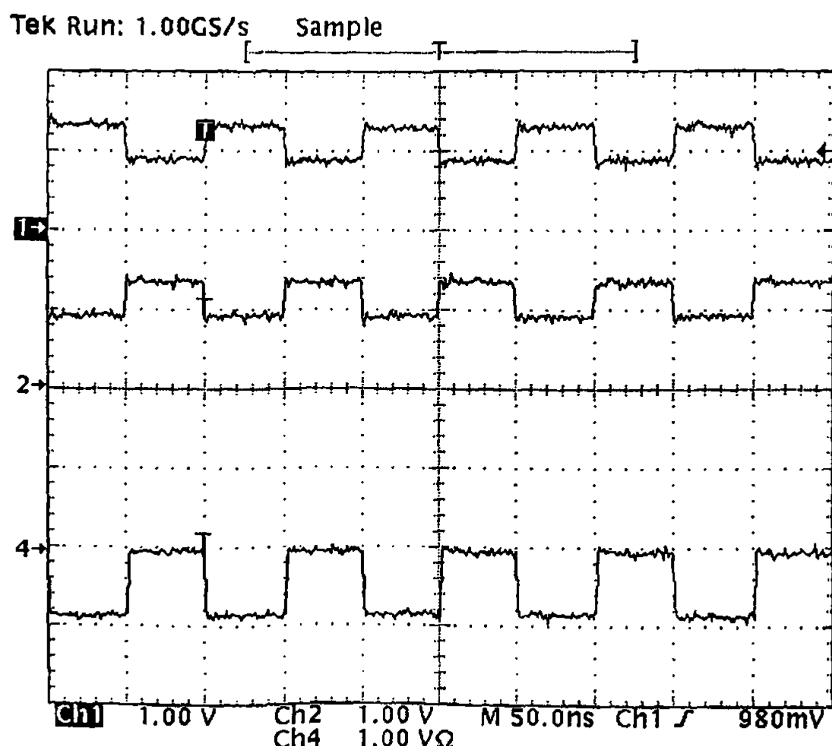


Εικόνα 6. LVDS→TTL @10MHz

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

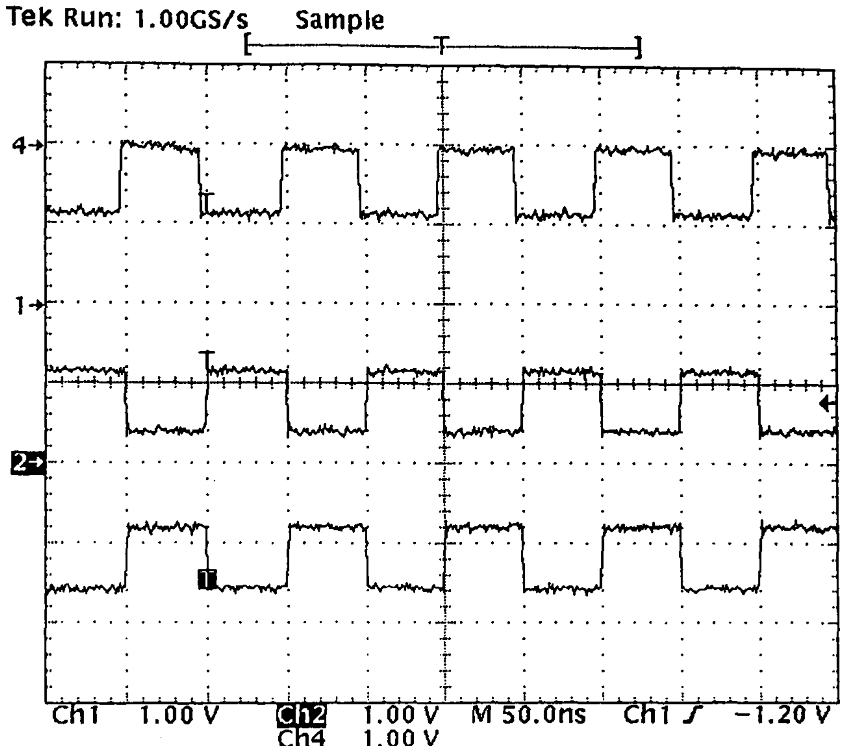


Εικόνα 7. NIM→ECL @10MHz

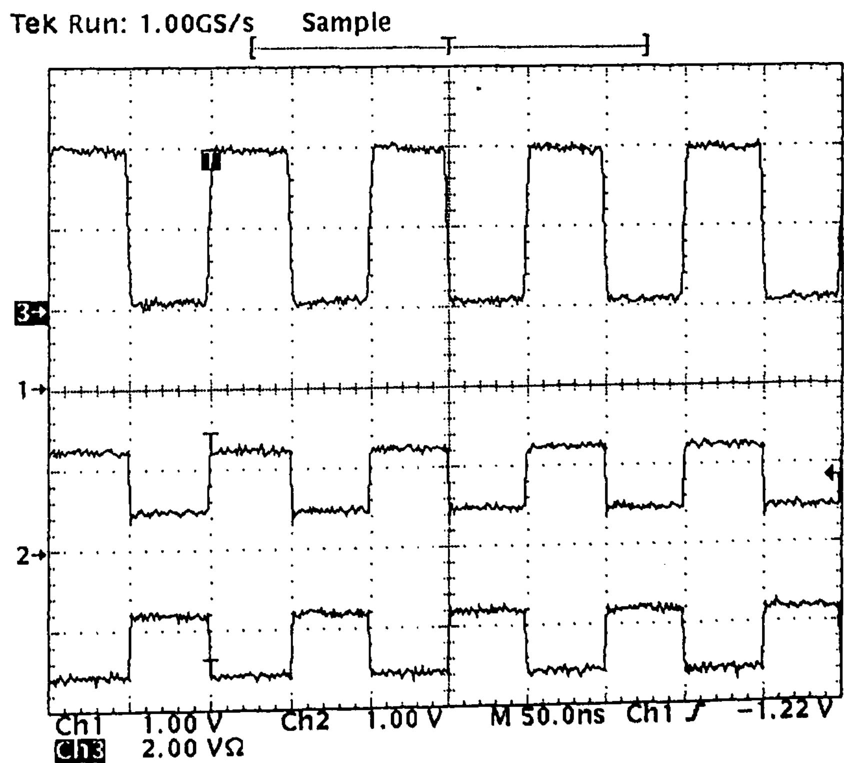


Εικόνα 8. LVDS→NIM @10MHz

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

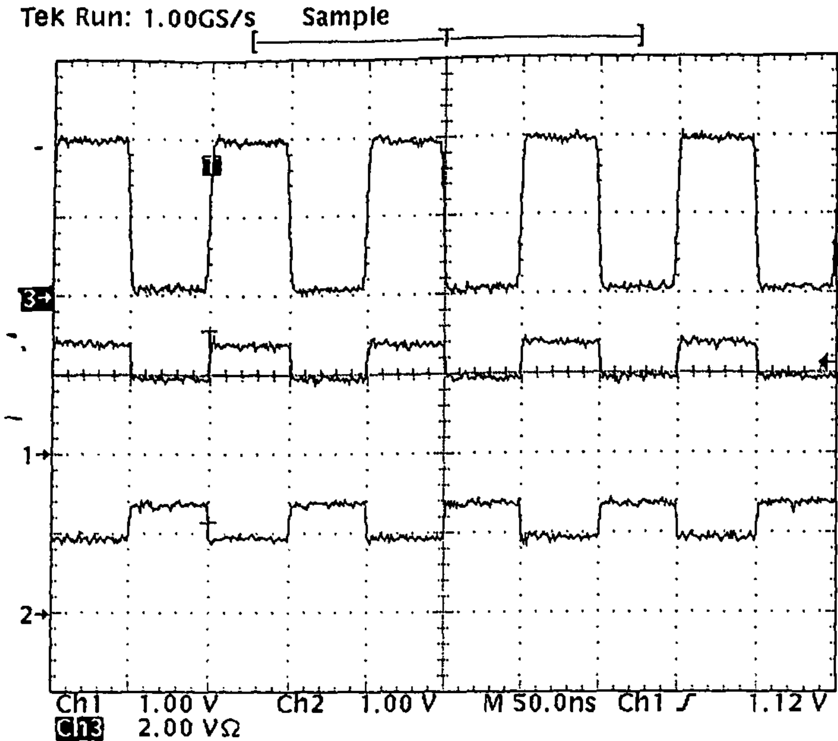


Εικόνα 9. NIM→ECL @10MHz

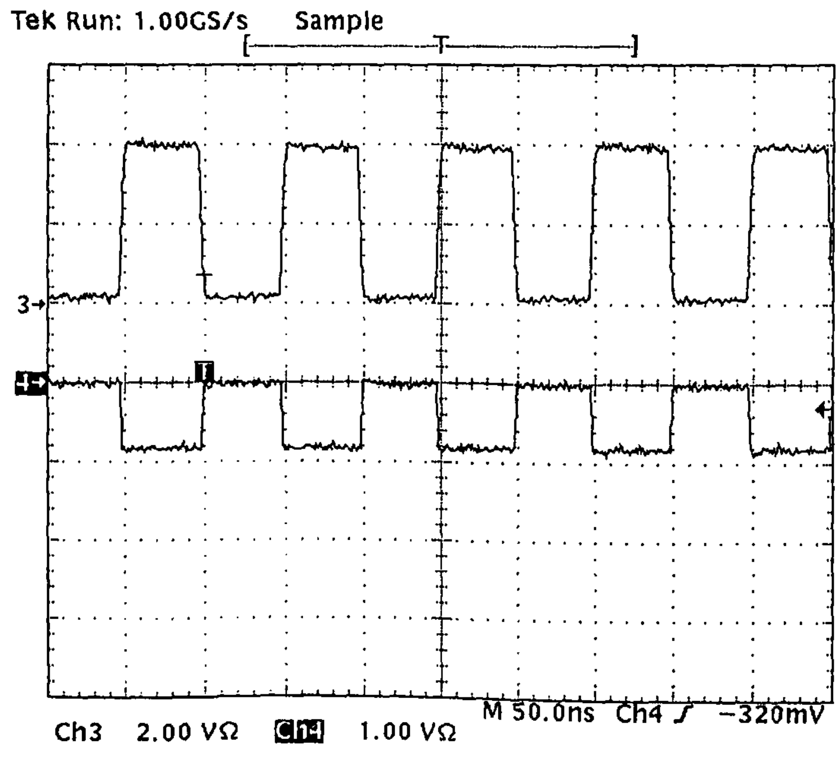


Εικόνα 10. TTL→ECL @10MHz

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

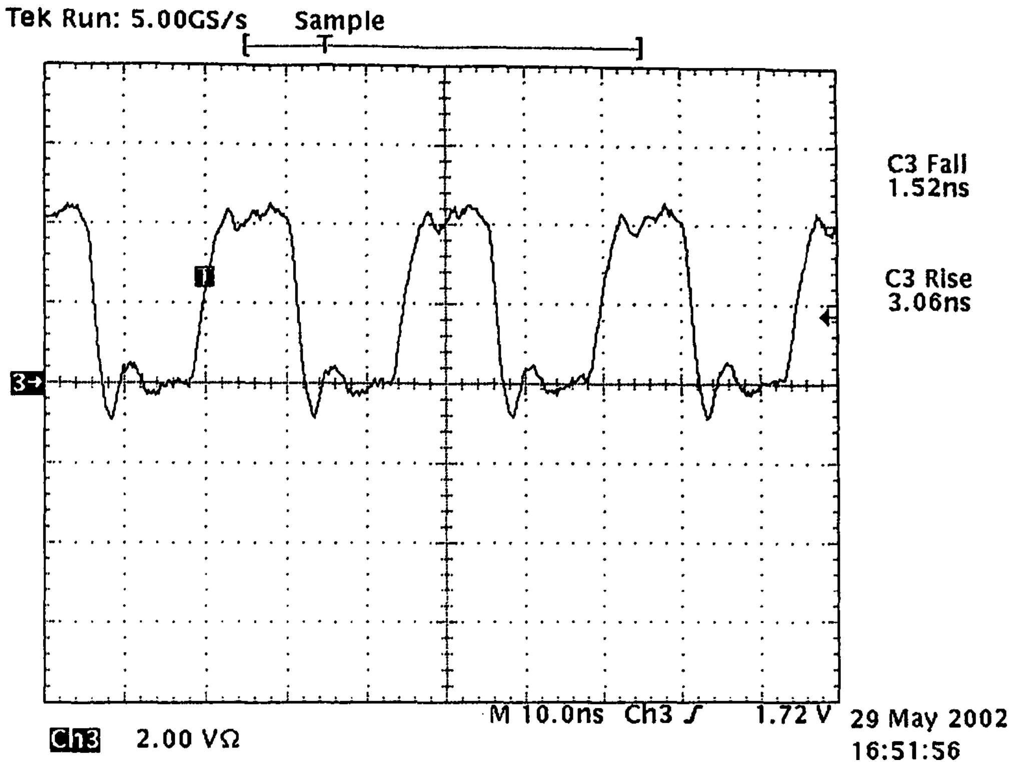


Εικόνα 11. TTL→LVDS @10MHz

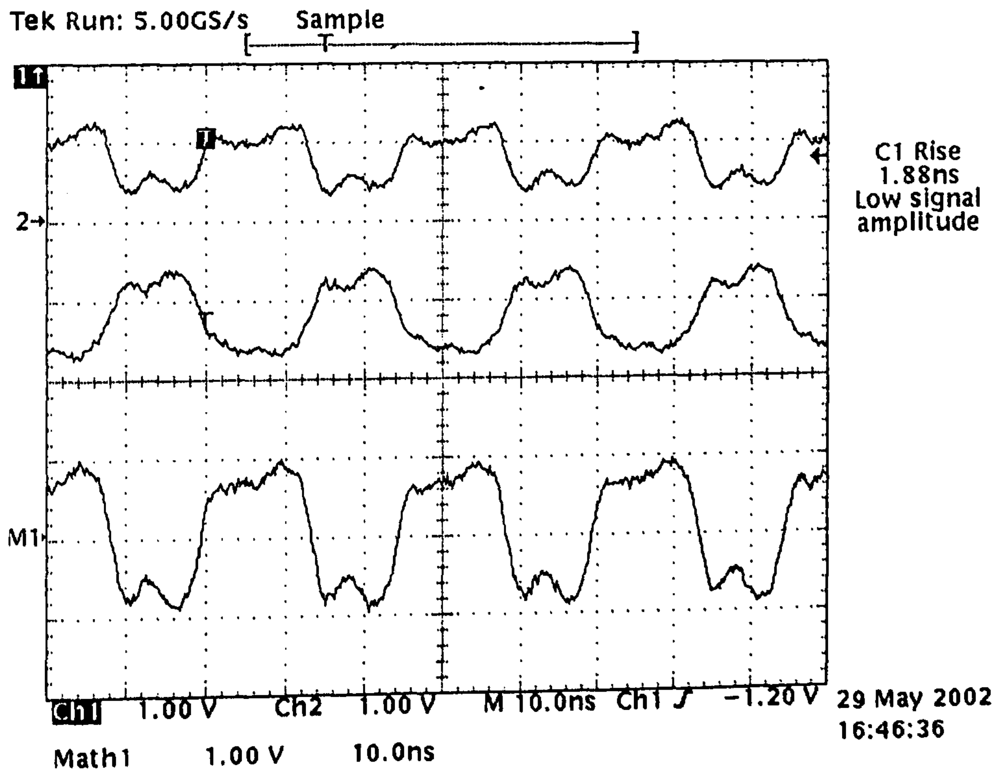


Εικόνα 12. TTL→NIM @10MHz

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

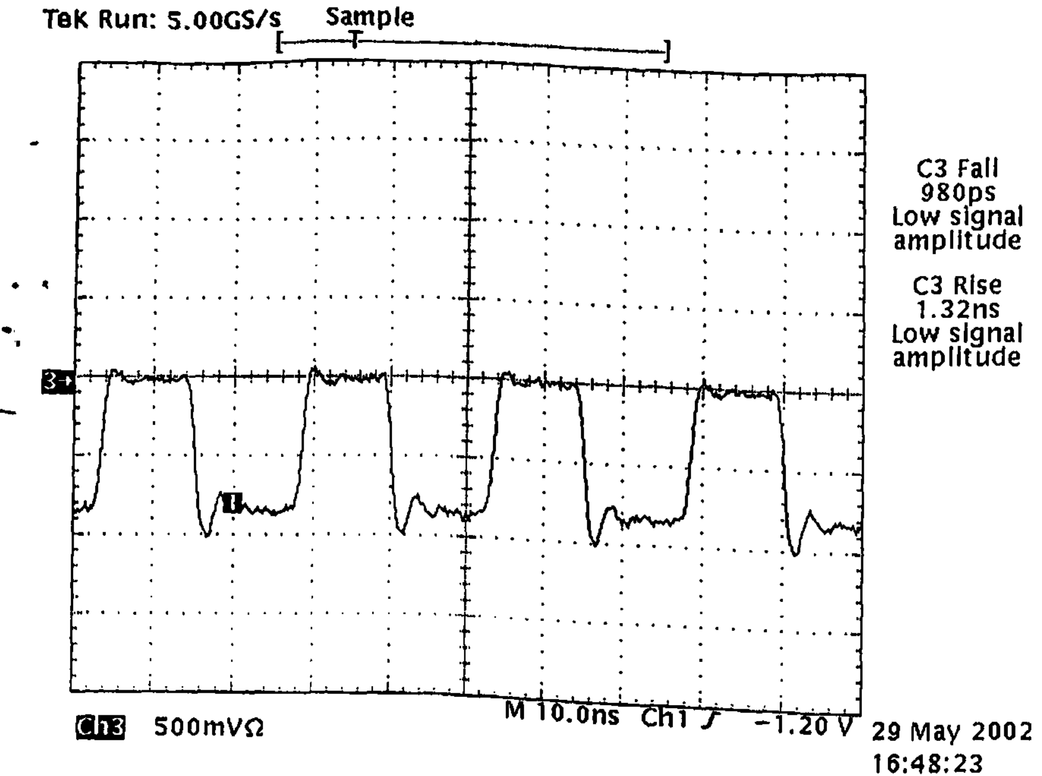


Εικόνα 5. TTL @40MHz

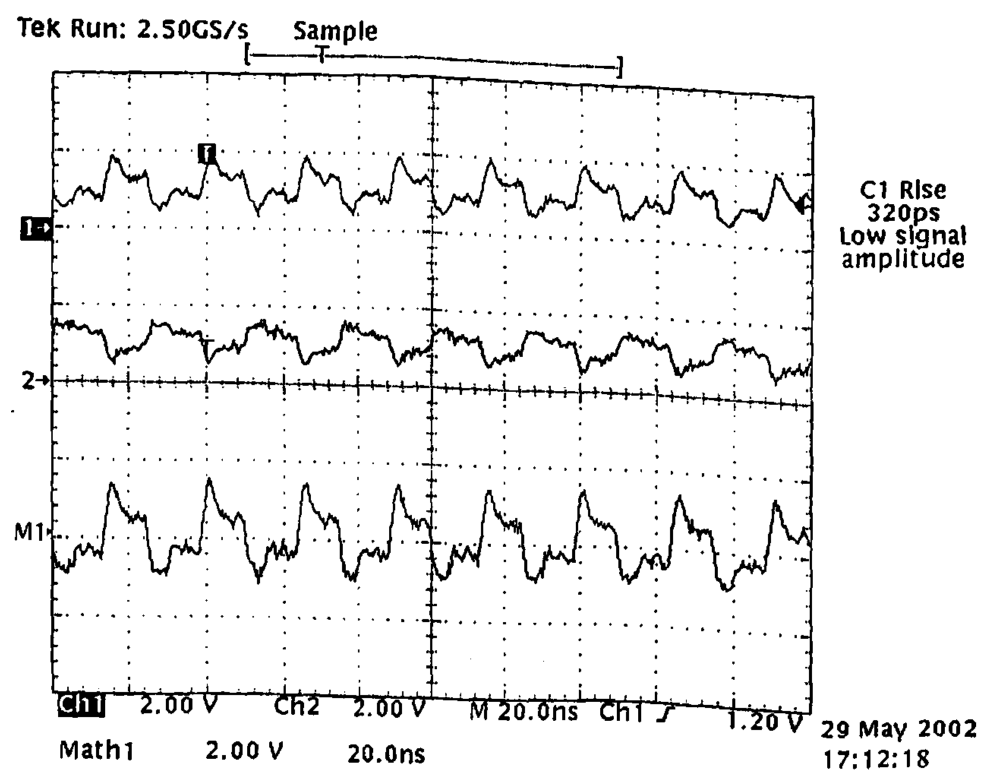


Εικόνα 5. ECL @40MHz

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ



Εικόνα 5. NIM @40MHz



Εικόνα 5. LVDS @40MHz

ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

2.6 Βιβλιογραφία

[1] Microelectronic Circuits 4th edition – A.Sedra & K.Smith
Oxford University Press 1997

[2] Techniques for Nuclear and Particle Physics Experiments 2nd Edition – W.R.Leo, Springer-Verlag 1994

[3] The Art of Electronics 2nd Edition – P.Horowitz & W.Hill
Cambridge University Press 1989

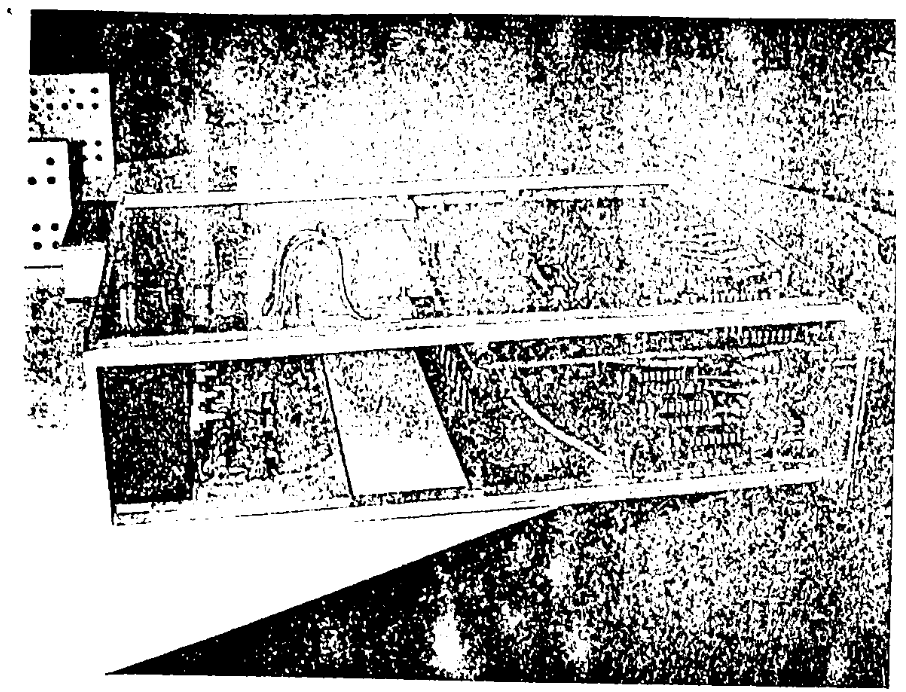
[4] LVDS Owner's Manual 2nd Edition
National Semiconductors 2000

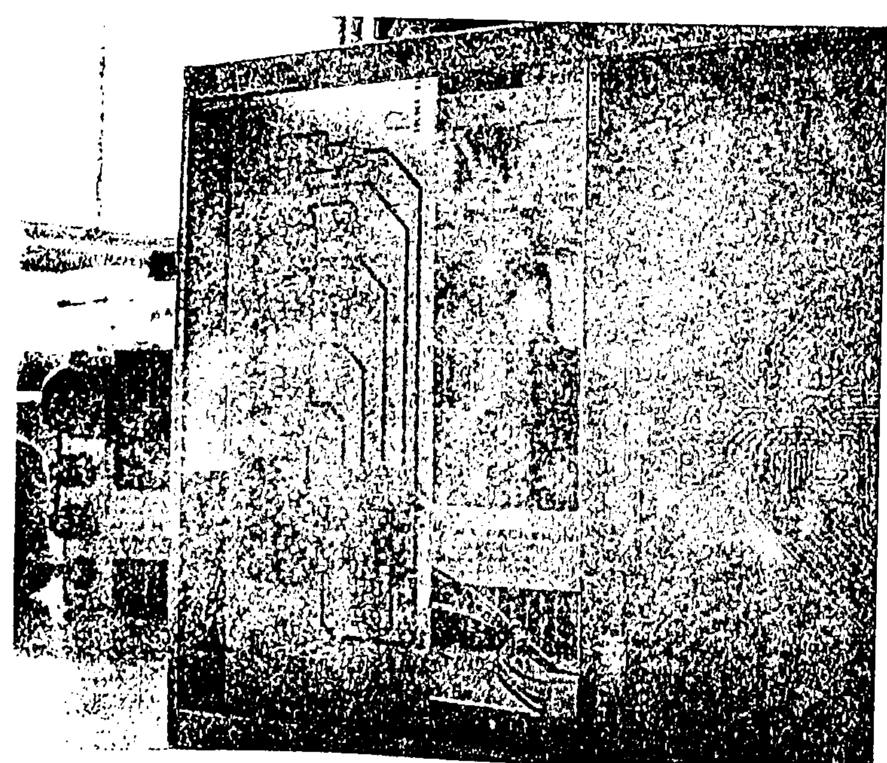
[5] Digital Electronics. Logic and Systems 3rd Edition - John Kershaw
Kent Pub Co 1987

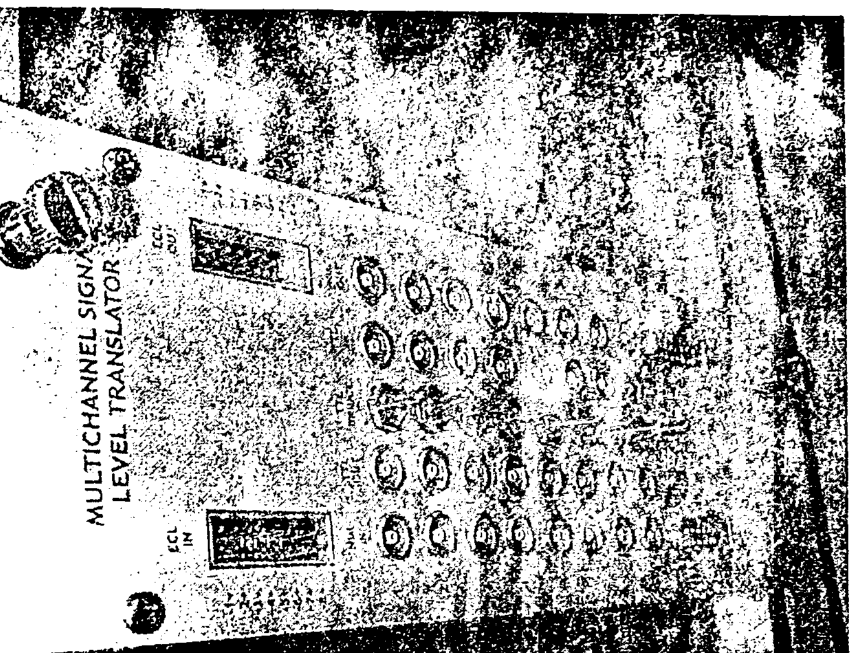
[6] MECL System Design Handbook 1st edition
On Semiconductors 1988

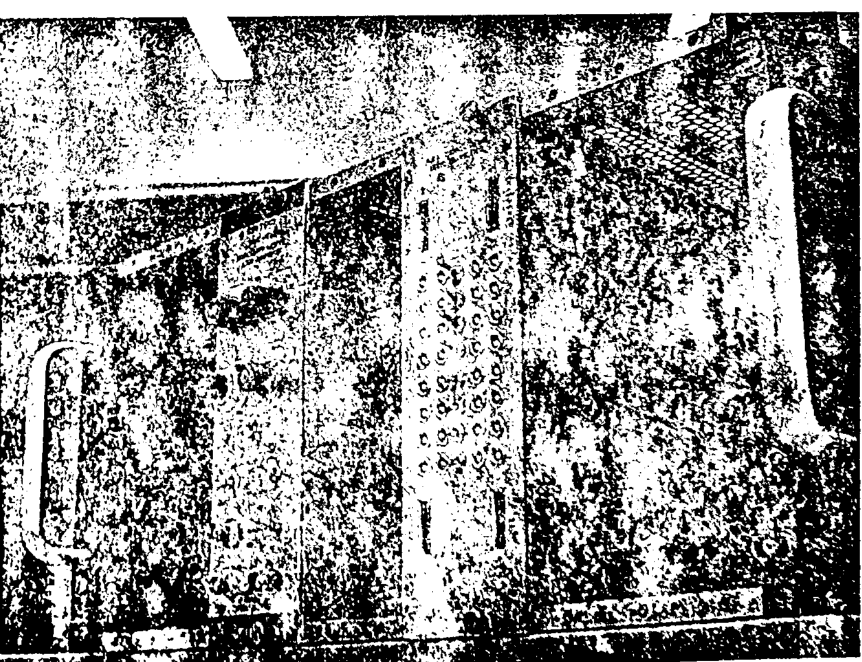
ΠΑΡΑΡΤΗΜΑ

Α. ΦΩΤΟΓΡΑΦΙΕΣ





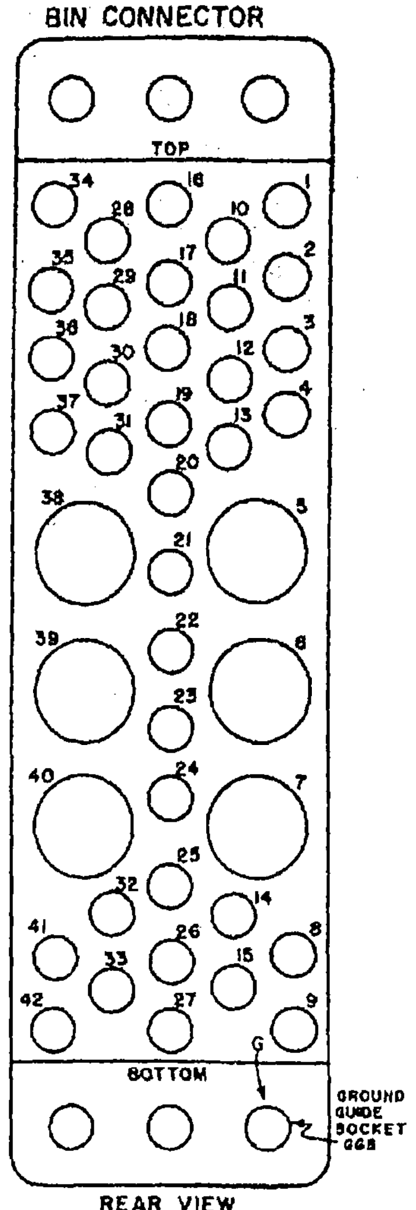

MULTICHANNEL SIGN
LEVEL TRANSLATOR
ECL
IN



ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

B. NIM CONNECTOR

pin	Function	pin	Function
1	+3V	23	Reserved
2	-3V	24	Reserved
3	Spare bus	25	Reserved
4	Reserved bus	26	Spare
5	Coaxial	27	Spare
6	Coaxial	28	+24V
7	Coaxial	29	-24V
8	200VDC	30	Spare bus
9	Spare bus	31	Spare
10	+6V	32	Spare
11	-6V	33	117 VAC (hot)
12	Reserved bus	34	Power Return GND
13	Spare	35	Reset (scaler)
14	Spare	36	Gate
15	Reserved	37	Reset (aux)
16	+12V	38	Coaxial
17	-12V	39	Coaxial
18	Spare bus	40	Coaxial
19	Reserved bus	41	117 VAC (neut)
20	Spare	42	HQ GND
21	Spare	G	GND Guide Pin
22	Reserved		



ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Γ. Εσωτερική λογική του XC9536

LOC=P14 IPAD ECL1 IBUF
LOC=P40 IPAD LVDS1 IBUF
LOC=P27 IPAD NIM1 IBUF
OR3 OBUF TTL1 OPAD LOC=P42

LOC=P13 IPAD ECL2 IBUF
LOC=P39 IPAD LVDS2 IBUF
LOC=P26 IPAD NIM2 IBUF
OR3 OBUF TTL2 OPAD LOC=P43

LOC=P12 IPAD ECL3 IBUF
LOC=P38 IPAD LVDS3 IBUF
LOC=P25 IPAD NIM3 IBUF
OR3 OBUF TTL3 OPAD LOC=P44

LOC=P11 IPAD ECL4 IBUF
LOC=P37 IPAD LVDS4 IBUF
LOC=P24 IPAD NIM4 IBUF
OR3 OBUF TTL4 OPAD LOC=P1

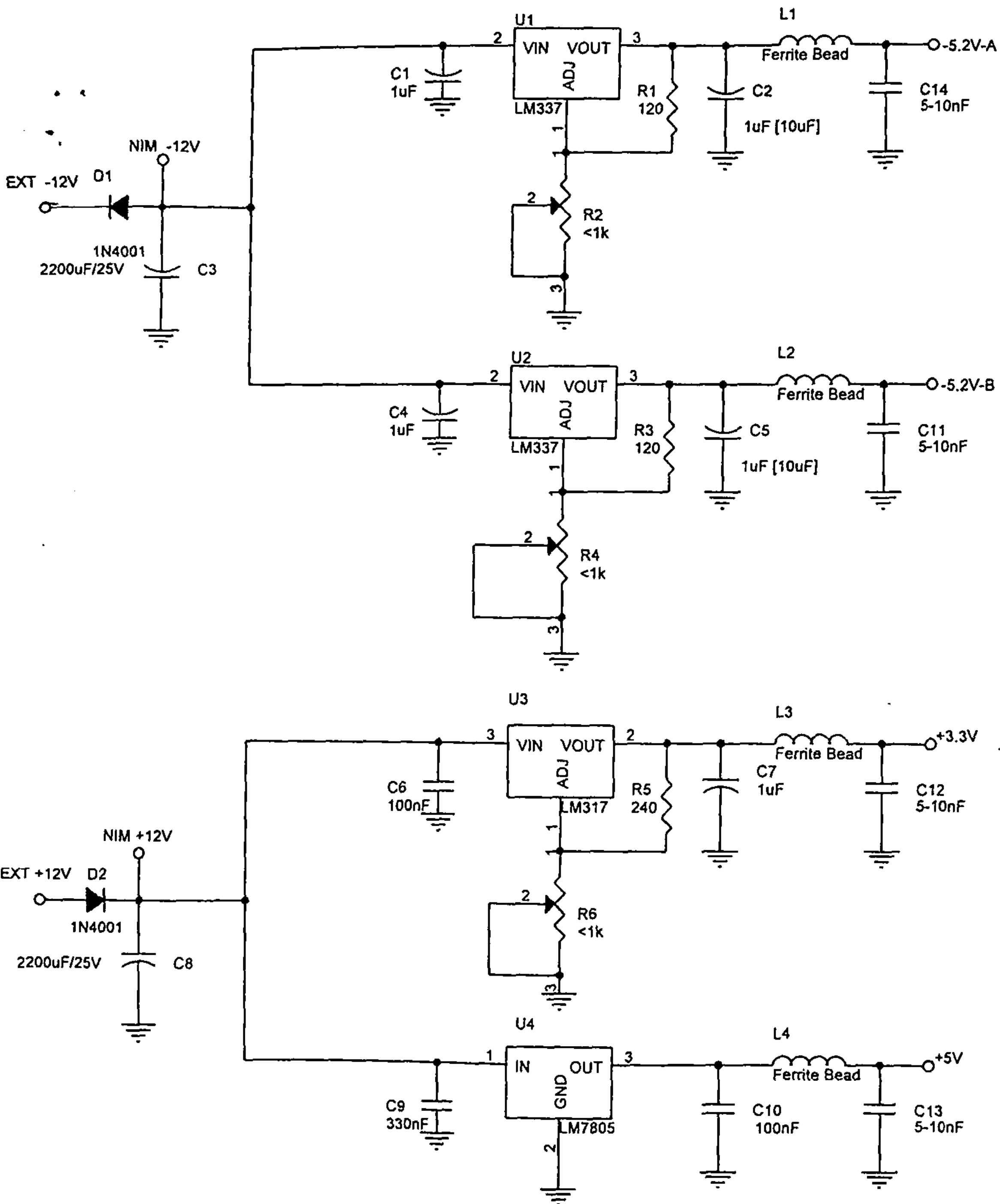
LOC=P9 IPAD ECL5 IBUF
LOC=P36 IPAD LVDS5 IBUF
LOC=P22 IPAD NIM5 IBUF
OR3 OBUF TTL5 OPAD LOC=P2

LOC=P8 IPAD ECL6 IBUF
LOC=P35 IPAD LVDS6 IBUF
LOC=P20 IPAD NIM6 IBUF
OR3 OBUF TTL6 OPAD LOC=P3

LOC=P7 IPAD ECL7 IBUF
LOC=P34 IPAD LVDS7 IBUF
LOC=P18 IPAD NIM7 IBUF
OR3 OBUF TTL7 OPAD LOC=P4

LOC=P6 IPAD ECL8 IBUF
LOC=P33 IPAD LVDS8 IBUF
LOC=P19 IPAD NIM8 IBUF
OR3 OBUF TTL8 OPAD LOC=P5

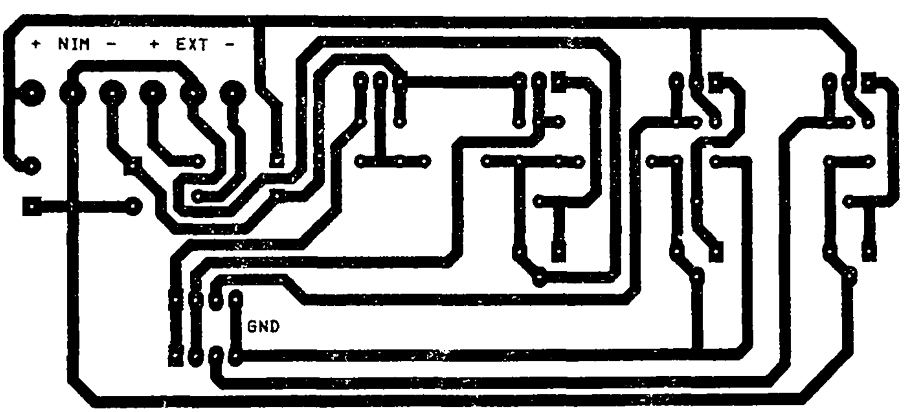
Δ. Τροφοδοτικό



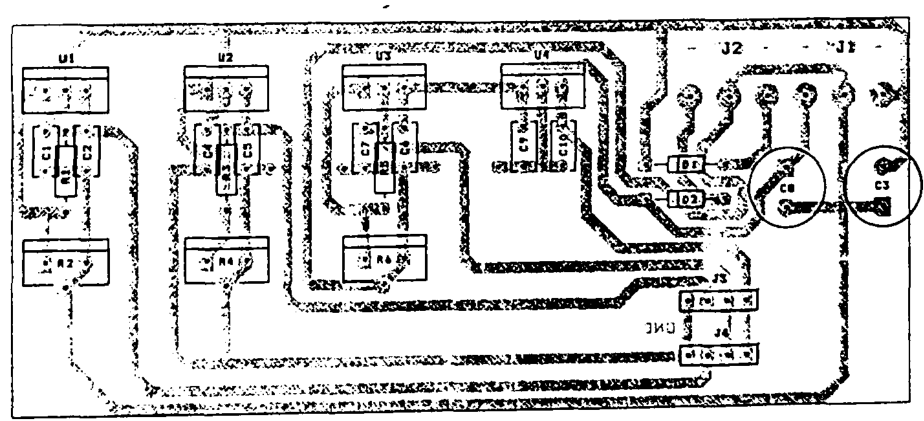
ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

Ε. ΤΥΠΩΜΕΝΑ ΚΥΚΛΩΜΑΤΑ

1. ΤΡΟΦΟΔΟΤΙΚΟ – BOTTOM LAYER

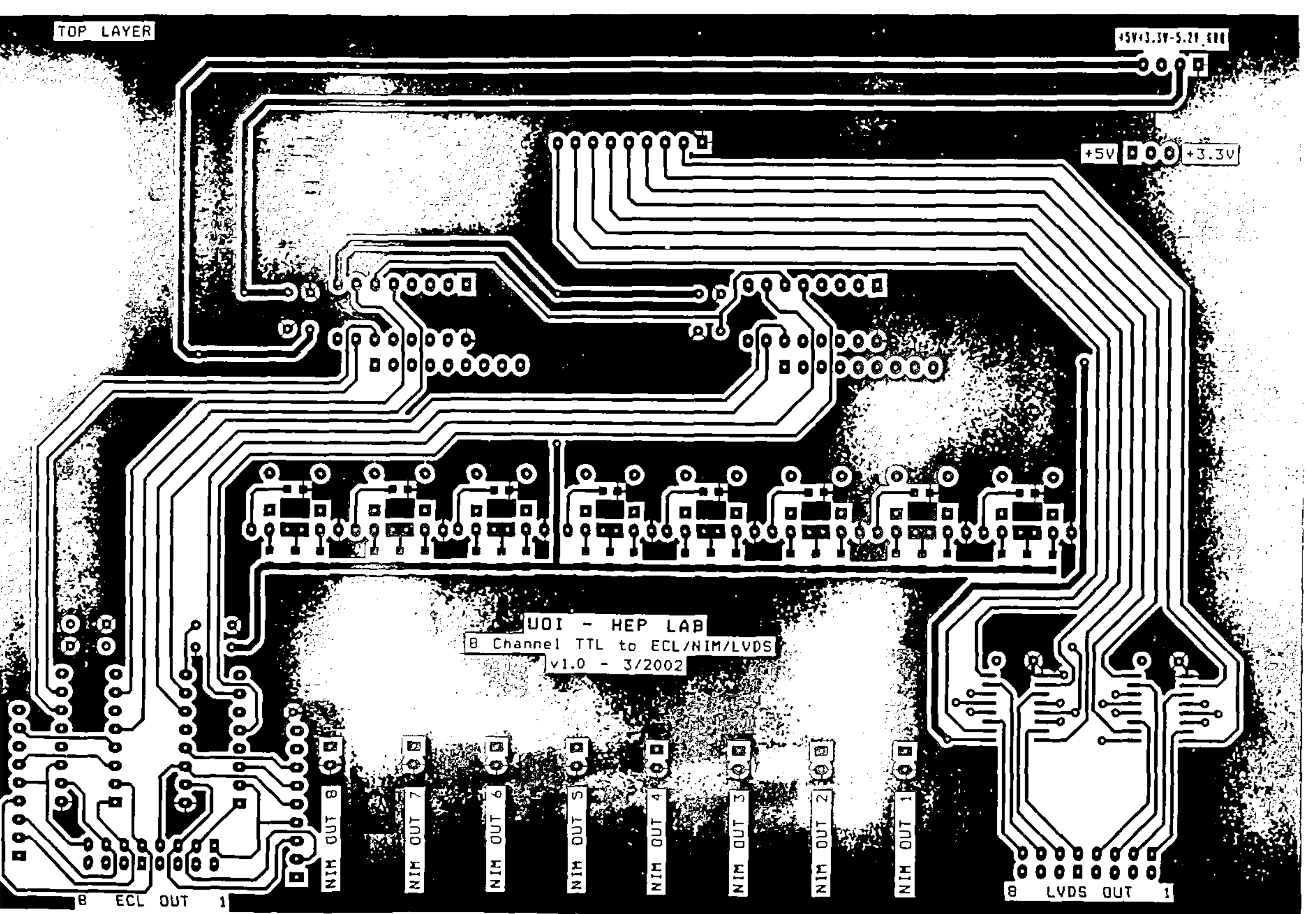


2. ΤΡΟΦΟΔΟΤΙΚΟ – COMPONENTS



ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

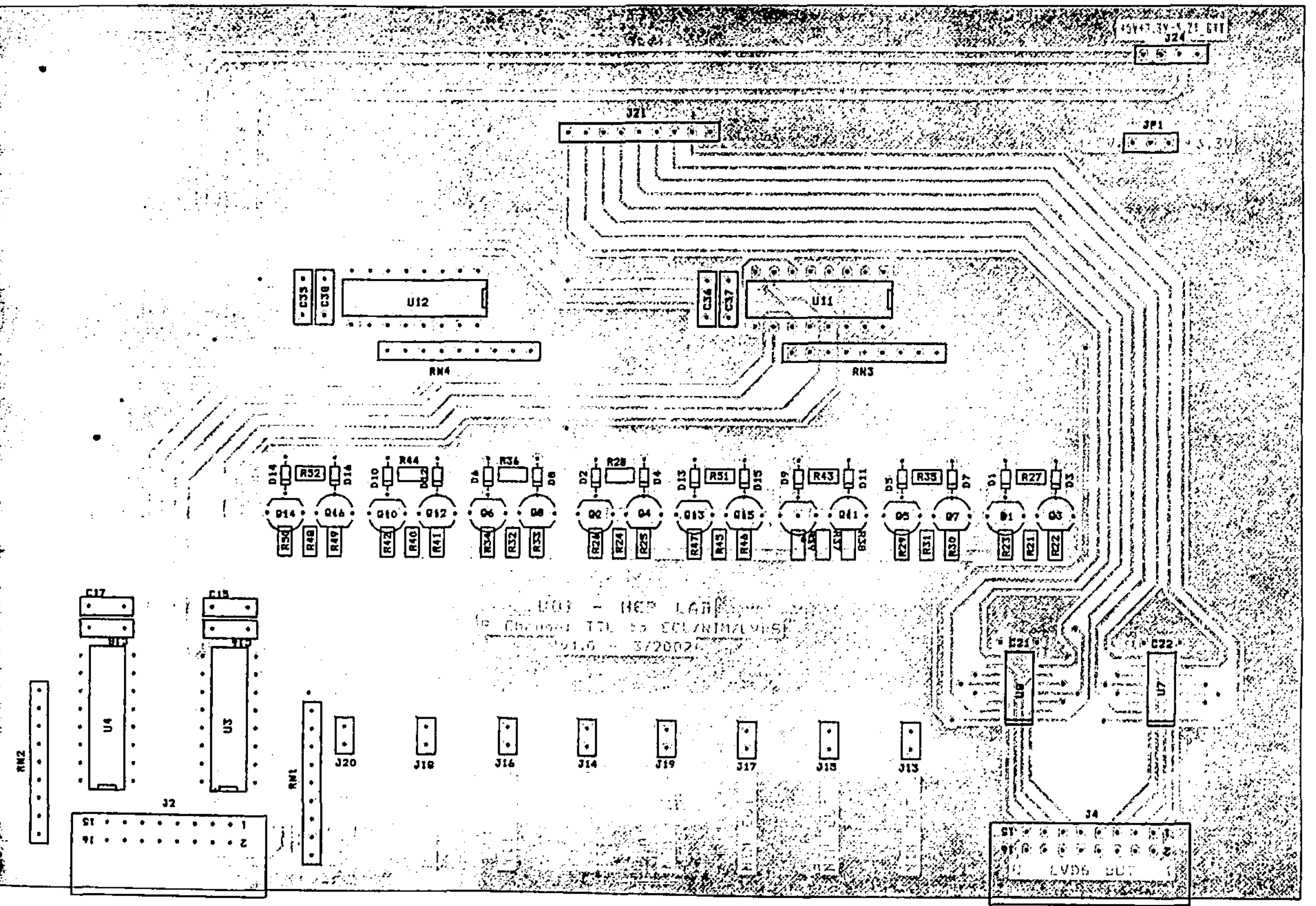
3. ΥΠΟΜΟΝΑΔΑ ΜΕΤΑΤΡΟΠΗΣ TTL ΣΕ ECL – NIM – LVDS – TOP LAYER



ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

4. ΥΠΟΜΟΝΑΔΑ ΜΕΤΑΤΡΟΠΗΣ TTL ΣΕ ECL – NIM – LVDS – BOTTOM LAYER

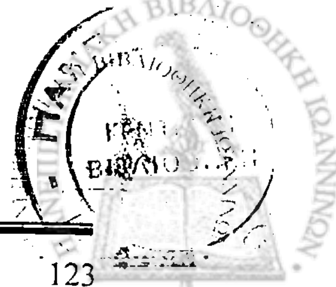
5. ΥΠΟΜΟΝΑΔΑ ΜΕΤΑΤΡΟΠΗΣ TTL ΣΕ ECL – NIM – LVDS – COMPONENTS



ΠΑΝΕΠΙΣΤΗΜΙΑΚΗ ΒΙΒΛΙΟΘΗΚΗ ΙΩΑΝΝΙΝΩΝ

6. ΥΠΟΜΟΝΑΔΑ ΜΕΤΑΤΡΟΠΗΣ ECL – NIM – LVDS ΣΕ TTL – TOP LAYER

7. ΜΟΝΑΔΑ ΜΕΤΑΤΡΟΠΗΣ ECL – NIM – LVDS ΣΕ TTL – BOTTOM LAYER



8. ΥΠΟΜΟΝΑΔΑ ΜΕΤΑΤΡΟΠΗΣ– NIM – LVDS ΣΕ TTL– COMPONENTS